Andrew S. Tanenbaum – David J. Wetherall

SZÁMÍTÓGÉP-HÁLÓZATOK

A Számítógép-hálózatok című könyv klasszikus alapműnek számít a hálózati technológiák témájában, és olyan alapismereteket közöl, amelyek nélkülözhetetlenek mind a gyakorló szakemberek, mind a tanulmányaikat folytatók számára. Tanenbaum professzor könyve először 1980-ban jelent meg, amikor a hálózatok még csak a tudományos érdekességek körébe tartoztak. Az azóta eltelt több mint 30 év alatt a technológia hatalmas fejlődésen ment keresztül, és a változások a könyv további kiadásaiban is nyomon követhetők. Ma a hálózatok világa a tartalom elosztásáról és a mobiltelefonoknak mint megannyi kis számítógépnek internetre kapcsolódásáról szól.

A könyvben számos változás történt az előző kiadás óta, hogy lépést tartson a számítógép-hálózatok folyamatosan változó világával. Az alábbi területeken történt bővülés vagy átdolgozás:

- vezeték nélküli hálózatok (802.11 és 802.16),
- az okostelefonok által használt 3G-hálózatok,
- RFID és szenzorhálózatok,
- tartalomelosztás tartalomszolgáltató hálózatok (CDN) alkalmazásával,
- egyenrangú társak (peer-to-peer, P2P) hálózata,
- média valós idejű továbbítása (tárolt forrásból, folyamként és élő forrásból),
- internetes telefonálás (IP-hálózaton keresztül történő beszédátvitel),
- késleltetéstűrő hálózatok.

Andrew S. Tanenbaum
David J. Wetherall

SZÁMÍTÓGÉP-HÁLÓZATOK

Harmadik, bővített, átdolgozott kiadás

www.panem.hu

Andrew S. Tanenbaum
David J. Wetherall

Számítógép-hálózatok

Harmadik, bővített, átdolgozott kiadás

Panem Könyvek®

A mű eredeti címe: Computer Networks. Fifth Edition.

Authorized translation from the English language edition, entitled COMPUTER NETWORKS, 5th Edition, 0132126958 by TANENBAUM, ANDREW S.; WETHERALL, DAVID J., published by Pearson Education, Inc, publishing as Prentice Hall,

A kiadásért felel a Taramix Kft. ügyvezetője, Budapest, 2013

ISBN 978-963-545-529-4

Lektorálta: Dr. Harangozó József
Fordította: Borbényi Judit, Hatwágner F. Miklós, Hollósi Gergely László, Horváth Dániel, Schulcz Róbert, Varga Gábor

panem@panem.hu
www.panem.hu

Tartalomjegyzék

Előszó

Az olvasó a könyv ötödik kiadását tartja a kezében*. Minden kiadás más-más fázisnak felelt meg a számítógép-hálózatok történetében. Az első kiadás megjelenésekor, 1980-ban a hálózatok még a tudományos érdekességek körébe tartoztak. 1988-ban, amikor a második kiadás megjelent, már használták őket az egyetemeken és a nagyobb cégeknél. 1996-ban, a harmadik kiadás megjelenésekor, a számítógép-hálózatok és főleg az internet már milliók számára napi valósággá vált. A negyedik kiadás idején, 2003-ban már általánosan elterjedtek a vezeték nélküli hálózatok és a mobil számítógépek a web és az internet elérésére. Az ötödik kiadás napvilágra kerülésekor a hálózatok világa a tartalom elosztásáról – például videóknak tartalomelosztó hálózatok (CDN) és egyenrangú társak (P2P) hálózata segítségével történő elosztásáról – és a mobiltelefonoknak mint megannyi kis számítógépnek internetre kapcsolódásáról szól.

Az ötödik kiadás újdonságai

A könyv számtalan változtatása között a legfontosabb Prof. David J. Wetherall társszerző megjelenése. Davidnek erős háttere van a hálózatok világában, több mint 20 éve foglalkozik nagyvárosi hálózatok tervezésével, illetve az internettel és a vezeték nélküli hálózatokkal. A University of Washington professzora, ahol az elmúlt évtizedben a számítógép-hálózatok és kapcsolódó témakörök területén oktatói és kutatói feladatokat látott el.

Természetesen a könyvben magában is számos változás történt, hogy lépést tartson a számítógép-hálózatok folyamatosan változó világával. Az alábbi területeken történt bővülés vagy átdolgozás:

- vezeték nélküli hálózatok (802.11 és 802.16),
- az okostelefonok által használt 3G-hálózatok,
- RFID és szenzorhálózatok,
- tartalomelosztás tartalomszolgáltató hálózatok (CDN) alkalmazásával,
- egyenrangú társak (peer-to-peer, P2P) hálózata,
- média valós idejű továbbítása (tárolt forrásból, folyamként és élő forrásból),
- internetes telefonálás (IP-hálózaton keresztül történő beszédátvitel),
- késleltetéstűrő hálózatok.

* Jelen magyar nyelvű kiadás az eredeti angol nyelvű könyv ötödik kiadásának adaptálása. *(A kiadó megjegyzése)*

Következzen egy részletes áttekintés fejezetről fejezetre.

Az 1. fejezet, hasonlóan a negyedik kiadáshoz, most is bevezetőként szolgál, tartalma azonban módosult és frissült. Az internet, a mobiltelefon-hálózatok, a 802.11, valamint az RFID és a szenzorhálózatok a számítógép-hálózatok egy-egy példájaként kerül bemutatásra. Az eredeti Ethernethez kapcsolódó anyagok – a vámpír csatlakozókkal együtt – kikerültek, ahogy az ATM-mel kapcsolatos részek is.

A 2. fejezet, amely a fizikai réteget fedi le, kibővült a digitális modulációval (többek között az OFDM-mel, melyet a vezeték nélküli hálózatokban használnak) és a (CDMA-n alapuló) 3G-hálózatokkal. Új technikák is előkerülnek, többek között az optikai szálnak az előfizetői szakaszban történő alkalmazása (Fiber to the Home, FttH) és az erősáramú hálózaton történő kommunikáció (power-line networking).

A 3. fejezet, amely a kétpontos (pont–pont) összeköttetésekkel foglalkozik, két módon tökéletesedett. A hibadetektáló és hibajavító kódokkal foglalkozó anyag frissítésén túl kiegészült egy, a modern kódokat leíró résszel is, amely kódok nagyon fontosak a gyakorlatban (például konvolúciós és LDPC-kódok). Példaként most a szinkron optikai hálózatok feletti csomagkapcsolást (Packet over SONET) és az ADSL-protokollt használjuk. Sajnos a protokoll-verifikációs részt kihagytuk, mivel keveset használják.

A 4. fejezetben, mely a MAC-alrétegről szól, az elvek változatlanok, de a technikák változtak. A példa hálózatokkal foglalkozó szekcióban ennek megfelelően számos változás történt, mely többek között a gigabit Ethernet, a 802.11, a 802.16, a Bluetooth- és a RFID-technikát tartalmazza. Ugyancsak frissült a LAN-kapcsolás, a VLAN-okat is beleértve.

Az 5. fejezet a hálózati réteggel foglalkozik. Itt a tárgyalás mélységében történtek változások, elsősorban a szolgáltatásminőség (mely a valós idejű média szempontjából fontos) és a hálózatok összekapcsolása területén. Bővültek a BGP-vel, OSPF-fel és CIDR-rel foglalkozó részek, valamint a többesküldéses útválasztás (multicast) bemutatása is. A fejezet a bárkinek-küldéses (anycast) útválasztást is tartalmazza.

A 6. fejezet, amely a szállítási réteget mutatja be, új anyagrészekkel bővült, amelyek a késleltetéstűrő hálózatokat és a torlódásvezérlést mutatják be általánosan; továbbá bizonyos részeit átdolgoztuk: a TCP torlódásvezérlését frissítettük és kiegészítettük. A ma már ritkán látható összeköttetés-alapú hálózatokat tartalmazó részt kihagytuk.

Az alkalmazásokról szóló 7. fejezet is frissült és bővült. Míg a DNS-sel és az elektronikus levelezéssel foglalkozó részek hasonlók a negyedik kiadásban leírtakhoz, addig az elmúlt évben számos fejlődés történt a web felhasználása, a médiafolyamok továbbítása és a tartalomszolgáltatás területén. Ennek megfelelően a webhez és a médiafolyam-továbbításhoz kapcsolódó részeket naprakésszé tettük. Teljesen új rész foglalkozik a tartalomszolgáltatással, beleértve a CDN-eket és a P2P-hálózatokat.

A biztonságról szóló 8. fejezet most is lefedi a bizalmasságot és hitelességet biztosító szimmetrikus és nyilvános kulcsú kriptográfiát. A gyakorlatban használt technikák bemutatását célzó részek frissültek, melyek a tűzfalakat és VPN-eket mutatják be, valamint kibővültek a 802.11 biztonságát és a Kerberos V5-öt bemutató részekkel.

A 9. fejezet egy frissített listát tartalmaz az ajánlott irodalomról, valamint egy mindenre kiterjedő irodalomjegyzéket a forrásokról. Ezeknek a cikkeknek és könyveknek több mint a felét 2000-ben vagy azután írták, a többi klasszikus alapműnek számít.

Rövidítések listája

A számítógépes könyvek tele vannak rövidítésekkel. Ez alól e könyv sem kivétel. Mire az olvasó a könyv végére ér, ismerősnek kell csengenie a következőknek: ADSL, AES, AJAX, AODV, AP, ARP, ARQ, AS, BGP, BOC, CDMA, CDN, CGI, CIDR, CRL, CSMA, CSS, DCT, DES, DHCP, DHT, DIFS, DMCA, DMT, DMZ, DNS, DOCSIS, DOM, DSLAM, DTN, FCFS, FDD, FDDI, FDM, FEC, FIFO, FSK, FTP, GPRS, GSM, HDTV, HFC, HMAC, HTTP, IAB, ICANN, ICMP, IDEA, IETF, IMAP, IMP, IP, IPTV, IRTF, ISO, ISP, ITU, JPEG, JSP, JVM, LAN, LATA, LEC, LEO, LLC, LSR, LTE, MAN, MFJ, MIME, MPEG, MPLS, MSC, MTSO, MTU, NAP, NAT, NRZ, NSAP, OFDM, OSI, OSPF, PAWS, PCM, PGP, PIM, PKI, POP, POTS, PPP, PSTN, QAM, QPSK, RED, RFC, RFID, RPC, RSA, RTSP, SHA, SIP, SMTP, SNR, SOAP, SONET, SPE, SSL, TCP, TDD, TDM, TSAP, UDP, UMTS, URL, VLAN, VSAT, WAN, WDM, és XML. De ne aggódjon! Az első előfordulásakor mindegyiket megjelöltük **vastag betűvel**, és pontosan meghatároztuk a jelentésüket. A móka kedvéért számolja meg, hogy a könyv elolvasása *előtt* mennyit ismer belőlük, írja fel a számot a margóra, majd számolja meg újra a könyv elolvasása *után*.

Hogyan használjuk a könyvet?

Annak érdekében, hogy a tanárok tankönyvként is használhassák a könyvet negyedéves vagy féléves bontásban, a fejezeteket alapvető és szabadon választható részekre tagoltuk. A tartalomjegyzékben csillaggal (*) jelöltek a szabadon választható alfejezetek. Ha egy főfejezet (például 2.7.) így van jelölve, akkor az összes alfejezete szabadon választható. Hasznos hálózati ismereteket tartalmaznak, de egy rövidebb kurzus esetén a folytonosság elvesztése nélkül kihagyhatók. Természetesen bátorítjuk a diákokat, hogy olvassák el ezeket az alfejezeteket is abban az esetben, ha van rá idejük, mert minden anyagrész naprakész és értékes információt tartalmaz.

Oktatási segédanyagok tanárok számára

A Pearson kiadó weboldalán, a *www.pearsonhighered.com/tanenbaum* címen az alábbi, jelszóval védett oktatási segédanyagok érhetők el. Felhasználónévért és jelszóért lépjen kapcsolatba a Pearson képviselőjével.

- a feladatok megoldását tartalmazó kézikönyv,
- PowerPoint-fóliák az előadások megtartásához.

Oktatási segédanyagok diákok számára

A diákok számára készült segédanyagok szabadon hozzáférhetők a könyvhöz társuló weboldalon, a *www.pearsohighered.com/tanenbaum* címen. Ezek:

- webes tartalmak, hiperhivatkozások különféle oktatási anyagokra, más szervezetek honlapjára, kérdés-felelet oldalakra stb.,
- ábrák, táblázatok és programok a könyvből,
- egy szteganográfiai bemutató,
- protokollszimulátorok.

Köszönetnyilvánítás

Sokan segítettek a munkánkban az ötödik kiadás létrehozása során. Hálával tartozunk az alábbi személyeknek az ötleteikért és javaslataikért: Emmanuel Agu (Worcester Polytechnic Institute), Yoris Au (University of Texas at Antonio), Nikhil Bhargava (Aircom International, Inc.), Michael Buettner (University of Washington), John Day (Boston University), Kevin Fall (Intel Labs), Ronald Fulle (Rochester Institute of Technology), Ben Greenstein (Intel Labs), Daniel Halperin (University of Washington), Bob Kinicki (Worcester Polytechnic Institute), Tadayoshi Kohno (University of Washington), Sarvish Kulkarni (Villanova University), Hank Levy (University of Washington), Ratul Mahajan (Microsoft Research), Craig Partridge (BBN), Michael Piatek (University of Washington), Joshua Smith (Intel Labs), Neil Spring (University of Maryland), David Teneyuca (University of Texas at Antonio), Tammy VanDegrift (University of Portland) és Bo Yuan (Rochester Institute of Technology). Melody Kadenko és Julie Svendsen adminisztratív segítséget nyújtott David számára.

Köszönet illeti Shivakant Mishrát (University of Colorado at Boulder) és Paul Nagint (Chimborazo Publishing, Inc.) a sok új fejezet végi feladat kigondolásáért. Tracy Dunkelberger, a szerkesztőnk a Pearsonnál, a szokásos segítőkész énjét mutatta. Melinda Haggerty és Jeff Holcomb jó munkát végzett, hogy minden gördülékenyen menjen. Steve Armstrong (LeTuorneau University) készítette a PowerPoint fóliákat. Stephen Turner (University of Michigan at Flint) művészi tökéletességgel ellenőrizte a weblap hiperhivatkozásait és a szimulátort, ami a könyvet kiegészíti. Olvasószerkesztőnk, Rachel Head szeme, mint a sasé, emlékezete egy elefánté. A hibajavításait elolvasva magunk is csodálkoztunk, hogy nem buktunk meg harmadikban.

Végül elérkeztünk a legfontosabb emberhez. Suzanne már 19 alkalommal élte át ugyanezt, és még mindig végtelen türelemmel és szerelemmel van irántam. Barbara és Marvin mindig arra inspirálnak, hogy jó tankönyveket készítsek. Daniel és Matilde szívesen fogadott jövevények a családunkban. Aron nem valószínű, hogy egyhamar olvasni fogja ezt a könyvet, de szereti a 8.54. ábrán látható szép képeket (AST). Katrin és Lucy mindig mosolyt csalt az arcomra. Köszönöm (DJW).

Andrew S. Tanenbaum
David J. Wetherall

1. Bevezetés

Az elmúlt három évszázad közül mindegyiket egy-egy technika uralta: a 18. századot az ipari forradalom során megjelenő nagy mechanikai rendszerek, a 19. századot a gőzgép, a 20. századot pedig az információgyűjtés, az információfeldolgozás és az információterjesztés. Egyebek között részesei lehettünk a telefonhálózatok világméretű elterjedésének, a rádió, a televízió feltalálásának, valamint a számítástechnikai iparág megszületésének és példátlan fejlődésének, továbbá a távközlési műholdak felbocsátásának és természetesen az internet elterjedésének.

A technika gyors fejlődése azt eredményezi, hogy ezek a területek gyorsan közelítenek egymás felé, és az információ gyűjtése, szállítása, tárolása és feldolgozása közti különbségek gyorsan eltűnnek. Egy olyan szervezet, amelynek több száz, nagy kiterjedésű földrajzi területen elszórtan elhelyezkedő irodája van, rendszerint elvárja, hogy képes legyen egyetlen gombnyomással ellenőrizni akár legtávolabbi kirendeltségének pillanatnyi állapotát is. Ahogy egyre jobban tudunk információt gyűjteni, feldolgozni és elosztani, az egyre kifinomultabb információfeldolgozás iránti igény is egyre gyorsabban nő.

Bár a számítástechnikai iparág a többi iparághoz (például autógyártás, légi közlekedés) képest viszonylag fiatal, a számítógépek rövid időn belül mégis látványos fejlődést értek el. Létezésük első két évtizedében a számítógépes rendszerek erősen egy helyre koncentrálódtak, ami általában egy nagy terem volt. Ezeknek a termeknek sokszor üvegablakai voltak, amelyeken keresztül a látogatók megbámulhatták a nagy elektronikus csodát. A közepes méretű vállalatok vagy egyetemek még csak egy-két számítógéppel rendelkeztek, de a legnagyobbaknak is legfeljebb csak néhány tucat volt belőlük. Akkoriban egyszerűen a tudományos-fantasztikum világába tartozott a gondolat, hogy 20 éven belül egy azonos teljesítményű, de egy bélyegnél is kisebb központi egységből több milliót gyártanak majd a tömegtermelésben.

A számítógépek és a távközlés egybeolvadása alapvetően befolyásolta a számítógépes rendszerek szervezését. Régen a felhasználók egy nagyméretű számítógépet tartalmazó terembe, a „számítóközpontba" vitték a futtatandó programjaikat. A „számítóközpont" mint fogalom ma már teljesen kihalt (bár adatközpontok, amelyek több ezer internetre kapcsolt szervert tartalmaznak, egyre gyakrabban előfordulnak). A régi modell az volt, hogy egy intézmény teljes számítástechnikai igényét egyetlen gép szolgálta ki. Ezt a modellt felváltotta az, hogy a feladatokat sok-sok különálló, de egymással összekapcsolt szá-

mítógép látja el. Az ilyen rendszereket **számítógép-hálózatoknak (computer networks)** nevezzük. Könyvünk ezeknek a hálózatoknak a tervezéséről és kialakításáról szól.

A „számítógép-hálózat" kifejezés ebben a könyvben mindenhol autonóm számítógépek olyan együttesét jelenti, amelyet egyetlen technika köt össze egymással. Két számítógépről akkor mondjuk, hogy összeköttetésben állnak, ha képesek információt cserélni egymással. A kapcsolatnak nem feltétlenül rézvezetéken keresztül kell megvalósulnia; fényvezető szálat, mikrohullámokat, infravörös fényt vagy kommunikációs műholdakat is használhatunk. A hálózatoknak sokféle méretük, alakjuk és formájuk lehet, amint azt később látni is fogjuk. Ezeket általában összekapcsolják egymással, hogy nagyobb hálózatokat alakítsanak ki belőlük, az **internet** talán a legismertebb példa a hálózatok hálózatára.

Az irodalomban a számítógép-hálózat és az **elosztott rendszer (distributed system)** fogalmai között jelentős mértékű keveredés tapasztalható. A legfőbb különbség az, hogy egy elosztott rendszerben a független számítógépek együttese egyetlen koherens rendszernek tűnik a felhasználói számára. Általában egyetlen modellje, vagy paradigmája van, amit a felhasználóinak mutat. Ennek a modellnek a megvalósításáért sokszor egy **köztes rétegnek (middleware)** vagy közbülső rétegnek nevezett szoftverréteg felel, ami közvetlenül az operációs rendszerre épül. Egy jól ismert példa elosztott rendszerre a **világháló (world wide web)**, amely az internetre épülve fut, és egy olyan modellt jelent, amelyben minden úgy néz ki, mintha egy dokumentum (weblap) lenne.

Egy számítógép-hálózatban ez a koherencia, az egységes modell és a közös szoftver hiányzik. A felhasználóknak az egyes gépeket kell használniuk; minden olyan próbálkozás hiányzik, ami egységessé próbálná tenni a gépek kinézetét és viselkedését. Ha a gépek különböző hardverrel és operációs rendszerrel rendelkeznek, az teljes mértékben látható a felhasználók számára is. Ha egy felhasználó egy programot egy távoli gépen szeretne lefuttatni, akkor be kell jelentkeznie arra a gépre, és ott kell lefuttatnia azt.

Az elosztott rendszer lényegében egy olyan szoftverrendszer, amely egy hálózatra épül rá. A szoftver biztosítja a nagyfokú egységességet és az átlátszóságot. Ebből kifolyólag a különbség egy számítógép-hálózat és egy elosztott rendszer között sokkal inkább a szoftverben (legfőképp az operációs rendszerben), mint a hardverben van.

Mindezek ellenére jelentős átfedés van a két téma között. Például mind az elosztott rendszereknek, mind a számítógép-hálózatoknak állományokat kell mozgatniuk. A különbség abban rejlik, hogy ki hozza létre ezt a mozgást: a rendszer vagy a felhasználó. Bár ez a könyv elsősorban a hálózatokra koncentrál, a témák közül sok az elosztott rendszerekben is jelentőséggel bír. További információért az elosztott rendszerekről lásd Tanenbaum és Van Steen [2007] munkáját.

1.1. A számítógép-hálózatok használata

Mielőtt belekezdenénk a műszaki kérdések részletes tárgyalásába, érdemes egy kis időt szentelni arra, hogy rámutassunk, miért érdeklik az embereket a számítógép-hálózatok, és hogy mire is lehet azokat használni. Tulajdonképpen, ha senki sem érdeklődne a számítógép-hálózatok iránt, akkor csak nagyon kevés épülne belőlük. Először a hagyományos üzleti felhasználásokat vesszük szemügyre, majd továbblépünk az otthoni hálózatokra és a mozgó felhasználók kapcsán a közelmúltban történt új fejlesztésekre, végül a társadalmi hatásokkal zárjuk a sort.

1.1.1. Üzleti alkalmazások

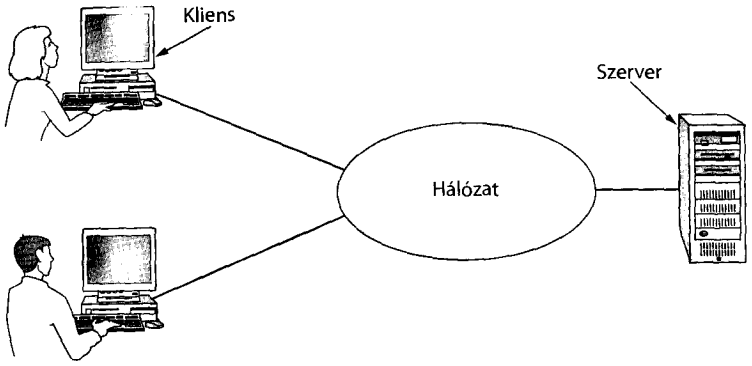
Sok vállalatnál használnak tetemes számú számítógépet. Például lehet, hogy egy cégnél minden munkatársra jut egy számítógép, amelyet termékek tervezésére, kiadványok szerkesztésére vagy könyvelésre használnak. Kezdetben talán ezek közül a számítógépek közül több is elszigetelten működött, de a vezetőség egy bizonyos ponton dönthetett úgy, hogy összeköti ezeket annak érdekében, hogy képesek legyenek szétosztani az információt a vállalaton belül.

Egy kicsit általánosabban megfogalmazva, itt **erőforrás-megosztásról (resource sharing)** van szó, a cél pedig az, hogy minden program, eszköz és legfőképpen adat mindenki számára elérhető legyen a hálózaton, tekintet nélkül az erőforrás és a felhasználó fizikai helyére. Egy nyilvánvaló és széles körben elterjedt példa, amikor irodai dolgozók egy csoportja megosztva használ egy nyomtatót. Egyiküknek sincs szüksége saját nyomtatóra, és egy nagy kapacitású hálózati nyomtató gyakran olcsóbb, gyorsabb és könnyebben karbantartható, mint nagyszámú egyedi nyomtató.

Mégis, talán még a fizikai eszközök (például nyomtatók, lapolvasók és CD-írók) megosztásánál is fontosabb az információ megosztása. Minden nagy és közepes méretű vállalat és sok kisebb cég léte nagymértékben a számítógépes információtól függ. A legtöbb cégnek vannak ügyféllistái, leltárai, pénzügyi nyilvántartásai és még sok minden más adata, ami a hálózatán elérhető. Egy bank, ha minden számítógépe felmondaná a szolgáltatást, öt percnél tovább nem tudná fenntartani magát. Egy modern termelőüzem, ahol számítógép vezérli a gyártósort, még 5 másodpercig sem tartana ki. Mára még egy kis utazási iroda vagy egy háromszemélyes ügyvédi iroda is nagymértékben függ a számítógép-hálózatoktól, hiszen az alkalmazottai a lényeges adatokhoz és dokumentumokhoz a hálózaton keresztül férhetnek hozzá.

A kisebb cégeknél általában minden számítógép egyetlen irodában, esetleg egyetlen épületben van. A nagyobb vállalatoknál azonban a számítógépek és az alkalmazottak lehetnek akár több országban, több tucatnyi irodába és üzembe szétszórva is. Mindezek ellenére egy New York-i üzletkötőnek néha szüksége lehet arra, hogy hozzáférhessen egy olyan termékkatalógus-adatbázishoz, ami Szingapúrban van. A **virtuális magánhálózatok (virtual private network, VPN)** révén pedig egyetlen kiterjesztett hálózattá kapcsolhatók össze a különböző telephelyeken levő hálózatok. Más szavakkal: annak a puszta ténye, hogy egy felhasználó történetesen épp 15 000 km-re van az adataitól, még nem szabad, hogy megakadályozza abban, hogy úgy dolgozzon vele, mintha azok helyben lennének tárolva. Ezt a célt úgy lehetne összefoglalni, hogy azt mondjuk, kísérletet teszünk arra, hogy véget vessünk a „földrajz zsarnokságának".

A legegyszerűbb módon úgy képzelhetjük el egy vállalat információs rendszerét, hogy az egy vagy több adatbázisból és néhány alkalmazottból áll, akiknek ezeket távolról el kell tudniuk érni. Ebben a modellben az adatokat nagy kapacitású számítógépeken tárolják, amiket **kiszolgálóknak** vagy **szervereknek (server)** hívunk. Sokszor ezeket egy központi



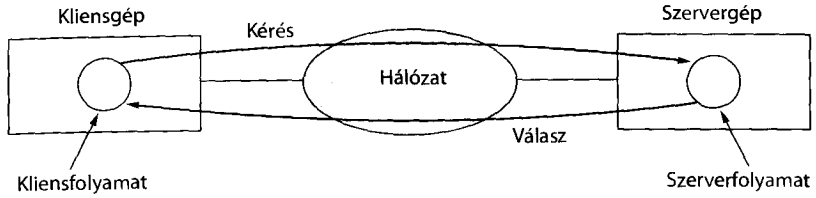
1.1. ábra. *Hálózat két klienssel és egy szerverrel*

épületben helyezik el, és rendszergazda tartja őket karban. Ezzel szemben az alkalmazottak asztalán egyszerűbb gépek vannak, amelyeknek **ügyfél** vagy **kliens (client)** a neve. Ezek segítségével távoli adatokhoz tudnak hozzáférni, amelyeket például felhasználhatnak egy éppen készülő táblázat készítésénél. (Néha úgy fogunk hivatkozni a kliensgép emberi felhasználójára, mint „a kliens", azonban mindig ki kell derülnie a szövegkörnyezetből, hogy a számítógépre gondolunk vagy pedig a felhasználójára.) A kliens- és szervergépeket egy hálózat köti össze, ahogyan azt az 1.1. ábra is mutatja. Figyeljük meg, hogy a hálózatot egy egyszerű ellipszissel jelöltük, minden egyéb részlet bemutatása nélkül. Ezt a formát akkor fogjuk használni, amikor a hálózatról elvont értelemben beszélünk. Amikor több részletre lesz szükség, ezeket rendelkezésre is bocsátjuk.

Ezt az egész elrendezést együtt **kliens–szerver-modellnek** hívjuk. Széles körben használatos és a hálózatok használatának nagy része ezen a modellen alapul. A modell legnépszerűbb megvalósítása a **webalkalmazásokhoz** köthető, melyek esetében a szerver az adatbázisa alapján weboldalakat generál válaszként a kliens kéréseire, melyek során az adatbázis is módosulhat. Például akkor, amikor valaki az otthonából hozzáfér egy weblaphoz a világhálón, akkor ezt a modellt használja úgy, hogy a távoli webszerver a szerver, és a felhasználó személyi számítógépe a kliens. A legtöbb esetben egy szerver nagyszámú klienst tud egyszerre kiszolgálni.

Ha részletesebben megnézzük a kliens–szerver-modellt, azt láthatjuk, hogy két folyamat (process) szerepel benne, egy a kliensgépen és egy a szervergépen. A kommunikáció olyan formában megy végbe, hogy először a kliensfolyamat egy üzenetet küld a hálózaton keresztül a szerverfolyamatnak. Ezután a kliensfolyamat várja a válaszüzenetet. Amikor a szerverfolyamat megkapja a kérést, elvégzi a kért munkát, vagy megkeresi a kért adatot, és egy választ küld erről vissza. Ezeket az üzeneteket mutatja az 1.2. ábra.

A számítógép-hálózatok kiépítésének ma már több köze van az emberekhez, mint az információhoz vagy akár a számítógépekhez. Egy számítógép-hálózat ugyanis nagyon hatékony **kommunikációs eszközt** ad az alkalmazottak kezébe. Lényegében minden vállalatnak, amelynek kettő vagy több számítógépe van, van **e-levelezése (elektronikus levelezés, electronic mail, e-mail)**, amit az alkalmazottak általában a napi kommunikációjuk nagy részéhez használnak. Valójában gyakori beszédtéma a kávéfőző gép körül összegyűlt alkalmazottak körében, hogy milyen sok e-levéllel kell mindenkinek



1.2. ábra. *A kliens–szerver-modellben szereplő kérések és válaszok*

megbirkóznia, amelynek ráadásul nagy része felesleges is, amióta a főnökök felfedezték, hogy ugyanazt a (sokszor semmitmondó) üzenetet minden alkalmazottnak elküldhetik egyetlen gombnyomással.

Az alkalmazottak közötti telefonhívások a telefontársaság helyett a számítógépes hálózaton keresztül is továbbíthatók. Ennek a megoldásnak a neve **IP-telefonálás (IP telephony)** vagy **IP-hálózaton keresztül történő hangátvitel (Voice over IP, VoIP)**, amennyiben internetes szolgáltatást vesz igénybe. A vonal végén levő mikrofon és hangszóró egyaránt tartozhat egy VoIP-képes telefonhoz vagy az alkalmazott számítógépéhez. A vállalatok számára ez csodálatos módja a telefonszámlán való spórolásnak.

A kommunikáció más, gazdagabb formáit is lehetővé teszik a számítógép-hálózatok. A hang mellé mozgókép is csatolható, hogy az egymástól távol tartózkodó alkalmazottak lássák és hallják egymást a megbeszélések közben. Ez a módszer hatékony lehetőség a korábban az utazásra áldozott költség és idő megtakarítására. Az **asztal megosztása (desktop sharing)** révén távoli dolgozók is láthatják a grafikus képernyőket, valamint interakcióba is léphetnek velük. Ezáltal két, egymástól messze dolgozó személy számára is egyszerűvé válik, hogy együtt írjanak és olvassanak egy táblát vagy közösen készítsenek el egy jelentést. Amikor egy dolgozó módosít valamit egy online dokumentumban, a többiek azonnal látják a változást ahelyett, hogy napokat kelljen várniuk egy levélre. Ez a gyorsulás olyan kiterjedt csoportok tagjai között is egyszerűvé teszi az együttműködést, amelyek számára ez korábban elképzelhetetlen volt. Még csak most kezdjük használni a távoli együttműködés olyan ambiciózus formáit, mint a távgyógyítás (például távoli betegmegfigyelés), de ezek sokkal fontosabbá is válhatnak a jövőben. Gyakran mondják, hogy a kommunikáció és a közlekedés versenyben állnak egymással, és bármelyik nyerjen, szükségtelenné teszi a másikat.

Sok vállalat számára a harmadik cél az üzletmenet elektronikus intézése, különösen a vásárlókkal és a beszállítókkal. Ezt az új modellt **e-kereskedelemnek** hívják **(elektronikus kereskedelem, electronic commerce, e-commerce)**, és gyors növekedést produkált az elmúlt néhány évben. Légitársaságok, könyvesboltok és egyéb kereskedők már felfedezték, hogy sok vásárló szereti azt a kényelmet, hogy otthonról vásárolhat. Ennek eredményeképpen sok vállalat biztosít interneten keresztül elérhető katalógust termékeiről és szolgáltatásairól, valamint fogad rendeléseket elektronikusan. Többek között gépjárművek, repülőgépek és számítógépek gyártói különböző beszállítóktól vásárolnak részegységeket, és ezekből szerelik össze a termékeiket. Számítógép-hálózatok használatával a gyártók igény szerint, elektronikusan adhatják fel a megrendeléseiket. Ez csökkenti a nagy raktárkészletek felhalmozása iránti szükségletet, valamint javítja a hatékonyságot.

1.1.2. Otthoni alkalmazások

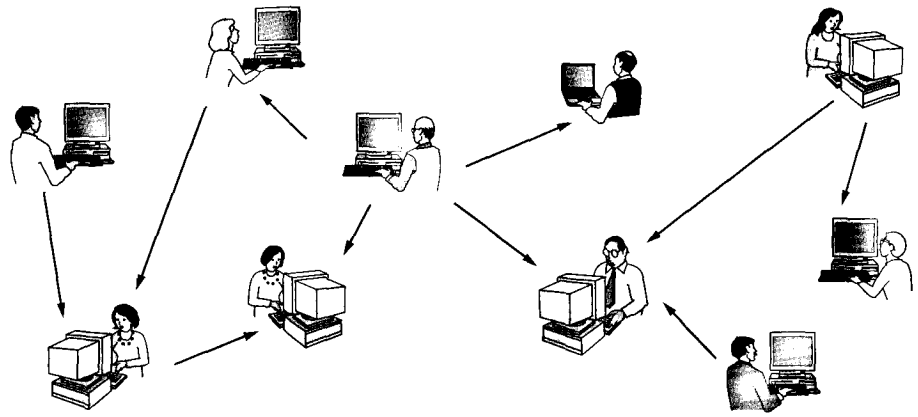
1977-ben Ken Olsen volt a Digital Equipment Corporation elnöke, ami akkor a második helyezett számítógép-eladó volt a világon (az IBM után). Amikor megkérdezték tőle, hogy a Digital befektetései miért nem követik nagyobb mértékben a személyi számítógépek piacának növekedését, azt válaszolta, hogy „Egy magánembernek semmi oka nincs arra, hogy személyi számítógép legyen az otthonában.". A történelem megmutatta, hogy ez másképp van, és a Digital azóta már nem létezik. Az emberek kezdetben szövegszerkesztés és játék céljából vásároltak számítógépet. A közelmúltban a legfőbb indokká valószínűleg az internet-hozzáférés biztosítása lépett elő. Mostanra pedig sok fogyasztói elektronikus eszköz, például set-top boxok, játékkonzolok vagy rádiós órák beágyazott számítógépekkel és számítógép-hálózati képességekkel készülnek, különös tekintettel a vezeték nélküli hálózatokra, és az otthoni hálózatokat széles körben használják szórakozásra, ide értve a zene, fényképek és videók hallgatását, megtekintését és létrehozását.

Az internet-hozzáférés **összeköttetést (connectivity)** biztosít az otthoni felhasználók számára távoli számítógépek felé. Mint a vállalatok, az otthoni felhasználók is elérhetnek információkat, kommunikálhatnak más emberekkel, valamint termékeket és szolgáltatásokat vásárolhatnak az e-kereskedelemben. A legnagyobb előny mostanság az otthonon kívüli kapcsolódás lehetősége. Bob Metcalfe, az Ethernet feltalálója azt feltételezte, hogy egy hálózat értéke a felhasználók számával négyzetesen arányos, mert hozzávetőlegesen ennyi a létrehozható kapcsolatok száma [Gilder, 1993]. Ez a feltételezés „Metcalfe törvényeként" ismert. Segít megmagyarázni azt, hogy hogyan fakad az internet óriási népszerűsége a méretéből.

A távoli adatok elérésének sok lehetséges módja van. Lehet szörfölni a világhálón fontos információért, vagy csak egyszerűen szórakozásból. Elérhető információ a művészetekről, az üzletről, a főzésről, a kormányról, az egészségről, a történelemről, hobbikról, a szabadidős tevékenységekről, a tudományról, a sportról, az utazásról és még sok más területről. A szórakozás túl sok formában van jelen ahhoz, hogy valamennyit felsoroljuk, és néhány olyanban is, amiket jobb fel sem sorolni.

Sok újság érhető már el a világhálón, ezek személyre is szabhatók. Például lehetséges azt mondani egy újságnak, hogy mindent látni akarunk a korrupt politikusokról, a nagy tüzekről, a hírességek botrányairól és a járványokról, de csak semmi foci, köszönjük szépen. Néha még az is lehetséges, hogy a kiválasztott cikkeket automatikusan letöltse gépünk a merevlemezére, amíg alszunk, vagy kinyomtassa a nyomtatónkon egy kicsivel a reggeli előtt. Ahogyan ez az irányzat előrehalad, nagymértékű munkanélküliséget fog okozni a 12 éves újságkihordó fiúk körében, de az újságok szeretik ezt, mivel a kiszállítás mindig is az előállítási lánc leggyengébb láncszeme volt. Természetesen ahhoz, hogy ez az üzleti modell működőképes legyen, ki kell találniuk, hogy hogyan kereshetnek pénzt ebben az új világban, ami nem is teljesen nyilvánvaló, mivel az internetfelhasználók elvárják, hogy minden ingyenes legyen.

A következő lépés az újságok (és a magazinok, valamint tudományos folyóiratok) után az internetes digitális könyvtár. Sok szakmai szervezet, mint például az ACM (*www.acm.org*) és az IEEE Computer Society (IEEE számítógépes társaság; *www.computer.org*) már ma is számos interneten elérhető folyóirat és konferenciaanyag tulajdonosa. Más csoportok gyorsan követik őket. Az elektronikus könyvolvasók és online könyvtárak a nyomtatott könyvet előbb-utóbb elavulttá tehetik. A szkeptikusoknak azt ajánlanám, hogy figyeljék meg, mekkora hatással volt a könyvnyomtatás a középkori díszes kéziratokra.



1.3. ábra. *Egy P2P-rendszerben nincsenek rögzített kliensek és szerverek*

Az információ nagy részét a kliens–szerver-modell szerint érjük el, de az információ elérésének létezik egy másik népszerű modellje is, melyet **egyenrangú társak közötti (peer-to-peer, P2P) kommunikációnak** nevezünk [Parameswaran és mások, 2001]. Ebben a formában laza csoportot alkotó személyek tudnak kommunikálni a csoportjukat alkotó többi személlyel, amint ezt az 1.3. ábra is mutatja. Elméletben minden személy képes kommunikálni egy vagy több emberrel; nincs rögzített felosztás a kliensek és a szerverek között.

Sok P2P-rendszernek, mint a BitTorrent [Cohen, 2003], egyáltalán nincs központi adatbázisa a tárolt tartalomról. Ehelyett minden felhasználó saját, helyi adatbázist tart karban, valamint biztosít egy listát a rendszer más, hozzá közeli tagjairól. Egy új felhasználó így elmehet egy már létező felhasználóhoz, hogy lássa, neki mije van, és kaphat tőle egy listát más tagokról, akiknél körülnézhet még több tartalom és név után. Ez a kereső eljárás a végtelenségig ismételhető, hogy így egy nagy helyi adatbázist hozzunk létre arról, hogy mik is találhatók a rendszerben. Ez olyan tevékenység, ami unalmassá válhatna az emberek számára, de egyben olyan is, amelyben a számítógépek remekelnek.

A P2P-kommunikációt gyakran használják zene és filmek megosztására. Igazán sikeressé 2000 körül vált a Napster nevű zenemegosztó szolgáltatás révén, melyet az írott történelem talán legnagyobb szerzői jogi pere után állítottak le [Lam és Tan, 2001; Macedonia, 2000]. A P2P-kommunikációnak jogszerű alkalmazásai is vannak. Például, amikor zenebarátok nyilvánosan terjeszthető zeneszámokat cserélnek, amikor családok egymásnak küldözgetnek fényképeket vagy mozgóképeket, vagy a felhasználók nyilvánosan elérhető programcsomagokat töltenek le. Tulajdonképpen az internet egyik legnépszerűbb alkalmazása, az e-levelezés is jellemzően P2P. A kommunikáció e formája várhatóan jelentős növekedésnek fog indulni a jövőben.

A fenti alkalmazások közül mindegyik egy személy és egy távoli, információval teli adatbázis közötti interakcióra épül. A hálózatok felhasználásának második széles kategóriája az emberek közti kommunikáció, amely alapvetően a 21. század válasza a 19. század

telefonjára. Az e-levelezést már sok millió ember használja szerte a világon, és a felhasználása továbbra is gyorsan növekszik. Megszokottá vált, hogy hangot és mozgóképeket is tartalmaz a szöveg és a képek mellett. Az illatok továbbítása még egy kis időt igényel.

Minden valamirevaló tizenéves az **azonnali üzenetküldés** (**instant messaging**) rabja. Ez a kommunikációs eszköz, mely a hozzávetőlegesen 1970 óta használt *talk* nevű UNIX-programból származik, azt teszi lehetővé két ember számára, hogy egymásnak szóló üzeneteket gépeljenek be valós időben. Léteznek többfelhasználós üzenetküldő szolgáltatások is, mint a **Twitter** szolgáltatás, melynek használatával az emberek rövid szöveges üzeneteket, úgynevezett „csiripeléseket" (tweet) küldhetnek baráti körüknek vagy az érdeklődő hallgatóság számára.

Az alkalmazások használhatják az internetet hang (például internetes rádióállomások) és videó (például YouTube) továbbítására. Amellett, hogy a távoli barátokat olcsón felhívhatjuk, ezek az alkalmazások olyan gazdag felhasználói élményeket is nyújthatnak, mint a távtanulás, ami azt jelenti, hogy anélkül a kellemetlenség nélkül lehet részt venni a reggel 8 órai előadáson, hogy az ágyból fel kellene kelni. Hosszú távon a hálózatoknak az a célkitűzése, hogy javítsák az emberek közötti kommunikációt, fontosabbnak bizonyulhat bármely más célkitűzésnél. A számítógép-hálózatok különösen fontossá válhatnak a földrajzi szempontból hátrányos helyzetű emberek számára, mert megadják nekik ugyanazt a lehetőséget a szolgáltatások elérésére, mint amit a nagyvárosok közepén élő emberek is élveznek.

A személyek közti kommunikáció és az információelérés között helyezhetők el a **közösségi hálózatok** (**social network**). Itt az információ áramlása olyan személyes kapcsolatok révén történik, melyeket az emberek egymás között igazolnak vissza. A legnépszerűbb közösségi hálózatok közül az egyik a **Facebook**. Lehetőséget ad felhasználói profiloldalak karbantartására, valamint személyes hírek és információk megosztására olyanokkal, akik a felhasználóval ismerősöknek regisztrálták magukat. Más közösségi hálós alkalmazásokban az emberek ismerősök ismerősei által mutatkozhatnak be egymásnak, hírüzeneteket küldhetnek a barátaiknak, mint a fenti Twitter esetében, és sok más funkció is elérhető lehet.

Még ennél lazább kötelékben is van lehetőség közös munkára, tartalom létrehozására. Például a **wiki** egy olyan típusú weboldal, amelyet egy közösség tagjai együttműködve szerkesztenek. A leghíresebb wiki a **Wikipédia**, egy bárki által szerkeszthető enciklopédia, de ezenkívül több ezer wiki létezik.

A harmadik kategória az elektronikus kereskedelem, a kifejezés lehetséges legtágabb értelmében. Az otthoni vásárlás manapság is népszerű, és lehetővé teszi, hogy több ezer cég katalógusait vizsgálhassuk át az interneten. Ezen katalógusok közül néhány interaktív, különböző nézőpontokból vagy testre szabható konfigurációkban mutatja meg a terméket. Miután a felhasználó elektronikus úton megvett egy terméket, de nem tudja az árut működésbe hozni, az internetes ügyfélszolgálatot keresheti fel.

Egy másik terület, ahol az e-kereskedelmet már napjainkban is széles körben használják, a pénzügyi intézmények elérése. Sokan már ma is elektronikusan fizetik a számláikat, felügyelik a bankszámláikat és kezelik a befektetéseiket. Ez az arány biztosan növekedni fog, ahogyan a hálózatok egyre biztonságosabbá válnak.

Az elektronikus bolhapiac olyan terület, amelyet szinte senki sem látott előre. A használt dolgok internetes árverezése jelentős iparággá vált. Szemben a hagyományos e-kereskedelemmel, amely a kliens–szerver-modellt követi, az internetes aukciók inkább

Rövidítés	Teljes angol név	Teljes magyar név	Példa
B2C	Business-to-consumer	Cég a vásárlónak	Könyvrendelés az interneten
B2B	Business-to-business	Cég a cégnek	Egy autógyártó abroncsokat rendel a beszállítótól
G2C	Government-to-consumer	Kormány a polgárnak	A kormány elektronikusan küldi szét az adóbevallási űrlapokat
C2C	Consumer-to-consumer	Vásárló a vásárlónak	Használt dolgok internetes árverezése
P2P	Peer-to-peer	Egyik társ a másiknak (egyenrangú társak)	Zene megosztása

1.4. ábra. *Az e-kereskedelem néhány formája*

P2P-rendszerek abban az értelemben, hogy a felhasználók egyszerre lehetnek eladók is és vásárlók is.

Ezek közül az e-kereskedelmi formák közül néhánynak olyan rövid neve alakult ki, amely az angol „2" és „to" szavak kiejtésének hasonlóságán alapul. A legnépszerűbbeket az 1.4. ábrán soroltuk fel.

Negyedik kategóriánk a szórakoztatás. Nagy lépéseket tett meg az otthonokban az elmúlt években, ahogy a zene, rádió- és televízióműsorok, valamint filmek interneten keresztül történő terjesztése vetekedni kezdett a hagyományos módszerekkel. A felhasználók kereshetnek, megvásárolhatnak és letölthetnek MP3 zeneszámokat és DVD-minőségű filmeket, és felvehetik őket a személyes gyűjteményükbe. Ma a tv-műsorok sok otthonba **IPTV**- (**IP TeleVision – IP-televízió**) rendszereken keresztül jutnak el, melyek a kábeltévés vagy rádiós műsorszórás helyett az IP-technikán alapulnak. A médiafolyam-alkalmazások révén a felhasználók internetes rádióállomások műsorába kapcsolódhatnak be, vagy a kedvenc tv-műsoraik legutóbbi részeit nézhetik vissza. Természetesen az összes ilyen tartalom szabadon továbbítható a házban a különböző eszközök, megjelenítők és hangszórók között, általában vezeték nélküli hálózaton.

Hamarosan lehetségessé válik, hogy kiválasszunk egy tetszőleges filmet vagy tv-programot az összes közül, amit valaha készítettek a világ összes országában, és azt azonnal meg is jelenítsük a képernyőnkön. Az új filmek interaktívakká válhatnak, vagyis a felhasználót néha megkérdezik arról, hogy merre menjen tovább a történet (megölje Macbeth Duncant vagy csak várakozzon tétlenül?), minden lehetőségre más-más képsorokkal felkészülve. Az élő tv-adások is interaktívakká válhatnak, az egész közönség részt vehet a kvízjátékokban, választhatnak a versenyzők közül és így tovább.

A szórakozás további formáját a játékok jelentik. Manapság is vannak már többszemélyes, valós idejű szimulációs játékaink, mint például bújócska egy képzeletbeli várbörtönben és repülésszimulátorok, amelyekben a játékosok egyik csapata próbálja lelőni az ellenséges csapat játékosait. A virtuális világok állandó színteréül szolgálnak a sok ezer felhasználó által megtapasztalt közös, osztott, háromdimenziós grafikával megjelenített valóságnak.

Az utolsó kategória a **mindenütt jelen lévő számítástechnika** (**ubiquitous computing**), amely esetében a számítástechnika beágyazódik a mindennapi életbe, ahogy az

Mark Weiser víziójában olvasható [1991]. Sok otthon már ma is fel van szerelve vezetékes biztonsági rendszerekkel, amelyek ajtó- és ablaknyitás-érzékelőket foglalnak magukba, és sok más érzékelő is létezik, amelyeket okos házakat felügyelő rendszerekbe lehet beépíteni. Ilyen például az energiafelhasználás mérése. A villany-, gáz- és vízórák a hálózaton keresztül is bejelenthetnék a felhasznált mennyiséget. Ez pénzt takarítana meg, mivel nem lenne szükség leolvasók kiküldésére. És a füstérzékelők hívhatnák a tűzoltóságot ahelyett, hogy nagy zajt csapnak (aminek vajmi kevés értelme van, ha senki sincs otthon). Ahogy az érzékelés és a telekommunikáció költségei csökkennek, egyre több mérés és bejelentés fog a hálózatokon keresztül megtörténni.

Egyre több fogyasztói elektronikus készüléket látnak el hálózati képességekkel. Például néhány felsőkategóriás fényképezőgép már vezeték nélküli hálózati kapcsolaton is képes kommunikálni, mely arra használható, hogy az elkészült fényképek egy közelben levő képernyőn jelenjenek meg. A hivatásos sportfotósok szintén valós időben küldhetik el fényképeiket a szerkesztőknek, először vezeték nélkül egy bázisállomásig, majd onnan az interneten keresztül tovább. Televíziók és hasonló készülékek, melyeket konnektorba kell dugni, használhatják az **erősáramú hálózatokat (power-line network)**[1] információ továbbítására a házban, mégpedig ugyanazon a vezetéken, amely a villanyáramot is továbbítja. Lehet, hogy nem túlságosan meglepő, hogy ezek a használati tárgyak hálózatba vannak kötve, de olyan berendezések is érzékelhetnek és kommunikálhatnak, melyekre egyáltalán nem gondolunk számítógépként. Például a zuhany rögzítheti a vízhasználatot, vizuális visszajelzéseket mutathat, miközben Ön beszappanozza magát, és jelentést készíthet egy otthoni környezetfigyelő alkalmazás számára, amikor végzett, hogy segítsen a vízszámlán takarékoskodni.

Egy **RFID**-nak nevezett technika (**Radio Frequency IDentification – rádiófrekvenciás azonosítás**) még messzebbre fogja vinni ezt az ötletet a jövőben. Az RFID-címkék bélyeg méretű passzív chipek (azaz nincs saját áramforrásuk), melyek máris jelen lehetnek könyveken, útlevelekben, háziállatokon, hitelkártyákon vagy más otthoni és azon kívüli cikkeken. Lehetővé teszi az RFID-olvasók számára, hogy megtalálják ezeket a termékeket és akár több méter távolságból is kommunikáljanak velük, az RFID-technika fajtájától függően. Eredetileg az RFID-et a vonalkódok leváltására vezették be. Még nem érte el a célját, mert a vonalkódok ingyenesen létrehozhatók, míg az RFID-címkék előállításának pár centes költsége van. Természetesen az RFID-címkék sokkal több mindenre képesek, és az áruk is gyorsan csökken. Átváltoztathatják a valós világot a dolgok internetévé [ITU, 2005].

1.1.3. Mozgó felhasználók

A hordozható számítógépek, mint például a noteszgépek (laptopok) és a kézi számítógépek (handheld computer) piaca a számítógépipar egyik leggyorsabban fejlődő részterülete. Az eladási mutatóik már megelőzték az asztali számítógépekét. Miért akarna bárki egy ilyet? Az emberek utazás közben gyakran akarják arra használni a hordozható elektronikus eszközeiket, hogy elektronikus leveleket, közösségi hálózatos üzeneteket küldjenek és fogadjanak, filmeket nézzenek meg, zenét töltsenek le, játékokat játsszanak

1 Itt a 230 voltos kisfeszültségű hálózatról van szó. *(A lektor megjegyzése)*

vagy egyszerűen csak szörföljenek a világhálón. Csupa olyan dolgot szeretnének csinálni, amit egyébként otthon vagy az irodában is. És szeretnék mindezt földön, vízen, levegőben bárhonnan megtenni.

Az internettel való **összekapcsolhatóság (connectivity)** sokat lehetővé tesz ezek közül a mobil felhasználási módok közül. Mivel az autókban és a repülőkön lehetetlen kábeles összeköttetést létesíteni, nagy az érdeklődés a vezeték nélküli hálózatok iránt. A telefontársaságok által üzemeltetett mobiltelefon-hálózatok a vezeték nélküli hálózatok legismertebb fajtája, mely lefedettséget biztosít a mobiltelefonok számára. A 802.11 szabványon alapuló vezeték nélküli **aktív helyek (hotspot)** pedig egy másik fajta vezeték nélküli hálózatot biztosítanak a mozgó számítógépek számára. Mindenütt feltűnnek, amerre az emberek járnak, foltszerű lefedettséget eredményezve kávézókban, szállodákban, repülőtereken, iskolákban, vonatokon és repülőgépeken. Bárki, akinek van noteszgépe és vezeték nélküli modemje, csak bekapcsolja a számítógépét, és máris van internetkapcsolata ugyanúgy, mintha számítógépét egy vezetékes hálózatra csatlakoztatta volna.

A vezeték nélküli hálózatok nagy értéket képviselnek a kamion, taxi vagy kézbesítő járművek és a szerelők központtal való kapcsolattartásában. Sok városban például a taxisofőrök egyéni vállalkozók, nem pedig egy taxivállalat alkalmazottai. Ilyen esetekben néha egy képernyő található a taxikban, amit a vezető lát. Amikor egy ügyfél telefonál, a központ diszpécsere begépeli a felvétel és a címzett állomás helyét. Ezt az információt megmutatják a sofőrök képernyőin, és elhangzik egy sípszó. Az első taxisofőr, aki megnyomja a megfelelő gombot, kapja meg a fuvart.

A vezeték nélküli hálózatok a hadsereg számára is fontosak. Ha bárhol a Földön röviddel a parancs kiadása után képesnek kell lenni arra, hogy megvívjanak egy háborút, valószínűleg nem jó ötlet, ha a helyi hálózati infrastruktúra felhasználására alapoznak. Jobb, ha viszik magukkal a sajátjukat.

Bár a vezeték nélküli hálózati technika és a mozgó számítástechnika gyakran összefügg, a kettő nem azonos, amint azt az 1.5. ábra is mutatja. Az ábrán a **telepített (fixed) vezeték nélküli** és a **mozgó vezeték nélküli** hálózatok közötti különbség látható. Néha még a noteszgépek is vezetékekkel csatlakoznak a hálózathoz. Például, ha egy utazó bedugja a noteszgépét a telefonaljzatba egy szállodai szobában, akkor ő mozgó eszközt használ, mégsem vezeték nélküli hálózaton van.

Másrészt viszont néhány vezeték nélküli számítógép nem mozgó eszköz. Olyan otthonokban, irodákban vagy szállodákban, ahol nem áll rendelkezésre a szükséges kábelezés, kényelmesebb lehet az asztali számítógépeket vagy médialejátszókat vezeték nélkül csatlakoztatni, mint vezetékeket telepíteni. Előfordulhat, hogy egy vezeték nélküli hálózat telepítéséhez mindössze arra van szükség, hogy vegyenek egy kis dobozt némi

Vezeték nélküli	Mozgó	Alkalmazások
Nem	Nem	Asztali számítógép egy irodában
Nem	Igen	Noteszgép használata egy szállodai szobában
Igen	Nem	Hálózat egy bekábelezetlen épületben
Igen	Igen	Kézi számítógéppel áruházi leltározás

1.5. ábra. *A vezeték nélküli hálózat és a mozgó számítástechnika különböző kombinációi*

elektronikával, azt kicsomagolják és bedugják a konnektorba. Ez a megoldás sokkal olcsóbb lehet, mintha munkások hada kábelezné be az épületet, kábelcsatornákat szerelve fel mindenhová.

Vannak azonban igazi mozgó vezeték nélküli alkalmazások is, amilyen például egy áruházi leltárt készítő, kézi számítógéppel fel-alá sétáló ember. Sok, nagy forgalmú repülőtéren az autókölcsönzők parkolóban dolgozó alkalmazottai vezeték nélküli hordozható számítógépet használnak. Leolvassák a visszaérkező autók vonalkódját vagy RFID-chipjét, a hordozható gép pedig, amelynek beépített nyomtatója van, felhívja a központi számítógépet, megszerzi tőle a kölcsönzési adatokat, és a helyszínen kinyomtatja a számlát.

A hordozható, vezeték nélküli alkalmazások kulcsfontosságú hajtóereje talán a mobiltelefon. A **rövidüzenet-szolgáltatás** vagy **SMS** (**Short Message Service**, **text messaging**, **texting**) félelmetesen népszerű. A mobiltelefon felhasználója begépelhet egy rövid üzenetet, melyet aztán a mobiltelefon-hálózat kézbesít egy másik előfizetőnek. Kevesen merték volna azt jósolni tíz évvel ezelőtt, hogy a rövid szöveges üzeneteket a mobiltelefonjukon unottan billentyűző tizenévesek mérhetetlen mennyiségű pénzt hoznak a telefontársaságoknak. De az SMS-ezés nagyon is jövedelmező, mivel a szolgáltatónak csak egy cent töredékébe kerül továbbítania egy szöveges üzenetet, és a szolgáltatásért ennél sokkal többet számláz.

A telefonok és az internet régóta várt konvergenciája végre elérkezett, és fel fogja gyorsítani a hordozható alkalmazások növekedését. Az **okostelefonok** (**smart phone**), mint a népszerű iPhone, a mobiltelefonok és a hordozható számítógépek tulajdonságait egyesítik. A (3G-s és 4G-s) mobiltelefon-hálózatok, amelyekhez ezek kapcsolódnak, nagy sebességű adatszolgáltatást nyújtanak az internet használatához, és a telefonhívások lekezeléséhez. Sok fejlett telefon képes vezeték nélküli aktív helyekhez is kapcsolódni, és automatikusan vált a hálózatok között, hogy a felhasználó számára leginkább megfelelő lehetőséget válassza.

Más fogyasztói elektronikus eszközök is képesek mobiltelefon-hálózatokhoz és aktív helyekhez kapcsolódni, hogy távoli számítógépekkel legyenek folyamatos összeköttetésben. Az elektronikus könyvolvasók letölthetnek egy frissen megvásárolt könyvet, egy folyóirat következő számát vagy a napi újságot, amikor fellépnek a hálózatra. A digitális képkeretek időben frissíthetik a kijelzőjüket az újonnan készített fényképekkel.

Mivel a mobiltelefonok ismerik saját helyüket, mert gyakran beépített **GPS-** (**Global Positioning System – globális helymeghatározó rendszer**) vevővel rendelkeznek, néhány szolgáltatás szándékosan helyfüggő. A mobil térképek és útvonaltervezők nyilvánvaló példái ennek, mivel a GPS-képes telefon és autó valószínűleg sokkal jobban tudja, hogy hol van, mint az ember maga. Ugyanez igaz a közeli könyvesboltra vagy kínai étteremre történő kereséskor, vagy a helyi időjárás-jelentéssel kapcsolatban. Más szolgáltatások rögzíthetik a hely adatait, például felcímkézhetik a fényképeket vagy videókat annak a helynek a koordinátáival, ahol készültek. Ennek a jelölési módszernek a neve „geo-tagging".

Egy új terület, amelyen mostanában kezdik használni a mobiltelefonokat, az **m-kereskedelem** (**mobile commerce – mozgó kereskedelem**) [Senn, 2000]. A mobiltelefon rövid szöveges üzeneteit használja a kifizetések engedélyezésére az étel- és italárusító automatáknál, mozijegyek esetében és más kis tételekhez a készpénz vagy a hitelkártyák helyett. A követelések ezt követően a mobiltelefon-számlán jelennek meg. Ha beépítésre került a mobiltelefonba az **NFC-** (**Near Field Communication – közeltéri kommunikáció**) technika, akkor viselkedhet RFID intelligens adatkártyaként, és kommunikálhat egy közeli olvasóval a fizetés lebonyolítása érdekében. A jelenség mögött megbúvó hajtóerő a hordozható eszközök gyártóiból és a hálózatüzemeltetőkből álló szövetség, akik nagy erőbedobással próbálják kitalálni, hogyan kaphatnának egy szeletet az e-kereskedelem tortájából. A bolt számára ez az eljárás megtakaríthatja a hitelkártya-társaságoknak fizetendő díj nagy részét, ami adott esetben több százalék is lehet. Természetesen ez a terv visszaüthet, mivel egy bolt vásárlói arra is használhatják a mobiltelefonjuk RFID- vagy vonalkódolvasóját, hogy vásárlás előtt megnézzék a versenytársak árait is, lehetővé téve, hogy azonnal részletes jelentést kapjanak arról, hogy hol máshol és milyen áron tudják ugyanazt megvenni.

Nagyban elősegíti az m-kereskedelmet az a tény, hogy a mobiltelefon-használók hozzá vannak szokva, hogy mindenért fizetniük kell (ellentétben az internethasználókkal, akik elvárják, hogy minden ingyenes legyen). Ha egy internetes webhely díjat számítana fel azért, hogy a kártyás fizetést lehetővé tegye vásárlói számára, akkor nagy füttykoncertre számíthatna felhasználói részéről. Ha egy mobiltelefon-szolgáltató próbálná meg lehetővé tenni az előfizetőinek, hogy egy áruházban az árukért úgy fizessenek, hogy a telefonjukat meglóbálják a kassza előtt, és ezért a kényelemért egy bizonyos összeget csapna hozzá a számlához, azt valószínűleg normálisnak fogadnánk el. Ezt a kérdést az idő dönti majd el.

Kétségtelen, hogy a hordozható és vezeték nélküli számítógépek felhasználása a jövőben gyorsan fog növekedni, ahogy a számítógépek egyre kisebb méretűek lesznek, mégpedig valószínűleg senki által előre nem látott módon. Tekintsünk meg ezek közül néhányat! A **szenzorhálózatok** (**sensor network**) olyan csomópontokból állnak, amelyek gyűjtik és vezeték nélkül továbbítják az információt, amit a fizikai világ állapotával kapcsolatban érzékelnek. A csomópontok olyan ismerős használati eszközök alkatrészei lehetnek, mint az autók vagy telefonok, de lehetnek kisméretű, önálló készülékek is. Például az autónk a fedélzeti diagnosztikai rendszer segítségével adatokat gyűjthet a pozíciójáról, a sebességről, a rázkódásáról és az üzemanyag-felhasználás hatékonyságáról, és feltöltheti ezeket az adatokat egy adatbázisba [Hull és mások, 2006]. Ezen adatok segítségével azonosíthatók a kátyúk, dugókat elkerülő útvonalakat tervezhetünk vagy megmondhatjuk, hogy „üzemanyag-zabálók" vagyunk-e az azonos útszakaszon közlekedőkhöz viszonyítva.

A szenzorhálózatok azáltal forradalmasítják a tudományt, hogy olyan viselkedésekről szolgáltatnak gazdag adatokat, amelyeket korábban nem lehetett megfigyelni. Erre példa az egyes zebrapéldányok vonulásának követése úgy, hogy egy kis érzékelőt helyezünk el minden állaton [Juang és mások, 2002]. Kutatóknak már sikerült egy vezeték nélküli számítógépet egy 1 mm oldalhosszúságú kockába belegyömöszölniük [Warneke és mások, 2001]. Ilyen kisméretű hordozható számítógépekkel lehetővé válhat akár kis madarak, rágcsálók vagy rovarok nyomon követése is.

A parkolóórákhoz hasonló földhözragadt felhasználási módok is jelentősek lehetnek, mert olyan adatokat dolgozhatnak fel, amelyek korábban nem álltak rendelkezésre. Az óra elfogadhatna hitelkártyát, és azonnal ellenőrizhetné is a vezeték nélküli kapcsolaton keresztül. Szintén bejelenthetné a vezeték nélküli hálózaton keresztül, hogy mikor van használatban. Ezáltal a sofőrök letölthetnének az autójukra egy friss parkolási térképet, hogy könnyebben találjanak szabad helyet. Amikor az óra lejár, természetesen megbizonyosodhatna róla, hogy az autó ott van-e még (például egy jel küldésével), és bejelenthetné a lejárat tényét a rendőrségnek vagy a parkolót üzemeltető társaságnak.

Az elvégzett becslések szerint csak az Egyesült Államok önkormányzatai így 10 milliárd dollárnyi többletbevételhez juthatnának [Harte és mások, 2000].

A **viselhető számítógépek (wearable computer)** további ígéretes alkalmazási lehetőséget alkotnak. Az okos, rádiós órákat már a Dick Tracy-képregényekben való 1946-os megjelenésük óta a lehetséges jövőként tartottuk számon; most pedig már megvásárolhatók. Más hasonló eszközök beültethetők, például szívritmus-szabályozók vagy inzulinpumpák. Némelyik vezeték nélküli hálózaton keresztül vezérelhető is. Ezáltal az orvosok könnyebben tudják ellenőrizni és finoman hangolni a működésüket. De kellemetlen problémákhoz is vezethet, ha ezek az eszközök olyan kevéssé biztonságosak, mint az átlagos PC, és könnyedén feltörhetők [Halperin és mások, 2008].

1.1.4. Társadalmi vonatkozások

A számítógép-hálózatok, akárcsak a nyomdászat 500 évvel ezelőtt, az átlagpolgárok számára lehetővé teszi, hogy új eszközök segítségével olyan módon terjesszenek és fogyasszanak tartalmakat, ahogy az korábban nem volt lehetséges. Sajnos a jóval együtt jár a rossz is, és ez az újfajta szabadság számos, egyelőre megoldatlan társadalmi, politikai és morális problémát vet fel. Hadd említsünk meg csak néhányat közülük, tekintettel arra, hogy egy alaposabb tanulmány legalább egy teljes könyvet kitenne.

A közösségi hálózatokon, hirdetőtáblákon, tartalommegosztó oldalakon, és temérdek sok más alkalmazás révén a hasonló gondolkodású emberek kicserélhetik véleményüket. Amíg a témák a műszaki dolgok vagy a hobbik (például a kertészkedés), addig nincs is túl sok probléma.

A gond akkor kezdődik, amikor olyan témákra terelődik a szó, amelyekre az emberek érzékenyek. Ilyen például a politika, a vallás vagy a szex. Az ilyen, nyilvánosan közzétett vélemények mélyen sérthetnek másokat. Vagy ami még rosszabb, nem mindig politikailag korrektek. Ráadásul az üzenetek nem feltétlenül csak szövegre korlátozódnak. Nagy felbontású színes fényképeket, de akár még videoklipeket is könnyű a számítógép-hálózaton keresztül másokkal megosztani. Vannak, akik követik az *„élni és élni hagyni"* szemléletet, de vannak olyanok is, akik úgy gondolják, hogy bizonyos anyagok (például szóbeli támadások bizonyos országok vagy vallások ellen, pornográfia stb.) közzététele teljesen elfogadhatatlan, és cenzúrára szorul. Különböző országokban eltérő és egymásnak ellentmondó törvények vannak érvényben ezen a területen. Ezért a viták könnyen eldurvulnak.

Voltak, akik beperelték a hálózatok üzemeltetőit, mivel szerintük – hasonlóan az újságokhoz és a folyóiratokhoz – ők a felelősek a rajtuk keresztül továbbított anyagok tartalmáért. A nyilvánvaló válasz az volt, hogy a számítógép-hálózat olyan, mint a telefonhálózat vagy a postahivatal, azaz nem várható el tőle, hogy felügyeljen arra, mit mondanak a felhasználók.

Ezek után bizonyára nem okoz nagy meglepetést, hogy néhány hálózatüzemeltető saját indítékai alapján tilt bizonyos tartalmakat. A P2P-alkalmazások felhasználói között előfordult már, hogy a hálózati hozzáférésüket megszüntették, mert a hálózat üzemeltetője nem találta kifizetődőnek, hogy az ilyen alkalmazások által generált nagy mennyiségű forgalmat továbbítsa. Ugyanezek az üzemeltetők valószínűleg szívesen bánnának eltérő módon a különböző cégekkel. Egy nagy és jól fizető társaság jó minőségű szolgáltatást kapna, egy piti kis szereplő pedig szegényesebb szolgáltatást. Ennek a gyakorlatnak az ellenzői úgy érvelnek, hogy a P2P és egyéb tartalmakat egyformán kellene kezelni, mert ezek mind csak a hálózathoz továbbított bitek. Azt az elvet, hogy a kommunikációt nem különböztetjük meg a tartalma, forrása vagy szolgáltatója alapján, **hálózatsemlegességnek (network neutrality)** nevezzük [Wu, 2003]. Szinte bizonyosak vagyunk afelől, hogy ez a vita még eltart egy darabig.

Sok más fél is részt vesz a tartalom feletti viaskodásban. Például a zene- és filmkalózkodás tüzelte a P2P-hálózatok irdatlan növekedését, ami nem okozott örömet a szerzői jogok birtokosainak, akik jogi közbelépéssel kezdtek fenyegetőzni (és néha be is váltották az ígéretüket). Manapság már automatizált rendszerek keresik a P2P-hálózatokon a jogsértő tartalmakat, és küldenek figyelmeztetést a hálózat üzemeltetőjének, valamint a jogsértéssel gyanúsított felhasználóknak. Az Egyesült Államokban ezeket a figyelmeztetéseket **DMCA eltávolítási felszólításként (DMCA takedown notice)** ismerik a **Digitális Millennium Szerzői Jogi Törvény (Digital Millennium Copyright Act)** szerint. Ez a keresés olyan, mint egy fegyverkezési verseny, mert nehéz megbízhatóan azonosítani a szerzői joggal kapcsolatos jogsértéseket. Még egy nyomtató is kerülhet a vádlottak padjára [Piatek és mások, 2008].

A számítógép-hálózatok révén nagyon egyszerűvé válik a kommunikáció. De azt is nagyon könnyűvé teszik, hogy a hálózat üzemeltetője belelessen a forgalomba. Ez konfliktusok melegágya olyan kérdésekben, hogy melyek az alkalmazottak jogai és melyek a munkavállalók jogai. Sokan olvasnak és írnak elektronikus levelet munka közben. Sok munkaadó követelte magának azt a jogot, hogy elolvashassa, és esetleg cenzúrázhassa is az alkalmazottak üzeneteit, beleértve a munka után, otthoni terminálról elküldött üzeneteket is. Nem minden munkavállaló ért ezzel egyet, különösen az utóbbi felével.

Egy másik kulcsfontosságú téma a kormány és az állampolgár viszonya. Az FBI sok internetszolgáltatónál telepített egy rendszert, ami arra szolgál, hogy megvizsgáljon minden bejövő és kimenő e-levelet, érdekes információ után kutatva bennük. A rendszert eredetileg húsevőnek (Carnivore) hívták, de az ez által keltett rossz sajtóvisszhang miatt átkeresztelték az ártatlanabbul hangzó DCS1000 névre [Blaze és Bellovin, 2000; Sobel, 2001 és Zacks, 2001]. A célja az, hogy az embereket megfigyelés alatt tartsa, annak reményében, hogy illegális tevékenységekre utaló adatokat talál. Az amerikai alkotmány negyedik módosítása megtiltja, hogy a kormány házkutatási engedély nélkül kutattasson bárkinél, de a kémek sajnálatára, a kormányzat ezt sokszor figyelmen kívül hagyja.

A kormánynak persze nincs monopóliuma abban, hogy fenyegetni tudja az emberek magánélethez való jogát. A magánszektor is kiveszi belőle a maga részét a **felhasználói profilok (user profiles)** előállításával. Például a sütiknek (cookie) nevezett kis állományok, amelyeket a webböngészők tárolnak el a felhasználók gépein, lehetővé teszik a vállalatok számára, hogy nyomon kövessék a felhasználó tevékenységét az interneten, sőt még akár azt is lehetővé tehetik, hogy hitelkártyák, igazolványok számai vagy más bizalmas információ szivárogjon ki mindenhová az interneten keresztül [Berghel, 2001]. A webalapú szolgáltatásokat nyújtó vállalatok hatalmas mennyiségű személyes információt tartanak nyilván felhasználóikról, ami lehetővé teszi számukra, hogy közvetlenül tanulmányozzák a felhasználók tevékenységét. Például a Google el tudja olvasni az Ön levelezését, és az érdeklődési köre alapján képes hirdetéseket megjeleníteni önnek, ha a cég e-mail szolgáltatását, a **Gmail**-t használja.

A mozgó eszközök esetében újabb fordulat a földrajzi helyzet bizalmas kezelése [Beresford és Stajano, 2003]. Az Ön hordozható eszköze számára történő szolgáltatás nyújtása során a hálózat üzemeltetője megtudja, hogy hol tartózkodik Ön a nap különböző időszakaiban. Ezzel lehetővé válik, hogy kövesse az Ön mozgását. Azt is megtudhatják, hogy melyik éjszakai szórakozóhelyre szokott járni és melyik orvosi rendelőt látogatja.

A számítógép-hálózatok lehetőséget adnak a személyiségi jogok erősítésére névtelen levelek küldésével is. Bizonyos esetekben ez hasznos. Amellett, hogy megakadályozza a cégeket abban, hogy megismerjék a felhasználók szokásait, lehetővé teszi a diákok, a katonák, a beosztottak és az állampolgárok számára például azt, hogy nyilvánosságra hozzák a tanárok, a hivatalnokok, a feletteseik vagy a politikusok által elkövetett szabálytalanságokat anélkül, hogy megtorlástól kellene tartaniuk. Ugyanakkor az Egyesült Államokban és sok más demokratikus országban a törvények kifejezetten lehetővé teszik azt, hogy a bíróság előtt a vádlott szembesülhessen a vádlójával, így a névtelen vádaskodás nem tekinthető bizonyítéknak.

Az internet lehetővé teszi, hogy a keresett adatokat gyorsan megtaláljuk, de közöttük sok a féligazság, a félrevezető vagy egyenesen helytelen információ. Az orvosi tanács, amit az internetről szerzünk be, lehet, hogy egy Nobel-díjastól származik, de ugyanúgy származhat egy bukott középiskolai diáktól is.

Más információ gyakran nemkívánatos módon érkezik. Az elektronikus levélszemét (junk mail, spam) az élet része lett, mert az ilyen levelek küldői több millió címet gyűjtöttek össze, és az üzletelni szándékozók olcsón küldhetnek számítógép által generált üzeneteket ezekre a címekre. A levélszemét így keletkező áradata vetekszik a valós személyektől érkező üzenetek mennyiségével. Szerencsére a szűrőprogramok kisebb-nagyobb sikerrel képesek elolvasni és kidobni a más számítógépek által generált levélszemetet.

Ismét más tartalmak bűnözői szándékkal készülnek. Az olyan weboldalak vagy e-levelek, amelyek aktív tartalommal is bírnak (ezek lényegében programok vagy makrók, amelyek lefutnak a vevő gépén), olyan vírusokat tartalmazhatnak, amelyek átveszik az irányítást a vevő számítógépe felett. Felhasználhatók banki jelszavak ellopására, vagy hogy rávegyék a számítógépet, hogy egy **botnet**[2], vagyis kompromittálódott számítógépek csoportjának tagjaként levélszemetet küldözgessen.

Az **adathalász (phishing)** üzenetek úgy álcázzák magukat, mintha megbízható féltől származnának, például a felhasználó bankjától, hogy rávegyék a címzettet arra, hogy érzékeny információt fedjen fel, például hitelkártyaszámot. A személyazonosság-tolvajlás is kezd komoly problémává válni, mivel a személyiségtolvajok már elég adatot tudnak összegyűjteni az áldozatukról ahhoz, hogy hitelkártyákat és más személyes iratokat tudjanak szerezni az áldozat nevében.

Nehéz feladat azt megakadályozni, hogy számítógépek embereket személyesítsenek meg az interneten. Ez a probléma vezetett az ún. **CAPTCHA**[3] biztonsági rendszer kifejlesztéséhez, amelynek alkalmazásakor a számítógép megkéri a személyt egy rövid felismerési feladat elvégzésére, például hogy gépeljen be torzított betűket egy képről, hogy így bizonyítsa emberi mivoltát [von Ahn, 2001]. Ez a folyamat a híres Turing-teszt egy változata, melyben az egyik fél kérdéseket tesz fel a hálózaton keresztül a másiknak azzal a céllal, hogy eldöntse, hogy ember-e a válaszadó.

2 A botnet együttműködő, kívülről csoportosan távvezérelhető botok hálózata. A bot valószínűleg a robot szóból ered. *(A lektor megjegyzése)*

3 A betűszó teljes szövege és jelentése: **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part (teljesen automatizált nyilvános (fordított) Turing-teszt a számítógép és az ember megkülönböztetésére). *(A lektor megjegyzése)*

Ezeknek a problémáknak jó részét egyszerűen meg lehetne oldani, ha a számítástechnikai ipar komolyan venné a számítógépes biztonságot. Ha minden üzenetváltás titkosítva és azonosítva lenne, sokkal nehezebb lenne visszaéléseket elkövetni. Ez a technika jól megalapozott témakör, és részletesen is foglalkozni fogunk vele a 8. fejezetben. A probléma az, hogy a hardver- és szoftvergyártók tudják, hogy a biztonsági képességek beépítése pénzbe kerül, és hogy vásárlóik nem is igénylik az ilyen szolgáltatásokat. Továbbá rengeteg problémát okoznak a hibás szoftverek, ami pusztán abból adódik, hogy a fejlesztők egyre több és több képességgel vértezik fel a programjaikat, ami elkerülhetetlenül ahhoz vezet, hogy hosszabb lesz a kód és ezért több lesz benne a hiba. Segíthetne a dolgon, ha ezeket az új képességeket megadóztatnák, de ezt valószínűleg sok helyen igen nehéz lenne eredményesen alkalmazni. Nagyon hasznos lenne például, ha a cégeknek pénzt kellene visszaadniuk a hibás szoftverek miatt, azt az egy dolgot kivéve, hogy ez tönkretenné az egész szoftveripart már az első évben.

A számítógép-hálózatok új jogi problémákat vetnek fel, amikor a régi törvényekkel kerülnek szembe. Jó példa erre az elektronikus szerencsejáték. A számítógépek évtizedek óta szimulálnak dolgokat, tehát miért ne szimulálhatnának játékautomatákat, rulettkerekeket, osztókat a 21-es játékban vagy más szerencsejátékkal kapcsolatos eszközöket? Nos, mert sok helyen ez illegális. A gond az, hogy sokszor a fogadás legális (például Angliában), és a kaszinótulajdonosok felfogták az internetes fogadásban rejlő lehetőségeket. Mi történik, ha a fogadó, a kaszinó és a szerver mind különböző országokban találhatók, amelyek ráadásul ellentmondó törvényekkel rendelkeznek? Jó kérdés.

1.2. Hálózati hardver

Itt az ideje, hogy figyelmünket a hálózatok alkalmazási és társadalmi vonatkozásai után (ez volna a desszert) a hálózattervezés műszaki problémáira (a spenótra) irányítsuk. Nincs olyan, általánosan elfogadott osztályozás, amelybe az összes hálózatot be lehetne sorolni, azonban van két fontosnak tűnő szempont: az átviteli technika és a méret. A továbbiakban ezt a kettőt vizsgáljuk.

Az átviteli technikának – tágabb értelemben véve – két széles körben használt típusa van: **adatszóró (broadcast)** átvitel és **kétpontos (point-to-point)** átvitel.

A kétpontos kapcsolat számítógép-párokat kötnek össze. Ahhoz, hogy kétpontos kapcsolatokból álló hálózaton eljusson a feladótól a címzettig egy rövid üzenet, melyet bizonyos körülmények között **csomagnak (packet)** is szoktak nevezni, lehet, hogy egy vagy több közbenső gépen is át kell mennie. Sokszor több, különböző hosszúságú útvonal is lehetséges, ezért a jó útvonalak megtalálása fontos kérdés a kétpontos hálózatokban. A kétpontos átvitelt, ahol egy adó és egy vevő van csak jelen, néha **egyesküldésnek (unicasting)** is szokták nevezni.

Ezzel szemben az adatszóró hálózatok egyetlen, közös kommunikációs csatornával rendelkeznek, és ezen osztozik a hálózat összes gépe. Ha bármelyik gép elküld egy csomagot, azt az összes többi gép megkapja. A címzettet a csomagon belüli címmezőben lehet megjelölni. Amikor egy gép csomagot kap, megnézi a címmezőt. Ha a csomagot neki szánták, akkor feldolgozza azt, ha pedig nem neki szánták, akkor figyelmen kívül hagyja.

Egy vezeték nélküli hálózat jó példája az adatszóró kapcsolatnak, ahol a kommunikáció közös egy lefedettségi tartományon belül, melyet a vezeték nélküli csatorna és az adógép határoz meg. Az analógia kedvéért képzeljük el, hogy valaki egy sokajtós folyosó végén elkiáltja magát: „Pista, gyere ide! Beszélni akarok veled.". Bár a felszólítást mindenki hallja, mégis csak Pista fog válaszolni. A többiek egyszerűen nem vesznek róla tudomást.

Az adatszórórendszerek általában lehetővé teszik, hogy a csomag címmezőjében egy speciális kód beállításával *minden* gép megcímezhető legyen. Ha az ilyen kóddal ellátott csomagot elküldjük, akkor a hálózat összes gépe megkapja és feldolgozza azt. Ezt a működési módot **adatszórás**nak (**broadcasting**) nevezzük. Néhány adatszórórendszerben arra is lehetőség nyílik, hogy a gépeknek csak egy meghatározott csoportját címezzük meg. Ez a **többesküldés** (**multicasting**).

Egy másik lehetséges szempont a hálózatok osztályozására, a méretük. A távolság azért fontos osztályozási szempont, mert eltérő mérettartományokban más-más technikát használnak.

Az 1.6. ábrán a több feldolgozóegységet tartalmazó rendszereket hozzávetőleges fizikai méretük szerint soroljuk osztályokba. Az ábra tetején a személyi hálózatok (personal area network) helyezkednek el. Ezek olyan hálózatok, amelyeket egyetlen embernek szántak. Ezután következnek a nagyobb kiterjedésű hálózatok. Ezeket helyi, nagyvárosi és nagy kiterjedésű hálózatokra bonthatjuk fel, méret szerint növekvő sorrendben. Végül pedig, a két vagy több hálózat összekapcsolásával létrejött hálózatot összekapcsolt hálózatnak (internetwork) hívjuk. Az egész világot átfogó internet jól ismert példa (de

Processzorok közötti távolság	Processzorok elhelyezkedése ugyanazon a(z)	Példa
1 m	Asztalon	Személyi hálózat
10 m	Szobában	Lokális hálózat
100 m	Épületben	
1 km	Egyetemen	
10 km	Városban	Nagyvárosi hálózat
100 km	Országban	Nagy kiterjedésű hálózat
1000 km	Földrészen	
10 000 km	Bolygón	Internet

1.6. ábra. *Összekapcsolt feldolgozó egységek osztályozása kiterjedés szerint*

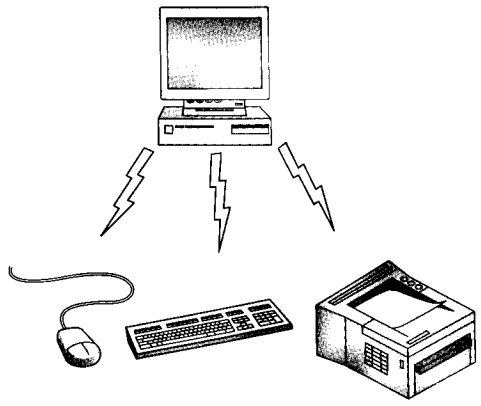
nem ez az egyetlen) egy összekapcsolt hálózatra. Hamarosan még nagyobb összekapcsolt hálózataink lesznek, ha megvalósul a **bolygóközi internet** (**Interplanetary Internet**), mely a világűrön keresztül köt majd össze hálózatokat [Burleigh és mások, 2003].

1.2.1. Személyi hálózatok

A **személyi hálózat** (**Personal Area Network, PAN**) segítségével egy ember környezetében levő eszközök kommunikálhatnak egymással. Jellemző példa az olyan vezeték nélküli hálózat, mely egy számítógépet köt össze a perifériáival. Majdnem minden számítógéphez csatlakoztatunk képernyőt, billentyűzetet, egeret és nyomtatót. Vezeték nélküli technikák használata híján ezeket a kapcsolatokat vezetékkel kellene kiépíteni. Megannyi új felhasználó számára okoz gondot megtalálni a megfelelő kábelt és bedugni a megfelelő lyukba (annak ellenére, hogy ezek általában színkódoltak), hogy a legtöbb számítógépkereskedő lehetőséget biztosít arra, hogy egy technikus végezze el ezeket a feladatokat a felhasználó otthonában. Ezeknek a felhasználóknak a megsegítésére néhány cég közösen létrehozott egy **Bluetooth**-nak nevezett vezeték nélküli hálózati technikát, melynek révén az ilyen eszközök kábelek nélkül csatlakoztathatók. Az elgondolás az, hogy ha az eszközök Bluetooth-képesek, akkor nincs szükség vezetékekre. Csak le kell tenni őket, be kell kapcsolni, és már működnek is együtt. Sokak számára ez az egyszerű kezelés nagy pozitívum.

A legegyszerűbb esetben a Bluetooth-hálózatok az 1.7. ábrán szemléltetett mester–szolga működésmódot követik. Általában a rendszeregység (a számítógép) a mester, és az egérhez, billentyűzethez stb. mint szolgákhoz beszél. A mester mondja meg a szolgáknak, hogy milyen címeket használjanak, mikor sugározhatnak, mennyi ideig adhatnak, milyen frekvenciákat használhatnak és így tovább.

A Bluetooth más alkalmazásokban is használható. Gyakran használják arra, hogy vezeték nélkül illesszenek fülhallgatót és mikrofont mobiltelefonhoz, valamint lehetővé teszi azt is, hogy a digitális zenelejátszó készülék kizárólag azáltal kapcsolódjon az autóhoz, hogy a közelébe viszik. Egy teljesen eltérő típusú PAN jön létre, amikor egy beültetett orvosi eszköz, például szívritmus-szabályozó, inzulinpumpa vagy hallókészülék



1.7. ábra. *Bluetooth PAN-konfiguráció*

kommunikál a felhasználó által kezelt távirányítóval. Részletesebben is tárgyaljuk majd a Bluetooth technikát a 4. fejezetben.

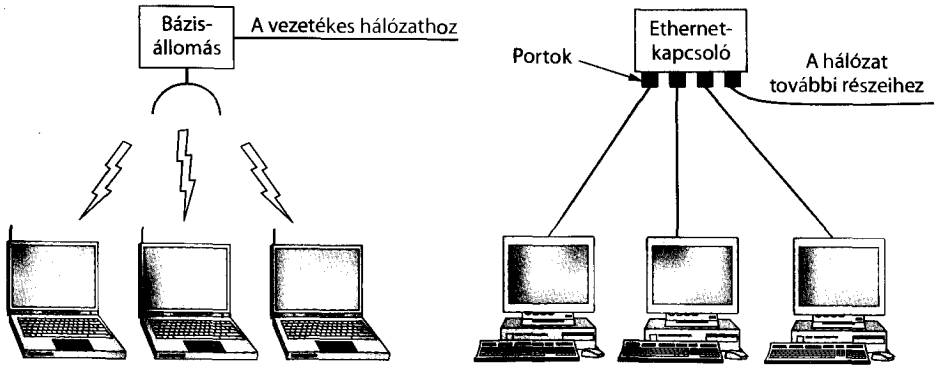
PAN-hálózatok más olyan technikákkal is létrehozhatók, melyek rövid távon kommunikálnak, mint az intelligens adatkártyákon és könyvtári könyveken jelen levő RFID. Az RFID-et a 4. fejezetben fogjuk tanulmányozni.

1.2.2. Lokális hálózatok

A nagyobb kiterjedés felé haladva a következő hálózat a **lokális hálózat** (**Local Area Network, LAN**). A LAN olyan magánhálózat, amely egyetlen épületen belül vagy annak környezetében üzemel, például egy lakásban, irodában vagy gyártelepen. Széles körben használják ezeket személyi számítógépek, valamint fogyasztói elektronikus eszközök összekapcsolására, lehetővé téve ezzel a közös erőforrások (például nyomtatók) megosztását és az információcserét. Amikor nagy cégek használnak LAN-okat, **vállalati hálózatokról** beszélünk (**enterprise network**).

Manapság nagyon népszerűek a vezeték nélküli LAN-ok, különösen lakásokban, régebbi irodaházakban, kávézókban és más olyan helyeken, ahol túl sok bajjal járna vezetékeket telepíteni. Ezekben a rendszerekben minden számítógépnél találhatunk egy rádiós modemet és egy antennát, melyeket arra használ, hogy más számítógépekkel kommunikáljanak. A legtöbb esetben minden számítógép az 1.8.(a) ábrán mutatott módon egy, a plafonra szerelt eszközzel beszélget. Ez az eszköz, melyet **hozzáférési pontnak** (**Access Point, AP**), **vezeték nélküli útválasztónak** (**wireless router**) vagy **bázisállomásnak** (**base station**) neveznek, csomagokat továbbít a vezeték nélküli számítógépek között, vagy köztük és az internet között. AP-nak lenni olyan, mint népszerű gyereknek lenni az iskolában, mert mindenki vele akar beszélgetni. Ám ha a többi számítógép elég közel van egymáshoz, közvetlenül is kommunikálhatnak P2P- (peer-to-peer) kiépítésben.

Létezik egy **IEEE 802.11** nevű szabvány a vezeték nélküli LAN-okra, népszerű nevén a **Wi-Fi**, mely név nagyon széleskörűen elterjedt. Ez 11 Mb/s-tól akár több száz Mb/s sebességgel képes működni. Ebben a könyvben ragaszkodni fogunk a hagyományokhoz, és megabit/sec-ban (1 Mb/s megfelel 1 000 000 b/s-nak), illetve gigabit/sec-ban (1 Gb/s



1.8. ábra. *Vezeték nélküli és vezetékes LAN-ok. (a) 802.11. (b) Kapcsolt Ethernet*

megfelel 1 000 000 000 b/s-nak) fogjuk mérni a vonalak sebességét. A 802.11 technikát a 4. fejezetben tárgyaljuk.

A vezetékes LAN-ok különféle átviteli technikákat alkalmaznak. A legtöbb LAN rézvezetéket használ, de némelyik fényvezető szálat. A LAN-ok mérete szigorúan korlátos, így az átviteli idő a legrosszabb esetben is korlátos és előre ismert. Az időkorlát ismerete segít a hálózati protokollok tervezésekor. A vezetékes LAN-ok jellemzően 100 Mb/s vagy 1 Gb/s sebességgel üzemelnek, alacsony a késleltetésük (néhány mikro- vagy nanoszekundum), és nagyon kevés hibát vétenek. Az újabb LAN-ok sebessége egészen 10 Gb/s-ig terjed. A vezeték nélküli hálózatokkal összehasonlítva a vezetékes LAN-ok minden téren jobbak. Egyszerűen könnyebb vezetéken vagy üvegszálon jeleket küldeni, mint a levegőn keresztül.

Sok vezetékes LAN topológiája kétpontos kapcsolatokból épül fel. Az IEEE 802.3-as szabvány – vagy népszerűbb nevén az **Ethernet** – messze a leggyakoribb típusa a vezetékes helyi hálózatoknak. Az 1.8.(b) ábrán példát láthatunk a **kapcsolt Ethernet** topológiára (**switched Ethernet**). Minden számítógép az Ethernet-protokollt használja, és egy kétpontos kapcsolattal egy **kapcsolónak** (**switch**) nevezett eszközhöz kapcsolódik. A kapcsolónak több **portja** van (kapu, bemeneti-kimeneti csatlakozási pont), melyekhez egy-egy számítógép csatlakozhat. A kapcsoló feladata az, hogy továbbítsa a csomagokat[4] a csatlakoztatott számítógépek között, a csomagokban levő címzés alapján meghatározva, hogy melyik számítógépnek kell küldeni.

Nagyobb LAN-ok építéséhez a kapcsolók egymásba is bedughatók a portjaikon keresztül. Mi történik, ha körkörösen csatlakoztatjuk őket? Működni fog így is a hálózat? Szerencsére a tervezők gondoltak erre az eshetőségre is. A hálózati protokoll feladata, hogy kitalálja, milyen útvonalon kell utazniuk a csomagoknak, hogy biztonságosan elérjék a kívánt számítógépet. A 4. fejezetben látni fogjuk, hogy ez hogyan működik.

Szintén lehetséges egy nagy fizikai LAN két kisebb, logikai részre osztása. Felmerül a kérdés, hogy miért lehet ez hasznos. Néha a hálózati eszközök elhelyezése nem követi a szervezet struktúráját. Például egy vállalat mérnöki és pénzügyi osztályához tartozó számítógépek ugyanazon a fizikai LAN-on lehetnek, mert ugyanabban az épületszárnyban vannak, de egyszerűbb lehet a rendszert felügyelni, ha a mérnöki és pénzügyi osztályok saját logikai hálózattal, ún. **virtuális helyi hálózattal** (**Virtual LAN, VLAN**) rendelkeznek. Ebben a felállásban minden portot megcímkézünk egy „színnel", mondjuk zölddel a mérnöki és pirossal a pénzügyi területet. Ezt követően a kapcsoló úgy továbbítja a csomagokat, hogy a zöld portokhoz csatlakoztatott számítógépek külön vannak választva a piros portokba dugott gépektől. Például egy piros porton küldött adatszóró csomag nem érkezik meg a zöld portokra, mintha két különálló LAN létezne. A VLAN-okról a 4. fejezet végén lesz szó.

Léteznek más LAN-topológiák is. Valójában a kapcsolt Ethernet annak az eredeti Ethernet-kialakításnak a modern változata, amely egyetlen közös kábelszakaszon továbbította a csomagokat adatszórással. Egyszerre legfeljebb egy gép volt képes sikeresen adni, és egy elosztott döntési folyamat oldotta fel az ütközéseket. Ez egy egyszerű algoritmuson alapult: a számítógépek bármikor adhattak, amikor a kábel kihasználatlan volt. Ha két vagy több cso-

4 Valójában a kapcsoló nem csomagokat, hanem kereteket továbbít a keretben lévő cím alapján. A csomagok a keretekbe ágyazódva kerülnek továbbításra (lásd 4. fejezet). *(A lektor megjegyzése)*

mag ütközött, minden számítógép várt egy véletlenszerű időtartamot, és később újra próbálkozott. Ezt a változatot az egyértelműség céljából **klasszikus Ethernetnek** vagy **normál Ethernetnek** fogjuk hívni, és ahogy sejthető, a 4. fejezetben fogunk róla többet megtudni.

Mind a vezeték nélküli mind a vezetékes adatszóró hálózatok működési elvük szerint lehetnek statikusak vagy dinamikusak aszerint, hogy a csatorna-hozzárendelés hogyan megy végbe. A statikus hozzárendelés egyik tipikus esete az, amikor diszkrét időintervallumokat definiálunk, és egy körforgó algoritmus szerint minden gép csak akkor küldhet adatszórással üzenetet, amikor elérkezett az ő időszelete. A statikus hozzárendelés elpazarolja a csatornakapacitást, amikor egy gépnek nincs továbbítandó üzenete a neki szánt időszelet alatt, emiatt a legtöbb rendszer inkább a dinamikus (azaz kérés alapján történő) csatorna-hozzárendeléssel próbálkozik.

A közös csatorna dinamikus hozzárendelésekor központosított és elosztott módszerek léteznek. Centralizált csatorna-hozzárendelés esetén mindig van egy olyan egység, amely meghatározza, hogy ki adhat következőnek (például a bázisállomás a mobiltelefon-hálózatok esetében). Egyik lehetséges módja ennek, hogy miután megkapta a kéréseket, valamilyen belső algoritmus alapján hoz döntést. Elosztott csatorna-hozzárendelés esetén nincs központi egység, hanem mindegyik gépnek magának kell eldöntenie, hogy ad-e vagy sem. Azt gondolhatnánk, hogy ez folyton káoszt eredményez, pedig nincs így. A későbbiekben látni fogunk olyan algoritmusokat, amelyeket arra találtak ki, hogy káoszveszély esetén rendet teremtsenek.

Érdemes egy kis időt szánnunk az otthoni LAN-ok kérdésére. Valószínű, hogy a jövőben minden háztartási gép képes lesz kommunikálni a többi otthoni eszközzel, és mindet el lehet majd érni az interneten keresztül. Ez a fejlesztés egyik azok közül a látnoki elgondolások közül, amelyeket senki sem kért (mint a tv-távirányítót vagy a mobiltelefont), de amint megjelentek, senki sem tudja elképzelni, hogyan is élhetett nélkülük.

Sok eszköznek már most is vannak hálózati képességei. Idetartoznak a számítógépek, olyan szórakoztatóelektronikai eszközök, mint a televíziók és DVD-lejátszók, telefonok és más fogyasztói elektronikus eszközök, mint a fényképezőgépek, rádiós ébresztőórák, és az infrastruktúra olyan elemei is, mint a közművek fogyasztásmérői vagy a hőfokszabályzók. Ez az irányvonal csak folytatódni fog. Például egy átlagos otthonban legalább egy tucat óra van (például különböző készülékekben), melyek mind alkalmazkodhatnának a nyári és téli időszámításhoz, ha elérnék az internetet. Valószínűleg a lakás távellenőrzése lenne a győztes megoldás, mivel sok felnőtt gyermek lenne hajlandó pénzt költeni arra, hogy segítsen az idősödő szüleinek, hogy biztonságban élhessenek az otthonukban.

Azt gondolhatnánk, hogy az otthoni hálózat nem más, mint csak egy másik LAN, valószínűleg más tulajdonságokkal, mint a többi hálózat. Először is a hálózatba kötött eszközöknek nagyon könnyen telepíthetőnek kell lenniük. A vezeték nélküli útválasztók a leggyakrabban visszavitt fogyasztói elektronikus cikkek közé tartoznak. Az emberek azért vesznek ilyet, mert az otthonukban egy vezeték nélküli hálózatot szeretnének. Azt tapasztalják azonban, hogy az eszköz a „doboz kinyitásakor" nem működik, majd visszaviszik ahelyett, hogy monoton zenét hallgatnának, miközben a technikai segélyvonalon várakoznak.

Másrészt a hálózatnak és az eszközöknek egyaránt üzembiztosan kell működniük. A légkondicionálóknak régebben egyetlen forgatókapcsolójuk volt négy állással: KI, GYENGE, KÖZEPES és ERŐS. Most 30 oldalas használati útmutatójuk van. Amint hálózatba lesznek kötve, számítsunk arra, hogy csak a biztonságról szóló fejezet maga 30

oldal lesz. Ez azért probléma, mert kizárólag a számítógép-felhasználók nyugodtak bele a nem működő termékekbe, az autó-, televízió- vagy hűtőgépvásárló közönség kevésbé toleráns. Ők azt várják el, hogy a termékek 100%-osan működjenek anélkül, hogy szerelőt kelljen hívni.

Harmadrészt, az alacsony árak nélkülözhetetlenek a sikerhez. Az emberek nem fognak 50 dollár felárat fizetni egy internetes hőmérséklet-szabályzóért, mert kevesen gondolják, hogy az otthoni hőmérséklet ellenőrzése a munkahelyről ennyire fontos lenne. Plusz 5 dollárért azonban lehet, hogy elkelne.

Negyedrészt, tudni kell egy vagy két eszközzel indulni, majd fokozatosan kiterjeszteni a hálózat által elért eszközök körét. Ez a formátumháborúk kizárásával jár. Ha azt mondjuk a vásárlóknak, hogy IEEE 1394 (FireWire) interfésszel felszerelt perifériákat vegyenek, majd pár évvel később visszalépünk, és kijelentjük, hogy az USB 2.0 a hónap interfésze, majd ezt a 802.11g-re változtatjuk – hoppá, nem, legyen inkább 802.11n – úgy értem, 802.16 (más típusú vezeték nélküli hálózat) – ez nagyon bosszantani fogja a vásárlókat. A hálózati interfésznek évtizedekig változatlannak kell maradnia, mint ahogy a televíziós műsorszórás szabványai is változatlanok.

Ötödrészt, a biztonság és a megbízhatóság is nagyon fontos lesz. Egy dolog elveszíteni néhány állományt egy e-levélben kapott vírus miatt, de hogy egy betörő a hordozható számítógépéről kikapcsolja a teljes biztonsági rendszerünket, majd kifossza az otthonunkat, az egészen más dolog.

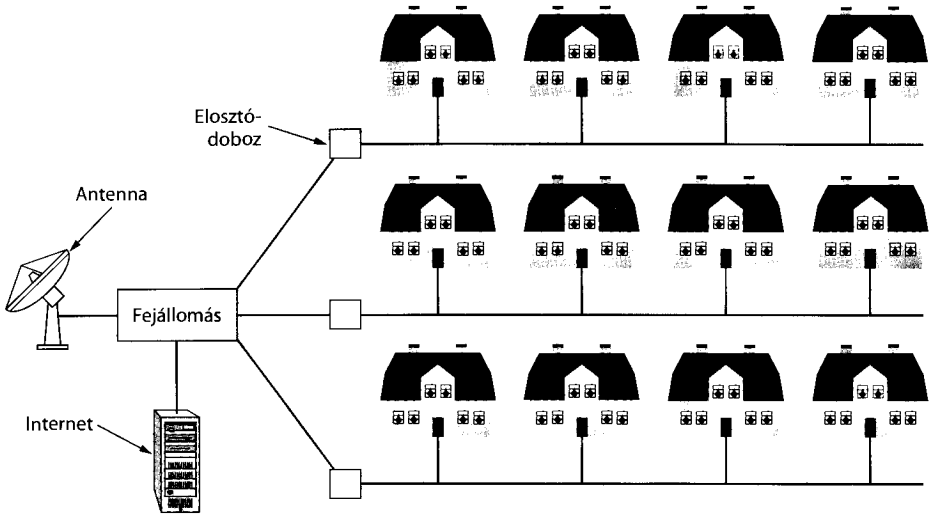
Érdekes kérdés, hogy vajon vezetékesek vagy vezeték nélküliek lesznek-e az otthoni hálózatok. A kényelem és az ár a vezeték nélküli hálózatoknak kedvez, mert nincs szükség kábelezés kialakítására vagy – ami még rosszabb – áttervezésére. A biztonsági szempont a vezetékes hálózatokat részesíti előnyben, mert a vezeték nélküli hálózatok által használt rádióhullámok elég jól keresztülhaladnak a falakon. Nincs mindenki elragadtatva attól az ötlettől, hogy a szomszédja potyázzon az internetkapcsolatán, és az e-levelezését olvassa. A 8. fejezetben azt fogjuk tanulmányozni, hogy hogyan növelhető a biztonság a titkosítás révén, de gyakorlatlan felhasználók esetében ezt könnyebb mondani, mint megtenni.

Egy harmadik – lehet, hogy vonzó – lehetőség az otthonokban már eleve meglevő hálózatok újrafelhasználása. Egy kézenfekvő jelölt az elektromos vezetékhálózat, mely az egész házban telepítve van. Az **erősáramú hálózatok** (**power-line network**) révén a konnektorba dugott eszközök az egész ház számára üzenetszórással továbbíthatnak információt. A tv-t mindenképpen be kell dugni, és ily módon ezzel egy időben internetkapcsolat is létesíthető. A nehézség abból adódik, hogy hogyan továbbítsuk egyszerre az áramot és az adatjeleket. A válasz egyik fele az, hogy ezek más frekvenciasávokat használnak.

Röviden tehát az otthoni LAN-ok sok lehetőséget és kihívást kínálnak. Ez utóbbi nagyrészt a könnyű kezelhetőségre, a megbízhatóságra, az alacsony árra, valamint a biztonságra vonatkozik, különösen műszaki területen nem jártas felhasználók esetében.

1.2.3. Nagyvárosi hálózatok

A **nagyvárosi hálózat** (**Metropolitan Area Network**, MAN) egy város egész területét fedi le. A MAN-ok legközismertebb példája a sok városban elérhető kábeltévé-hálózat. Ez a rendszer azokból a régebben megosztottan használt közösségi antennarendszerek-



1.9. ábra. *Egy kábeltévé-hálózatra alapozott nagyvárosi hálózat*

ből nőtte ki magát, amelyeket az olyan területeken használtak, ahol a földi sugárzású adókat egyébként nagyon gyenge minőségben lehetett venni. Ezekben a korai rendszerekben egy nagy antennát telepítettek egy közeli domb tetejére, és onnan kábelen vitték át a jelet az előfizetők házaiba.

Kezdetben ezek helyben tervezett, *ad hoc* jellegű (alkalomnak megfelelően összeállított) rendszerek voltak. Később a nagyobb cégek is elkezdtek beszállni az üzletbe, és a helyi önkormányzatoktól egész városok behálózására szóló megbízásokat nyertek el. A következő lépést a tv-programok kidolgozása jelentette, kialakultak kifejezetten kábeles terjesztésre szánt csatornák is. Ezek a csatornák sok esetben nagyon szűk témákra szakosodtak, mint például csak hírek, sport, főzés, kertészet és más hasonló témák. Mindazonáltal egészen az 1990-es évek végéig kizárólag tv-adások vételére használtuk a kábeltévé-hálózatokat.

Amikor az internet már nagy tömegeket kezdett vonzani, a kábeltévé-üzemeltetők lassan ráébredtek arra, hogy néhány helyen megváltoztatva a rendszert, kétirányú internet-hozzáférést tudnak szolgáltatni a frekvenciaspektrum addig nem használt részein. Ez volt az a pillanat, amikor a kábeltévérendszer elkezdett a kizárólag tv-adások elosztására alkalmas eszközből átalakulni nagyvárosi hálózattá. Első közelítésben egy MAN az 1.9. ábrán látható rendszerhez hasonlóan néz ki. Az ábrán látható, hogy mind a tv-jeleket, mind az internetet bevezetik a központi **fejállomásba** (**cable headend**), hogy az szétossza a hálózathoz kapcsolódó házak között. Erre a témára még részletesen vissza fogunk térni a 2. fejezetben.

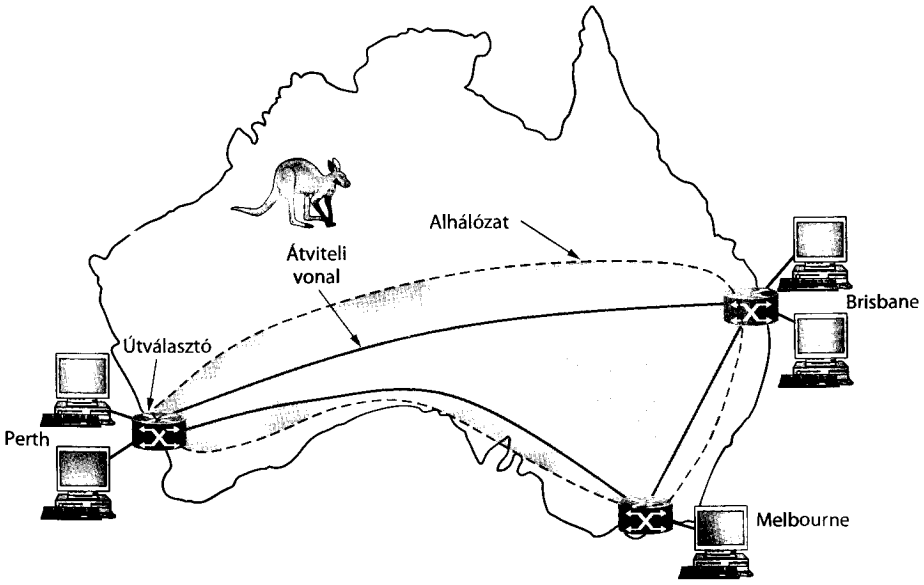
A kábeltévé-hálózat nem az egyetlen létező MAN. A közelmúlt nagy sebességű vezeték nélküli internet-hozzáférést érintő fejlesztéseinek eredménye egy másik MAN, amelyet az IEEE 802.16-os szabvány rögzít, és **WiMAX** néven ismert. A 4. fejezetben fogjuk ezt megvizsgálni.

1.2.4. Nagy kiterjedésű hálózatok

A **nagy kiterjedésű hálózat** (**Wide Area Network**, **WAN**) nagy földrajzi kiterjedésű területeket fed le, sokszor egy egész országot vagy kontinenst. A téma tárgyalását a vezetékes WAN-okkal fogjuk kezdeni, egy olyan vállalat példáján keresztül, melynek több városban van kirendeltsége.

Az 1.10. ábrán szereplő nagy kiterjedésű hálózat az ausztráliai Perth, Melbourne és Brisbane városokban található irodákat köti össze. Mindegyik irodában találhatók számítógépek, amelyeket felhasználói programok (vagyis alkalmazások) futtatására szántak. A hagyományos szóhasználatot követve ezeket a gépeket **gazdagépeknek** vagy **hosztoknak** (**host**) fogjuk nevezni. A hálózat maradék része a hosztokat összekötő **kommunikációs alhálózat** (**communication subnet**) vagy röviden **alhálózat** (**subnet**). Az alhálózat feladata az, hogy üzeneteket vigyen át az egyik hoszttól a másikig ahhoz hasonlóan, ahogyan egy telefonrendszer szavakat (valójában csak hangokat) visz át az egyik előfizetőtől a másikig.

A legtöbb nagy kiterjedésű hálózatban az alhálózat két különböző elemből épül fel: az átviteli vonalakból és a kapcsoló elemekből. Az **átviteli vonalak** (**transmission line**) biteket tudnak mozgatni gépek között. Készülhetnek rézvezetékből, fényvezető szálból vagy lehetnek akár rádiós kapcsolatok is. A **kapcsolóelemek** (**switching element**) vagy csak **kapcsolók** (**switch**), olyan speciális számítógépek, amelyek két vagy több átviteli vonalat kötnek össze egymással. Amikor adatok érkeznek egy bejövő vonalon, a kapcsolóelem kiválaszt egy kimenő vonalat, és azon a vonalon továbbítja az adatokat. Ezeket



1.10. ábra. *Három ausztrál telephelyet összekötő nagy kiterjedésű hálózat*

a kapcsoló számítógépeket sokféleképpen nevezték a múltban, a mai szóhasználatban leginkább elterjedt nevük a **router**[5] (**útválasztó**).

Ide kívánkozik egy, az „alhálózat" kifejezéssel kapcsolatos rövid megjegyzés. Eredetileg ennek a szónak az *egyetlen* jelentése a küldő hoszt és a címzett hoszt között csomagokat továbbító átviteli vonalak és útválasztók összessége volt. Figyelemmel kell lennünk arra, hogy szerzett egy újabb, második jelentést is a hálózati címzéssel kapcsolatban. Azt a jelentést az 5. fejezetben fogjuk tárgyalni, addig azonban a szó eredeti jelentéséhez (átviteli vonalak és útválasztók összessége) tartjuk magunkat.

Az alapján, ahogy eddig jellemeztük, a WAN tulajdonképpen egy nagy vezetékes LAN-hoz hasonlónak látszik, de van néhány fontos különbség, melyek túlmutatnak a hosszú vezetékeken. Általában egy WAN-ban a hosztok és az alhálózat más-más tulajdonában és felügyelete alatt állnak. A mi példánkban az alkalmazottak is felelősek lehetnek a saját számítógépeikért, míg a vállalat informatikai osztálya felel a hálózat többi részéért. Tisztább határokat fogunk látni az elkövetkező példákban, melyekben az internetszolgáltató vagy a telefontársaság üzemelteti az alhálózatot. A tisztán kommunikációs feladatok (az alhálózat) különválasztása az alkalmazásokkal kapcsolatos feladatoktól (a hosztok) nagyban leegyszerűsíti a teljes hálózat tervezését.

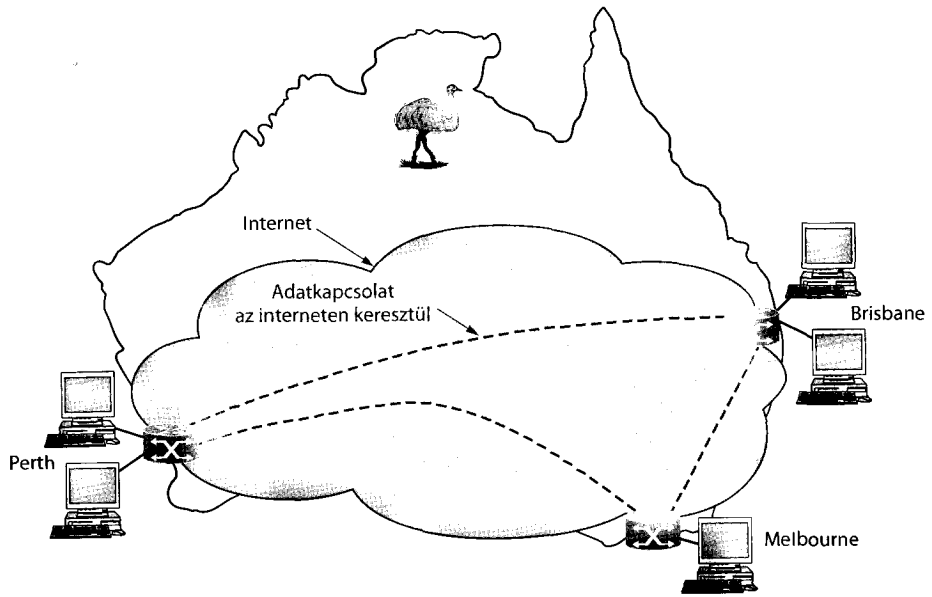
A második különbség, hogy az útválasztók általában különböző hálózati technikákat kapcsolnak össze. Az irodákban található hálózat lehet például kapcsolt Ethernet, míg a nagy távolságú átviteli vonalak lehetnek SONET-kapcsolatok (melyeket a 2. fejezetben fogunk tárgyalni). Valamilyen eszköznek össze kell ezeket kapcsolnia. A figyelmes olvasó észreveheti, hogy ez túlmutat a hálózat definícióján. Ez azt jelenti, hogy sok WAN-t valójában **összekapcsolt hálózatok** (**internetwork**) alkotnak, melyek több hálózatból összeálló összetett hálózatok. Az összekapcsolt hálózatokról a következő szakaszban olvashatunk majd többet.

Az utolsó különbség, hogy miket csatlakoztatunk az alhálózathoz. Ezek lehetnek önálló számítógépek, mint a LAN-ok esetében, vagy lehetnek egész LAN-ok. Ez az a mód, ahogy a nagyobb hálózatok kisebbekből felépülnek. Ami az alhálózatot illeti, ugyanezt a feladatot teljesíti.

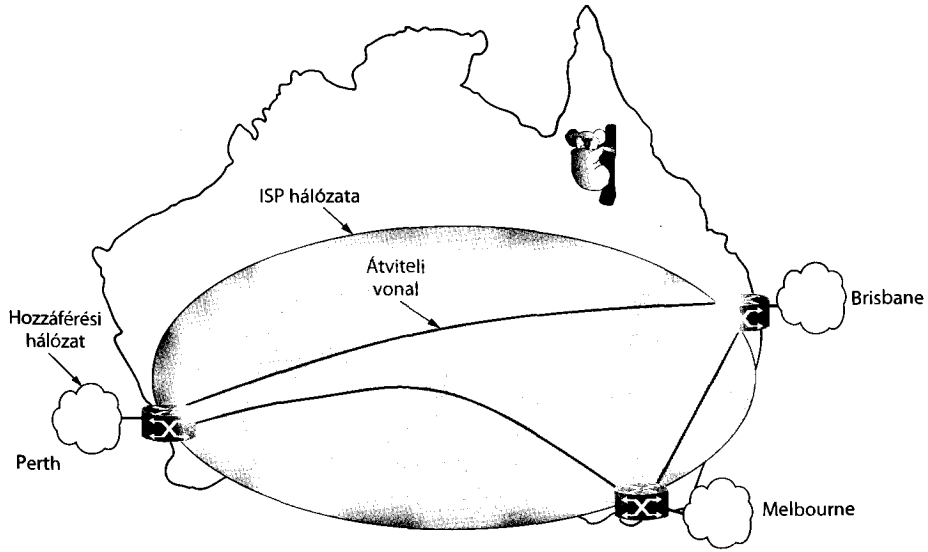
Most megtehetjük, hogy megvizsgáljuk a WAN-ok két további változatát. Először is egy vállalat ahelyett, hogy dedikált átviteli vonalakat bérelne, az internetre kötheti az irodáit. Ez lehetővé teszi, hogy az irodák közötti összeköttetések virtuális kapcsolatokként működjenek, melyek az alatta lévő internet kapacitását használják. Ezt az 1.11. ábrán mutatott elrendezést **virtuális magánhálózatnak** (**Virtual Private Network, VPN**) nevezzük. A dedikált kialakítással összehasonlítva a VPN-nél megtaláljuk a virtualizáció szokásos előnyeit, vagyis azt, hogy egy erőforrás (az internetre kapcsolódás) rugalmas újrafelhasználását biztosítja. Hogy ezt belássuk, gondoljuk csak végig, hogy mennyire egyszerű egy negyedik irodát hozzáadni a hálózathoz. A VPN azonban magában hordja a virtualizáció szokásos hátrányát is, mely az alatta lévő erőforrások feletti irányítás hiánya. Egy dedikált vonallal a kapacitás tisztán látható. Egy VPN-t használva viszont a kapott kapacitás az internetszolgáltatással ingadozhat.

A másik változat az, hogy az alhálózatot egy másik vállalat üzemelteti. Az alhálózat üzemeltetőjét **hálózati szolgáltatónak** (**network service provider**) nevezzük, és az irodák az ügyfelei. Ezt a struktúrát láthatjuk az 1.12. ábrán. Az alhálózat üzemeltetője más ügyfelekkel is kiépít hálózati kapcsolatot, feltéve, hogy tudnak fizetni, és az üzemeltető képes szolgáltatást nyújtani a számukra. Mivel eléggé csalódást keltő lenne a hálózati szolgáltatás, ha az ügyfelek csak egymásnak tudnának csomagokat küldeni, az alhálózat

5 A továbbiakban a router szó helyett az útválasztó szót fogjuk használni. *(A lektor megjegyzése)*



1.11. ábra. *Virtuális magánhálózatot használó nagy kiterjedésű hálózat*



1.12. ábra. *Internet-szolgáltatói hálózatot használó nagy kiterjedésű hálózat*

üzemeltetője más hálózatokhoz is kapcsolódni fog, amelyek az internet részei. Az ilyen alhálózat üzemeltetőjét **internetszolgáltatónak (Internet Service Provider, ISP)** nevezzük, az alhálózat pedig az **ISP-hálózat (ISP network)**. Az ügyfelek, akik az ISP-hez kapcsolódnak, internetszolgáltatást kapnak.

Felhasználhatjuk az internet-szolgáltatói hálózatot arra, hogy bemutassunk néhány kulcskérdést, melyeket a következő fejezetekben fogunk tanulmányozni. A legtöbb WAN-ban a hálózat számos átviteli vonalból áll, és mindegyik egy-egy útválasztópárt köt össze. Ha két olyan útválasztó akar egymással kommunikálni, amelyek között nincs közvetlen átviteli vonal, akkor azt csak közvetetten tehetik meg, más útválasztókon keresztül. Létezhet több olyan út is a hálózatban, amelyik összeköti ezt a két útválasztót. Azt a módot, ahogyan a hálózat eldönti, hogy melyik útvonalat használja, **útválasztó** (vagy **forgalomirányító**) **algoritmusnak (routing algorithm)** nevezzük. Sok változata létezik. Azt, ahogyan az egyes útválasztók eldöntik, hogy egy csomagot hova továbbítsanak következő lépésként, **továbbító algoritmusnak (forwarding algorithm)** nevezzük. Ezeknek is sok változata létezik. Mindkét típusból néhány algoritmust részletesen is meg fogunk vizsgálni az 5. fejezetben.

Más típusú WAN-ok nagyban hagyatkoznak a vezeték nélküli technikákra. A műholdas rendszerekben minden földi számítógép rendelkezik egy antennával, amelynek segítségével adni és adásokat fogadni tud a Föld körüli pályán keringő műholdon keresztül. Minden útválasztó hallja a *műholdról* érkező jeleket, és egyes esetekben a többi útválasztó felfelé, a *műholdra* küldött adásait is. A műholdas hálózatok természetüknél fogva adatszórásos rendszerek, és ezért az olyan esetekben a leghasznosabbak, ahol az adatszórás képessége fontos.

A mobiltelefon-hálózat egy másik példa a vezeték nélküli technikát alkalmazó WAN-ra. Ez a rendszer máris megélt három generációt, és küszöbön áll a negyedik. Az első generáció analóg volt, és kizárólag hangátvitelre volt alkalmas. A második generáció digitális volt, de szintén kizárólag hangátvitelt támogatott. A harmadik generáció is digitális, és egyaránt használható hang és adat továbbítására. A mobiltelefon-bázisállomások sokkal nagyobb területet fednek le, mint a vezeték nélküli LAN-ok, a hatósugaruk kilométerekben mérhető, nem pedig tíz méterekben. A bázisállomásokat egy gerinchálózat kapcsolja össze, mely általában vezetékes. A mobiltelefon-hálózatok adatátviteli sebessége gyakran az 1 Mb/s nagyságrendben mozog, ami sokkal alacsonyabb, mint a vezeték nélküli LAN-ok által elérhető 100 Mb/s nagyságrend. A 2. fejezetben rengeteg mondanivalónk lesz ezekről a hálózatokról.

1.2.5. Összekapcsolt hálózatok

A világon számos hálózat létezik, és ezek hardvere és szoftvere sok esetben eltér egymástól. Azok a felhasználók, akik egy adott hálózathoz kapcsolódnak, gyakran szeretnének más hálózatokhoz kapcsolódó felhasználókkal is kommunikálni. Ez az igény váltotta ki a különböző, egymással sokszor nem kompatibilis hálózatok összekapcsolását. Az ily módon összekapcsolt hálózatokat együttesen **összekapcsolt hálózatnak (internetwork** vagy röviden **internet)** hívjuk. Ezeket a kifejezéseket általános értelemben fogjuk használni, szemben a világméretű internettel, a világhálóval (ami egy kitüntetett internet).

Az utóbbi az ISP-k hálózatát használja arra, hogy vállalati, otthoni és sok más típusú hálózatokat összekapcsoljon. A világhálót részletesen is bemutatjuk a könyv későbbi fejezeteiben.

Az alhálózatot, a hálózatot és az összekapcsolt hálózatot gyakran összekeverik. Az „alhálózat" kifejezés a nagy kiterjedésű hálózatok esetén értelmezhető a leginkább, ahol a hálózat üzemeltetőjének tulajdonát képező útválasztók és az átviteli vonalak együttesét jelenti. Ehhez hasonlóan a telefonhálózatok is nagy sebességű vonalak által összekötött telefonközpontokból, valamint kis sebességű vonalakkal csatlakoztatott otthonokból és cégekből állnak. Ezek a vonalak és központok a telefontársaság tulajdonát képezik, és az felügyeli a működésüket. A telefonhálózatban az átviteli vonalak és a telefonközpontok alkotják az alhálózatot. A telefonkészülékek, akárcsak a hosztok, nem képezik részét az alhálózatnak.

A hálózatot az alhálózat és a hosztok együttesen alkotják. A „hálózat" szót is gyakran használják azonban szabadabb értelmezésben. Egy alhálózatot is jellemezhetünk hálózatként, mint ahogy az 1.12. ábra internet-szolgáltatói hálózata esetében tettük. Egy internetet is nevezhetünk egyszerűen hálózatnak, mint az 1.10. ábrán bemutatott WAN-t. Hasonlóan fogunk eljárni a könyvben, és amikor meg kívánjuk különböztetni a hálózatot más összeállítású rendszerektől, ragaszkodni fogunk az eredeti definíciónkhoz, miszerint a hálózat egyetlen hálózati technikával összekapcsolt számítógépek együttese.

Essen több szó arról, hogy miből áll egy összekapcsolt hálózat. Tudjuk, hogy egy internet különálló hálózatok összekapcsolásával keletkezik. A mi nézőpontunkból egy LAN és egy WAN, vagy két LAN összekötése internetet alkot, de ezen a területen kevés egyetértés tapasztalható a szóhasználatban. Két hasznos ökölszabály van. Az egyik az, hogy ha különálló szervezetek fizettek a hálózat különböző részeinek megépítéséért, és külön-külön tartják karban a saját részüket, akkor internetről van szó, nem pedig egyetlen hálózatról. Abban az esetben is valószínűleg két külön hálózattal van dolgunk, ha a két rész különböző átviteli technikán alapul (például adatszórás vagy kétpontos összeköttetés, illetve vezetékes vagy vezeték nélküli kapcsolat).

Mélyebbre tekintve beszélnünk kell arról, hogy hogyan lehetséges két különböző hálózatot összekapcsolni. Annak a gépnek, amely két vagy több hálózatot kapcsol össze, és biztosítja az átjárhatóságot mind hardver, mind szoftver szempontjából, az általános elnevezése **átjáró (gateway)**. Az átjárókat az alapján a réteg alapján különböztetjük meg, amelyikben tevékenységüket végzik a protokollhierarchiában. A következő szakasszal kezdődően sokkal többet fogunk elmondani a hálózati rétegekről és protokollhierarchiákról, de egyelőre úgy képzeljük el ezeket, hogy a felsőbb rétegek az alkalmazásokhoz kötődnek szorosabban, például a webhez, míg az alsóbb rétegek az átviteli összeköttetésekhez, például az Ethernethez.

Mivel egy internet kialakításának előnye éppen az, hogy számítógépeket köt össze hálózatokon átívelő módon, nem szeretnénk túlságosan alacsony szintű átjárókat használni, mert különben képtelenné válnánk eltérő típusú hálózatok között kapcsolatot létesíteni. A túl magas szintű átjárók sem kívánatosak, mert a kapcsolat csak bizonyos alkalmazások esetében lenne működőképes. A középen elhelyezkedő „pont jó" szintet gyakran hálózati rétegnek nevezzük, és az útválasztók azok az átjárók, amelyek a hálózati rétegben továbbítják a csomagokat. Tehát mostantól úgy is felismerhetünk egy internetet, hogy útválasztókkal épült hálózatot keresünk.

1.3. Hálózati szoftver

Az első számítógép-hálózatoknál a legfőbb tervezési szempont a hardver volt, és csak azután jött a szoftver. Ez a módszer ma már nem működik. A hálózati szoftverek nagymértékben strukturálódtak. A következő alfejezetekben a szoftverek strukturálási módját fogjuk részletesen megvizsgálni. Az itt leírt módszer kulcsfontosságú a könyv további részei szempontjából, és többször is vissza fogunk még térni rá.

1.3.1. Protokollhierarchiák

A tervezés bonyolultságának csökkentése érdekében, a legtöbb hálózatot úgy alakítják ki, hogy azok egymásra épülő **rétegeket** vagy **szinteket** (**layer, level**) képezzenek. A rétegek száma, elnevezése, tartalma és feladata más és más a különböző hálózatokban. Minden réteg célja az, hogy bizonyos szolgáltatásokat (service) nyújtson a felette elhelyezkedő rétegeknek, miközben elrejti előlük a szolgáltatások tényleges megvalósításának részleteit. Egy lehetséges értelmezés szerint minden réteg egy olyan virtuális gép, amely a felette levő rétegek számára szolgáltatásokat nyújt.

Ez tulajdonképpen ismerős koncepció, amely a számítástechnika minden területén használatos. Hívják információelrejtésnek (information hiding), absztrakt adattípusok használatának, adatbeágyazódásnak (data encapsulation) és objektumorientált programozásnak (object-oriented programming, OOP) is. Az alapötlet az, hogy egy meghatározott szoftver- (vagy hardver-) elem szolgáltatást nyújt a felhasználóinak, de a belső állapotára és az algoritmusaira vonatkozó információt elrejtve tartja előlük.

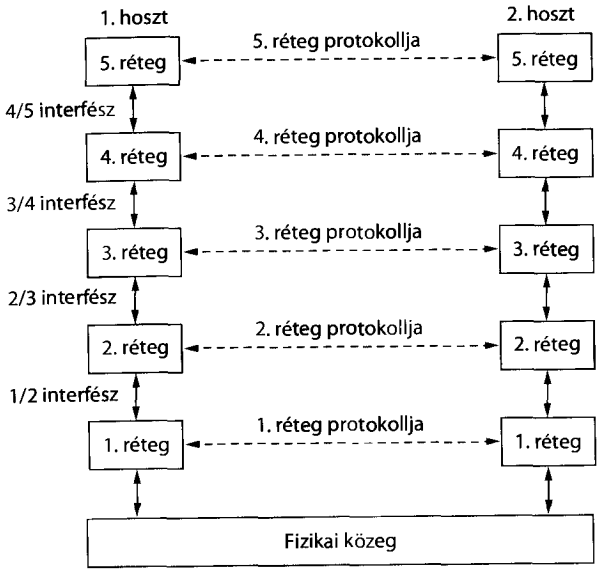
Az egyik gép n-edik rétege párbeszédet folytat egy másik gép n-edik rétegével. A párbeszéd írott és íratlan szabályait együttesen az n-edik réteg **protokolljának** (**protocol**) nevezzük. A protokoll lényegében olyan megállapodás, amely az egymással kommunikáló felek közötti párbeszéd szabályait rögzíti. Egy analóg példával élve, amikor egy nőt bemutatnak egy férfinak, akkor a nőn múlik, hogy kinyújtja-e a kezét, a férfi pedig eldöntheti, hogy kezet fog vele vagy pedig kezet csókol neki. Hogy mi történik, az attól függ, hogy a hölgy egy üzleti tárgyaláson részt vevő amerikai ügyvédnő vagy egy bálon megjelenő európai hercegnő. A protokoll megsértése nagyban megnehezítené, sőt akár lehetetlenné is tenné a kommunikációt.

Az 1.13. ábrán egy ötrétegű hálózatot láthatunk. Azokat az entitásokat, amelyeket a különböző gépek azonos rétegei tartalmaznak, **társentitásoknak** (**peer**) nevezzük. Más szóval a társentitások azok az entitások, amelyek a protokoll segítségével kommunikálnak egymással.

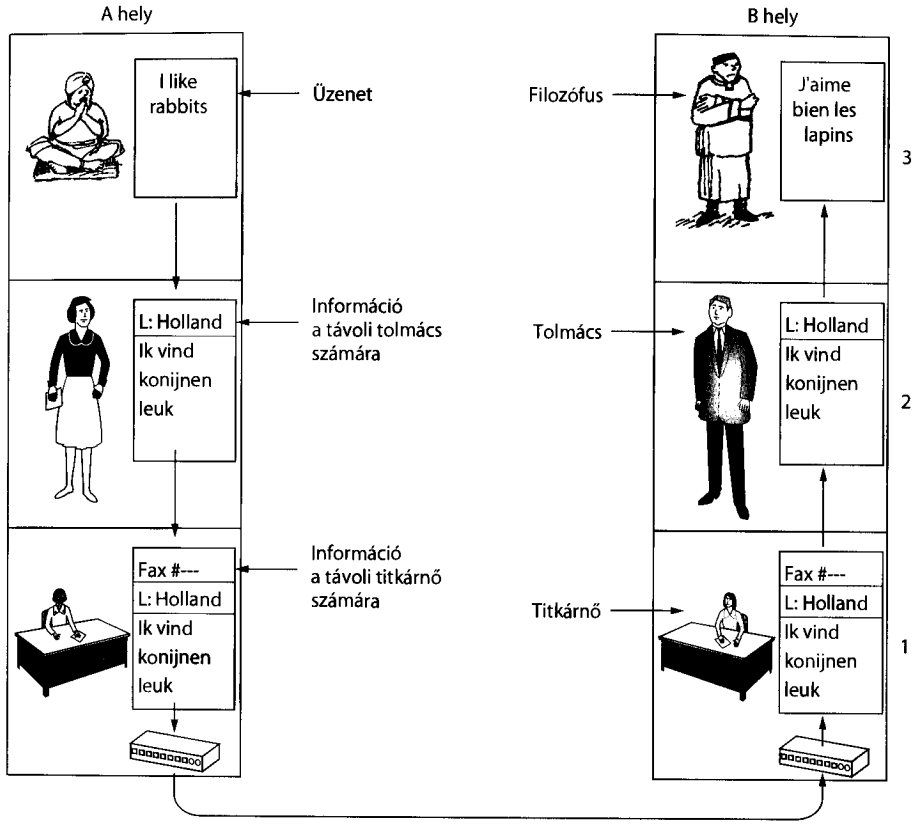
A valóságban az egyik gép n-edik rétegéből az adatok nem közvetlenül jutnak át egy másik gép n-edik rétegébe, hanem valamilyen vezérlőinformációval kiegészítve mindegyik réteg közvetlenül az alatta levőnek továbbítja az adatokat egészen addig, amíg azok a legalsó rétegig el nem jutnak. Az első réteg alatt a **fizikai közeg** (**physical medium**) található, amelyen a valódi kommunikáció zajlik. Az 1.13. ábrán a virtuális kommunikációt szaggatott vonalakkal, a fizikai kommunikációt pedig folytonos vonalakkal jelöltük.

Az egymással szomszédos rétegek között **interfész** (**interface**) található. Az interfész azt definiálja, hogy az alacsonyabban levő réteg milyen elemi műveleteket és szolgáltatásokat nyújt a magasabban levő réteg számára. Amikor a hálózattervezők eldöntik, hogy hány réteget tartalmazzon egy hálózat, és hogy mi legyen az egyes rétegek feladata, akkor a legfontosabb szempont az, hogy a rétegek közötti interfész minél világosabb legyen. Ehhez persze az szükséges, hogy minden réteg jól definiált feladatokkal rendelkezzen. Azonkívül, hogy a rétegek közötti információcserét minimalizálják, a jól meghatározott interfészek azt is könnyebbé teszik, hogy valamelyik réteg tényleges megvalósítását lecseréljük egy teljesen újra (például lecseréljük az összes telefonvonalat műholdas kapcsolatra), mivel mindössze annyit várunk el az új megvalósítástól, hogy pontosan ugyanazokat a szolgáltatásokat nyújtsa a felette levő rétegnek, mint a korábbi megvalósítás. Gyakran előfordul, hogy a különböző hosztok egyazon protokoll eltérő (gyakran más cégek által készített) megvalósításait használják. Valójában a protokoll maga is megváltozhat valamelyik rétegben anélkül, hogy a fölötte vagy alatta található rétegek egyáltalán észrevennék.

A rétegek és protokollok halmazát **hálózati architektúrának** (**network architecture**) nevezzük. Az architektúra specifikációjának elegendő információt kell tartalmaznia ahhoz, hogy az implementálást végző szakember minden réteghez meg tudja írni a programot, illetve meg tudja építeni a hardvert úgy, hogy az helyesen alkalmazza a megfelelő protokollt. Az implementáció részletei és az interfészek specifikációja nem része az architektúrának, mivel ezek a gép belsejében rejtve maradnak, tehát kívülről nem láthatók. Az sem szükséges, hogy a hálózat összes gépén ugyanazok az interfészek legyenek, feltéve, hogy az összes gép helyesen használja a protokollokat. Ha egy adott rendszerben minden réteg egyetlen protokollal rendelkezik, akkor a rendszer protokolljainak összességét **protokollkészletnek** (**protocol stack**) nevezzük. A hálózati architektúrák, a protokollkészletek és maguk a protokollok alkotják a könyv leglényegesebb témáit.



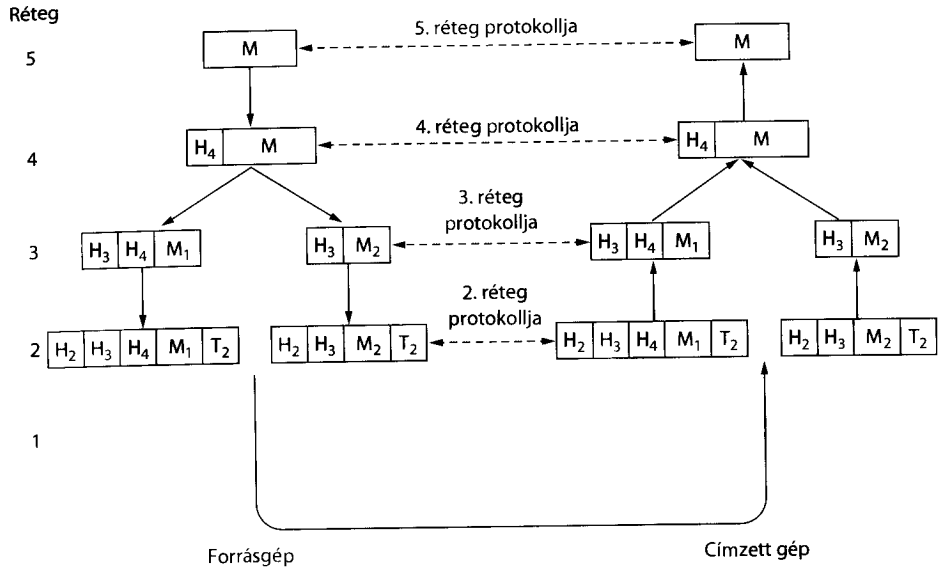
1.13. ábra. *Rétegek, protokollok és interfészek*



1.14. ábra. *A filozófusból, tolmácsból és titkárnőből álló architektúra*

A következő analógia talán segít a többréteges kommunikáció elvének megértésében. Képzeljünk el két filozófust (ők a 3. réteg társfolyamatai), akik közül az egyik urdu nyelven és angolul beszél, a másik pedig kínaiul és franciául. Mivel nincs közös nyelvük, ezért mindkettőjük egy-egy tolmácsot alkalmaz (ők a 2. réteg társfolyamatai). Mindkét tolmácsnak van egy titkárnője (ők az 1. réteg társfolyamatai). Az 1-es filozófus az *oryctolagus cuniculus* iránti ragaszkodását szeretné kifejezni a társának. Ahhoz, hogy ezt megtehesse, elküld egy üzenetet (angolul) a 2/3 interfészen keresztül a tolmácsának, ahogy ez az 1.14. ábrán is látható. Az üzenet tartalma a következő: „I like rabbits" („Szeretem a nyulakat"). A tolmácsok megegyeztek egy semleges nyelvben, a hollandban, így tehát az üzenet a következőképpen hangzik: „Ik wind konijnen leuk". A nyelv megválasztása a 2. réteg protokolljának a feladata, és kizárólag ennek a rétegnek a társfolyamataitól függ.

A tolmács ezek után átadja az üzenetet a titkárnőnek, hogy továbbítsa azt, mondjuk faxon (1. réteg protokollja). Amikor a másik félhez megérkezik az üzenet, akkor azt lefordítják franciára, majd pedig tovább kerül a 2/3 interfészen keresztül a 2-es filozófushoz. Vegyük észre, hogy az egyes protokollok teljesen függetlenek egymástól, amennyiben az interfészek nem változnak. A tolmács hollandról átválthat mondjuk finnre, ha



1.15. ábra. *Példa az 5. réteg virtuális kommunikációját megvalósító információáramlásra*

akar, feltéve, hogy megegyeztek benne, és egyikük sem változtatja meg az interfészt az 1. vagy a 3. réteg felé. Hasonlóképpen, a titkárnők fax helyett e-levelet vagy telefont is használhatnak anélkül, hogy megzavarnák (sőt anélkül, hogy egyáltalán tájékoztatnák) a többi réteget. További információt mindegyik folyamat csak a saját társfolyamatának küldhet. Ez az információ már nem jut el az eggyel magasabb réteghez.

Nézzünk meg most egy sokkal inkább műszaki jellegű példát. A kérdés az, hogy hogyan tegyük lehetővé a kommunikációt az 1.15. ábrán látható ötrétegű hálózat legfelső rétege számára. Az 5. rétegben egy alkalmazói folyamat létrehoz egy üzenetet (jelöljük ezt M-mel), majd átadja a 4. rétegnek, hogy továbbítsa azt. A 4. réteg az üzenet azonosítása céljából egy fejrészt (**header**) illeszt az üzenet elejére, és továbbadja a 3. rétegnek. A fejrész vezérlőinformációt tartalmaz, például címzést, hogy a fogadógépre a 4. rétegben megérkezhessen az üzenet. Az egyes rétegekben használt vezérlőinformáció más példái a sorszámozás (ha az alsóbb réteg nem őrzi meg az üzenetsorrendet), a méret és az idő megadása.

A 4. réteg által elküldött üzenetek mérete sok hálózatban tetszőlegesen nagy lehet, ugyanakkor a 3. réteg protokollja szinte minden hálózatban meghatároz egy maximális üzenethosszt. Következésképpen a 3. rétegnek kisebb egységekre, csomagokra kell bontania a felülről hozzá érkező üzeneteket, és minden csomagot ki kell egészítenie egy fejrésszel. Példánkban az M üzenetet két részre osztottuk: M_1-re és M_2-re, melyek egymástól függetlenül továbbítódnak.

A 3. réteg kiválasztja a megfelelő kimeneti vonalat, majd továbbadja a csomagot a 2. rétegnek. A 2. réteg nem csak fejrészt, hanem egy farokrészt (trailer) is hozzácsatol a csomaghoz, és az így kapott egységet adja át az 1. rétegnek a fizikai továbbítás céljából. A vevő oldalon az üzenet rétegről rétegre felfelé halad, miközben a fejrészek leválnak róla. Az n-edik réteg alatti rétegek fejrészei sosem juthatnak el az n-edik rétegig.

Az 1.15. ábrán látható rajz kapcsán a legfontosabb az, hogy megértsük a különbséget a virtuális és a valódi kommunikáció, illetve a protokoll és az interfész között. Például a 4. réteg társfolyamatai a saját kommunikációjukat következetesen „horizontálisnak" tekintik, hiszen a 4. réteg protokollját használják. Valószínűleg mindkét folyamatnak van egy *KüldésATúloldalra* és egy *VételATúloldalról* nevű eljárása még akkor is, ha valójában nem közvetlenül egymással, hanem a 3/4 interfészen keresztül az alsóbb rétegekkel kommunikálnak.

A társfolyamat-absztrakció alapvetően fontos a hálózat tervezéséhez. Ez teszi ugyanis lehetővé, hogy a teljes hálózat megtervezésének átláthatatlanul bonyolult feladatát több kisebb, jól kezelhető tervezési feladatra osszuk, azaz, hogy az egyes rétegeket külön-külön tervezhessük meg.

Bár az 1.3. alfejezet címe „Hálózati szoftver", érdemes megjegyezni, hogy a protokollhierarchia alacsonyabb szintjeit gyakran hardvereszközökkel vagy firmware-rel valósítják meg. Még bonyolult algoritmusokat is implementálnak hardverben.

1.3.2. A rétegek tervezési kérdései

A számítógép-hálózatok tervezésének legfontosabb kérdései rétegről rétegre haladva újra felmerülnek. Az alábbiakban a legfontosabbak közül emelünk ki néhányat.

A megbízhatóság kérdéskörébe az tartozik, hogy a hálózat annak ellenére is helyesen működjön, hogy önmagukban megbízhatatlan alkotóelemekből épül fel. Gondoljunk csak egy hálózaton keresztül utazó csomag bitjeire. Valamekkora eséllyel ezek a bitek sérülten (invertálva) érkeznek meg, melynek oka lehet véletlenszerű elektromos zaj, kiszámíthatatlan vezeték nélküli jelek, illetve hardver- vagy szoftverhibák és így tovább. Hogyan lehetséges mégis, hogy észrevegyük és kijavítsuk ezeket a hibákat?

A fogadott adatok hibáinak észrevételére, azaz a **hibajelzésre** (**error detection**) az egyik lehetséges módszer a kódolás használata. A hibásan vett adatokat újra lehet küldeni, mígnem egyszer csak helyesen érkeznek meg. Hathatósabb kódolás révén **hibajavítás** (**error correction**) is lehetséges, azaz a helyes üzenet helyreállítható a ténylegesen fogadott, esetlegesen hibás bitekből. Mindkét eljárás redundáns információ hozzáadásával működik. Az alsóbb rétegekben azért használják ezeket, hogy az egyes összeköttetéseken átküldött csomagokat védjék, míg a felsőbb rétegekben azt ellenőrzik, hogy a helyes adatok érkeztek-e válaszként.

Egy másik megbízhatósági kérdés a működő útvonalak megtalálása a hálózatban. Gyakran több lehetséges útvonal is létezik egy forrásállomás és egy célállomás között, és egy nagy hálózatban lehetnek meghibásodott kapcsolatok vagy útválasztók. Tegyük fel, hogy Németországban épp nem működik a hálózat. Ha Londonból Rómába Németországon keresztül küldenénk csomagokat, azok nem érnének célba, de ehelyett elküldhetnénk azokat Londonból Rómába Párizson keresztül is. A hálózatnak automatikusan kell meghoznia ezeket a döntéseket. Ezt a témakört **útválasztásnak** (**routing**) nevezzük.

A második tervezési problémakör a hálózat fejlődésével foglalkozik. Idővel a hálózatok nagyobbra nőttek, és újfajta elemeket kell a meglévő hálózathoz csatlakoztatni. Az előzőekben láttuk azt a kulcsfontosságú strukturálási módot, mely a teljes probléma felosztásával és a megvalósítás részleteinek elrejtésével segíti a változást: ez a **protokollok rétegezése** (**protocol layering**). De léteznek más stratégiák is.

Mivel a hálózatokban általában sok számítógép van összekötve, minden rétegben kell lennie egy olyan mechanizmusnak, amely egy üzenet küldőjét és vevőjét azonosítja. Ezt a mechanizmust **címzésnek** (**addressing**) nevezik az alsóbb, illetőleg **névkezelésnek** (**naming**) a felsőbb rétegek esetében.

A növekedés egyik velejárója az, hogy a különböző hálózati technikák gyakran másmás korlátokkal rendelkeznek. Például nem minden kommunikációs csatorna tartja meg a rajtuk elküldött üzenetek eredeti sorrendjét, ezért olyan megoldásokat kellett bevezetni, melyek sorszámozzák az üzeneteket. Egy másik példa a hálózatok által továbbítható üzenetek méretkorlátai közötti eltérés. Ez olyan eljárásokat igényel, melyek szétbontják, továbbítják, majd újra összerakják az üzeneteket. Ennek a teljes témának az összefoglaló neve **internetworking**, azaz a **hálózatok összekapcsolásával** foglalkozó terület.

Amikor a hálózatok nagyon nagyra nőnek, új problémák kerülnek elő. A városokban kialakulhatnak forgalmi dugók, elfogyhatnak a telefonszámok, és eltévedni is könnyű. Nem sok embernek okoz ezek problémát a saját körzetében, de városi léptékben gondolkodva már nagy gondot okozhat. Azokat a kialakításokat, melyek akkor is jól működnek, amikor egy hálózat nagyra növekszik, **skálázhatónak** (**scalable**) nevezzük.

A harmadik tervezési kérdés az erőforrások kiosztása. A hálózatok az alattuk levő erőforrások felhasználása révén szolgálják a hosztokat. Ilyen erőforrás például az átviteli vonalak kapacitása. Hogy szolgáltatásukat jól végezhessék, olyan mechanizmusokra van szükségük, melyekkel úgy osztják fel az erőforrásaikat, hogy az egyik hoszt ne nagyon zavarja a másikat.

Sok konstrukcióban a sávszélesség kiosztása dinamikusan, a hosztok rövid távú szükségletei szerint történik ahelyett, hogy minden hoszt a teljes sávszélességnek egy rögzített hányadát kapná, melyet vagy kihasznál, vagy nem. Ezt a működési módot **statisztikai multiplexelésnek** (**statistical multiplexing**) hívjuk, ami azt jelenti, hogy az elosztás az igényekre vonatkozó statisztikák alapján történik. Ez a megoldás az alsóbb rétegekben alkalmazható egyetlen kapcsolatra, vagy a felsőbb rétegekben a teljes hálózatra, vagy akár a hálózatot használó alkalmazásokra is.

Minden rétegben felmerülő erőforrás-kiosztási probléma az, hogy hogyan akadályozzuk meg azt, hogy a gyorsabban adó gépek elárasszák adatokkal a lassabban vevőket. Ezek a módszerek gyakran a vevő és az adó közötti visszacsatoláson alapulnak. Ezt a témakört **forgalomszabályozásnak** (**flow control**) nevezzük. Néha viszont abból adódik a probléma, hogy túl sok számítógép kíván túl sok forgalmat átküldeni, és a hálózat nem képes mindent kézbesíteni. A hálózat ilyen túlterhelődését **torlódásnak** (**congestion**) hívjuk. Az egyik követhető stratégia az, hogy minden számítógép csökkentse az átviteli igényét, amikor torlódást tapasztal. Ez a módszer is használható minden rétegben.

Érdekes észrevétel, hogy egy hálózat a sávszélesség mellett más erőforrásokat is kínálhat. Olyan használati esetekben, mint az élő mozgókép továbbítása, nagyban számít az időben történő kézbesítés. A legtöbb hálózatnak egyszerre kell szolgáltatást nyújtania olyan alkalmazások számára, melyek **valós idejű** (**real-time**) **továbbítást** igényelnek, ugyanakkor nagy átbocsátóképességre van szükségük. Azoknak a mechanizmusoknak, melyek összeegyeztetik ezeket az egymással versengő igényeket, **szolgáltatásminőségi mechanizmus** (**quality of service**) a neve.

Az utolsó nagy tervezési kérdés a hálózat biztonságossá tétele azáltal, hogy megvédjük a különböző fenyegetések ellen. Az egyik fenyegetés, melyet már említettünk, a kommunikáció lehallgatása. A **titkosságot** (**confidentiality**) nyújtó technikák védenek ez ellen a veszély ellen, és több rétegben is használjuk ezeket. A **hitelesítési** (**authentication**) eljárások akadályozzák meg, hogy valaki egy másik szereplőt személyesítsen meg. Használhatók például arra, hogy megkülönböztessük a hamis banki weboldalakat a valódiaktól, vagy hogy a mobiltelefon-hálózat szolgáltatója megbizonyosodjon róla, hogy egy hívás valóban az Ön készülékéről jön, így Ön ki fogja fizetni a számlát. Más mechanizmusok a **sértetlenséget** (**integrity**) védik az üzenetek titokban történő módosítása ellen, mint például, ha a „Vonjon le a számlámról 10 dollárt!" üzenetet megváltoztatná valaki a „Vonjon le a számlámról 1000 dollárt!" üzenetre. Az összes felsorolt elv a kriptográfián alapul, melyet a 8. fejezetben fogunk tanulmányozni.

1.3.3. Összeköttetés-alapú és összeköttetés nélküli szolgáltatások

A rétegek két különböző szolgáltatást[6] nyújthatnak a felettük levő rétegek számára: összeköttetés-alapú és összeköttetés nélküli szolgáltatást. Ebben az alfejezetben ezt a két szolgáltatástípust vizsgáljuk meg, és ismertetjük a kettő közötti különbségeket is.

Az **összeköttetés-alapú szolgáltatás** (**connection-oriented service**) a távbeszélőrendszerrel modellezhető. Ahhoz, hogy valakivel beszélni tudjunk, fel kell emelnünk a telefonkagylót, tárcsázni kell a számot, ezután beszélgethetünk, majd végül le kell tennünk a telefont. Hasonló módon, egy összeköttetés-alapú hálózati szolgáltatás igénybevételéhez a szolgáltatást igénybe vevő felhasználó először létrehozza az összeköttetést, majd felhasználja, végül pedig lebontja azt. Az összeköttetés lényege az, hogy úgy működik, mint egy cső: az adó a cső egyik végén belerakja a dolgokat (biteket), a vevő pedig a másik végén ugyanabban a sorrendben kiveszi azokat. A legtöbb esetben a bitek sorrendje megmarad, vagyis ugyanabban a sorrendben érkeznek meg, mint ahogyan elküldték őket.

Egyes esetekben az összeköttetés létesítését követően a küldő fél, a fogadó fél és az alhálózat **egyezkedésbe** (**negotiation**) kezdenek az olyan használandó paraméterekkel kapcsolatban, mint például az üzenetek maximális hossza, a kívánt szolgáltatásminőség és más hasonló kérdések. Általában valamelyik fél javasol valamit, amit a másik fél elfogadhat vagy elutasíthat, illetve ellenjavaslatot is tehet. Az **áramkör** (**circuit**) egy másik elnevezése az olyan összeköttetésnek, mely hozzárendelt erőforrásokkal rendelkezik, például rögzített sávszélességgel. Ez a telefonhálózatokból eredeztethető, ahol az áramkör azt a rézvezetékből álló útvonalat jelentette, amin egy beszélgetés zajlott.

Ezzel szemben az **összeköttetés nélküli szolgáltatás** (**connectionless service**) a levéltovábbító postai rendszerrel modellezhető. Minden egyes üzenet (levél) rendelkezik egy teljes célcímmel, és minden üzenet az összes többitől független útvonalon továbbítódik a rendszer köztes csomópontjain keresztül. Az üzeneteket eltérő kontextusban

6 Az angol *service* szónak a magyar nyelvben számítógépes és kommunikációs környezetben mind a *szolgálat*, mind a *szolgáltatás* megfelelője használatos. Ebben a könyvben a szolgáltatás szót használjuk. *(A lektor megjegyzése)*

különféleképpen nevezhetjük: a **csomag** (**packet**) a hálózati réteg szintjén levő üzenet. Azt a működési módot, amikor a köztes csomópont teljes hosszában fogad egy üzenetet, mielőtt továbbküldené a következő csomópontnak, **tárol-és-továbbít típusú kapcsolásnak** (**store-and-forward switching**) nevezzük. Ennek az alternatíváját, mely során az üzenet továbbküldése már azelőtt megkezdődik, mielőtt a csomópont teljességében megkapná, **átfuttató kapcsolásnak** (**cut-through switching**) hívjuk. Ha két üzenetet küldünk ugyanarra a címre, akkor általában az ér oda előbb, amelyiket előbb küldtük el. Persze az is lehetséges, hogy az elsőnek elküldött üzenet annyit késik, hogy a második ér oda előbb.

Minden szolgáltatás jellemezhető a megbízhatósági fokával is. Bizonyos szolgáltatások megbízhatók abban az értelemben, hogy sosem vesztenek el adatot. Egy megbízható szolgáltatást rendszerint úgy valósítanak meg, hogy a vevőnek minden megkapott üzenetet nyugtáznia kell, így a küldő biztos lehet abban, hogy az üzenet megérkezett. A nyugtázási folyamat plusz időt és késleltetést jelent, ami legtöbbször megéri, de persze van, amikor nem kívánatos.

A megbízható összeköttetés-alapú szolgáltatás egyik tipikus alkalmazása a fájlátvitel (file transfer). A fájl tulajdonosa biztos szeretne lenni abban, hogy az összes bit rendben megérkezik, és ráadásul ugyanabban a sorrendben, ahogy elküldte. Kevés olyan felhasználó van, aki fájlátvitelnél olyan szolgáltatást részesítene előnyben, amelyik időnként összekever vagy elveszt néhány bitet. Még akkor sem vennének igénybe olyat, ha az sokkal gyorsabb lenne.

A megbízható összeköttetés-alapú szolgáltatásoknak két altípusa van: az üzenetsorozat és a bájtfolyam. Az első esetben megmaradnak az üzenethatárok. Ha elküldünk két 1024 bájtos üzenetet, akkor azok mindig két különálló 1024 bájtos üzenet formájában érkeznek meg, és sohasem egy 2048 bájtos üzenetként. A második esetben viszont az összeköttetés egyszerűen csak egy bájtfolyam, és nincsenek üzenethatárok. Ha egy 2048 bájtos üzenet érkezik a vevőhöz, akkor sehogy sem tudja megállapítani, hogy azt két 1024 bájtos üzenet, egy 2048 bájtos üzenet vagy 2048 darab egybájtos üzenet formájában küldték-e el. Ha egy könyv oldalait a hálózaton egyenként, külön üzenetek formájában küldjük el egy fényszedő gépre, akkor fontos, hogy megmaradjanak az üzenethatárok. Amikor viszont egy DVD-filmet töltünk le, akkor csak arra van szükségünk, hogy a bájtfolyam a szerverről eljusson a számítógépünkre. A filmen belül az üzenethatároknak nincs jelentősége.

Bizonyos alkalmazások esetén a nyugtázásból adódó késleltetés elfogadhatatlan. Ilyen alkalmazás például a digitalizált hangforgalom az **IP-hálózaton keresztül történő hangátvitelben** (**Voice over IP, VoIP**). A telefonon beszélgetők számára sokkal inkább elfogadható az, hogy néha egy kis zajt halljanak a vonalon, mint az, hogy a nyugtázások miatt késleltetés jelenjen meg. Hasonlóképpen, videokonferencia továbbításakor néhány hibás pixel még nem jelent problémát, viszont annál idegesítőbb, amikor a kép az átviteli hibák kijavítása miatt folyton megakad.

Nem minden kapcsolat igényel összeköttetést. Például a kéretlen reklámok küldői (spammer) nagyon sok címzettnek küldenek levélszemetet (junk mail). A levélszemetet küldő felhasználók valószínűleg nem akarnak azzal küszködni, hogy minden egyes levél elküldésekor felépítsenek, majd lebontsanak egy összeköttetést. Még csak a 100%-os kézbesítési arány sem fontos, különösen akkor nem, ha még drágább is. Csak arra van

	Szolgáltatás	Példa
Összeköttetés-alapú	Megbízható üzenetfolyam	Könyvlapok sorozata
	Megbízható bájtfolyam	Filmletöltés
	Megbízhatatlan összeköttetés	Internettelefon
Összeköttetés nélküli	Megbízhatatlan datagram	Elektronikus kacatlevelezés
	Nyugtázott datagram	Szöveges üzenetküldés
	Kérés-válasz	Adatbázis-lekérdezés

1.16. ábra. *Hat különböző típusú szolgáltatás*

szükség, hogy egy olyan lehetőség nyíljon az üzenetek elküldésére, ami nagy valószínűséggel célba juttatja azokat, de erre garanciát nem vállal. A nem megbízható (tehát nem nyugtázott) összeköttetés nélküli szolgáltatást gyakran **datagramszolgáltatásnak (datagram service)** is hívják a távirat analógiájára, amelynél szintén nem lehet nyugtát küldeni a feladónak. Annak ellenére, hogy nem megbízható, a későbbiekben tisztázandó okok miatt a legtöbb hálózatban ez az uralkodó szolgáltatástípus.

Vannak olyan esetek, amikor kényelmesebb az, ha nem létesítünk összeköttetést egy rövidebb üzenet továbbításához, de a megbízhatóság alapvető fontosságú. Ezekben az esetekben **nyugtázott datagramszolgáltatást (acknowledged datagram service)** érdemes igénybe venni, ami olyan, mint a tértivevényes levélkézbesítés. Amikor a feladó megkapja a tértivevényt, akkor teljesen biztos lehet abban, hogy a levelet kikézbesítették a címzettnek, és nem veszett el útközben. A mobiltelefonon történő szöveges üzenetküldés példázza ezt a működést.

Egy újabb szolgáltatás a **kérés-válasz szolgáltatás (request-reply service)**. Ennél a szolgáltatásnál az adó datagram formájában elküld egy kérést, amire érkezik a válasz. Kérés-válasz szolgáltatást használunk leggyakrabban a kliens–szerver-modell szerinti kommunikáció megvalósításakor: a kliens küld egy kérést, melyre a szerver válaszol. Például egy mobiltelefon-kliens küldhet egy lekérdezést egy térképszervernek, hogy elkérje tőle az aktuális pozíciója körüli térképadatokat. Az eddig tárgyalt szolgáltatástípusokat az 1.16. ábrán látható táblázatban foglaltuk össze.

Az a tény, hogy megbízhatatlan kommunikációt használunk, elsőre nagyon zavarónak tűnhet. Tulajdonképpen miért is részesítené bárki a megbízható kommunikációval szemben a megbízhatatlan kommunikációt előnyben? Először is, megbízható (vagyis a mi értelmezésünk szerinti nyugtázott) kommunikáció esetleg nem érhető el az adott rétegben. Például az Ethernet nem biztosít megbízható kommunikációt. A csomagok néha megsérülhetnek a továbbítás során. Ennek a problémának a megoldása a magasabb szintű protokollok feladatkörébe tartozik. Másodszor, a megbízható szolgáltatással szükségszerűen együtt járó nagyobb késleltetések elfogadhatatlanok lehetnek, különösen az olyan valós idejű alkalmazásokban, mint például a multimédia-alkalmazások. Mindezen okok indokolják mind a megbízható, mind a megbízhatatlan kommunikáció együttélését, létezését.

1.3.4. Szolgáltatási primitívek

Egy szolgáltatást formálisan a **primitívek (primitives)** vagy elemi műveletek (operations) olyan halmazával adhatunk meg, amelyek az azt igénybe vevő folyamat számára rendelkezésre állnak a szolgáltatás eléréséhez. A primitívekkel egy művelet elvégzésére lehet utasítást adni egy szolgáltatónak, vagy beszámolót lehet kérni egy társentitás tevékenységéről. Amennyiben a protokollkészlet az operációs rendszerben található (ez gyakran van így), a primitívek általában rendszerhívások. Ezek a hívások rendszermódba (kernel mode) kényszerítik a számítógépet, vagyis az operációs rendszernek adják át a gép feletti irányítást, amely így el tudja küldeni a szükséges csomagokat.

A szolgáltatás természete meghatározza a rendelkezésre álló primitívek készletét. Egy összeköttetés-alapú szolgáltatásnak más primitívjei vannak, mint egy összeköttetés nélkülinek. Az 1.17. ábrán szolgáltatási primitívek egy olyan legkisebb készlete látható, amely már alkalmas lehet megbízható bájtfolyam megvalósítására kliens–szerver-környezetben. Ismerősen fognak csengni a Berkeley-csatlakozó határfelület rajongói számára, mivel a primitívek ennek az interfésznek az egyszerűsített változatai.

Ezek a primitívek felhasználhatók például egy kérés-válasz típusú párbeszédhez kliens–szerver-környezetben. Ennek szemléltetésére vázoljunk fel egy egyszerű, nyugtázott datagramszolgáltatást megvalósító protokollt.

Először a szerver végrehajt egy LISTEN primitívet, amellyel azt jelzi, hogy felkészült a bejövő összeköttetések fogadására. A LISTEN-t gyakran egy blokkoló hatású rendszerhívással valósítják meg. Miután a primitívet végrehajtotta, a szerverfolyamat addig nem lép tovább, amíg az összeköttetés létesítésére kérés nem érkezik.

Ezt követően a kliensfolyamat egy CONNECT primitívet hajt végre, hogy összeköttetést építsen ki a szerver felé. A CONNECT hívásban meg kell jelölni, hogy kihez akarunk kapcsolódni, ezért általában egy paraméterben szerepel a szerver címe is. Az operációs rendszer ilyenkor általában egy csomagot küld el a társentitásnak, amellyel az összeköttetés-létesítési kérést jelzi, amint azt (1) is mutatja az 1.18. ábrán. A kliensfolyamatot ekkor a rendszer felfüggeszti addig, amíg válasz nem érkezik.

Amikor a csomag megérkezik a szerverhez, azt az ottani operációs rendszer dolgozza fel. Amikor a rendszer azt látja, hogy a csomagban összeköttetés-létesítési kérés van, megnézi, hogy van-e olyan folyamat, amelyik erre vár. Ha van, akkor megszünteti a várakozó folyamat blokkolását. A kiszolgáló folyamat ilyenkor létrehozhatja a kapcsolatot az ACCEPT hívással. Ez egy nyugtát (2) küld vissza a kliens folyamat számára, jelezve a

Primitív	Jelentése
LISTEN	Blokkolt várakozás bejövő kapcsolatfelvételre
CONNECT	Összeköttetés létrehozása egy várakozó társentitással
ACCEPT	Bejövő kapcsolat elfogadása a társentitástól
RECEIVE	Blokkolt várakozás bejövő üzenetre
SEND	Üzenet küldése a társentitásnak
DISCONNECT	Összeköttetés bontása

1.17. ábra. *Hat szolgáltatási primitív egy egyszerű összeköttetés-alapú szolgáltatás megvalósításához*

kapcsolat elfogadását. Ennek a nyugtának a megérkezése futó állapotba helyezi a klienst. Ezen a ponton a kliens és a szerver egyaránt fut, és közöttük az összeköttetés létrejött.

Nyilvánvaló párhuzam látszik az itt leírt protokoll és az élet között, például abban az esetben, amikor egy ügyfél (kliens) felhívja egy cég ügyfélszolgálatát. Az ügyfélszolgálat munkatársa reggelente odaül a telefonhoz, hátha az csörögni fog. Később az ügyfél hívást kezdeményez, és amikor az ügyfélszolgálat munkatársa felveszi a telefont, az összeköttetés létrejön.

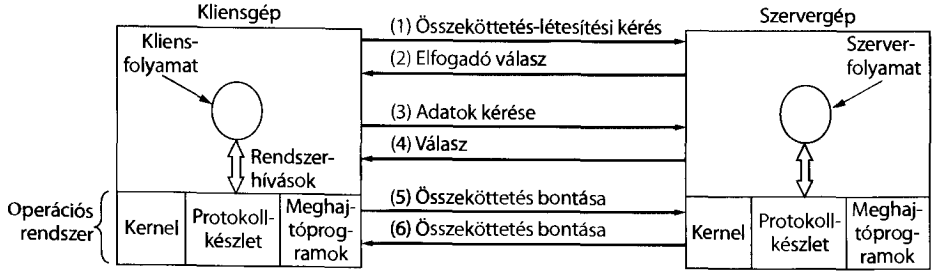
A szerver számára a következő lépés egy RECEIVE primitív végrehajtása, amellyel felkészül az első kérés fogadására. Általában ezt a szerver azonnal megteszi, amint továbbindul a LISTEN hívást követő blokkolódás után, még mielőtt a nyugta megérkezhetne a klienshez. A RECEIVE meghívása blokkolja a szervert.

Ez után a kliens egy SEND primitívet hajt végre a kérés (3) elküldésére, amelyet egy RECEIVE végrehajtása követ azért, hogy a választ venni tudja. Amikor a kérést tartalmazó csomag megérkezik a szervergépre, a rendszer továbbindítja a szerverfolyamatot, hogy az feldolgozhassa a kérést. Miután elvégezte a kért munkát, a szerverfolyamat egy SEND segítségével küldi el a választ (4) a kliensnek. Ennek a csomagnak a megérkezése továbbindítja a klienst, amely így megnézheti a választ. Ha további kérései vannak, azokat most küldheti el.

Ha a kliens befejezte a munkát, a DISCONNECT primitív segítségével szüntetheti meg az összeköttetést (5). Általában az első DISCONNECT egy blokkoló hívás, amely felfüggeszti a klienst, és egy csomagot küld el a szervernek, amellyel azt közli vele, hogy a kapcsolatra nincsen tovább szüksége. Amikor a szerver megkapja ezt a csomagot, szintén egy DISCONNECT hívást hajt végre, amellyel nyugtázza a kérést a kliens felé és elengedi a kapcsolatot (6). Amikor a szerver csomagja visszaérkezik a klienshez, a kliensfolyamat továbbindul, és az összeköttetés megszakad. Röviden összefoglalva így működik az összeköttetés-alapú kommunikáció.

Természetesen az élet nem ennyire egyszerű. Ebben a rendszerben sok minden el tud romlani. Lehet rossz az időzítés (például a CONNECT előbb van, mint a LISTEN), elveszhetnek csomagok, és még sok más hibaforrás is lehet. Ezeket a dolgokat később még részletesen meg fogjuk tárgyalni, de pillanatnyilag az 1.18. ábrán látható modell elégségesen összefoglalja, hogyan működik a kliens–szerver-kommunikáció nyugtázott datagramok használata esetén, hogy figyelmen kívül hagyhassuk a csomagvesztést.

Azt látva, hogy az összeköttetés felépítéséhez ebben a protokollban hat csomagra van szükség, felmerülhet az olvasóban az a kérdés, hogy miért nem használunk inkább ösz-



1.18. ábra. *Egy egyszerű kliens–szerver-kommunikáció nyugtázott datagramszolgáltatás használata esetén*

szeköttetés nélküli protokollt. A válasz az, hogy egy tökéletes világban használhatnánk ilyet, és mindössze két csomagra lenne szükség: egy a kéréshez és egy a válaszhoz. Amikor azonban nagyobb üzeneteket kell továbbítani (például egy néhány megabájtos állományt), történhetnek átviteli hibák és csomagvesztések is, máris egy teljesen más helyzetben találjuk magunkat. Ha a válasz több száz csomagból áll, és ezek közül néhány elveszhet az átvitel során, honnan tudhatná a kliens, hogy ténylegesen elveszett-e valami útközben. Honnan tudhatná a kliens, hogy az utolsónak érkezett csomag az utolsónak feladott-e? Most tegyük fel, hogy a kliens még egy állományt kér. Hogyan tudná megkülönböztetni a második állomány első csomagját az első állomány korábban elveszett első csomagjától, ami hirtelen mégis megérkezik? Röviden összefoglalva, egy egyszerű kérés/válasz protokoll gyakran nem alkalmas arra, hogy megbízhatatlan hálózaton alkalmazzuk. A 3. fejezetben egy sor olyan protokollt fogunk megvizsgálni, amelyek megoldják ezt és más problémákat is. Egyelőre azonban elég azt kijelentenünk, hogy néha az igényeknek nagyon megfelel, ha egy megbízható bitfolyam áll rendelkezésre a folyamatok között.

1.3.5. A szolgáltatások kapcsolata a protokollokkal

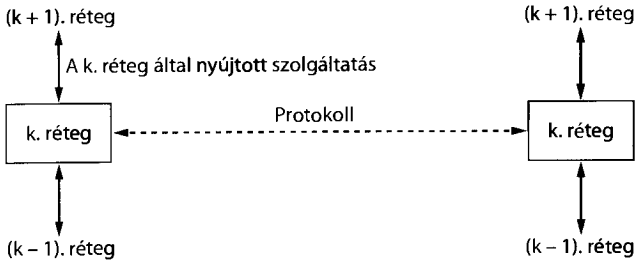
A szolgáltatás és a protokoll különböző fogalmak, mégis gyakran összekeverik őket. A kettő közötti különbség nagyon fontos, ezért ismételten szeretnénk azt hangsúlyozni. A *szolgáltatás* nem más, mint olyan primitívek (elemi műveletek) halmaza, amelyet egy adott réteg a felette levő rétegek számára biztosít. A szolgáltatás azt definiálja, hogy egy réteg a felhasználó nevében milyen műveleteket képes végrehajtani, de arról nem mond semmit, hogy mindezt hogyan kell implementálni. A szolgáltatás két szomszédos réteg közötti interfésszel kapcsolatos, ahol az alsó réteg a szolgáltató, a felső réteg pedig a szolgáltatás felhasználója.

Ezzel szemben a *protokoll* olyan szabályok halmaza, amelyek azt mondják meg, hogy milyen legyen a formátuma és mi legyen a jelentése azoknak a kereteknek, csomagoknak és üzeneteknek, amelyeket egy adott rétegen belül a társentitások küldözgetnek egymásnak. Az entitások a protokollokat használják arra, hogy a szolgáltatásdefiníciókat implementálják. Ha akarják, szabadon megváltoztathatják a protokolljaikat, feltéve, hogy a felhasználó számára látható szolgáltatások ettől nem változnak meg. Ily módon a szolgáltatást és a protokollt teljesen ketté lehet választani. Ez egy kulcsfontosságú elgondolás, melyet minden hálózattervezőnek jól kell értenie.

Másképp megfogalmazva: a szolgáltatások a rétegek közötti interfésszel kapcsolatosak, ahogyan az az 1.19. ábrán is látható. Ezzel szemben a protokollok a különböző gépeken elhelyezkedő társentitások között elküldött csomagokkal kapcsolatosak. Fontos, hogy ne keverjük össze e két fogalmat.

Érdemes összehasonlítást tenni a programozási nyelvekkel. A szolgáltatás olyan, mint egy absztrakt adattípus vagy egy objektum egy objektumorientált nyelvben. Definiálja azokat a műveleteket, amelyeket az objektumon végre lehet hajtani, de nem mondja meg, hogy a műveleteket hogyan kell implementálni. A protokoll a szolgáltatás *implementációjának* felel meg, és mint ilyen, láthatatlan a szolgáltatást igénybe vevő számára.

Sok régebbi protokoll nem tett különbséget a szolgáltatás és a protokoll között. Ezekben egy tipikus réteg akár egy olyan SEND PACKET szolgáltatási primitívvel is rendel-



1.19. ábra. *A szolgáltatások és a protokollok viszonya*

kezhetett volna, amelyet a felhasználónak egy teljesen összeállított csomagra mutató pointert biztosít. Ez azt jelenti, hogy a protokoll minden változása azonnal látható volt a felhasználó számára. A legtöbb hálózattervezéssel foglalkozó szakember az ilyen protokollokat nagy baklövésnek tartja.

1.4. Hivatkozási modellek

Miután elméletben megtárgyaltuk a rétegekbe szervezett hálózatokat, itt az ideje tanulmányoznunk néhány példát is. Két fontos hálózati architektúrát fogunk megvizsgálni, az OSI és a TCP/IP hivatkozási modellt. Bár az OSI-modellhez kapcsolódó *protokollokat* már szinte egyáltalán nem használják, maga a *modell* tulajdonképpen elég általános, így még mindig érvényes, az egyes rétegeknél megtárgyalandó feladatok pedig manapság is nagyon fontosak. A TCP/IP-modell ezzel eléggé ellentétes tulajdonságokkal bír: maga a modell nem túl hasznos, de protokolljai széles körben használatosak. Mindezek miatt mindkét modellt részletesen meg fogjuk vizsgálni. Egy másik fontos indokunk az, hogy néha a bukásokból többet lehet tanulni, mint a sikerekből.

1.4.1. Az OSI hivatkozási modell

Az OSI hivatkozási modell – az átviteli közeg ábrázolása nélkül – az 1.20. ábrán látható. Ez a modell a Nemzetközi Szabványügyi Szervezet (International Standards Organization, ISO) ajánlásán alapul, és a különböző rétegekben használt protokollok nemzetközi szabványosítása terén az első lépésnek tekinthető [Day és Zimmermann, 1983]. Ezt a modellt hivatalosan ISO **OSI (Open System Interconnection – nyílt rendszerek összekapcsolása)** hivatkozási modellnek nevezik, mivel nyílt rendszerek összekapcsolásával foglalkozik. A nyílt rendszerek olyan rendszerek, amelyek képesek más rendszerekkel való kommunikációra. Az egyszerűség kedvéért mi csak **OSI-modellnek** nevezzük a továbbiakban.

Az OSI-modellnek hét rétege van. A hét rétegre történő felosztás elvei a következők voltak:

1. A rétegek különböző absztrakciós szinteket képviseljenek.

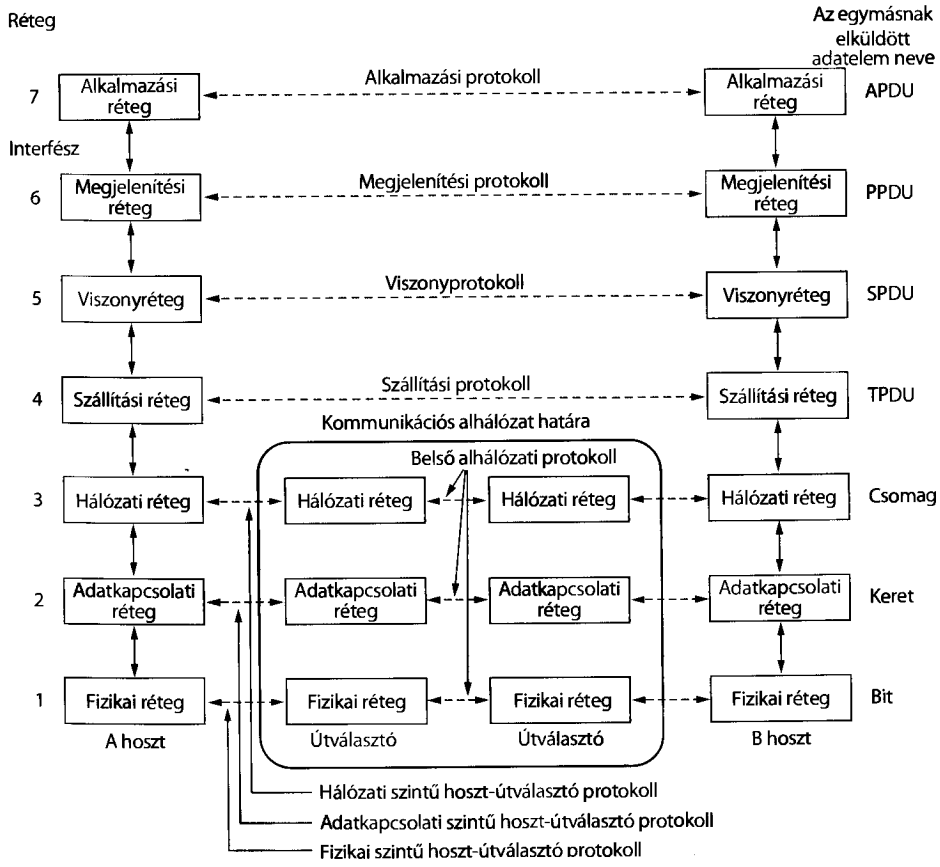
2. Minden réteg jól definiált feladatot hajtson végre.

3. A rétegek feladatának definiálásakor a nemzetközileg szabványosított protokollokat kell figyelembe venni.

4. A rétegek határait úgy kell meghatározni, hogy a rétegek közötti információcsere minimális legyen.

5. A rétegek számának elég nagynak kell lenni ahhoz, hogy eltérő feladatok ne kerüljenek szükségtelenül ugyanabba a rétegbe, viszont elég kicsinek kell lennie ahhoz, hogy az architektúra ne váljon kezelhetetlenné.

A továbbiakban a modell egyes rétegeit fogjuk egyenként bemutatni a legalsóval kezdve. Ne felejtsük el, hogy az OSI-modell nem hálózati architektúra, mivel nem specifikálja az egyes rétegek által használt szolgáltatásokat és a protokollokat. Csak annyit mond, hogy



1.20. ábra. *Az OSI hivatkozási modell*

mit kell tenniük a rétegeknek. Ugyan az ISO az egyes rétegekhez szabványokat is kidolgozott, azonban magának a hivatkozási modellnek ezek nem részei. Viszont valamennyit közzétették, mint különálló nemzetközi szabványt. A *modellt* (részleteiben) széles körben használják, annak ellenére, hogy a kapcsolódó protokollok már rég feledésbe merültek.

A fizikai réteg

A **fizikai réteg** (**physical layer**) feladata az, hogy továbbítsa a biteket a kommunikációs csatornán. A rétegnek biztosítania kell azt, hogy az egyik oldalon elküldött 1-es bit a másik oldalon is 1-esként érkezzen meg, ne pedig 0-ként. Ez a réteg tipikusan olyan kérdésekkel foglalkozik, hogy mekkora feszültséget kell használni a logikai 1, és mekkorát a logikai 0 reprezentálásához, mennyi ideig tart egy bit továbbítása, az átvitel megvalósítható-e egyszerre mindkét irányban, miként jön létre az összeköttetés, hogyan bomlik le, ha már nincs szükség rá, hány érintkezője van a hálózati csatlakozóknak, mire lehet használni az egyes érintkezőket stb. A tervezési szempontok itt főleg az interfész mechanikai, elektromos és eljárási kérdéseire, valamint a fizikai réteg alatt elhelyezkedő fizikai átviteli közegre vonatkoznak.

Az adatkapcsolati réteg

Az **adatkapcsolati réteg** (**data link layer**) fő feladata az, hogy a fizikai átviteli rendszer szerény adottságait egy olyan vonallá alakítsa, amely a hálózati réteg számára felderítetlen átviteli hibáktól mentesnek látszik. Ezt a tényleges hibák elfedésével valósítja meg, hogy a hálózati réteg már ne lássa azokat. Ezt a feladatot úgy oldja meg, hogy az átviendő adatokat küldő fél oldalán **adatkeretekbe** (**data frame**) – általában néhány száz vagy néhány ezer bájt – tördeli, és ezeket sorrendben továbbítja. Ha a szolgáltatás megbízható, a fogadó fél egy **nyugtázókerettel** (**acknowledgement frame**) nyugtázza minden egyes keret helyes vételét.

Az adatkapcsolati rétegben (és a legtöbb felsőbb rétegben is) felmerül az a kérdés, hogy hogyan lehet megelőzni azt, hogy egy gyors adó annyi csomaggal árasszon el egy lassú vevőt, amennyit az már nem képes fogadni. Valamilyen forgalomszabályozó eszközre van szükség ahhoz, hogy az adót tájékoztatni lehessen arról, ha a vevő készen áll további adatok fogadására.

Az adatszóró hálózatok adatkapcsolati rétegében még egy dolgot szabályozni kell: az osztott csatornához való hozzáférést. Az adatkapcsolati réteg egy alrétege foglalkozik ezzel a feladattal, amelyet **közeghozzáférés-vezérlő alrétegnek** (**medium access control sublayer**) neveznek.

A hálózati réteg

A **hálózati réteg** (**network layer**) az alhálózat működését irányítja. A legfontosabb kérdés itt az, hogy milyen útvonalon kell a csomagokat a forrásállomástól a célállomásig eljuttatni. Az útvonalak meghatározása történhet statikus táblázatok felhasználásával, amelyeket „behuzaloznak" a hálózatba, és csak nagyon ritkán változtatnak. Az útvonalat minden egyes párbeszéd (például terminálviszony) előtt külön is meghatározhatjuk. Végül az útvonal kiválasztása lehet kifejezetten dinamikus: ilyenkor minden csomag számára a hálózat aktuális terhelésének ismeretében egyenként kerül kijelölésre az útvonal.

Ha egyszerre túl sok csomag van jelen az alhálózatban, akkor azok akadályozhatják egymást, és ekkor torlódások alakulhatnak ki. Az ilyen torlódások kezelése is a hálózati réteg feladatai közé tartozik, a felsőbb rétegekkel együttműködve, melyek a hálózati terhelésüket alakítják a helyzetnek megfelelően. Még általánosabban: a nyújtott szolgáltatásminőség (késleltetés, átviteli idő, sebességingadozás stb.) szintén a hálózati réteg hatáskörébe tartozik.

Ha viszont a csomagnak az egyik hálózatból át kell mennie egy másikba ahhoz, hogy elérje a címzett állomást, akkor még több probléma jelentkezik: az első hálózatban használt címzési mód más, mint a második hálózatban; a második hálózat egyáltalán nem fogadja a csomagot, mert az túl hosszú; a két hálózat protokollja különbözik és így tovább. A hálózati rétegnek a feladata az, hogy legyőzze ezeket az akadályokat, és lehetővé tegye az egymástól eltérő hálózatok összekapcsolását.

Az adatszóró hálózatokban az útvonalválasztás viszonylag egyszerű feladat, így ezekben a hálózatokban a hálózati réteg gyakran elég vékony, sőt van, amikor nem is létezik.

A szállítási réteg

A **szállítási réteg** (**transport layer**) legfontosabb feladata az, hogy adatokat fogadjon a viszonyrétegtől, – ha szükséges – feldarabolja azokat kisebb egységekre, továbbítsa ezeket a hálózati rétegnek, és biztosítsa azt, hogy minden kis egység hibátlanul megérkezzen a másik oldalra. Ráadásul, mindezt hatékonyan kell elvégezni és oly módon, hogy a felsőbb rétegek számára rejtve maradjanak a hardvertechnikában az idő folyamán jelentkező változások.

A szállítási rétegben dől el az is, hogy milyen típusú szolgáltatásokat nyújt a viszonyrétegnek és ezzel tulajdonképpen az is, hogy milyen szolgáltatások állnak a hálózat felhasználóinak rendelkezésére. A szállítási összeköttetések legnépszerűbb fajtája egy olyan hibamentes kétpontos csatorna, amely a küldés sorrendjében továbbítja az üzeneteket és a bájtokat. Más lehetséges szállítási szolgáltatástípusok is vannak, mint például a különálló üzenetek továbbítása anélkül, hogy garanciát vállalnánk a kézbesítés sorrendjére, vagy az üzenetek adatszórásos szállítása egyszerre több címzetthez. A szolgáltatás típusa akkor dől el, amikor az összeköttetés felépül. (Tulajdonképpen egy ténylegesen hibamentes csatornát lehetetlen elérni; ez a kifejezés igazából azt jelenti, hogy a hibaarány olyan kicsi, hogy az a gyakorlatban figyelmen kívül hagyható.)

A szállítási réteg egy igazi végpontok közötti réteg, a forráshosszttól egészen a célhosztig tart. Másképpen megfogalmazva: a forrásgépen futó egyik program beszélget egy hasonló programmal a célgépen, felhasználva az üzenetek fejléceit és a vezérlőüzeneteket. Az alacsonyabb szintű rétegek protokolljait a gépek a közvetlen szomszédjukkal való kommunikációhoz használják, nem pedig a tényleges küldő és a fogadó kommunikál velük, hiszen ezeket több útválasztó is elválaszthatja egymástól. Az egymáshoz kapcsolódó 1–3. réteg és a végpontok közötti 4–7. réteg közötti különbség az 1.20. ábrán látható.

A viszonyréteg

A **viszonyréteg** (**session layer**) teszi lehetővé, hogy két gép egy **viszonyt** (**session**) vagy **munkamenetet** hozzon létre egymás között. A viszonyok sokféle szolgáltatást valósítanak meg, mint például a **párbeszédirányítás** (**dialog control**) – az adás jogának kiosztása és nyomon követése, a **vezérjelkezelés** (**token management**) – annak megakadályozására szolgál, hogy ketten egyszerre próbálják ugyanazt a kritikus műveletet végrehajtani, és a **szinkronizáció** (**synchronization**) – ellenőrzési pontokat iktat a hosszú adásokba, hogy egy hibát követő helyreállás után az ellenőrzési ponttól lehessen folytatni az adást).

A megjelenítési réteg

Szemben az alacsonyabb szintű rétegekkel, a **megjelenítési réteg** (**presentation layer**) nem a bitek mozgatásával foglalkozik, hanem az átvitt információ szintaktikájával és szemantikájával. Annak érdekében, hogy a különböző adatábrázolást használó gépek kommunikálni tudjanak, a párbeszéd során használt adatszerkezeteket és a „vezetékeken" használandó szabványos kódolást absztrakt módon kell definiálni. A megjelenítési réteg ezekkel az absztrakt adatszerkezetekkel foglalkozik, és lehetővé teszi a magasabb szintű adatszerkezetek (például banki ügyféladatlap) definiálását, valamint a gépek közötti átvitelét.

Az alkalmazási réteg

Az **alkalmazási réteg** (**application layer**) olyan protokollok változatos sokaságát tartalmazza, amelyekre a felhasználóknak gyakran szüksége van. Egy széleskörűen használt alkalmazási protokoll a **HTTP** (**HyperText Transfer Protocol – hipertext-átviteli protokoll**), amely a világháló működésének alapja. Amikor egy böngésző meg akar szerezni egy weblapot, a HTTP segítségével küldi el a kért lap nevét a szervernek. A szerver erre visszaküldi neki a lapot. További alkalmazási protokollok léteznek állományok átvitelére, e-levelezésre és a hálózati hírcsoportok eléréséhez.

1.4.2. A TCP/IP hivatkozási modell

A továbbiakban térjünk át az OSI hivatkozási modellről a számítógép-hálózatok ősének tekintett ARPANET, illetve annak leszármazottja, a világméretű INTERNET hivatkozási modelljére. Bár később lesz még szó az ARPANET történetéről, azonban elöljáróban érdemes néhány dolgot megemlíteni. Az ARPANET az amerikai védelmi minisztérium (U.S. Department of Defense, DoD) által támogatott kísérleti hálózat volt. Alkalmanként több száz egyetemi és kormányzati számítógépet kötött össze bérelt telefonvonalak segítségével. Miután később műholdas és rádiós hálózatokat is hozzákapcsoltak, és az akkori protokollok csak nehezen tudtak együttműködni, egy új hivatkozási modell vált szükségessé. Ezért már a kezdetektől fogva az volt a legfőbb tervezési szempont, hogy lehetővé tegyék tetszőlegesen sok hálózat zökkenőmentes összekapcsolását. Később ez az architektúra – a két legjelentősebb protokollja alapján – **TCP/IP hivatkozási modell** néven vált ismertté, amelyet elsőként Cerf és Kahn [1974] definiált, majd Leiner és mások [1985] is behatóan foglalkozott vele. A modell mögött rejlő tervezési problémákról Clark [1988] munkájában olvashatunk.

Mivel a DoD erősen aggódott amiatt, hogy akármelyik nagy értékű hoszt, útválasztó vagy hálózatok közötti átjáró (gateway) egy szempillantás alatt megsemmisülhet, ezért egy másik lényeges tervezési szempont az volt, hogy a hálózat az éppen folyó beszélgetések megszakítása nélkül át tudja vészelni az alhálózat esetleges veszteségeit. Más szóval, a DoD azt akarta, hogy amíg a forrás- és célállomások jól működnek, a kapcsolatok ne szakadjanak meg még akkor sem, ha egy köztük levő másik gép vagy valamelyik átviteli vonal hirtelen meghibásodik. Ráadásul, egy flexibilis architektúrára volt szükség, mivel az alkalmazások a fájlátviteltől kezdve a valós idejű beszédátvitelig bezárólag rendkívül eltérő igényeket támasztottak.

A kapcsolati réteg[7]

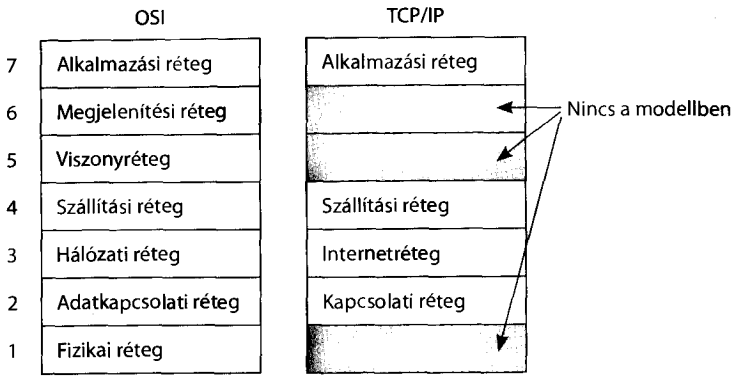
Mindezek az elvárások olyan csomagkapcsolt hálózathoz vezettek, amely egy összeköttetés nélküli rétegen alapul, és különböző hálózatok között is működőképes. A modell legalsó rétege, a **kapcsolati réteg** (**link layer**) azt írja le, hogy milyen képességekkel kell rendelkeznie az olyan átviteli elemeknek, mint amilyenek a soros vonalak vagy a klasszikus Ethernet, hogy megfeleljenek ennek az összeköttetés nélküli internetréteg igényeinek. A kifejezés megszokott értelmében nem is valódi réteg, hanem egy csatlakozási felület a hosztok és az átviteli összeköttetések között. A TCP/IP-modellről szóló korai cikkek és könyvek keveset szólnak róla.

Az internetréteg

Az **internetréteg** (**internet layer**) az összekötő kapocs, amely az egész architektúrát összefogja. Az 1.21. ábrán láthatjuk, hogy hozzávetőlegesen az OSI hálózati rétegnek feleltethető meg. A feladata az, hogy lehetővé tegye a hosztok számára, hogy bármely hálózatba csomagokat tudjanak küldeni, illetve a csomagok egymástól függetlenül célba jussanak (akár más hálózatokba is). Az sem gond, ha a csomagok nem az elküldés sorrendjében érkeznek meg, ugyanis, ha erre van szükség, akkor a magasabb rétegek visszarendezik őket a megfelelő sorrendbe. Ne felejtsük el, hogy itt az „internet" szót most általános értelemben használjuk annak ellenére, hogy ez a réteg az internetben is jelen van.

Vegyünk egy hasonló példát, mondjuk a (csigalassúságú) postát. Ha valaki bedob egy adag külföldre szóló levelet a postaládába, akkor kis szerencsével azok jó része meg is érkezik a helyes külföldi címre. Útjuk során a levelek nagy valószínűséggel keresztülmennek egy-két nemzetközi postaközponton, azonban ebből a feladó semmit nem vesz észre.

7 Eredetileg ezt a réteget Network Interface-nek nevezték, és magába foglalta a fizikai réteg funkcióit is. *(A lektor megjegyzése)*



1.21. ábra. *A TCP/IP hivatkozási modell*

Ráadásul minden országnak (azaz hálózatnak) saját bélyege és saját szabványos méretű borítékja van. Ezenkívül a kézbesítés szabályai is rejtve maradnak az ügyfelek elől.

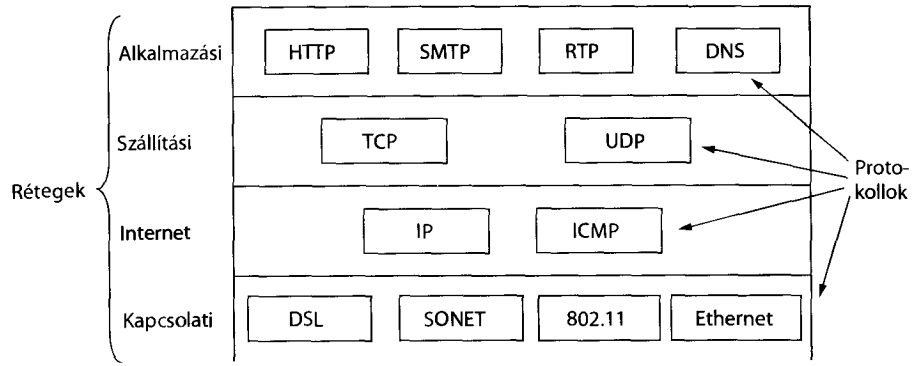
Az internetréteg meghatároz egy hivatalos csomagformátumot, illetve egy protokollt, amelyet **internetprotokollnak** (**Internet Protocol, IP**) hívnak, plusz egy ezt kísérő másik protokollt, az **internetes vezérlőüzenet protokollt** (**Internet Control Message Protocol, ICMP**), mely az előbbi működését segíti. Az internetréteg feladata az, hogy kézbesítse az IP-csomagokat a címzetteknek. A csomagok útvonalának meghatározása az egyik legfontosabb feladat, a másik a torlódások elkerülése (bár az IP nem bizonyult túlságosan hatékonynak ez utóbbiban).

A szállítási réteg

A TCP/IP-modellben az internetréteg fölötti réteget általában **szállítási rétegnek**[8] nevezik. Feladata az OSI-modell szállítási rétegéhez hasonlóan az, hogy lehetővé tegye a küldő és címzett hosztokban található társentitások közötti párbeszédet. Két különböző szállítási protokollt definiálunk a következőkben. Az egyik az **átvitelvezérlő protokoll** (**Transmission Control Protocol, TCP**), amely egy megbízható összeköttetés-alapú protokoll. Feladata az, hogy hibamentes bájtos átvitelt biztosítson bármely két gép között az interneten. A beérkező bájtos adatfolyamot diszkrét méretű üzenetekre osztja, majd azokat egyesével továbbítja az internetrétegnek. A címzett hoszt TCP-folyamata összegyűjti a beérkezett üzeneteket, és egyetlen kimeneti adatfolyamként továbbítja azokat. A TCP forgalomszabályozást is végez annak érdekében, hogy egy gyors forráshoszt csak annyi üzenetet küldjön egy lassabb címzett hosztnak, amennyit az fogadni képes.

A másik protokoll ebben a rétegben a **felhasználói datagram protokoll** (**User Datagram Protocol, UDP**), amely egy nem megbízható, összeköttetés nélküli protokoll. Jelentősége akkor van, amikor nem szükséges sem az üzenetek TCP-féle sorba rendezése, sem a forgalomszabályozás. Elsősorban olyan egylövetű, kliens–szerver típusú kérés-

8 Eredetileg ezt a réteget End-to-end vagy Host-to-host rétegnek nevezték. Ez jobban kifejezte a célját. *(A lektor megjegyzése)*



1.22. ábra. *A TCP/IP hivatkozási modell néhány protokollja, melyet tanulmányozni fogunk*

válasz alkalmazásokban terjedt el, ahol a gyors válasz sokkal fontosabb, mint a pontos. Ilyen alkalmazás például a beszéd- vagy videoátvitel. Az IP, a TCP és az UDP kapcsolatát az 1.22. ábra szemlélteti. Mivel az itt látható modell fejlesztés eredménye, ezért az IP-t még sok más hálózat is használja.

Az alkalmazási réteg

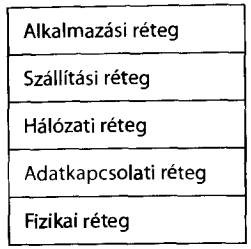
A TCP/IP-modellben nincs viszony- és megjelenítési réteg. Azért nem kerültek bele a modellbe, mert nem mutatkozott irántuk igény. Ehelyett az alkalmazások tartalmazzák azokat a viszony és megjelenítési funkciókat, amelyekre szükségük van. Az OSI-modellel kapcsolatos tapasztalok is bizonyítják ezt a nézetet: a legtöbb alkalmazás nemigen használja ki e két réteget.[9]

A szállítási réteg fölött az **alkalmazási réteg** (**application layer**) található. Ez tartalmazza az összes magasabb szintű protokollt. A korai protokollok között találhattuk a virtuális terminál (TELNET), a fájltranszfer (FTP) és az elektronikus levelezés (SMTP) protokolljait. Az évek során sok más protokollal bővült a lista. Az 1.22. ábrán felsorolt fontosabb, később tárgyalandó protokollok közé tartozik a körzetnévkezelő rendszer (Domain Name System, DNS), ami a hosztnevek és hálózati címek megfeleltetésére szolgál, a HTTP, a világháló oldalainak letöltésére szolgáló protokoll, és az RTP, a valós idejű média (hang és mozgókép) továbbításának protokollja.

1.4.3. A könyvben használt modell

Amint azt már korábban is említettük, az OSI hivatkozási modell erőssége a *modell* önmaga (nem számítva a megjelenítési és viszonyréteget), mely kivételesen hasznosnak bi-

9 Ezzel a nézettel sokan vitatkoznak. Például a megjelenítési réteg hiánya miatt a titkosítás érdekében számos új alkalmazási protokollt kellett kidolgozni, a korábbi alkalmazási protokollok titkosított változataként. *(A lektor megjegyzése)*



1.23. ábra. *A könyvben használt hivatkozási modell*

zonyult a számítógép-hálózatok leírásában. Ezzel szemben a TCP/IP hivatkozási modell erősségét a protokollok adják, melyeket sok éve széles körben használnak. Mivel a számítógép-tudománnyal foglalkozó szakemberek „azt szeretik megenni, amit maguk főztek", ezért a könyv alapjául az 1.23. ábrán szemléltetett hibrid modellt fogjuk használni.

Ez a modell öt rétegből áll, a fizikai rétegtől indul, az adatkapcsolati, hálózati és szállítási rétegen keresztül jut fel az alkalmazási rétegig. A fizikai réteg határozza meg, hogy hogyan kell villamos (vagy más analóg) jelek formájában biteket továbbítani különféle átviteli közegeken keresztül. Az adatkapcsolati réteg azzal foglalkozik, hogy hogyan lehet közvetlenül összekapcsolt számítógépek között adott megbízhatósággal véges hosszú üzeneteket átküldeni. Az Ethernet és a 802.11 egy-egy példa az adatkapcsolati rétegbeli protokollokra.

A hálózati réteg azzal foglalkozik, hogyan lehet több adatkapcsolatot hálózattá alakítani, valamint hálózatok hálózatából hogyan lehet összekapcsolt hálózatot úgy kialakítani, hogy távoli számítógépek között csomagokat küldhessünk. Itt a feladat olyan útvonal megtalálása, amelyen a csomagokat küldeni kell. Ebben a rétegben az IP lesz a fő tanulmányozott protokoll. A szállítási réteg erősíti a hálózati réteg kézbesítési garanciáit, általában fokozott megbízhatósággal, és olyan kézbesítési absztrakciókat nyújt, mint a megbízható bájtfolyam, ami sok különféle alkalmazás igényeit elégíti ki. A TCP fontos példája a szállítási réteg protokolljainak.

Végezetül az alkalmazási réteg tartalmazza azokat a programokat, amelyek a hálózatot használják. Sok, de nem minden hálózati alkalmazás rendelkezik felhasználói felülettel, ilyen például egy webböngésző. Mindenesetre minket a programoknak az a része érdekel, amelyik a hálózatot használja. A webböngésző esetében ez a HTTP-protokoll. Fontos segédprogramok is találhatók az alkalmazási rétegben, mint amilyen a DNS, melyet nagyon sok alkalmazás használ.

A fejezeteink sorrendje is ezen a modellen alapszik. Ilyen módon meg tudjuk tartani az OSI-modell értékét a hálózati architektúrák értelmezésében, de közben olyan protokollokra koncentrálhatunk, melyek a gyakorlatban fontosak, a TCP/IP-től és a kapcsolódó protokolloktól kezdve, egészen az olyan újabb protokollokig, mint amilyen a 802.11, a SONET és a Bluetooth.

1.4.4. Az OSI és a TCP/IP hivatkozási modell összehasonlítása

Az OSI és a TCP/IP hivatkozási modellnek sok közös tulajdonsága van. Mindkettő hierarchikusan egymásra épülő, de egymástól független protokollokon alapul. Az egyes

rétegek funkciója is nagyjából megegyezik. Például a szállítási és az alatta levő többi réteg azért van benne mindkét modellben, hogy hálózatfüggetlen, végpontok közötti szállítási szolgáltatást nyújtson az egymással kommunikálni szándékozó folyamatok számára. Ezek a rétegek alkotják a szállítási szolgáltatót. A szállítási réteg feletti rétegek mindkét modellben a szállítási réteg alkalmazásorientált felhasználói.

Mindezen alapvető hasonlóságok ellenére a két modell sok eltérést is mutat. Ebben a bekezdésben most csak a leglényegesebb különbségekről ejtünk néhány szót. Fontosnak tartjuk megjegyezni, hogy a *hivatkozási modelleket* hasonlítjuk össze, nem pedig a *protokollkészleteket*. Magukat a protokollokat később tárgyaljuk. A TCP/IP- és az OSI-modell összehasonlításával egyébként egy teljes könyv foglalkozik [Piscitello és Chapin, 1993].

Az OSI-modell három fogalom köré összpontosul:

1. szolgáltatások,
2. interfészek,
3. protokollok.

Az OSI-modellnek az a legnagyobb vívmánya, hogy éles különbséget tesz e három fogalom között. Mindegyik réteg szolgáltatásokat nyújt a felette levő rétegnek. A *szolgáltatás* azt definiálja, hogy egy réteg mit csinál, nem pedig azt, hogy a felette levő entitások hogyan érik el az adott szolgáltatást, illetve, hogy a réteg hogyan működik.

A réteg *interfésze* megmondja a felette levő folyamatoknak, hogy hogyan vehetik igénybe az adott réteg szolgáltatásait. Megadja a lehetséges paramétereket és azt, hogy milyen eredményt vár. Ez sem tartalmaz semmit arról, hogy a réteg hogyan is működik belül.

Egy adott rétegben található *társprotokollok* működése csak a rétegre tartozik. Egy konkrét feladat elvégzéséhez (tehát szolgáltatás nyújtásához) a réteg olyan protokollt használ, amilyet csak akar. Tetszése szerint válthat egyikről a másikra anélkül, hogy a felette levő rétegek szoftvereinek működését befolyásolná.

Ez a koncepció igen közel áll az objektumorientált programozás koncepciójához. Egy objektum, mint például egy réteg, számos olyan metódussal (működéssel) rendelkezik, amelyeket objektumon kívüli folyamatok (processzek) kívülről meghívhatnak. Ezeknek a metódusoknak a szemantikája határozza meg azoknak a szolgáltatásoknak a halmazát, amelyet az objektum felkínál. A metódusok paraméterei és az eredményei az objektum interfészét alkotják. Az objektumon belül található kód az ő saját protokollja, és az a külvilág számára láthatatlan.

A TCP/IP-modell kezdetben nem tett ilyen világos különbséget a szolgáltatás, az interfész és a protokoll között, bár később voltak rá kísérletek, hogy kicsit OSI-szerűbbé tegyék a modellt. Például az internetrétegben csak a SEND IP PACKET és a RECEIVE IP PACKET tekinthető valódi szolgáltatásnak. Következésképpen, az OSI-modell protokolljai jobban el vannak rejtve, mint a TCP/IP-modellé, és emiatt viszonylag könnyebben lehet őket módosítani a technológiai fejlődés előrehaladtával. A protokollok rétegezésével az egyik legfőbb célunk éppen az, hogy az ilyen változtatásokat el tudjuk végezni.

Az OSI-modellt még a protokollok kidolgozása *előtt* találták ki. Ennek köszönhetően a modellt nem befolyásolta egyetlen konkrét protokollkészlet sem, és emiatt kellően általá-

nos tudott maradni. Gondot csak az jelentett, hogy a tervezőknek kevés tapasztalata volt ezen a szakterületen, és nemigen tudták, hogy melyik funkciót melyik réteghez rendeljék.

Az adatkapcsolati réteg, például, eredetileg csak a kétpontos hálózatokkal foglalkozott. Amikor megjelentek az adatszóró hálózatok, a modellbe egy új alréteget kellett bepréselni. Amikor aztán az OSI-modell alapján elkezdtek hálózatokat építeni (csodák csodája), rájöttek arra, hogy azok nem felelnek meg a szolgáltatások specifikációinak, ezért az eltérésekből fakadó problémák megoldására konvergencia alrétegeket illesztettek a modellbe. És végül, kezdetben a bizottság arra számított, hogy minden országban csak egyetlen hálózat lesz, amelyet az adott ország kormánya tart majd fenn, és ez a hálózat az OSI-protokollt fogja majd használni. Akkoriban még senki nem gondolt a hálózatok összekapcsolására. Röviden szólva a dolgok másképp alakultak.

A TCP/IP-modellel viszont pont a fentiek ellenkezője történt: először megvolt a protokoll, majd a modell tulajdonképpen a meglevő protokollok leírását adta meg. A protokolloknak a modellbe történő beillesztésével nem is volt semmi gond, tökéletesen ment. Az egyetlen bökkenő csak az volt, hogy ez a *modell* semelyik más protokollrendszerhez nem illeszkedett. Következésképpen alkalmatlan volt arra, hogy más, nem TCP/IP-hálózatokat leírjunk vele.

Hogy a filozofikus hangvétel helyett kicsit közérthetőbben fogalmazzunk, a két modell között a legnyilvánvalóbb különbség a rétegek számában van. Az OSI-modellben hét réteg van, míg a TCP/IP-modellben csak négy. Mindkettőben van hálózati (vagy internetwork), szállítási és alkalmazási réteg, de a többi réteg már nem egyezik meg.

További különbség jelentkezik az összeköttetés-alapú, illetve az összeköttetés nélküli kommunikáció területén. Az OSI-modell mindkettőt támogatja a hálózati rétegben, a szállítási rétegben viszont már csak az összeköttetés-alapú kommunikációt. (Ez azért lényeges, mert a szállítási réteg szolgáltatásai a felhasználó számára is láthatók.) A TCP/IP-modell hálózati rétegében csak összeköttetés nélküli átviteli mód létezik, ugyanakkor a szállítási réteg mindkét változatot támogatja, és a választást a felhasználóra bízza. Az összeköttetés típusának kiválasztása különösen fontos az egyszerűbb kérés-válasz protokollok esetén.

1.4.5. Az OSI hivatkozási modell és protokolljainak bírálata

Sem az OSI-, sem a TCP/IP-modell és azok protokolljai sem tökéletesek. Mindkettőt lehet bizonyos mértékig bírálni, és ezt meg is szokták tenni. Ebben és a következő bekezdésben ismertetünk néhány ilyen bírálatot. Először az OSI-modellel kezdjük, és aztán térünk majd rá a TCP/IP-modellre.

1989-ben, a könyv második kiadásának megjelenésekor még a témával foglalkozó legtöbb szakembernek úgy tűnt, hogy az OSI-modell és protokolljai meghódítják majd a világot, és minden mást elsöpörnek az útjukból. Mégsem ez történt. Vajon miért nem? Egy kis kitekintés a tanulságokra biztosan nem hiábavaló. Az okok ugyanis a következők voltak:

1. rossz időzítés,
2. rossz technológia,

3. rossz implementálás,
4. rossz üzletpolitika.

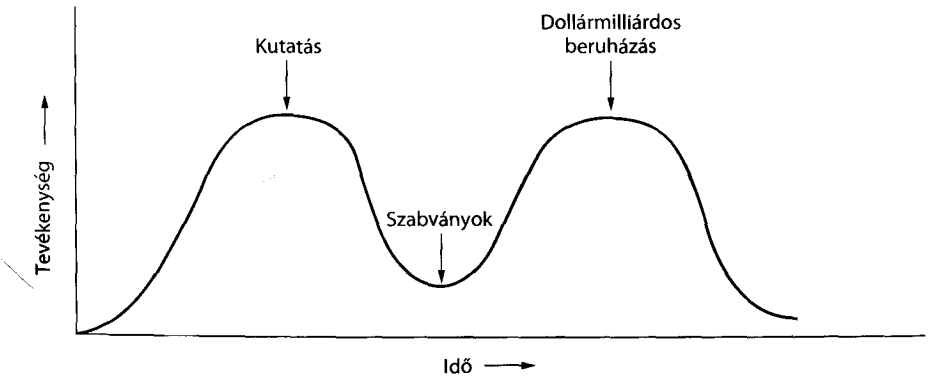
Rossz időzítés

Először vizsgáljuk meg az első okot, a rossz időzítést. Egy szabvány megjelentetésének időpontja rendkívül erősen befolyásolhatja annak sikerét. David Clark, az M.I.T. munkatársa kidolgozott egy elméletet a szabványokról, amelyet ő a *két elefánt apokalipszise* névvel illetett. Hogy mit is jelent ez, azt az 1.24. ábra szemlélteti.

Az ábra azt mutatja be, hogy egy új dolog megvalósítása mennyi munkát igényel. A dolog felfedezése után óriási munka következik, viták zajlanak, cikkek jelennek meg, találkozókra kerül sor. Aztán hamarosan egy kis szünet következik, majd a vállalatok is felfedezik maguknak a dolgot, és több milliárd dolláros beruházások indulnak el.

Nagyon fontos, hogy a szabványosítást a két „elefánt" közötti időben kell elvégezni. Ha túl korán, még a kutatások befejezése előtt készül el, akkor az újdonságról még keveset tudunk, ami rossz szabványt eredményez. Ha viszont túl későn írjuk meg, akkor addigra már számos vállalat, különböző irányokban nagy beruházásokba kezdett, és ezért nagyrészt figyelmen kívül hagyják a szabványt. Ha a két elefánt közötti idő nagyon szűkös (mert mindenki igyekszik minél előbb elkészülni vele), akkor a szabvány kifejlesztésén dolgozó emberek könnyen összeroppanhatnak.

Sajnos úgy tűnik, hogy az OSI-protokollok bizony összeroppantak. Mire az OSI-protokollok megjelentek, addigra a versenytárs TCP/IP-protokollok már széles körben elterjedtek a kutatóegyetemeken. Ugyan a több milliárd dolláros beruházások még nem érték el a csúcsot, az oktatási szférában a piac már olyan nagy volt, hogy sok kereskedő cég elkezdte óvatosan árusítani a TCP/IP-termékeket. Amikor az OSI megjelent a piacon, nem akart támogatni egy második protokollkészletet egészen addig, amíg rá nem kényszerítették erre, így kezdetben nem tudta eladni a termékeit. A vállalatok egymásra vártak az első lépés megtételét illetően, így aztán mivel egyikük sem lépett, az OSI-nál semmi nem történt.



1.24. ábra. *A „két elefánt apokalipszise"*

Rossz technológia

A második ok, amit az OSI sosem értett meg, az az, hogy mind a modell, mind a protokollok hibásak. A rétegek száma inkább politikai, mint műszaki okokból kifolyólag lett hét, és közülük kettő (a viszony és a megjelenítési) majdnem üres, míg a másik kettő (az adatkapcsolati és a hálózati) túltelített.

Az OSI-modell és az általa definiált szolgáltatások és protokollok különösen bonyolultak. Ha egymásra pakolnánk a kinyomtatott szabványokat, a papírkupac több mint fél méter magas lenne. A megvalósításuk nehéz, és működésük is kevéssé hatékony. Ezzel kapcsolatban Paul Mockapetris találós kérdése [idézi Rose, 1993] juthat eszünkbe:

- Mit kapsz, amikor gengsztert keresztezel egy nemzetközi szabvánnyal?
- Valakit, aki olyan ajánlatot tesz neked, amit nem értesz.

Szintén érthetetlen az is, hogy miért jelennek meg újra és újra az OSI egyes rétegeiben olyan funkciók, mint amilyen a címzés, a forgalomszabályozás vagy a hibajavítás. Saltzer és mások [1984] a könyvükben rámutattak arra, hogy a hatékonyság érdekében a hibajavítást a legfelső rétegbe kell tenni, tehát gyakran teljesen fölösleges és gazdaságtalan az alacsonyabb rétegekben többször megismételni.

Rossz implementálás

A modell és a protokollok rendkívüli bonyolultsága miatt nem csoda, hogy az implementációk kezdetben terjedelmesek, kezelhetetlenek és lassúak voltak. Mindenki megbukott, aki próbálkozott vele. Nem telt bele sok idő, és az „OSI"-ról mindenkinek a „gyenge minőség" jutott az eszébe. Bár az idők során egyre jobbak lettek a termékek, a kialakult kép nem változott.

Ugyanakkor a TCP/IP egyik első implementációja a Berkeley-féle UNIX része volt, és nem csak nagyon jó, de még ingyenes is volt. Az emberek gyorsan rászoktak, így komoly felhasználói tábora alakult ki. Ennek köszönhetően egyre jobb lett a termék, ami tovább növelte a felhasználók körét. Ebben az esetben tehát a spirális pálya felfelé irányult, nem pedig lefelé.

Rossz üzletpolitika

A kezdeti implementációk miatt különösen az oktatási szférában sokan azt hitték a TCP/IP-ről, hogy a UNIX része, márpedig ott a UNIX igen népszerű volt az 1980-as években.

Az OSI-ra ugyanakkor mindenki úgy tekintett, mintha az európai távközlési minisztériumok, az Európai Gazdasági Közösség és az amerikai kormány alkotása lett volna. Ez csak részben volt igaz, és az sem segített a helyzeten, hogy kormányhivatalnokok egy csoportja megpróbálta a szerencsétlen kutatók és programozók nyakába varrni a kudarcot. Voltak néhányan, akik ezt az esetet hasonlóan ítélték meg ahhoz, mint amikor az 1960-as években az IBM bejelentette, hogy a PL/I lesz a jövő programozási nyelve, vagy amikor a DoD később ezt úgy módosította, hogy az Ada lesz az.

1.4.6. A TCP/IP hivatkozási modell bírálata

A TCP/IP-modellnek és protokolljainak szintén megvannak a maga hibái. Először is a modell nem tesz világos különbséget a szolgáltatás, az interfész és a protokoll fogalma között. Megfelelő szoftvermérnöki tapasztalat kell ahhoz, hogy különbséget tudjunk tenni specifikáció és implementáció között, amit az OSI nagyon gondosan kezel, és amivel a TCP/IP pedig egyáltalán nem foglalkozik. Így tehát a TCP/IP-modell aligha használható irányadóként új technológiákon alapuló hálózatok tervezésénél.

Másodsorban, a TCP/IP-modell egyáltalán nem általános érvényű, és a TCP/IP-n kívül más protokollkészletek leírására nem igazán alkalmas. Például a TCP/IP-modell segítségével szinte lehetetlen lenne leírni Bluetooth-t.

Harmadsorban, a kapcsolati réteg a hagyományos értelemben véve nem is valódi réteg – legalábbis abban az értelemben nem, ahogyan azt a protokollok kapcsán gondolnánk –, hanem csak interfész (a hálózati és az adatkapcsolati réteg között). Márpedig döntő fontosságú, hogy különbséget tudjunk tenni az interfész és a réteg között. E tekintetben nem szabad felületesnek lennünk.

Negyedsorban, a TCP/IP-modell nem különbözteti meg (sőt meg sem említi) a fizikai és az adatkapcsolati réteget, pedig ez két teljesen különböző dolog. A fizikai rétegnek a rézvezeték, az optikai kábel és a vezeték nélküli kommunikáció átviteli jellemzőivel kell foglalkoznia. Az adatkapcsolati réteg feladata pedig a keretek elejének és végének jelzése, valamint a keretek megbízható továbbítása két gép között. Egy jó modellnek külön rétegként kell kezelnie e kettőt. A TCP/IP-modell sajnos nem ezt teszi.

Végül meg kell említeni, hogy bár az IP- és a TCP-protokollt alaposan átgondolták, és jól implementálták, a többi protokoll nagy része ad hoc jellegű volt. Ezeket többnyire egy tucat egyetemista készítette ütve-vágva, amíg el nem fáradtak. Mivel a protokollimplementációk ingyenesek voltak, ezért széles körben elterjedtek, mélyen beépültek a rendszerekbe, és emiatt nehezen lehetett azokat lecserélni, ami még ma is kisebb-nagyobb problémákhoz vezet. Például a virtuális terminál protokollt, a TELNET-et, egy másodpercenként tíz karaktert feldolgozó mechanikus Teletype terminálhoz tervezték. Egyáltalán nem is ismeri a grafikus felhasználói felületet, sem az egeret. Mindezek ellenére, a mintegy 30 év eltelte után még mindig használják.

1.5. Hálózati példák

A számítógép-hálózatok témakör sok különféle hálózatot fed le, kicsiket és nagyokat, közismerteket és kevésbé közismerteket. Különbözők lehetnek a célok, a méretek és az alkalmazott műszaki megoldások. A következő szakaszban néhány példán keresztül azt fogjuk bemutatni, hogy milyen változatos terület is a számítógép-hálózatoké.

Első példánk az internet lesz, amely valószínűleg a világ legismertebb hálózata. Megismerkedünk a történetével, fejlődésével és műszaki megoldásaival. Ezután a mobiltelefon-hálózatokkal fogunk foglalkozni. Műszakilag nagyon különböznek az internettől, ezért szépen szembeállíthatók. Következőnek bemutatjuk a 802.11 nevű hálózatot, a

vezeték nélküli LAN-ok uralkodó szabványát. Végül vetünk egy pillantást az RFID-ra és a szenzorhálózatokra, melyek olyan technikák, amelyek a hálózat alkalmazását kiterjesztik a fizikai világra és a mindennapi tárgyakra.

1.5.1. Az internet

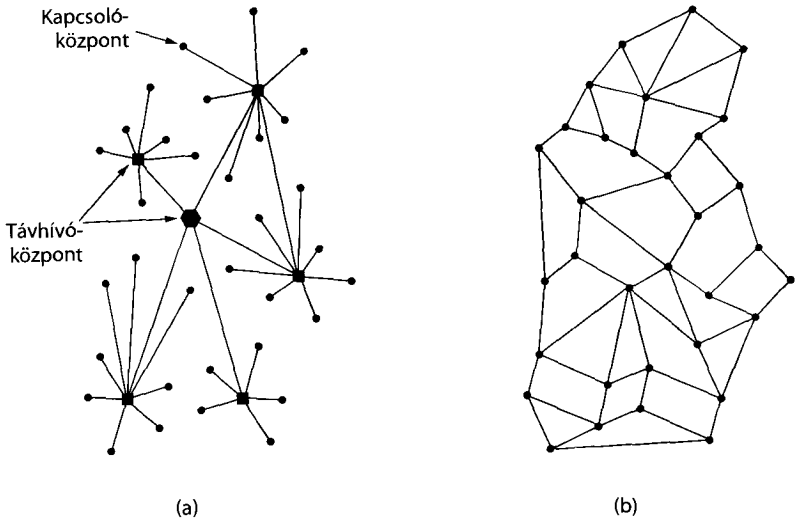
Az internet tulajdonképpen nem is hálózat, hanem különféle hálózatok óriási gyűjteménye, amelyek néhány közös protokollt használnak, és néhány közös szolgáltatást nyújtanak. Ez egy nem szokványos rendszer abból a szempontból, hogy senki sem tervezte meg, és senki sem ellenőrzi. Hogy ezt megérthessük, kezdjük a történetet az elején ott, hogy miért és hogyan fejlesztették ki. Amennyiben kíváncsi az internet csodálatos történetére, olvassa el John Naughton [2000] könyvét. Ez ugyanis egyik azok közül a ritka könyvek közül, amelyet nemcsak olvasni élvezetes, de a kötet végén 20 oldalnyi hivatkozást és további olvasnivalót is találhat az olvasó. Természetesen számos műszaki könyv jelent már meg az internetről és annak protokolljairól is. További olvasmányként például Maufer [1999] könyve szolgálhat.

Az ARPANET

A történet az 1950-es évek végén kezdődik. A hidegháború tetőfokán az amerikai védelmi minisztérium (Department of Defense, DoD) egy olyan parancsnoki és irányítási hálózatot akart létrehozni, amely képes túlélni egy atomháborút. Abban az időben a teljes katonai kommunikáció a nyilvános telefonhálózatot használta, amelyet sebezhetőnek tartottak. Ennek a vélekedésnek a kiváltó oka az 1.25.(a) ábrából kiolvasható. A fekete pontok a telefonközpontokat jelölik, amelyek közül mindegyikhez több ezer telefon kapcsolódik. Ezek a központok hasonlóképpen egy magasabb szintű kapcsoló központhoz (távhívóközponthoz) kapcsolódnak. Ezzel egy olyan országos hálózat alakul ki, amely csak nagyon kevéssé redundáns. A rendszer sebezhetősége éppen abban rejlik, hogy elég néhány kulcsfontosságú távhívóközpontot elpusztítani ahhoz, hogy a rendszer elszigetelt részekre essen szét.

1960 körül a DoD megbízta a RAND Corporationt, hogy keressen megoldást a problémára. Az egyik munkatársuk, Paul Baran az 1.25.(b) ábrán látható, nagymértékben elosztott és hibatűrő rendszert javasolta. A központok között vezető utak hossza itt már nagyobb, mint amennyit egy analóg jel torzulás nélkül képes megtenni, ezért Baran egy digitális csomagkapcsoló megoldást javasolt bevezetni a központokban. Baran jó néhány jelentésben fejtette ki elképzelése részleteit a DoD-nek. A Pentagon hivatalnokainak megtetszett az ötlet, és felkérték az Egyesült Államokban akkor még monopolhelyzetet élvező AT&T-t egy prototípus megépítésére. Az AT&T gondolkodás nélkül elutasította Baran gondolatát. A világ akkor legnagyobb és leggazdagabb cége nem akarta hagyni, hogy egy fiatal mitugrász tanítsa meg neki, hogyan kell telefonrendszert építeni. A vezetők ezért azt mondták, hogy Baran hálózatát nem lehet megépíteni, és ezzel végzetes csapást mértek az ötletre.

Sok év eltelt, és a DoD-nek még mindig nem volt jobb parancsnoki és irányítási hálózata. A soron következő történések megértéséhez először vissza kell tekintenünk 1957 októberére, amikor a Szovjetunió megelőzte az Egyesült Államokat az űrversenyben a Szputnyik, vagyis az első műhold fellövésével. Amikor Eisenhower elnök megpróbálta



1.25. ábra. *(a) A telefonhálózat felépítése. (b) A Baran által javasolt elosztott kapcsolóhálózat*

kideríteni, hogy ki volt a felelős az amerikai késlekedésért, elképedve tapasztalta, hogy a hadsereg, a haditengerészet és a légierő veszekszik a Pentagon kutatásokra szánt költségvetésén. Azonnali válaszlépésként megteremtett egy védelmi célú központi kutatási szervezetet, az **ARPA**-t (**Advanced Research Project Agency – Fejlett Kutatások Ügynöksége**). Az ARPA-nak nem voltak tudósai és laboratóriumai, sőt tulajdonképpen csak egyetlen irodája volt és (a Pentagonnál szokásoshoz képest) szerény költségvetése. A munkája abból állt, hogy támogatásokat és szerződéseket adott a szerinte ígéretes ötletekkel előálló egyetemeknek és vállalatoknak.

Az első néhány évben az ARPA még csak azt próbálta kitapasztalni, hogy tulajdonképpen mi legyen a feladata, de 1967-ben a hálózatok felkeltették az ARPA akkori igazgatója, Larry Roberts figyelmét. Különféle szakértőkkel beszélt, hogy el tudja dönteni, mit tegyen. Az egyik szakértő, Wesley Clark egy olyan csomagkapcsolt alhálózat megépítését javasolta, amelyen minden hosztnak saját útválasztója van.

Roberts eleinte kételkedett, de később megtetszett neki az ötlet, és készített egy némileg homályos előadást [Roberts, 1967] az ACM SIGOPS konferenciájára, amelyet 1967 végén tartottak az operációs rendszerek alapelveiről a Tennessee-i Gatlinburgban. Roberts nagy meglepetésére a konferencia egy másik előadása is egy hasonló rendszert írt le, amely ráadásul nem csak terv volt, hanem már meg is valósították az angol National Physical Laboratoryban (Nemzeti Fizikai Laboratórium, NPL) Donald Davis irányítása alatt. Az NPL rendszere nem volt országos hálózat, hanem csak néhány számítógépet kötött össze az NPL területén belül, de ennek ellenére is azt bizonyította, hogy a csomagkapcsolás megvalósítható. Mindezen felül pedig a cikk hivatkozott Baran addigra elutasított ötletére. Roberts úgy hagyta el Gatlinburgot, hogy már elhatározta, megépíti azt a hálózatot, amely később **ARPANET** néven vált ismertté.

Az alhálózatnak átviteli vonalakkal összekapcsolt miniszámítógépekből, ún. **csomóponti gépekből** (**Interface Message Processor**, **IMP**) kellene felépülnie. A nagyobb

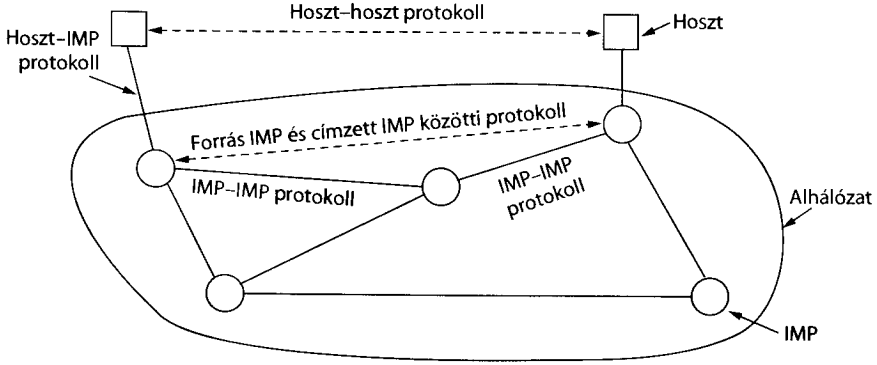
megbízhatóság érdekében mindegyik csomóponti gépnek legalább két másik csomóponti géphez kellene csatlakoznia. Az alhálózat egy datagramos alhálózat lenne, így néhány vezeték vagy csomóponti gép elpusztítása esetén az üzeneteket automatikusan más alternatív útvonalakon lehetne továbbítani.

Minden hálózati csomópontban kell lennie egy csomóponti gépnek és egy hosztnak, lehetőleg ugyanabban a szobában és rövid vezetékekkel összekötve. A hoszt legfeljebb 8063 bites üzeneteket küldhetne a hozzá csatlakozó csomóponti gépnek, amely legfeljebb 1008 bites csomagokra darabolná fel azokat, majd egymástól függetlenül továbbítaná a csomagokat a címzett állomás felé. Továbbítás előtt minden csomag esetén meg kellene várni, amíg a teljes csomag megérkezik, tehát ez az alhálózat az első elektronikus, tárol-és-továbbít típusú csomagkapcsolt hálózat lenne.

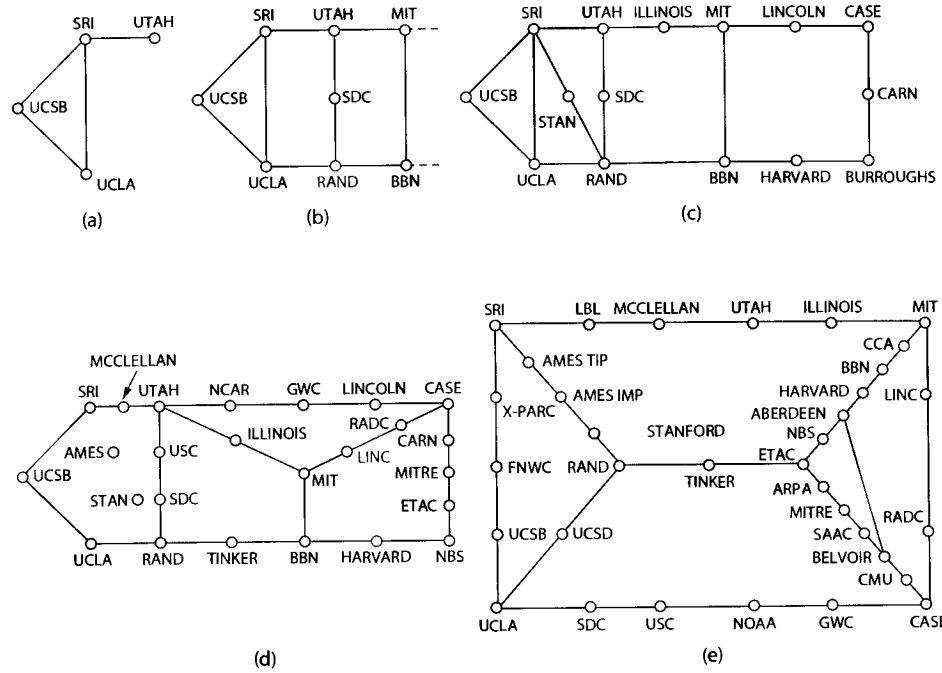
Az ARPA tendert írt ki az alhálózat megépítésére, amire 12 cég jelentkezett. A beadott pályázatok kiértékelése után az ARPA a Cambridge-i BBN tanácsadó céget választotta ki a munka elvégzésére. 1968 decemberében a BBN-nel megkötötték a szerződést az alhálózat kiépítésére és az alhálózat szoftverének megírására. A csomóponti gépek a Honeywell DDP-316 miniszámítógép egy speciálisan módosított változatai voltak. Ezek a gépek 12 K 16 bites szót tartalmazó memóriával rendelkeztek. A csomóponti gépekben nem volt diszk, mert a mozgó alkatrészeket nem tartották elég megbízhatónak. A csomóponti gépeket a telefontársaságoktól bérelt 56 kb/s-os vonalak segítségével kapcsolták egymáshoz. Bár ma már az 56 kb/s-os vonalakat leginkább azok a tizenévesek használják, akik nem tudják az ADSL-t vagy a kábeles kapcsolatot megfizetni, akkoriban még ez volt a legjobb, amit kapni lehetett.

A szoftvert az alhálózatnak és a hosztoknak megfelelően két részre osztották. Az alhálózati szoftver egyrészt a hoszt–csomóponti gép kapcsolatnak a csomóponti gép felőli protokollját és a csomóponti gép–csomóponti gép protokollt tartalmazta, valamint a nagyobb megbízhatóság érdekében a forrás csomóponti gép és a címzett csomóponti gép közötti protokollt. Az ARPANET eredeti tervét az 1.26. ábra mutatja.

Az alhálózaton kívül szintén szükség volt bizonyos protokollokra. Idetartozott a hoszt–csomóponti gép kapcsolat hoszt felőli oldala, a hoszt–hoszt protokoll és az alkalmazás szoftvere. Hamarosan kiderült, hogy a BBN befejezettnek tekintette a feladatát



1.26. ábra. *A kezdeti ARPANET-hálózat*



1.27. ábra. *Az ARPANET fejlődése. (a) 1969. december (b) 1970. július (c) 1971. március (d) 1972. április (e) 1972. szeptember*

azzal, hogy a hoszttól a csomóponti géphez érkező üzeneteket egyszerűen csak áttette a címzett csomóponti gép és a címzett hoszt közötti vonalra.

Robertsnek csak egyetlen problémája volt: a hosztoknak szoftverre is szükségük volt. Hogy megbirkózzon ezzel a problémával, 1969 nyarán egy találkozót hívott össze a Utah állambeli Snowbirdbe, a hálózatokkal foglalkozó kutatóknak, akik akkoriban többnyire még egyetemisták voltak. Az egyetemisták azt várták, hogy egy hálózati szakember majd elmagyarázza nekik a teljes hálózat és a hozzá kapcsolódó szoftver nagy tervét, és aztán mindenkinek kiad egy kis részt, hogy azt írja meg. Teljesen meghökkentek, amikor megtudták, hogy nincs hálózati szakember, és nem létezik nagy terv sem. Nekik maguknak kellett kitalálniuk, hogy mit is kell csinálni.

Ugyanakkor 1969 decemberében kezdett kibontakozni egy olyan kísérleti hálózat, amelynek négy csomópontja volt; egy az UCLA-n, egy az UCSB-n, egy az SRI-n és egy a Utahi Egyetemen. Azért választották ezt a négy helyet, mert mind a négynél igen sokan dolgoztak ARPA-szerződéssel, továbbá mindegyiküknél különböző típusú és egymással inkompatibilis számítógépek voltak (csak hogy még viccesebb legyen a helyzet). Az első hoszt–hoszt üzenetet két hónappal korábban küldte az UCLA csomópontról az SRI csomópontra a Len Kleinrock által vezetett csoport (Kleinrock a csomagkapcsolás elméletének úttörője). A hálózat gyorsan terebélyesedett, ahogy egyre több csomóponti gépet szállítottak le és helyeztek üzembe, és hamarosan behálózta az egész országot. Az 1.27. ábra azt mutatja be, hogy hogyan terjeszkedett az ARPANET az első három évben.

A még gyerekcipőben járó ARPANET fejlődése érdekében az ARPA kutatásokba kezdett a műholdas hálózatok és a mobil csomagkapcsolású rádiós hálózatok területén is. Az egyik híres demonstrációs kísérletben, egy Kaliforniában közlekedő teherautó a csomagkapcsolású rádiós hálózat segítségével üzeneteket küldött az SRI-nek, ahonnan az ARPANET-en továbbították azokat a keleti partra. Onnan az üzenetek a műholdas hálózaton keresztül jutottak el a londoni University College-be. Ezáltal lehetőség nyílt arra, hogy Kaliforniában egy teherautón utazó kutató egy londoni számítógépet használhasson.

Ez a kísérlet ugyanakkor azt is világossá tette, hogy az ARPANET protokolljai nem igazán megfelelők több hálózatból álló rendszerek esetén. Ez az észrevétel a protokollok további fejlesztéséhez vezetett, aminek csúcspontja a TCP/IP-modell és a TCP/IP-protokollok kifejlesztése volt [Cerf és Kahn, 1974]. A TCP/IP-t kifejezetten az internethálózatokon való kommunikációra tervezték, amire egyre nagyobb szükség is volt, miután az ARPANET-hez kapcsolódó hálózatok száma rohamosan nőtt.

Annak érdekében, hogy ösztönözzék az új protokollok befogadását, az ARPA számos megbízást kötött a TCP/IP megvalósítására különböző számítógépes platformokon, többek között IBM-, DEC- és HP-rendszereken, valamint a Berkeley-féle UNIX operációs rendszeren. A Berkeley Egyetem (University of California at Berkeley) kutatói újraírták a TCP/IP-t a Berkeley UNIX soron következő, 4.2BSD kiadásában egy új programozási felület használatával, melyet **hálózati interfésznek (socket)** neveztek el. Számos alkalmazást, segédprogramot, valamint hálózati menedzsment programot írtak annak érdekében, hogy bemutassák, mennyire kényelmes a hálózatok használata socketek segítségével.

Az időzítés tökéletes volt. Sok egyetem pont akkoriban rendelte meg második vagy harmadik VAX számítógépét egy LAN-nal együtt, ami összekapcsolta a számítógépeket, viszont nem volt hozzá hálózati szoftverük. Amikor a 4.2BSD megjelent a TCP/IP-vel, a socketekkel és számos hálózati segédprogrammal, a teljes programcsomagot pillanatok alatt átírták. Ráadásul a TCP/IP segítségével könnyű volt a LAN-okat az ARPANET-hez csatlakoztatni, és ezt a lehetőséget sokan ki is használták.

Az 1980-as években további hálózatokkal, főleg LAN-okkal bővült az ARPANET. Ahogy a gépek száma nőtt, egyre költségesebbé vált egy bizonyos hoszt megkeresése, ezért létrehozták a **DNS- (Domain Name System – körzetnévkezelő rendszer)** rendszert. A DNS-rendszer célja az, hogy a gépeket körzetekbe szervezze, és a hosztok neveit leképezze az IP-címükre. Azóta a DNS egy olyan általánosított, elosztott adatbázis-rendszerként működik, amelyben az elnevezésekkel kapcsolatos mindenféle információt eltárolnak. Ezzel részletesebben is foglalkozunk majd a 7. fejezetben.

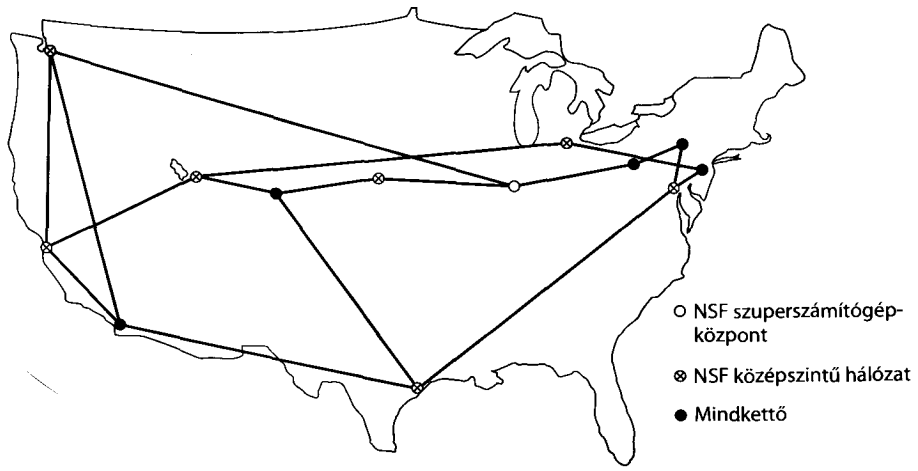
NSFNET

Az 1970-es évek végére az NSF (National Science Foundation, Amerikai Nemzeti Tudományos Alap) is látta, mekkora hatással van az ARPANET az egyetemi kutatásra azzal, hogy lehetővé teszi az országban szétszórtan dolgozó tudósoknak az adatok megosztását és a kutatási programokban való együttműködést. Az ARPANET-re való csatlakozáshoz azonban szükség volt egy kutatási szerződésre a DoD-vel, amivel sok egyetem nem rendelkezett. Az NSF kezdeti válasza erre az volt, hogy megalapította a **CSNET (Computer Science Network – számítástudományi hálózat)** nevű hálózatot 1981-ben. Számítástudományi tanszékeket és ipari kutatólaboratóriumokat csatlakoztatott betárcsázós és bérelt vonalakkal az ARPANET-hez. Az 1980-as évek végén az NSF továbblépett, és úgy döntött, hogy létrehozza az ARPANET egy olyan utódját, mely minden egyetemi kutatócsoport számára rendelkezésre állna.

Az NSF úgy határozott, hogy először egy olyan gerinchálózatot épít, amely a hat szuperszámítógépes központját köti össze San Diegóban, Boulderben, Champaignben, Pittsburghben, Ithacában és Princetonban. Minden szuperszámítógép kapott egy kistestvért, egy LSI-11 mikroszámítógépet, amelyeket **fuzzballnak** („szőrcsomó") hívtak. A fuzzballok 56 kb/s-os vonalakkal összekötve alkották az alhálózatot, az ARPANET műszaki megoldásával azonos módon. A szoftver azonban másmilyen volt: a fuzzballok a TCP/IP-t használták egészen a kezdetektől fogva, így ez az első TCP/IP-re épülő WAN lett.

Az NSF ezenkívül néhány (később körülbelül 20) olyan területi hálózatot is támogatott, amelyek a gerinchálózatra kapcsolódtak. Ezek segítségével az egyetemek, a kutató laboratóriumok, a könyvtárak és a múzeumok felhasználói tudtak hozzáférni a szuperszámítógépekhez és kommunikálni egymással. A teljes hálózatot, vagyis a gerinchálózatot és a területi hálózatokat együtt **NSFNET**-nek keresztelték el. A hálózat az ARPANET-hez a Carnagie-Mellon egyetem géptermében kapcsolódott, ahol egy IMP-t kötöttek össze egy fuzzballal. Az első NSFNET gerinchálózat az 1.28. ábrán látható az Egyesült Államok térképére rajzolva.

Az NSFNET azonnal nagy sikereket könyvelhetett el, és a beindítás pillanatától kezdve túlterhelt volt. Az NSF rögtön belekezdett az utód megtervezésébe és szerződést kötött a michigani székhelyű MERIT konzorciummal az üzemeltetésére. 448 kb/s-os fényvezetőszálas csatornákat béreltek az MCI-től (ami azóta már egyesült a WorldCommal), és ezek képezték a gerinchálózat második változatának alapját. IBM PC-RT-ket állítottak be útválasztónak. Az új hálózat is csak kevés ideig bírta, és így 1990-re a második gerinchálózatot 1,5 Mb/s-os sebességűre fejlesztették.



1.28. ábra. *Az NSFNET gerinchálózata 1988-ban*

A növekedés folytatódása arra ébresztette rá az NSF-et, hogy a kormányzat nem tudja a végtelenségig pénzelni a hálózatok kiépítését. Ezenkívül a kereskedelmi szervezetek is csatlakozni akartak, de az NSF szabályozása tiltotta, hogy kereskedelmi szervezetek olyan hálózatot használjanak, amelyért korábban az NSF fizetett. Ezekből az okokból kifolyólag az NSF arra biztatta a MERIT-et, az MCI-t és az IBM-et, hogy hozzanak létre egy nonprofit céget, az **ANS**-t (**Advanced Networks and Services – fejlett hálózatok és szolgáltatások**), ami az első lépés volt a kereskedelmi célú hálózatok felé. 1990-ben az ANS átvette az NSFNET feletti irányítást, és az addig 1,5 Mb/s-os vonalak sebességét 45 Mb/s-osra növelte. Ez a hálózat, az ANSNET, 5 évig működött, mielőtt eladták az America Online-nak. Ekkor már sok vállalat biztosított IP-szolgáltatást piaci alapon, és már a kormányzat is tisztán látta, hogy ki kell vonulnia a hálózati szolgáltatások területéről.

Az NSF négy különböző hálózatüzemeltetővel kötött szerződést egy-egy **NAP** (**Network Access Point – hálózati hozzáférési pont**) kiépítésére, hogy megkönnyítse ezt a váltást, és hogy biztosítsa azt, hogy minden területi hálózat minden másik területi hálózattal kommunikálhasson. Ez a négy hálózatüzemeltető a PacBell (San Francisco), az Ameritech (Chicago), az MFS (Washington) és a Sprint (New York City és vonzáskörzete) volt. Ahhoz, hogy gerinchálózati szolgáltatást nyújthassanak az NSF-nek, a hálózatüzemeltetőknek az összes NAP-hoz kapcsolódniuk kellett.

Ez a kialakítás azt jelentette, hogy egy tetszőleges területi hálózatból származó csomag több gerinchálózati szolgáltató közül választhatott a saját NAP-ja és a címzett NAP közötti út megtételéhez. Ennek folyományaként a gerinchálózati szolgáltatók versenyhelyzetbe kerültek. A területi hálózatok üzemeltetői a szolgáltatások és az árak alapján választhattak közülük, és az ötlet célja éppen ez volt. Az addigi egyetlen létező gerinchálózatot így leváltotta egy piaci alapon működő, versenyhelyzetet teremtő infrastruktúra. Az amerikaiak szeretik kritizálni a szövetségi kormányt az általa bevezetett újítások alacsony száma miatt, de a DoD és az NSF teremtették meg az internetet később megalapozó infrastruktúrát, hogy azután annak üzemeltetését az iparnak adják át.

Az 1990-es években sok más ország és terület is épített átfogó kutatási hálózatot, amelyek gyakran készültek az ARPANET és az NSFNET mintájára. Ezek közül kettő az EuropaNET és az EBONE, két olyan európai hálózat, amelyek 2 Mb/s-os vonalakkal indultak, és azokat később 34 Mb/s-ra bővítették. A hálózati infrastruktúra üzemeltetése egy idő után Európában is az ipar kezébe került át.

Az internet rengeteget változott a kezdeti idők óta. A világháló (world wide web, www) 1990-es évek elején történő felbukkanásával robbanásszerű növekedésnek indult. Az internet Systems Consortium friss adatai 600 millió fölöttire teszik a látható internetes hosztokat. Bár ez csak egy alsó becslés, így is messze túlszárnyalja azt a néhány millió hosztot, ami az első webről szóló, a CERN-ben tartott konferencia idején, 1994-ben létezett.

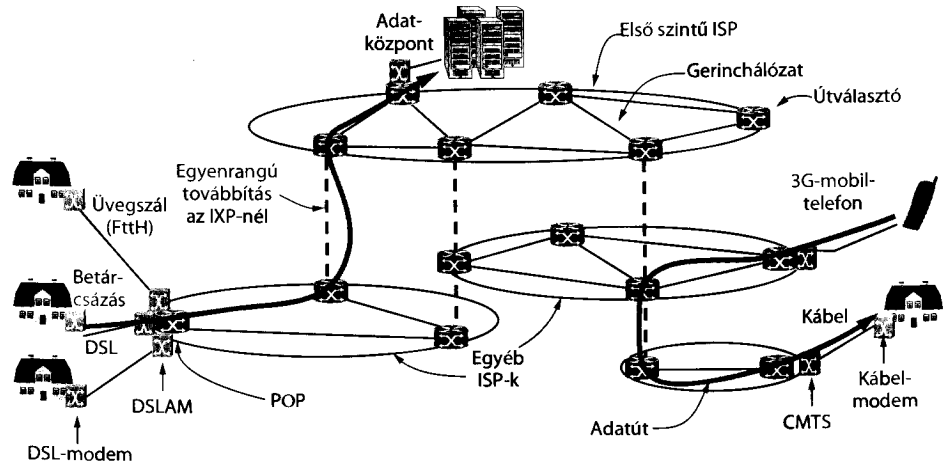
Az internet használatának a módja is gyökeresen megváltozott. Kezdetben a tudományos e-levelezés, a hírcsoportok, a távoli bejelentkezés és a fájlátvitel volt meghatározó. Később ez változott a mindenki által használt e-levelezés, majd a web és az olyan P2P-tartalommegosztás irányában, mint amilyen a mára már megszüntetett Napster. Most a valós idejű médiaelosztás, a közösségi hálózatok (például a Facebook) és a mikroblogolás (például a Twitter) vannak felszálló ágban. Ezek a váltások gazdagabb médiatartalmakat

hoztak az internetre, és így nagyobb forgalmat generálnak. Valójában az internet domináns forgalomtípusa időről időre megváltozni látszik, ahogy például új és jobb zenével vagy filmekkel kapcsolatos megoldások válnak nagyon gyorsan nagyon népszerűvé.

Az internet felépítése

Az internet felépítése szintén rengeteget változott a robbanásszerű növekedése közben. Ebben a szakaszban rövid áttekintést adunk a mai internetről. A kép egy kicsit zavaros, mivel a telefontársaságok, kábelszolgáltatók és ISP-k üzleti tevékenységével kapcsolatban folyamatosan nagy a felfordulás, úgy hogy nehéz kideríteni, hogy az egyes szereplők tulajdonképpen mit is szolgáltatnak. Ennek a felfordulásnak az egyik hajtóereje a távközlés konvergenciája, mely révén egyetlen hálózat lesz használható több korábbi szolgáltatás kiváltására. Például „triple play" (három szolgáltatás egy szolgáltatótól) esetében egyetlen társaságtól vehetünk igénybe telefon-, tv- és internetszolgáltatást ugyanazon a hálózati kapcsolaton keresztül annak reményében, hogy pénzt takarítunk meg. Mindezek folyományaként az itt leírt kép szükségszerűen egyszerűbb a valóságnál. És ami ma még igaz, holnapra lehet, hogy megváltozik.

A teljes kép az 1.29. ábrán látható. Vizsgáljuk meg az ábra egyes részeit, kezdve egy otthoni számítógéptől (az ábra szélein). Hogy a számítógép az internethez kapcsolódjon, egy **internetszolgáltatóhoz** (**Internet Service Provider**), vagy egyszerűen csak **ISP**-hez csatlakozik, akitől a felhasználó **internet-hozzáférést** (**Internet access**) vagy **kapcsolódást** (**connectivity**) vásárol. Ez lehetővé teszi a számítógépnek, hogy csomagokat cseréljen az interneten elérhető összes többi hoszttal. A felhasználó weben való szörfözésre irányuló csomagokat küldhet, vagy ezer más felhasználási módot választhat, mindez nem számít. Sokféle internet-hozzáférés létezik, és ezeket általában az általuk nyújtott sávszélesség és az áruk alapján különböztetjük, de a legfontosabb jellemzőjük a kapcsolódás.



1.29. ábra. *Az internet felépítésének áttekintése*

Az ISP-hez történő kapcsolódásnak egy gyakori módja a lakásba vezető telefonvonal felhasználása, ebben az esetben a telefontársaság az ISP. A **DSL**-technika, ami a **Digital Subscriber Line (digitális előfizetői vonal)** rövidítése, a házakba futó telefonvonalat hasznosítja digitális adatátvitelre. A számítógép egy **DSL-modemnek** nevezett eszközhöz csatlakozik, mely a digitális jelek és a telefonvonalon akadálytalanul átvihető analóg jelek közti átalakítást végzi. A vonal másik végén egy **DSLAM (Digital Subscriber Line Access Multiplexer – digitális előfizetői vonal hozzáférési multiplexer)** nevű eszköz végzi a jelek visszaalakítását.

Az ISP-hez való kapcsolódásnak számos további népszerű módját mutatja az 1.29. ábra. A DSL a helyi telefonvonal sávszélességének nagyobb részét használja fel, mintha beszéd helyett biteket küldenénk át egy hagyományos telefonhívással. Ez utóbbit **betárcsázós (dial-up)** kapcsolatnak nevezik, és mindkét végpontján egy-egy más típusú modemmel valósítják meg a jelek átalakítását és visszaalakítását. A **modem** szó a „modulátor-demodulátor" rövidítése, és egy olyan eszközre vonatkozik, amelyik digitális biteket alakít analóg jelekké és vissza.

Egy következő módszer az, hogy a kábeltévé-hálózatot használják jelek küldésére. A DSL-hez hasonlóan ez is a létező infrastruktúra újrafelhasználása, jelen esetben a kihasználatlan kábeltévé-csatornáké. A lakásokban található végponti eszköz neve **kábelmodem (cable modem)**, a **fejállomáson** levő eszközt pedig a **CMTS**-nek (**Cable Modem Termination System – kábelmodem-véglezáró rendszer**) nevezik.

A DSL és a kábeltévés technika a rendszertől függően másodpercenként a megabit töredékétől annak többszöröséig terjedő sávszélességet nyújthat az internet eléréséhez. Ezek a sebességek sokkal nagyobbak, mint a betárcsázós sebességek, amelyek 56 kb/s sebességre korlátozottak a hanghívások keskeny sávszélesség igénye miatt. A betárcsázós sebességnél lényegesen nagyobb sebességű internet-hozzáférést **széles sávú (broadband) hozzáférésnek** nevezzük. A név a gyorsabb hálózatokban alkalmazott nagyobb sávszélességre utal, nem pedig bármilyen adott sebességre.

Az eddig említett hozzáférési módokat az átviteli út „utolsó mérföldjén" vagy utolsó szakaszán elérhető sávszélesség korlátozza. A lakásokba menő fényvezető szál alkalmazásával gyorsabb internet-hozzáférés biztosítható, 10–100 Mb/s nagyságrendben. Ezt a konstrukciót **FttH-nak (Fiber to the Home – üvegszál a lakásig)** nevezzük. Üzleti területen dolgozó cégek számára logikus lehet nagy sebességű átviteli vonalat bérelni az irodától a legközelebbi ISP-hez. Például Észak-Amerikában egy T3-as vonal hozzávetőlegesen 45 Mb/s sebességgel üzemel.

Vezeték nélküli technikákat is használnak internet-hozzáférésre. Egy példa, melyet rövidesen körül fogunk járni, a 3G mobiltelefonos hálózatoké. 1 Mb/s vagy nagyobb adatátviteli sebességet képesek nyújtani a lefedettségi területen belül tartózkodó mobiltelefonok és kötött pozíciójú előfizetők számára.

Most már képesek vagyunk csomagokat küldeni a lakások és az ISP között. Azt a helyet, ahol az előfizetői csomagok belépnek az ISP-hálózatába, az ISP **POP**-jének (**Point of Presence – szolgáltatási pont**) nevezzük. A következőkben arról lesz szó, hogy milyen módon közlekednek a csomagok a különböző ISP-k POP-jei között. Ettől a ponttól kezdve a rendszer teljesen digitális és csomagkapcsolt.

Az ISP hálózata lehet regionális, nemzeti vagy nemzetközi méretű. Már láttuk, az ISP hálózata az általa kiszolgált különféle városokhoz tartozó POP-kban elhelyezett, egymással kapcsolatban álló útválasztókból és az ezeket összekötő hosszú átviteli vonalakból áll. Ezeket a berendezéseket az ISP **gerinchálózatának (backbone)** nevezzük. Ha egy csomag célja egy olyan hoszt, amelyet közvetlenül az ISP szolgál ki, akkor a csomag a gerinchálózaton keresztül egyenesen a címzett hoszthoz kerül. Egyéb esetekben a csomagot egy másik ISP-nek kell továbbítani.

A szolgáltatói hálózatok a forgalom kicserélése céljából az **IXP-nek (Internet eXchange Point – internetkapcsolódási pont)** nevezett helyeken kapcsolódnak egymáshoz. Az összekapcsolt ISP-k úgynevezett **egyenrangú továbbítással (peering)**[10] kapcsolódnak egymáshoz. Világszerte sok IXP található a városokban. Az 1.29. ábrán függőlegesen lettek berajzolva, mert a szolgáltatói hálózatok földrajzilag részben átfedik egymást. Lényegében egy IXP egy olyan szoba, amely tele van útválasztókkal, és minden ISP-hez tartozik legalább egy. A szobán belül egy LAN köt össze minden útválasztót, lehetővé téve azt, hogy a csomagokat bármely ISP-gerinchálózatról bármely másik ISP-gerinchálózatra továbbítani lehessen. Az IXP-k lehetnek nagy és önálló szervezetek is. Az egyik legnagyobb az Amsterdam Internet Exchange, amelyhez több száz ISP kapcsolódik, és amelyen keresztül több száz Gb/s forgalmat bonyolítanak le egymás között.

Az adatcserélő központokban megvalósuló peering jellege az ISP-k közötti üzleti kapcsolattól függ. Sokféle üzleti viszony lehetséges. Például egy kisebb ISP fizethet egy nagyobb ISP-nek a kapcsolódásért, ahogy az ügyfelek is megvásárolják a szolgáltatást az internetszolgáltatótól. Ebben az esetben a kisebb ISP úgymond a **tranzitforgalomért (transit)** fizet. Egy másik lehetőség, hogy két nagy ISP úgy dönt, mindketten külön fizetség nélkül juttathatnak el bizonyos mennyiségű forgalmat a másik hálózatába. A számos internetparadoxon közül az egyik, hogy azok az ISP-k, amelyek nyilvánosan versenyeznek egymással az előfizetőkért, gyakran együttműködnek a színfalak mögött a továbbítás területén [Metz, 2001].

Egy csomag útvonala az interneten keresztül az ISP-k összekapcsolódási megegyezéseitől függ. Ha egy csomagot továbbító szolgáltató társult a címzett szolgáltatóval, akkor a csomag közvetlenül is megérkezhet a társhoz. Ellenkező esetben a csomagot a legközelebbi olyan helyre irányítja, ahol egy fizetett tranzit szolgáltatóhoz kerül, hogy az kézbesítse a csomagot. Két példaútvonalat rajzoltunk be az 1.29. ábrára. A csomagok útvonala gyakran nem a legrövidebb út lesz az interneten keresztül.

A táplálkozási lánc legtetején a nagy gerinchálózati szolgáltatók foglalnak helyet, olyan cégek, mint az AT&T vagy a Sprint. Ezek a cégek üzemeltetik a nagy nemzetközi gerinchálózatokat, több ezer, fényvezető szálakkal összekötött útválasztó segítségével. Ezek az ISP-k nem fizetnek a tranzitforgalomért. Általában **első szintű (tier 1)** ISP-nek nevezzük őket, és úgymond ezek alkotják az internet gerincét, mivel mindenki másnak hozzájuk kell kapcsolódnia, hogy elérje a teljes internetet.

A nagy mennyiségű tartalmat kínáló cégek, mint a Google és a Yahoo! olyan **adatközpontokban (data center)** helyezik el a számítógépeiket, amelyeknek jó minőségű kapcsolata van az internet többi része felé. Ezeket az adatközpontokat számítógépek számára tervezik, nem pedig emberek számára készülnek, és tömve lehetnek állványokon álló számítógépekkel, amelyet **szerverfarmnak (server farm)** neveznek. A szerverelhe-

10 A peering valójában ennél sokkal több. Azt is jelenti, hogy az azonos szintű ISP-k az egymástól kapott forgalomért kölcsönösen nem számolnak fel továbbítási díjat. *(A lektor megjegyzése)*

lyezéssel (**colocation, hosting**) foglalkozó adatközpontokban az ügyfelek közvetlenül az ISP POP-jénél helyezhetik el a berendezéseiket, hogy rövid és gyors összeköttetés legyen a szerverek és az ISP gerinchálózata között. Az internetes szerver elhelyezési iparág egyre inkább virtualizálttá kezd válni, tehát egyre gyakoribb a szerverfarmokon futó virtuális gépek bérlése a fizikai számítógépek telepítése helyett. Ezek az adatközpontok olyan méretűek (tíz- vagy százezer géppel is rendelkezhetnek), hogy az elektromos áram adja a költségek nagy részét, így az adatközpontok gyakran olyan helyre épülnek, ahol olcsó a villamos áram.

Ezzel az internet gyors bemutatásának a végéhez értünk. A következő fejezetekben még sok mindent fogunk elmondani egyes elemeiről és azok felépítéséről, az alkalmazott algoritmusokról és protokollokról. Érdemes itt még megjegyeznünk, hogy átalakulóban van, hogy mit jelent az internetre csatlakozva lenni. A korábbi álláspont az volt, hogy egy számítógép az interneten van, ha: (1) a TCP/IP-protokollkészletet futtatja, (2) rendelkezik IP-címmel és (3) képes IP-csomagot küldeni az interneten levő összes többi hosztnak. A szolgáltatók azonban gyakran újrahasznosítják az IP-címeket az alapján, hogy mely számítógépek vannak éppen használatban, és az otthoni hálózatokban levő gépek is gyakran osztoznak egy közös IP-címen. Ez a gyakorlat aláássa a második feltételt. Az olyan biztonsági megfontolások, mint a tűzfalak szintén részben megakadályozhatják, hogy egy számítógép csomagokat fogadjon, aláaknázva a harmadik feltételt. Mindezeknek a nehézségeknek az ellenére logikus lépés az ilyen számítógépeket is az interneten levőnek tekinteni, amíg az ISP-jükhöz csatlakoznak.

Futólag azt is érdemes érintenünk, hogy egyes vállalatok is összekapcsolták a már meglévő saját belső hálózataikat, és ezt sokan közülük az internetet megalapozó módszerek használatával tették meg. Ezeket az **intraneteket** általában csak a cég területén belülről vagy céges noteszgépekről lehet elérni, de ettől eltekintve ugyanúgy működnek, mint az internet.

1.5.2. Harmadik generációs mobiltelefon-hálózatok

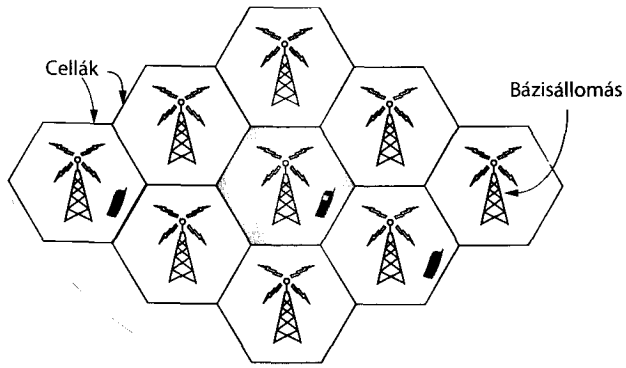
Az emberek még az internetezésnél is jobban szeretnek telefonon beszélgetni, és ez tette a mobiltelefon-hálózatot a legsikeresebb hálózattá a világon. Világszerte több mint négy milliárd előfizetőt tartanak számon. Hogy kellő megvilágításba helyezzük ezt a számot, ez hozzávetőleg a világ népességének 60%-a, és több mint az internetes hosztok és helyhez kötött telefonvonalak összesített száma [ITU, 2009].

A mobiltelefon-hálózat felépítése az óriási növekedéssel párhuzamosan nagyban megváltozott az elmúlt 40 évben. Az első generációs mobiltelefonos rendszerek a hanghívásokat még folyamatosan változó (analóg) jelként továbbították (digitális) bitek sorozata helyett. Az Egyesült Államokban 1982-ben telepített **AMPS (Advanced Mobile Phone System – fejlett mobiltelefon-rendszer)** széles körben használt első generációs rendszer volt. A második generációs mobiltelefon-rendszerek a hanghívások digitális továbbítására váltottak, hogy növeljék a kapacitást és a biztonságot, valamint lehetővé tegyék szöveges üzenetek küldését. A **GSM (Global System for Mobile Communications – globális mobilkommunikációs rendszer)**, melyet 1991-től kezdve telepítettek, és azóta a világon a legszélesebb körben használt mobiltelefonos rendszerré vált, egy második generációs (2G) rendszer.

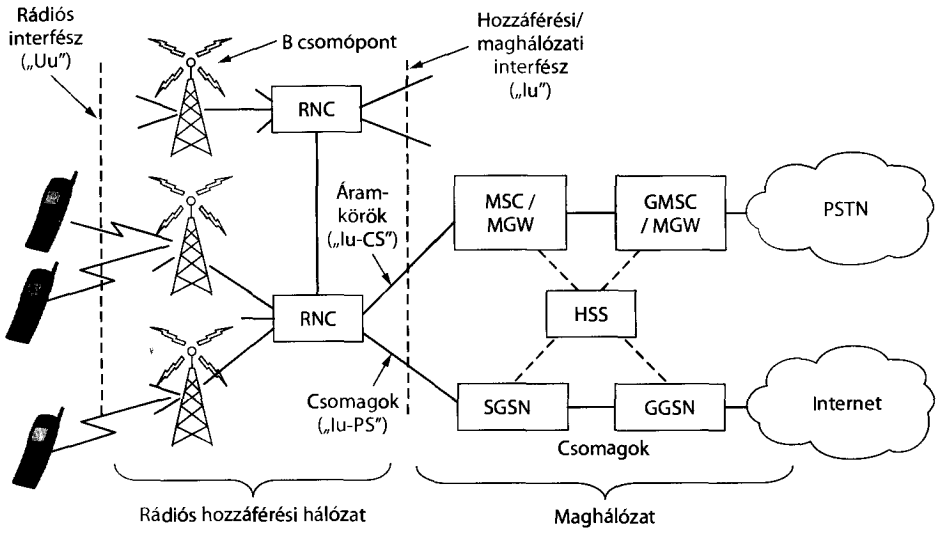
A harmadik generációs, vagyis 3G-s rendszereket először 2001-ben állították üzembe, és egyaránt kínálnak digitális hangszolgáltatást és széles sávú digitális adatszolgáltatást is. Rengeteg szakzsargonnal járnak, és sok különböző szabvány közül választhatunk. A 3G-t hozzávetőlegesen úgy határozta meg az ITU (egy nemzetközi szabványosító testület, melyet a következő szakaszban fogunk tárgyalni), hogy legalább 2 Mb/s sebességet biztosít álló felhasználóknak, és 384 kb/s sebességet mozgó járműveken. Az **UMTS (Universal Mobile Telecommunications System – univerzális mobiltávközlési rendszer)**, vagy más nevén **WCDMA (Wideband Code Division Multiple Access – széles sávú kódosztásos többszörös hozzáférésű rendszer)** a fő harmadik generációs rendszer, melyet világszerte telepítenek. Letöltési irányban akár 14 Mb/s sebességet is képes biztosítani, és közel 6 Mb/s sebességet a feltöltési irányban. A jövőbeli kiadások több antenna és rádióadó használatával nyújtanak majd ennél is nagyobb sebességet a felhasználóknak.

A 3G-rendszerek szűkös erőforrása, csakúgy, mint a korábbi 2G- és 1G-rendszerek esetében is, a rádiós spektrum. A kormányzatok engedélyezik a spektrum bizonyos részeinek használatát a mobiltelefon-hálózatok üzemeltetőinek, gyakran spektrumárverés keretében, melynek során a szolgáltatóknak licitálniuk kell. Az engedéllyel megszerzett spektrumszelet birtoklása egyszerűbbé teszi a rendszerek tervezését és működtetését, mivel senki más nem forgalmazhat az adott spektrumban, de ez jellemzően komoly összegekbe kerül. Például 2000-ben az Egyesült Királyságban öt darab 3G-licencet értékesítettek körülbelül 40 milliárd dollár összértékben.

A spektrum szűkössége az, ami az 1.30. ábrán látható **celluláris**, azaz sejtszerű hálózati elrendezéshez (**cellular network**) vezetett. A felhasználók közötti rádiós interferencia kezelésére a lefedettségi területet cellákra osztják. A cellán belül a felhasználók számára a hálózat olyan csatornákat oszt ki, amelyek nem zavarják egymást, és nem interferálnak túlságosan a szomszédos cellákkal sem. Ez teszi lehetővé a spektrum jó kihasználását, vagyis a **frekvencia újrahasznosítását** a szomszédos cellákban, ami növeli a hálózat kapacitását. Az első generációs rendszerekben, amelyek minden egyes hanghívást külön frekvenciasávban továbbítottak, a frekvenciát gondosan meg kellett választani, hogy ne ütközzön a szomszédos cellákkal. Így egy adott frekvencia számos cella közül mindig csak egyben volt használható. A modern 3G-rendszerekben minden frekvencia használható, de olyan módon, hogy a szomszédos cellákban okozott interferencia szintje



1.30. ábra. *A mobiltelefon-hálózatok cellás felépítése*



1.31. ábra. *Az UMTS 3G mobiltelefon-hálózat felépítése*

még elfogadható legyen. A cellás elvnek több változata is létezik, amilyen például a cella adótornyán elhelyezett irányított vagy szektorantennák használata az interferencia csökkentése végett, de az alapötlet ugyanaz.

A mobiltelefon-hálózat felépítése nagy mértékben különbözik az internet felépítésétől. Számos részből áll, mint az UTMS-hálózat felépítésének egyszerűsített változatát mutató 1.31. ábrán is látható. Először is ott van a **rádiós interfész (air interface)**. Ez a kifejezés a mobil eszköz (például telefonkészülék) és a **mobiltelefonos bázisállomás (cellular base station)** közötti rádiófrekvenciás kommunikációs protokoll speciális neve. A rádiós interfésznek az elmúlt évtizedekben történt fejlődése tette lehetővé a vezeték nélküli adatátviteli sebesség növelését. Az UMTS rádiós interfész a **kódosztásos többszörös hozzáférésen (Code Division Multiple Access, CDMA)** alapul, ezt a technikát a 2. fejezetben fogjuk tanulmányozni.

A mobiltelefonos bázisállomás és a vezérlője együtt alkotja a **rádiós hozzáférési hálózatot (radio access network)**. Ez a rész a mobiltelefon-hálózat vezeték nélküli oldala. A vezérlő csomópont vagy **RNC (Radio Network Controller – rádiós hálózatvezérlő)** irányítja a spektrum felhasználását. A bázisállomás valósítja meg a rádiós interfészt. Ennek a neve **Node B (B csomópont)**, egy ideiglenesnek szánt elnevezés, mely végül megmaradt.

A mobiltelefon-hálózat további részei szállítják tovább a rádiós hozzáférési hálózat forgalmát. Ezeknek az együttes neve **maghálózat (core network)**. Az UMTS-maghálózat az azt megelőző második generációs GSM-maghálózat továbbfejlődésével alakult ki. Viszont valami meglepő is történik az UMTS-maghálózatban.

A kezdetek óta dúl a háború a csomagkapcsolt hálózatok (azaz az összeköttetés nélküli alhálózatok) és az áramkörkapcsolt vagy vonalkapcsolt hálózatok (tehát az összeköttetés-alapú alhálózatok) támogatói között. A csomagkapcsolás legfőbb támogatói az internetes közösségből érkeznek. Az összeköttetés nélküli architektúrában minden

csomag az összes többitől függetlenül továbbítódik. Ennek következménye az, hogy ha néhány útválasztó kiesik egy munkamenet során, egészen addig nem történik baj, amíg a rendszer képes dinamikusan újrakonfigurálni magát, hogy a következő csomagok utat találjanak a célhoz, még ha az el is tér attól, amit a megelőző csomag bejárt.

Az áramkörkapcsolt tábor a telefontársaságok világából érkezik. A telefonrendszerben a hívónak fel kell hívnia a hívott fél telefonszámát, és meg kell várnia az összeköttetés felépülését, mielőtt beszélni kezdene vagy adatot küldene. Az összeköttetés felépítésekor egy útvonal alakul ki a hálózaton keresztül, amely a hívás végéig fennmarad. Minden szó vagy csomag ugyanazt az útvonalat követi. Ha egy vonal vagy kapcsoló berendezés kiesik az útvonalon, a hívás megszakad, ezáltal kevésbé hibatűrő, mint az összeköttetés nélküli architektúra.

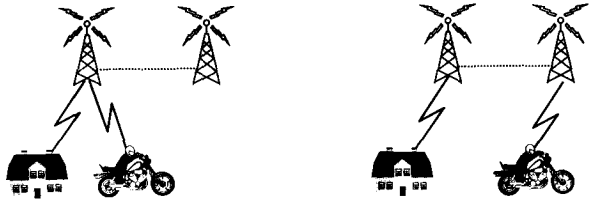
Az áramkörkapcsolás előnye, hogy egyszerűbb vele a szolgáltatás minőségét biztosítani. Az összeköttetés előre történő felépítésével az alhálózat lefoglalhatja az olyan szükséges erőforrásokat, mint amilyen az összeköttetések sávszélessége, a kapcsolók pufferterülete és CPU-ideje. Ha nem áll rendelkezésre elegendő erőforrás az összeköttetés létrehozására tett kísérletkor, a hálózat a hívást visszautasítja, és a hívó egyfajta foglaltság jelzést kap. Így ha egyszer felépült az összeköttetés, jó minőségű szolgáltatást tesz lehetővé.

Egy összeköttetés nélküli hálózatban ha túl sok csomag érkezik egy útválasztóhoz egy adott pillanatban, az útválasztó eldugul, és valószínűleg csomagokat fog veszteni. A küldő ezt előbb vagy utóbb észreveszi, és újraküldi a csomagokat, de a szolgáltatás minősége egyenetlen lesz, a hálózat pedig alkalmatlan hang vagy mozgókép továbbítására, hacsak a hálózat nem túlságosan terhelt. Mondanunk sem kell, hogy a megfelelő hangminőség elég fontos a telefontársaságok számára, ez magyarázza azt, hogy előnyben részesítik az áramkörkapcsolást.

A meglepetés az 1.31. ábrán az, hogy a maghálózatban egyaránt vannak csomagkapcsolt és áramkörkapcsolt berendezések. Ez azt jelzi, hogy a mobiltelefon-hálózat átmeneti időszakot él meg, és a mobiltársaságok akár egyik, akár mindkét lehetőséget megvalósíthatják. A régebbi mobiltelefon-hálózatok a hagyományos telefonhálózathoz hasonlóan áramkörkapcsolt maghálózattal rendelkeztek a hanghívások továbbításához. Ez az örökség az UMTS-hálózatban az **MSC (Mobile Switching Center – mobil-kapcsolóközpont)**, a **GMSC (Gateway Mobile Switching Center – átjáró mobil-kapcsolóközpont)** és az **MGW (Media Gateway – médiaátjáró)** képében van jelen, mely elemek olyan áramkörkapcsolt hálózatokon keresztül alakítanak ki összeköttetéseket, mint a **PSTN (Public Switched Telephone Network – nyilvános kapcsolt telefonhálózat)**.

Az adatszolgáltatás sokkal fontosabb tényezőjévé vált a mobiltelefon-hálózatoknak, mint korábban volt, kezdve a szöveges üzenetekkel és a korai csomagkapcsolt adatátviteli szolgáltatásokkal, mint amilyen a **GPRS (General Packet Radio System – általános csomagrádiós rendszer)** volt a GSM-rendszerben. A régi adatszolgáltatások néhányszor tíz kb/s nagyságrendű sebességet voltak képesek elérni, de a felhasználók ennél többre vágytak. Az újabb mobiltelefon-hálózatok a Mb/s többszörösével képesek szállítani az adatcsomagokat. Összehasonlítás végett, egy hanghívás 64 kb/s sávszélességgel továbbítódik, vagy tömörítéssel ennek jellemzően a harmadával-negyedével.

Ennek az adatforgalomnak a továbbítása végett az UMTS maghálózat csomópontjai közvetlenül egy csomagkapcsolt hálózathoz kapcsolódnak. Az **SGSN (Serving GPRS Support Node – kiszolgáló GPRS támogató csomópont)** és a **GGSN (Gateway GPRS**



1.32. ábra. *Mobiltelefonos átadás (a) előtt és (b) után*

Support Node – átjáró GPRS támogató csomópont) továbbítják az adatcsomagokat a mobilok és az olyan külső csomagkapcsolt hálózatok csatlakozópontjai között, mint például az internet.

Ez a szemléletváltás folytatódni fog a most tervezett vagy üzembe álló mobiltelefon-hálózatokkal. Az internetes protokollokat még a csomagkapcsolt hálózaton keresztüli mobiltelefonos hanghívások összeköttetésének felépítésére is használják, a VoIP formájában. Az IP és a csomagok a rádiós hozzáféréstől egészen a maghálózatig jelen vannak. Természetesen az IP-hálózatok felépítése is változóban van, hogy támogassák a jobb szolgáltatásminőséget. Ha nem így lenne, akkor az akadozó beszéd és a szakadozó videó nem tenne túl jó benyomást a fizető ügyfelekre. Az 5. fejezetben fogunk visszatérni erre a témakörre.

Egy további különbség a mobiltelefon-hálózatok és a hagyományos internet között a mobilitás kérdése. Amikor egy felhasználó áthalad egy mobiltelefonos bázisállomás hatósugarából egy másik bázisállomás hatósugarába, az adatok áramlását át kell irányítani a régi bázisállomásról az új bázisállomásra. Ezt a módszert **átadásnak** (**handover** vagy **handoff**) nevezik, és az 1.32. ábrán illusztrálja a működését.

A mobil készülék és a bázisállomás is kezdeményezheti az átadást, amikor a jel minősége leromlik. Néhány, jellemzően a CDMA-technikán alapuló cellás hálózatban lehetséges úgy kapcsolódni az új bázisállomáshoz, hogy még nem szakítottuk meg a kapcsolatot az előzővel. Ez javítja a kapcsolat minőségét a mobil számára, mert nincs szakadás a szolgáltatásban; a telefon ténylegesen két bázisállomáshoz kapcsolódik egy rövid ideig. Az átadásnak ezt a módját **puha átadásnak** (**soft handover**) nevezik, hogy megkülönböztessék a **kemény átadástól** (**hard handover**), mely során a mobiltelefon bontja a kapcsolatot a régi bázisállomással, mielőtt az újhoz csatlakozna.

Egy ehhez kapcsolódó probléma az, hogy egyáltalán hogyan találhatunk meg egy mobiltelefont bejövő hívás esetén. Minden mobiltelefon-hálózat rendelkezik egy **HSS**-sel (**Home Subscriver Server – otthoni előfizetőt kiszolgáló szerver**) a maghálózatában, ami számon tartja az összes előfizető helyzetét más, hitelesítésre és jogosultság-ellenőrzésre használt profiladatok társaságában. Így minden mobiltelefon megtalálható a HSS lekérdezésével.

Az utolsó érintendő terület a biztonság. A múltban a telefontársaságok sokkal komolyabban vették a biztonságot, mint az internetes cégek, mert számlázniuk kellett a szolgáltatásért, és el kellett kerülniük a (fizetési) visszaéléseket. Sajnos önmagában ez nem jelent túl sokat. Mindenesetre az 1G-től a 3G-ig vezető fejlődés során a mobiltelefon-társaságok elő tudtak állni néhány alapvető biztonsági eljárással a mobiltelefonok számára.

A 2G GSM-rendszer óta a mobiltelefon két részből áll: egy kézibeszélőből (telefonkészülékből) és egy kivehető chipből, mely az előfizetőt azonosítja és előfizetői információt tárol. A chip hétköznapi neve **SIM-kártya** (**SIM card**), mely a **Subscriber Identity Module** (**előfizető-azonosító modul**) rövidítése. A SIM-kártyák áttehetők más készülékekbe, ezáltal működőképessé téve azokat, és a SIM-kártyák jelentik a mobiltelefonos biztonság alapjait. Amikor egy GSM-felhasználó más országba utazik nyaralni vagy üzleti ügyben, gyakran magával viszi a készülékét, de érkezéskor néhány dollárért új SIM-kártyát vesz, hogy roamingdíjak nélkül kezdeményezhessen helyi hívásokat.

A visszaélések visszaszorítása céljából az SIM-kártyán tárolt információt használja a mobiltelefon-hálózat, hogy hitelesítse az előfizetőt, és ellenőrizze, hogy engedélyezett-e számára a hálózat használata. Az UMTS esetében a mobiltelefon is felhasználja a SIM-kártyán található információt annak ellenőrzésére, hogy engedélyezett (legitim) hálózattal kommunikál-e.

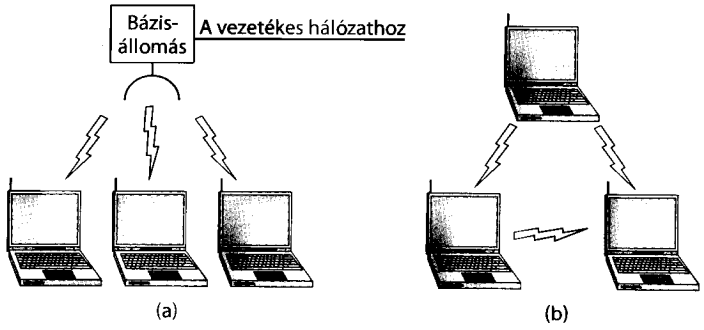
A biztonság másik szempontja a titkosság. A vezeték nélküli jelek minden, a közelben tartózkodó vevőhöz eljutnak, így ahhoz, hogy számukra ellehetetlenítsük a hallgatózást, a SIM-kártyán található titkosító kulcsokat használjuk az adás rejtjelezésére. Ez a megközelítés sokkal jobban szolgálja a titkosságot, mint az 1G-hálózatok idejében, melyeket könnyű volt megcsapolni, de még mindig nem csodaszer a titkosító módszerek gyengeségei miatt.

A mobiltelefon-hálózatokra központi szerep vár a jövő hálózatai között. Sokkal inkább szólnak már a mobil széles sávú alkalmazásokról, mint a hanghívásokról, és ez fontos következményekkel jár a jövő rádiós interfészei, maghálózat-architektúrái és biztonsága szempontjából. A gyorsabb és jobb 4G-technikák már tervezés alatt állnak **LTE** (**Long Term Evolution – hosszú távú fejlődés**) néven, miközben a 3G-hálózatok kialakítása és üzembe helyezése még folyamatban van. Más technikák is kínálnak széles sávú internet-hozzáférést rögzített pozíciójú és mozgó kliensek számára, mindenekelőtt a **WiMAX** név alatt futó 802.16 hálózatok. Teljesen elképzelhető, hogy az LTE és a WiMAX össze fognak ütközni, és nehéz megjósolni, hogy mi lesz a szembenállásuk végkimenetele.

1.5.3. Vezeték nélküli LAN-ok: 802.11

Már kevéssel a hordozható számítógépek megjelenése után is sokan álmodtak arról, hogy egyszer majd csak besétálnak az irodába, és a hordozható számítógépük valamilyen varázslatos módon azonnal felkapcsolódik az internetre. Amint ez az igény felmerült, több csoport is elkezdett módszereket kidolgozni a cél elérésére. A leggyakorlatiasabb megközelítés az, hogy mind az irodai számítógépeket, mind a noteszgépeket kis hatósugarú rádió-adóvevőkkel szereljük fel, így téve lehetővé köztük a kommunikációt.

Ez a munka hamar eredménnyel járt: megjelentek a vezeték nélküli LAN-ok, amelyeket sok cég forgalmazott. A baj az volt, hogy esélyünk sem volt két, egymással kompatibilis LAN-t találni köztük. A megoldások számának ilyen mértékű növekedése azt eredményezte, hogy egy *X* márkájú rádióval épített számítógépet nem lehetett működésre bírni egy *Y* márkájú bázisállomással felszerelt szobában. Végül az 1990-es évek közepén az iparág úgy döntött, hogy jó ötlet lenne kidolgozni egy vezeték nélküli LAN-szabványt, ezért megbízták az IEEE azon bizottságát, amelyik a vezetékes LAN-okat is szabványosította, hogy dolgozzon ki egy szabványt a vezeték nélküli LAN-okra.



1.33. ábra. *(a) Vezeték nélküli hálózat bázisállomással. (b) Ad hoc hálózat*

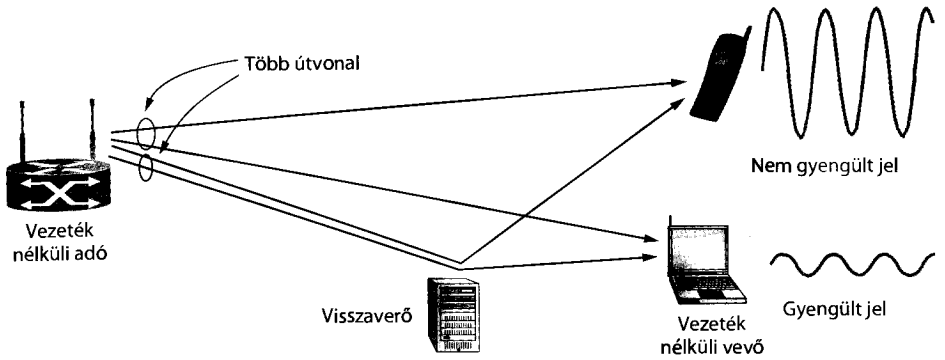
Az első kérdés volt a legkönnyebb: mi legyen a neve? Az összes többi LAN-szabvány számozva volt 802.1-től 802.10-ig, így a vezeték nélküli szabványt 802.11-nek nevezték el. Ennek az egyik gyakori beceneve a **Wi-Fi**,[11] de mivel ez egy fontos szabvány, és megérdemli a tiszteletet, a rendes nevén, 802.11-nek fogjuk nevezni.

A többi már nehezebbnek bizonyult. Az első probléma az volt, hogy találni kellett egy alkalmas frekvenciasávot, amely – lehetőleg világszerte – használható. A használt megközelítés a mobiltelefon-hálózatok esetében látottak ellenkezője volt. A drága, engedélyköteles spektrum helyett a 802.11 rendszerek olyan szabadon felhasználható sávokban működnek, mint az ITU-R által kijelölt **ISM** (**Industrial, Scientific and Medical – ipari, tudományos és orvosi**) sávok (például 902–928 MHz; 2,4–2,5 GHz; 5,725–5,825 GHz). Ezt a spektrumot bármilyen eszköz használhatja, ha betartja azt a szabályt, hogy úgy korlátozza az adó teljesítményét, hogy más eszközök is működhessenek a környezetében. Természetesen ez azt is jelenti, hogy a 802.11 rádióknak esetleg vetélkedniük kell a vezeték nélküli telefonokkal (cordless phones), garázsajtó-nyitókkal és mikrohullámú sütőkkel.

A 802.11 hálózatok alkotóelemei a kliensek, például noteszgépek és mobiltelefonok, valamint az épületben elhelyezett infrastruktúra, mely **AP**-kból (**Access Point – hozzáférési pont**) áll. A hozzáférési pontokat gyakran nevezik **bázisállomásnak** (**base station**) is. A hozzáférési pont a vezetékes hálózathoz csatlakozik, és a kliensek közötti összes kommunikáció egy-egy hozzáférési ponton keresztül zajlik. Az is lehetséges, hogy két, egymás hatósugarában levő kliens közvetlenül kommunikáljon egymással, például két számítógép egy hozzáférési pont nélküli irodában. Ezt a felállást alkalmi vagy **ad hoc hálózatnak** (**ad hoc network**) nevezzük. Sokkal ritkábban használt, mint a hozzáférési pont által vezérelt mód. Mindkét működési mód látható az 1.33. ábrán.

A 802.11 adatátvitelt nehezíti, hogy a vezeték nélküli átviteli feltételek a környezet legkisebb változásával is módosulhatnak. A 802.11 által használt frekvenciákon a rádiójeleket a szilárd testek visszaverhetik, így a jelek (több útvonal mentén) többször is megérkezhetnek a vevőhöz. A visszhangok kioltathatják vagy felerősíthetik egymást, a vett jel nagyfokú ingadozását okozva. Az így létrejövő interferencia jelenség az úgynevezett **többutas terjedés miatti jelgyengülés** vagy **féding** (**multipath fading**), mely az 1.34. ábrán figyelhető meg.

11 Hivatalosan Wi-Fi a neve, de sok helyen WiFi, Wifi, wifi névvel is illetik. A betűszó a Hi-Fi mintájára készült szójáték. *(A lektor megjegyzése)*



1.34. ábra. *Jelgyengülés többutas terjedés esetén*

A kulcsötlet ahhoz, hogy úrrá legyünk a vezeték nélküli átvitel változó körülményei fölött, az **útvonal-diverzitás** (**path diversity**), vagyis az információ több, független útvonalon történő elküldése. Ily módon az információ még abban az esetben is valószínűsíthetően megérkezik, ha az egyik útvonal gyenge a féding miatt. A független útvonalak kezelése jellemzően a digitális modulációs eljárásba épül be a fizikai rétegben. A lehetőségek között megtaláljuk az engedélyezett sávon belüli különböző frekvenciák használatát, az eltérő antennapárok közötti eltérő útvonalak meglétét, vagy a bitsorozatok különböző ideig tartó ismétlését.

A 802.11 különböző verziói felhasználták ezeket a módszereket. A kezdeti (1997-es) szabvány olyan vezeték nélküli LAN-t definiált, mely azáltal tudott elérni 1 vagy 2 Mb/s sebességet, hogy a különböző frekvenciák között ugrásokat végzett vagy szétszórta a jelet a teljes engedélyezett spektrumban. Az emberek szinte azonnal panaszkodni kezdtek, hogy túl lassú, így további munka kezdődött a gyorsabb változatok kidolgozására. A szórt spektrumú működést kiegészítették, és ebből lett az (1999-es) 802.11b szabvány, mely akár 11 Mb/s sebességre is képes. A 802.11a (1999) és a 802.11g (2003) szabványok egy másik, **OFDM**-nek (**Orthogonal Frequency Division Multiplexing – ortogonális frekvenciaosztásos multiplexelés**) nevezett modulációs eljárásra váltottak. Ez a spektrum egy széles sávját sok kis szeletre bontja fel, melyeken keresztül párhuzamosan küldi az egyes biteket. Ez a javított működési mód, melyet a 2. fejezetben tanulmányozunk majd, egészen 54 Mb/s sebességig tornázta fel a 802.11a/g hálózatok sebességét. Ez jelentős növekedés, de az emberek még ennél is nagyobb átviteli teljesítményre mutattak igényt. A legújabb verzió a 802.11n (2009). Szélesebb frekvenciasávokat használ, valamint számítógépenként legfeljebb négy antennát, hogy akár 450 Mb/s sebességet érjen el.

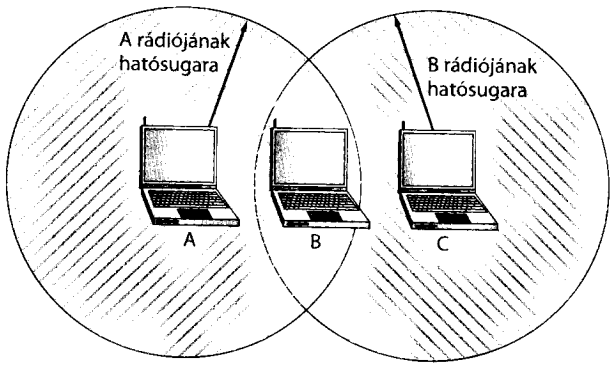
Mivel a vezeték nélküli technológiák természetüknél fogva adatszórásos átviteli közeggel dolgoznak, a 802.11-es rádióknak is meg kell küzdeniük azzal a problémával, hogy az egy időben küldött adások ütközni fognak, ami zavart okozhat a vételben. Ennek a problémának a kiküszöbölésére a 802.11 a **CSMA**- (**Carrier Sense Multiple Access – vivőjel-érzékeléses többszörös hozzáférés**) módszert használja, mely a klasszikus vezetékes Ethernetből vesz ötleteket, amelyet – ironikus módon – egy Hawaii-on fejlesztett,

ALOHA-nak nevezett korai vezeték nélküli hálózatból merített. Az adás megkezdése előtt a számítógép mindig vár egy rövid, véletlenszerű hosszúságú ideig, és elhalasztja az adást, ha azt hallja, hogy valaki más már éppen ad. Ezzel a módszerrel sokkal kisebb a valószínűsége annak, hogy két számítógép egyszerre adjon. Mindazonáltal ez mégsem működik annyira jól, mint a vezetékes hálózatok esetében. Hogy lássuk az okát, nézzük meg az 1.35. ábrát. Tegyük fel, hogy az *A* számítógép a *B* számítógépnek ad, de *A* rádiójának hatótávolsága túl kicsi ahhoz, hogy a *C* számítógépet elérje. Ha *C B*-nek akar adni, belehallgathat az éterbe az adás megkezdése előtt, de az a tény, hogy nem hall semmit, még nem jelenti azt, hogy az adás sikeres is lesz. Mivel *C* nem hallhatja *A*-t az adás megkezdése előtt, ütközés fog történni. Minden ütközés után a küldő egyre hosszabb, de továbbra is véletlenszerű hosszúságú időt vár, majd újraküldi a csomagot. Ennek és más nehézségek ellenére a gyakorlatban elegendően jól működik ez a módszer.

A következő probléma a mobilitás. Valamilyenfajta átadás-átvételre van szükség, amikor egy hordozható számítógépet átvisznek az általa éppen használt bázisállomás hatókörzetéből egy másik bázisállomás hatókörzetébe. A megoldás az, hogy a 802.11 hálózat több cellából épül fel, melyek közül mindegyik rendelkezik egy-egy bázisállomással, valamint egy olyan elosztórendszerből, ami összeköti a cellákat. A elosztórendszer gyakran kapcsolt Ethernet, de bármilyen más technika is lehet. A kliens a mozgása közben találhat olyan hozzáférési pontot, melynek erősebb a jele, mint amit éppen használ, és megváltoztathatja a társításukat. Kívülről az egész rendszer egyetlen vezetékes LAN-nak tűnik.

Mindezek tudatában a mobilitás a 802.11 hálózatokban még csak korlátozott értékkel bír a mobiltelefon-hálózatokban megvalósított mobilitással összehasonlítva. Jellemzően a 802.11-et ritkán mozgó (ún. nomád) kliensek használják, melyek egyik rögzített helyről egy másikra mennek ahelyett, hogy mozgás közben használnák azokat a felhasználóik. A mobilitásra nincs igazán szükség ebben a vándorló felhasználási módban. Amikor pedig ténylegesen kihasználjuk a 802.11 mobilitását, általában csak egyetlen 802.11 hálózaton belül tesszük azt, ami akár egy nagyobb épületet is lefedhet. A jövőbeli módszereknek eltérő hálózatok és technikák (például 802.21) között is biztosítaniuk kell majd a mobilitást.

Végezetül lássuk a biztonság kérdését. Mivel a vezeték nélküli adás adatszórással történik, a közelben levő számítógépek könnyen megkapják a nem nekik szánt csomagokat. Ennek megelőzésére a 802.11 szabvány egy titkosító eljárást is tartalmazott, melyet **WEP**



1.35. ábra. *Az egyes rádiók hatósugara nem feltétlenül fedi le a teljes rendszert*

(**Wired Equivalent Privacy – vezetékessel egyenértékű titkosság**) néven ismerhetünk. A cél az volt, hogy a vezeték nélküli biztonsági szint összemérhető legyen a vezetékessel. Az ötlet dicsérendő, de sajnálatos módon a titkosító eljárás hibás volt, és hamarosan fel is törték [Borisov és mások, 2001]. Azóta újabb, eltérő kriptográfiai módszereket alkalmazó algoritmusok váltották fel a 802.11i szabvány formájában, melyet **Wi-Fi Protected Access** (**Wi-Fi védett hozzáférés**) névvel is illetnek (kezdetben ez a **WPA** volt, de mára a **WPA2** leváltotta).

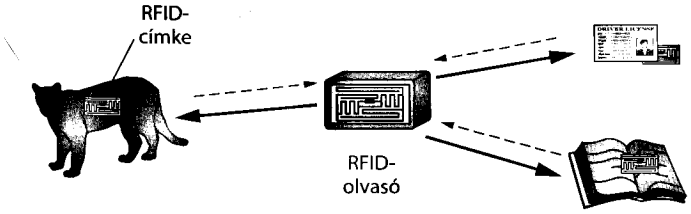
A 802.11 forradalmat okozott a vezeték nélküli hálózatok terén, mely a jövőben is folytatódni fog. Az épületeken túlmenően gyors ütemben telepítik ezeket a hálózatokat vonatokon, repülőkön, hajókon és gépjárművekben, hogy az emberek bárhol használhassák a világhálót, amerre járnak. A mobiltelefonok és mindenféle fogyasztói elektronikus eszköz, kezdve a játékkonzoloktól a digitális fényképezőgépekig, képes kommunikálni ezzel a technikával. Részletesen visszatérünk majd rá a 4. fejezetben.

1.5.4. Az RFID és a szenzorhálózatok

Az eddig tanulmányozott hálózatok mind olyan számítást végző egységekből állnak, amiket könnyű felismerni, például számítógépekből és mobiltelefonokból. A **rádiófrekvenciás azonosítással** (**Radio Frequency IDentification – RFID**) a mindennapi tárgyak is részévé válhatnak egy számítógép-hálózatnak.

Egy RFID-címke (RFID tag) úgy néz ki, mint egy postai bélyeg méretű matrica, ami felragasztható (vagy beépíthető) egy tárgyra, hogy azt követni lehessen. Az objektum lehet egy tehén, egy útlevél, egy könyv vagy egy raklap is. A címke egy egyedi azonosítóval ellátott kis mikrochipből és egy antennából áll, mely a rádiófrekvenciás jeleket veszi. A követési pontokon telepített RFID-olvasók megtalálják a címkéket, amikor azok az olvasók hatósugarába érnek, és lekérdezik a bennük tárolt információt, ahogy az 1.36. ábrán is látható. Az alkalmazások között megtalálhatjuk a valakinek vagy valaminek az azonosságellenőrzését, az ellátási lánc kezelését, időmérést versenyeken, valamint a vonalkódok felváltását.

Több típusa is van az RFID-nak, és mindegyik különböző tulajdonságokkal rendelkezik, de talán a leginkább lenyűgöző szempont az, hogy a legtöbb RFID-címkének nincs szüksége sem elektromos csatlakozóra, sem akkumulátorra. Ehelyett a működésükhöz szükséges összes energiát az RFID-olvasóból jövő rádióhullámok formájában kapják meg. Ezt a technikát **passzív RFID**-nak nevezzük, hogy megkülönböztessük a (kevésbé gyakori) **aktív RFID**-tól, amelynél a címkének van saját áramforrása.



1.36. ábra. *Az RFID hálózatba köti a mindennapi objektumokat*

Az RFID gyakori formája az **UHF RFID** (**Ultra-High Frequency RFID – ultranagyfrekvenciás** vagy **ultrarövidhullámú RFID**). Raklapokon és bizonyos jogosítványokon használják. Az olvasók az Egyesült Államokban a 902–928 MHz-es sávban adnak. A címkék több méterről is képesek kommunikálni azáltal, hogy megváltoztatják az olvasó jelének visszaverődését, és az olvasó venni tudja ezt a visszavert jelet. Ezt a működési módot **visszaverődésnek** (**backscatter**) nevezik.

Egy másik népszerű típus a **HF RFID** (**High Frequency RFID – nagyfrekvenciás** vagy **rövidhullámú RFID**). Ez 13,56 MHz-en működik, és valószínűleg ezt találhatjuk meg egy útlevélben, hitelkártyában, könyveken és érintésmentes fizetési rendszerekben. A HF RFID hatótávolsága rövid, jellemzően legfeljebb egy méter, mert a fizikai elve az *indukción* alapul a visszaverődés helyett. Léteznek még további RFID-típusok is, melyek más frekvenciákat használnak, például az **LF RFID** (**Low Frequency RFID – kisfrekvenciás** vagy **hosszúhullámú RFID**), amelyet még a HF RFID előtt fejlesztettek ki, és állatok nyomon követésére használtak. Valószínűleg ilyen típusú RFID-t találnánk meg a macskánkban.

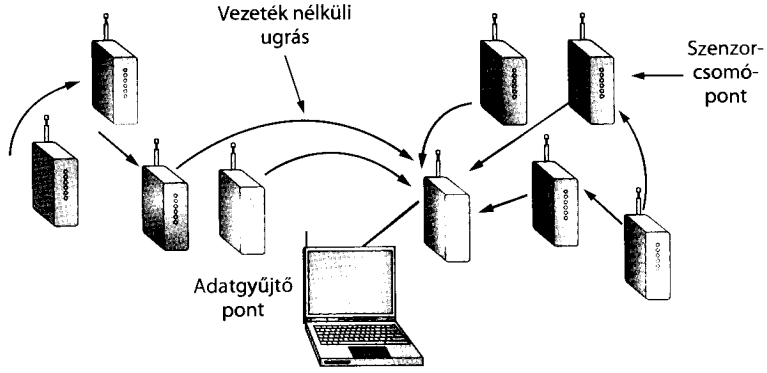
Az RFID-olvasóknak meg kell birkózniuk azzal a helyzettel is, ha több címke kerül a hatókörzetükbe. Ez azt jelenti, hogy egy címke nem válaszolhat csak úgy, ha meghall egy olvasót, máskülönben a több címkéről érkező jelek ütközhetnének. A megoldás hasonló a 802.11 esetében választott megközelítéshez: a címkék rövid, véletlenszerű ideig várnak, mielőtt válaszolnának az azonosítójukkal, ami által az olvasó leszűkítheti a kört az egyes címkékre, majd tovább kérdezheti őket.

A biztonság is probléma. Az olvasók azon képessége, hogy könnyen kövessenek egy tárgyat, és így az azt használó embert, a magánszféra megsértésével járhat. Sajnos nehéz biztonságossá tenni az RFID-címkéket, mert nincs meg bennük az erős kriptográfiai algoritmusok futtatásához szükséges számítási és kommunikációs teljesítmény. Ehelyett gyenge biztonsági intézkedéseket (például jelszavakat) használnak. Ha egy személyi igazolványt távolról leolvashat egy határőr, mi akadályoz meg bárkit is abban, hogy az igazolvány gazdáját az ő tudta nélkül kövesse? Nem sok.

Az RFID-címkék azonosító chipként kezdték, de gyorsan teljes értékű számítógépekké váltak. Például sok címke rendelkezik felülírható és később lekérdezhető memóriával, hogy a felcímkézett objektummal történt eseményekről lehessen információt tárolni. Rieback és mások [2006] demonstrálták, hogy ez annyit tesz, hogy az RFID-chipek is ki vannak téve a szokásos számítógépes kártevőknek, de ezúttal csak a macskánk vagy az útlevelünk terjesztheti az RFID-vírust.

A képességek terén előrelépést jelentenek az RFID-hoz képest a **szenzorhálózatok** (**sensor network**). A szenzorhálózatokat azért telepítik, hogy a fizikai világ bizonyos paramétereit monitorozzák. Eddig nagyrészt csak tudományos kísérletezéshez használták ezeket, például madarak élőhelyének, vulkanikus tevékenységeknek, zebrák vándorlásának a megfigyelésére, de az üzleti felhasználásukra sem kell már sokat várnunk: használhatók lesznek az egészségügyben, berendezések rázkódásának felügyeléséhez vagy fagyasztott, hűtött, illetve romlandó termékek útjának követésére.

A szenzorcsomópontok kis számítógépek, gyakran mindössze kulcstartónyi méretben, melyek érzékelőkkel rendelkeznek a hőmérséklet, a rezgés és más környezeti tulajdonságok mérésére. Sok csomópontot helyezünk el a vizsgálandó környezetbe. Jellemzően saját tápforrással rendelkeznek, de energiához juthatnak a napból vagy a mozgásukból is. Mint az RFID esetében, az elégséges energia biztosítása itt is kulcskérdés, és a csomópontoknak nagy odafigyeléssel kell kommunikálniuk ahhoz, hogy az érzékelőikkel rögzített információt el tudják juttatni egy külső gyűjtőpontba. A megszokott stratégia az, hogy a csomópontok önszerveződő hálózatot alkotnak, és továbbítják egymás üzeneteit, ahogy az 1.37. ábra is mutatja. Ezt a fajta rendszert **többugrásos hálózatnak** (**multihop network**) nevezik.



1.37. ábra. *Egy szenzorhálózat többugrásos topológiája*

Az RFID és a szenzorhálózatok a jövőben várhatóan sokkal több képességgel lesznek felruházva, és sokkal több helyen lesznek jelen. Kutatók máris egyesítették a két technika előnyeit olyan programozható RFID-címkék formájában, melyek fény-, mozgás- és más érzékelőkkel rendelkeznek.

1.6. A hálózatok szabványosítása

Számos hálózatépítéssel és hálózatüzemeltetéssel foglalkozó cég létezik, és mindegyiknek saját elképzelése van arról, hogy hogyan kell a dolgokat csinálni. Koordináció nélkül teljes káosz uralkodna, és a felhasználók nem tudnák elérni a céljaikat. Az egyetlen kiút a hálózatok szabványosítása. A jó szabványok nemcsak a különböző számítógépek közötti kommunikációt teszik lehetővé, hanem bővítik a szabványnak megfelelő termékek piacát is. A nagyobb piac végül tömegtermeléshez, gazdaságosabb gyártáshoz, jobb implementációk megjelenéséhez vezet, és további olyan előnyökkel jár, amelyek az árakat csökkentik, a szabvány befogadását pedig növelik.

A következő alfejezetekben rövid bepillantást nyerünk a nemzetközi szabványosítás fontos, de alig ismert világába. Lássuk először, hogy mi tartozik egy szabványba. Egy értelmesen gondolkozó ember úgy vélhetné, hogy egy szabvány azt mondja meg, hogy hogyan kell működnie egy protokollnak, hogy az implementációja könnyen elvégezhető legyen. Ez az ember nagy tévedésben lenne.

A szabványok azt definiálják, ami az együttműködéshez (interoperability) kell: se többet, se kevesebbet. Ez lehetővé teszi a teljes piac felemelkedését, miközben lehetőséget ad a gyártóknak, hogy versenyezzenek egymással a termékeik minőségében. Például a

802.11 szabvány több átviteli sebességet is meghatároz, de nem mondja ki, hogy egy adónak mikor melyiket kell használnia, ami pedig kulcsfontosságú a jó teljesítmény eléréséhez. Ez annak a döntése lesz, aki a terméket készíti. A különböző termékek együttműködését elérni így azonban gyakran nehéz feladat, mivel sok döntéshelyzet adódik a megvalósítás során, és a szabványok is sok lehetőséget határoznak meg. A 802.11 esetében annyi probléma merült fel, hogy egy később általános gyakorlattá váló stratégia szerint megalakult a **Wi-Fi Alliance** (**Wi-Fi Szövetség**) nevű ipari csoportosulás, hogy a 802.11 szabvány együttműködési kérdéseivel foglalkozzon.

Ehhez hasonlóan a protokollokat leíró szabványok is csak a kommunikáló felek közötti üzeneteket definiálják, a gépen belüli szolgáltatási interfészeket nem, kivéve, ha ez a protokoll magyarázatát szolgálja. A valós szolgáltatási interfészek gyakran nem is nyíltak. Például az a mód, ahogy a TCP kapcsolódik az IP-hez a számítógépen belül, egyáltalán nem számít, ha egy távoli hoszttal kívánunk kommunikálni. Csak az számít, hogy a távoli hoszt is beszélje a TCP/IP-t. Valójában a TCP-t és az IP-t rendszerint együtt valósítják meg, bármilyen kifejezett interfész nélkül. Mindezek ellenére a jó szolgáltatási interfészek, mint a jó API-k, nagyon fontosak ahhoz, hogy ténylegesen használják is a protokollokat, és a legjobbak (mint a Berkeley-csatlakozók) nagyon népszerűvé válhatnak.

A szabványoknak két kategóriája van: a *de facto* és a *de jure*. A **de facto** (latinul „tényleges") szabványok azok a szabványok, amelyek hivatalos tervezés nélkül maguktól alakultak ki. A HTTP, a webet működtető protokoll de facto szabványként kezdte meg az életét. A korai webböngészők része volt, melyeket Tim Berners-Lee fejlesztett ki a CERN-ben, és a protokoll használata a web növekedésével indult be igazán. Egy másik példa a Bluetooth. Eredetileg az Ericsson fejlesztette ki, de ma mindenki ezt használja.

A **de jure** (latinul „törvényes") szabványok ezzel szemben olyan hivatalos szabványok, amelyeket bizonyos szabványosítási szervezetek elfogadtak. A nemzetközi szabványosítási szervezeteket általában két nagy csoportra oszthatjuk. Az egyik csoportba azok tartoznak, amelyek államközi szerződések útján jöttek létre, a másikba pedig az önkéntesen, nem szerződéses alapon létrehozott szervezetek. A számítógép-hálózatok szabványosításának világában mindkét típusú szervezetből van jó néhány, a legfontosabbak az ITU (International Telecommunication Union), az ISO (International Standards Organization), az IETF (Internet Engineering Task Force – Internet Működtetését Koordináló Testület) és az IEEE (Institut of Electrical and Electronics Engineers). A továbbiakban ezekről lesz szó.

A gyakorlatban nagyon bonyolult a viszony a szabványok, a vállalatok és a szabványosítási szervezetek között. A de facto szabványok gyakran de jure szabvánnyá fejlődnek, különösképpen, ha sikeresek. Ez történt a HTTP esetében, amit gyorsan felkarolt az IETF. A szabványosítási testületek gyakran ratifikálják egymás szabványait, ami úgy tűnhet, mintha egymás vállát veregetnék, hogy növeljék egy technológia piacát. Manapság sok, adott technikák körül kialakuló alkalmi üzleti szövetség játszik jelentős szerepet a hálózati szabványok kifejlesztésében és finomításában. Például a **3GPP** (**Third Generation Partnership Project – harmadik generációs társulási projekt**) távközlési szervezetek olyan együttműködése, amely az UMTS 3G mobiltelefonos szabványokat irányítja.

1.6.1. Ki kicsoda a hírtávközlés világában?

A világ telefontársaságainak jogi helyzete igencsak eltérő az egyes országokban. Az egyik véglet az Egyesült Államok, ahol 2000-nél is több önálló (de legtöbbször nagyon kisméretű), magántulajdonban levő telefontársaság működik. Néhány az AT&T 1984-es feldarabolásával jött létre (ami akkor a világ legnagyobb vállalata volt, és az amerikai telefonok 80%-át ez a cég szolgálta ki), illetve az 1996-os távközlési törvénnyel (Telecommunications Act), mely teljesen újraszabályozta a piacot a verseny előmozdítása érdekében.

A másik végletet azok az országok jelentik, amelyek kormánya a távközlés területén monopóliummal rendelkezik, és egyedül uralja a levélküldést, a távírást, a távbeszélőrendszert, és sokszor a rádiót, valamint a televíziót is. A világ legtöbb országa ebbe a kategóriába esik. Vannak olyan esetek, amikor a távközlési felügyelőség egy külön állami cég, de van, amikor egyszerűen csak egy kormányszerv, amit rendszerint **PTT-nek** (**Post, Telegraph & Telephone administration**) hívnak. A fejlődés világszerte a liberalizáció és a szabadpiaci verseny irányába halad, nem pedig az állami monopólium felé.

Tekintettel az előbb említett különböző szolgáltatókra, nyilvánvalóan szükség van világméretű kompatibilitásra annak érdekében, hogy a különböző országokban élő emberek (illetve számítógépek) kapcsolatba kerülhessenek egymással. Ez az igény tulajdonképpen már régóta létezik. 1865-ben számos európai ország képviselői megalakították a mai **ITU-T** (**International Telecommunication Union – Nemzetközi Távközlési Egyesülés**) elődjét, a **CCITT-t** (**Comité Consultatif International Téléphonique et Télégraphique – Nemzetközi Távíró és Telefon (technikai) Konzultatív Bizottság**). A CCITT feladata az volt, hogy szabványosítsa a nemzetközi hírtávközlést, amely akkoriban még csak a távírást jelentette. Már akkor is nyilvánvaló volt azonban, hogy problémához fog vezetni az, ha az országok egyik fele a Morse-kódot használja, a másik fele pedig valamilyen más kódot. Amikor a telefon nemzetközi szintű szolgáltatássá vált, a CCITT magára vállalta a telefonrendszerek szabványosítását is. 1947-ben a CCITT az ENSZ egyik ügynöksége lett. Az ITU-T története 1960-ban kezdődött, de a mai nevét és formáját csak 1992-ben nyerte el.

Az ITU-T körülbelül 200 kormányt tart nyilván a tagjai között, köztük az ENSZ szinte valamennyi tagja szerepel. Mivel az Egyesült Államok nem rendelkezik PTT-vel, valaki másnak kellett elvállalnia a képviseletét az ITU-T-ben. A választás a külügyminisztériumra esett, amit azzal indokolhattak, hogy az ITU-T nemzetközi ügyekkel foglalkozik, ami a külügyminisztérium szakterülete. Az ITU-T-nek 700-nál is több szekciója és társult tagja van. Vannak köztük telefontársaságok (például AT&T, Vodafone, Sprint), telekommunikációs eszközgyártók (például Cisco, Nokia, Nortel), számítástechnikai gyártók (például Microsoft, Agilent, Toshiba), chipgyártók (például Intel, Motorola, TI), médiavállalatok (például AOL Time Warner, CBS, Sony) és más érdeklődő vállalatok is (például Boeing, CBS, VeriSign).

Az ITU-nak az ITU-T-n kívül még két fő ágazata van. Mi elsősorban az **ITU-T**-re (**Telecommunications Standardization Sector – Távközlési Szabványosítási Ágazat**) fogunk koncentrálni, amely a távbeszélő- és adatátviteli rendszerekkel foglalkozik. Az **ITU-R** (**Radiocommunications Sector – Rádiókommunikációs Ágazat**) a rádiófrekvenciák kiosztását koordinálja a világszerte egymással versengő érdekcsoportok között. A harmadik ágazat az **ITU-D** (**Development Sector – Fejlesztési Ágazat**) az információs és kommunikációs technikák fejlesztését támogatja, hogy szűkítse a „digitális

szakadékot" az információs technikákhoz ténylegesen hozzáférő, és azokat csak korlátozottan elérő országok között.

Az ITU-T feladata az, hogy műszaki javaslatokat tegyen a telefonok, a távírók és az adatkommunikáció interfészeire. Ezek gyakran válnak nemzetközileg elfogadott szabványokká, de a szó szoros értelmében az ITU-T ajánlásai csak műszaki javaslatokat tartalmaznak, melyeket egy ország a kedve szerint vagy befogadhat, vagy nem (mert a kormányok olyanok, mint a 13 éves kisfiúk: nem veszik jó néven, ha parancsolgatnak nekik). Gyakorlatilag bármelyik ország szabadon kiépíthet egy olyan távbeszélőrendszert, amely a többi országétól különbözik, azonban ebben az esetben számolnia kell azzal, hogy elvágja magát a külvilágtól. Ez működhet például Észak-Koreában, de más országokban komoly problémát jelentene.

Az ITU-T-ben a tényleges munkát a tanulmányi csoportok (Study Group) végzik. Jelenleg 10 tanulmányi csoport van, amelyek gyakran akár 400 emberből is állnak, és az általuk lefedett témák a telefonbeszélgetések számlázásától a multimédiás szolgáltatásokig és a biztonságig terjednek. Például az SG 15 szabványosítja az internethez való kapcsolódásra használt DSL technikát. A tényleges munkavégzést az teszi lehetővé, hogy a tanulmányi csoportokat munkacsoportokra (Working Party) osztják, amelyeket tovább bontanak szakértői csapatokra (Expert Team), és még ezeket is tovább osztanak ad hoc csoportokra. Ami bürokráciának indul, az mindig az is marad.

Mindezek ellenére az ITU-T ténylegesen is véghezvisz dolgokat. Megalakulása óta 3000-en felüli ajánlást készített, amelyek közül sokat a gyakorlatban is használnak. Például a H.264 ajánlást (amely egyben ISO-szabvány is MPEG-4 AVC néven) széles körben használják videotömörítésre, és az X.509 nyilvános kulcsú tanúsítványokat használják a webböngészés biztonságossá tételére és digitálisan aláírt elektronikus levelek küldésére.

Ahogyan teljessé válik a hírtávközlés 1980-as években elkezdődött átmenete a teljesen belföldi szolgáltatástól a teljesen globális szolgáltatásig, és a szabványok egyre fontosabbakká válnak, egyre több és több szervezet akar majd részt venni kialakításukban. Az ITU-val kapcsolatos további információkat lásd Irmer [1994] dolgozatában.

1.6.2. Ki kicsoda a nemzetközi szabványok világában?

A nemzetközi szabványokat az **ISO** (**International Organization for Standardization – Nemzetközi Szabványügyi Szervezet**) adja ki, amely egy 1946-ban alakult önkéntes, nem államközi szerződéseken alapuló szervezet. Az ISO tagságát 157 tagállam nemzeti szabványügyi szervezete alkotja. A tagok között megtalálható az ANSI (Egyesült Államok), a BSI (Nagy-Britannia), az AFNOR (Franciaország), a DIN (Németország) és még további 153 szervezet.

Az ISO a legkülönbözőbb témákban ad ki szabványokat, a csavaroktól és a csavaranyáktól (szó szerint) kezdve a telefonpóznák bevonatáig mindent ideértve, és akkor még nem említettük az olyan szabványokat, mint a kakaóbabé (ISO 2451), a halászhálóké (ISO 1530) vagy a női alsóneműké (ISO 4416), illetve azt a rengeteg további témát, melyekről senki sem gondolná, hogy szabványosítva lennének. A távközlési szabványok kérdésében az ISO és az ITU-T gyakran együttműködik (az ISO tagja az ITU-T-nek),

Szám	Téma
802.1	A LAN-ok áttekintése és felépítése
802.2 ↓	Logikai kapcsolatvezérlés
802.3 *	Ethernet
802.4 ↓	Vezérjeles sín (rövid ideig használták termelőüzemekben)
802.5	Vezérjeles gyűrű (az IBM LAN-megoldása)
802.6 ↓	Kettőzött várakozási soros kettőzött sín (Dual Queue Dual Bus – DQDB; korai nagyvárosi hálózat)
802.7 ↓	Széles sávú megoldásokkal foglalkozó műszaki tanácsadó csoport
802.8 †	Fényvezetőszálas megoldásokkal foglalkozó műszaki tanácsadó csoport
802.9 ↓	Izokron (isochronous) LAN-ok (valós idejű alkalmazásokhoz)
802.10 ↓	Virtuális LAN-ok és biztonság
802.11 *	Vezeték nélküli LAN-ok (Wi-Fi)
802.12 ↓	Igények prioritásai (a Hewlett-Packard AnyLAN-ja)
802.13	Szerencsétlen szám, senkinek sem kellett
802.14 ↓	Kábelmodemek (elavult, mivel egy ipari egyesülés előbb végzett a fejlesztéssel)
802.15 *	Személyi hálózatok (Bluetooth, Zigbee)
802.16 *	Széles sávú vezeték nélküli hálózatok (WiMAX)
802.17	Ellenálló csomaggyűrű (resilient packet ring)
802.18	Rádiós szabályozási kérdésekkel foglalkozó műszaki tanácsadó csoport
802.19	A felsorolt szabványok együttélésével foglalkozó műszaki tanácsadó csoport
802.20	Mobil széles sávú vezeték nélküli hálózatok (hasonló a 802.16e-hez)
802.21	Médiafüggetlen átadás (technikák közötti roaming támogatására)
802.22	Vezeték nélküli regionális hálózatok

1.38. ábra. *A 802 munkacsoportjai. A fontosakat *-gal jelöltük. A ↓-lal jelölt munkacsoportokat befagyasztották. A †-tel jelölt már megszüntette magát*

hogy elkerüljék azt a kínos helyzetet, hogy két hivatalos, egymással kölcsönösen nem kompatibilis nemzetközi szabványt dolgoznak ki.

Több mint 17 000 szabványt adtak eddig ki, beleértve az OSI-szabványokat is. Az ISO-nak 200-nál is több TC-je (Technical Committee – Műszaki Bizottság) van, amelyeket a megalakulásuk sorrendjében számoztak be. Ezek közül mindegyik külön szakterülettel foglalkozik. A TC1 bizottság például a csavarokkal és a csavaranyákkal foglalkozik, és a csavarok menetemelkedését szabványosítja. A JTC1 foglalkozik az információs technológiákkal, többek között hálózatokkal, számítógépekkel és szoftverekkel. Ez az első (és eddig

egyetlen) JTC (Joint Technical Committee – Egyesített Műszaki Bizottság), melyet 1987-ben hoztak létre a TC7-ben és az IEC-ben (egy további szabványügyi szervezet) folyó tevékenységek egybeolvasztásával. A Műszaki Bizottságok albizottságokra (subcommittee, SC), azok pedig munkacsoportokra (working group, WG) vannak felosztva.

Az igazi munkát világszerte több mint 100 000 önkéntes végzi a munkacsoportokban. Ezek az „önkéntesek" legtöbbször olyan cégek megbízásából dolgoznak egy ISO-anyagon, amelyeknek a termékei éppen szabványosítás alatt állnak. Mások kormányhivatalnokként azon fáradoznak, hogy egy országukban elfogadott szabvány nemzetközi szabvánnyá váljon. Sok munkacsoportban egyetemi szakemberek is dolgoznak.

A szabványok elfogadása az ISO-ban mindig a lehető legszélesebb körű egyetértésen alapul. A szabványosítási folyamat úgy indul, hogy valamelyik ország szabványügyi szerve egy adott szakterületen nemzetközi szabványosítást lát szükségesnek. Ilyenkor megalakul egy új munkacsoport, amelynek feladata egy **bizottsági javaslat (Committee Draft, CD)** kidolgozása. A bizottsági javaslatot körbeadják a különböző tagszervezeteknek, amelyeknek hat hónap áll a rendelkezésére, hogy véleményezzék azt. Ha a nagy többség jónak találja, akkor egy átdolgozott dokumentumot, egy ún. **nemzetközi szabványtervezetet (Draft International Standard, DIS)** kell elkészíteni, amelyet ismét körbeadnak véleményezésre és szavazásra. Ennek a fordulónak az eredménye alapján elkészítik a **nemzetközi szabványt (International Standard, IS)**, amelyet aztán jóváhagynak és kiadnak. Nagy viták esetén a bizottsági javaslat és a nemzetközi szabványtervezet számos változtatáson mehet keresztül, mire végre megszavazzák, és emiatt az egész folyamat akár évekig is elhúzódhat.

A **NIST (National Institute of Standards and Technology – Nemzeti Szabványügyi és Technológiai Intézet)** az Egyesült Államok Kereskedelmi Minisztériumának hivatala. Ez korábban NBS (National Bureau of Standards – Nemzeti Szabványügyi Hivatal) néven volt ismert. Ez a szervezet olyan szabványokat ad ki, amelyek az Egyesült Államok kormányának beszerzéseinél kötelező érvényűek. Ez alól csak a Hadügyminisztérium kivétel, amelynek saját szabványai vannak.

A szabványosítás világának egy másik fontos szereplője az **IEEE (Institute of Electrical and Electronics Engineers – Villamos- és Elektronikai Mérnökök Intézete)**, amely a világ legnagyobb szakmai szervezete. Azonkívül, hogy rengeteg folyóiratot ad ki, és több száz konferenciát rendez meg évente, IEEE szabványokat is dolgoz ki a villamosmérnöki tudományok és az informatika területén. Az IEEE 802-es bizottsága sok LAN-fajtát szabványosított, ezeknek egy részét meg is fogjuk vizsgálni a könyv további részében. A tényleges munkát az 1.38. ábrán felsorolt munkacsoportok végzik. A sikeres munkacsoportok aránya a megalakulás óta alacsony; egy 802.x szám kiosztása még nem garantálja a sikert is. De a sikeres történeteknek (főként a 802.3 és a 802.11) iparra és a világra gyakorolt hatása óriási volt.

1.6.3. Ki kicsoda az internetszabványok világában?

Az egész világot átfogó internet is rendelkezik saját szabványosítási rendszerrel, amely nagyon különbözik az ITU-T és az ISO rendszerétől. A különbséget durva közelítéssel úgy foglalhatnánk össze, hogy az ITU és az ISO szabványosítási konferenciáira öltönyben érkeznek a résztvevők, míg az internet szabványosítási konferenciáin a résztvevők farmert viselnek (kivéve a San Diegóban megrendezett találkozókat, amikor rövidnadrágot és pólót).

Az ITU-T és az ISO értekezletein olyan vállalati ügyintézők és közalkalmazottak ülnek, akiknek a szabványosítás a munkájuk. A szabványosítást jó dolognak tartják, és ennek szentelik életüket. Ezzel szemben az internetes szakemberek alapvetően az anarchiát részesítik előnyben, bár néha egyetértésre is szükségük van ahhoz, hogy a dolgok előre haladjanak. Így akármilyen fájdalmas is, időnként szükség van szabványokra. Az MIT-n dolgozó David Clark egyszer a következő híressé vált megjegyzést tette az internet szabványosítására: „döcögő egyetértés és futó programok".

Amikor az ARPANET-et kiépítették, a DoD létrehozott egy informális bizottságot a hálózat felügyeletére. 1983-ban ezt a bizottságot átnevezték **IAB**-ra **(Internet Activities Board – Internet Koordinációs Testületnek)**, és kissé kibővítették a hatáskörét is. Az lett a feladata, hogy az ARPANET és az Internet kutatóit többé-kevésbé ugyanabba az irányba terelje, akárcsak egy jó pásztor a nyájat. Az „IAB" betűszó jelentése később módosult, és ma az internet felépítését felügyelő testületet **(Internet Architecture Board – Internet Architektúra Testületet)** jelöli.

Az IAB körülbelül 10 tagja közül mindegyik vezetett valamilyen fontos témával foglalkozó munkacsoportot. Az IAB évente többször összeült, hogy megtárgyalja az eredményeket, és hogy visszacsatolást adjon a DoD-nek és az NSF-nek, amelyek akkoriban a támogatás legnagyobb részét adták. Amikor egy új szabványra volt szükség (például egy új útválasztó algoritmusra), az IAB tagjai alaposan átbeszélték a dolgot, és utána bejelentették a változtatást. Az akkoriban a szoftverfejlesztés legfontosabb embereinek számító egyetemistáknak ezután már csak meg kellett valósítaniuk a szükséges változtatásokat. A kommunikáció alapja egy jelentéssorozat volt, amelyeknek **RFC (Request for Comments – megjegyzések bekérése)** volt a neve. Az RFC-ket online tárolják, és így bármely érdeklődő lekérheti őket a www.ietf.org/rfc címről. A jelentéseket meg is számozták a megjelenésük sorrendje szerint. Mára már több mint 5000 ilyen jelentés készült, közülük sokra ebben a könyvben is hivatkozni fogunk.

1989-re az internet olyan méreteket öltött, hogy ez a nagyfokú informális stílus tovább már nem állta meg a helyét. Akkoriban jó néhány forgalmazó kínált TCP/IP-termékeket, és nem akartak változtatni azokon csak azért, mert tíz kutatónak jobb ötlete támadt. 1989 nyarán az IAB-t újból átszervezték. A kutatókat az **IRTF (Internet Research Task Force – Internetkutatásokat Koordináló Testület)** szervezetbe tömörítették, amely a mérnököket összefogó **IETF (Internet Engineering Task Force – Internet Működtetését Koordináló Testület)** szervezettel együtt az IAB részlege lett. Az IAB tagságát kibővítették, és a kutatócsoportok szakemberein kívül más szervezetek képviselői is helyet kaptak benne. Kezdetben az újjászervezett IAB egy állandóan megújuló csoport volt, amiben egy képviselő 2 évig dolgozhatott, és az új képviselőket a régiek javaslata alapján nevezték ki. Később aztán megalakult az **Internet Society (Internet Társaság)**, amelyet az internet iránt érdeklődő szakemberek hoztak létre. Az Internet Society hasonló szerepet tölt be, mint az ACM és az IEEE. Választott tisztségviselők irányítják, és azok jelölik ki az IAB-képviselőket.

Az IAB kettéválasztásának célja az volt, hogy az IRTF-ben a hosszú távú kutatási célokra, míg az IETF-ben a rövid távú mérnöki kutatási célokra összpontosítsanak. Az IETF-et további munkacsoportokra osztották fel, és mindegyiknek saját feladatokat ad-

tak. A munkacsoportok elnökeiből álló vezetőtestület eleinte sokszor összeült, hogy a mérnöki fejlesztéseket irányítsa. A munkacsoportok többek közt új alkalmazásokkal, felhasználói információkkal, OSI-integrációval, útválasztással és címzéssel, adatbiztonsággal, hálózatmenedzsmenttel és szabványokkal foglalkoztak. Időnként olyan sok munkacsoport dolgozott (néha több mint 70), hogy külön szakterületek jöttek létre, és csak a szakterületek elnökei ültek össze a vezetői értekezleteken.

Az eddigieken kívül az ISO mintájára kialakult egy sokkal formálisabb szabványosítási folyamat is. Ahhoz, hogy valamiből **szabványtervezet (Proposed Standard)** legyen, az alapelképzelést nagyon világosan el kell magyarázni egy RFC-ben, és szakmai berkekben kellő érdeklődésnek kell lennie iránta. Ez biztosítja azt, hogy csak alaposan átgondolt javaslatokkal álljanak elő. Ahhoz, hogy a szabványtervezetből **előzetes szabvány (Draft Standard)** legyen, 4 hónapig legalább két különböző helyen alapos tesztelésnek kell alávetni egy működő implementációt. Ha az IAB meggyőződött arról, hogy az elképzelés jó, és a szoftver működik, akkor az elképzelést ismertető RFC-t **Internetszabványnak (Internet Standard)** nyilvánítja. Néhány internetszabvány DoD-szabvány (MIL-STD) is lett, ami által kötelező érvényűvé vált a DoD beszállítói számára.

A webes szabványokhoz a **World Wide Web Konzorcium (World Wide Web Consortium – W3C)** fejleszt ajánlásokat és irányelveket, hogy előmozdítsa a világháló hosszú távú növekedését. Ez egy ipari konzorcium, melyet Tim Berners-Lee vezet, és 1994-ben hozták létre, amikor a web igazán elkezdett beindulni. A W3C-nek jelenleg több mint 300 tagja van világszerte, és 100-nál is több W3C-ajánlást állított elő (így hívják a szabványaikat), melyek olyan területeket fednek le, mint a HTML és a webes magánszféra védelme.

1.7. Mértékegységek

A keveredések elkerülése érdekében érdemes kijelentenünk, hogy ebben a könyvben a számítástechnika általános szokásainak megfelelően az SI mértékegységeket használjuk. A legfontosabb SI-előtagokat az 1.39. ábrán soroltuk fel. Az előtagokat általában az első betűjükkel jelölik, az egynél nagyobb egységeket nagybetűvel (KB, MB stb.). Egyetlen kivétel ez alól (történelmi okokból) a kb/s a kilobit/másodperc jelölésére. Egy 1 Mb/s-os kommunikációs vonal 10^6 bitet továbbít másodpercenként és egy 100 psec-os (avagy 100 ps-os) óra 10^{-10} másodpercenként üt egyet. Mivel a milli és a mikro egyaránt „m" betűvel kezdődik, ezért választani kellett közülük. Általában az „m" a millit jelöli és a „μ" (a kis görög mű betű) jelöli a mikrót.

Azt is érdemes megjegyezni, hogy az általános mérnöki szóhasználatban a memória-, tárterület-, állomány- és adatbázisméretek esetében a mértékegységeknek ettől kissé eltérő jelentése van. Itt a kilo 2^{10}-t (1024) jelent, nem 10^3-t (1000), mivel a memóriaméretek mindig 2 hatványai. Így egy 1 KB-os memória nem 1000 bájtot tartalmaz, hanem 1024 bájtot. Jegyezzük meg, hogy a nagy „B" betű ebben a használatban bájtot jelent (8 bitet), szemben a kis „b" betűvel, ami bitet jelent. Hasonlóképpen egy 1 MB-os memória 2^{20} (1 048 576) bájtot, 1 GB memória 2^{30} (1 073 741 824) bájtot, egy 1 TB-os adatbázis pedig 2^{40} (1 099 511 627 776) bájtot tartalmaz. Mindazonáltal egy 1 kb/s-os átviteli vonal 1000 bitet visz át egy másodperc alatt, és egy 10 Mb/s-os LAN 10 000 000 b/s-os sebességgel üzemel, mivel ezek a sebességek nem 2-hatványok. Sajnos sokan keverik ezt a két rendszert, különösen a lemezméretek esetében. A kétértelműség elkerülése érdekében ebben a könyvben mindenhol a KB, MB, GB és TB jelek rendre 2^{10}, 2^{20}, 2^{30} és 2^{40} bájtot jelöl, a kb/s, Mb/s, Gb/s és Tb/s pedig rendre 10^3, 10^6, 10^9 és 10^{12} b/s-ot jelöl.

Hatványkitevős alak	Számmal kiírva	Előtag[12]	Jelölés
10^{-3}	0,001	milli	m
10^{-6}	0,000 001	mikro	μ
10^{-9}	0,000 000 001	nano	n
10^{-12}	0,000 000 000 001	piko	p
10^{-15}	0,000 000 000 000 001	femto	f
10^{-18}	0,000 000 000 000 000 001	atto	a
10^{-21}	0,000 000 000 000 000 000 001	zepto	z
10^{-24}	0,000 000 000 000 000 000 000 001	yokto	y
10^{3}	1 000	kilo	K vagy k
10^{6}	1 000 000	mega	M
10^{9}	1 000 000 000	giga	G
10^{12}	1 000 000 000 000	tera	T
10^{15}	1 000 000 000 000 000	peta	P
10^{18}	1 000 000 000 000 000 000	exa	E
10^{21}	1 000 000 000 000 000 000 000	zetta	Z
10^{24}	1 000 000 000 000 000 000 000 000	yotta	Y

1.39. ábra. *A legfontosabb SI-előtagok*

1.8. Röviden a továbbiakról

Ez a könyv a számítógép-hálózatok elméletét és gyakorlatát egyaránt tárgyalja. A legtöbb fejezet a vonatkozó elvek tárgyalásával kezdődik, amelyet néhány, azokat bemutató példa követ. Ezek a példák általában az internet és a vezeték nélküli hálózatok témaköréből kerülnek ki, mivel ezek mind fontosak, és nagyon különbözők. Más példákat is mutatunk, ahol azok jobban szemléltetik az elméletet.

12 Írott szövegben az előtag mindig kisbetűs, függetlenül attól, hogy egynél kisebb vagy egynél nagyobb mértékről van szó. *(A lektor megjegyzése)*

A könyv elrendezése az 1.23. ábrán megadott hibrid modell szerkezetét követi. A 2. fejezetben a legalsó rétegtől indulunk el, a továbbiakban pedig egyre feljebb és feljebb haladunk a protokollhierarchiában. A második fejezet az adatkommunikáció területéről nyújt háttér-információt. Lefedi a vezetékes és a vezeték nélküli átviteli rendszereket. Ez az anyagrész a fizikai réteggel foglalkozik, de csak a felépítését tárgyaljuk, a hardverrel nem nagyon foglalkozunk. Megtárgyalunk számos olyan, a fizikai réteggel kapcsolatos példát is, mint a nyilvános kapcsolt telefonhálózat, a mobiltelefon- vagy a kábeltévéhálózat.

A 3. és 4. fejezet két részben tárgyalja az adatkapcsolati réteget. A 3. fejezet azzal foglalkozik, hogy hogyan lehetséges átküldeni egy csomagot egy adatkapcsolaton, ideértve a hibafelismerést és a hibajavítást is. Megvizsgáljuk a DSL-t (amit széles sávú internethozzáférésre használnak telefonvonalon keresztül), mint az adatkapcsolati protokollok valós életből vett példáját.

A 4. fejezetben a közeg-hozzáférési alréteget vizsgáljuk meg. Az adatkapcsolati rétegnek ez a része azzal foglalkozik, hogy hogyan használhat több számítógép egy közös átviteli csatornát. A megtekintett példák között ott lesznek olyan vezeték nélküli hálózatok, mint a 802.11 és az RFID, és olyan vezetékes LAN-ok is, mint a klasszikus Ethernet. Ebben a fejezetben lesz szó azokról az adatkapcsolati rétegbeli kapcsolókról is, amelyek a LAN-okat kötik össze.

Az 5. fejezet a hálózati réteggel foglalkozik, főképpen az útválasztással, aminek kapcsán sok statikus és dinamikus útválasztó algoritmust is bemutatunk. Azokban a hálózatokban is előfordulhat, amelyeknél jó az útválasztás, hogy amikor a hálózatban nagyobb forgalomra van igény, mint amit az kezelni tud, némely csomag késni fog vagy elveszik. Ezért azt is megtárgyaljuk, hogy a torlódásokat hogyan kell kezelni, hogy garanciát vállalhassunk egy bizonyos szolgáltatásminőségre. A különböző hálózatok összekapcsolása internetté számtalan problémához vezethet, ezekről szintén ebben a fejezetben esik szó. Az internet hálózati rétegét nagy részletességgel vizsgáljuk meg.

A 6. fejezet a szállítási réteggel foglalkozik. A hangsúlyt főként az összeköttetés-alapú protokollokra és a megbízhatóságra helyezi, mivel ezeket sok alkalmazáshoz használják. Az internet mindkét szállítási protokollját, az UDP-t és a TCP-t, valamint azok teljesítőképességét részletesen vizsgáljuk.

A 7. fejezet az alkalmazási réteg protokolljaival és alkalmazásaival foglalkozik. Az első téma a DNS, az internet telefonkönyve. Ezután az e-levelezés következik, amelynek a protokolljait is megtárgyaljuk. A világhálóval folytatjuk a témák sorát, szó esik a statikus és a dinamikus tartalomról, valamint arról, hogy mi történik a kliens, illetve a szerver oldalán. Ezt követően a hálózati multimédiát vizsgáljuk meg, ezen belül például a folyamszerű hangátvitelt és a hálózati videózást. Végezetül a tartalomszolgáltató hálózatokról ejtünk szót, ideértve a P2P-hálózatok technikáit is.

A 8. fejezet a hálózatok biztonságáról szól. Ez a téma valamilyen vonatkozásban minden réteggel kapcsolatban áll. Ebből kifolyólag úgy a legegyszerűbb, ha az összes többi réteg alapos megtárgyalása után kerítünk rá sort. A fejezet elején a titkosítás tudományába vezetjük be az olvasót, később pedig megmutatjuk, hogyan használható a titkosítás a kommunikáció, az e-levelezés és a web biztosítására. A könyv végén olyan területekről esik szó, ahol a biztonság összeütközésbe kerül a személyes adatok védelmével, a szólásszabadsággal, a cenzúrával vagy más társadalmi jelenségekkel.

A 9. fejezet egy jegyzetekkel ellátott listát tartalmaz az ajánlott olvasmányokról, fejezetek szerint csoportokba rendezve. Ezt azoknak szántuk, akiket mélyebben is érdekel a hálózatok témája. Ebben a fejezetben kapott helyett egy ábécébe rendezett irodalomjegyzék is, amelyben minden, a könyvben hivatkozott mű szerepel.

A szerzők weblapja a Pearsonnál: *http://www.pearsonhighered.com/tanenbaum*. Ez egy olyan oldal, ahol számos hiperhivatkozás található egy sor kapcsolódó témára, így oktatási segédanyagok, „gyakran ismételt kérdések" oldala, vállalatok, ipari tömörülések, szakmai szervezetek, szabványosítási szervezetek, műszaki megoldások, cikkek és más anyagok érhetők el.

1.9. Összefoglalás

A számítógép-hálózatok számos szolgáltatást nyújtanak cégek és magánszemélyek, otthoni és úton levő felhasználók számára egyaránt. A cégek szemszögéből gyakran a személyi számítógépek megosztott szervereket használó hálózatai biztosítják a céges adatok elérhetőségét. Ezek általában a kliens–szerver-modellt követik: az alkalmazottak munkaállomásai a kliensek, amelyek a gépteremben elhelyezett nagy teljesítményű szervereket érhetik el. Az egyének számára a hálózatok egy sor információt és szórakozási lehetőséget tesznek elérhetővé, valamint egy újfajta adásvételi módot a termékek és szolgáltatások forgalmazására. A magánszemélyek az otthonukban jellemzően a telefon- vagy kábeltévé-szolgáltatójukon keresztül kapcsolódnak az internetre, de egyre gyakrabban alkalmaznak vezeték nélküli hozzáférést is noteszgépek és telefonok esetében. A technikai fejlődés új típusú mobil alkalmazások létrehozását, valamint háztartási gépekbe és más fogyasztói eszközökbe beágyazott számítógépekből hálózatok kialakítását teszi lehetővé. Ugyanez a fejlődés társadalmi kérdéseket is felvet, például a magánszférával kapcsolatban.

A hálózatokat első közelítésben LAN-okra, MAN-okra, WAN-okra és internetekre bonthatjuk. A LAN-ok egy épületet fednek le, és nagy sebességre képesek. A MAN-ok egy várost fednek le, mint például a kábeltévé-rendszer, amelyet ma már sokan használnak internet-hozzáférésre is. A WAN-ok egy országot vagy egy kontinenst szolgálnak ki. A hálózatok felépítésére használt technikák egy része kétpontos (point-to-point) jellegű (például egy vezeték), míg mások adatszórást (broadcast) használnak (például a vezeték nélküli megoldások). A hálózatokat összekapcsolhatjuk útválasztókkal, hogy interneteket alakítsunk ki belőlük, melyek legnagyobb és legismertebb példája a nagybetűs internet. Az olyan vezeték nélküli hálózatok, mint amilyen például a 802.11 LAN-ok és a 3G mobiltelefon-hálózat rendkívül népszerűvé kezdenek válni.

A hálózati szoftver a protokollok köré épül, amelyek a folyamatok kommunikációjának szabályait határozzák meg. A legtöbb hálózat támogatja a protokollhierarchiák használatát. Ezekben minden réteg szolgáltatásokat nyújt a felette elhelyezkedőnek, és elrejti azok elől az alacsonyabb szintű rétegekben használt protokollok részleteit. A protokollkészletek általában az OSI-modellen vagy a TCP/IP-modellen alapulnak. Mindkettőben van adatkapcsolati, hálózati, szállítási és alkalmazási réteg, de a többi rétegben különböznek. A tervezési szempontok között szerepel a megbízhatóság, az erőforrás-

hozzárendelés, a skálázhatóság, a biztonság és más témák. A könyv nagy része a protokollokkal és azok tervezésével foglalkozik.

A hálózatok szolgáltatásokat kínálnak felhasználóiknak. A szolgáltatások köre az összeköttetés nélküli, a lehetőségekhez mérten a legjobb kézbesítést megkísérlő (best effort) csomagtovábbítástól az összeköttetés-alapú, garanciákkal alátámasztott továbbításig terjed. Egyes hálózatokban az is előfordulhat, hogy valamelyik réteg összeköttetés nélküli szolgáltatást nyújt, de a felette levő réteg összeköttetés-alapú szolgáltatást biztosít.

A közismert hálózatok között megtalálható az internet, a 3G mobiltelefon-hálózat és az IEEE 802.11-es vezeték nélküli LAN. Az internet az ARPANET-ből fejlődött ki, amelyből később más hálózatok hozzákapcsolásával internetet alakítottak ki. A mai internet tulajdonképpen sok ezer hálózat összessége, melyek mind a TCP/IP-protokollkészletet használják. A harmadik generációs mobiltelefon-hálózat vezeték nélküli és mobil internet-hozzáférést nyújt a felhasználóinak több Mb/s-os sebességgel, valamint természetesen a hanghívásokat is továbbítja. Az IEEE 802.11 szabványon alapuló vezeték nélküli LAN-ok sok otthonban és kávézóban vannak jelen, és 100 Mb/s fölötti adatkapcsolati sebességet is képesek nyújtani. Az új típusú hálózatok is felemelkedőben vannak, például a beágyazott szenzorhálózatok és az RFID-technikán alapuló hálózatok.

Ahhoz, hogy több számítógép beszélgetni tudjon egymással, nagyarányú szabványosításra van szükség, mind a hardver, mind a szoftver terén. Az olyan szervezetek, mint az ITU-T, az ISO, az IEEE és az IAB felügyelik a szabványosítási folyamat különböző részeit.

1.10. Feladatok

1. Tegyük fel, hogy van egy bernáthegyi kutyája, Bundás, amelyet arra képzett ki, hogy konyakos hordó helyett egy dobozt vigyen a nyakában, amelyben három 8 mm-es kazettát helyezett el. (Amikor az embernek megtelik a merevlemeze, az vészhelyzetnek tekinthető.) Minden egyes kazetta 7 gigabájt kapacitású. A kutya 18 km/h-s sebességgel odamehet magához, bárhol is tartózkodik éppen. Milyen távolságtartományban van Bundásnak nagyobb adatátviteli sebessége, mint egy 150 Mb/s-os vonalnak (adminisztrációs többlet nélkül)? Hogyan változik a válasz, ha (i) Bundás sebességét megduplázzuk, (ii) minden kazetta kapacitását megduplázzuk, (iii) az átviteli vonal adatsebességét megduplázzuk?

2. A lokális hálózatok egyik lehetséges alternatívája egy olyan nagyméretű időosztásos rendszer, amelyben minden felhasználónak van egy terminálja. Ismertessük egy lokális hálózatot használó kliens–szerver-rendszer két előnyös tulajdonságát!

3. Egy kliens–szerver-rendszer teljesítménye két hálózati tényezőtől függ: a hálózat sávszélességétől (hány bitet tud átvinni másodpercenként) és a késleltetésétől (hány másodpercig tart, amíg az első bit eljut a szervertől a kliensig). Adjon példát olyan hálózatra, amelynek mind a sávszélessége, mind a késleltetése nagy! Ezután mondjon egy olyan példát is, ahol a sávszélesség és a késleltetés egyaránt kicsi!

4. A sávszélességen és a késleltetésen kívül még milyen paramétert kell megadnunk ahhoz, hogy jól jellemezzük a szolgáltatásminőséget egy olyan hálózatban, amelyet (i) digitális beszédtovábbításra, (ii) mozgókép átvitelére, (iii) pénzügyi tranzakciók továbbítására használunk?

5. A tárol-és-továbbít típusú csomagkapcsolt rendszerek késleltetésének egyik befolyásoló tényezője az, hogy mennyi ideig tart, amíg egy kapcsoló gép tárol és továbbít egy csomagot. Ha a kapcsolási idő 10 μsec, akkor lehet-e ez fontos tényező egy olyan kliens–szerver-rendszerben, ahol a kliens New Yorkban, a szerver pedig Kaliforniában van? Feltételezhető, hogy a jelek terjedési sebessége a rézvezetékben és a fényvezető szálban egyaránt a fény vákuumban mért sebességének 2/3-a.

6. Egy kliens–szerver-rendszer olyan műholdas hálózatot használ, amelyben a műhold 40 000 km-es magasságban van. Mennyi egy kérésre adott válasz késleltetése a lehető legjobb esetben?

7. A jövőben, amikor már mindenkinek az otthonában lesz egy olyan terminál, amely a számítógép-hálózathoz csatlakozik, lehetővé válnak az azonnali népszavazások egy fontos törvényjavaslattal kapcsolatban. Végül is a parlamentet fel lehetne oszlatni, hiszen mindenki közvetlenül is ki tudná nyilvánítani az akaratát. Egy ilyen közvetlen módon gyakorolt demokráciának az előnyei mindenki számára nyilvánvalók. Vitassuk meg a hátrányait is!

8. Egy két pont közötti összeköttetéseken alapuló alhálózatban öt útválasztót kell összekapcsolni. A tervező az egyes útválasztók közé nagy, közepes vagy kis sebességű vonalakat tehet, de arra is van lehetősége, hogy ne kösse össze azokat. Ha egy számítógépnek 100 ms-ra van szüksége ahhoz, hogy előállítson és megvizsgáljon egy adott topológiát, akkor mennyi ideig tart az összes végigvizsgálása?

9. Az adatszóró alhálózatok nagy hátránya, hogy elpazarolják a sávszélesség egy részét, amikor egyszerre több hoszt is ugyanahhoz a csatornához akar hozzáférni. Az egyszerűség kedvéért tegyük fel, hogy a rendelkezésre álló időt diszkrét időszeletekre osztjuk fel, és ugyanazt az időszeletet mind az n hoszt p valószínűséggel akarja egyszerre használni. Az időszeletek mekkora hányadát pazaroljuk el az ütközések miatt?

10. Mondjunk két okot arra, hogy miért érdemes protokollrétegeket használni! Mondjunk egy okot a hátrányára is!

11. A Különleges Festők nevű cég igazgatójának az az ötlete támad, hogy a helyi sörfőzdével együttműködve kifejleszthetnének egy láthatatlan sörösdobozt (környezetvédelmi okokból). Az igazgató megkéri a jogi osztályt, hogy tanulmányozza a tervet, a jogi osztály pedig továbbítja azt a mérnöki csoporthoz. Ezt követően a főmérnök felhívja telefonon a másik céget, hogy megbeszéljék a műszaki részleteket. A mérnökök az eredményről tájékoztatják a saját jogi osztályaikat, amelyek ezután telefonon tisztázzák egymás közt a jogi kérdéseket. Végül a két cég igazgatója megvitatja

egymással a terv pénzügyi vonatkozásait. Az OSI-modell értelmében a többrétegű protokollok milyen elveit sérti meg ez a kommunikációs példa?

12. Vegyünk két olyan hálózatot, amelyek megbízható, összeköttetés-alapú szolgáltatást nyújtanak. Az egyik egy megbízható bájtfolyam szolgáltatást, a másik pedig egy megbízható üzenetfolyam szolgáltatást nyújt. Ugyanarról van szó a két esetben? Amennyiben igen, miért teszünk mégis különbséget közöttük? Ha pedig nem, akkor adjon példát olyan tulajdonságra, amiben különböznek!

13. Mit jelent a hálózati protokollok témakörében az „egyezkedés" (negotiation)? Adjon rá példát!

14. Az 1.19. ábra egy szolgáltatást mutat. Vannak az ábrán implicit módon szereplő szolgáltatások is? Ha igen, hol? Ha nem, miért nem?

15. Egyes hálózatokban az adatkapcsolati réteg kezeli az átviteli hibákat, a megrongálódott keretek újraküldését kérve. Amennyiben egy csomag megrongálódásának a valószínűsége p, hányszor kell egy csomagot átlagosan elküldeni ahhoz, hogy megérkezzen? Feltételezhetjük, hogy a nyugták sohasem vesznek el.

16. Adott egy rendszer, aminek n rétegű protokollhierarchiája van. Az alkalmazások M bájt hosszúságú üzeneteket állítanak elő. Minden rétegben egy h bájt hosszúságú fejrész adódik az üzenethez. Mekkora hányadát foglalják le a hálózat sávszélességének a fejrészek?

17. Mi a leglényegesebb különbség a TCP és az UDP között?

18. Az 1.25.(b) ábra alhálózatát arra tervezték, hogy egy atomháborút is túléljen. Hány bombára lenne szükség ahhoz, hogy a csomópontok halmaza két, egymástól független halmazra essen szét? Feltételezzük azt, hogy egy bomba egy csomópontot és az összes hozzá kapcsolódó vonalat megsemmisíti.

19. Az internet mérete körülbelül 18 havonta megduplázódik. Az internetes hosztok pontos számát senki sem tudja, de a becslések szerint ez 2009-ben körülbelül 600 millió volt. Ezt az adatot felhasználva számolja ki az internetes hosztok várható számát 2018-ban! Elhiszi ezt a becslést? Indokolja meg, hogy miért igen, vagy miért nem!

20. Amikor egyik számítógépről a másikra viszünk át egy fájlt, akkor két nyugtázási stratégia lehetséges. Az első változatban a fájlt csomagokra osztjuk fel, és azokat a vevő egyenként nyugtázza, de az egész átviteli folyamatra nem ad nyugtát. A másik változatban a vevő a csomagokra nem ad nyugtát, viszont az egész fájl átvitelét nyugtázza, amikor az teljesen megérkezett. Vitassuk meg a két változatot!

21. A mobiltelefon-hálózatokban a szolgáltatóknak tudniuk kell, hogy hol található az előfizető mobiltelefonja (és így maga az előfizető is). Magyarázza meg, hogy miért

rossz ez az előfizető számára. Majd adjon indoklást arra is, hogy ez miért jó az előfizetőnek.

22. Mennyi volt egy bit hossza méterben az eredeti 802.3-as szabvány szerint? Számoljon 10 Mb/s-os átviteli sebességgel, és használja azt a feltevést, hogy a koaxkábelen a terjedési sebesség a fény vákuumban mért sebességének 2/3 része.

23. Egy kép 1600 × 1200 képpontos méretű, 3 bájt/képpontos színfelbontású. Tegyük fel, hogy a kép nincs tömörítve. Mennyi ideig tart átvinni ezt a képet egy 56 kb/s-os modemes csatornán? Egy 1 Mb/s-os kábelmodemen? Egy 10 Mb/s-os Etherneten? 100 Mb/s-os Etherneten? Gigabites Etherneten?

24. Az Ethernet és a vezeték nélküli hálózatok között hasonlóságok és különbségek egyaránt előfordulnak. Az Ethernet egyik tulajdonsága az, hogy egyszerre csak egy keret utazhat egy Etherneten. Hasonlít ebben a 802.11 az Ethernetre? Válaszát fejtse ki!

25. Adjuk meg két előnyét és két hátrányát annak, hogy a hálózati protokollokat nemzetközileg szabványosítják!

26. Amikor egy rendszer egy rögzített és egy elmozdítható részből épül fel (például egy CD-ROM-meghajtó és maga a CD-ROM), fontos, hogy az ilyen rendszert szabványosítsuk, ugyanis csak így érhető el, hogy a különböző gyártók által forgalomba hozott rögzített és elmozdítható részek együtt is működőképesek legyenek. Soroljunk fel három olyan dolgot a számítástechnika területén kívül, amelynél nemzetközi szabvány létezik! Ezek után soroljunk fel három olyat is, amelynél nem létezik ilyen szabvány!

27. Tegyük fel, hogy a k. réteg műveleteinek megvalósítására használt algoritmusokat meg kell változtatni. Hogyan érinti ez a $k-1$. és a $k+1$. réteg működését?

28. Tegyük fel, hogy megváltozik a k. réteg által nyújtott szolgáltatás (az általa biztosított műveletek köre). Hogyan érinti ez a $k-1$. és a $k+1$. réteg szolgáltatásait?

29. Soroljon fel lehetséges okokat arra vonatkozóan, hogy egy kliens válaszideje miért lehet nagyobb, mint az optimális késleltetés!

30. Készítsen egy listát az olyan mindennapos tevékenységeiről, amelyekhez számítógép-hálózatokat használ. Hogyan változna az élete, ha ezeket a hálózatokat hirtelen valaki lekapcsolná?

31. Derítse ki, hogy milyen hálózatokat használnak az iskolájában vagy munkahelyén! Írja le az ezekben használt hálózattípusokat, topológiákat és kapcsolási módszereket!

32. A *ping* program segítségével egy tesztelő csomagot küldhetünk egy adott helyre, hogy megmérjük, mennyi időt utazik oda és vissza. Használja most a *ping*et arra, hogy ki-

derítse, mennyi ideig tart, amíg a csomag a tartózkodási helyétől különféle más ismert helyekre eljut! Ezekből az adatokból készítsen egy olyan grafikont, amely az interneten való csomaghaladás idejét, az útidőt ábrázolja a távolság függvényében! A legjobb az, ha egyetemeket használ, mivel az egyetemi szerverek helye nagyon pontosan ismert. Például, a *berkeley.edu* a kaliforniai Berkeleyben, a *mit.edu* a Massachusetts-i Cambridge-ben, a *vu.nl* a hollandiai Amszterdamban, a *www.usyd.edu.au* az ausztráliai Sydneyben, a *www.uct.ac.za* pedig a dél-afrikai Fokvárosban van.

33. Menjen el az IETF webhelyére a *www.ietf.org* címen, és derítse ki, mivel foglalkozik! Válasszon ki egy tetszőleges tervet, és írjon egy féloldalas jelentést a problémáról és az arra vonatkozó javasolt megoldásról.

34. Az internet nagyszámú hálózat összessége. Ezek elrendezése határozza meg az internet topológiáját. Az internet topológiájáról tekintélyes mennyiségű információ található online. Egy kereső szolgáltatás segítségével derítsen ki többet az internet topológiájáról, és egy rövid jelentésben foglalja össze a találtakat!

35. Keresse meg az interneten, hogy hol található néhány kapcsolódási pont a szolgáltatók között, melyek az interneten továbbított csomagok forgalomirányításában vesznek részt.

36. Készítsen el egy programot, amely az üzenetek áramlását valósítja meg a hétrétegű protokoll-modell legfelső és legalsó rétege között! A program tartalmazzon külön protokoll-függvényt minden réteghez. A protokollok fejrészei legfeljebb 64 karakterből álló sorozatok. Minden protokoll-függvény két paraméterrel rendelkezik: a felsőbb rétegből átadott üzenet (egy karaktertömb) és az üzenet mérete. A függvény beszúrja a fejrészt az üzenet elé, kiírja az új üzenetet a szabványos kimenetre, majd meghívja az alatta levő réteg protokoll-függvényét. A program bemenete egy alkalmazásból jövő üzenet (egy 80 vagy kevesebb karakterből álló sorozat).

2. A fizikai réteg

Ebben a fejezetben a protokollmodell legalsó rétegével, a fizikai réteggel foglalkozunk. Ez definiálja a hálózatok villamos, időzítési és egyéb interfészeit, amelyek a biteket mint jeleket a csatornákon keresztül továbbítják. A fizikai réteg az alap, amelyre a hálózat épül. A különféle fizikai csatornák határozzák meg a teljesítőképességet (átbocsátóképesség, várakozási idő és hibaarány), tehát jó kiindulási pontot jelentenek utazásunkhoz a hálózatok világába.

A tárgyalást az adatátvitel elméleti analízisével kezdjük, és rögtön az is kiderül, hogy az anyatermészet korlátot szab a csatornákon átvihető adatmennyiségre. Ezután az átviteli közegek három fajtáját vizsgáljuk meg: a vezetékes (rézvezeték és üvegszál), a vezeték nélküli (földi rádiós) és a műholdas átviteli közegeket. A mai hálózatokban használt főbb átviteli megoldásokkal kapcsolatos háttér-információval is szolgálunk.

Ezek után következik a digitális moduláció, ami arról szól, hogy az analóg jeleket miként alakítják át digitális bitekké, majd újra vissza. Majd a multiplexelési eljárásokat tekintjük át, feltárva, hogy egyszerre több beszélgetés miként bonyolítható le ugyanazon az átviteli közegen keresztül interferencia nélkül.

Végezetül példaként megvizsgálunk három kommunikációs rendszert, amelyeket a nagy kiterjedésű számítógépes hálózatok gyakorlati megvalósításaiban is használnak: a (vezetékes) telefonrendszert, a mobiltelefon-rendszert és a kábeltévérendszert. Ezek közül mindegyik fontos a gyakorlatban, ezért mindegyikkel részletesen foglalkozunk.

2.1. Az adatátvitel elméleti alapjai

Információt úgy lehet vezetéken továbbítani, hogy valamilyen fizikai jellemzőt, például feszültséget vagy áramerősséget változtatunk rajta. Ha a feszültség vagy az áramerősség változását egy egyváltozós időfüggvénnyel, $f(t)$-vel írjuk le, akkor modellezni tudjuk a jelek viselkedését, és így lehetőség nyílik a jelek matematikai eszközökkel történő elemzésére. A következő szakaszokban ezzel az elemzéssel foglalkozunk majd.

2.1.1. Fourier-analízis

A 19. század elején Jean-Baptiste Fourier francia matematikus bebizonyította, hogy bármely T periódusidejű, periodikus $g(t)$ függvény előállítható szinuszos és koszinuszos tagok (általában végtelen) összegeként:

$$g(t)=\frac{1}{2}c+\sum_{n=1}^{\infty}a_n\sin(2\pi nft)+\sum_{n=1}^{\infty}b_n\cos(2\pi nft) \qquad (2.1)$$

ahol $f = 1/T$ az alapfrekvencia, a_n és b_n pedig az n-edik **harmonikus** (tag) szinuszos, illetve koszinuszos amplitúdója. Ezt a felbontást **Fourier-sornak** nevezzük. A Fourier-sor alapján az eredeti függvény visszaállítható, azaz a T periódusidő és az amplitúdók ismeretében az eredeti időfüggvény meghatározható a (2.1) összeg alapján.

Egy időkorlátos adatjel (az összes valódi jel ilyen) tárgyalásakor azt feltételezzük, hogy a teljes jelalak örökké ismétlődik (azaz a T és $2T$ közötti intervallumbeli viselkedés ugyanaz, mint a 0 és T közötti intervallumban).

Az a_n amplitúdót bármilyen $g(t)$ függvényhez ki tudjuk számolni, ha a (2.1) egyenlet mindkét oldalát megszorozzuk $\sin(2\pi kft)$-vel, majd az így kapott kifejezést integráljuk 0 és T között. Mivel

$$\int_0^T \sin(2\pi kft)\sin(2\pi nft)dt=\begin{cases}0 \text{ ha } k\neq n\\ T/2 \text{ ha } k=n\end{cases}$$

ezért az összegnek csak egyetlen tagja marad: a_n. A b_n-es kifejezések összege kiesik. Hasonlóan, ha a (2.1) egyenlet mindkét oldalát $\cos(2\pi kft)$-vel szorozzuk meg, majd 0 és T között integrálunk, akkor megkapjuk b_n-t. Ha viszont az egyenlet mindkét oldalát egyből integráljuk, akkor megkaphatjuk a c-t. Az előbb említett műveletek végrehajtása után a következőket kapjuk:

$$a_n=\frac{2}{T}\int_0^T g(t)\sin(2\pi nft)dt \qquad b_n=\frac{2}{T}\int_0^T g(t)\cos(2\pi nft)dt \qquad c=\frac{2}{T}\int_0^T g(t)dt$$

2.1.2. Sávkorlátozott jelek

A fentiek oly módon kapcsolódnak az adatátvitelhez, hogy a valóságos csatornák különböző frekvenciájú jeleket különbözőképpen befolyásolnak. Tegyük fel, hogy egy 8 bites bájt formájában kódolt ASCII „b" karaktert akarunk elküldeni. A továbbítandó bitminta a 01100010. A 2.1.(a) ábra bal oldalán azt láthatjuk, hogy a számítógép kimenetén hogyan változik a feszültség értéke. A jel Fourier-sorának együtthatói:

$$a_n=\frac{1}{\pi n}[\cos(\pi n/4)-\cos(3\pi n/4)+\cos(6\pi n/4)-\cos(7\pi n/4)]$$

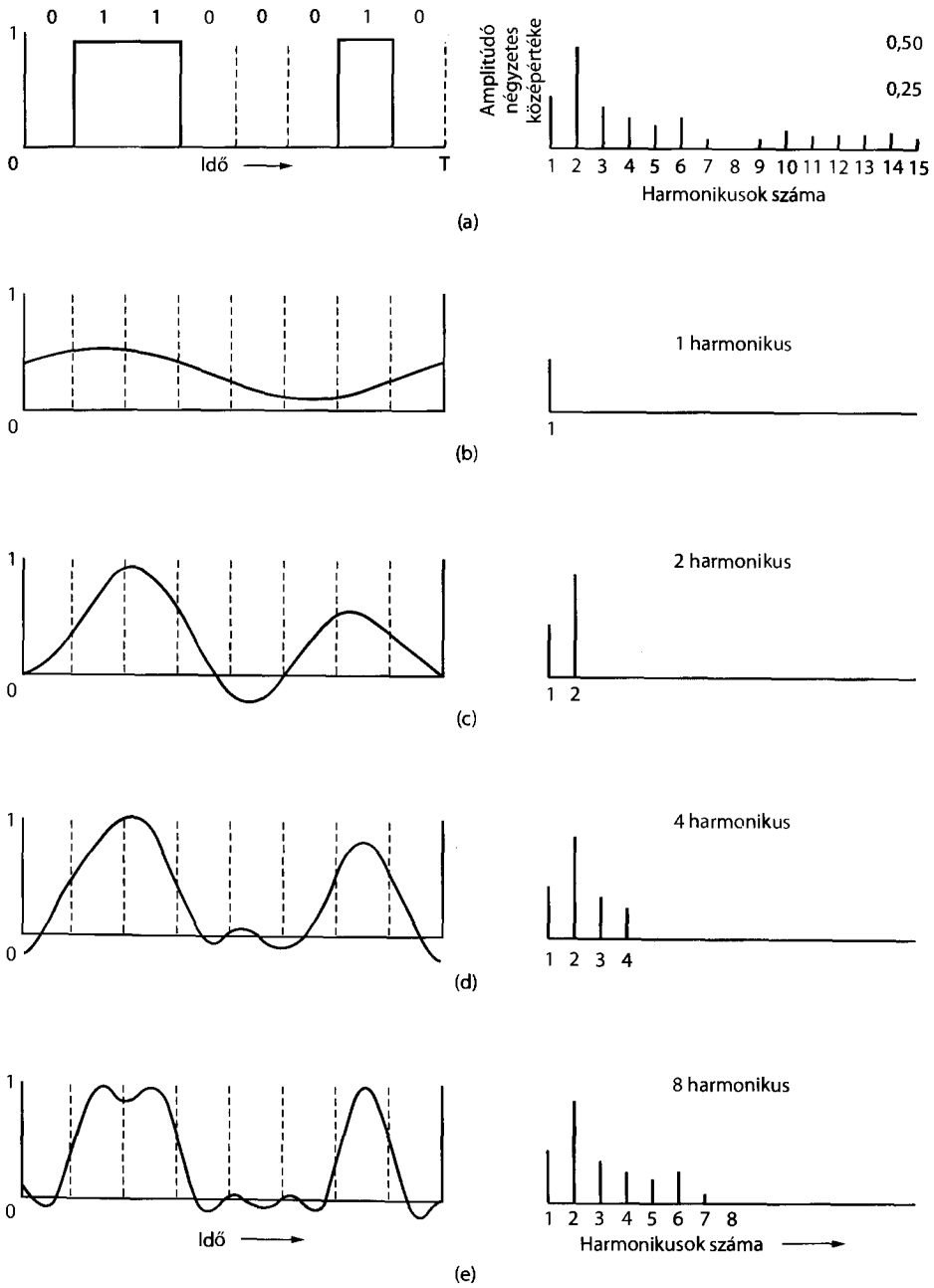
$$b_n=\frac{1}{\pi n}[\sin(3\pi n/4)-\sin(\pi n/4)+\sin(7\pi n/4)-\sin(6\pi n/4)]$$

$$c=3/4$$

A 2.1.(a) ábra jobb oldalán az első néhány harmonikus amplitúdójának négyzetes középértékét, azaz a $\sqrt{a_n^2+b_n^2}$ kifejezést láthatjuk. Ezek az értékek azért érdekesek, mert négyzetösszegük arányos az adott frekvencián továbbított energiával.

Nincs olyan adatátviteli eszköz, amely a jeleket energiaveszteség nélkül tudná továbbítani. Ha a Fourier-sor összes tagja azonos mértékben csillapodna, akkor az elküldött jelnek csak az amplitúdója csökkenne, de a jelalak nem torzulna (tehát ugyanolyan szép négyszögletű hullámalak lenne, mint amilyet a 2.1.(a) ábrán láthatunk). Sajnos a valós átviteli közegek a Fourier-sor egyes tagjait különböző mértékben csillapítják, így a jelalak mindig torzul. Általában 0 és egy bizonyos f_c frekvencia között a komponensek lényegében csillapítás nélkül terjednek, míg e felett az f_c vágási frekvencia felett a komponensek erősen csillapodnak. [A frekvencia mértékegysége egyébként rezgés/másodperc vagy Hertz (Hz)]. Azt a frekvenciatartományt, amelyen belül a csillapítás mértéke nem túl nagy, **sávszélességnek (bandwidth)** nevezzük. A gyakorlatban a csillapítás megváltozása nem igazán éles, ezért a sávszélesség 0-tól addig a frekvenciáig tart, amelynél a jel teljesítménye az eredeti jel teljesítményének felére csökken.

A sávszélesség az átviteli közeg fizikai tulajdonsága, amely többek között a rézvezeték vagy az üvegszál kialakításától függ. Gyakran használnak szűrőket, hogy tovább korlátozzák egy jel sávszélességét. A 802.11 vezeték nélküli csatornák például megközelítőleg 20 MHz-et használhatnak, ezért a 802.11 rádiók szűrői a jel sávszélességét erre a méretre korlátozzák. Egy másik példa szerint a hagyományos (analóg) televíziócsatornák egyenként 6 MHz-et foglalnak el a vezetéken vagy az éterben. Ez a szűrés lehetővé teszi, hogy több jel osztozzon a spektrum egy adott tartományán, ami megnöveli a teljes rendszer hatékonyságát. Ez azt jelenti, hogy egyes jeleknél a frekvenciatartomány nem nullától kezdődik, de ez nem probléma. A sávszélesség továbbra is az áteresztő frekvenciasáv szélessége, és a hordozható információ csak ettől a szélességtől függ, és nem a kezdő és a záró frekvenciától. Azokat a jeleket, amelyek nullától a maximális frekvenciáig terjedő sávban futnak **alapsávi (baseband)** jeleknek hívjuk. Azokat a jeleket pedig, amelyek átviteli sávját egy nagyobb frekvenciatartomány felé tolják el – a vezeték nélküli átvitelben minden esetben –, **áteresztősávi (passband)** jeleknek nevezzük. Most vizsgáljuk meg, hogy hogyan nézne ki a 2.1.(a) ábrán látható jelalak, ha a sávszélesség olyan kicsi lenne, hogy csak a legkisebb frekvenciákat lehetne továbbítani (vagyis a jelalak időfüggvényét a (2.1) egyenlet első néhány tagjával közelítenénk). A 2.1.(b) ábrán az a jelalak látható, amelyet akkor kapnánk, ha a csatorna csak az első harmonikust (az alapharmonikust) engedné át. A 2.1.(c)–(e) ábrákon a továbbított jel spektruma és visszaállítás utáni jelalakja látható a nagyobb sávszélességű csatornák esetén. Digitális átvitel esetén a cél az, hogy megfelelő pontosságú jelet kapjunk ahhoz, hogy a küldött bitfolyam rekonstruálható legyen. Ezt már könnyen megtehetjük a 2.1.(e) ábrán, tehát felesleges további harmonikusok használata azért, hogy pontosabb jelalakot kapjunk.



2.1. ábra. *(a) Bináris jel és Fourier-együtthatóinak négyzetes középértéke (root-mean-square, rms). (b)–(e) Az eredeti jel sorozatos közelítése*

b/s	T (ms)	Alapharmonikus (Hz)	Elküldött harmonikusok száma
300	26,67	37,5	80
600	13,33	75	40
1200	6,67	150	20
2400	3,33	300	10
4800	1,67	600	5
9600	0,83	1200	2
19200	0,42	2400	1
38400	0,21	4800	0

2.2. ábra. *Az adatsebesség és a harmonikusok közötti kapcsolat a példa alapján*

Ha adott az adatsebesség b b/s, akkor (például) 8 bit egyenként való elküldéséhez $8/b$ másodpercnyi idő szükséges, vagyis az első harmonikus frekvenciája $b/8$ Hz. A gyakran **beszédminőségű vonalnak** (**voice-grade line**) is nevezett szokványos telefonvonalaknak valamivel 3000 Hz felett van egy vágási frekvenciája, amelyet mesterségesen építettek a rendszerbe. Ez a korlátozás azt jelenti, hogy a legmagasabb átvitt harmonikus durván $3000/(b/8)$, vagyis $24\,000/b$ (a vágás nem éles).

A 2.2. ábrán az áteresztett harmonikusok számát adtuk meg egy táblázatban, különböző adatsebességek esetén. Ezekből a számokból kiderül, hogyha megpróbálunk 9600 b/s-os adatsebességgel egy beszédminőségű telefonvonalon adatokat továbbítani, akkor a 2.1.(a) ábrán látható jelekből a 2.1.(c) ábrán látható jelek lesznek. Ez viszont az eredeti bináris jelek pontos vételét igen megnehezíti. Nyilvánvaló, hogy 38,4 kb/s-nál jóval nagyobb adatsebesség esetén semmi esélyünk nincs arra, hogy *digitális* jeleket továbbítsunk, még akkor sem, hogyha az átviteli eszköz teljesen zajmentes. Magyarán a sávszélesség korlátozása korlátozza az adatsebességet is, és ez még zajmentes csatorna esetén is igaz. Persze vannak olyan ügyes kódolási eljárások, amelyek több különböző feszültségszintet használnak, és jóval nagyobb adatsebességet lehet velük elérni. Ezekről az eljárásokról később még lesz szó ebben a fejezetben.

Sok félreértésre ad okot, hogy a sávszélesség mást jelent a villamosmérnökök és az informatikusok számára. Villamosmérnökök számára az (analóg) sávszélesség (ahogy fentebb is szerepel) egy Hz-ben mérhető mennyiség, míg az informatikusok számára a (digitális) sávszélesség a csatorna maximális adatsebességét jelenti, egy bit/sec-ban mérhető mennyiséget. Az adatsebesség a digitális adatátvitelre használt fizikai csatorna analóg sávszélességének használatából adódó végeredmény, és a kettő nem független, ahogy ez a későbbiekben részletesen látható. Ebben a könyvben a szövegkörnyezetből mindannyiszor egyértelmű lesz, hogy az analóg sávszélességről (Hz) vagy a digitális sávszélességről (bit/sec) van-e épp szó.

2.1.3. A csatorna maximális adatsebessége

Henry Nyquist, az AT&T mérnöke már 1924-ben észrevette, hogy még egy tökéletes csatornának is véges az átviteli kapacitása. Egy olyan egyenletet vezetett le, amely egy véges sávszélességű zajmentes csatorna maximális adatsebességét fejezi ki. 1948-ban Claude Shannon folytatta Nyquist munkáját, és kiterjesztette a véletlen (vagyis termodinamikus) zajnak kitett csatornákra is [Shannon, 1948]. Ez a tanulmány az egész információelmélet legfontosabb tanulmánya. Mi itt most csak röviden fogjuk összefoglalni az azóta klasszikussá vált eredményeiket.

Nyquist bebizonyította, hogy ha egy tetszőleges jelet egy B sávszélességű aluláteresztő szűrőn bocsátunk át, akkor a szűrt jelből másodpercenként vett (pontosan) $2B$ minta alapján az eredeti jel helyreállítható. Másodpercenként $2B$ mintánál többet nem érdemes venni a jelből, mivel a szűrő kiszűrné azokat a nagyobb frekvenciájú komponenseket, amelyeket a mintavételezéssel helyre tudnánk állítani. Ha a jelnek V különböző diszkrét szintje van, akkor a Nyquist-frekvenciatétel a következőt mondja ki:

$$\text{Maximális adatsebesség} = 2B \log_2 V \text{ bit/sec} \tag{2.2}$$

Például egy zajmentes, 3 kHz sávszélességű csatornán bináris (azaz kétszintű) jelek továbbítása esetén nem lehet 6000 b/s-nál nagyobb adatsebességet elérni.

Eddig csak a zajnélküli csatornákról ejtettünk szót. Ha a csatornán véletlen zaj is jelen van, a helyzet azonnal romlani kezd. Véletlen (termikus) zaj pedig a rendszerben levő molekulák mozgása miatt mindig van jelen. A jelenlévő termikus zaj mennyiségét a jel és a zaj teljesítményének arányával mérik, amelynek **jel/zaj viszony (Signal-to-Noise Ratio, SNR)** a neve. Ha a jel teljesítményét S-sel, a zaj teljesítményét N-nel jelöljük, akkor a jel/zaj viszony S/N. Általában a hányadost $10 \log_{10} S/N$ logaritmikus skálán ábrázoljuk, annak hatalmas értéktartománya miatt. A logaritmikus skála egységeit **decibelnek (dB)** hívjuk, ahol a „deci" tizedet jelent, a „bel" pedig Alexander Graham Bell, a telefon feltalálójának állít emléket. Ha $S/N = 10$, akkor ez 10 dB, ha $S/N = 100$, akkor ez 20 dB, ha $S/N = 1000$, akkor ez 30 dB és így tovább. A sztereo erősítők gyártói gyakran úgy jellemzik a terméküknek azt a sávszélességét (frekvenciatartományát), amelyben a termékük lineárisan működik, hogy a sáv két szélén a –3 dB-es pontokhoz tartozó frekvenciákat adják meg. Ezek azok a pontok, amelyeknél az erősítési tényező megközelítőleg 0,5, mivel $10 \log_{10} 0{,}5 \approx -3$.

Shannon legjelentősebb eredménye az az összefüggés, amelyben a maximális adatsebességet vagy **kapacitást** egy olyan zajos csatornára adja meg, amelynek sávszélessége B, jel/zaj viszonya pedig S/N:

$$\text{Maximális adatsebesség} = B \log_2 (1 + S/N) \text{ bit/sec} \tag{2.3}$$

Ez adja meg a valódi csatornákon elérhető legnagyobb kapacitást. Például az **ADSL (Asymmetric Digital Subscriber Line – aszimmetrikus digitális előfizetői vonal)**, ami internet-hozzáférést tesz lehetővé normál telefonvonalon keresztül, körülbelül 1 MHz-es sávszélességet használ. A jel/zaj viszony nagymértékben a felhasználó lakása és a telefonközpont távolságától függ, és egy megközelítőleg 40 dB-es jel/zaj viszony 1-2 km-es rövid vezetéken jónak tekinthető. Ezekkel a jellemzőkkel a csatorna 13 Mb/s-nál nagyobb átvitelt nem tesz lehetővé, függetlenül attól, hogy a jelnek hány szintje van, illetve, hogy milyen gyakorisággal veszünk mintát belőle. A gyakorlatban a felhasználók az ADSL 12 Mb/s névleges értékénél gyakran kisebb sebességekkel találkoznak. Ez az adatsebesség tulajdonképpen elég jó. A több mint 60 évnyi híradástechnikai fejlesztések jelentősen csökkentették a Shannon-kapacitás és a valóságos csatornák kapacitásának különbségét.

Shannon képlete információelméleti megfontolásokon alapul, és minden olyan csatornára érvényes, amelyben termikus zaj van jelen. Az ellenpéldák ugyanabba a kategóriába esnek, mint az örökmozgók. Az ADSL 13 Mb/s-os sebességének növeléséhez a jel/zaj viszony javítására (például digitális ismétlők vonalba helyezésével a fogyasztókhoz közelebb) vagy a sávszélesség növelésére van szükség, ahogy ez az ADSL2+ nevű továbbfejlesztés esetében történt.

2.2. Vezetékes átviteli közegek

A fizikai réteg célja az, hogy egy bitfolyamot szállítson az egyik géptől a másikig. A tényleges átvitelhez különféle fizikai közegeket használhatunk fel. Mindegyiknek megvan a maga alkalmazási területe, sávszélesség, késleltetés, költség, a telepítés, valamint a karbantartás nehézsége szerint. A közegeket durva közelítéssel két csoportba oszthatjuk: vezetékes közegekre, mint például a rézvezeték vagy az üvegszál, és vezeték nélküli közegekre, mint például a levegőben terjedő rádió vagy lézer. A vezetékes átviteli közegeket a 2.2. alfejezet tárgyalja, a vezeték nélküli közegeket a következő, 2.3. szakaszban vizsgáljuk meg.

2.2.1. Mágneses hordozó

Az adatok egyik számítógéptől a másikig való szállításának egyik leggyakoribb módja, hogy mágneses szalagra vagy valamilyen lemezre (például írható DVD-re) írjuk őket, majd fizikailag elszállítjuk a szalagokat vagy lemezeket a célgéphez, és ott újra beolvassuk az adatokat. Bár ez a módszer közel sem olyan kifinomult, mint ha egy geoszinkron kommunikációs műholdat használnánk, de gyakran költséghatékonyabb megoldás. Ez kiváltképp az olyan alkalmazásoknál igaz, ahol a nagy sávszélesség vagy a kis bitenkénti költség kulcsfontosságú tényező.

Egyszerű fejszámolással is beláthatjuk ezt az érvet. Az iparban szabványos Ultrium kazetta 800 gigabájtos kapacitású. Egy 60 × 60 × 60 cm-es dobozban körülbelül 1000 db ilyen kazetta elfér, amely összesen 800 terabájt vagy 6400 terabit (6,4 petabit) kapacitást jelent. Egy kazettákkal teli dobozt a Federal Express vagy más vállalat 24 órán belül bárhova kiszállít az Egyesült Államokon belül. Ennek az átvitelnek a tényleges sávszélessége 6400 terabit/86 400 s, vagyis kicsit több mint 70 Gb/s. Ha a címzett csak egy órányi autóútra van, akkor a sávszélesség több mint 1700 Gb/s-ra növekszik. Nincs olyan számítógép-hálózat, amely ezt akár csak megközelíteni tudná. Természetesen a hálózatok egyre gyorsabbakká válnak, de a kazetták tárolási sűrűsége is egyre növekszik.

Ha egy pillantást vetünk az árra, hasonló képet láthatunk. Egy Ultrium kazetta körülbelül 40 dollárba kerül, ha nagy tételben vesszük. Egy kazettát legalább tízszer lehet újra használni, így a kazetták költsége talán 4000 dollár dobozonként és használatonként. Adjunk ehhez még hozzá 1000 dollárt a szállításért (bár ez valószínűleg sokkal kevesebb), és így körülbelül 5000 dolláros költséggel szállítunk 800 TB-ot, vagyis a szállítás gigabájtonként alig valamivel drágább, mint fél cent. Ez bármilyen hálózattal szemben verhetetlen. A történet tanulsága:

Soha ne becsüld le egy olyan furgon sávszélességét, amely kazettákkal telepakolva száguld az autópályán!

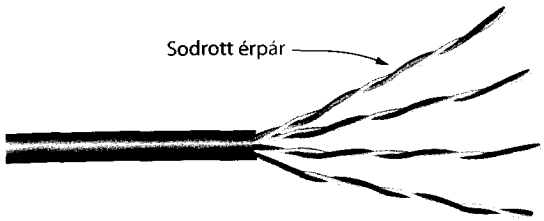
2.2.2. Sodrott érpár

Bár sávszélesség szempontjából a mágnesszalag kiváló, sajnos a késleltetése igen jelentős. Az adatátviteli időt percekben vagy órákban lehet mérni, nem pedig milliszekundumokban. A legtöbb alkalmazás esetén *online* összeköttetésre van szükség. A legrégebbi, de még ma is a legelterjedtebb átviteli közeg a **sodrott érpár** (**twisted pair**). A sodrott érpár két szigetelt rézhuzalból áll, melyek tipikusan kb. 1 mm vastagságúak. A huzalok a DNS-hez hasonlóan spirálszerűen egymás köré vannak sodorva. A sodrás oka az, hogy két párhuzamos huzal kiváló antennaként működik. Amikor a vezetékeket összesodorják, az egyes sodrott huzalokból érkező hullámok kioltják egymást, tehát az eredményül kapott huzal kevésbé sugároz. A jel általában az érpár két huzalának feszültségkülönbségeként kerül átvitelre. Ez kevésbé érzékeny a külső zajra, hiszen a zaj mindkét huzalt ugyanolyan mértékben befolyásolja, viszont a különbséget változatlanul hagyja.

A sodrott érpárt leggyakrabban a telefonrendszerekben használják. Szinte majdnem minden telefonkészüléket sodrott érpár köt össze a telefontársaság (telco) telefonközpontjával. Mind a telefonhívások, mind pedig az ADSL-internetforgalom szintén ezeken a vonalakon keresztül bonyolódik. A sodrott érpár akár több kilométeres szakaszon is erősítés nélkül használható, de nagyobb távolságok esetén már szükség van erősítőkre. Amikor hosszabb távolságon keresztül több sodrott érpár fut egymás mellett (például amikor egy épületből az összes vezeték a telefonközpontba megy), akkor a sodrott érpárokat egy kötegbe fogják, és ezt a köteget mechanikai védelemmel látják el. Ha az érpárok nem lennének sodorva, akkor a kötegen belül biztosan zavarnák egymás forgalmát. A világ azon részein, ahol a telefonvonalakat telefonpóznákon vezetik, még ma is gyakran láthatunk ilyen több centiméter átmérőjű érpárkötegeket.

A sodrott érpár alkalmas mind analóg, mind digitális jelátvitelre. A vezetékek sávszélessége a vastagságától és az áthidalt távolságtól függ, de sok esetben néhány Mb/s sebességet is el lehet velük érni pár kilométeres távolságon belül. Megfelelő teljesítményüknek és alacsony áruknak köszönhetően a sodrott érpárokat széles körben használják, és ez várhatóan így marad még jó néhány évig.

A sodrott érpárnak számos változata van. A sok irodaházban telepített közönséges változatát **5-ös kategóriájú** (**Category 5**) vagy „Cat 5"-ös kábelezésnek nevezzük. Az 5-ös kategóriájú sodrott érpár két finoman egymás köré sodrott, szigetelt vezetékből áll. Általában négy ilyen érpárt fognak össze egy műanyag köpennyel, ami védi, és egyben tartja a nyolc vezetéket. Ezt az elrendezést a 2.3. ábra szemlélteti.



2.3. ábra. *5-ös kategóriájú UTP-kábel négy sodrott érpárral*

A különféle LAN-szabványok különbözőképpen használják a sodrott érpárokat. Például a 100 Mb/s-os Ethernet két párat használ (a négyből), irányonként egy-egy párat. A nagyobb sebesség érdekében az 1 Gb/s-os Ethernet mind a négy párat egyidejűleg használja mindkét irányba; ez azt igényli a fogadótól, hogy a helyben átvitt jelet alkotóelemeire bontsa.

Néhány általános terminológia ezzel rendben is volna. Azokat az összeköttetéseket, amelyeket egyidejűleg két irányba is lehet használni, mint egy kétsávos utat, **duplex** (**full-duplex**) adatkapcsolatnak hívjuk. Ezzel szemben, azokat az adatkapcsolatokat, amelyeket bármelyik irányba, de egyszerre csak egyfelé lehet használni, mint egy egyvágányú vasútvonalat, **fél-duplex** (**half-duplex**) adatkapcsolatnak nevezzük. A harmadik csoportba azok az adatkapcsolatok tartoznak, amelyek csak egy irányba teszik lehetővé az adatforgalmat, hasonlóan egy egyirányú utcához. Ezeket **szimplex** (**simplex**) adatkapcsolatnak hívjuk.

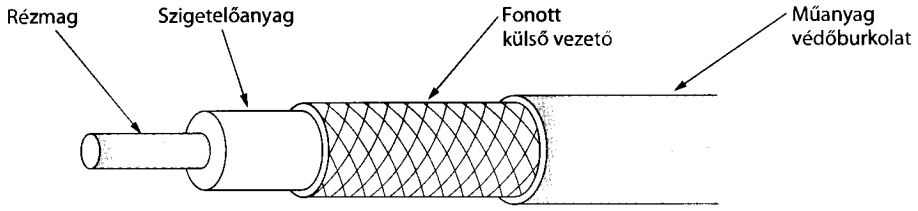
Visszatérve a sodrott érpárokhoz, a Cat 5 a korábbi **3-as kategóriájú** (**Cat 3**) kábeleket váltotta fel, hasonló kábellel és ugyanolyan csatlakozóval, de méterenként több sodrással. A több sodrás kevesebb áthallást és nagyobb távolságokon is jobb minőségű jelet eredményez, így ezek megfelelőbbek a nagy sebességű számítógépes kommunikációhoz, különösen a 100 Mb/s-os és 1 Gb/s-os Ethernet LAN-okhoz.

Az új vezetékezések valószínűleg **Cat 6** vagy **Cat 7** kategóriájúak. Ezeknek a kategóriáknak még szigorúbb a specifikációja a nagyobb sávszélességű jelek kezelésére vonatkozóan. Néhány 6-os és magasabb kategóriájú vezeték 500 MHz-es jelekre kalibrált, és alkalmas a hamarosan telepítésre kerülő 10 Gb/s-os adatkapcsolatok támogatására is.

A Cat 6 kategóriáig terjedő vezetéktípusokat gyakran nevezik **UTP**-nek (**Unshielded Twisted Pair – árnyékolatlan sodrott érpár**), mivel egyszerűen vezetékekből és szigetelésekből állnak. Ezzel szemben a 7-es kategóriájú (Cat 7) kábelek esetén az egyes sodrott érpárokat árnyékolják, csakúgy, mint a teljes kábelt (de még a műanyag védőköpenyen belül). Az árnyékolás csökkenti a szomszédos vezetékek közötti külső interferenciával és az áthallásokkal szembeni érzékenységet annak érdekében, hogy megfeleljenek az elvárt teljesítőképességre vonatkozó előírásoknak. A kábelek emlékeztetnek azokra a kiváló minőségű, de vastag és drága árnyékolt sodrott érpáros kábelekre, amelyeket az IBM vezetett be az 1980-as évek elején, és amelyek később az IBM telephelyein kívül sehol sem bizonyultak népszerűnek. Kétségkívül itt az ideje az újabb próbálkozásnak.

2.2.3. Koaxiális kábel

Egy másik, széles körben használt átviteli közeg a **koaxiális kábel** (**coaxial cable**), amit a kedvelői egyszerűen csak „koax"-nak hívnak. Mivel ez jobb árnyékolással rendelkezik, mint a sodrott érpár, ezért nagyobb sebességgel nagyobb távolságot lehet vele áthidalni. Kétfajta koaxiális kábel létezik. Az egyik az 50 Ω-os kábel, amelyet elsősorban digitális átvitelhez használnak. A másik a 75 Ω-os kábel, amelyet elsősorban analóg átvitel esetén és a kábeltelevíziózásban használnak. A kettő közötti eltérésnek inkább történelmi, semmint műszaki okai vannak (például a korai dipól antennáknak 300 Ω-os impedanciájuk volt, és könnyű volt hozzájuk 4:1 arányú impedanciaillesztő transzformátort építeni). A 90-es évek közepétől a kábeltelevízió-szolgáltatók elkezdtek internet-hozzáférést biztosítani kábelen keresztül, ami az adatátvitelhez tette fontosabbá a 75 Ω-os kábelt.



2.4. ábra. *Koaxiális kábel*

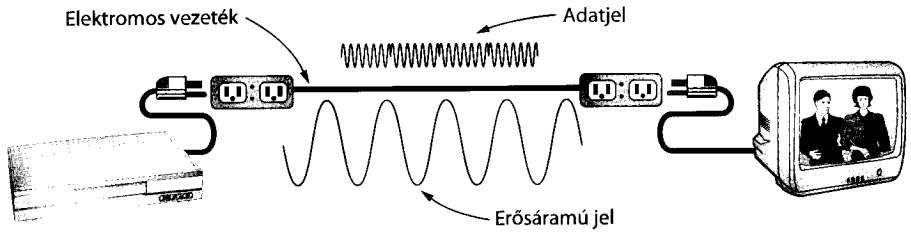
A koaxiális kábel közepén tömör rézhuzalmag van, amelyet szigetelő vesz körül. A szigetelő körül sűrű szövésű hálóból álló rézvezető található. A külső rézvezetőt műanyag burkolattal vonják be mechanikai védelem céljából. A koaxiális kábel szerkezetét a 2.4. ábrán láthatjuk.

A koaxiális kábel kialakítása és árnyékolása a nagy sávszélesség és a kiváló zajérzéketlenség jó kombinációját adja. Az elérhető sávszélesség a kábel minőségétől és hosszától függ. A mai modern kábelek sávszélessége néhány GHz. A koaxiális kábeleket régen gyakran használták a telefonrendszeren belüli nagy távolságokat áthidaló vonalakon, de ezeket manapság már nagyrészt lecserélték üvegszálakra. A koaxot azonban még mindig széleskörűen alkalmazzák a kábeltelevíziózásban és a nagyvárosi hálózatokban.

2.2.4. Erősáramú vezetékek

A telefon- és kábeltelevízió-hálózatok nem az egyetlen forrásai azoknak a vezetékeknek, amelyek adatkommunikációs célra újrahasznosítanak. Létezik egy még sokkal elterjedtebb vezetékezés: az elektromos hálózat vezetékei. Az erősáramú vezetékek elektromos áramot szállítanak a házakhoz, ahol azt elektromos vezetékezéssel osztják szét a fali csatlakozókhoz.

Az erősáramú vezetékek adatkommunikációra történő használata régi gondolat. Az áramszolgáltató vállalatok sok éve használják kis sebességű kommunikációhoz az erősáramú vezetékeket, mint például távméréshez vagy háztartási eszközök távvezérléséhez (például X10 szabvány). Az utóbbi években újra feltámadt az érdeklődés az ezeken a vezetékeken történő nagy sebességű kommunikáció iránt, mind házon belül – mint például a LAN –, mind a házon kívül, a széles sávú internet-hozzáféréshez. Mi a leggyakoribb forgatókönyvet tekintjük át: az elektromos vezetékek házon belüli használatát.



2.5. ábra. *Háztartási elektromos vezetékezést használó hálózat*

Az erősáramú vezetékek hálózatként való használatának kényelme világos. Egyszerűen csak be kell dugni a tv-készüléket és a vevőt a konnektorba, ahogy egyébként is tennénk, hiszen áramra van szükségük, majd az elektromos vezetékeken keresztül filmek küldhetők és fogadhatók. Ez az összeállítás látható a 2.5. ábrán. Nincs szükség egyéb csatlakozóra vagy antennára. Az adatjel a kis frekvenciás tápjelre van ráültetve (az aktív, vagy „forró" vezetéken), és mindkét jel ugyanazt a vezetéket használja egy időben.

A háztartási elektromos vezetékek hálózatként való használatának nehézsége az, hogy a vezetékeket eredetileg áramjelek elosztására tervezték. Ez a feladat merőben más, mint az adatjelek továbbítása, amiben a háztartási vezeték nagyon gyengén teljesít. Az elektromos jelek 50-60 Hz-en továbbítódnak és a vezetékezés csillapítja a nagy sebességű adatkommunikációhoz szükséges, lényegesen nagyobb frekvenciájú (MHz-es) jeleket. A vezetékek elektromos tulajdonságai házanként eltérőek, valamint a készülékek ki- és bekapcsolásával is módosulnak, ami az adatjelek összevissza változását okozza. A készülékek ki- és bekapcsolásakor a tranziens áram széles frekvenciatartományon okoz elektromos zajt. A sodrott érpárok gondos sodrása nélkül az elektromos vezetékek antennaként működnek, külső jeleket szednek fel, és saját jeleiket sugározzák le. Ez a tulajdonság azt jelenti, hogy az előírt követelményeknek való megfelelés érdekében az adatjel nem eshet az engedélyezett frekvenciatartományba, mint például az amatőr rádiós hullámsávba.

Mindezen nehézségek ellenére az a praktikus, ha legalább 100 Mb/s sebességgel továbbítanak normál háztartási elektromos vezetéken olyan kommunikációs módszerek alkalmazásával, amelyek ellenállnak a lecsökkentett frekvenciának és a hibacsomóknak. Az erősáramú vezetékek hálózatként való használata során sok termék alkalmaz különféle egyedi szabványt, ezért a nemzetközi szabványok kidolgozása folyamatban van.

2.2.5. Üvegszálak

A számítógépiparban sokan rettenetesen büszkék arra, hogy a Moore-törvénynek megfelelően milyen gyors az iparág fejlődése, ami nagyjából kétévenként a chipenkénti tranzisztorszám megduplázódását jósolja [Schaller, 1997]. Az eredeti (1981-es) IBM PC 4,77 MHz-es órajellel működött. Huszonnyolc évvel később a PC-k négymagos CPU-i már

3 GHz-en működtek, ami 2500-szoros növekedést jelent, vagy 16-szoros növekedést évtizedenként. Lenyűgöző.

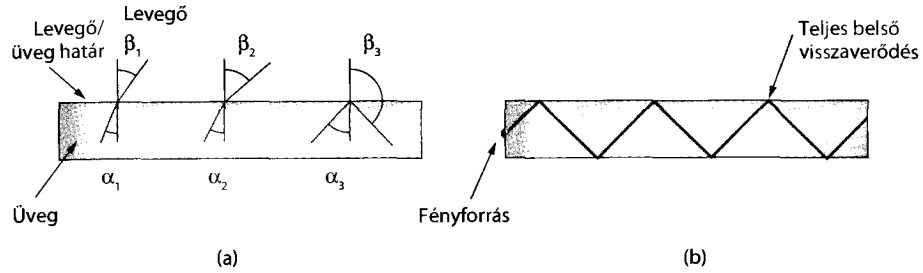
Ugyanebben az időszakban a nagy kiterjedésű adatkommunikáció sebessége 45 Mb/s-ról (a T3 vonal a telefonrendszerben) 100 Gb/s-ra nőtt (modern távolsági vonalnál). Ez a növekedés nem kevésbé lenyűgöző – több mint 2000-szeres és megközelíti a 16-szoros növekedést évtizedenként, miközben ezzel egyidejűleg a hibaarány bitenként 10^{-5}-ről majdnem nullára zuhant. Továbbá, az egyes CPU-k kapacitása kezdi megközelíteni a fizikai határokat, ez az oka annak, hogy mostanában a CPU-k chipenkénti számát növelik. Ezzel ellentétben, az üvegszálas technikával az elérhető legnagyobb sávszélesség több mint 50 000 Gb/s (50 Tb/s), és ennek a határnak még csak a közelében sem vagyunk. Az adatsebesség mai, körülbelül 100 Gb/s-os gyakorlati felső határa abból ered, hogy képtelenek vagyunk gyorsabban átalakítani a villamos jeleket optikai jelekké és vissza. Nagyobb kapacitású adatkapcsolatok kialakítása érdekében egyszerűen több csatorna kerül párhuzamosan kialakításra egy szálban.

Ebben a bekezdésben arról lesz szó, hogy az üvegszálon történő adatátvitel hogyan működik. A számítástechnika és a kommunikáció jelenleg is zajló versenyében még a kommunikáció áll nyerésre az optikai hálózatok miatt. Ebben burkoltan benne van egy lényegében végtelen sávszélességű vezeték és egy új, szokásos bölcsesség, miszerint az összes számítógép reménytelenül lassú, ezért a hálózatoknak mindenáron meg kell próbálni elkerülni a rajtuk végzett számítást, nem érdekes, hogy ez mekkora sávszélességvesztéssel jár. Időbe fog telni, míg ezt a változást a rézvezeték sávszélességét korlátozó Shannon-tétel szellemében nevelkedett informatikusok és mérnökök új generációja elfogadja.

Természetesen ez a forgatókönyv nem a teljes történetet meséli el, hiszen nem tartalmazza a költségeket. A fogyasztók eléréséhez szükséges utolsó mérföldnyi üvegszál telepítésének és a kis sávszélességű és korlátozott elérésű vezetékek kikerülésének költsége óriási. Továbbá, több energiát igényel a bitek szállítása, mint kiszámítása. Mindig lesznek persze az egyenlőtlenségnek olyan szigetei, ahol a számítás vagy a kommunikáció lényegében ingyenes. Az internet szélein például feldolgozási és tárkapacitással próbáljuk megoldani a tartalom tömörítését és ideiglenes tárolását, csupán azért, hogy jobban kihasználhassuk az internetes kapcsolatokat. Az interneten belül éppen az ellenkezőjét tehetjük, olyan cégekkel, mint például a Google, amely hatalmas mennyiségű adatot továbbít a hálózaton keresztül oda, ahol olcsóbb tárolni és számolni vele.

Az üvegszálakat a hálózatok gerincében nagy távolságú átvitelre, nagy sebességű LAN-ok (habár eddig a réznek mindig sikerült felzárkózni) és gyors internet-hozzáférések, mint amilyen például az **FttH** (**Fiber to the Home – üvegszál a lakásig**) esetén használják. Egy üvegszálas adatátviteli rendszernek három fő komponense van: a fényforrás, az átviteli közeg és a fényérzékelő (detektor). A fényimpulzus megléte szokás szerint a logikai 1 bitet jelenti, míg az impulzus hiánya a logikai 0 bitet. Az átviteli közeg egy rendkívül vékony üvegszál. Ha a detektorba fény jut, akkor a detektor villamos jelet állít elő. Ha az üvegszál egyik végére fényforrást, a másik végére pedig detektort teszünk, akkor egy olyan egyirányú adatátviteli rendszert kapunk, amely villamos jeleket fogad, átalakítja azokat fényimpulzusokká, továbbítja a fényimpulzusokat, majd a kábel másik végén a fényimpulzusokat visszaalakítja villamos jelekké.

Az ilyen adatátviteli rendszerek a fény elszivárgása miatt csak a fizikusok számára jelentenek érdekességet, a gyakorlati életben azonban használhatatlanok. Amikor a fény az egyik



2.6. ábra. *(a) Egy üvegszál belsejében a fénysugár három különböző szögben érkezik az üveg és a levegő határához. (b) Teljes belső visszaverődés miatt a fénysugár az üvegszálon belül marad*

közegből átlép egy másikba, mondjuk üvegből a levegőbe, akkor az üveg és a levegő találkozásánál a fény megtörik, ahogy ez a 2.6.(a) ábrán is látható. Az ábra egy olyan fénysugarat mutat, amely α_1 szögben érkezik meg a határfelülethez, és β_1 szögben halad tovább. A visszaverődés mértéke függ a két közeg fizikai jellemzőitől (elsősorban azok törésmutatójától). Ha a beesési szög nagyobb egy bizonyos határértéknél, akkor a fény nem lép ki a levegőre, hanem visszaverődik az üvegbe. Így ha a fénysugár beesési szöge egyenlő a *határszöggel* vagy nagyobb annál, akkor a fénysugár az üvegszálon belül marad, ahogy ezt a 2.6.(b) ábra is szemlélteti, és akár több kilométert is megtehet gyakorlatilag veszteség nélkül.

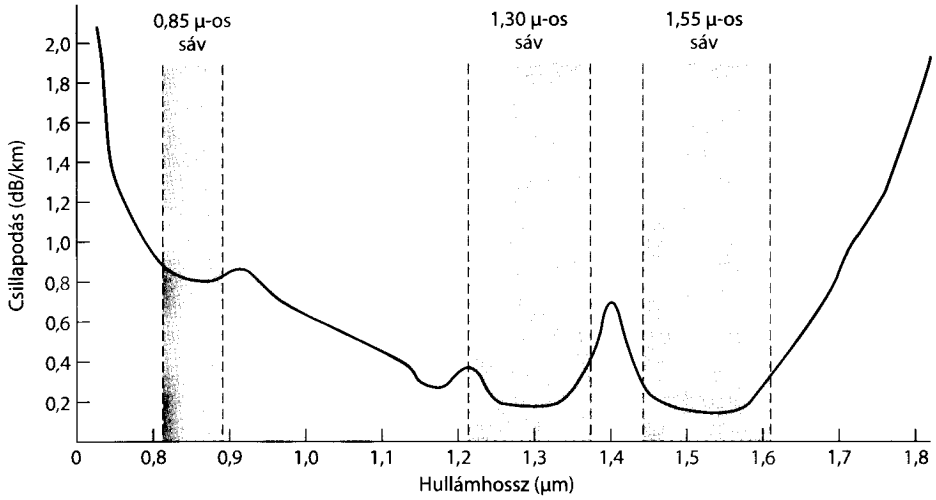
A 2.6.(b) ábrán csak egyetlen fénysugár látható, mivel azonban a határszöggel azonos vagy annál nagyobb szögben beeső sugarak mind az üvegszálon belül maradnak, ezért egyszerre sok, különböző szögben visszaverődő fénysugár halad az üvegszálban. Minden egyes sugárnak más és más az ún. **módusa**, ezért az ilyen üvegszálat **többmódusú szálnak** nevezik.

Ha viszont az üvegszál átmérőjét néhány fényhullámhossznyira lecsökkentjük, akkor az üvegszál hullámvezetőként viselkedik, és a fény visszaverődés nélkül, egyenes vonal mentén terjed a vezetékben. Az ilyen üvegszálat **egymódusú szálnak** nevezik. Az egymódusú szálak jóval drágábbak, viszont nagyobb távolságok áthidalására használhatók. A jelenleg kapható egymódusú üvegszálak másodpercenként 100 gigabitet képesek 100 km-re továbbítani erősítés nélkül. Laboratóriumi körülmények között még ennél nagyobb sebességeket is értek el rövidebb távolságok esetén.

Fény továbbítása üvegszálon

Az üvegszál üvegből készül, az üveg pedig homokból. A homok olcsó és a természetben korlátlan mennyiségben fellelhető anyag. Az üveggyártást már az egyiptomiak is ismerték, bár ők még nem tudtak 1 mm-nél vékonyabb átlátszó üveget készíteni. Az ablaknak is alkalmas, átlátszó üveget a reneszánsz korban fejlesztették ki. A mai üvegszálakban az üveg annyira átlátszó, hogy ha az óceánt víz helyett ezzel az üveggel töltenénk meg, akkor az óceán fenekét olyan tisztán lehetne látni, mint ahogy tiszta időben a földfelszínt egy repülőgép fedélzetéről.

A fényerősség csökkenését az üvegben a fény hullámhossza (valamint az üveg néhány fizikai tulajdonsága) határozza meg. Ezt a bemenő jel és kimenő jel teljesítményének



2.7. ábra. *Üvegszálban terjedő fény csillapodása az infravörös tartományban*

hányadosával határozzuk meg. Az optikai kábelnek használt üvegszálban a csillapítás a 2.7. ábrán látható módon alakul, decibel per üvegszál folyókilométer egységben.

Ha például a jel teljesítményének csökkenése kétszeres (a felére csökken), a $10 \log_{10} 1/2 = -3$ dB csillapításnak felel meg. Az ábra a spektrum infravöröshöz közeli részét mutatja, amelyet a gyakorlatban is használnak. A látható fény hullámhossza ennél valamivel kisebb, 0,4-től 0,7 mikronig terjed (1 mikron = 10^{16} méter). Az SI-rendszer igazi elkötelezettjei 400 nm-ként és 700 nm-ként hivatkoznának ezekre a hullámhosszokra, de mi inkább ragaszkodunk a hagyományos szóhasználathoz.

Az optikai kommunikáció jelenleg három hullámhossz-tartományt használ leginkább, amelyek középpontja 0,85, 1,30 és 1,55 mikronnál van. Mindhárom sáv szélessége a 25 000 és 30 000 GHz közötti tartományba esik. A 0,85 mikronos sávot használták először. Ebben nagyobb a csillapítás, ezért rövidebb távolságokra lehet alkalmazni, de ennél a hullámhossznál a lézer és az elektronika készülhet azonos anyagból (gallium-arzenidból). Az utóbbi két sávban jók a csillapítási tulajdonságok (kevesebb mint 5 százalék kilométerenként). Az 1,55 mikronos sávot manapság széleskörűen használják erbium-adalékolású erősítőkkel, amelyek közvetlenül az optikai tartományban működnek.

A szálon végigküldött fényimpulzusok hosszanti irányban szétszóródnak terjedés közben. Ezt a szóródást **kromatikus diszperziónak** (**chromatic dispersion**; „a színek szétszóródása") nevezik, és mértéke a hullámhossztól függ. Az egyik lehetséges módszer a szétszóródott impulzusok átfedésének megakadályozására az, hogy növeljük a közöttük hagyott távolságot, de ezt csak a jelzési sebesség csökkentésével lehet elérni. Szerencsére felfedezték, hogy ha az impulzusokat egy bizonyos alakúra formáljuk (ez a koszinusz hiperbolikusz reciprokával függ össze), akkor szinte minden szóródási hatást kiejthetünk. Így lehetségessé válik, hogy ezer kilométerekre küldjünk impulzusokat bármilyen észrevehető jelalaktorzulás nélkül. Ezeket az impulzusokat **szolitonoknak** (**soliton**) nevezték el. Tekintélyes mennyiségű kutatás folyik annak érdekében, hogy a szolitonok a laborokból kikerüljenek a hétköznapi életbe.

Üvegszálas kábelek

Az üvegszálas optikai kábel hasonlít a koaxiális kábelre, a szövött árnyékolástól eltekintve. A 2.8.(a) ábra oldalnézetben mutat egyetlen üvegszálat. Középen található az üvegmag, amelyben a fény terjed. Többmódusú szál esetén a mag 50 mikron átmérőjű, azaz körülbelül olyan vastag, mint egy emberi hajszál. Egymódusú szál esetén a mag 8–10 mikron átmérőjű.

Az üvegmagot olyan üvegköpeny veszi körül, amelynek a törésmutatója kisebb mint a magé, így a fénysugár a magon belül marad. A szálat kívülről műanyag védőburkolattal látják el a köpeny védelme érdekében. A fénykábelben általában több üvegszálat fognak össze, és azokat egy műanyag csőbe helyezve védik a külső behatásoktól. A 2.8.(b) ábrán egy háromszálas kábel keresztmetszetét láthatjuk.

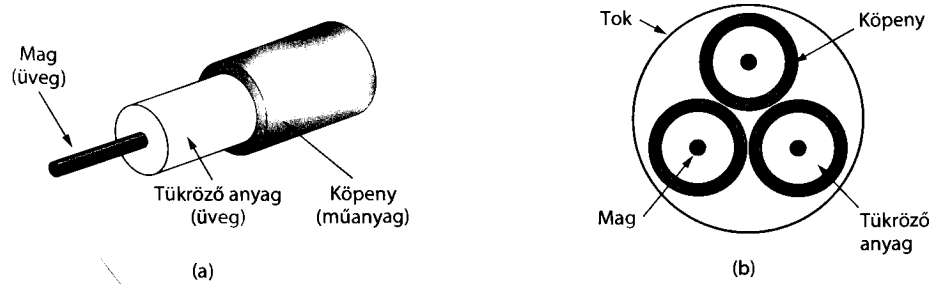
A szárazföldi fénykábeleket általában egy méter mélyre fektetik, ahol gyakran okoznak kárt a markológépek és a rágcsálók. A tengeri kábeleket a partok közelében vízi eke segítségével beszántják a tengerfenék alá, míg a mélyebb vizekben egyszerűen csak leengedik a kábeleket a tengerfenékre, ahol a halászhajók és a cápák időnként megtépázzák azokat.

Az üvegszálak háromféleképpen csatlakoztathatók egymáshoz. Az egyik módszer az, hogy az üvegszál végeit megfelelő csatlakozókkal látjuk el, és ezeket dugjuk össze. A csatlakozók 10–20% veszteséget okoznak, viszont megkönnyítik a rendszer újrakonfigurálását.

A második lehetőség, hogy a szálakat mechanikusan egymáshoz illesztjük. Ennek a módszernek az a lényege, hogy mindkét szálat meghatározott szögben óvatosan lemetsszük, majd a metszett végeket összeillesztjük, és egy szorítóval összefogjuk. Az illesztés pontossága úgy javítható, hogy az egyik üvegszálba belevilágítunk, és a két szálat finoman addig mozgatjuk, amíg a kijövő jel intenzitása a lehető legnagyobb nem lesz. A mechanikai összeillesztést egy rutinos szakember akár 5 perc alatt is el tudja végezni, és ez a csatlakoztatási mód csak 10% veszteséget okoz.

A harmadik lehetőség a két szál összehegesztése. A hegesztett szál majdnem olyan jó, mint egy gyárilag húzott szál, de azért még itt is van némi csillapítás. Mindhárom csatlakoztatási mód esetén van egy kis visszaverődés az illesztésnél, és a visszaverődött fény interferálhat az eredeti jellel.

A fényimpulzusok előállítására kétféle fényforrást használnak: az egyik a LED (Light Emitting Diode), a másik pedig a félvezető lézer. A két fényforrás sok mindenben kü-



2.8. ábra. *(a) Üvegszál oldalnézetben. (b) Három üvegszálból álló kábel keresztmetszete*

Jellemző	LED	Félvezető lézer
Adatsebesség	Kicsi	Nagy
Üvegszál típusa	Többmódusú	Többmódusú vagy egymódusú
Távolság	Kicsi	Nagy
Élettartam	Hosszú	Rövid
Hőmérséklet-érzékenység	Kicsi	Jelentős
Ár	Olcsó	Drága

2.9. ábra. *Fényforrásként szolgáló félvezető diódák és LED-ek összehasonlítása*

lönbözik egymástól. A 2.9. ábrán látható táblázatban a leglényegesebb különbségeket foglaltuk össze. A fény hullámhosszát a forrás és az üvegszál között elhelyezkedő Fabry–Perot- vagy Mach–Zehnder-interferométerrel lehet változtatni. A Fabry–Perot-interferométer egy olyan rezonanciaüregből áll, amelyet két, egymással párhuzamos tükör határol. A fény merőlegesen esik be a tükrökbe. Az üreg hosszának változtatásával a fény hullámhosszának egész számú többszöröseit lehet előállítani. A Mach–Zehnder-interferométer a fénysugarakat két olyan nyalábra osztja, amelyek az interferométeren belül közel azonos távolságot tesznek meg, majd a kimeneti ponton összefókuszálják őket, így csak bizonyos hullámhosszak esetén lesznek azonos fázisban.

Az üvegszál másik végén egy fotodióda található, amely elektromos impulzusokat állít elő, ha fény esik rá. A fotodióda tipikus késleltetése 1 ns körül van, ez korlátozza az adatsebességet kb. 1 Gb/s-ra. A termikus zaj szintén problémát jelent, ezért a fénysugárnak elegendő energiával kell rendelkeznie ahhoz, hogy detektálni lehessen. Ha a fényimpulzusok elég nagy energiával rendelkeznek, akkor a bithibaarány tetszőlegesen kicsi lehet.

Az üvegszál és a rézvezeték összehasonlítása

Igencsak tanulságos lehet az üvegszál és a rézvezeték összehasonlítása. Az üvegszálnak rengeteg előnye van. Rögtön azzal kezdjük, hogy az üvegszálnak jóval nagyobb a sávszélessége, mint a rézvezetéknek. Ez önmagában véve még csak a nagy sebességű hálózatok esetén jelentene előnyt. Tekintettel azonban a kis csillapításra, a hosszú vonalakon csak 30 km-enként van szükség ismétlőkre, szemben a rézvezetékkel, ahol kb. 5 km-enként. Ez bizony jelentős megtakarítást jelent. Az üvegszál másik nagy előnye, hogy nem érzékeny az áramimpulzusokra, az elektromágneses zavarokra és az elektromos hálózati kimaradásokra. A levegőben található korrodáló hatású vegyületek sem ártanak neki, ezért ideális megoldást jelent erősen korrodáló ipari környezetben.

A telefontársaságok rendkívüli módon kedvelik az üvegszálakat, méghozzá két dolog miatt: egyrészt mert vékonyak, másrészt mert pehelykönnyűek. Számtalan kábelcsatorna már most is teljesen tele van, így nincs hely újabb vezetékek számára. Az összes rézvezeték fényvezető kábelre történő kicserélésével ki lehetne üríteni a kábelcsatornákat, és jó pénzért el lehetne adni a rézvezetékeket a színesfém-feldolgozóknak, tekintettel

magas réztartalmukra. Az üvegszál könnyebb is, mint a rézvezeték. Ezer darab 1 km hosszú sodrott érpár súlya 8000 kg. Két üvegszálnak nagyobb a kapacitása, ugyanakkor csak 100 kg-ot nyom. Ez jelentősen csökkenti a szállítás költségeit, mivel kevesebb szállítóeszközt kell fenntartani. Üvegszálas hálózatokban új útválasztók üzembe helyezésének költségei is jóval kisebbek.

Végül az üvegszálból nem szivárog el fény, és megcsapolni is igen nehéz azt. Ez kiváló védelmet jelent a potenciális lehallgatók ellen.

A két rossz hír az, hogy az üvegszál kevésbé ismert megoldás és olyan ismeretek is szükségesek hozzá, amelyekkel nem minden mérnök rendelkezik, valamint hogy a szálak könnyen megsérülhetnek, ha túlságosan meghajlítják őket. Mivel a fényvezetős átvitel természeténél fogva egyirányú, a kétirányú kommunikációhoz vagy két szálra, vagy egy szálon két frekvenciasávra van szükség. Végül, az üvegszálak interfészei többe kerülnek, mint az elektromos interfészek. Mindezek ellenére a néhány méternél nagyobb távolságokat áthidaló, helyhez kötött (nem mobil) adatkommunikáció jövője nyilvánvalóan az üvegszálban van. Az üvegszálak és a belőlük épített hálózatok részletes tárgyalását lásd Hecht [2005] könyvében.

2.3. Vezeték nélküli adatátvitel

A mi korunkban jelentek meg az infomániások: olyan emberek, akiknek állandóan információra van szükségük, ezért mindig a hálózaton akarnak lenni. Ezeknek a felhasználóknak a sodrott érpár, a koax és a száloptika használhatatlan. „Sláger"-adataikat anélkül akarják megkapni a hordozható számítógépeikre, noteszgépeikre, ingzsebben, kézben vagy karórájukon hordozható számítógépeikre, hogy közben hozzá lennének láncolva a földi kommunikációs infrastruktúrához. Ezekre a felhasználói igényekre a vezeték nélküli kommunikáció a válasz.

A következő szakaszokban általánosan tekintjük át a vezeték nélküli kommunikációt, mivel sok fontosabb alkalmazása is van amellett, hogy internetkapcsolatot biztosítani azoknak, akik a tengerpartról akarnak szörfölni a világhálón. A vezeték nélküli rendszerek egyes körülmények között a rögzített eszközök számára is kínálnak előnyöket. Például ha egy épülethez a terep adottságai miatt (hegyek, dzsungel, mocsarak stb.) nehéz elvezetni egy üvegszálat, akkor a vezeték nélküli megoldás jobb lehet. Érdemes megjegyezni, hogy a modern vezeték nélküli digitális kommunikáció a Hawaii-szigeteken kezdődött, ahol a Csendes-óceán nagy területen szigetelte el a felhasználókat a számítóközpontjuktól, és a telefonrendszer használhatatlannak bizonyult.

2.3.1. Az elektromágneses spektrum

Amikor mozognak az elektronok, elektromágneses hullámokat keltenek maguk körül. Ezek az elektromágneses hullámok a szabad térben (sőt még a vákuumban is) tovaterjednek. Az elektromágneses hullámok létezését elsőként James Clerk Maxwell skót fizikus ismerte fel 1865-ben, majd később, 1887-ben Heinrich Hertz német fizikus elsőként

állított elő, és figyelt meg ilyen hullámokat. Az elektromágneses hullám másodpercenkénti rezgésszámát **frekvenciának** (f) nevezzük. A frekvencia mértékegysége – Heinrich Hertz tiszteletére – a Hertz (Hz). Két egymást követő hullámcsúcs (vagy hullámvölgy) közötti távolságot **hullámhossznak** hívunk, és a görög λ (lambda) betűvel jelölünk.

Ha egy elektronikus áramkörhöz megfelelő méretű antennát csatlakoztatunk, akkor az elektromágneses hullámokat szét lehet úgy szórni, hogy kicsivel arrébb venni lehessen azokat. Az összes vezeték nélküli átviteli mód ezen az elven alapul.

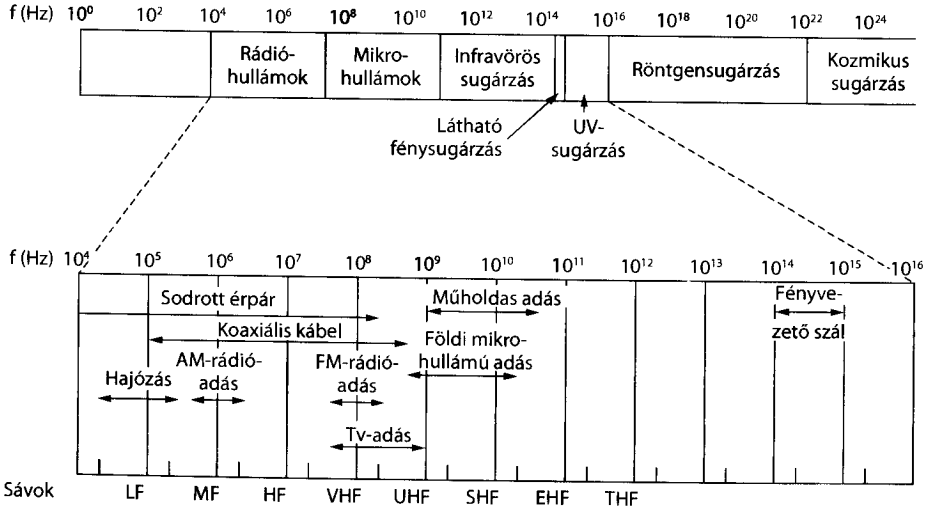
A vákuumban minden elektromágneses hullám a frekvenciájától függetlenül ugyanazzal a sebességgel terjed. Ezt a sebességet **fénysebességnek** (c) hívjuk, és értéke kb. 3×10^8 m/s, azaz kb. 30 cm/ns. Rézben és üvegszálban ez a sebesség nagyjából a 2/3-ára csökken, és kismértékben frekvenciafüggővé válik. A fénysebesség egyben a végső sebességhatár is. Semmilyen tárgy vagy jel nem képes ennél gyorsabban haladni.

Az f, a λ és a (vákuumbeli) c között az alábbi összefüggés áll fenn:

$$\lambda f = c \tag{2.4}$$

Mivel c konstans, f ismeretében λ meghatározható, és ez fordítva is igaz. Például egy 100 MHz-es jel hullámhossza kb. 3 m, egy 1 GHz-es jel hullámhossza 30 cm és egy 10 cm hullámhosszú jel frekvenciája pedig 3 GHz.

Az elektromágneses spektrum a 2.10. ábrán látható. A rádióhullám, a mikrohullám, az infravörös hullám és a látható fény a spektrumnak az a része, amely amplitudó-, frekvencia- vagy fázismoduláció révén alkalmas információtovábbításra. Az ultraibolya, a röntgen- és a gamma-sugarak a nagyobb frekvencia miatt még jobbak lennének, de ezeket nehéz előállítani és modulálni, nem terjednek jól az épületekben, és veszélyesek az élővilágra. A 2.10. ábra alján található sávokat az ITU által megadott hivatalos elnevezésekkel illettük. Az LF sáv hullámainak hullámhossza 1 és 10 km között van (a megfelelő



2.10. ábra. *Az elektromágneses spektrum és felhasználása a távközlésben*

frekvenciatartomány kb. 30 kHz-től 300 kHz-ig terjed). Az LF, az MF és a HF rövidítés a kisfrekvenciás (Low Frequency), a középfrekvenciás (Medium Frequency), illetve nagyfrekvenciás (High Frequency) hullámokat jelenti. Persze, amikor az elnevezések születtek, akkor még senki nem gondolt arra, hogy a 10 MHz-es tartomány fölé menjen, így az ennél magasabb sávokat *Very, Ultra, Super, Extremely* és *Tremendously High* frekvenciasávoknak nevezték el. Ezek fölött már nincsen neve a sávoknak, pedig a *hihetetlenül* (Incredibly), a *megdöbbentően* (Astonishingly) és a *bámulatosan* (Prodigiously) nagy frekvencia (IHF, AHF és PHF) elnevezések nem hangoznának rosszul.

A Shannon-egyenletből (2.3) tudjuk, hogy az elektromágneses hullámmal továbbítható információ mennyisége függ a vett jel teljesítményétől és arányos a sávszélességgel. A 2.10. ábrából kiderül, hogy a hálózatos szakemberek miért szeretik annyira az üvegszálakat. Sok GHz-nyi sávszélesség áll rendelkezésre adatátvitelhez a mikrohullámú tartományban, és még több az üvegszálban, hiszen ez a logaritmikus skálánkon még távolabb jobbra helyezkedik el. Nézzük meg például a 2.7. ábrán látható 1,3 mikronos sávot, ami 0,17 mikron szélességű. Ha a (2.4) egyenletet használjuk, hogy megállapítsuk a kezdő- és végfrekvenciákat a kezdő- és véghullámhosszokból, akkor arra az eredményre jutunk, hogy a frekvenciatartomány megközelítőleg 30 000 GHz. Elfogadható, 10 dB-es jel/zaj viszonnyal ez 300 Tb/s.

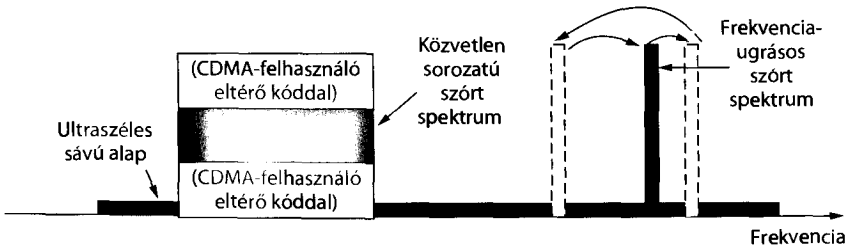
A legtöbb átvitel viszonylag keskeny frekvenciasávot használ (vagyis $\Delta f / f \ll 1$). A jeleiket erre a keskeny frekvenciasávra koncentrálják, hogy minél hatékonyabban kihasználják az adott spektrumot és elfogadható adatsebességet érjenek el elegendő teljesítményű átvitellel. Ennek ellenére egyes esetekben széles sávot használnak, három különböző változatban. A **frekvenciaugrásos szórt spektrumú (Frequency Hopping Spread Spectrum, FHSS)** átvitel esetén az adó frekvenciáról frekvenciára ugrál, másodpercenként több százszor. Ez a módszer népszerű a katonai rendszerekben, mivel nehézzé teszi az adások felderítését, és szinte lehetetlen a zavarásuk. Jó ellenállást mutat a többutas jelgyengüléssel (multipath fading) és a keskeny sávú interferenciával szemben is, mivel a vevő nem ragad elég hosszú időre egy gyenge frekvencián ahhoz, hogy befejezze a kommunikációt. Ez a robusztusság hasznos a spektrum zsúfolt részei számára, például az ISM-sáv számára is, amelyet röviden ismertetünk. Ezt a módszert kereskedelmi rendszerekben is alkalmazzák, például mind a 802.11 korábbi verziói, mind a Bluetooth ezt a megoldást használja.

Érdekes háttértörténet, hogy ennek a módszernek az egyik feltalálója az osztrák születésű szexbomba, Hedy Lamarr volt. Ő volt az első olyan nő, aki meztelenül tűnt fel a mozivásznon (az *Extázis* című 1933-as cseh filmben). Az első férje egy fegyvergyáros volt, aki elmesélte neki, hogy milyen könnyen lehet zavarni azokat a rádiójeleket, amelyekkel akkoriban a torpedókat irányították. Amikor felfedezte, hogy a férje Hitlernek ad el fegyvereket, teljesen elborzadt, és szobalánynak álcázva magát elszökött férjétől. Hollywoodba menekült, hogy ott folytassa filmszínésznői karrierjét. A szabadidejében feltalálta a frekvenciaugrást, hogy segítse az amerikai háborús erőfeszítéseket. Az ő elrendezése 88 frekvenciát használt, ami a zongora billentyűinek (és frekvenciáinak) száma. A találmányért ő és a barátja, George Antheil zeneszerző megkapták az Egyesült Államok 2 292 387-es számú szabadalmát. Az amerikai haditengerészetet ennek ellenére sem sikerült meggyőzniük arról, hogy a találmányuknak gyakorlati haszna is van, ezért sosem kaptak jogdíjat a szabadalom után. A szabadalom csak évekkel a lejárata után vált népszerűvé.

A szórt spektrum másik formája a **közvetlen sorozatú szórt spektrum (Direct Sequence Spread Spectrum, DSSS)**, amely egy széles frekvenciasávon teríti szét a jelet. Ez a megoldás egyre népszerűbb az üzleti világban, mint spektrálisan hatékony mód arra, hogy több jel ugyanazon a frekvenciasávon osztozzon. Ezekhez a jelekhez különböző kódokat rendelnek, ennek a módszernek a neve **CDMA (Code Division Multiple Access – kódosztásos többszörös hozzáférés)**, amire ebben a fejezetben még később visszatérünk. A 2.11. ábrán ez a módszer látható a frekvenciaugrással összehasonlítva. Ez az alapja a 3G mobiltelefon-hálózatoknak és a GPS (Global Positioning System – globális helymeghatározó rendszer) is ezt használja. A közvetlen sorozatú szórt spektrum, mint a frekvenciaugrásos szórt spektrum, még különböző kódok nélkül is képes tolerálni a keskeny sávú interferenciát és a többutas terjedés miatti jelgyengülést, mivel a szükséges jelnek csak egy töredéke veszik el. Néhány korábbi, 802.11b vezeték nélküli LAN is ezt a megoldást használja. A szórt spektrumú kommunikáció történetének lebilincselő és részletes leírását lásd Scholtz [1982] művében.

A harmadik, széles sávú kommunikációs módszer az **UWB (Ultra-WideBand – ultraszéles sáv)** kommunikáció. Az UWB gyors impulzusok sorozatát küldi, váltogatva azok pozícióját az információ továbbítása érdekében. A gyors átmenet olyan jelet eredményez, amely nagyon széles frekvenciasávban terjed, ritka eloszlásban. Az UWB-t olyan jelekként definiálják, amelynek a sávszélessége legalább 500 MHz vagy a jelek középfrekvenciájához tartozó frekvenciasáv legalább 20%-a. Az UWB is látható a 2.11. ábrán. Ezzel a nagy sávszélességgel az UWB-nek lehetősége van nagy sebességű kommunikációra. Mivel széles frekvenciasávon terjed, jelentős mennyiségű, más keskeny sávú jelektől származó, viszonylag erős interferenciát tud tolerálni. Legalább ennyire fontos, hogy nem okoz káros interferenciát más keskeny sávú rádiójeleknél, mivel az UWB nagyon kis energiát közöl egy adott frekvencián rövid távú átvitel esetén. Erre mondják azt, hogy elfér más jelek alatt (**underlay**). A békés együttélésnek köszönhetően az alkalmazások a vezeték nélküli PAN-okban akár 1 Gb/s sebességgel működnek, azonban a kereskedelmi siker vegyes. Használható továbbá szilárd objektumokon (földön, falon és emberi testeken) átlátó képalkotó rendszerekben, illetve precíz helymeghatározó rendszerekben.

Most (a rádióval kezdve) azt fogjuk megtárgyalni, hogyan használják a 2.11. ábra elektromágneses spektrumának egyes részeit. Azt feltételezzük, hacsak másképp nem állítjuk, hogy minden átvitel keskeny frekvenciasávot használ.



2.11. ábra. *Szórt spektrumú és ultraszéles sávú (UWB) kommunikáció*

2.3.2. Rádiófrekvenciás átvitel

A rádióhullámok egyszerűen előállíthatók, nagy távolságra jutnak el, és könnyen áthatolnak az épületek falain, így széles körben használják ezeket mind kültéri, mind beltéri alkalmazásokban. A rádióhullámok minden irányba terjednek, így az adót és a vevőt nem kell fizikailag precízen egymáshoz illeszteni.

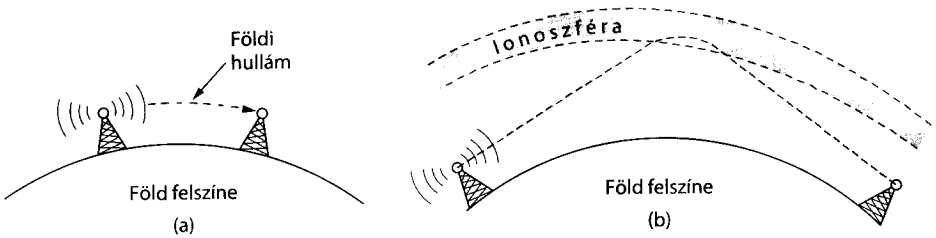
Legtöbbször jó, hogy a rádióhullámok minden irányba terjednek, de van, amikor ez problémát jelent. Az 1970-es években a General Motors elhatározta, hogy az új Cadillacek fékrendszerébe számítógéppel vezérelt blokkolásgátlót épít be. Amikor az autó vezetője rálépett a fékpedálra, a számítógép folyamatosan megnyomta és elengedte a féket, így az autó kerekei nem blokkoltak. Egy szép napon egy ohiói autópályarendőr rádiótelefonján felhívta a központot, és arra lett figyelmes, hogy a mellette haladó Cadillac hirtelen úgy elkezdett ugrálni, mint egy bakkecske. Amikor a rendőr félreállította az autót, a sofőr mentegetőzött, hogy ő nem csinált semmit, a kocsija viszont meghibbant.

Végül kezdett tisztázódni a kép: a Cadillacek időnként megvadultak, de csak Ohio jelentősebb autópályáin, és csak akkor, amikor az autópálya-rendőrség szolgálatban volt. A General Motors sokáig nem értette, hogy miért nincs semmi gond a Cadillacekkel más államokban és Ohio alacsonyabb rendű útjain. Hosszas kutatás után rájöttek arra, hogy a Cadillacben levő vezetékek olyan antennaként működnek, amelyek az ohiói autópálya-rendőrség új rádiós rendszerének frekvenciájára érzékenyek.

A rádióhullámok terjedési tulajdonságai frekvenciafüggők. Kis frekvencián a rádióhullámok minden akadályon áthatolnak, viszont a teljesítményük a forrástól távolodva erősen – a levegőben nagyjából $1/r^2$ szerint – csökken, mivel a jel energiájából nagyobb felületen felületegységre kevesebb jut. Ezt a csillapítást **szakaszveszteségnek (path loss)** nevezzük. A nagyfrekvenciás rádióhullámok egyenes vonal mentén terjednek, és a tárgyakról visszaverődnek. A szakaszveszteség szintén csökkenti a teljesítményt, habár a vételi jel erősen függhet a visszaverődéstől is. Az eső és egyéb akadályok jobban elnyelik a nagyfrekvenciás rádióhullámokat, mint a kisfrekvenciásokat. A rádióhullámokat a villamos motorok és más elektronikus berendezések minden frekvenciatartományban zavarják.

Érdekes összehasonlítani a rádióhullámok csillapodását a vezetékes közegben továbbított jelek csökkenésével. Üvegszál, koax és sodrott érpár esetén a jelerősség távolságegységenként ugyanannyival csökken, például sodrott érpár esetén 100 méterenként 20 dB-lel. A rádió esetében a jelerősség csökkenése a távolság kétszeresével arányos, például szabad térben 6 dB-lel duplázódásonként. Mivel a rádióhullámok nagyon messzire eljutnak, ezért komoly problémát jelent a felhasználók közötti interferencia. Emiatt minden országban szigorúan engedélyhez kötik a rádióadóval ellátott eszközök használatát, néhány eset kivételével, amelyekről később lesz szó ebben a fejezetben.

A VLF-, LF- és MF-frekvenciasávokban a rádióhullámok a 2.12.(a) ábrán látható módon a földfelszínt követik. Ezeket a hullámokat akár 1000 km távolságra is venni lehet kisebb frekvenciák esetén. Nagyobb frekvenciákon a hatótávolság csökken. Az AM-rádióadások az MF-sávot használják, ezért nem lehet tisztán fogni a bostoni rádiók adásait New Yorkban. Ebben a sávban a rádióhullámok átjutnak az épületek falain, ezért tudjuk a zsebrádiót lakásunkban is hallgatni. Ezek a sávok azért nem alkalmasak adatkommunikációra, mert viszonylag kicsi az általuk biztosított sávszélesség [lásd (2.3) egyenlet].



2.12. ábra. *(a) A VLF-, az LF- és az MF-sávban a rádióhullámok követik a Föld felszínének a görbületét. (b) A HF-sávban a rádióhullámok visszaverődnek az ionoszféráról*

A HF- és a VHF-sávokban a földközeli hullámokat a földfelszín kezdi elnyelni. Azok a hullámok viszont, amelyek eljutnak az ionoszféráig, a 2.12.(b) ábrán látható módon visszaverődnek a földre. (Az ionoszféra a földfelszín felett 100 és 500 km közötti magasságban található légréteg, amelyben elektromosan töltött részecskék mozognak.) Bizonyos légköri feltételek mellett a hullámok többször is visszaverődhetnek. Az amatőr rádiósok ezeket a sávokat használják nagy távolságú beszélgetéseikhez. A hadsereg szintén használja a HF- és a VHF-sávot.

2.3.3. Mikrohullámú átvitel

100 MHz felett a hullámok szinte teljesen egyenes vonalban terjednek, és így jól fókuszálhatók. Ha a teljes energiát egy kicsi nyalábba sűrítjük egy parabolaantenna (mint az ismerős műholdas tv-antenna) használatával, akkor jelentősen megnő a jel/zaj arány, de az adó és a vevő antennáit nagyon pontosan kell egymás felé irányítani. Ez az irányítottság még azt is lehetővé teszi, hogy több egymás mellett elhelyezett adó interferencia nélkül kommunikáljon több egymás mellett levő vevővel, ha néhány, minimális távolságtartási szabályt betartanak. Az üvegszálak előtt évtizedekig ezek a mikrohullámok jelentették a nagy távolságú telefonátvitel lelkét. Az MCI (amely az AT&T egyik első versenytársa volt a piac felszabadítása után) teljes rendszere az egymástól néhányszor tíz kilométerre levő tornyok között folyó mikrohullámú kommunikációra épült. Még a cég neve is erre utalt (Microwave Communications, Inc., MCI – Mikrohullámú Távközlési Vállalat). Az MCI azóta már átállt az üvegszálas megoldásokra, és – cégegyesülések és csődök hosszú sorozatán keresztülmenve – a távközlési piac átrendeződése során a Verzion része lett.

Mivel a mikrohullámok egyenes vonal mentén terjednek, ezért a földfelszín görbülete problémát jelent, ha az adótornyok túlságosan messze vannak egymástól. (Gondoljunk csak egy San Francisco és Amsterdam közötti kapcsolatra.) Ezért meghatározott távolságonként ismétlőkre van szükség. Minél magasabbak az adótornyok, annál messzebbre lehetnek egymástól. Az ismétlők egymástól mért távolsága durván az adótornyok magasságának négyzetgyökével arányos. Ez azt jelenti, hogy 100 m magas tornyok esetén az ismétlőket egymástól 80 km távolságra lehet telepíteni.

A kisfrekvenciás rádióhullámokkal szemben a mikrohullámok nem képesek áthatolni az épületek falain. Ráadásul, az adóegység hiába fókuszálja jól a mikrohullámú sugarakat, azok a levegőben mindenképpen szóródnak valamennyire. A hullámok egy kis része

megtörhet az alacsonyabb légköri rétegeknél, ezek a hullámok valamivel később érnek célba, mint a közvetlen beérkező hullámok. A megtört hullámok fázisa nem egyezik meg a közvetlen beérkező hullámokéval, így ezek akár ki is olthatják egymást. Ez a jelenség, a **többutas jelgyengülés (multipath fading)**, sokszor komoly gondot okoz. A jelgyengülés függ az időjárástól és a frekvenciától. Egyes szolgáltatók a csatornáik 10%-át készenlétben tartják arra az esetre, ha az elgyengülés időlegesen tönkretenné valamelyik frekvenciasávot.

Az egyre szélesebb sávok iránti igény egyre nagyobb frekvenciák használatára készteti az üzemeltetőket. A 10 GHz-ig terjedő sávok használata mindennapos, de körülbelül 4 GHz-nél egy új probléma merül fel: a víz elnyeli a sugarakat. Ezek a hullámok csak néhány centiméter hosszúak, és ezeket elnyeli az eső. Ez a hatás igen kiváló lenne, ha az ember egy hatalmas szabadtéri mikrohullámú sütő építését tervezné az arra repülő madarak megsütésére, de a kommunikáció számára súlyos probléma. Hasonlóan a többutas jelgyengüléshez, az egyetlen megoldás itt is az, hogy az esős területeken keresztülmenő kapcsolatokat leállítják, és a forgalmat más irányba terelik.

Összefoglalva az eddigieket, a mikrohullámú átvitelt olyan széles körben használják a nagy távolságú távbeszélőrendszerekben, a mobiltelefon-hálózatokban, a televíziós műsorszórásban és még sok más területen, hogy komoly frekvenciahiány lépett fel. A fénykábellel szemben ugyanis számos előnye van. A legfontosabb talán az, hogy nem kell áthaladási engedélyt szerezni. Bőven elég 50 km-enként egy kis földdarabot megvenni, oda egy mikrohullámú adótornyot építeni, és a telefonrendszert átvezetve rajta közvetlenül is tudunk kommunikálni. Ennek a módszernek köszönheti az MCI, hogy olyan gyorsan fejlődött, amikor nagy távolságú telefonszolgáltatással kezdett el foglalkozni. (A Sprint más utat választott. Ezt a céget a Southern Pacific Railroad hozta létre, amely nagy mennyiségű útvonal birtokában volt, és a fényvezető kábeleket egyszerűen az utak mellé a földbe fektette.)

A mikrohullámú technika viszonylag nem drága. Két egyszerű adótorony felépítése (ami akár egy négy huzallal kifeszített oszlop is lehet) és egy-egy antenna ráhelyezése olcsóbb lehet, mint 50 kilométernyi fényvezető kábel lefektetése egy zsúfolt városrészben vagy a hegyekben. Még a telefontársaságoktól bérelt fényvezető kábeleknél is olcsóbb, különösen akkor, ha a telefontársaságnak még nem fizették ki azoknak a rézvezetékeknek a teljes árát, amelyeket fényvezető kábelekre cserélt le.

Az elektromágneses spektrum politikai vonatkozásai

A teljes fejetlenség elkerülése érdekében országos és nemzetközi egyezmények szabályozzák, hogy ki milyen frekvenciát használhat. Mivel mindenki nagyobb adatsebességet szeretne elérni, mindenki szélesebb spektrumot akar. Az egyes országok kormányai adják ki a sávokat az AM- és FM-rádiók, a tv-k és a mobiltelefonok számára, valamint a telefontársaságok, a rendőrség, a hajózás, a navigáció, a hadsereg, a kormány és más, egymással versengő felhasználók számára. A világ egyes országai között az ITU-R egyik ügynöksége, a WARC próbálja összehangolni ezt a frekvenciakiosztást, hogy olyan készülékeket lehessen gyártani, amelyek több országban is működnek. Az egyes országokra azonban nem kötelező érvényűek az ITU-R javaslatai, néhány alkalommal már az FCC (Federal Communications Commission – szövetségi kommunikációs bizottság),

az Egyesült Államok frekvenciakiosztásáért felelős szervezete is elutasította az ITU-R javaslatait (általában azért, mert valamelyik politikailag erős csoporttól kellett volna elvenni a spektrum egy részét).

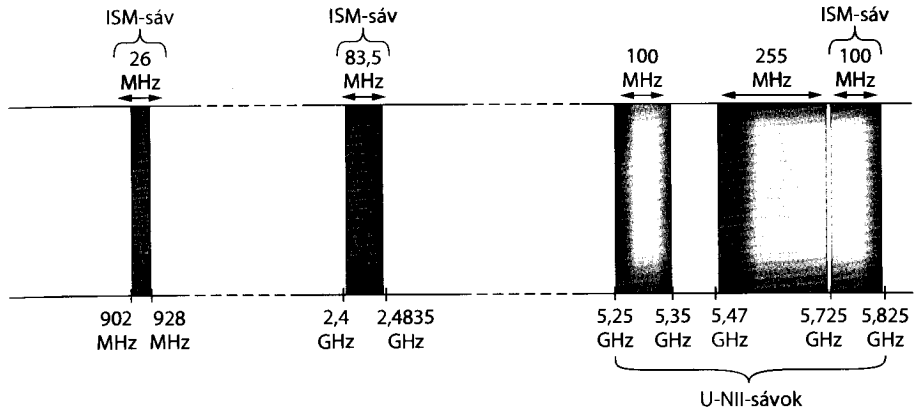
Amikor a spektrum egy darabját már egy bizonyos felhasználásra (például mobiltelefonok) kijelölték, még mindig nyitva áll az a másik kérdés, hogy az egyes szolgáltatók mely frekvenciákat használhatják. Ennek a problémának a megoldására eddig háromféle algoritmust alkalmaztak. A legrégibb, amelyet gyakran **szépségversenynek** hívnak, abból áll, hogy minden szolgáltató elmagyarázza azt, hogy szerinte miért az ő javaslata szolgálja leginkább a közérdeket, majd a kormányhivatalnokok eldöntik, hogy a szép történetek közül melyik tetszett nekik a legjobban. Az, hogy egy kormányhivatalnok több milliárd dollárt érő jogokat ad a kedvenc vállalatának, gyakran vezet megvesztegetéshez, korrupcióhoz, rokonok előnyben részesítéséhez vagy még ezeknél is rosszabb dolgokhoz. Továbbá, még egy olyan, egyébként teljesen őszinte és lelkiismeretes kormányhivatalnoknak is sok magyarázkodnivalója lenne, aki szerint egy külföldi cég a helyi cégek közül bármelyiknél jobban tudná elvégezni a munkát.

Ez a megfigyelés vezetett a 2. algoritmushoz, amelyben **sorsolást** tartanak az érdeklődő vállalatok között. Ezzel az ötlettel az a baj, hogy olyan cégek is részt vehetnek a sorsoláson, akik egyáltalán nem akarják használni a spektrumot. Ha mondjuk egy gyorsétterem vagy egy cipőbolthálózat nyeri meg a sorsolást, akkor nagy profittal és kockázatmentesen eladhatja a jogokat egy hálózati szolgáltatónak.

Sokan kritizálták ezt a módszert azért, hogy ilyen nagy hasznot ad át éber, de egyébként teljesen véletlenül kiválasztott vállalatoknak. Ez a kritika vezetett a 3. algoritmushoz: a sávszélességet **elárverezik**, és a legtöbbet ajánló cégnek adják oda. Amikor Angliában, 2000-ben elárverezték a harmadik generációs mobiltelefon-rendszerekhez használható frekvenciákat, körülbelül 4 millió dollár bevételre számítottak. A licit végén 40 millió dollárt kaptak, miután a szolgáltatók vadul egymásra kínáltak. Mind halálosan meg volt rémülve attól, hogy lekési a mobiltelefon-piac hajóját. Ez az esemény kapzsi gondolatokat ébresztett a közeli kormányokban is, és arra ösztönözte őket, hogy ők is árveréseket rendezzenek. A módszer működött, de néhány szolgáltatót akkora adósságba juttatott, hogy most a csőd szélén állnak. Még a legjobb esetekben is sok évig fog tartani, amíg a jogosítványok díja megtérül.

A frekvenciák szabályozásának egy teljesen más megközelítése az, hogy egyáltalán nem szabályozzuk azokat. Mindenkit hagyjunk úgy adni, ahogy akar, de szabályozzuk az adáshoz használható teljesítményt, így az állomások hatótávolsága lerövidül, és nem alakul ki közöttük interferencia. Ennek megfelelően a legtöbb kormány néhány olyan frekvenciasávot tart fenn, amelyek használatát nem köti engedélyhez. Ezek az úgynevezett **ISM-(Industrial, Scientific, Medical – ipari, tudományos, orvosi)** sávok. A garázskapuk távirányítói, a vezeték nélküli telefonok, a rádióvezérelt játékok, a vezeték nélküli egerek és még számtalan más háztartási eszköz használja az ISM-sávokat. A nem koordinált berendezések közötti interferencia minimalizálására az FCC azt ajánlja, hogy minden, az ISM-sávokat használó eszköz korlátozza az adóteljesítményét (például 1 wattra), és használjon más technikákat a jelek szétszórására egy adott frekvenciatartományban. Az eszközöknek továbbá arra is ügyelniük kell, hogy elkerüljék az interferenciát a radarállomásokkal.

Az ISM-sávok pontos helye kissé eltérő az egyes országokban. Az Egyesült Államokban például az 1 watt alatti teljesítményű eszközök a 2.13. ábrán látható frekvenciasávokat használhatják az FCC engedélye nélkül. A 900 MHz-es sávot használta a 802.11 korai verziója, de ez a sáv zsúfolt. A 2,4 GHz-es sáv a legtöbb országban elérhető, a Bluetooth és a 802.11b/g vezeték nélküli LAN-ok közül jó néhány ebben a sávban működik, habár ebben a sávban a mikrohullámú sütők és a radarállomások interferenciát okoznak. A spektrum 5 GHz-es része tartalmazza az **U-NII (Unlicensed National Information Infrastructure – engedély nélküli nemzeti információs infrastruktúra)** sávokat. Az 5 GHz-es sávok viszonylag fejletlenek, de mivel itt a legnagyobb a sávszélesség és a 802.11a ezeket a sávokat használja, népszerűségük gyorsan nő.



2.13. ábra. *A vezeték nélküli eszközök által az Egyesült Államokban használt ISM- és U-NII sávok*

Az engedély nélküli sávok zajos siker volt az elmúlt évtizedben. A spektrum szabad használatának lehetősége rengeteg találmány megszületését eredményezte a vezeték nélkül LAN és PAN területén, létjogosultságát az olyan technikák széles körű elterjedése is igazolja, mint a 802.11 és a Bluetooth. Az újítások folytatásához még szélesebb spektrum szükséges. Az FCC 2009-es döntése egy izgalmas fejlemény az Egyesült Államokban, ami lehetővé teszi a 700 MHz körüli **üres helyek (white spaces)** engedély nélküli használatát. Az üres helyek olyan frekvenciasávok, amelyek kiosztásra kerültek, de helyileg nincsenek használatban. Az analógról a teljesen digitális televíziózásra történő 2010-es átállás üres helyeket szabadított fel 700 MHz körül az Egyesült Államokban. Az üres helyek használatának egyetlen nehézsége, hogy azoknak az eszközöknek, amelyeknek nincs engedélye, alkalmasnak kell lenniük a közelben lévő engedéllyel bíró adók észlelésére, beleértve a vezeték nélküli mikrofonokat is, amelyeknek elsőbbségük van a frekvenciasáv használatára

Egy másik nagy felbolydulás a 60 GHz-es sáv körül zajlik. Az FCC 2001-ben megnyitotta az 57 és 64 GHz közötti tartományt engedélyhez nem kötött műveletek számára. Ez a tartomány a spektrum hatalmas részét teszi ki, többet mint az összes ISM-sáv együttvéve, ezért alkalmas arra, hogy támogassa azokat a nagy sebességű hálózatokat, amelyek a nagy felbontású televízióadás levegőn át történő sugárzásához szükségesek a nappalinkban. 60 GHz-en az oxigén elnyeli a rádióhullámokat. Ez azt jelenti, hogy a jelek nem terjednek távolra, ezáltal kitűnően alkalmasak kis hatótávolságú hálózatokhoz. A nagy frekvenciák (a 60 GHz az EHF (Extremly High Frequency – extrém nagy frekvencia)

vagy „milliméteres" sávban található, éppen az infravörös sugárzás alatt) eleinte kihívást jelentettek a berendezések készítőinek, de a termékek ma már piacon vannak.

2.3.4. Infravörös átvitel

A vezeték nélküli infravörös hullámokat elsősorban a kis hatótávolságú kommunikációra használják előszeretettel. A televíziók, a videomagnók és a Hi-Fi-készülékek távirányítóiban mind infravörös hullámú adóegység található. Az infravörös hullám viszonylag jól irányítható, olcsó és könnyen előállítható. Van azonban egy óriási hátránya: szilárd testeken nem képes áthatolni. (Próbaképpen álljunk be a távirányító és a tv-készülék közé, és nézzük meg, hogy működik-e a távirányító.) Általánosságban azt mondhatjuk, hogy minél jobban közeledünk a kisfrekvenciás rádióhullámoktól a látható fény felé, a hullámok annál inkább fényhullámként, és annál kevésbé rádióhullámként viselkednek.

Mindezek ellenére előnyökkel is jár az a tény, hogy az infravörös hullámok nem tudnak áthatolni a falakon. Azt is jelenti ugyanis, hogy egy épület egyik szobájában működő infravörös rendszer és a szomszédos szobák vagy épületek rendszerei között nem lép fel interferencia: nem irányíthatjuk a szomszédjaink tv-jét a saját távirányítónkkal. Mindezen felül az infravörös rendszerek lehallgatási biztonsága éppen emiatt jobb a rádiós rendszerekénél. Az ISM-sávokon kívül üzemelő rádiós rendszerekkel ellentétben az infravörös rendszerek üzemeltetéséhez a fenti okok miatt nincsen szükség külön engedélyre. Az infravörös kommunikációnak korlátozott haszna van az asztalon, például összeköttetést biztosíthat egy noteszgép és egy nyomtató között az **IrDA (Infrared Data Association – Infravörös Adatátviteli Társaság)** szabványának megfelelően, de egyébként nem egy fontos szereplő a távközlés világában.

2.3.5. Látható fényhullámú átvitel

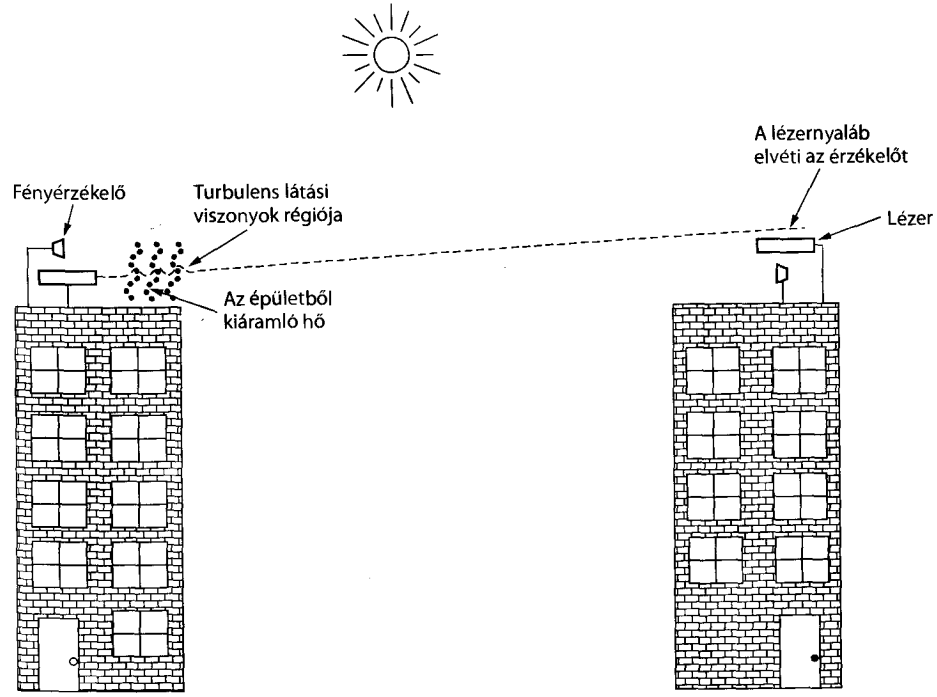
A vezeték nélküli fényjelzést vagy **szabadtéri optikai rendszereket** már évszázadok óta használják. Paul Revere nevezetes útja előtt bináris fényjeleket küldött a bostoni Old North Church tornyából. Ennek egy modern változata az, amikor két épület lokális hálózatát a tetejükre szerelt lézerek segítségével kapcsoljuk össze. A lézert alkalmazó optikai jelátvitel alapvetően egyirányú, így mindkét épületnek külön lézerforrásra és fényérzékelőre van szüksége. Ez a megoldás igen nagy sávszélességgel rendelkezik nagyon olcsón és viszonylag biztonságos, mert a keskeny lézersugarat nehéz letapogatni. Viszonylag egyszerű egy ilyen rendszert kiépíteni, és szemben a mikrohullámmal, nincs szükség az FCC engedélyére.

A nagyon keskeny lézersugár nem csak előnyös, hanem bizonyos tekintetben hátrányos is. Ahhoz, hogy egy 1 mm széles lézersugarat egy 500 m-re levő, 1 mm széles célra irányítsunk, Annie Oakley[1] célzóképességére lenne szükségünk. Annak érdekében,

1 A szerző itt egy amerikai céllövő bajnoknőre utal, aki céllövő tehetsége révén vált híressé, és aki lövöldözős show-műsorával az 1900-as évek elején az első amerikai női szupersztár lett. *(A lektor megjegyzése)*

hogy a lézersugarak kissé szórjanak, lencséket helyeznek a fény útjába. Hogy fokozzuk a nehézségeket, a szél és a hőmérséklet-ingadozás is torzíthatja a sugarat, továbbá a lézersugarak esőn és sűrű ködön nem képesek áthatolni, napsütésben viszont remekül működnek. Persze ezeknek a tényezőknek a többsége nem számít, amikor a cél két űrhajó összekapcsolása.

Könyvünk egyik szerzője (AST) jelen volt egyszer egy modern szállodában megrendezett európai konferencián, ahol a szervezők gondosan előkészítettek egy terminálszobát a résztvevők számára, hogy az unalmas előadások alatt e-leveleiket tudják olvasgatni. Mivel a helyi telefontársaság nem volt hajlandó három napra egy csomó vonalat kiépíteni, ezért a szervezők egy lézert tettek föl a tetőre, és megcélozták vele a néhány kilométerre levő egyetem számítógépközpontjának épületét. A konferencia előtti estén az egészet kipróbálták, és tökéletesen működött. Másnap reggel 9 órakor, verőfényes napsütésben a kapcsolat teljesen megszűnt, és egész nap nem működött. Ez a jelenség a következő két napon ugyanúgy megismétlődött. A konferencia után a szervezők rájöttek a probléma nyitjára. A tűző nap annyira felmelegítette a tetőt, hogy megindult egy felfelé irányuló hőáramlás, ahogy ez a 2.14. ábrán látható. Ez a turbulens áramlás eltérítette a lézersugarakat, és a detektor előtt táncoltatta azokat, a hőségben vibráló aszfalthoz hasonlóan. A tanulság az, hogy a vezeték nélküli optikai kapcsolatot megfelelő hibahatárral kell tervezni ahhoz, hogy ideális és kevésbé ideális körülmények között is jól működjön.



2.14. ábra. *A hőáramlások megzavarhatják a lézeres távközlési rendszerek működését. Az ábra olyan kétirányú rendszert mutat, amelyben két lézerforrás található*

A vezeték nélküli optikai távközlés ma még egzotikus hálózati technológiának tűnhet, de hamarosan sokkal elterjedtebbé válhat. Körülvesznek bennünket fényt érzékelő kamerák, fényt kibocsátó LED-es megjelenítők vagy kijelzők (display). A kommunikáció ezen kijelzők fölötti rétegbe helyezhető azáltal, hogy az információt olyan mintákba kódoljuk, amelyek LED-eket ki-be kapcsolgatnak úgy, hogy az az emberi érzékelés küszöbszintje alatt van. A látható fénnyel történő kommunikáció ilyen formában eredendően biztonságos és egy kis sebességű hálózatot képez a kijelző közvetlen környezetében. Ezáltal a mindenütt jelen lévő számítástechnika sokféle esetét képzelhetjük el. A megkülönböztető jelzést használó járművek villogó fénye figyelmeztethetné a közeli közlekedési lámpákat és járműveket, hogy segítsenek megtisztítani az utat. A tájékoztató jelzések térképet sugározhatnának. Még a karácsonyi fények is olyan dalokat játszhatnának, amelyek összhangban vannak a kijelzőjükkel.

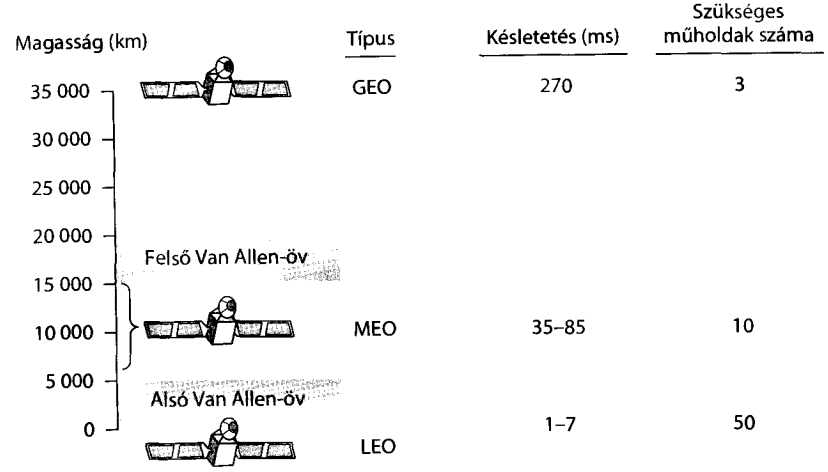
2.4. Kommunikációs műholdak

Az 1950-es években és az 1960-as évek elején olyan kommunikációs rendszereket próbáltak kialakítani, amelyekben a fémborítású meteorológiai léggömbök verték volna vissza a jeleket. Sajnos a vett jelek túl gyengék voltak ahhoz, hogy ezt a rendszer a gyakorlatban is használni lehetett volna. Valamivel később az amerikai haditengerészet felfedezte, hogy az égen van egy állandó gömb – a Hold –, amit a léggömbök helyett használhatnak. A Holdról visszaverődő jelekre alapozva megépítettek egy ténylegesen is működőképes rendszert a hajók és a part közötti kommunikáció számára.

Az égi kommunikáció fejlődésének következő lépésére az első kommunikációs műhold fellövéséig kellett várni. A kulcsfontosságú különbség egy mesterséges és egy igazi hold között az, hogy a mesterséges hold fel tudja erősíteni a jeleket, mielőtt visszaküldi azokat a földre.

A kommunikációs műholdak rendelkeznek néhány olyan érdekes tulajdonsággal, amelyek több alkalmazás számára is vonzóvá teszi azokat. A kommunikációs műholdat a legegyszerűbb úgy elképzelni, mint egy hatalmas mikrohullámú ismétlőt az űrben. A műholdak jó néhány **transzponderrel** (**transponder**) rendelkeznek, amelyek közül mindegyik a spektrum egy részét figyeli, felerősíti a bejövő jelet, és azután egy másik frekvencián küldi vissza, elkerülve ezzel a bejövő jellel való interferenciát. Ezt az üzemmódot **hajlított cső** (**bent pipe**) módnak is hívják. Digitális feldolgozással kiegészíthető annak érdekében, hogy az adatáramot a teljes sávban külön módosítani lehessen vagy vissza lehessen irányítani, vagy a műhold akár digitális információt is fogadhasson, és adatszórással visszaküldhessen. A jelek ilyen formában történő regenerálása javítja a teljesítőképességet a hajlított cső módhoz viszonyítva, mert a műhold nem erősíti fel a felfelé menő jelben lévő zajt. A lefelé menő nyalábok lehetnek szélesek és beteríthetik a földfelszín jelentős részét, vagy lehetnek keskenyek, így a beterített földfelszín átmérője csak néhány száz kilométer.

Kepler törvényének értelmében egy műhold keringési ideje a pálya sugarának 3/2-edik hatványával arányos, vagyis minél magasabban van a műhold, annál hosszabb a keringési periódus. A Föld felszínéhez közel a periódusidő körülbelül 90 perc, így az alacsony



2.15. ábra. *Kommunikációs műholdak és néhány tulajdonságuk, köztük a földfelszín feletti magasság, az oda-vissza út késleltetése és a teljes földfelület lefedéséhez szükséges műholdak száma*

pályás műholdak elég gyorsan eltűnnek szem elől, és ezért sokra van belőlük szükség ahhoz, hogy állandó lefedettséget biztosíthassunk. Körülbelül 35 800 km-es magasságban a periódus éppen 24 óra. Körülbelül 384 000 km-es magasságban a periódus nagyjából egy hónap, amint azt a Hold bármely rendszeres megfigyelője is tanúsíthatja.

A műholdak keringési ideje fontos, de nem az egyetlen tényező, ami meghatározza az elhelyezésük magasságát. Egy másik befolyásoló tényező a Van Allen-övek léte, amelyek a Föld mágneses mezejének fogságában rekedt erős töltésű részecskék rétegei. Egy itt keringő műholdat valószínűleg gyorsan elpusztítanának ezek a nagy energiájú, töltéssel rendelkező részecskék, amelyeket a Föld mágneses mezeje tart fogva. Ezek a tényezők három olyan térséget különítenek el, ahová biztonságosan lehet műholdat telepíteni. Ezek a térségek és néhány tulajdonságuk látható a 2.15. ábrán. A most következő részben röviden bemutatjuk az egyes térségekben elhelyezett műholdakat.

2.4.1. Geostacionárius műholdak

Arthur C. Clarke tudományos-fantasztikus író 1945-ben kiszámolta, hogy egy egyenlítő körüli körpályán 35 800 km-es magasságban keringő műhold mozdulatlannak tűnne az égen, így szükségtelenné válna a követése [Clarke, 1945]. Ezután leírt egy teljes kommunikációs rendszert, amely ezeket a (személyzettel ellátott) **geoszinkron** vagy **geostacionárius** (**geostationary** – **Földhöz képest mozdulatlan**) műholdakat használta. A rendszer leírását Clarke a röppályákkal, a napelemekkel, a rádiófrekvenciákkal és a kilövési eljárással tette teljessé. Sajnálatos módon arra a következtetésre jutott, hogy a műholdak nem használhatók a gyakorlatban, mivel lehetetlen nagy energiaigényű és törékeny vákuumcsöves erősítőket Föld körüli pályára állítani. Így aztán nem is foglalkozott tovább ezzel az ötlettel, de még írt róla néhány tudományos-fantasztikus történetet.

A tranzisztor feltalálása teljesen megváltoztatta ezt a helyzetet. Az első kommunikációs műholdat, a Telstart, 1962 júliusában bocsátották fel. Azóta a kommunikációs műholdak piaca többmilliárd dolláros üzletté vált, és a külső világűr vonatkozásában ez az egyetlen, amely óriási profitot hozóvá vált. Ezeket a magasan repülő műholdakat gyakran **GEO-** (**Geostationary Earth Orbit – Földhöz képest mozdulatlan pályájú**) műholdaknak is hívják.

Amennyiben az interferenciának elejét szeretnénk venni, napjaink műszaki lehetőségei mellett nem bölcs dolog 2 foknál kisebb távolságot tartani az olyan geostacionárius műholdak között, amelyek az egyenlítő síkjában vannak. 2 fokos térközzel csak 360/2 = 180 ilyen műholdat tudunk egyszerre elhelyezni az Egyenlítő körül. A rendelkezésre álló sávszélesség növelése érdekében azonban minden transzponder több frekvenciát és polarizációt is használhat.

A teljes égi káosz eluralkodását az ITU akadályozza meg, a keringési pályák (orbit) egyes állásainak (slot) kiosztásával. Ez a folyamat nagymértékben politikai, olyan országok is követelik a „nekik járó" műholdhelyeket (hogy bérbe adhassák azokat a legjobb ajánlat megtevőjének), amelyek még csak mostanában nőttek ki a kőkorszakból. Más országok ezzel szemben azt az álláspontot támogatják, hogy a nemzeti tulajdoni jogok nem terjednek ki a Holdig, és így egyetlen ország sem rendelkezhet az országa feletti műholdpozíciók tulajdoni jogával. A csatározásokat csak tovább erősíti, hogy a kereskedelmi célú kommunikáció nem az egyetlen alkalmazás. A tv-adók, a kormányok és a hadsereg is akarnak egy-egy szeletet a Föld körüli pályák tortájából.

A modern műholdak elég nagyok is lehetnek, a súlyuk 4000 kg-ig terjed, és több kilowatt elektromos teljesítményt is felvehetnek a napelemeikről. A Nap, a Hold és a Föld tömegvonzása hajlamos elmozdítani a műholdakat a kijelölt helyükről, de ezt a hatást kis fedélzeti rakétahajtóművekkel ellensúlyozzák. Ezt a finomhangolási tevékenységet **pozicionálásnak** (**station keeping**) hívják. Amikor azonban a hajtóművek üzemanyaga körülbelül tíz év elteltével kifogy, a műhold tehetetlenül sodródni és billegni kezd, és ezért ki kell kapcsolni. Egy idő után a pálya instabillá válik, majd a műhold belép a légkörbe, és ott elég vagy esetleg a felszínre zuhan.

Nem egyedül a keringési pályákon rendelkezésre álló helyekért folyik a verseny. A frekvenciák is gyakran vita tárgyát képezik, mivel a lefelé irányuló adások és a földi mikrohullámú adások között interferencia léphet fel. Az ITU ezért elkülönített bizonyos frekvenciasávokat a műholdas felhasználók számára. A legfőbb sávokat a 2.16. ábrán soroltuk fel. A C sáv volt az első, amelyet a kereskedelmi műholdas rendszer számára jelöltek ki. A sáv két

Sáv	Lefelé irányuló	Felfelé irányuló	Sávszélesség	Problémák
L	1,5 GHz	1,6 GHz	15 MHz	Kis sávszélesség, zsúfolt
S	1,9 GHz	2,2 GHz	70 MHz	Kis sávszélesség, zsúfolt
C	4,0 GHz	6,0 GHz	500 MHz	Földi interferencia
Ku	11 GHz	14 GHz	500 MHz	Eső
Ka	20 GHz	30 GHz	3500 MHz	Eső; eszközök költsége

2.16. ábra. *A legfontosabb műholdas frekvenciasávok*

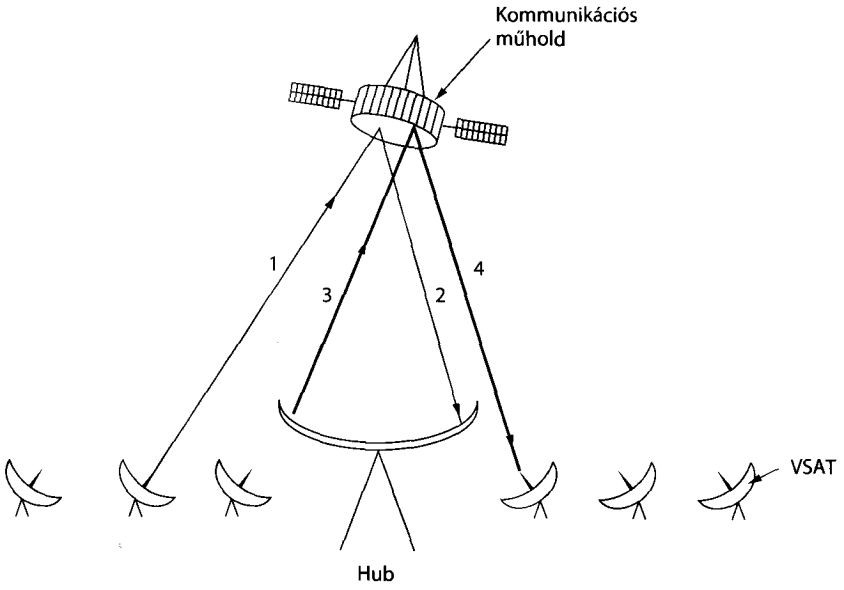
frekvenciatartományt tartalmaz, a lefelé haladó (a műholdról érkező) forgalom használja az alsó sávot, a felfelé haladó (a műholdra küldött) forgalom pedig a felső sávot. Annak érdekében, hogy a forgalom mindkét irányba egyszerre mehessen, két egyirányú csatornát kell használni. A C sávok már a műholdak nélkül is túlterheltek, mivel a földi továbbítású mikrohullámú kapcsolatok is ezeket használják. A következő két sávot (L és S) egy nemzetközi egyezmény alapján jelölték ki 2000-ben, azonban ezek keskenyek és túlterheltek.

A kereskedelmi telekommunikációs szolgáltatók által használható következő frekvenciasáv a Ku (K under – K alatt) sáv. Ez a sáv (még) nem zsúfolt, és az ezeken a frekvenciákon működő műholdakat akár egymástól 1 fok távolságra is lehet telepíteni. Felmerül azonban egy másik probléma is: az eső. A víz kiválóan el tudja nyelni ezeket a rövid mikrohullámokat. Szerencsére a nagy viharok általában helyhez kötöttek, ezért a probléma megkerülhető azzal, hogy több, egymástól nagy távolságra levő földi állomást telepítünk egyetlen helyett. Ennek a megoldásnak azonban az az ára, hogy több antennára, kábelre és elektronikára van szükség ahhoz, hogy gyorsan lehessen állomást váltani. A kereskedelmi műholdas forgalom számára még egy frekvenciasávot kijelöltek, ez a Ka (K above – K felett). Az ebben a sávban használható eszközök azonban egyelőre drágák. Ezeken a kereskedelmi sávokon kívül sok kormányzati és hadi sáv is létezik.

Egy modern műholdon körülbelül 40 transzponder van, amelyek közül mindegyik 80 MHz-es sávszélességgel rendelkezik. Általában minden transzponder hajlított csőként üzemel, de a legújabb műholdaknak már némi beépített feldolgozási képességük is van, ami az ennél összetettebb működést is lehetővé teszi. A legelső műholdakon a transzponderek és a csatornák egymáshoz rendelése statikus volt: a rendelkezésre álló sávszélességet egyszerűen rögzített frekvenciasávokra osztották. Manapság minden transzpondernyalábot időszeletekre osztanak, amelyeket felváltva használhat több felhasználó. Ezt a két módszert (frekvenciaosztásos és időosztásos multiplexelést) részletesen is tanulmányozni fogjuk a fejezet hátralevő részében.

Az első geostacionárius műholdaknak egyetlen széles nyalábjuk volt, amely a Föld felszínének körülbelül 1/3-át fedte le. Az így lefedett területet a műhold **lábnyomának** (**footprint**) hívják. A mikroelektronika árának, méretének és teljesítményfelvételének hatalmas csökkenésével ennél sokkal kifinomultabb adási stratégiák is lehetővé váltak. Minden műhold több antennával és transzponderrel rendelkezik. Minden lefelé irányuló nyalábot egy-egy kis földrajzi területre lehet irányítani, így több felfelé és lefelé irányuló adás is haladhat egymással párhuzamosan. Ezek az úgynevezett **pontnyalábok** (**spot beam**) általában ellipszis alakúak, és az átmérőjük mindössze néhány száz kilométer is lehet. Egy, az Egyesült Államokat kiszolgáló távközlési műhold általában egyetlen széles nyalábbal rendelkezik a 48 összefüggő állam lefedésére és egy-egy pontnyalábbal Alaszka és Hawaii részére.

A kommunikációs műholdak világának egyik új fejleménye a **VSAT-nak** (**Very Small Aperture Terminal – nagyon kis nyílásszögű terminál**) is nevezett, kis költségű mikroállomások kifejlesztése [Abramson, 2000]. Ezek az apró terminálok 1 méteres vagy még kisebb antennával rendelkeznek (a GEO-antennáknál szokásos 10 m-rel szemben), és körülbelül 1 watt teljesítménnyel képesek adni. A felfelé irányuló csatorna általában 1 Mb/s-os sebességre képes, a lefelé irányuló csatorna azonban gyakran néhány megabit/s sebességű. A közvetlen sugárzású műholdas tv-adások gyakran alkalmazzák ezt a megoldást az egyirányú átvitel megvalósítására.



2.17. ábra. *Hubot használó VSAT*

Sok VSAT-rendszerben a mikroállomások teljesítménye nem elegendően nagy ahhoz, hogy közvetlenül (illetve természetesen a műholdon keresztül) kommunikálhassanak egymással. A közvetlen kapcsolat helyett egy különleges földi állomást, a **hubot** használják a VSAT-terminálok közötti forgalom átjátszására, amely nagy méretű és nagy nyereségű antennával rendelkezik. Ezt az elrendezést a 2.17. ábra mutatja be. Ebben a működési módban vagy a vevőnek, vagy az adónak rendelkeznie kell egy nagy antennával és egy nagy teljesítményű erősítővel. Az olcsó végfelhasználói állomásokért hosszabb késleltetésekkel kell fizetnünk.

A VSAT-ok rendkívül hasznosak lehetnek a ritkán lakott területeken. Nem sokan vannak tisztában azzal a ténnyel, hogy a Föld népességének több mint a fele több mint egy órányi járóföldre lakik a legközelebbi telefontól. Több ezer kis faluba telefonkábeleket kihúzni messze túlmutat a legtöbb harmadik világbeli kormányzat költségvetésén, de az 1 méteres, napelemmel működtetett VSAT-antennák telepítése gyakran megvalósítható. A VSAT-ok olyan megoldást nyújtanak, amellyel össze lehet kötni a világ minden részét.

A kommunikációs műholdak sok tulajdonságukban radikálisan különböznek a felszíni kétpontos kapcsolatoktól. Először is, a GEO-műholdaknál a hosszú oda-vissza út annak ellenére nagy késleltetést jelent, hogy a jelek fénysebességgel (majdnem 300 000 km/s) terjednek. A küldőtől a végcélig az átviteli idő 250 és 300 ms között mozog, a felhasználó és a földi állomás távolságától, valamint a műhold horizont feletti magasságától függően. A késleltetés tipikus értéke 270 ms körül van (540 ms körül egy hubos VSAT-rendszerben).

Az összehasonlítás kedvéért: a földi mikrohullámú kapcsolatok terjedési késleltetése nagyjából 3 μs/km, a koaxiális kábelek és az üvegszálak késleltetése körülbelül 5 μs/km. Az utóbbi azért lassabb, mint az előző, mert az elektromágneses jelek gyorsabban terjednek a levegőben, mint a szilárd anyagokban.

A műholdak egy másik fontos tulajdonsága az, hogy természetüknél fogva adatszóró közegként viselkednek. A transzponder lábnyomán belül nem kerül többe néhány ezer állomásnak küldeni egy adást, mint egyetlen egynek. Ez a tulajdonság néhány alkalmazásban nagyon hasznos. Például elképzelhető az, hogy egy műhold népszerű weblapokat juttat el nagy területen szétszórt nagyszámú számítógépnek. Bár az adatszórás kétpontos vonalakkal is szimulálható, a műholdas adatszórás mégis sokkal olcsóbb lehet. A biztonság és a bizalmasság szemszögéből nézve viszont a műholdas rendszerek katasztrofálisak: mindenki minden adást hallhat. A titkosítás létfontosságú, amennyiben biztonságos átvitelt kell megvalósítanunk.

A műholdak további tulajdonsága, hogy egy üzenet átvitelének költsége nem függ az üzenet által megtett út hosszától. Egy tengeren túli hívás ugyanannyiba kerül, mint egy hívás az utca túloldalára. A műholdak előnye továbbá a kiváló bithibaarány, valamint a szinte azonnali telepíthetőség, ami a katasztrófavédelemben és a katonai hírközlésben is fontos szempont.

2.4.2. Közepes röppályás műholdak

A GEO-műholdaknál sokkal alacsonyabban, a két Van Allen-öv között találjuk a **MEO-** (**Medium-Earth Orbit – közepes röppályás**) műholdakat. A Földről nézve ezek a műholdak lassan sodródnak a földrajzi hosszúsági vonalak mentén, és mintegy 6 óránként kerülik meg a Földet. Ennek megfelelően ezeket a műholdakat követni kell, amíg végighaladnak az égen. Mivel a GEO-knál alacsonyabban vannak, kisebb a lábnyomuk a felszínen, és kisebb teljesítményű adókra van szükség a távolság áthidalására. Manapság ezeket a műholdakat inkább navigációs rendszerekben használják, mint távközlési célokra, így most nem tárgyaljuk őket részletesebben. A **GPS** (**Global Positioning System – globális helymeghatározó rendszer**) 30 darab, körülbelül 20 200 km magasan keringő műholdja például MEO-műhold.

2.4.3. Alacsony röppályás műholdak

Ahogy egyre lejjebb haladunk, elérünk a **LEO-** (**Low-Earth Orbit – alacsony röppályás**) műholdakhoz. Ezekből a műholdakból a gyors mozgás miatt sok szükséges egy teljes rendszerhez. Másrészt, mivel a műholdak közel vannak a Föld felszínéhez, a földi állomásoknak nem kell nagy teljesítményűnek lenniük, és az oda-vissza út késleltetése is csak néhány milliszekundum. A pályára állítás is lényegesen olcsóbb. Ebben a szakaszban beszédátvitelre szolgáló műholdrendszerek közül példaként kettőt, az Iridiumot és a Globalstart fogjuk megvizsgálni.

A műholdak alkalmazásának első 30 évében csak ritkán használtak alacsony röppályás műholdakat, mivel nagyon gyorsan jönnek-mennek az égen. 1990-ben a Motorola szűz területre lépett azzal, hogy 77 alacsony röppályás műhold fellövésének engedélyezésére adott be kérvényt az FCC-hez. A tervnek **Iridium** lett a neve, mivel az iridium a 77-es számú elem. A tervet később felülvizsgálták, és az új tervben már csak 66 műhold szerepelt, így át kellett volna keresztelni Diszpróziumra (ez a 66-os számú elem), de ez

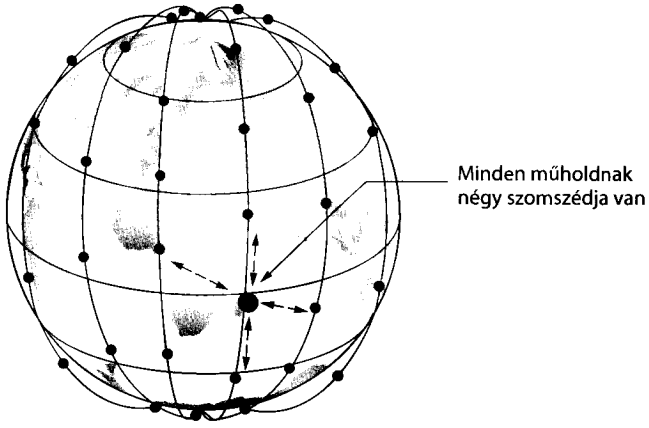
valószínűleg sokkal inkább úgy hangozhatott, mint valamilyen betegség neve. A rendszer lényege az volt, hogy amint egy műhold eltűnik a felhasználó látóteréből, egy másik lép a helyére. Ez a javaslat nagy befektetési lázat eredményezett a többi kommunikációs vállalat körében is. Hirtelen mindenki alacsony röppályás műholdláncot akart fellőni.

Miután hét év alatt sikerült összeszedni a partnereket és a pénzt, a kommunikációs szolgáltatás 1998 novemberében indult el. Sajnos a nagy és nehéz műholdas telefonok iránti piaci kereslet elhanyagolható volt, mivel a mobiltelefon-hálózatok látványosan nagyot fejlődtek 1990 óta. Mindezek következtében az Iridium nem volt nyereséges, és 1999 augusztusában csődbe jutott a történelem egyik leglátványosabb vállalati fiaskójában. A műholdakat és más vagyontárgyakat (amelyek értéke kb. 5 milliárd dollár volt) később egy befektető 25 millió dollárért vette meg egy árverésen. Más műholdas üzleti vállalkozások azonnal követték a példáját.

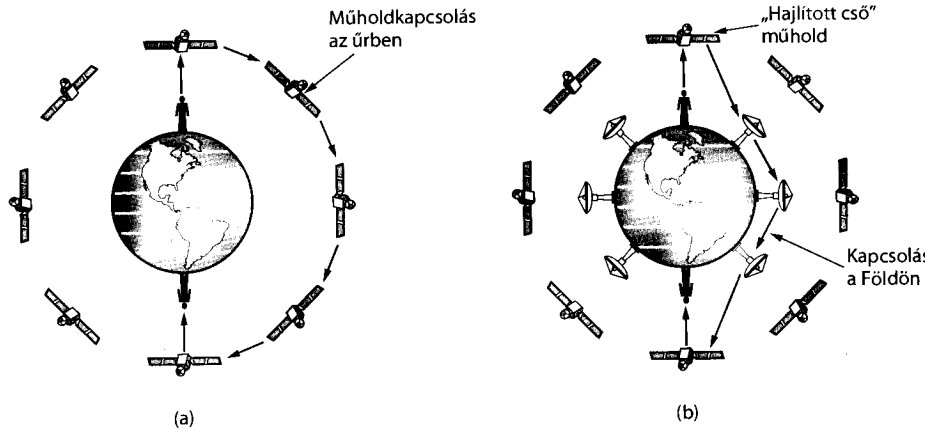
Az Iridium szolgáltatása 2001 márciusában újraindult, és azóta töretlenül fejlődik. Hang- és adatszolgáltatást, valamint személyhívó, fax- és navigációs szolgáltatást nyújt bárhol: földön, vízen vagy levegőben kézi készülékek segítségével, amelyek közvetlenül kommunikálnak az Iridium-műholdakkal. A vásárlók között megtaláljuk a hajózással, repüléssel és olajkitermeléssel foglalkozó cégeket, valamint az olyan embereket, akik olyan helyekre utaznak, ahol nincs telekommunikációs infrastruktúra (például sivatagok, hegyek, a Déli-sark és néhány harmadik világbeli ország).

Az Iridium-műholdakat 750 km-es magasságban helyezték el, kör alakú sarki röppályákon. Észak–déli láncokba rendeződnek, amelyekben 32 szélességi fokonként követik egymást a műholdak, ahogyan azt a 2.18. ábra is mutatja. Minden műholdnak legfeljebb 48 cellája (pontnyalábja) lehet, és 3840 csatornája van, amelyek közül néhányat a személyhívó és a navigációs szolgáltatás használ, a többi pedig az adat- és beszédátvitelt szolgálja.

Miként a 2.18. ábrán is látható, a hat műholdlánc az egész Földet lefedi. Az Iridium érdekes tulajdonsága, hogy a távoli felhasználók kommunikációja az űrben történik, a 2.19.(a) ábrán is látható módon. Megfigyelhetjük az ábrán, hogy a küldő az Északi-sarkon kapcsolódik egy, éppen a feje fölött álló műholdhoz. Minden műholdnak négy szomszédja van, amelyekkel kommunikálni tud, kettő ugyanabban a láncban (látható)



2.18. ábra. *Az Iridium-projekt műholdjai hat láncot alkotnak a Föld körül*



2.19. ábra. *(a) Átjátszás az űrben. (b) Felszíni átjátszás*

és kettő a szomszédos láncokban (nem látható). A műholdak átjátsszák a hívást ezen a rácson keresztül, míg végül az megérkezik a hívott félhez a Déli-sarkon.

Az Iridium-rendszer alternatív megoldása a **Globalstar**. Ez a rendszer 48 LEO-műholdra épít, de az Iridiumétól eltérő kapcsolási módszert alkalmaz. Az Iridium a hívásokat egyik műholdról a másikra továbbítja, amely kifinomult kapcsolóberendezéseket igényel a műholdak fedélzetén. A Globalstar ezzel szemben a hagyományos hajlítottcső-kialakítást használja. A 2.19.(b) ábrán az Északi-sarkról indított hívást visszaküldik a földre, ahol egy nagy földi állomás veszi azt. A hívást ezután földi hálózaton keresztül irányítják a hívott félhez legközelebbi földi állomáshoz, és onnan egy műholdra, amelyik egy hajlítottcső-kapcsolaton keresztül továbbítja a hívott félhez, amint azt az ábra is mutatja. Ez a megoldás azért előnyös, mert a bonyolultabb dolgok nagy részét a földön tartja, ahol könnyebb azzal boldogulni. A nagy földi antennák használata azt is lehetővé teszi, hogy nagy teljesítményű jeleket küldhessünk a műholdnak, és gyenge jeleket is venni tudjunk, így a telefonok teljesítményfelvételét is csökkenteni lehet. Végül is a telefon csak néhány milliwattnyi teljesítményt ad le, így a földi állomásra visszatérő jel még azzal együtt is elég gyenge, hogy a műhold is felerősíti útközben.

Évente továbbra is körülbelül 20 műholdat állítanak pályára, beleértve az egyre nagyobb, 5000 kg-ot is meghaladó tömegű műholdakat is. Persze vannak nagyon kicsi műholdak is a költségtudatosabb szervezetek számára. Annak érdekében, hogy az űrkutatást elérhetőbbé tegyék, a Kaliforniai Műszaki Egyetem és a Stanford Egyetem tudósai 1999-ben összeültek, hogy szabványosítsák a kisméretű műholdakat és egy hordozórakétát, ami jelentősen csökkentené a fellövés költségeit [Nugent és mások, 2008]. A **CubeSat**-ok (**kockaműholdak**) olyan kis műholdak, melynek 10 cm × 10 cm × 10 cm-es méretű, 1 kg-nál kisebb tömegű kockákból álló egységek, és amelyek egyenként kevesebb mint 40 000 dollárért állíthatók pályára. A hordozórakéta másodlagos hasznos teherként repíti fel kereskedelmi küldetése alkalmával. A másodlagos hasznos teher lényegében egy négyzet keresztmetszetű cső, amelyben maximum 3 kockaműhold van, és rugók segítségével bocsátja azokat Föld körüli pályára. Körülbelül 20 kockaműholdat állítottak eddig pályára, és sok továbbin dolgoznak. A legtöbb ilyen műhold a földi állomásokkal az UHF- és VHF-sávokon kommunikál.

2.4.4. A műholdak és az üvegszál összehasonlítása

A műholdas és a földi kommunikáció összehasonlítása nagyon tanulságos. 20 évvel ezelőtt még joggal állíthattuk, hogy a kommunikáció jövője a kommunikációs műholdakban testesül meg. A telefonhálózat tulajdonképpen keveset változott az addig eltelt 100 évben, és annak sem mutatta semmi jelét, hogy az elkövetkező 100 évben megváltozna. Ennek a kőkorszaki elképzelésnek nagyrészt a szabályozási környezet volt az alapja. A telefontársaságoktól elvárták, hogy jó beszédátviteli szolgáltatást nyújtsanak ésszerű árakon (amelyet meg is tettek), és cserébe garanciát kaptak befektetéseik megtérülésére. 1200 b/s-os modemek álltak azok rendelkezésére, akik adatokat akartak továbbítani. Nagyjából ez volt minden, ami akkor elérhető volt.

A piaci verseny 1984-es amerikai és valamivel későbbi európai bevezetése után mindez teljesen megváltozott. A telefontársaságok elkezdték fényvezető kábelekre lecserélni a távolsági átviteli hálózataikat, és olyan, nagy sávszélességű szolgáltatásokat kezdtek nyújtani, mint például az ADSL (Asymmetric Digital Subscriber Line – aszimmetrikus digitális előfizetői vonal). Azzal a hosszú ideje fennálló gyakorlattal is szakítottak, hogy a helyi szolgáltatás támogatása érdekében mesterségesen magas áron kínálják a távolsági szolgáltatásokat. Hirtelen úgy tűnt, hogy hosszú távon mégis a földi üvegszálas összeköttetések lesznek a nyertesek.

Mindezek ellenére a kommunikációs műholdaknak számos olyan nagy alkalmazási területe van a piacon, amelyen az üvegszálak nem versenyeznek vele (egyes esetekben nem is tudnak). Először is, amikor a gyors telepítés létfontosságú, akkor a műholdak könnyen nyernek. A gyors reakció hasznos háború idején katonai kommunikációs rendszereknél és katasztrófák esetén békeidőben. Például a 2004. decemberi hatalmas szumátrai földrengés és az azt követő szökőár után a távközlési műholdak 24 órán belül képesek voltak helyreállítani a kommunikációt az első jeladók számára. A gyors reagálás azért volt lehetséges, mert adott egy fejlett műholdas szolgáltatói piac, ahol a jelentős szereplők, mint például az Intelsat, a maga több mint 50 műholdjával, szinte bárhol képes kapacitást bérbe adni, ahol szükséges. A meglévő műholdas hálózatok ügyfelei számára egy VSAT könnyen és gyorsan felállítható, hogy megabit/sec sebességű adatkapcsolatot biztosítson a világ egy másik pontjával.

A második kommunikációs alkalmazási terület olyan helyeken van, ahol a földi infrastruktúra kevéssé fejlett. Manapság sokan akarnak mindenhol kommunikálni, ahová csak mennek. A mobiltelefon-hálózatok jól lefedik a sűrűn lakott területeket, de a lefedettség nem megfelelő egyéb helyeken (például a tengeren vagy a sivatagban). Az Iridium viszont beszédátviteli szolgáltatást nyújt bárhol a Földön, még a Déli-sarkon is. A földi infrastruktúra telepítése a terepviszonyoktól és a szükséges szolgalmi jogoktól függően költséges is lehet. Indonéziának például a belföldi telefonforgalomra saját műholdja van. Egy műhold pályára állítása olcsóbb volt, mint több ezer tenger alatti kábel lefektetése a szigetvilág 13 677 szigete között.

A harmadik alkalmazási terület az, ahol lényeges az adatszórási képesség. Egy műholdon keresztül elküldött üzenetet több ezer földi állomás is vehet egyszerre. A műholdakat ezért arra is használják, hogy tv-műsorokat továbbítsanak a helyi állomásoknak.

Jelenleg hatalmas piaca van a digitális televízió- és rádióadások műholdas sugárzásának közvetlenül azokhoz a végfelhasználókhoz, akik lakásukban és a kocsijukban műholdvevővel rendelkeznek. Természetesen mindenféle egyéb tartalom is sugározható. Például egy olyan szervezet számára, amelyik több ezer viszonteladónak egyfolytában küldi részvények, értékpapírok vagy árucikkek árfolyamát, egy műholdas rendszer sokkal olcsóbb lehet, mint a földfelszínen megvalósítani az adatszórást.

Röviden összefoglalva, úgy tűnik, hogy a jövő legfőbb kommunikációs eszköze a földfelszíni üvegszálak és a cellaalapú rádiózás kombinációja lesz, de néhány különleges felhasználásra mégis a műholdak alkalmasabbak. Létezik azonban egy mindezekre kiterjedő feltétel: a gazdaságosság. Bár az üvegszálak nagyobb sávszélességet kínálnak, teljességgel elképzelhető, hogy a földfelszíni és a műholdas kommunikáció agresszív árversenyt fog folytatni egymással. Ha a műszaki fejlesztések radikálisan lecsökkentik a műholdak telepítésének költségét (ha például egy jövőbeli űrsikló egyetlen fellövéssel több tucat műholdat tud pályára állítani), vagy az alacsony röppályás műholdak fejlődése nagyon beindul, akkor nem biztos, hogy minden piacon az üvegszál fog győzni.

2.5. Digitális moduláció és multiplexelés

Most, hogy már áttanulmányoztuk a vezetékes és a vezeték nélküli csatornákat, figyelmünket a digitális információ továbbítására fordítjuk. A vezetékes és vezeték nélküli csatornák analóg jeleket hordoznak, mint amilyen például az állandóan változó villamos feszültség, a fényintenzitás vagy a hangerősség. A digitális információ továbbításához ki kell találnunk, hogy az analóg jelek miként ábrázoljanak biteket. Azt az átalakítási folyamatot, amely a bitek és az azokat ábrázoló jelek közötti átalakítást végzi, **digitális modulációnak** nevezzük.

Olyan módszerekkel kezdünk, amelyek közvetlenül alakítják át a biteket jelekké. Ezek a módszerek **alapsávú átvitelt** eredményeznek, amelyben a jel komponensei a nulla és a maximum közötti tartományba eső frekvenciát foglalják el, a jelsebességtől függően. Ez a vezetékeknél általános. Ezután olyan módszerek következnek, melyek változtatják a vivőjel amplitúdóját, fázisát vagy frekvenciáját a bittovábbítás érdekében. Ezek a módszerek **áteresztő sávú átvitelt** eredményeznek, amiben a jel komponensei a vivőjel frekvenciája körüli frekvenciasávot foglalják el. Ez leginkább a vezeték nélküli és az optikai csatornákra jellemző, ahol a jeleknek egy megadott frekvenciasávban kell tartózkodniuk.

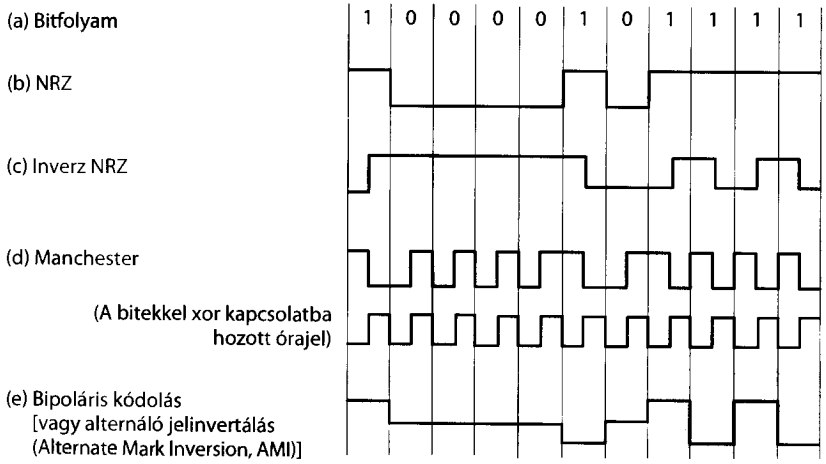
A csatornákon gyakran több jel osztozik. Végtére is, sokkal célszerűbb egy vezetéket használni több jel továbbítására, mint minden jelnek külön vezetéket telepíteni. Ezt a fajta megosztást **multiplexelésnek** hívjuk. Ez sokféleképpen elérhető. Láthatunk majd példákat idő-, frekvencia- és kódosztásos multiplexelésre is.

A fejezetben leírt modulációs és multiplexelési technikákat széles körben alkalmazzák vezetékes, üvegszálas, földi vezeték nélküli és műholdas csatornák esetében. A következő szakaszokban példahálózatokon mutatjuk be ezeket működés közben.

2.5.1. Alapsávú átvitel

A digitális moduláció legközvetlenebb formája, ha a pozitív feszültséget használjuk az 1, és a negatív feszültséget a 0 ábrázolására. Üvegszál esetében a fény jelenléte képviseli az 1-et és a hiánya a 0-t. Ezt a sémát **NRZ**-nek (**Non-Return-to-Zero – nullára vissza nem térő**) nevezzük. A furcsa névnek történelmi okai vannak, és egyszerűen csak azt jelenti, hogy a jel követi az adatot. Erre látható példa a 2.20.(b) ábrán.

Ha egyszer elküldésre került, az NRZ-jel továbbterjed a vezetéken. A másik végén a vevő bitekké alakítja oly módon, hogy szabályos időközönként mintát vesz a jelből. Ez a jel nem pontosan úgy fog kinézni, mint a küldött jel. A vevő oldalán a csatorna és a zaj által csillapított és torzított lesz. A bitek dekódolásához a vevő a jelből vett mintákat szimbólumokká képezi le. Az NRZ-nél egy pozitív feszültségszint továbbítódik, jelezve azt, hogy 1-est küldtek, és negatív feszültségszint továbbítódik azt jelezve, hogy 0-t küldtek.



2.20. ábra. *Vonali kódok: (a) bitek, (b) NRZ, (c) NRZI, (d) Manchester, (e) bipoláris vagy AMI*

Az NRZ jó kiindulópont a vizsgálatainkhoz, mert egyszerű, de a gyakorlatban ritkán használják önmagában. Az összetettebb sémák a mérnöki elgondolásnak jobban megfelelő jelekké tudják alakítani a biteket. Ezeket a sémákat vonali kódoknak nevezzük. Később olyan vonali kódokat mutatunk be, amelyek segítenek a sávszélesség hatékony kihasználásánál, az órajel visszaállításánál és az egyenfeszültség egyensúlyának megteremtésénél.

Sávszélesség hatékony kihasználása

NRZ esetén a jel ciklikusan változik a negatív és a pozitív szint között 2 bitenként (1-esek és 0-k váltakozása esetén). Ez azt jelenti, hogy legalább $B/2$ Hz sávszélességre van szükségünk, ha a bitsebesség B bit/sec. Ez a reláció a Nyquist-képletből jön [(2.2) egyenlet]. Ez egy alapvető határérték, tehát az NRZ nem futhat gyorsabban nagyobb sávszélesség használata nélkül. A sávszélesség gyakorta korlátozott erőforrás, még vezetékes csatornák esetén is. Nagyobb frekvenciás jelek nagyobb mértékű csillapítást szenvednek, emiatt kevésbé hasznosak, valamint gyorsabb elektronikát is igényelnek.

A korlátozott sávszélesség használatára egy módszer az, ha több mint két jelszintet alkalmazunk. Például négy feszültségszint használatával 2 bitet is elküldhetünk egyszerre, egyetlen **szimbólumként.** Ez a megoldás mindaddig működik, amíg a jelteljesítmény a vevőnél eléggé nagy ahhoz, hogy a négy szint megkülönböztethető legyen. A jelszintváltozás sebessége ekkor az adatsebesség fele, tehát a szükséges sávszélesség lecsökkent.

A jel változásának sebességét **jelsebességnek (baud rate)** vagy **szimbólumsebességnek (symbol rate)** hívjuk, hogy megkülönböztessük az adatsebességtől. Az adatsebesség egyenlő a jelsebesség szorozva a szimbólumonkénti bitek számával. Egy korábban használatos név a szimbólumsebességre, főleg a telefonmodemnek nevezett, digitális adatokat telefonvonalakon keresztül továbbító eszközökkel összefüggésben, a jelsebesség. A szakirodalomban az „adatsebesség" (bit rate) és a „jelsebesség" (baud rate) gyakran szerepel tévesen.

Fontos megjegyezni, hogy a jelszintek száma nem szükségszerűen kettő valamelyik hatványa. Gyakran nem is az, néhány szintet hibavédelemre, illetve a vevő kialakításának egyszerűsítésére használnak.

Órajel visszaállítása

Minden olyan séma esetén, ami biteket kódol szimbólumokká, a vevőnek tudnia kell, hogy mikor ér véget egy szimbólum és mikor kezdődik a következő, annak érdekében, hogy helyesen dekódolja a biteket. Az NRZ-nél, ahol a szimbólumok egyszerűen feszültségszintek, a 0-k és 1-esek hosszú sora változatlanul hagyja a jelszintet. Egy idő után elég nehéz megkülönböztetni a biteket, mivel 15 nulla majdnem úgy néz ki, mint 16, hacsak nincs egy nagyon pontos óránk.

Egy pontos óra segíthetne ezen a gondon, de meglehetősen drága megoldás lenne kommersz eszközöknél. Ne felejtsük el, hogy olyan adatkapcsolatokon mérjük a bitidőket, amelyek sok megabit/sec sebességgel működnek, tehát az óra eltérése a leghosszabb engedélyezett működés során is kevesebb lehetne a mikroszekundum tört részénél. Ez elfogadható lehetne lassú adatkapcsolatok vagy rövid üzenetek esetén, de ez nem egy általános megoldás.

Egy módszer lehet az, amikor egy külön órajelet küldenek a vevőnek. Egy másik órajelvezeték nem nagy probléma számítógépsínek vagy olyan rövid kábelek esetén, amelyekben sok vezeték fut párhuzamosan, de a legtöbb hálózati adatkapcsolat esetén ez pazarlás, hiszen ha lenne még egy vezetékünk jel küldésére, akkor azt akár már adat továbbítására is használhatnánk. Ügyes trükk lehet az órajel és az adatjel KIZÁRÓ VAGY (XOR) kapcsolatba hozása, és így már nem szükséges egy külön vezeték. Az eredmény a 2.20(d) ábrán látható. Az óra bitidőnként egy órajel-átmenetet állít elő, vagyis a bitsebesség kétszeresével működik. Amikor az órajel KIZÁRÓ VAGY kapcsolatba kerül a 0 szinttel, akkor egy L-H jelátmenetet hoz létre, ami egyszerűen az órajel. Ez a jelátmenet egy logikai 0. Amikor az órajel KIZÁRÓ VAGY kapcsolatba kerül az 1-es szinttel, akkor megfordul és egy H-L jelátmenet jön létre. Ez a jelátmenet egy logikai 1-es. Ezt a sémát **Manchester-kódolásnak** hívjuk, és a klasszikus Ethernethez használták.

A Machester-kódolás hátránya, hogy az órajel miatt kétszer akkora sávszélességet igényel, mint az NRZ, és már megtanultuk, hogy a sávszélesség gyakran számít. Egy másik módszer azon az elképzelésen alapul, hogy az adatkódolásnál kell biztosítani, hogy elég átmenet legyen a jelben, tekintettel arra, hogy az NRZ-nek csak 0-k és 1-esek hosszú sorozata esetén vannak órajel-visszaállítási problémái. Ha gyakoriak az átmenetek, akkor a vevőnek egyszerű szinkronban maradni a bejövő szimbólumfolyammal.

Első lépésként egyszerűsíthetjük a helyzetet azzal, hogy az 1-est *átmenetként*, a 0-t pedig *nem átmenetként* kódoljuk, vagy fordítva. Ezt a kódolást **NRZI**-nek (**Non-Return-to-Zero Inverted – invertált nullára vissza nem térő**) nevezzük. Erre példát a 2.20.(c) ábrán láthatunk. A számítógépes perifériák csatlakoztatásához használatos népszerű **USB- (Universal Serial Bus – univerzális soros sín)** szabvány NRZI-kódolást használ, amelynek használatával az 1-esek hosszú sorozata nem jelent gondot.

Természetesen a 0-k hosszú sorozata még mindig megoldandó problémát okoz. Ha mi lennénk a telefontársaság, akkor egyszerűen csak megkövetelnénk, hogy a küldő ne továbbítson túl sok 0-t. A régebbi, USA-ban használt digitális telefonvonalak, név szerint a **T1 vonalak** esetében valóban előírták a megfelelő működéshez, hogy nem küldhető egymás után 15-nél több 0. A probléma tényleges megoldása érdekében megtörhetjük a 0-k sorozatát a továbbítandó bitek kisebb csoportjaira történő leképezéssel úgy, hogy az egymás utáni 0-k csoportjai kicsivel hosszabb mintákba kerülnek leképezésre, amelyekben nincs túl sok egymást követő 0.

Ennek elérésére egy jól ismert kódolás a **4B/5B**. Minden 4 bitnek egy 5 bites mintát feleltetünk meg egy rögzített leképezési táblával. Az ötbites mintára azért esett a választás, mert így soha nem lesz több mint három egymást követő 0. A leképezés a 2.21. ábrán látható. Ez a séma 25% többletet jelent, ami jobb, mint a Manchester-kódolás 100%-os többlete. Mivel 16 bemeneti és 32 kimeneti kombináció van, ezért néhány kimeneti kombináció nincs használatban. Félretéve a túl sok egymás utáni 0-t tartalmazó kombinációkat, még mindig marad néhány kód. Ráadásul ezeket a nem adat jellegű kódokat arra is használhatjuk, hogy a fizikai réteget vezérlő jeleket ábrázoljanak. Például, néhány esetben az „11111" egy tétlen vonalat jelent, az „11000" pedig egy keret kezdetét jelzi.

Adat (4B)	Kódszó (5B)	Adat (4B)	Kódszó (5B)
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

2.21. ábra. 4B/5B-leképezés

Alternatív megközelítés, hogy az adatok véletlenszerűnek tűnjenek, rejtjelezés néven ismert. Ebben az esetben nagyon valószínű, hogy gyakori átmenetek várhatók. A **rejtjelező (scrambler)** úgy működik, hogy továbbítás előtt KIZÁRÓ VAGY kapcsolatba hozza az adatot egy ál-véletlensorozattal. Ez a keverés annyira véletlenszerűvé teszi az adatot, amennyire az ál-véletlensorozat az (feltételezve, hogy a keverés független az ál-véletlensorozattól). A vevő aztán KIZÁRÓ VAGY kapcsolatba hozza a beérkező biteket ugyanazzal az ál-véletlensorozattal, hogy visszaállítsa a valódi adatot. A gyakorlati használhatóság érdekében jó, ha az ál-véletlensorozat egyszerűen létrehozható. Gyakran megadják egy egyszerű véletlenszám-generátor kezdeti értékét.

A rejtjelezés előnyös, mert nem okoz sávszélesség- vagy időtöbbletet. Ami azt illeti, gyakran segít kondicionálni a jelet, hogy ne azokban a domináns frekvenciakomponensekben legyen az energiája, amelyeket az ismétlődő adatminták okoznak, ami esetleg elektromos interferenciát okoz. A rejtjelezés azért is segít, mert a véletlenszerű jelek általában „fehérek", vagyis az energiájuk a frekvenciakomponenseken egyenletesen szétterül.

A rejtjelezés azonban nem garantálja, hogy nem lesznek hosszú sorozatok. Időnként lehetséges, hogy nem lesz szerencsénk. Amennyiben az adat ugyanaz, mint az ál-véletlensorozat, a KIZÁRÓ VAGY kapcsolat eredménye 0. Ez a végeredmény nehezen jelezhető előre, hosszú ál-véletlensorozatnál nem fordul elő gyakran. Rövid vagy megjósolható ál-véletlensorozattal azonban rosszindulatú felhasználók számára lehetségessé válhat olyan bitminták küldése, amelyek rejtjelezését követően a hosszú 0 sorozat áll elő, és ez a kapcsolat meghibásodását okozza. Ez volt a hibája a telefonhálózatokban IP-csomagoknak SONET-adatkapcsolatokon keresztül történő küldésével kapcsolatos korábbi verziós szabványoknak [Malis és Simpson, 1999]. A felhasználók „gyilkos csomagokat" tudtak küldeni, amelyek garantáltan problémákat okoztak.

Kiegyensúlyozott jelek

Azokat a jeleket, amelyeknek még rövid időtartam alatt is ugyanakkora pozitív feszültsége van, mint negatív, **kiegyensúlyozott jeleknek (balanced signals)** hívjuk. Az átlaguk nulla, ami azt jelenti, hogy nincs egyenáramú (DC) összetevőjük. Az egyenáramú összetevő hiánya előny, mert néhány csatorna, mint például a koax kábel vagy a transzformátorral ellátott vezetékek, fizikai tulajdonságaiknak köszönhetően erősen csillapítják az egyenáramú összetevőt. Továbbá, az egyik, **kapacitív csatolásnak** nevezett mód, amivel a vevőt összekapcsoljuk a csatornával, csak a jel váltóáramú (AC) részét továbbítja. Bármely olyan esetben, ha olyan jelet küldünk, aminek az átlaga nem nulla, csak pazaroljuk az energiát, hiszen az egyenáramú összetevő kiszűrésre kerül.

A kiegyensúlyozás segít abban, hogy átmenetet biztosítsunk az órajel visszaállítására, hiszen mind a pozitív, mind a negatív feszültség jelen van. Egyszerű lehetőséget ad a vevők kalibrálására is, mivel a jelek átlaga mérhető és döntési küszöbértékként használható a szimbólumok dekódolásához. Kiegyensúlyozatlan jelek esetén a sűrűn ismétlődő 1-esek miatt például az átlag eltolódhat a valódi döntési szinttől, ami több szimbólum hibás kódolását okozná.

A kiegyensúlyozott kód előállításának lényegre törő módja az, ha két feszültségszintet használunk a logikai 1-es ábrázolására (mondjuk +1 V-ot és –1 V-ot), valamint 0 V-ot

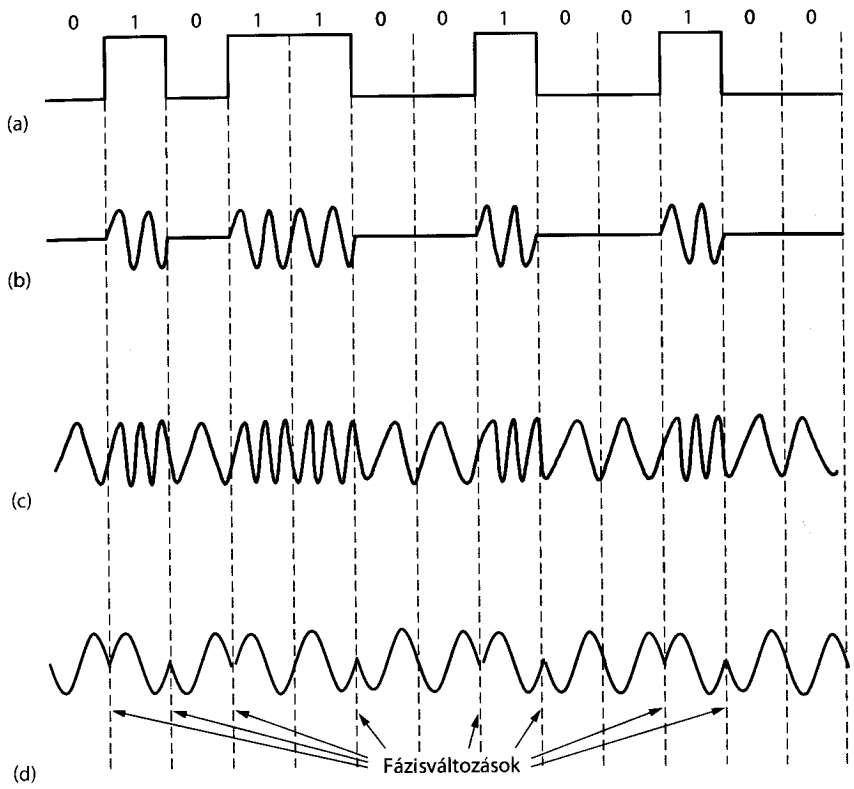
a logikai nulla ábrázolására. Az 1-es bit küldésekor az adó váltakozva a +1 V és a –1 V feszültségszinteket használja, így tehát az egymás után küldött jelek átlaga nulla lesz. Ezt a sémát **bipoláris kódolásnak** hívjuk. Telefonhálózatokban **AMI**-nak (**Alternate Mark Inversion – alternáló jelinvertálás**) hívják, arra a régi terminológiára építve, amiben az 1-et „mark"-nak (jelnek), a 0-t pedig „space"-nek (jelhiánynak) nevezik. Ennek egy példája látható a 2.20.(e) ábrán.

A bipoláris kódolás feszültségszintet használ az egyensúly eléréséhez. Alternatívaként használhatjuk például a 4B/5B leképezést az egyensúly eléréséhez (akárcsak az átmeneteket az órajel helyreállításához). Erre a fajta kiegyensúlyozott kódolásra példa a **8B/10B** vonali kód, ami 8 bit bemenetet képez le 10 bit kimenetre, tehát 80%-os hatékonyságú csakúgy, mint a 4B/5B vonali kód. A 8 bitet egy 5 bites és egy 3 bites csoportra bontjuk fel, amelyek közül az előbbit 6 bitre, az utóbbit pedig 4 bitre képezünk le. A 6 bites és a 4 bites szimbólumokat ezután összekapcsoljuk. Minden csoportban néhány bemeneti minta leképezhető kiegyensúlyozott kimenti mintává, amikben ugyanannyi 0 és 1-es van. Például a „001"-et leképezi „1001"-re, ami kiegyensúlyozott. De nincs elegendő kombináció minden kimeneti minta kiegyensúlyozásához. Ezekre az esetekre, minden bemenő mintát két kimenő mintára képezünk le. Egyik kap egy extra 1-est, míg a másodlagos minta egy extra 0-t. Például, a „000"-t leképezzük „1011"-re és az azt kiegészítő „0100"-ra is. Mivel a bejövő biteket kimenő bitekké képezzük le, a kódoló megjegyzi a korábbi szimbólum **diszparitását**. A diszparitás a 0-k és az 1-esek száma összesen, aminél a jel kiegyensúlyozatlan. A kódoló ez után kiválaszt egy kimeneti mintát vagy annak másodlagos mintáját, hogy csökkentse a diszparitást. A 8B/10B kódolás esetén a diszparitás legfeljebb 2 bit lesz. Így a jel soha nem kerül messze a kiegyensúlyozottól. Továbbá, soha nem lesz 5-nél több egymást követő 1-es vagy 0, hogy az órajel visszaállítását is segítse.

2.5.2. Áteresztő sávú átvitel

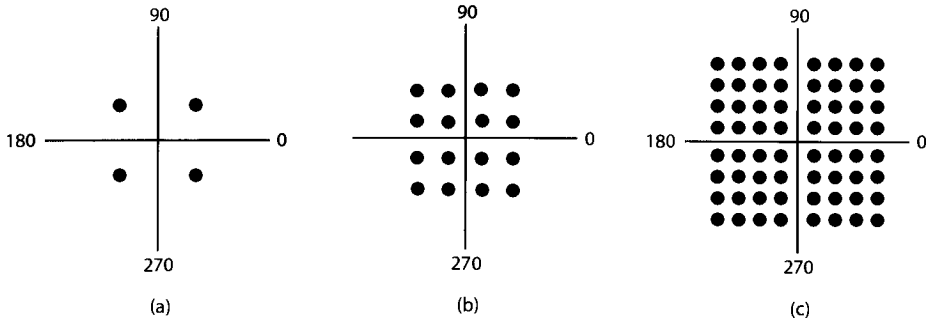
Gyakran olyan frekvenciatartományt szeretnénk használni információ csatornán keresztüli küldéséhez, ami nem nullánál kezdődik. A vezeték nélküli csatornák esetén nem praktikus nagyon kis frekvenciájú jelek küldése, mert az antenna átmérőjének mérete egyenlő a jel hullámhosszának hányadosával, ami kisfrekvencián nagyon nagy lesz. Mindenesetre, a jogszabályi feltételek és az interferencia elkerülésének szükségessége határozza meg általában, hogy milyen frekvenciát választunk. Még vezetékek esetében is hasznos egy adott frekvenciasávba helyezni a jelet, mert így a különböző jelek egyszerre lehetnek jelen a csatornán. Ezt a fajta átvitelt áteresztő sávú átvitelnek nevezzük, mert a jel továbbítására egy tetszőleges frekvenciasávot használunk.

Szerencsére a fejezet korábbi részeiből származó alapvető eredményeink sávszélességben vagy a frekvenciasáv szélességében vannak megadva. Az abszolút frekvenciaértékek nem számítanak az átviteli kapacitás vonatkozásában. Ez azt jelenti, hogy vehetünk egy **alapsávi jelet**, ami 0-tól B Hz-ig tart, és eltolhatjuk úgy, hogy egy S-től $S+B$ Hz-ig tartó **áteresztősávot** foglaljon el anélkül, hogy megváltozna a hordozott információ mennyisége még akkor is, ha a jel másképp fog kinézni. Hogy feldolgozzuk a jelet a vevőnél, visszatolhatjuk azt az alapsávig, ahol a szimbólumok egyszerűbben detektálhatók.



2.22. ábra. *(a) bináris jel, (b) amplitúdóbillentyűzés, (c) frekvenciabillentyűzés, (d) fázisbillentyűzés*

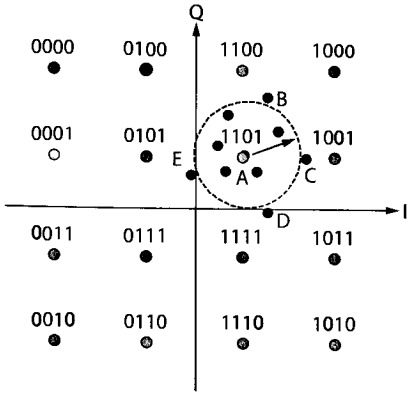
A digitális moduláció áteresztő sávú átvitellel valósul meg az áteresztősávban elhelyezkedő vivőjel változtatásával vagy modulálásával. Modulálhatjuk a vivőjel amplitúdóját, frekvenciáját vagy a fázisát. Ezeknek a módszereknek van megfelelő neve. Az **ASK** (**Amplitude Shift Keying – amplitúdóbillentyűzés**) esetében két különböző amplitúdót használunk a 0 és az 1 ábrázolására. A 2.22.(b) ábrán egy olyan példa látható, ahol az egyik szint nullától különböző, a másik pedig nulla. Kettőnél több szint több szimbólum ábrázolására használható. Ehhez hasonlóan, az **FSK** (**Frequency Shift Keying – frekvenciabillentyűzés**) esetében kettő vagy több különböző frekvencia használatos. A 2.21.(c) ábrán látható példa csak két frekvenciát használ. A **PSK** (**Phase Shift Keying – fázisbillentyűzés**) legegyszerűbb formájában a vivőhullám fázisszögét szisztematikusan 0 vagy 180 fokkal eltolják minden egyes szimbólumperiódusban. Mivel két fázis van, ezt a modulációt **BPSK**-nak (**Binary Phase Shift Keying – bináris fázisbillentyűzés**) hívják. A bináris szó itt a két szimbólumra utal, nem arra, hogy a szimbólumok két bitet ábrázolnak. Erre példa a 2.22.(c) ábrán látható. Egy, a csatorna sávszélességét jobban kihasználó séma, ha négy fázisszöget használunk, például 45, 135, 225 és 315 fokot, amikor szimbólumonként 2 bitnyi információt vihetünk át. Ezt a verziót **QPSK**-nak (**Quadrature Phase Shift Keying – kvadratúra fázisbillentyűzés**) hívjuk.



2.23. ábra. *(a) QPSK, (b) QAM-16, (c) QAM-64*

Ezek a sémák kombinálhatók és használhatunk több szintet, hogy szimbólumonként több bitet továbbíthassunk. Mivel a frekvencia és a fázis összefügg, ezért egyszerre csak az egyik modulálható, lévén a frekvencia a fázis időbeli változásának sebessége. Általában az amplitúdóbillentyűzést és a fázisbillentyűzést kombinálják. A 2.23. ábrán három példa látható. Mindegyik példánál a pontok az egyes szimbólumokhoz tartozó megengedett amplitúdó- és fáziskombinációt mutatják. A 2.23.(a) ábrán egyenlő távolságra elhelyezkedő pontokat látunk, 45, 135, 225, és 315 fokoknál. A fázisszöget a pontot az origóval összekötő egyenes és a pozitív x tengely által bezárt szög mutatja. Az amplitúdó az origótól való távolság. Ez az ábra a QPSK-t ábrázolja.

Ezt a fajta diagramot **konstellációs diagramnak** vagy **csillagképdiagramnak** (**constellation diagram**) hívják. A 2.23.(b) ábrán egy sűrűbb konstellációjú modulációs sémát láthatunk. Az amplitúdók és a fázisok tizenhat kombinációja kerül felhasználásra, tehát a modulációs séma szimbólumonként 4 bit továbbítására használható. Ezt **QAM-16**-nak hívják, ahol a QAM a **kvadratúra amplitúdómoduláció** (**Quadrature Amplitude Modulation**) rövidítése. A 2.23.(c) ábra egy még sűrűbb modulációs séma 64 különböző kombinációval, így szimbólumonként 6 bit továbbítható. Ennek a neve **QAM-64**. Ennél magasabb rendű QAM-ek is használatosak. Ahogy az már sejthető



1001 küldése esetén:

Pont	Dekódolt érték	Bithibák
A	1101	0
B	1100	1
C	1001	1
D	1111	1
E	0101	1

2.24. ábra. *Gray-kódolású QAM-16*

ezekből a konstellációkból, egyszerűbb olyan elektromos eszközöket készíteni, amelyek az egyes tengelyek értékeinek, és nem az amplitúdó és a fázisértékek kombinációjából állítják elő a szimbólumokat. A minták ezért néznek ki úgy, mint a négyzetek és nem úgy, mint a koncentrikus körök.

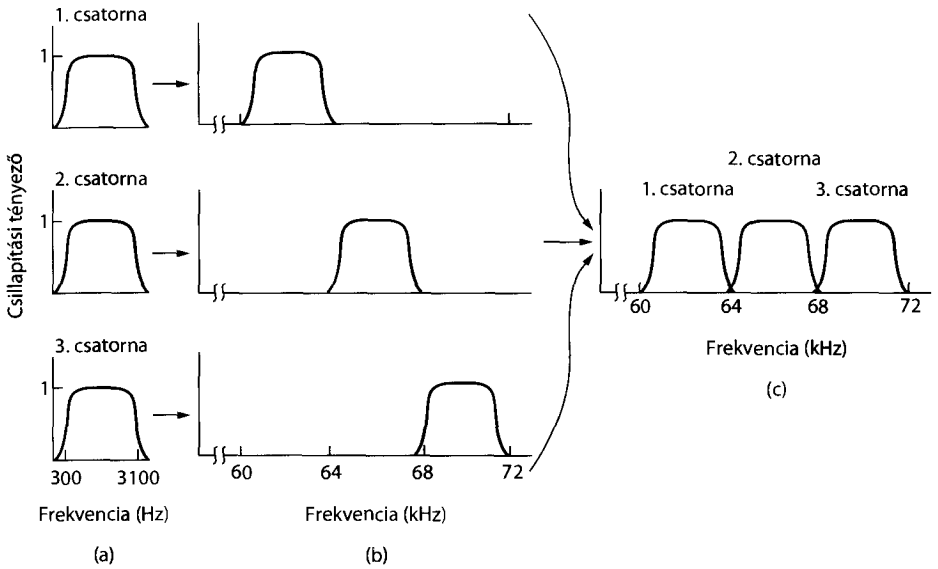
Az eddig látott konstellációk nem mutatják meg, hogy a bitek miként vannak hozzárendelve a szimbólumokhoz. A hozzárendelés során fontos szempont, hogy egy kis zajlöket a vevő oldalán ne vezessen sok bithibához. Ez megtörténhet, ha egymást követő bitértékeket rendelünk a szomszédos szimbólumokhoz. Ha a QAM-16 esetén például egy szimbólum jelképezi a 0111-et és a szomszédos szimbólum pedig az 1000-át, ha a vevő hibásan a szomszédos szimbólumot veszi le, akkor ez azt okozza, hogy minden bit rossz lesz. Jobb megoldás az, amikor a bitek szimbólumokká történő leképezése úgy megy végbe, hogy a szomszédos szimbólumok csak 1 bitpozícióban térnek el. Ezt a fajta leképezést **Gray-kódnak** nevezzük. A 2.24. ábrán egy olyan QAM-16-os csillagkép látható, amelyen Gray-kódolást alkalmaztak. Most, ha a vevő a hibás szimbólumot dekódolja, akkor ez csak egyetlen bithibát okoz abban az elvárt esetben, ha a kódolt szimbólum közel áll a továbbított szimbólumhoz.

2.5.3. Frekvenciaosztásos multiplexelés

Az eddig látott modulációs sémák egy jel küldését tették lehetővé a bitek vezetékes vagy vezeték nélküli összeköttetésen történő továbbításához. A méretezés gazdaságossága azonban fontos szerepet játszik a hálózatok használatában. Két iroda között lényegében ugyanannyiba kerül a nagy sávszélességű átviteli vonal telepítése és fenntartása, mint a kis sávszélességűé (azaz a költség az árok kiásásából ered, nem abból, hogy milyen típusú kábelt vagy üvegszálat fektetnek bele). Következésképpen multiplexelési sémák kerültek kialakításra, hogy meg lehessen osztani a vonalat több jel között.

A **frekvenciaosztásos multiplexelés** (**Frequency Division Multiplexing, FDM**) az áteresztő sávú átvitel előnyeit használja ki a csatorna megosztásához. A frekvenciatartományt felosztja frekvenciasávokra, ahol minden felhasználó kizárólagosan birtokol bizonyos sávot a jel küldéséhez. Az AM-rádiószórás illusztrálja az FDM-et. A rendelkezésre álló frekvenciatartomány körülbelül 500 és 1500 kHz közé eső, nagyjából 1 MHz-es sáv. A különböző logikai csatornákhoz (állomásokhoz) különböző frekvenciákat rendelnek hozzá oly módon, hogy minden csatorna a rendelkezésre álló frekvenciatartománynak csak egy kis részét veszi igénybe, és a csatornák frekvenciája között elég nagy a távolság ahhoz, hogy ne zavarják egymás adásait.

Részletesebb példát a 2.25. ábra mutat, hogy a frekvenciaosztással hogyan lehet három, beszédátvitelre szánt telefonvonalat egy csatornára multiplexelni. Hangcsatornák esetén a szűrők a rendelkezésre álló sávszélességet körülbelül 3100 Hz-re korlátozzák. Ha több csatornát multiplexelnek össze, akkor a csatornák megfelelő szétválasztása érdekében 4000 Hz-et biztosítanak minden csatorna számára. A többletet **védősávnak** (**guard band**) nevezik. Ez biztosítja a csatornák megfelelő elkülönítését. Először a hangcsatornák frekvenciáját tolják el, mindegyiket különböző mértékben. Ezt követően összefogják, nyalábolják a csatornákat, mivel minden csatorna máshol helyezkedik el a frekvenciatartományban. Meg kell azonban jegyeznünk, hogy bár a sávok között van



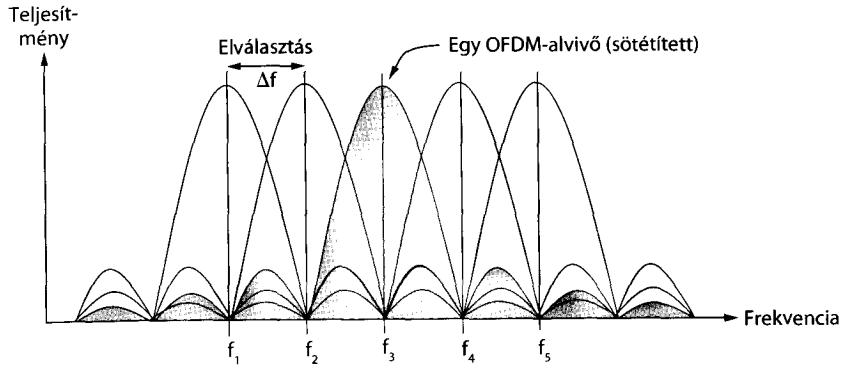
2.25. ábra. *Frekvenciaosztásos multiplexelés. (a) Az eredeti sávszélességek. (b) Növelt frekvenciájú sávszélességek. (c) Multiplexelt csatorna*

valamekkora távolság, a szomszédos csatornák mégis átlapolódnak egy kicsit, mivel a szűrők nem vágnak élesen. Ez az átlapolódás azt jelenti, hogy ha az egyik csatorna szélén megjelenik egy nagy tüske, akkor a szomszédos csatorna azt nem termikus zajnak fogja érzékelni.

Évek óta ezt a sémát használják a hívások multiplexelésére a telefonos rendszerekben, de manapság az időosztásos multiplexelést részesítik előnyben. Az FDM-et azonban továbbra is használják telefonos hálózatokban, valamint mobiltelefon-, földfelszíni vezeték nélküli és műholdas hálózatokban, finomabb felosztás esetén.

Digitális adatok küldésekor a frekvenciasáv hatékonyan felosztható védősávok alkalmazása nélkül. Az **OFDM (Orthogonal Frequency Division Multiplexing – ortogonális frekvenciaosztásos multiplexelés** esetén a csatorna több alvivőre (subcarrier) van felosztva, amelyek függetlenül küldenek adatokat (például QAM-mel). Az alvivők szorosan helyezkednek el a frekvenciatartományban. Ezért az egyes alvivőkből származó jelek a szomszédos tartományba is átérnek. Ahogy azonban a 2.26. ábrán is látható, az egyes alvivők frekvenciaválaszát úgy tervezték meg, hogy a szomszédos alvivők közepén az értékük nulla legyen. Ezért az alvivők mintavételezhetők a középfrekvenciáikban anélkül, hogy a szomszédaikkal interferálnának. Ahhoz, hogy ez működjön, védőidő szükséges a szimbólumjelek egy részének megismétléséhez, így meglesz a kívánt frekvenciaválasz. Ez a többletterhelés azonban sokkal kisebb, mint ami a sok védősávhoz szükséges.

Az OFDM ötlete már régen felmerült, de csak az utolsó évtizedben kerültek széles körben alkalmazásra, miután felismerték, hogy hatékonyan megvalósítható a digitális adatoknak az összes alvivő fölötti Fourier-traszformációjával (ahelyett, hogy minden alvivőt külön modulálnának). Az OFDM-et a 802.11, a kábelhálózatok és az elektromos



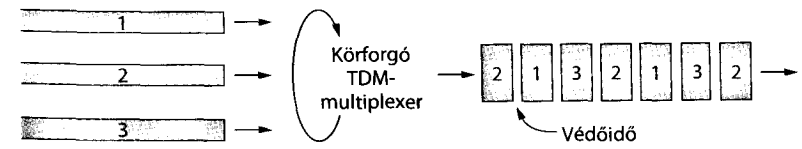
2.26. ábra. *Ortogonális frekvenciaosztásos nyalábolás (OFDM).*

hálózatok használják és a 4. generációs mobiltelefon-rendszerekhez tervezik használni. Általában a digitális információ nagy sebességű folyamát felosztják sok kisebb sebességű folyamra, amelyek párhuzamosan kerülnek átvitelre az alvivőkön. Ez a felosztás azért hasznos, mert a csatorna csillapításai egyszerűbben kezelhetők alvivő szinten. Néhány alvivő nagyon erősen csillapodhat, és kikerülhet azok közül az előnyben részesített alvivők közül, amelyek jól vehetők.

2.5.4. Időosztásos multiplexelés

Az FDM egyik alternatívája az **időosztásos multiplexelés (Time Division Multiplexing, TDM)**. Itt a felhasználók körforgó módszerrel változnak, mindegyik felhasználó adott időközönként megkapja a teljes sávszélességet egy kis időre. A 2.27. ábra 3 folyam TDM-mel történő multiplexelésére mutat példát. Az egyes bemeneti folyamoktól érkező bitek rögzített **időszeletet** kapnak és egy csoportosított folyamba kerülnek a kimeneten. Ez a folyam az egyéni folyamok összegzett sebességével fut. Ahhoz, hogy ez működjön, a folyamoknak időben szinkronizáltnak kell lenniük. A védő-frekvenciasávhoz hasonlóan **védőidő** (guard time) is alkalmazható a kis időzítési eltérésekhez való alkalmazkodás érdekében.

A TDM-et gyakran a telefon- és mobiltelefon-hálózatok részeként használják. A félreértések elkerülése érdekében tisztázzuk, hogy ez nagyban különbözik az alternatív **STDM**-től **(Statistical Time Division Multiplexing – statisztikai időosztásos multiplexelés**). A „statisztikai" előtag azt jelzi, hogy az egyéni folyamok nem rögzített ütemezés szerint járulnak hozzá a multiplexelt folyamhoz, hanem az igényük statisztikája alapján. Az STDM másik neve a csomagkapcsolás.



2.27. ábra. *Időosztásos multiplexelés (TDM)*

2.5.5. Kódosztásos multiplexelés

Van egy harmadik típusú multiplexelés, amely teljesen másképp működik, mint az FDM és a TDM. A **CDM** (**Code Division Multiplexing – kódosztásos multiplexelés**) a szórt spektrumú kommunikáció egy formája, amelyben a keskeny sávú jel szélesebb frekvenciasávban terjed. Ez jobb interferenciatűrést biztosít, és lehetővé teszi, hogy különböző felhasználók jelei ugyanazt a frekvenciasávot használják. Mivel a kódosztásos multiplexelést általában az utóbbi célra használják, ezt általában **CDMA**-nak (**Code Division Multiple Access – kódosztásos többszörös hozzáférés**) hívják.

A CDMA minden állomás számára teljes időben a teljes frekvenciasávban történő átvitelt teszi lehetővé. A több párhuzamos átvitelt kódolással választják szét. Mielőtt belevágnánk az algoritmus leírásába, vegyünk egy hasonlatot: repülőtéri váró, amelyben rengeteg pár beszélget. A TDM annak felel meg, mintha a párok egymás után beszélnének. Az FDM olyan, mintha az emberek különböző hangmagasságokban beszélnének, ki magas hangon, ki mélyen, de minden pár egyszerre, egymástól teljesen függetlenül folytat párbeszédet. A CDMA ahhoz hasonlít, mintha minden pár egyszerre beszélne, de más-más nyelven. A franciául beszélő páros csakis a francia nyelvre figyel, és minden egyebet zajként kezel. Ennek megfelelően a CDMA kulcsa az, hogy képesek legyünk kiszűrni a hasznos jelet, miközben minden egyebet eldobunk, mintha véletlenszerű zaj lenne. A következőkben a CDMA-ról egy leegyszerűsített leírást adunk.

CDMA esetén minden bitidőt m rövid intervallumra, úgynevezett **töredékre** (**chip**) osztanak. Tipikusan 64 vagy 128 töredék van bitenként, de a példánkban az egyszerűség kedvéért csak 8 töredék/bitet használunk. Minden állomáshoz egy m bites kód, más néven **töredéksorozat** (**chip sequence**) tartozik. Oktatási célból érdemes bipoláris (két feszültségszintet használó) jelölést alkalmazni ezeknek a kódoknak –1 és +1-ből álló sorozattal történő leírásához. A töredéksorozatokat zárójelbe írjuk.

Egy 1-es bit elküldéséhez az állomás elküldi a saját töredéksorozatát. Ha 0-t akar továbbítani, akkor a töredéksorozatának egyes komplemensét küldi el. Semmilyen egyéb minta használata nincs megengedve. Ilyen módon $m = 8$ esetén, ha az A állomás töredéksorozata (–1 –1 –1 +1 +1 –1 +1 +1), akkor 1-es bit küldéséhez ezt a töredéksorozatot, 0-s bit továbbításához pedig a (+1 +1 +1 –1 –1 +1 –1 –1) sorozatot használja. Valójában csak jelek kerülnek elküldésre ezzel a feszültségszinttel, de számunkra ez elég, hogy sorozatokban gondolkozzunk.

Ha a továbbítandó információ mennyiségét b b/s-ról mb töredék/s-ra növeljük minden állomásnál, akkor a CDMA által igényelt sávszélesség is m-szeresére növekszik a CDMA-t nem használó állomás által igényelt sávszélességhez képest (amennyiben sem a moduláción, sem a kódolási eljáráson nem változtatunk). Ha 100 állomás számára egy 1 MHz-es sáv áll rendelkezésre, akkor FDM-technikával mindegyiknek 10-10 kHz áll rendelkezésére, így 1 bit/Hz-cel számolva 10 kb/s-mal adhatnak. CDMA használata esetén minden állomás számára rendelkezésre áll az egész 1 MHz-es sáv, így a töredéksebesség 100 töredék/bit ahhoz, hogy az állomás 10 kb/s-os sebességét szétszórjuk a csatornán.

A 2.28.(a) és (b) ábrán a négy példaállomáshoz rendelt töredéksorozatot mutatunk be, valamint az általuk ábrázolt jeleket. Minden állomás saját egyedi töredéksorozattal rendelkezik. Használjuk az **S** jelölést az S állomás m töredéksorozatára, valamint $\overline{\mathbf{S}}$ jelölést a sorozat negáltjára. A töredéksorozatoknak páronként **ortogonálisaknak** kell lenniük, vagyis minden **S** és **T** párra a normalizált skaláris szorzatnak (amit az $\mathbf{S}\bullet\mathbf{T}$ jelöl) 0-nak kell lennie. Az ilyen ortogonális töredéksorozat előállításához használt metódust **Walsh-kódnak** nevezzük. Matematikai összefüggésekkel kifejezve a töredéksorozatok ortogonalitása:

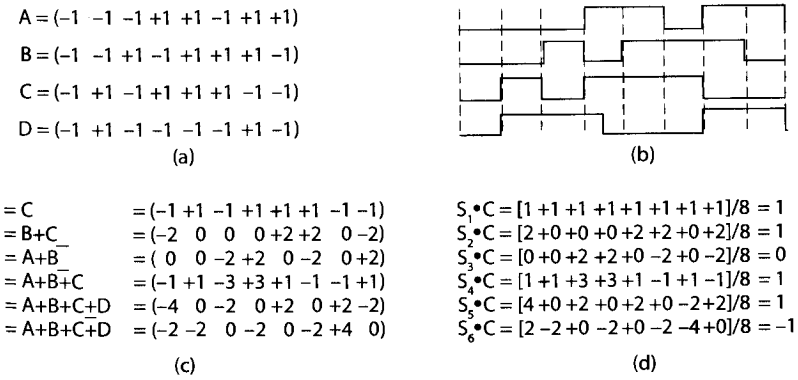
$$\mathbf{S}\bullet\mathbf{T} \equiv \frac{1}{m}\sum_{i=1}^{m} S_i T_i = 0 \tag{2.5}$$

Vagyis, ahány töredékpár egyezik, annyinak kell különböznie is a sorozatokban. A későbbiekben döntő jelentőségű lesz az ortogonális tulajdonság. Vegyük észre, hogy amennyiben $\mathbf{S}\bullet\mathbf{T} = 0$, akkor $\mathbf{S}\bullet\overline{\mathbf{T}}$ szintén 0! Minden töredéksorozatra igaz, hogy az önmagával számított normalizált skaláris szorzata 1:

$$\mathbf{S}\bullet\mathbf{S} = \frac{1}{m}\sum_{i=1}^{m} S_i S_i = \frac{1}{m}\sum_{i=1}^{m} S_i^2 = \frac{1}{m}\sum_{i=1}^{m} (\pm 1)^2 = 1$$

Ez azért van így, mert az m összetevő mindegyike 1, így az összegük m. Vegyük észre azt is, hogy $\mathbf{S}\bullet\overline{\mathbf{S}} = -1$.

Egy bitidő alatt minden állomásra igaz, hogy vagy 1-es bitet küld töredéksorozatának továbbításával, vagy 0-s bitet küld a töredéksorozat negáltjának továbbításával, vagy egyszerűen csendben marad, vagyis semmit sem továbbít. Pillanatnyilag tekintsük úgy, hogy minden állomás szinkronban van, így mindegyik töredéksorozat ugyanabban a pillanatban kezdődik. Amikor kettő vagy több állomás egyszerre ad, akkor a bipoláris jeleik lineárisan összeadódnak. Ha például egy töredékperiódus alatt három állomás +1-et, egy pedig –1-et ad, akkor az eredmény +2 lesz. Úgy is gondolhatunk erre, mint feszültségek összegére: három állomás +1 voltot, egy pedig –1 voltot ad a kimenetén, ami 2 voltot eredményez. A 2.28.(c) ábrán hat olyan példát láthatunk, amelyek egy vagy több állomás egyidejű forgalmazása mellett jöttek létre. Az első példában a C állomás ad 1-es



2.28. ábra. *(a) Négy állomás töredéksorozata. (b) A sorozatok által ábrázolt jelek. (c) Hat példaátvitel. (d) C állomás jelének visszaállítása*

bitet, így egyszerűen *C* saját töredéksorozatát kapjuk meg. A második példában *B* és *C* egyaránt 1-es bitet adnak, így a bipoláris töredéksorozatuk összegét kapjuk, részletezve:

$$(-1\ -1\ +1\ -1\ +1\ +1\ +1\ -1) + (-1\ +1\ -1\ +1\ +1\ +1\ -1\ -1) = (-2\ 0\ 0\ 0\ +2\ +2\ 0\ -2)$$

Ahhoz, hogy egy adott állomás által generált bitsorozatot visszaállíthassunk, ismernünk kell még annak töredéksorozatát. A visszaállítás elvégezhető, ha a vett sorozat (az összes forgalmazó állomás jeleinek lineáris összege) és a figyelni kívánt állomás töredéksorozatának normalizált skaláris szorzatát képezzük. Ha tehát a vett sorozat **S**, a vevő pedig a **C** töredéksorozatú állomásra figyel, akkor egyszerűen kiszámítja az $\mathbf{S} \bullet \mathbf{C}$ normalizált skaláris szorzatot.

Ahhoz, hogy megértsük, hogyan működik az eljárás, tegyük fel, hogy egyszerre *A* és *C* 1-et, míg *B* 0-t küld! A vevő az $\mathbf{S} = \mathbf{A} + \overline{\mathbf{B}} + \mathbf{C}$ sorozatot látja, és a következő számítást végzi:

$$\mathbf{S} \bullet \mathbf{C} = (\mathbf{A} + \overline{\mathbf{B}} + \mathbf{C}) \bullet \mathbf{C} = \mathbf{A} \bullet \mathbf{C} + \overline{\mathbf{B}} \bullet \mathbf{C} + \mathbf{C} \bullet \mathbf{C} = 0 + 0 + 1 = 1$$

Az első két tag eltűnik, mivel a töredéksorozatokat előrelátóan, a (2.5) képletnek megfelelően, páronként ortogonálisnak választottuk meg. Most már érthető, miért is volt olyan fontos ez a tulajdonság a töredéksorozatok megválasztásánál.

Azért, hogy egyértelművé tegyük a dekódolás menetét, vegyük ismét elő a 2.28.(d) ábra hat példáját! Tegyük fel, hogy a vevő mind a hat összegzett kódból (S_1-től S_6-ig) a *C* állomás által küldött biteket szeretné kinyerni! Kiszámítja a vett **S** sorozatok és a 2.28.(a) ábra **C** töredéksorozatának skaláris szorzatait, majd veszi ezek 1/8-át, mivel jelen esetben $m = 8$. A példa tartalmaz olyan eseteket, amikor a *C* csendes, amikor 1 bitet küld, és amikor 0 bitet küld, önmagában és más átvitelekkel kombinálva. Mint láthatjuk, minden alkalommal a helyes bitet sikerült dekódolni. Tisztára olyan, mintha franciául beszélnénk!

Elméletben elegendő kapacitás esetén a vevő az összes adóra egyszerre tud figyelni, ha mindegyikre egyszerre futtatja a dekódoló algoritmust. Ezt azonban könnyebb mondani, mint a valós életben megvalósítani, és hasznos tudni, hogy melyik adó adhat éppen.

A tanulmányozott ideális, vagyis zajmentes CDMA-rendszerben a párhuzamosan adó állomások száma korlátlanul nagy lehet, hosszabb töredéksorozatok használatával. 2^n állomás esetén a Walsh-kód 2^n darab 2^n hosszúságú ortogonális töredéksorozatot tud biztosítani. Jelentős korlátozás azonban, hogy szinkronizált vevőket feltételeztünk. A szinkronizálás közel sem igaz néhány alkalmazás esetén, mint például mobiltelefon-hálózatok esetén (amelyekben az 1990-es évek elejétől kezdve a CDMA-t széles körben alkalmazzák). Ez különböző kialakításokhoz vezet. Ehhez a témakörhöz ebben a fejezetben később még visszatérünk és leírjuk, hogy az aszinkron CDMA miben különbözik a szinkron CDMA-tól.

A CDMA-t mobiltelefon-hálózatok mellett a műholdak és kábelhálózatok is használják. A rövid bevezetőben számos, problémát okozó tényezővel nem foglalkoztunk. Részletesebb leírást Viterbi [1995], valamint Lee és Miller [1998] munkája tartalmaz. Ezeknek a hivatkozott munkáknak a megértéséhez azonban jelentős mennyiségű távközlési alapismeret szükséges.

2.6. A nyilvános kapcsolt telefonhálózat

Amikor két olyan számítógép között kell kapcsolatot teremteni, amelyek ugyanahhoz a céghez vagy szervezethez tartoznak, és elég közel vannak egymáshoz, akkor a legegyszerűbb megoldás az, ha a két gépet egy vezetékkel közvetlenül összekötjük. Így működnek a lokális hálózatok. Ha viszont a távolságok már nagyok, több gépről van szó, vagy a vezetékeknek közutakat, közterületeket kellene keresztezniük, akkor a magánvezetékek lefektetése szinte megfizethetetlenül drága. Ráadásul, a legtöbb országban tilos magánvezetékeket köztulajdonban levő területet keresztezve vagy az alatt vezetni. Következésképpen a hálózattervezők kénytelenek igénybe venni a már meglevő távközlési eszközöket.

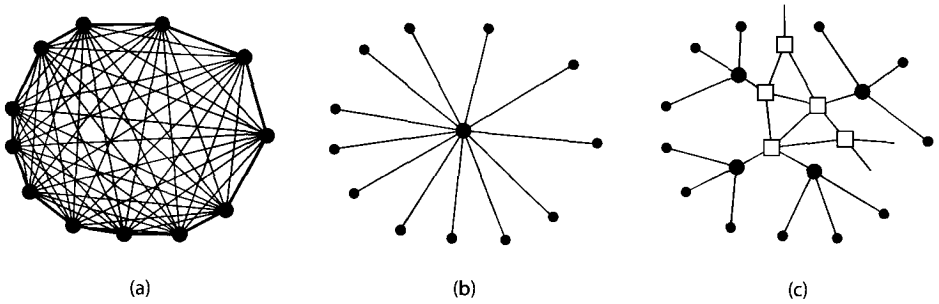
Ezeket az eszközöket – kiváltképp a **nyilvános kapcsolt telefonhálózatot** (**Public Switched Telephone Network, PSTN**) – rendszerint korábban tervezték teljesen más célra, mégpedig emberi beszéd többé-kevésbé felismerhető módon történő továbbítására. Ezeknek a távközlési eszközöknek a számítógépek közötti kommunikáció szempontjából nincs lényeges szerepe. Hogy érzékeltessük a probléma nagyságrendjét, tételezzük fel, hogy egy olcsó kommersz kábel fut a két számítógép között, amely 1 G/s-os vagy gyorsabb adatátvitelre képes. Ezzel ellentétben egy tipikus ADSL, a telefonmodem gyors alternatívája, körülbelül 1 M/s-os sebességre képes. A különbség a kettő között akkora, mint egy repülőgépes utazás és egy kellemes kószálás sebessége között.

Bizonyos esetekben a távbeszélőrendszer annyira összefonódik a (nagy távolságú) számítógép-hálózatokkal, hogy érdemes egy kis időt szentelni a bemutatásukra. A korlátozó tényezőt nem a telefonhálózatban lévő trönkök és kapcsológépek jelentik, hanem az „utolsó néhány kilométer", amelyen keresztül az ügyfél a hálózathoz csatlakozik. Ez a helyzet változik, ahogy az üvegszálas és digitális technikákat fokozatosan bevezetik a hálózat szélén, de ez idő- és pénzigényes folyamat. A hosszú várakozás során azok a számítógéprendszer-tervezők, akik legalább három nagyságrenddel jobb teljesítményű rendszerekkel szoktak dolgozni, időt és fáradságot nem kíméltek, hogy kitalálják, hogyan lehetne hatékonyabban használni a telefonhálózatot.

A következő szakaszokban bemutatjuk a telefonhálózatot és annak működését. A telefonrendszer belső működésével kapcsolatos további információért lásd Bellamy [2000] munkáját.

2.6.1. A távbeszélőrendszer felépítése

Amikor Alexander Graham Bell 1876-ban feltalálta a telefont (éppen pár órával vetélytársa, Elisha Gray előtt), már óriási szükség volt a találmányára. Kezdetben az üzletet csak a párosával árusított telefonkészülékek jelentették. A készülékek közötti vezeték kihúzása a felhasználó feladata volt. Ha egy telefontulajdonos *n* számú másik telefontulajdonossal akart beszélni, akkor mind az *n* házhoz külön vezetéket kellett kihúznia. Egy év múlva a városokat vadul behálózták a háztetők és a fák között kifeszített vezetékek. Hamarosan nyilvánvalóvá vált, hogy a 2.29.(a) ábrán bemutatott modell, amelyben minden egyes telefon az összes többi telefonnal össze van kötve, nem fog működni.



2.29. ábra. *Különböző távbeszélőrendszer-topológiák. (a) Teljesen összekapcsolt hálózat. (b) Központosított kapcsoló. (c) Kétszintű hierarchia*

Bell felismerte ezt a problémát, és megalapította a Bell Telefontársaságot, amely 1878-ban létrehozta az első telefonközpontot (a connecticuti New Havenben). A társaság minden ügyfél házához vagy irodájához kihúzott egy vezetéket. Telefonálás előtt az ügyfélnek meg kellett forgatnia egy kart a készüléken. Ennek hatására a telefontársaság központjában megszólalt egy csengő, ami a telefonkezelőnek jelzett. Ezt követően a kezelő egy kapcsolókábel (jumper cable) segítségével manuálisan összekötötte a hívó és a hívott fél vezetékét. Egy ilyen kezdetleges távbeszélőrendszer modelljét láthatjuk a 2.29.(b) ábrán.

Rövid időn belül mindenfelé megjelentek a Bell System kapcsolóközpontjai (switching office). Az emberek hamarosan már városok közötti távolsági hívásokat akartak lebonyolítani, így a telefontársaságnak össze kellett kapcsolnia a kapcsolóközpontokat is. A korábbi probléma azonban újra előkerült; a kapcsolóközpontok között kihúzott vezetékek hamarosan teljesen kezelhetetlenné váltak, ezért egy másodszintű kapcsolóközpontot kellett kialakítani. Kis idő múlva már több másodszintű kapcsolóközpontra volt szükség, ahogy ez a 2.29.(c) ábrán is látható. Végül is a távbeszélőrendszer hierarchiája ötszintű lett.

1890-re a távbeszélőrendszer három fő része már mind a helyén volt: (1) a kapcsolóközpontok, (2) az ügyfelek és a kapcsolóközpontok közti vezetékek (amelyek ma már kiegyenlített, szigetelt sodrott érpárok, szemben a régi egyvezetékes, földben záródó áramkörökkel), valamint (3) a telefonközpontok közötti nagy távolságú összeköttetések. A távbeszélőrendszerről rövid történeti áttekintést kaphatunk Hawley [1991] művében.

Bár azóta mindhárom területen történtek fejlesztések, a Bell System eredeti hálózata lényegében érintetlen maradt az elmúlt 100 év során. A most következő leírás nagyon leegyszerűsített, de ennek ellenére megragadja a lényeget. Minden telefonból két rézvezeték indul ki, amelyek egyenesen a telefontársaság legközelebbi **helyi központjába** (**local central office**) vagy más néven **végközpontjába** (**end office**) futnak be. Ez a távolság a városokban rövidebb, a ritkán lakott területeken hosszabb, de mindkét esetben általában 1 és 10 km között van. Csak az Egyesült Államokban körülbelül 22 000 helyi központ van. Azt a kéteres vezetéket, amely az egyes felhasználók telefonkészülékei és a helyi központ között halad, **előfizetői szakasznak** vagy **előfizetői huroknak** (**local loop**) hívják. Ha a világon levő összes előfizetői hurkot egymáshoz kötnénk, az így kapott vezeték ezerszer elérne a Holdig és vissza.

Volt idő, amikor az AT&T vagyonának 80%-át tették ki az előfizetői hurkok rézvezetékei. Akkoriban tulajdonképpen az AT&T volt a világ legnagyobb „rézbányája". Szerencsére ez a tény nem volt túl ismert a befektetői világban. Ugyanis, ha ezt megtudta volna valamelyik cégbefektető, akkor az megvette volna az AT&T-t, felszámolta volna a telefonszolgáltatást, kiszedte volna a vezetékeket a földből, és gyors haszon reményében eladta volna azokat egy színesfém-feldolgozónak.

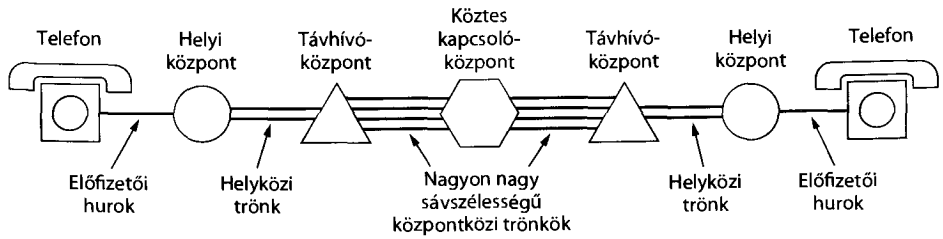
Ha egy adott helyi központhoz kapcsolódó állomásról olyan állomást hívunk, amelyik ugyanahhoz a helyi központhoz kapcsolódik, akkor a kapcsolás során a két előfizetői hurok között közvetlen elektromos kapcsolat jön létre a helyi központon belül. Ez a kapcsolat a hívás ideje alatt végig fennmarad.

Ha viszont a hívott fél készüléke egy másik helyi központhoz kapcsolódik, akkor az előző módszer nem használható. Minden helyi központ kapcsolatban áll egy vagy több közeli ún. **távhívóközponttal** (**toll office**), amit **tandemközpontnak** (**tandem office**) is hívnak, ha ugyanazon a körzeten belül helyezkedik el, mint a helyi központ. A helyi és a távhívóközpont közötti vonalakat **helyközi trönköknek** (**toll connecting trunk**) hívják. A különböző kapcsolóközpontok száma és topológiája az országos telefonhálózat sűrűségétől függően országonként változik.

Amennyiben a hívó és a hívott fél helyi központja ugyanahhoz a távhívóközponthoz csatlakozik a helyközi trönkön keresztül (ami az egymáshoz közeli helységek közötti távolsági hívásoknál gyakran megesik), akkor az összeköttetés a távhívóközponton belül jön létre. A 2.29.(c) ábrán egy olyan telefonhálózatot láthatunk, amely csak telefonkészülékeket (kis pontok), helyi központokat (nagy pontok) és távhívóközpontokat (négyszögek) tartalmazza.

Ha a hívó és a hívott fél nem ugyanahhoz a távhívóközponthoz tartozik, akkor a kapcsolási hierarchiában egy még magasabb szinten jön létre az összeköttetés. A távhívóközpontok elsődleges, körzeti és regionális kapcsolóközpontokból álló hálózaton keresztül, nagy sávszélességű **központközi trönkök** (**intertoll trunk** vagy **interoffice trunk**) segítségével kapcsolódnak egymáshoz. Az AT&T 1984-es feldarabolása előtt az USA távbeszélőrendszere hierarchikus útválasztást használt, egy útvonal megtalálásához feljebb kellett lépni a hierarchián egy közösen használt kapcsolóközpontig. Ezt azután egy rugalmasabb, nem hierarchikus útválasztás váltotta fel. A 2.30 ábra bemutatja egy nagy távolságú összeköttetés lehetséges útvonalát.

A távközlésben igen sokféle átviteli közeget használnak. A modern irodaépületekkel szemben, amelyekben általában 5-ös kategóriájú (Cat 5) vezetékeket használtak, az előfizetői hurkok többségében 3-as kategóriájú (Cat 3) sodrott érpárból állnak, az üvegszálak csak éppen kezdenek megjelenni. A kapcsolóközpontok között koaxiális kábeleket, mikrohullámú összeköttetést, leggyakrabban pedig optikai kábeleket használnak.



2.30. ábra. *Tipikus áramköri út egy nagy távolságú hívás esetén*

A múltban a telefonrendszer teljes egészében analóg volt, és a tényleges beszédjelet villamos feszültségjel formájában továbbította a forrástól a célig. Az üvegszálak, a digitális elektronika és a számítógépek megjelenése óta az összes trönk és kapcsoló digitális lett, így az előfizetői hurok maradt a rendszer utolsó analóg továbbítású része. A digitális átvitel azért előnyös, mert így egy nagy távolságot áthidaló hívásban nem szükséges nagy pontossággal olyan analóg hullámformákat visszaállítani, amelyek már sok erősítőn keresztülhaladtak. Elég, ha helyesen meg tudjuk különböztetni az 1-est a 0-tól. Ez a tulajdonság a digitális átvitelt megbízhatóbbá teszi az analógnál. A digitális rendszer mindezen felül olcsóbb is és könnyebben karbantartható.

Összefoglalva, a telefonrendszer három fő összetevőből épül fel:

1. előfizetői hurkok (a házakhoz és az irodákhoz menő analóg sodrott érpárok),

2. trönkök (a kapcsolóközpontokat összekötő digitális üvegszálak),

3. kapcsolóközpontok (ahol a hívásokat átteszik az egyik trönkről a másikra).

Mielőtt visszatérnénk ennek a három összetevőnek a részletes vizsgálatához, egy kis kitérőt teszünk a telefonok politikai vonatkozásainak területére. Az előfizetői hurkok biztosítják mindenki számára a rendszer elérését, így kritikus darabjai a rendszernek. Sajnos ezek egyben a rendszer leggyengébb láncszemei is. A nagy távolságokat áthidaló trönkökkel kapcsolatban a legfőbb kérdés az, hogy hogyan fogjuk össze a sok hívást, amelyet azután ugyanazon a szálon továbbítunk. Ezt a feladatot multiplexelésnek (multiplexing) nevezik, és ennek megvalósítására FDM-et és TDM-et használnak. Végül pedig a kapcsolásnak két alapvetően különböző módja van, amelyeket szintén tanulmányozni fogunk.

2.6.2. Távközlési politika

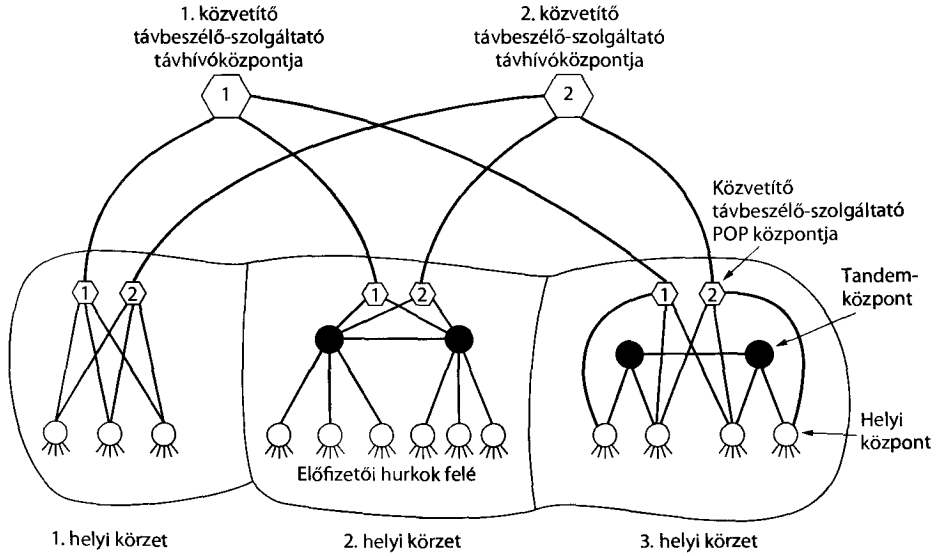
1984 előtt évtizedekig a Bell System látta el mind a helyi, mind a nagy távolságú szolgáltatásokat az Egyesült Államok legnagyobb részén. Az 1970-es években az amerikai kormány kezdte úgy érezni, hogy a Bell System utódja, az AT&T jogtalan monopóliumra tör, ezért pert indított ellene, és a feldarabolását kezdeményezte. A kormány megnyerte a pert, és 1984. január 1-jétől az AT&T-t több vállalatra felosztották. Létrejött az AT&T Long Lines, megalakult 23 **Bell Üzemeltető Vállalat** (**Bell Operating Company, BOC**) és még néhány kisebb cég. A 23 BOC több regionális BOC-ba (RBOC) tömörült a gazdaságosabb üzemeltetés érdekében. Az Egyesült Államok távközlési rendszerének jellege egy bírósági ítélet (*nem* pedig a törvényhozás döntése) következtében pillanatok alatt megváltozott.

A vagyonátadás pontos részleteit az ún. **MFJ- (Modified Final Judgement - módosított végső ítélet)** dokumentum tartalmazza, ami egy oximoron[2] – ha az ítélet módosítható, akkor nyilvánvalóan nem a végső. Mindenesetre ez az ítélet versenyhelyzetet teremtett, javult a szolgáltatások minősége, az árak mind az egyéni felhasználók, mind a cégek számára csökkentek. A helyi szolgáltatás ára azonban növekedett, mivel a távolsági hívásokból származó keresztfinanszírozások eltávolításra kerültek, és a helyi szolgáltatásnak önfenntartónak kell lennie. Sok más országban is azt fontolgatják, hogy versenyhelyzetet teremtenek a távbeszélőrendszerek területén.

2 Az oximoron szónoki fogás, amelyben az állítás és ellenállítás együtt szerepel az ellentmondás kihangsúlyozására. *(A lektor megjegyzése)*

A tanulmányunkhoz közvetlenül az tartozik, hogy az új versengő keretrendszer egy kulcsfontosságú technikai képességgel bővítette a telefonhálózat architektúráját. Hogy világos legyen, ki mit tehet, az Egyesült Államok területét 164 **helyi körzetre** (**Local Access and Transport Areas, LATA**) osztották fel. A LATA nagyjából akkora terület, mint amennyit egy körzetszám lefed. Minden helyi körzeten belül van általában egy **helyi szolgáltató** (**Local Exchange Carrier, LEC**), amely a helyi körzeten belül a hagyományos beszédátviteli szolgáltatáson belül monopóliummal rendelkezik. A legfontosabb helyi szolgáltatók a Bell Üzemeltető Vállalatok, de van olyan helyi körzet, amelyben a helyi szolgáltatóként üzemelő 1500 telefontársaságból is van egy-kettő.

A helyi körzetek közötti összes forgalom lebonyolítását a **közvetítő szolgáltatók** (**IntereXchange Carriers, IXC**) végzik. Kezdetben az AT&T Long Lines volt az egyetlen jelentős közvetítő szolgáltató, de ma már a Verizon és a Sprint a két legtőkeerősebb versenytárs ezen a területen. Az AT&T feldarabolásakor az egyik cél az volt, hogy egyenlő feltételeket biztosítson valamennyi közvetítő szolgáltatónak a vonalak minőségét, a tarifákat és telefonszámok hosszát illetően. A rendszer felépítését a 2.31. ábra szemlélteti. Az ábrán három helyi körzetet láthatunk, mindhárom több helyi központtal rendelkezik. A 2-es és 3-as helyi körzetben a tandemközpontok hierarchikusan épülnek egymásra (ezek a helyi körzeten belüli távhívóközpontok).



2.31. ábra. *A helyi körzetek, a helyi szolgáltatók és a közvetítő szolgáltatók közötti kapcsolat. A körök helyi szolgáltatók telefonközpontjait jelölik. A hatszögek a közvetítő szolgáltatókat jelölik*

Ha egy közvetítő szolgáltató (IXT) valamelyik helyi körzetből (LATA) szeretne hívásokat kezdeményezni, akkor egy **POP** (**Point of Presence – szolgáltatási pont**) kapcsolóközpontot kell kiépítenie. Minden közvetítő szolgáltatót és valamennyi helyi központot össze kell kötnie egy helyi szolgáltatónak (LEC). Ez vagy közvetlenül történik, mint az 1-es, 3-as helyi körzetben, vagy közvetve, mint a 2-es helyi körzetben. Ráadásul az összes közvetítő szolgáltató esetén a felépített kapcsolatnak mind technikailag, mind pénzügyileg azonos paraméterekkel kell rendelkeznie. Ily módon egy előfizető mondjuk az 1-es helyi körzetben tetszőlegesen megválaszthatja, hogy melyik közvetítő szolgáltatót használja, amikor egy 3-as helyi körzetbeli előfizetőt akar felhívni.

Az MFJ-dokumentum azt is tartalmazta, hogy a közvetítő szolgáltatóknak tilos a helyi telefonszolgáltatásban, a helyi szolgáltatóknak pedig a helyi körzetek közötti telefonszolgáltatásban részt venni. Ugyanakkor bármilyen más üzleti tevékenységet (például rántott csirkét forgalmazó étteremhálózat üzemeltetése) egyaránt végezhettek. 1984-re a kép teljesen letisztult. Szerencsére azonban a technika fejlődése túllépett a jogon. Sem a kábeltelevízióról, sem a mobiltelefonról nem szólt a dokumentum. Ahogy a kábeltelevíziózás fejlődött, és a mobiltelefon-rendszer népszerűsége robbanásszerűen nőtt, mind a helyi, mind a közvetítő szolgáltatók elkezdték felvásárolni a kábeltévés és a mobiltelefonhálózatok üzemeltetőit, vagy egyszerűen csak egyesültek azokkal.

1995-ben az amerikai törvényhozás felismerte, hogy a különböző szolgáltatások szétválasztása tovább már nem tartható fenn, ezért rendeletet adott ki arról, hogy a kábeltévé-társaságok, a helyi telefontársaságok, a nagy távolságú szolgáltatók és a mobiltelefon-hálózatok üzemeltetői részt vehetnek egymás üzleti vállalkozásaiban. Az volt az elképzelés, hogy minden cégnek legyen lehetősége egy olyan integrált szolgáltatáscsomagot nyújtani az ügyfeleinek, amely tartalmazza a kábeltévét, a telefont és egyéb informatikai szolgáltatásokat. Cél volt még az is, hogy a különböző cégek árban és a szolgáltatások minőségében egymással versenyezzenek. A rendeletet 1996 februárjában iktatták törvénybe. Ennek eredményeképpen néhány Bell-üzemeltető (BOC) közvetítő szolgáltatóvá (IXC) vált, míg néhány másik vállalat, például a kábeltévé-üzemeltetők, a helyi telefonbeszélgetések piacán kezdtek versenyezni a helyi szolgáltatókkal (LEC).

Az 1966-os törvény egyik érdekes tulajdonsága az a kitétel, hogy a helyi szolgáltatóknak helyi számhordozhatóságot kell biztosítaniuk. Ez azt jelenti, hogy az előfizetők anélkül válthatnak szolgáltatót, hogy a kapcsolási számukat is meg kellene változtatniuk. A mobiltelefonszámok hordozhatósága (a rögzített és mobilvonalak között) követte ezt a trendet 2003-ban. Ez a rendelkezés nagy akadályt hárít el az előfizetők útjából, és sokkal hajlamosabbá teszi őket helyi szolgáltatójuk lecserélésére. Ennek eredményeképpen az amerikai telekommunikációban nagyobb versenyhelyzet alakult ki, és ezt a példát néhány más ország is elkezdte követni. A többi ország azonban megvárja, hogy egy ilyen kísérlet hogyan sül el Amerikában, mielőtt a módszert a saját területén is alkalmazná. Ha jól működik, akkor ugyanazt kell tenni máshol is, ha pedig rosszul, akkor valami mást kell megpróbálni.

2.6.3. Az előfizetői hurok: modemek, ADSL és üvegszál

Itt az ideje, hogy belekezdjünk a telefonrendszer működésének részletes tárgyalásába. A tárgyalást kezdjük azzal a résszel, ami a legtöbb olvasónak ismerős: a kéteres előfizetői hurokkal, amely a telefontársaság helyi központját és a magánházakat köti össze. Az előfizetői hurkot „utolsó kilométer"-ként is emlegetik annak ellenére, hogy hossza akár több kilométer is lehet. Az előfizetői hurok több mint 100 évig analóg jelzésrendszert használt, és ez az elkövetkező néhány évben még valószínűleg így is marad, mivel a digitális rendszerre való átállás drága.

Sok erőfeszítést szenteltek annak, hogy a már telepített réz előfizetői hurkokon minél több adatot tudjanak átküldeni. A telefonmodemek digitális adatokat küldenek a számítógépek között azon a keskeny csatornán keresztül, amelyet a telefonhálózat biztosít a hanghíváshoz. Ezeket régebben széleskörűen használták, de mostanra nagyrészt felváltották az olyan széles sávú technikák, mint amilyen például az ADSL, amely újrahasznosítja az előfizetői hurkot digitális adatok küldésére az ügyféltől a helyi központig, ahol az adatokat kiszedik és továbbítják az internet irányába. A modemeknek és az ADSL-nek egyaránt kezelnie kell a régi előfizetői hurkok korlátait: a viszonylag kis sávszélességet, a jelek csillapítását és torzulását, valamint az elektromos zajra való érzékenységet, mint amilyen zaj például az áthallás.

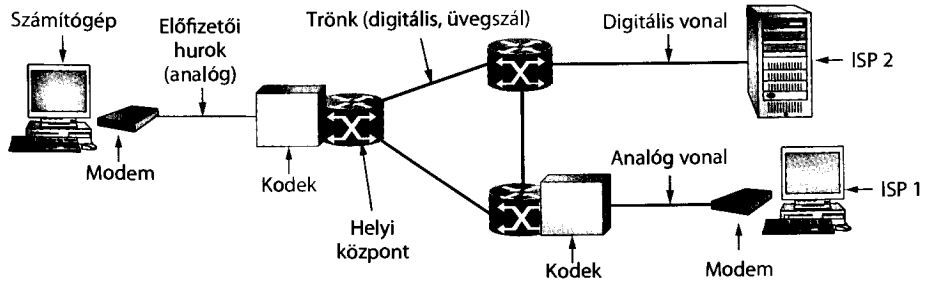
Bizonyos helyeken az előfizetői hurkot modernizálták az üvegszál kihúzásával az otthonokig (vagy azok közelébe). Az üvegszál a jövő technológiája. Ezek a telepítések támogatják a számítógép-hálózatokat a legelejétől kezdve olyan előfizetői hurokkal, amely elegendő sávszélességet biztosít az adatszolgáltatásokhoz. A korlátozó tényező nem az előfizetői hurok fizikai kialakítása, hanem az, hogy az emberek mit fognak kifizetni.

Ebben a részben az előfizetői hurkot fogjuk tanulmányozni, a régit és újat egyaránt. Itt foglalkozunk telefonmodemekkel, az ADSL-lel, valamint az üvegszálas kialakítással.

Modemek

Ahhoz, hogy biteket lehessen átküldeni az előfizetői hurkon, vagy más erre szolgáló fizikai csatornán, a biteket analóg jellé kell alakítani, amelyek átvihetők a csatornán. Ezt az átalakítást az előző fejezetben tárgyalt digitális modulációs módszerekkel valósítják meg. A csatorna másik végén az analóg jelet visszaalakítják bitté.

Azt a készüléket, amely egy digitális bitsorozatot átalakít egy, a biteket ábrázoló analóg jelsorozattá, **modemnek** hívják, amely szó a *mo*dulátor és a *dem*odulátor szavakból képzett rövidítés. Különböző típusú modemek léteznek: telefonmodemek, DSL-modemek,



2.32. ábra. *Analóg és digitális átvitel használata két számítógép közötti hívás során. Az átalakításokat a modemek és a kodekek végzik*

kábelmodemek, vezeték nélküli modemek stb. A modem beépíthető számítógépbe (amely telefonmodemek esetén általános) vagy külön dobozba (amely DSL- és kábelmodemek esetén általános). Logikailag a modem a (digitális) számítógép és az (analóg) telefonrendszer közé kerül, ahogy azt a 2.32. ábra mutatja.

A telefonmodemeket arra használják, hogy biteket küldjenek a két számítógép között a beszédátvitelre szánt telefonvonalon, a beszéd helyett, amely általában kitölti a vonalat. A fő nehézség ennek végrehajtásában az, hogy a beszédátvitelre szánt vonal 3100 Hz-re van korlátozva, amely a beszéd átvitelére elegendő. Ez a sávszélesség azonban majdnem négy nagyságrenddel kisebb, mint az Ethernet vagy 802.11 (Wi-Fi) által használt sávszélesség. Nem meglepő módon a telefonmodemek adatsebessége szintén négy nagyságrenddel kisebb, mint az Ethernet vagy a 802.11 hálózat adatsebessége.

Nézzük a számokat, hogy kiderüljön, miért ez a helyzet. A Nyquist-tétel azt mondja ki, hogy még egy tökéletes 3000 Hz-es vonalon (a telefonvonal határozottan nem ilyen) sincs értelme 6000 baudnál gyorsabban küldeni mintát. A gyakorlatban a legtöbb modem 2400 szimbólum/s vagy 2400 baud sebességgel küld, és arra összpontosít, hogy minél több bitet tudjon egy szimbólumba sűríteni, miközben a forgalmat egyszerre engedélyezi mindkét irányba (különböző technikákat használva a különböző irányokhoz).

A legegyszerűbb 2400 b/s-os modem 0 voltot használ a logikai 0, és 1 voltot a logikai 1 ábrázolására, 1 bitet szimbólumonként. Egy lépéssel feljebb négy különböző szimbólumot tud használni, mint a QPSK négy fázisában, így 2 bit/szimbólum esetén 4800 b/s-os adatsebesség érhető el.

A nagyobb sebességek a technika fejlődésével váltak elérhetővé. A nagyobb sebesség nagyobb mintahalmazt vagy **konstellációt** igényel. Sok szimbólumnál már kis mennyiségű zaj is az észlelt amplitudóban vagy fázisban hibát okozhat. A hibalehetőség csökkentése érdekében, a nagyobb sebességű modemek szabványai a minták egy részét hibajavításra használják. Ezeket a sémákat **TCM**-nek (**Trellis Code Modulation – Trellis-kódmoduláció**) hívják [Ungerback, 1987].

A **V.32** modemszabvány 32 csillagképpontot használ 4 adatbit átviteléhez és 1 ellenőrzőbitet, így 2400 baud mellett 9600 b/s-os adatsebességet ér el hibajavítással. A 9600 b/s után a következő lépés a 14 400 b/s. Ez a **V.32 bis** szabvány, amelyik 6 adatbitet, illetve 1 paritásbitet visz át szimbólumonként 2400 baud jelsebességgel. Ez után következik a **V.34**, amely 28 800 b/s-on üzemel és 12 adatbitet továbbít szimbólumonként 2400 bauddal. A csillagkép itt 1000 pontot tartalmaz. Az utolsó modemtípus ebben a sorozatban a **V.34 bis**, amely 14 adatbitet használ szimbólumonként 2400 baudos jelsebesség mellett, így 33 600 b/s-ot ér el.

Miért állunk meg itt? Annak oka, hogy a szabványos modemek megállnak 33 600 b/s-os sebességnél az, hogy a telefonrendszer Shannon-korlátja körülbelül 35 kb/s, az előfizetői hurok átlagos hossza és e vonalak minősége alapján. Ennél nagyobb sebesség megsértené a fizika törvényeit (termodinamika).

Egyféleképp azonban változtatható a helyzet. A telefontársaság helyi központjában az adatokat digitális formátumra alakítják át a telefonhálózaton való átküldéshez (a telefonhálózat gerinchálózati része már régen át lett alakítva analógról digitálisra). A 35 kb/s-os korlát arra az esetre vonatkozik, ahol két előfizetői hurok van, a telefonvonal mindkét végén egy-egy. Ezek közül mindegyik zajt ad a jelhez. Ha meg tudnánk szabadulni az egyik előfizetői huroktól, akkor növelni lehetne a jel/zaj arányt, és a maximális adatsebesség megduplázható lenne.

Ezzel a megoldással működnek az 56 kb/s-os modemek. Egyik végen jellemzően egy internetszolgáltató található, amely kiváló minőségű digitális adatfolyamot kap a legközelebbi helyi központtól. Így, ha a kommunikáció egyik végén a jel kiváló minőségű, ahogy ez mostanság a legtöbb internetszolgáltató esetén fennáll, a maximum adatsebesség akár 70 kb/s lehet. A két otthoni felhasználó között, akik modemmel és analóg vonallal rendelkeznek, a maximális adatsebesség továbbra is 33,6 kb/s.

Annak oka, hogy 56 kb/s-os modemet használnak (a 70 kb/s-os modem helyett), a Nyquist-tételre vezethető vissza. A telefoncsatorna a telefonrendszeren belül digitális mintákat visz át. Minden telefoncsatorna 400 Hz széles, ha a védősávot is számítjuk. Ennek visszaalakításához szükséges minták száma másodpercenként 8000. A mintánkénti bitek száma az Egyesült Államokban 8, amelyek közül egyet vezérlési célokra használnak, így 56 000 b/s jut a felhasználói adatoknak. Európában mind a 8 bit a felhasználók rendelkezésére áll, így itt 64 000 b/s-os modemeket is lehetne használni, de az 56 000-es sebességet választották annak érdekében, hogy nemzetközi egyetértés legyen a szabványban.

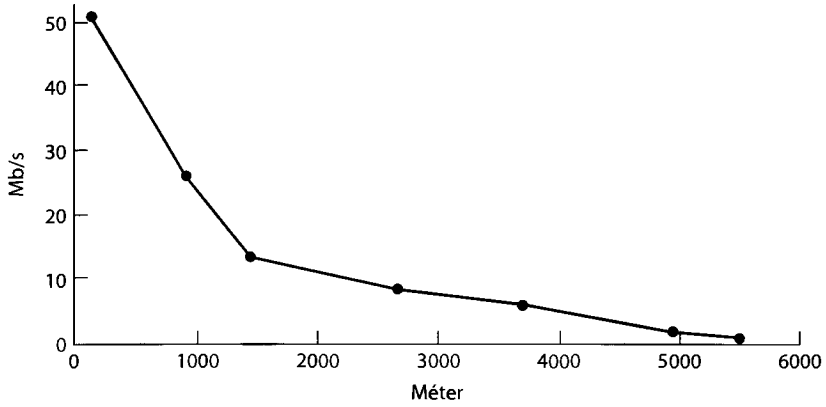
A végeredmény a **V.90** és a **V.92** modemszabvány. Ezek feltöltésre (a felhasználótól az internetszolgáltató felé) 33,6 illetve 46 kb/s-os sebességű csatornát biztosítnak, letöltésre pedig (az internetszolgáltatótól a felhasználó felé) 56 kb/s-osat. Az aszimmetria oka, hogy általában több adat érkezik az internetszolgáltatótól a felhasználóhoz, mint a másik irányba. Ez azt is jelenti, hogy a korlátozott sávszélességből több foglalható le a letöltésre annak érdekében, hogy megnöveljék az esélyt a tényleges 56 kb/s-os működésre.

Digitális előfizetői vonalak

Amikor a telefonos ipar végre eljutott az 56 kb/s-ig, elégedetten megveregette a saját vállát a jól végzett munkáért. Mindeközben a kábeltévéipar 10 Mb/s-os sebességet kínált a megosztott kábeleken, és a műholdas cégek már az 50 Mb/s feletti ajánlataikat tervezték. Ahogyan az internet-hozzáférés az üzletmenetük egyre fontosabb elemévé vált, a telefontársaságok (LEC-k) kezdték észrevenni, hogy versenyképesebb termékre van szükségük. A válaszlépésük az volt, hogy új, digitális szolgáltatásokat ajánlottak az előfizetői hurkon keresztül.

Eredetileg több, egymást átfedő nagy sebességű ajánlat is volt, amelyek közül mindegyik egy **xDSL** (**Digital Subscriber Line – digitális előfizetői vonal**) alakú nevet kapott, az *x* helyén különböző betűkkel. A hétköznapi telefonvonalnál nagyobb sávszélességű szolgáltatásokat néha **széles sávúnak** (**broadband**) is nevezik, bár ez a szóhasználat inkább reklámfogás, mint műszaki koncepció. Később szót ejtünk arról, amelyik a szolgáltatások közül a legnépszerűbb lett, és ez az aszimmetrikus DSL, az **ADSL** (**Asymmetric Digital Subscriber Line – aszimmetrikus digitális előfizetői vonal**). A DSL és xDSL kifejezést is használni fogjuk.

A modemek ilyen mértékű lassúságának az az oka, hogy a telefonokat emberi beszéd átvitelére találták fel, és az egész rendszert gondosan erre a célra optimalizálták. Az adatok mindig is mostohagyerekek voltak. Azon a ponton, ahol az egyes előfizetői hurkok befutnak a helyi központba, a vezetékek egy szűrőn mennek keresztül, ami elnyomja a



2.33. ábra. *A DSL-en keresztül 3-as kategóriájú UTP-vel elérhető adatsebesség a távolság függvényében*

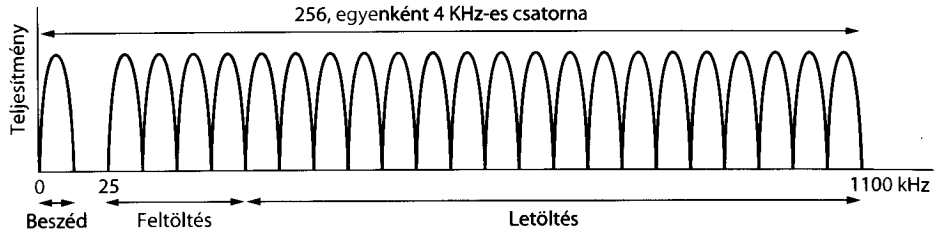
300 Hz alatti és a 3400 Hz feletti frekvenciákat. A vágás nem éles (a 300 Hz és a 3400 Hz a –3 dB-es pontok helye), így a sávszélességet általában 4000 Hz-nek adják meg, annak ellenére, hogy a –3 dB-es pontok távolsága csak 3100 Hz. Az adatoknak is ez a keskeny sáv áll rendelkezésére.

Az xDSL-t működtető trükk az, hogy amikor a felhasználó előfizet rá, akkor bejövő vonalát egy olyan, másfajta kapcsolóra kötik át, amelyen nincs rajta ez a szűrő, és így kihasználhatóvá teszik az előfizetői hurok teljes kapacitását. Ezután a korlátozó tényező már nem a szűrő által mesterségesen meghatározott 3100 Hz széles sáv, hanem az előfizetői hurok fizikai törvények által meghatározott sávszélessége.

Sajnos az előfizetői hurok kapacitása nagyon gyorsan csökken a helyi központtól való távolsággal, mivel a jel egyre nagyobb mértékben csillapodik a vezeték mentén. Ez a sodrott érpár vastagságától és általános minőségétől is függ. Az elérhető sávszélességet a távolság függvényében a 2.33. ábrán ábrázoltuk. Az ábra grafikonja azt feltételezi, hogy az összes többi tényező optimális (új vezetékek, szerény vastagságú kötegek stb.).

Az ábra következményei nagy fejtörést okoznak a telefontársaságoknak. Amikor kiválasztják, hogy mekkora sebességet ajánljanak, akkor egyben azt is meghatározzák, hogy a helyi központok mekkora sugarú környezetében tudják nyújtani a szolgáltatást. Ez azt jelenti, hogy amikor egy túl távoli ügyfél akar előfizetni a szolgáltatásra, akkor lehet, hogy azt mondják neki az ügyfélszolgálaton, hogy „Köszönjük szépen az érdeklődését, de 100 méterrel messzebb lakik a legközelebbi helyi központtól, mint ahol még megkaphatná a szolgáltatást. Közelebb tudna költözni?" Minél kisebbre választják meg a sebességet, annál nagyobb ez a sugár, és így több ügyfelet tudnak kiszolgálni. De minél kisebb a sebesség, annál kevésbé vonzó a szolgáltatás, és így kevesen lesznek, akik hajlandók fizetni érte. Ez az a pont, ahol az üzleti és a műszaki dolgok találkoznak.

Az összes xDSL-szolgáltatást bizonyos célok szem előtt tartásával tervezték. Először is, a szolgáltatásoknak működniük kell a már létező 3-as kategóriájú sodrott érpáros előfizetői hurkokon. Másodszor, a változások nem érinthetik az ügyfelek korábban vásárolt telefon- és faxkészülékeit. Harmadszor, sokkal gyorsabbnak kell lenniük 56 kb/s-nál. Negyedszer, folyamatos szolgáltatást kell nyújtaniuk rögzített havidíjjal, de percdíj nélkül.



2.34. ábra. *ADSL működése diszkrét többtónusú modulációval*

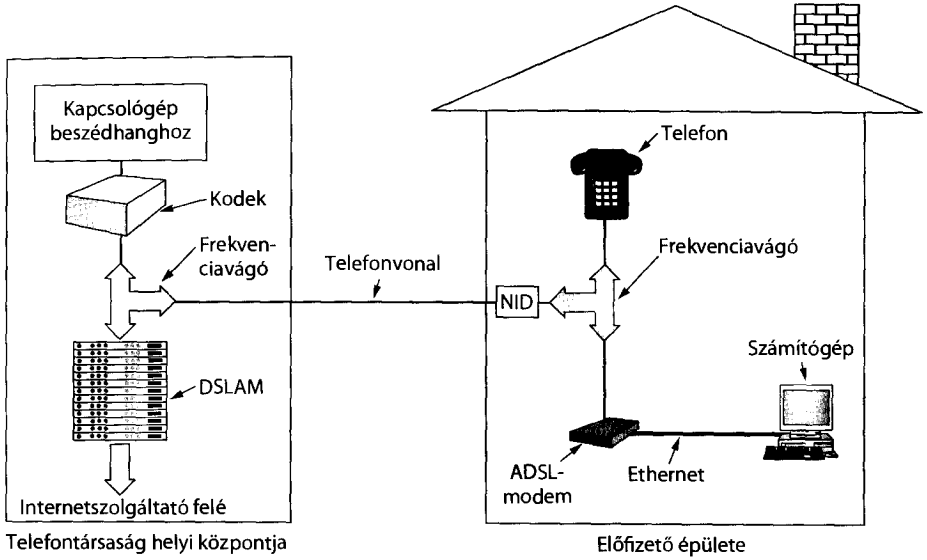
Ezen műszaki célok kielégítéséhez az előfizetői hurkon rendelkezésre álló körülbelül 1,1 MHz-es sávszélességet 256 független csatornára osztották fel, amelyek közül mindegyik 4312,5 Hz széles. Ezt a felosztást a 2.34. ábra mutatja. Az előző szakaszban tárgyalt OFDM-sémát használják arra, hogy adatokat küldjenek át ezeken a csatornákon keresztül, habár ADSL-környezetben gyakran hívják ezt **DMT**-nek (**Discrete MultiTone – diszkrét többtónusú**). A 0-s csatornát használják a **POTS-hoz** (**Plain Old Telephone Service – egyszerű régi telefonszolgáltatás**). Az 1–5. csatornát nem használják a hang és adatjelek egymással való interferenciájának megakadályozása érdekében. A fennmaradó 250 csatornából egyet a feltöltési, egyet pedig a letöltési forgalom vezérlésére tartanak fenn. A többi a felhasználói adatoké.

Elméletben az összes fennmaradó csatornát lehetne használni egyetlen duplex adatfolyamként, de a nem kívánt felharmonikusok, az áthallás és más okok miatt a gyakorlati rendszerek jóval az elméleti határ alatt maradnak. A szolgáltatótól függ a feltöltési és a letöltési forgalom által használt csatornák számának meghatározása. A két irány 50-50%-os keveréke műszakilag lehetséges, de a legtöbb szolgáltató a sávszélesség 80-90% körüli részét a letöltési forgalomnak tartja fenn, mivel a legtöbb felhasználó több adatot tölt le, mint amennyit fel. Ez a választás vezetett az „A" betűhöz az ADSL-ben. Gyakori az a megosztás, hogy 32 csatornát jelölnek ki a feltöltéshez, és a többi a letöltésé. A sávszélesség növelésének érdekében lehetséges a legfelső feltöltési csatornák közül néhány kétirányúsítása is, de ehhez az optimalizáláshoz külön visszhangtörlő áramkörök beépítése szükséges.

A nemzetközi ADSL-szabványt (**G.dmt** néven) 1999-ben fogadták el. Ez akár 8 Mb/s-os letöltési és 1 Mb/s-os feltöltési sebességet is megenged. Ezt felváltotta egy második generáció 2002-ben, amelyet ADSL2-nek hívnak. Ennek számos továbbfejlesztése akár 12 Mb/s-os letöltési, és 1 Mb/s-os feltöltési sebességet is lehetővé tett. Jelenleg az ADSL2+ van érvényben, amely megduplázza a letöltési sebességet 24 Mb/s-ra azáltal, hogy kétszeresére növeli a sávszélességet, és így 2,2 MHz-et használ a sodrott érpáron.

Az itt megadott számok azonban a legjobb lehetséges sebességet mutatják jó vonalak esetén, amelyek közel (1-2 km-re) vannak a helyi központhoz. Néhány vonal támogatja ezt a sebességet, és néhány szolgáltató kínálja ezt a sávszélességet. Jellemzően a szolgáltatók 1 Mb/s letöltési és 256 kb/s feltöltési (alapszolgáltatás), 4 Mb/s-os letöltési és 1 Mb/s-os feltöltési (tökéletesített szolgáltatás), illetve 8 Mb/s-os letöltési és 4 Mb/s-os feltöltési (kiemelt szolgáltatás) sebességet kínálnak.

Minden csatornán belül a QAM-moduláció kerül felhasználásra nagyjából 4000 szimbólum/s sebességgel. Minden csatornán állandóan figyelik a vonal minőségét, és hoz-



2.35. ábra. *Az ADSL-eszközök egy tipikus elrendezése*

záigazítják az adatsebességet nagyobb vagy kisebb csillagképre-váltással, mint ahogy az a 2.23. ábrán látható. Így a különböző csatornák adatsebessége eltérő lehet. Akár 15 bit/szimbólum is küldhető a csatornán nagy jel/zaj viszony (SNR) esetén, illetve 2, 1 vagy 0 bit/szimbólum kis jel/zaj viszony mellett a szabványtól függően.

A 2.35. ábrán egy tipikus ADSL-elrendezés látható. Ebben a megoldásban a telefontársaság technikusának egy **NID**-t (**Network Interface Device – hálózati interfészeszköz**) kell telepítenie az előfizető épületébe. Ez a kis műanyag doboz jelzi azt a pontot, ahol a telefontársaság tulajdona véget ér, és ahol a felhasználó tulajdona kezdődik. A NID-hez közel (vagy néha abba beleépítve) van egy **frekvenciavágó** (**splitter**), ami egy analóg szűrő, amely a beszédjel által használt 0–4000 Hz-es tartományt választja le az adatokról. A beszédjelet a hagyományos telefonokhoz és faxokhoz irányítják, az adatjeleket pedig egy ADSL-modemhez, amely digitális jelfeldolgozást használ az OFDM megvalósításához. Mivel a legtöbb ADSL-modem külső egység, a számítógéppel nagy sebességű összeköttetésének kell lennie. Ezt általában Ethernettel, USB-kábellel vagy 802.11 segítésével történik.

A vezeték másik végén, a helyi központ oldalán egy hasonló frekvenciavágót helyeznek el. Itt a jelből kiszűrik a beszédjelet, és a hagyományos kapcsolóközponthoz irányítják. A jel 26 kHz feletti részét egy újfajta eszközhöz irányítják, amelyet **DSLAM**-nek (**DSL Access Multiplexer – DSL hozzáférési multiplexer**) neveztek el. Ez az eszköz egy ugyanolyan digitális jelfeldolgozót tartalmaz, mint az ADSL-modem. Miután a digitális jelből visszaállították a bitfolyamot, csomagokra bontják, amelyeket ezután az internetszolgáltató felé továbbítanak.

Az ADSL- és a beszédrendszer ilyen teljes szétválasztása viszonylag könnyűvé teszi az ADSL telepítését a telefontársaságok számára. Csupán arra van szükség, hogy vegyenek egy DSLAM-et és egy frekvenciavágót (splittert), azután pedig a frekvenciavágóra rákösök az ADSL-előfizetőket. Más nagy sávszélességű rendszerek (például az ISDN) telepítése a már meglevő kapcsolóeszközök sokkal nagyobb mértékű megváltoztatásával jár.

A 2.35. ábra elrendezésének egyik hátránya, hogy egy NID-et és egy frekvenciavágót kell elhelyezni a felhasználó épületében. Ezeknek a telepítését csak a telefontársaság egyik technikusa képes elvégezni, és ez jelentős többletköltséget jelent (mivel a technikust el kell juttatni az előfizető házához). Ezért egy másik, frekvenciavágó nélküli rendszert is szabványosítottak (a nem hivatalos neve G.lite). Ez megegyezik a 2.35. ábrán látható elrendezéssel, de a frekvenciavágó nélkül. A rendelkezésre álló telefonvonalat ez a megoldás minden változtatás nélkül használja. Az egyetlen különbség az, hogy egy mikroszűrőt tettek minden telefondugóba, a telefon vagy az ADSL-modem, illetve a vezeték közé. A telefon mikroszűrője egy olyan aluláteresztő-szűrő, amely a 3400 Hz feletti frekvenciákat szűri ki, az ADSL-modem mikroszűrője pedig egy olyan felüláteresztő-szűrő, amely a 26 kHz alatti frekvenciákat szűri ki. Ez a rendszer azonban nem annyira megbízható, mint a frekvenciavágóval felszerelt, így a G.lite-ot csak 1,5 Mb/s-os sebességig lehet használni (a frekvenciavágós ADSL 8 Mb/s-ával szemben). Az ADSL-lel kapcsolatos további információt Starr [2003] munkája tartalmaz.

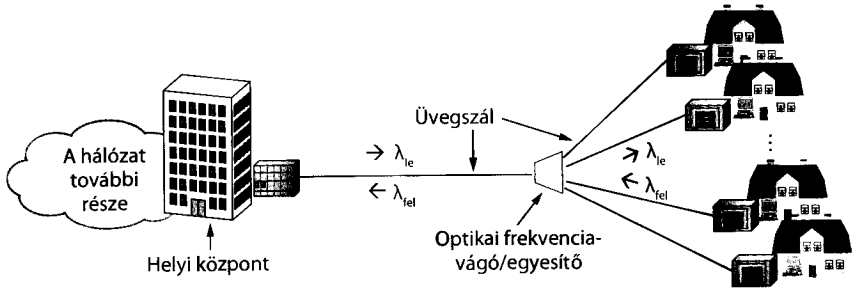
Üvegszál a lakásig

A telepített réz előfizetői hurkok korlátozzák az ADSL- és a telefonmodemek teljesítőképességét. Ahhoz, hogy ezek gyorsabb és jobb hálózati szolgáltatásokat nyújtsanak, a távközlési társaságok, amikor csak lehet, üvegszál telepítésével frissítik az előfizetői hurkokat egészen a lakásokig vagy irodákig vezető teljes útvonalon. Ennek eredményét **FttH**-nak (**Fiber to the Home – üvegszál a lakásig**) nevezik. Az FttH-technika már rendelkezésre áll egy ideje, de a telepítése csak 2005-ben kezdődött, amikor a DSL-t vagy kábelt használó előfizetők körében megjelent a nagy sebességű internet iránti igény, mert filmeket akartak letölteni. Az USA-ban lévő házak körülbelül 4%-a csatlakozik az FttH-hoz akár 100 Mb/s-os internet-hozzáférési sebességgel.

Az „FttX"-nek számos változata létezik (ahol *X* a pincét, az elosztót vagy a környéket jelenti). Ezek azt mutatják, hogy az üvegszál telepítése a ház közelébe ért. Ebben az esetben a réz (sodrott érpár vagy koaxiális kábel) már elég nagy sebességet biztosít az utolsó néhány méteren. Az, hogy az üvegszálat mennyire viszik közel a házakhoz, gazdasági döntés, a várt bevétel és a költség közötti egyensúly mérlegelésével. Minden esetre az eredmény az, hogy az üvegszál átlépte a hagyományos „utolsó kilométer" határt. Az FttH-val részletesebben is foglalkozunk.

Ahogy korábban a rézvezetékek, az üvegszálas előfizetői hurok is passzív. Ez azt jelenti, hogy nincs szükség elektromos berendezésre a jelek erősítéséhez vagy egyéb módon történő feldolgozásához. Az üvegszál egyszerűen jelet szállít a lakás és a helyi központ között. Ez csökkenti a költséget és javítja a megbízhatóságot.

Általában a házaktól jövő üvegszálakat egyesítik, így körülbelül 100 házanként csak egy üvegszál éri el a helyi központot. Letöltési irányban az optikai frekvenciavágó felosztja a helyi központtól érkező jelet, így az eléri az összes házat. A biztonság érdekében titkosítás szükséges, ha csak egy háznak szabad dekódolnia a jelet. Feltöltési irányban az optikai egyesítő összefésüli a házaktól érkező jeleket egyetlen jellé, amelyet a helyi központ megkap.



2.36. ábra. *Passzív optikai hálózat FttH-hoz*

Ezt az architektúrát **PON**-nak (**Passive Optical Network – passzív optikai hálózat**) hívják, és a 2.36. ábra mutatja be. Általános, hogy az összes háznál egyetlen hullámhosszt osztanak meg a letöltéshez, illetve egy másikat a feltöltéshez. A hatalmas sávszélességű és kis csillapítású üvegszál még frekvenciavágás esetén is azt jelenti, hogy a PON nagy sebességet tud biztosítani a felhasználók számára, akár 20 km-es távolságig. A tényleges adatsebesség és más részletek a PON típusától függenek. Általánosan kétfélét használnak: a **GPON**-t (**Gigabit-capable PON – gigabitre felkészített PON**) a távközlésből származik, így ezt az ITU-szabvány adja meg, illetve az **EPON-t** (**Ethernet PON – Ethernet-alapú PON**) – ez inkább összhangban van a hálózat világával, így ezt az IEEE-szabvány adja meg. Mindkettő gigabites sebességgel működik és különböző szolgáltatások forgalmát szállítja, mint amilyen például az internet-, video- és beszédforgalom. A GPON például 2,4 Gb/s letöltés, illetve 1,2 vagy 2,4 Gp/s feltöltési sebességet biztosít.

Néhány protokoll szükséges az üvegszál kapacitásának megosztásához a különböző házak közötti helyi központban. A letöltési irány egyszerű. A helyi központ mindegyik háznak üzenetet tud küldeni, tetszőleges sorrendben. A feltöltési irányban azonban a különböző házaktól jövő üzenetek nem küldhetők egyszerre, mert akkor a különböző jelek összeütköznek. A házak nem hallják egymás adását, ezért nem tudnak figyelni átvitel előtt. A megoldás az, hogy a házban lévő eszköz időszeletet kér, majd a helyi központban lévő berendezés időszeletet biztosít számára. Ahhoz, hogy ez működjön, rendelkezésre áll egy sorrendező folyamat a házak adási idejének beállításához, így a helyi központban vett összes jel szinkronizált lesz. A kialakítás a kábelmodemekéhez hasonló, amelyről a fejezet későbbi részében lesz szó. A PON-ok jövőjével kapcsolatos további információt Grobe és Elbers [2008] munkája tartalmaz.

2.6.4. Trönkök és multiplexelés

A telefonhálózatban lévő trönkök nem csak sokkal gyorsabbak, mint az előfizetői hurkok, hanem két további szempontból is különböznek. A telefonhálózat központi része digitális információt visz át, nem analóg információt, azaz bitet, és nem hangot. Ez szükségessé teszi a helyi központban a digitális formátumra való átalakítást a nagy távolságú trönkökön való átvitel érdekében. A trönkök egyidejűleg több ezer, vagy akár több millió hívást visznek át. Ez a megosztás a gazdaságosság szempontjából fontos, mivel

két kapcsolóközpont között a nagy sávszélességű trönk és a kis sávszélességű trönk telepítésének és fenntartásának költsége lényegében megegyezik. Ez a TDM- és FDM-multiplexelés különböző változataival érhető el

Alább részletesen megvizsgáljuk a hangjelek digitalizálásának módját, mivel így azok átvihetők a telefonhálózaton. Ezután megnézzük, hogy a TDM hogyan használható bitek trönkökön történő átvitelére, az üvegszálas átvitelhez használt TDM-rendszert is beleértve (SONET). Ezután az üvegszálas átvitelre alkalmazott FDM következik, amelyet hullámhosszosztásos multiplexelésnek hívnak.

Hangjelek digitalizálása

A telefonhálózat telepítésének korai fázisában a hálózat magja a hanghívásokat analóg információként kezelte. Sok éven keresztül FDM-technikákat használtak a 4000 Hz-es hangcsatornák (3100 Hz-es felhasználói sávszélesség és védősávok) egyre nagyobb egységekbe történő multiplexeléséhez. Például a 60 kHz és 108 kHz közötti frekvenciasávba eső 12 csatornát **csoportnak** (**group**) nevezik. Öt ilyen csoport (tehát 60 hangcsatorna) egyesítését **főcsoportnak** (**supergroup**) hívják, és így tovább. Ezeket az FDM-módszereket még mindig használják a rézvezetékeken és a mikrohullámú csatornákon. Az FDM azonban analóg áramköröket igényel, és a számítógép tevékenységével sem összeegyeztethető. Ezzel szemben a TDM teljes egészében kezelhető digitális elektronikával, ezért sokkal jobban elterjedt az elmúlt években. Mivel a TDM csak digitális adatok továbbításához használható és az előfizetői hurkok analóg jeleket állítanak elő, szükség van egy analóg/digitális átalakításra a helyi központban, ahová az egyes előfizetői hurkok beérkeznek, hogy nyalábolásra kerüljenek a kimenő trönkökre.

Az analóg jeleket a helyi központban egy **kodek** (kódoló-dekódoló) nevű eszköz digitalizálja. A kodek 8000 mintát vesz másodpercenként (125 μs/minta), mivel Nyquist tételének értelmében ennyi kell ahhoz, hogy minden információt kinyerjünk egy 4 kHz-es sávszélességű telefoncsatornából. Kisebb mintavételezési sebesség mellett információt veszítenénk, nagyobb sebességgel pedig nem tudnánk ennél több információt kinyerni a csatornából. A jel amplitúdójából vett minden mintát 8 bites számmá kvantálja.

Ezt a megoldást **PCM-nek** (**Pulse Code Modulation – impulzuskód-moduláció**) nevezték el. A PCM alkotja a modern telefonhálózatok lelkét. Ennek következtében a telefonrendszerben előforduló szinte minden időköz 125 μs-nak a többszöröse. A beszédátvitelre szánt telefonhívások normál tömörítetlen adatsebessége 8 bit 125 μs-os időközönként, vagyis 64 kb/s.

A vonal másik végén az analóg jel visszaállítására kerül sor a kvantált mintákat időben visszajátszva (és kisimítva). Ez nem lesz teljesen ugyanaz, mint az eredeti analóg jel volt, még abban az esetben sem, ha a Nyquist-törvény szerint történt is a mintavételezés, mivel a minták kvantálva vannak. A kvantálásból származó hiba csökkentése érdekében a kvantálási szinteket egyenlőtlenül osztják fel. Logaritmikus skálát használnak, amely több bitet biztosít a kisebb amplitúdójú, mint a nagy amplitúdójú jeleknek. Ily módon a hiba a jel amplitúdójával arányos.

Kétféle kvantálást használnak széles körben: **μs-törvényt** Észak-Amerikában és Japánban, míg az **A-törvényt** Európában és a világ többi részén. Mindkét változatot az ITU

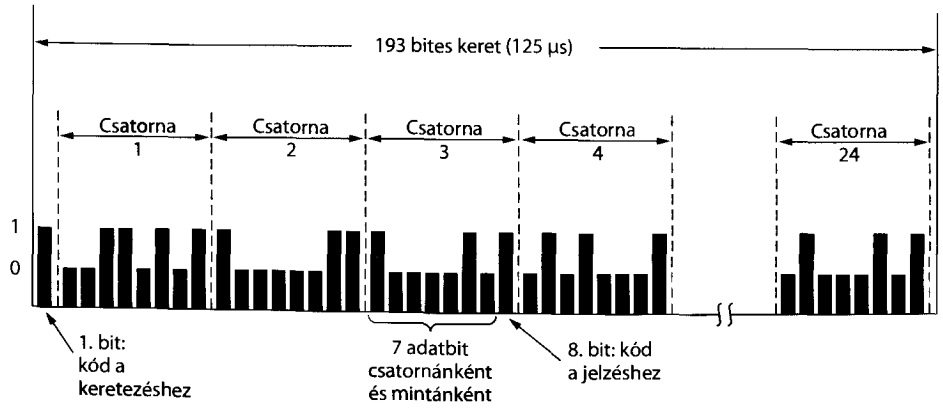
G.711 szabványa határozza meg. A folyamat egyenértékű megközelítéseként képzeljük el, hogy a jel dinamikus tartományát (vagy a legnagyobb és legkisebb lehetséges értékek közötti arányt) előbb tömörítik a(z) (egyenletes) kvantálás előtt, majd kiterjesztik az analóg jel újbóli létrehozásakor. Emiatt ezt **kompandálásnak** (összenyomás-kiterjesztés) hívják. A mintákat digitalizálás után is lehet tömöríteni, így ezek 64 kb/s-nál kisebb sebességet igényelnek. Ezt a témakört azonban meghagyjuk akkorra, amikor majd az olyan audioalkalmazásokat vizsgáljuk, mint például az IP-n keresztül történő hangátvitel.

Időosztásos multiplexelés

A TDM PCM-en alapul, és arra használják, hogy több hanghívást vigyen át a trönkökön, minden hívásból 125 μs-onként vett minta küldésével. Amikor a digitális átvitel már kezdett megvalósítható megoldásnak látszani, az ITU (akkor még CCITT) nem tudott megegyezésre jutni egy nemzetközi PCM-szabványról. Ennek következtében manapság sok, egymással inkompatibilis megoldást használnak a világ különböző országaiban.

Az Észak-Amerikában és Japánban használatos megoldás, a **T1**-vivő a 2.37. ábrán látható. (Műszakilag helyesen a formátumot DS1-nek és a vivőt T1-nek hívják, de az iparban széleskörűen elterjedt hagyományt követve ezt a finom megkülönböztetést itt sem tesszük meg.) A T1-vivő 24 beszédcsatornát továbbít egybenyalábolva. Mind a 24 csatorna 8-8 bitet tud a kimeneti folyamba beszúrni.

A keret 24 × 8 = 192 bitből, plusz egy vezérlési célra használt bitből áll, tehát 193 bitet kell továbbítani 125 μs-onként. Ez tulajdonképpen végeredményben 1,544 Mb/s-os adatsebességet jelent, amelyből 8 kb/s-ot jelzésre használnak. A 193. bitet keretszinkronizálásra és jelzésre használják. Az egyik változatban a 193. bitet 24 keretből álló csoportban használják, amelyet **kiterjesztett szuperkeretnek (extended superframe)** hívnak. 6 bit (a 4., 8., 12., 16., 20. és 24. pozíción) az alábbi váltakozó bitmintát kapja: 001011... A vevő normális körülmények között folyamatosan ellenőrzi ezt a bitmintát, és így győződik meg arról, hogy nem esett-e ki a szinkronból. Ha kiesik a szinkronból, akkor ezt a mintát keresi és hitelesíti a hibaellenőrző kódot az újraszinkronizálódás



2.37. ábra. *T1 vivő (1,544 Mb/s)*

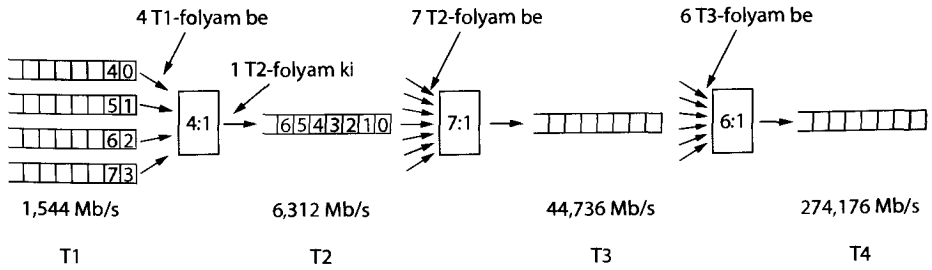
érdekében. A fennmaradó 12 bit vezérlőinformációként kerül felhasználásra a hálózat működtetéséhez és fenntartásához, mint amilyen például a távoli végről jövő teljesítményjelentés.

A T1 formátumnak számos változata van. A korábbi változatok sávon **belül (in-band)** küldtek jelzési információt, ami azt jelenti, hogy ez az adattal egy csatornában kerül elküldésre, néhány adatbit felhasználásával. Ez a kialakítás a **csatornához kapcsolódó jelzés (channel associated signaling)** egyik formája, mivel minden csatorna saját jelzési alcsatornával rendelkezik. Egyik elrendezésben minden csatornán a 8-bites minta legkisebb helyi értékű bitje felhasználásra kerül, minden hatodik keretben. Ennek beszédes neve van: **elrabolt-bit jelzés (robbed-bit signaling)**. Ennek alapötlete az, hogy néhány „elrabolt" bit nem számít hanghívás esetén. Senki nem fogja hallani a különbséget.

Adatok esetén azonban más a helyzet. Rossz bitek kézbesítése finoman szólva sem hasznos. Ha a T1 régebbi változatai adatokat vittek át, akkor a 8-ból csak 7 bit volt használható, vagyis az adatsebesség 56 kb/s volt mind a 24 csatornán. Ehelyett a T1 újabb változatai üres csatornákat biztosítanak, amelyeken az összes bit adatküldésre használható. Az üres csatornák olyan csatornák, amelyeket a T1-vonalat bérlő vállalatok használni akarnak, amikor adatokat küldenek át a telefonhálózaton hangminták helyett. A hanghívások jelzése **sávon kívül (out-of-band)** történik, ami azt jelenti, hogy különálló csatorna kerül felhasználásra, nem az adatcsatorna. A jelzés gyakran **közös csatornás jelzéssel (common-channel signaling)** történik, amelyben osztott jelzési csatorna található. Erre a célra a 24 csatorna közül az egyik használható.

Észak-Amerika és Japán kivételével a világon szinte mindenütt a 2,048 Mb/s-os **E1**-vivőt használják a T1 helyett. Az E1-vivő az alapnak tekintett 125 μs-os keretben 32 darab 8 bites mintát továbbít. Ebből 30 csatorna adatátvitelre, legfeljebb 2 csatorna pedig jelzésre szolgál. Minden négy keretből álló csoport 64 jelzőbitet tartalmaz, amelyeknek az egyik fele jelzésre szolgál (vajon a jelzés a csatornával kapcsolatos vagy általános), a másik felét keretszinkronizálásra használatos, illetve országonként kívánság szerint szabadon felhasználható.

Az időosztásos multiplexelés azt is lehetővé teszi, hogy több T1-vivőt multiplexeljünk magasabb rendű vivőkre. A 2.38. ábra azt mutatja, hogy ezt hogyan tehetjük meg. A bal oldalon négy T1-csatorna látható, amelyeket egy T2-csatornára multiplexelnek. A T1-keretekben elhelyezett 24 beszédcsatorna multiplexelését bájtonként végzik, ezzel szemben a T2-es és annál magasabb szinteken a multiplexelés bitenként történik. A négy darab, egyenként 1,544 Mb/s-os T1 folyam összesen 6,176 Mb/s-ot eredményez, de a T2



2.38. ábra. *T1-vivők multiplexelése nagyobb vivőkre*

a gyakorlatban mégis 6,312 Mb/s-os. A többletbiteket a keretezéshez és a vivő csúszásainak kivédésére használják. A T1-et és a T3-at széles felhasználói kör használja, míg a T2 és a T4 kizárólag a telefonrendszeren belül használatosak, ezért nem túl ismertek.

A következő szinten hét T2-folyam bitenkénti nyalábolásával jön létre egy T3-folyam. Ezután hat T3-folyamot egyesítenek egy T4-folyammá. Minden lépésben néhány többletbitet építenek a folyamba a keretezéshez és a hibajavításhoz arra az esetre, ha a küldő és a fogadó közötti szinkronizáció elveszne.

Ahogy kicsi az egyetértés az Egyesült Államok és a világ többi országa között az alapvivőt illetően, úgy abban sincs egyetértés, hogy a vivőket hogyan multiplexeljék nagyobb sávszélességű vivőkre. Mivel az Egyesült Államokban használt multiplexelési mód – azaz először 4, majd 7, végül 6 csatorna összefogása – nem nyerte el a többi ország tetszését, ezért az ITU ezt úgy szabványosította, hogy minden szinten 4 csatornát kell összefogni. Ezenkívül a keretezést és a hibajavítást is másképpen definiálta. A 32, 128, 512, 2048 és 8192 csatornából álló ITU-hierarchia megfelelő sebességei a következők: 2,048; 8,848; 34,304; 139,264 és 565,148 Mb/s.

SONET/SDH

Az üvegszálas rendszerek megjelenésekor minden telefontársaságnak saját optikai TDM-rendszere volt. Miután 1984-ben az AT&T feldarabolódott, a helyi telefontársaságoknak több olyan nagy távolságú szolgáltatóhoz kellett kapcsolódniuk, amelyek mind különböző üvegszálas TDM-rendszert használtak. Így nyilvánvalóan elkerülhetetlenné vált a szabványosítás. 1985-ben a Bellcore, az RBOC kutatási részlege elkezdett dolgozni egy szabványon, amit **szinkron optikai hálózatnak (Synchronous Optical NETwork, SONET)** neveztek el.

Később az ITU is csatlakozott hozzájuk, és 1989-ben megszületett a SONET-szabvány és ezzel egy időben egy sor ITU-ajánlás (G.707, G.708 és G.709). Az ITU-ajánlásokat SDH-nak **(Synchronous Digital Hierarchy – szinkron digitális hierarchia)** nevezték el. Az SDH csak kicsit különbözik a SONET-től. Egyelőre úgy tűnik, hogy az Egyesült Államokban és sok más országban a nagy távolságú telefonhálózatok fizikai rétege olyan trönkökből áll, amely a SONET-et használja. Minderről bővebben Bellamy [2000], Goralski [2002] és Shapard [2001] műveiben olvashatunk.

A SONET-nek négy fő célja volt. Az első és legfontosabb, hogy a SONET segítségével lehetőség nyíljon különböző vivők együttműködésére. Ennek a célnak az eléréséhez egy olyan közös jelzésrendszer kialakítására volt szükség, amely a hullámhosszakat, az időzítéseket, a keretek szerkezetét stb. szabványosította.

Másodszor, szükség volt egy olyan eszközre, amely egységessé tette az Egyesült Államok, Európa és Japán digitális rendszereit, ugyanis mindhárom rendszer 64 kb/s-os PCM-csatornákon alapult, azonban mindegyik másképpen (egymással inkompatibilis módon) kombinálta ezeket a csatornákat.

Harmadrészt, a SONET-nek lehetővé kellett tennie több digitális csatorna multiplexelését. Amikor a SONET-et kidolgozták, az Egyesült Államokban a legnagyobb sebességű vivő a 44,736 Mb/s-os T3 volt. A T4-vivő ugyan elméletileg már létezett, azonban még nem használták. A T4-vivő fölött pedig még definiálva sem volt további sebesség. A SONET egyik küldetése éppen az volt, hogy folytassa a hierarchiát a Gb/s-os, továbbá az a fölötti tartományokba. A SONET-nél kisebb frekvenciájú vivők SONET-csatornákba történő multiplexelésére is egységes eljárást kellett kidolgozni.

Negyedrészt, a SONET-nek üzemeltetési, adminisztrációs és karbantartási feladatokat is el kellett látnia. A korábbi rendszerek ugyanis nemigen jeleskedtek ebben.

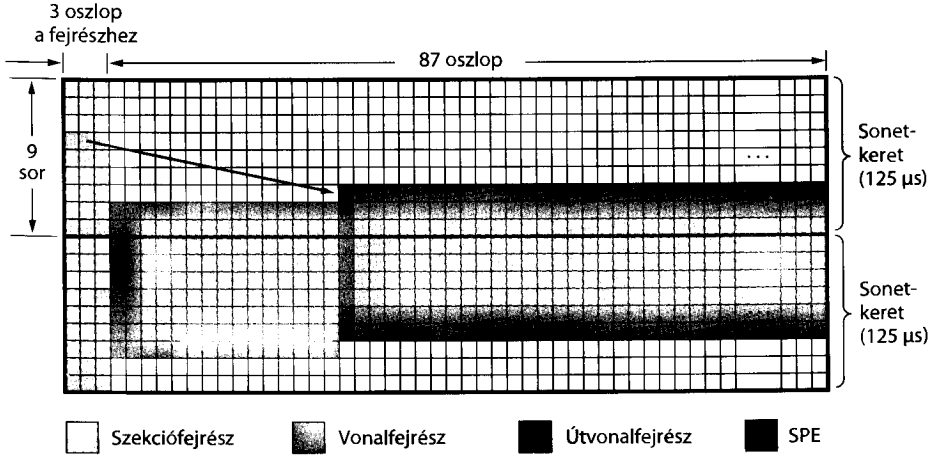
Eredetileg a SONET-et egy hagyományos TDM-rendszernek szánták, amiben az üvegszál teljes sávszélességét egy csatornának tekintették, és ez a csatorna időszeleteket biztosított a különböző alcsatornáknak. Mint ilyen, a SONET szinkronrendszer. Minden vevő és adó egy közös órához van kötve. Működését egy olyan mester órajel vezérli, amelynek pontossága kb. 10^{-9} s. A bitek mesterórajel segítségével a SONET-vonalakon rendkívül pontos időközönként kerülnek továbbításra.

A SONET-keret alapja egy 810 bájtos blokk, ami 125 μs-onként kerül ki az átviteli vonalakra. Mivel a SONET szinkronrendszer, ezért a kereteket attól függetlenül elküldi, hogy van-e bennük hasznos adat vagy nincs. A másodpercenként elküldött 8000 keret pontosan illeszkedik a digitális távbeszélőrendszerekben használt PCM-csatornák mintavételi frekvenciájához.

A 810 bájtos SONET-keret a legjobban egy olyan téglalap alakú bájtmezővel írható le, amelynek 90 oszlopa és 9 sora van. Ez azt jelenti, hogy $8 \times 810 = 6480$ bit továbbítódik másodpercenként 8000-szer, azaz a teljes adatsebesség 51,84 Mb/s. Ez a sebesség a SONET alapcsatornája, az **STS–1 (Synchronous Transport Signal–1 – 1-es számú szinkron szállítójel)**. Az összes SONET-trönk sebessége az STS-1 sebesség többszöröse.

Az első három oszlop minden keretben a rendszermenedzsment-információ számára van fenntartva, ahogy ez a 2.39. ábrán is látható. Ebből az első három sor a szekció fejrészét tartalmazza, a következő hat sor pedig a vonalét. A szekció fejrésze mindig a szekció adatai előtt generálódik, és a szekció végén kerül ellenőrzésre, míg a vonal fejrésze mindig a vonal elején képződik, és a vonal végén kerül ellenőrzésre.

Egy SONET-küldő 810 bájtos kereteket küld közvetlenül egymás után, szünetek nélkül. Akkor is küldi a kereteket, amikor nincs átviendő adat (ebben az esetben értéktelen



2.39. ábra. *Két egymást követő SONET-keret*

bitekkel tölti ki az adatbitek helyét). A vevő nézőpontjából mindez egy folytonos bitfolyamnak tűnik, így jogosan merül fel az a kérdés, hogy honnan tudja a vevő, hogy hol kezdődnek az egyes keretek? A válasz az, hogy minden keret első két bájtja egy rögzített mintát tartalmaz, amelyet a vevő folyamatosan keres a bitfolyamban. Ha sok egymás utáni keretben ugyanazon a helyen találja meg ezt a mintát, akkor azt feltételezi, hogy szinkronban van az adóval. Elméletileg a felhasználó is rendszeresen beilleszthetné ezt a mintát az adatrészbe, de ezt a gyakorlatban nem teheti meg, többek között azért, mert egy keretbe több felhasználó adatai vannak multiplexelve.

A maradék 87 oszlop $87 \times 9 \times 8 \times 8000 = 50{,}112$ Mb/s felhasználói adatot tartalmazhat. Persze az adatmező, amelyet **SPE**-nek (**Synchronous Payload Envelope – szinkron-adatokat tartalmazó boríték**) hívnak, nem mindig az első sor negyedik oszlopában kezdődik. Az SPE bárhol kezdődhet a kereten belül. Az első bájtra mutató pointert a vonal fejrészének első sora tartalmazza. Az SPE első oszlopa az elérési út fejrésze (tehát a végpontok közötti elérési út alréteg protokolljának a fejrésze).

Még rugalmasabbá teszi a rendszert az a lehetőség, hogy az SPE a 2.39. ábrán is látható módon a SONET-kereten belül bárhol kezdődhet és akár a következő keretbe is átlóghat. Ha például hasznos adatok érkeznek a forráshoz egy értéktelen SONET-keret készítése közben, akkor az adatot az éppen készülő keretbe is beilleszthetti, nem kell megvárnia vele a következő keret elejét.

A SONET/SDH multiplexelési hierarchiáját a 2.40. ábrán tüntettük fel. A sebességeket az STS-1-től az STS-768-ig definiálták, nagyjából a T3-vonaltól a 40 Gb/s-ig. A jövőben még nagyobb sebességeket is definiálnak majd a jövőben, a 160 Gb/s-os OC-3072 lesz a következő vonal, ha az technikailag megvalósíthatóvá válik. Az STS-*n*-nek megfelelő üvegszálas vivőt (optical carrier) OC-*n*-nek hívják. Ezek bitről bitre megegyeznek egy olyan bitátrendezés kivételével, amely a szikronizáláshoz szükséges. Az SDH elnevezései különböznek ettől és az OC-3-tól kezdődnek, mert az ITU-alapú rendszerekben nincsen 51,84 Mb/s-hoz közeli sebesség. Bemutattuk a szokásos sebességeket az OC-3-tól kezdve, és a 4 többszörösével növelve. A bruttó adatsebességbe az összes többletbitet is beleszámítjuk. Az SPE adatsebesség nem tartalmazza a vonal- és szekciófejrész-biteket. A felhasználói adatsebességbe semmilyen többletbit nem számít bele, csak a 86 adatoszlopot kell beleszámolni.

SONET		SDH	Adatsebesség (Mb/s)		
Villamos	**Optikai**	**Optikai**	**Bruttó**	**SPE**	**Felhasználói**
STS-1	OC-1		51,84	50,112	49,536
STS-3	OC-3	STM-1	155,52	150,336	148,608
STS-12	OC-12	STM-4	622,08	601,344	594,432
STS-48	OC-48	STM-16	2488,32	2405,376	2377,728
STS-192	OC-192	STM-64	9953,28	9621,504	9510,912
STS-768	OC-768	STM-256	39813,12	38486,016	38043,648

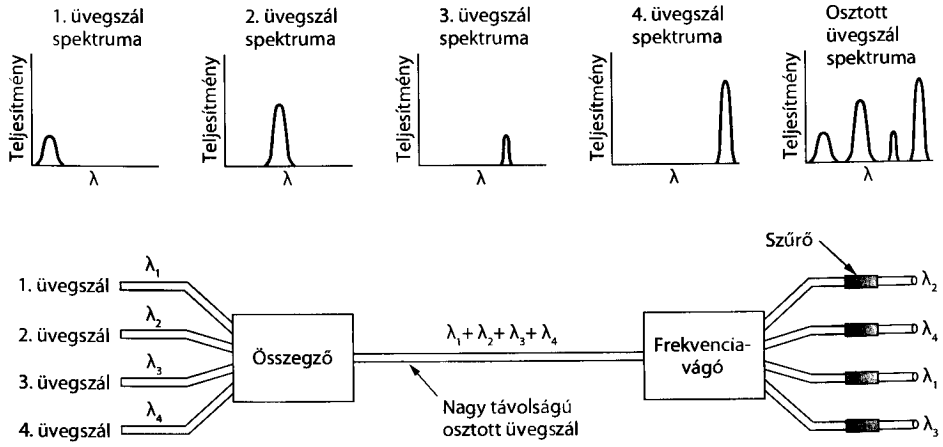
2.40. ábra. *A SONET- és az SDH-adatsebességek*

Egyébként, amikor az egyik vivőt, mondjuk az OC-3-at nem multiplexelik, és mégis adatokat továbbítanak ilyen adatsebességgel egyetlen forrásgépből, akkor a jelölést egy *c* betűvel (az angol *concatenated* szónak megfelelően) egészítik ki. Tehát a 155,52 Mb/s-os OC-3 vivő három különböző OC-1 vivőből áll, míg az OC-3c egy olyan vivőt jelöl, amely egyetlen forrásgép adatait továbbítja 155,52 Mb/s-os sebességgel. Az OC-3c adatfolyamban összefogott OC-1 adatfolyamok oszloponként átlapolják egymást, tehát először az első adatfolyam első oszlopa, majd a második adatfolyam első oszlopa, végül a harmadik adatfolyam első oszlopa, ezt követően pedig az első adatfolyam második oszlopa stb. kerül továbbításra, ami egy 270 oszlop széles és 9 sor magas keretet jelent.

Hullámhosszosztásos multiplexelés

A frekvenciaosztásos multiplexelés egyik fajtája, valamint TDM is alkalmazásra kerül az üvegszálas csatornák rettenetesen nagy sávszélességének kihasználása érdekében. Ennek neve **WDM (Wavelength Division Multiplexing – hullámhosszosztásos multiplexelés)**. Az üvegszálakon alkalmazott WDM alapelveit a 2.41. ábrán tüntettük fel. Az ábrán négy olyan szál találkozik egy optikai összegzőben (optical combiner), amelyek energiája más és más hullámhosszokon van. A hullámhosszösszegző a négy nyalábot egyetlen üvegszálra nyalábolja össze a távoli célállomás felé való továbbításhoz. A túloldalon a nyalábot annyi üvegszálra osztják ismét szét, amennyi a bemeneti oldalon is van. Minden kimeneti szál magjának egy rövid darabja különleges kialakítású, amely egyetlen hullámhossz kivételével mindent kiszűr. Az így keletkező jeleket ezután a címzett állomáshoz lehet irányítani, vagy más kombinációban újra össze lehet nyalábolni további multiplexelt szállításra.

Valójában ebben semmi újdonság nincs. Ez a működési mód tulajdonképpen a frekvenciaosztátos multiplexelés nagyon nagy frekvencián, a WDM kifejezés az üvegszálas csatornák leírásához tartozik a hullámhossz vagy „szín" alapján, és nem a frekvencia



2.41. ábra. *Hullámhosszosztásos multiplexelés*

alapján. Miután mindegyik csatorna saját frekvenciatartománnyal (vagyis hullámhosszal) rendelkezik, és ezek a tartományok nem lapolódnak át, a csatornákat egyetlen nagy távolságú üvegszálra lehet multiplexelni. A villamos FDM-től mindössze annyiban különbözik, hogy az optikai rácsnak köszönhetően ez az optikai rendszer teljesen passzív, ezért rendkívül megbízható.

WDM leginkább annak köszönheti a népszerűségét, hogy egyetlen csatornán lévő energia általában csak néhány gigahertz széles sávban van, mivel jelenleg lehetetlen ennél gyorsabb jelátalakítást megvalósítani az elektromos és a fényvezető közegek között. Ha sok, különböző hullámhosszon működő csatornát használunk párhuzamosan, akkor az eredő sávszélesség a csatornák számával egyenes arányban nő. Mivel az üvegszálakon a sávok körülbelül 25 000 GHz-esek (lásd 2.7. ábra), elméletileg 2500, egyenként 10 Gb/s-os csatorna fér el egy szálon még 1 bit/Hz-es sűrűségnél is (és ennél nagyobb sebességek is lehetségesek).

A WDM-megoldások a számítástechnikát is megszégyenítő sebességgel fejlődtek. A WDM-et 1990 körül találták fel. Az első kereskedelmi rendszerekben 8 csatorna volt, amelyek sebessége egyenként 2,5 Gb/s volt. 1998-ra olyan rendszerek kerültek a piacra, amelyek 40 darab 2,5 Gb/s-os csatornát használtak. 2006-ra pedig már olyan rendszerek is a piacon voltak, amelyek 192, egyenként 10 Gb/s-os csatornával, illetve 64, egyenként 40 Gb/s-os csatornával rendelkeztek, ami összesen 2,56 Tb/s-os sebességet jelent. Ez ahhoz elég, hogy 80 darab normál hosszúságú DVD-filmet vigyünk át egyetlen másodperc alatt. A csatornák szorosan helyezkednek el az üvegszálon, 200, 100 vagy 50 GHz-es elválasztással. Cégek által szervezett bemutatókon némi túlzott büszkélkedés után ennek a kapacitásnak a tízszeresét mutatták be laborkörülmények között, de az, hogy ez a technika kikerüljön az éles környezetbe, még pár évbe beletelik. Abban az esetben, ha a csatornák száma nagyon nagy, és a hullámhosszak nagyon közel helyezkednek el egymáshoz, a rendszerre gyakran hivatkoznak **DWDM**-ként (**Dense WDM – sűrű WDM**) is.

A WDM-technika egyik mozgatója az a fejlesztés, amely arra irányul, hogy az összes komponens optikai legyen. Korábban 100 km-enként szét kellett bontani a csatornákat, hogy azokat egyenként villamos jellé alakítsák át az erősítéshez. Az erősítő után a jeleket egyenként vissza kellett alakítani optikaivá, majd ezeket a továbbküldéshez újra össze kellett nyalábolni. Manapság már teljesen optikai erősítők állítják helyre a jeleket 1000 km-enként anélkül, hogy optikai/villamos átalakításra lenne szükség.

A 2.41. ábra példáján egy rögzített hullámhosszokkal dolgozó rendszer látható. Az 1. bemeneti szál bitjei a 3. kimeneti szálra mennek, a 2. bemeneti szálon érkező bitek az 1. kimeneti szálra mennek stb. Lehetséges azonban olyan WDM-rendszert is építeni, amely kapcsolt. Egy ilyen eszközben a hangolhatóság biztosítására a kimeneti szűrőket Fabry–Perot- vagy Mach–Zehnder-interferométerek valósítják meg. Ezek az eszközök lehetővé teszik, hogy a kiválasztott frekvenciákat a vezérlő számítógép dinamikusan módosítsa. Ez a lehetőség nagy rugalmasságot ad ahhoz, hogy üvegszálak rögzített halmazából a telefonhálózaton keresztül számos különböző hullámhosszú útvonalat alakítsanak ki. Az optikai hálózatokkal és a WDM-mel kapcsolatos további információ Ramaswami és mások [2009] munkájában található.

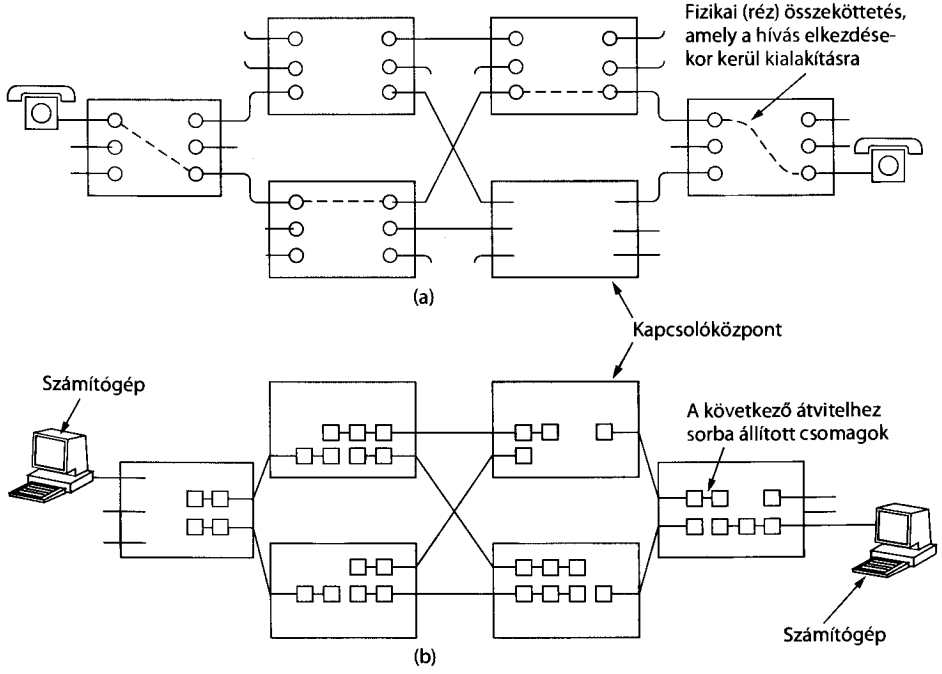
2.6.5. Kapcsolási módok

Egy átlagos távközlési mérnök szempontjából a telefonhálózatnak két fő része van: a külső berendezések (az előfizetői hurkok és a trönkök, amelyek a telefonközponton kívül vannak) és a belső berendezések (a kapcsolók). Eddig csak a külső berendezésekkel foglalkoztunk, ezért itt az ideje, hogy a belső berendezéseket is szemügyre vegyük.

Manapság két különböző kapcsolási módszer van használatban: a vonalkapcsolás és a csomagkapcsolás. A hagyományos telefonrendszer vonalkapcsolásra épül, de a csomagkapcsolás az IP-n keresztül történő hangátviteli technika elterjedésével egyre népszerűbbé válik. Részletesen tárgyaljuk a vonalkapcsolást, majd összehasonlítjuk a csomagkapcsolással. Mindkét kapcsolás elég fontos ahhoz, hogy visszatérjünk hozzájuk a hálózati rétegben.

Vonalkapcsolás

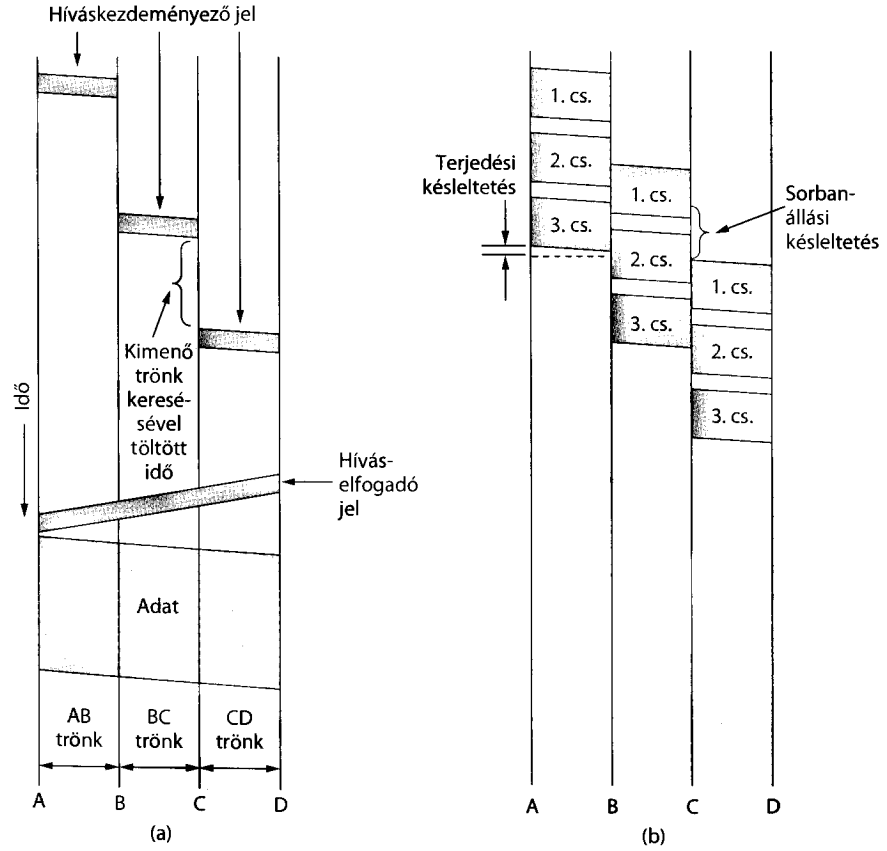
Amikor felhívunk valakit vagy a számítógép végrehajt egy telefonhívást, akkor a távbeszélőrendszer kapcsolóberendezése keres egy olyan fizikai vonalat vagy áramkört (ami lehet rézvezeték, üvegszál vagy akár rádióhullám), amelynek segítségével a saját telefonkészülékünket a hívott fél készülékével összekapcsolja. Ezt a kapcsolási módot **vonalkapcsolásnak** vagy **áramkörkapcsolásnak** (**circuit switching**) nevezzük. A vonalkapcsolást vázlatosan a 2.42.(a) ábrán láthatjuk. A hat négyszög mindegyike egy-



2.42. ábra. *Kapcsolási módok. (a) Vonalkapcsolás. (b) Csomagkapcsolás*

egy telefonközpontot jelent (helyi központ, távhívóközpont stb.). Ezen az ábrán minden központnak három bemenő és három kimenő vonala van. Amikor a központon keresztül létrejön egy összeköttetés, akkor a hívást kezdeményező bemenő vonal és valamelyik kimenő vonal között fizikai kapcsolat létesül. Ezt jelképezik a szaggatott vonalak.

A távbeszélőrendszerek kezdeti időszakában a kapcsolat úgy jött létre, hogy a telefonkezelő a hívó és a hívott fél vezetékét egy áthidaló vezetékkel (jumper cable) kötötte össze. Ezzel kapcsolatban van egy érdekes kis történetünk az automata telefonközpontról, amit a 19. században egy Almon B. Strowger nevű temetkezési vállalkozó fejlesztett ki. Nem sokkal a telefon feltalálása után, amikor valaki meghalt, valamelyik hozzátartozója felhívta a telefonközpontot, és a következőt mondta a kezelőnek: „Kérem, kapcsoljon egy temetkezési vállalkozót!" Mr. Strowger legnagyobb sajnálatára azonban két ilyen vállalkozó is volt a városban, és éppen a másik vállalkozó felesége volt a telefonkezelő. Hamar rájött tehát arra, hogy vagy feltalálja az automata telefonközpontot, vagy tönkremegy. Végül az első utat választotta. Még közel 100 évvel a történet után is világszerte **Strowger-kapcsolónak** nevezték a telefonközpontot. (Arról már nem szól a történet, hogy a munkanélkülivé vált központos kapott-e állást a tudakozóban, ahol olyan kérdésekre kellett válaszolnia, mint például: „Meg tudná adni, kérem, egy temetkezési vállalkozó telefonszámát?")



2.43. ábra. *Események időzítése (a) vonalkapcsolás esetén, (b) csomagkapcsolás esetén*

A 2.42.(a) ábrán látható elrendezés persze nagyon leegyszerűsített, hiszen két telefonkészülék között a fizikai útvonal egyes részei akár olyan mikrohullámú vagy üvegszálas kapcsolatok is lehetnek, amelyekre több ezer telefonhívást multiplexelnek. Az alapelképzelés viszont továbbra is érvényes: ha egyszer létrejön egy összeköttetés, akkor a két végpont között dedikált kapcsolat létesül, és az folyamatosan fennáll addig, amíg a hívás véget nem ér.

A vonalkapcsolás egyik fontos tulajdonsága, hogy a végpontok közötti összeköttetést még az adatok továbbítása *előtt* kell létrehozni. A tárcsázás és a kicsengetés között akár 10 másodperc is eltelhet, sőt távolsági vagy nemzetközi hívásoknál még több is. Eközben a távbeszélőrendszer egy útvonalat keres, ahogy azt a 2.43.(a) ábra szemlélteti. Ne feledjük el, hogy mielőtt az adatátvitel megkezdődhetne, a híváskezdeményező jelzésnek el kell jutnia egészen a hívott készülékig, és a nyugtának vissza kell érkeznie. Sok olyan számítógépes alkalmazás van (például hitelkártya ellenőrzése vásárláskor), amelynél a hosszú kapcsolatfelépítési idő megengedhetetlen.

Ha viszont a kapcsolat felépült, akkor a telefonáló felek között létrejött fenntartott összeköttetésnek köszönhetően az adatok késleltetése lényegében csak az elektromágneses hullámok terjedési sebességéből adódik, ami 1000 km-enként kb. 5 ms. Ugyancsak a felépített kapcsolatnak köszönhető, hogy nincs torlódásveszély, tehát ha megtörtént a kapcsolás, akkor azt követően már sosem kapunk foglalt jelet. A kapcsolás létrejötte előtt persze kaphatunk foglaltsági jelzést, amennyiben a telefonközpont vagy a trönk túlterhelt.

Csomagkapcsolás

Egy másik kapcsolási módszer a 2.42.(b) ábrán látható **csomagkapcsolás** (**packet switching**), amelyet az 1. fejezet ír le. Ennél a kapcsolási módnál a csomagok azonnal átküldésre kerülnek, amint rendelkezésre állnak. A vonalkapcsolással ellentétben nem kell előre kialakítani dedikált útvonalat. Az útválasztók tárol-és-továbbít (store-and-forward) típusú átvitelt használnak az egyes csomagoknak saját útvonalukon történő célba juttatásához. Ez az eljárás nem olyan, mint a vonalkapcsolás, ahol az összeköttetés kialakításának eredményeképp lefoglalásra kerül a sávszélesség az adó és a vevő közötti teljes útvonalon. A vonalon lévő összes adat ezt az útvonalat követi. Más tulajdonságok mellett azáltal, hogy az összes adat ugyanazon az útvonalon halad, csak sorrendben érkezhetnek meg. Csomagkapcsolás esetén nincs rögzített útvonal, így a különböző csomagok különböző útvonalon mehetnek, a küldés időpontjában fennálló hálózati feltételektől függően, így a csomagok nem feltétlenül sorrendben érkeznek meg.

Csomagkapcsolt hálózatban a csomagok méretének szigorú felső korlátja van. Ez biztosítja azt, hogy hosszabb időre (például több milliszekundumra) senki nem tudja kisajátítani az adatátviteli vonalakat, ezért a csomagkapcsolt hálózatok kifejezetten alkalmasak interaktív adatforgalom lebonyolítására. A csomagok késleltetése lecsökken, mivel egy hosszú üzenet első csomagja továbbítható, mielőtt a második teljes egészében megérkezne a címzetthez. A csomag összegyűjtésre kerül az útválasztó memóriájá-

ban, mielőtt elküldésre kerülne a következő útválasztónak, az ebből származó tárol-és-továbbít késleltetés azonban meghaladja a vonalkapcsolás késleltetését. Vonalkapcsolás esetén ugyanis a bitek folyamatosan mennek a vezetéken.

A vonal- és a csomagkapcsolás sok dologban különbözik egymástól. Mivel csomagkapcsolás esetén nincs lefoglalva sávszélesség, ezért előfordulhat, hogy a csomagoknak várniuk kell a továbbításra. Ez **sorbanállási késleltetést** (**queuing delay**) hoz be, valamint torlódást okoz, ha túl sok csomag kerül egyszerre elküldésre. Nem fenyeget azonban az a veszély, hogy foglaltságjelzés kerül kiadásra és a hálózat használhatatlanná válik. Ezért a torlódás máskor jelentkezik vonalkapcsolás esetén (hívásfelépítéskor) és csomagkapcsolásnál (csomagok küldésekor).

Ha egy áramkört egy bizonyos felhasználó lefoglalt, de nincsen forgalmazásra váró adata, akkor az adott áramkör sávszélessége veszendőbe megy, mivel más forgalom nem használhatja ki. A csomagkapcsolás nem pazarolja a sávszélességet, ezért a teljes rendszer szempontjából hatékonyabb. Ezt a kompromisszumot fontos megérteni ahhoz, hogy felfoghassuk a vonalkapcsolás és a csomagkapcsolás közötti különbséget. A kompromisszumot aközött kell megtennünk, hogy garantált szolgáltatásminőséget adunk rossz erőforrás-kihasználás mellett, vagy nem garantált szolgáltatást jól kihasznált erőforrásokkal.

A csomagkapcsolás kevésbé érzékeny a hibákra, mint a vonalkapcsolás. Tulajdonképpen ez az, amiért kitalálták. Ha a vonalkapcsolásnál egy kapcsoló meghibásodik, akkor minden ezen keresztül irányított áramkör megszakad, és egyiken sem lehet ezután adatforgalmat bonyolítani. Csomagkapcsolás használatával ilyenkor a csomagokat a kiesett kapcsolót kikerülő elterelő útvonalakra lehet irányítani.

Az utolsó különbség a vonal- és a csomagkapcsolás között a számlázás algoritmusában van. Vonalkapcsolásnál a számlázás a kezdetektől fogva az idő és a távolság alapján történt. A mobiltelefonok esetében a távolság általában nem játszik szerepet, kivéve a nemzetközi hívásoknál, és az idő is csak kismértékben (vagyis például egy olyan díjcsomag, amelyik 2000 ingyenes percet tartalmaz drágább, mint egy olyan, amelyik 1000 ingyenes percet tartalmaz, valamint néha az éjszakai és hétvégi hívások olcsóbbak a szokásosnál). Csomagkapcsolás esetén a kapcsolat ideje nem számít, de a forgalom mennyisége néha igen. Az internetszolgáltatók az otthoni felhasználóknak általában rögzített havidíjat számláznak, mert nekik ez kevesebb munkát jelent, és az előfizetők is könnyen átlátják ezt a modellt. A gerinchálózati szolgáltatók ezzel szemben a forgalmazott adatmennyiség alapján számláznak a területi hálózatoknak.

A különbségeket a 2.44. ábra táblázatában foglaltuk össze. A telefonhálózatok hagyományosan vonalkapcsolást használnak, hogy kiváló minőségű telefonhívásokat biztosítsanak, a számítógép-hálózatok pedig csomagkapcsolást használnak az egyszerűség és hatékonyság érdekében. Vannak azonban jelentős különbségek. Néhány régebbi számítógép-hálózat a felszín alatt vonalkapcsolást használt (mint például az X.25)[3], illetve néhány új telefonhálózat csomagkapcsolást használt IP-n keresztül történő hangátviteli technikával. Ez kívülről a felhasználó számára úgy néz ki, mint egy normál telefonhívás, de a hálózaton belül a hangadatok csomagjai csomagkapcsolással kerülnek továbbításra.

3 Az X.25 hálózat virtuálisáramkör-alapú csomagkapcsolt hálózat. A szerző ebben téved. *(A lektor megjegyzése)*

Tulajdonság	Vonalkapcsolt	Csomagkapcsolt
Összeköttetés kiépítése	Szükséges	Nem szükséges
Dedikált fizikai útvonal	Igen	Nem
Minden csomag ugyanazon az útvonalon halad	Igen	Nem
A csomagok sorrendben érkeznek meg	Igen	Nem
Egy kapcsoló kiesése végzetes	Igen	Nem
Rendelkezésre álló sávszélesség	Rögzített	Változó
A torlódások lehetséges ideje	Összeköttetés létesítésekor	Minden csomagnál
Veszhet kárba sávszélesség	Igen	Nem
Tárol-és-továbbít átvitel	Nem	Igen
Számlázás	Perc alapon	Csomag alapon

2.44. ábra. *A vonalkapcsolt és a csomagkapcsolt hálózatok összehasonlítása*

Ez a megközelítés lehetővé teszi a feltörekvő piacok számára olcsó nemzetközi hívások lebonyolítását telefonkártyán keresztül, de elképzelhető, hogy rosszabb minőséggel, mint amilyet a telefontársaság nyújtana.

2.7. A mobiltelefon-rendszer

A hagyományos telefonrendszer, még ha egy szép napon több gigabites üvegszálakkal is fog rendelkezni a végpontok közötti teljes szakaszon, akkor sem fogja tudni kiszolgálni a felhasználók egy bizonyos, egyre növekvő csoportját. Ez a csoport pedig az utazó, folyton úton levő felhasználóké. Az emberek manapság elvárják, hogy telefonálhassanak repülőről, autóból, uszodából és az esti kocogás közben is. Néhány éven belül azt is el fogják várni, hogy ezekről a helyekről és más helyekről is küldhessenek e-leveleket, továbbá a világhálón is szörfölhessenek. Ennek következtében a vezeték nélküli telefonálás iránt hatalmas mértékű az érdeklődés. A következő szakaszokban ezt a témát fogjuk részletesen tanulmányozni.

A mobiltelefon-rendszereket széles körben használják beszéd- és adattovábbításra. A **mobiltelefonok** három egymást követő **generáción** (**1G, 2G, 3G**) mentek keresztül műszaki fejlődésük során:

1. analóg beszédtovábbítás,
2. digitális beszédtovábbítás,
3. digitális beszéd- és adattovábbítás (internet, e-levelezés stb.).

(A mobiltelefonok nem keverendőek össze a **zsinór nélküli telefonokkal** (**cordless phone**), amik olyan eszközök, amelyek egy bázisállomásból és egy kézibeszélőből állnak. Ezeket egyetlen készletben adják el egyetlen háztartáson belüli együttes használatra. Hálózat kialakítására nem alkalmasak, ezért nem is vizsgáljuk ezeket a továbbiakban.)

Bár a tárgyalás legnagyobb része az ezekben a rendszerekben alkalmazott műszaki megoldásokkal foglalkozik, érdemes megjegyezni, hogy a politikai döntéseknek és az apró piaci fogásoknak óriási hatása lehet. Az első mobiltelefon-rendszert az AT&T dolgozta ki Amerikában, az FCC pedig ennek a rendszernek a használatát jelölte ki országosan kötelezőnek. Ennek eredményeképpen az egész Egyesült Államoknak egyetlen (analóg) rendszere alakult ki, és egy Kaliforniában vásárolt mobiltelefon New Yorkban is működött. Ezzel szemben, amikor a mobiltelefonok megérkeztek Európába, minden ország saját rendszert fejlesztett ki, ami kudarchoz vezetett.

Európa azonban tanult a hibából, és amikor a digitális rendszerek megjelentek, a kormányok által működtetett telefontársaságok képviselői összejöttek, és egyetlen rendszert szabványosítottak (a GSM-et), ezért bármely európai mobiltelefon Európában bárhol működik. Az amerikai kormány ebben az időben éppen túl volt annak a döntésnek a meghozatalán, hogy nem szabad szerepet vállalnia a szabványosítási eljárásokban, így a piacra hagyta a digitális mobiltelefon-rendszer szabványosítását. Ez a döntés azt eredményezte, hogy a különböző eszközgyártók különbözőféle mobiltelefonokat kezdtek gyártani. Ennek következményeként az Egyesült Államokban ma két nagy, egymással inkompatibilis digitális mobiltelefon-rendszer van használatban (valamint egy kisebb).

Annak ellenére, hogy az elején az Egyesült Államok vezetett a mobiltelefonok száma és használata terén, Európa azóta messze átvette a vezetést. Ennek egyik oka az, hogy egész Európában egyetlen rendszert használnak, de más okok is vannak. Egy másik terület, ahol Európa és Amerika különbözött, az a telefonszámok kérdése. Amerikában a mobiltelefonok kapcsolási számai keverednek a hagyományos (telepített) telefonok kapcsolási számaival, így a hívó semmiből sem látja, hogy mondjuk a (212) 234-5678 egy vezetékes telefoné (olcsó vagy ingyenes hívás) vagy egy mobiltelefoné (drága hívás). Annak érdekében, hogy megelőzzék a felhasználók telefonhasználattal kapcsolatos félelmeit, a telefontársaságok úgy döntöttek, hogy a mobiltelefon tulajdonosával fizettetik meg a bejövő hívás díját. Ennek következtében sokan vonakodtak mobiltelefont vásárolni, mivel attól tartottak, hogy hatalmas számlájuk lesz pusztán a bejövő hívások fogadása miatt. Európában a mobiltelefonoknak különleges körzetszámuk van (a kék és zöld számokhoz hasonlóan), így azokat azonnal fel lehet ismerni. Ebből kifolyólag a „hívó fizet" szokásos szabálya Európában a mobiltelefonokra is vonatkozik (a nemzetközi hívások kivételével, ahol a költségeket megosztják a két fél között).

A harmadik tényező, amely Európában nagy hatással van a mobiltelefonok terjedésére, az előre fizetett (felöltőkártyás) mobiltelefonok széles körű használata (néhány területen akár 75%). Ezeket sok boltban ugyanazokkal a formaságokkal lehet megvásárolni, mint egy rádiót. Fizesd és vidd. Előre fel vannak töltve 20 vagy 50 euróval, és egy titkos PIN-kód segítségével újra fel lehet tölteni, amikor az egyenleg lenullázódik. Ennek következtében Európában gyakorlatilag minden tizenéves és néhány ennél fiatalabb gyerek is rendelkezik (általában felöltőkártyás) mobiltelefonnal, hogy szüleik könnyebben

megtalálhassák őket. Mindezt ráadásul anélkül, hogy a gyerek óriási számlát tudna csinálni. Ha a mobiltelefont csak ritkán használják, akkor lényegében ingyen van, mivel nincs havi előfizetési díj, és a bejövő forgalmat sem számlázzák ki.

2.7.1. Első generációs (1G) mobiltelefonok: analóg beszédátvitel

Eleget beszéltünk már a mobiltelefonok politikai és piaci vonatkozásairól. Most vegyük szemügyre a műszaki megoldásokat, a legkorábbi rendszerrel kezdve. A mozgó rádiótelefonokat elvétve már a 20. század korai évtizedeiben is használták a katonai és a hajózási távközlésben. 1946-ban telepítették az első autótelefon-rendszert St. Louisban. Ez a rendszer egy magas épület tetejére felszerelt egyetlen adót használt és egyetlen csatornája volt, amelyet adásra és vételre egyaránt használtak. Mielőtt a felhasználó beszélni kezdett, meg kellett nyomnia egy gombot, amely bekapcsolta az adót, és kiiktatta a vevőt. Ezeket az úgynevezett **átkapcsolásos rendszereket** (**push-to-talk system**) számos városban telepítették az 1950-es évek végén. A CB-rádiók, a taxik és a tv-műsorokban látható rendőrök gyakran élnek ezzel a műszaki megoldással.

Az 1960-as években telepítették az **IMTS**-t (**Improved Mobile Telephone System – javított mobiltelefon-rendszer**), amely szintén egy nagy teljesítményű (200 wattos) adót használt, amelyet egy domb tetején helyeztek el. Ennek a rendszernek már két frekvenciája volt, egy az adáshoz és egy a vételhez, így az átkapcsológombra már nem volt szükség. Mivel az egyes mobiltelefonoktól eredő kommunikáció befelé másik csatornán haladt, mint a kifelé haladó jelek, a mobilok felhasználói nem hallhatták egymást (a taxikban használatos átkapcsolásos rendszerrel ellentétben).

Az IMTS 23 csatornát támogatott, amelyek a 150-től 450 MHz-ig terjedő sávon voltak szétszórva. A csatornák kis száma miatt a felhasználóknak gyakran kellett hosszan várniuk a tárcsahang megjelenésére. A domb tetején felállított adótorony nagy adási teljesítményéből kifolyólag pedig a szomszédos rendszereknek több száz kilométerre kellett lenniük egymástól az interferencia elkerüléséhez. Mindent összevetve, a korlátozott kapacitás volt az oka, hogy a rendszer a gyakorlatban kevéssé volt használható.

A fejlett mobiltelefon-rendszer (AMPS)

Az **AMPS** (**Advanced Mobile Phone System – fejlett mobiltelefon-rendszer**) megjelenésekor mindez megváltozott. Ezt a rendszert a Bell Labs fejlesztette ki, és először az Egyesült Államokban telepítették 1982-ben. TACS néven Nagy-Britanniában is használták, továbbá Japánban is, ahol MCS-L1 volt a neve. Bár az AMPS-t 2008 óta lényegében nem használják, ennek ellenére megvizsgáljuk a rá épülő 2 és 3G-s rendszerek jobb megértése érdekében.

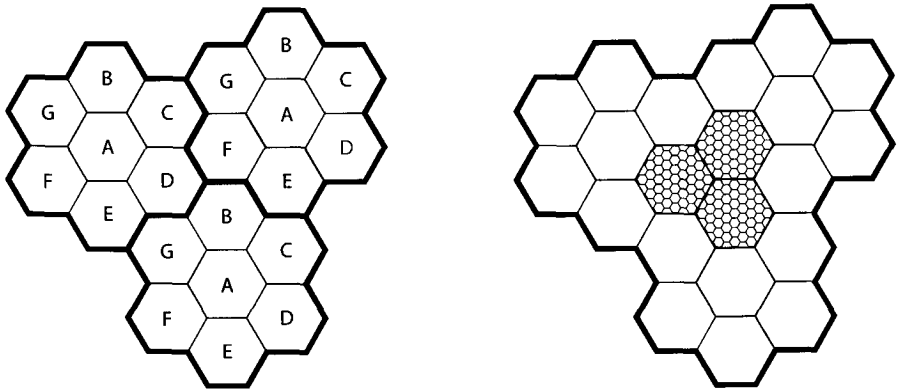
Minden mobiltelefon-rendszerben **cellákra** (**cell**) osztják a földrajzi területet (innen ered az angol „cell phone" név is). Az AMPS-ben a cellák átmérője általában 10 és 20 km között van, a digitális rendszerekben ennél kisebb. Minden cella egy olyan frekvenciahalmazt használ, amelynek egyik elemét sem alkalmazzák a szomszédjai. A viszonylag kis cellák használata és a frekvenciák újrahasználása a közeli (de nem szomszédos)

cellákban az a két kulcsfontosságú ötlet, amely a cellás rendszereket sokkal nagyobb kapacitásúvá teszi az előző rendszereknél. Míg egy 100 km-es átmérőjű IMTS-rendszer egyetlen hívást tud kezelni minden frekvencián, egy AMPS-rendszernek akár 100 különálló, 10 km-es cellája lehet ugyanezen a területen, és így 10–15 beszélgetést kezelhet minden frekvencián, egymástól távoli cellákban. A cellás rendszer így legalább egy nagyságrenddel megnöveli a rendszer kapacitását, de a cellák méretének csökkentésével a kapacitás akár több nagyságrenddel is megnövelhető. Mindezen felül a kisebb cellaméret azzal is jár, hogy kisebb adóteljesítményre van szükség, amely kisebb és olcsóbb adókhoz, illetve telefonokhoz vezet.

A frekvencia-újrahasznosítás ötletét a 2.45.(a) ábra szemlélteti. A cellák a valóságban többé-kevésbé kör alakúak, de a hatszög alakú cellákat könnyebb modellezni. A 2.45.(a) ábrán a cellák körülbelül azonos méretűek és hét cellából álló egységekbe vannak csoportosítva. Minden betű egy frekvenciacsoportot jelöl. Figyeljük meg, hogy az egyes frekvenciacsoportok körül olyan, nagyjából két cella széles tartományok vannak, ahol az adott frekvenciát nem használják újra! Ezek biztosítják a kellő távolságot és a kismértékű interferenciát.

Fontos kérdés, hogy hol tudjuk megfelelő magasságban elhelyezni a bázisállomás-antennákat. Mivel nehéz ilyen helyeket találni, és a római katolikus egyház az egész világon jelentős számú lehetséges antennahellyel rendelkezik, amelyek ráadásul ugyanannak a szervezetnek a kezelésben is vannak, ezért néhány telekommunikációs szolgáltató egyezséget kötött az egyházzal.

Amikor egy adott területen belül a felhasználók száma akkorára növekszik, hogy az már túlterheli a rendszert, csökkentik a teljesítményt, és a túlterhelt cellákat kisebb **mikrocellákra** (**microcell**) bontják fel annak érdekében, hogy a 2.45.(b) ábrán is látható módon többször lehessen újrahasználni a frekvenciákat. A telefontársaságok néha ideiglenes mikrocellákat is telepítenek a nagy sportesemények, rockkoncertek és más olyan helyek körzetébe, ahol nagyszámú mobilhasználó gyűlik össze néhány órára. Az ideiglenes mikrocellákat műholdas kapcsolatra képes hordozható adókkal valósítják meg.



2.45. ábra. *(a) A frekvenciákat nem használják újra a szomszédos cellákban. (b) Több felhasználó kiszolgálása kisebb cellák alkalmazásával*

Minden cella közepén található egy bázisállomás, amellyel a cellában tartózkodó összes telefon kapcsolatban van. A bázisállomás egy számítógépből és egy antennából, valamint az ahhoz kapcsolódó adóvevőből áll. A kisebb rendszerekben minden bázisállomás összeköttetésben áll egyetlen **MTSO**-nak (**Mobile Telephone Switching Office – mobiltelefon-kapcsolóállomás**) vagy **MSC**-nek (**Mobile Switching Centre – mobil-kapcsolóközpont**) nevezett eszközzel. A nagyobb rendszerekben több MSC-re is szükség lehet, amelyek közül mindegyik egy második szintű MSC-hez csatlakozik, és így tovább. Az MSC-k a telefonhálózatban használatos helyi központok megfelelői, és legalább egy telefonhálózati helyi központtal összeköttetésben is állnak. Az MSC-k a bázisállomásokkal, egymással és a vezetékes telefonhálózattal egy csomagkapcsolt hálózaton keresztül kommunikálnak.

Az egyes mobiltelefonok minden pillanatban logikailag egy bizonyos cellához tartoznak, és az adott cella bázisállomásának irányítása alatt állnak. Amikor egy mobiltelefon fizikailag elhagyja a cellát, és a cella bázisállomása azt veszi észre, hogy a telefon jele gyengülni kezd, megkérdezi a szomszédos bázisállomásokat, hogy ők mekkora teljesítményt észlelnek a telefon felől. Amikor a válasz megérkezik, a bázisállomás átadja a telefon felügyeletét annak a cellának, amelyik a legerősebb jelet veszi tőle, vagyis ahol a telefon éppen tartózkodik. A telefon ezután értesítést kap az új főnökéről, és felkérik arra, hogy váltson csatornát, ha éppen hívása van folyamatban (mivel a régit nem használja a szomszédos cellákban). Ez az **átadásnak** (**handoff**) nevezett folyamat körülbelül 300 ms-ig tart. A csatornakiosztást a rendszer agya, az MSC végzi, a bázisállomások tulajdonképpen csak rádiós átjátszóállomások.

Csatornák

Az AMPS FDM-et használ a csatornák elválasztásához. A rendszer 832 duplex csatornát használ, amelyek szimplex csatornapárokból állnak. A módszer neve **FDD** (**Frequency Division Duplex – frekvenciaosztásos kettőzés**). A 832 szimplex adási csatorna 824 MHz és 849 MHz között helyezkedik el, a 832 szimplex vételi csatorna pedig 869 MHz és 894 MHz között kapott helyet. Minden egyes szimplex csatorna 30 kHz sávszélességű.

A 832 csatornát négy kategóriába osztják. A vezérlési csatornákat (bázistól a mobil felé) a rendszer felügyeletére használják. A hívási csatornák (bázistól a mobil felé) azt a célt szolgálják, hogy a mobilfelhasználókat értesítsék a beérkező hívásokról. A hozzáférési csatornák (kétirányú) a hívások felépítéséhez és a csatornák kiosztásához szükségesek. Végül az adatcsatorna (kétirányú) a beszéd, a faxok és az adatok továbbítására szolgál. Mivel a szomszédos cellákban nem lehet az azonos frekvenciákat használni és 21 csatorna vezérlési célra van fenntartva minden egyes cellában, így az egy cellában használható beszédcsatornák száma sokkal kisebb mint 832, általában 45 körül mozog.

Híváskezelés

Az AMPS-ben minden mobiltelefon rendelkezik egy 32 bites gyári számmal és egy 10 számjegyű hívószámmal, amelyeket PROM-ban tárol. A hívószám egy 10 biten tárolt 3 jegyű körzetszámból és egy 24 bites, 7 számjegyű előfizetői számból áll. Amikor a tele-

font bekapcsolják, végigkeresi a 21 előre beprogramozott vezérlési csatornát, hogy megtalálja a legerősebb jelet. A telefon ezután szétküldi a 32 bites gyári számát és a 34 bites telefonszámát. Az AMPS az összes vezérlési információhoz hasonlóan ezt a csomagot is digitális formában, többszörözve és hibajavító kóddal ellátva viszi át, de maguk a beszédcsatornák analóg továbbításúak.

Amikor a bázisállomás meghallja a bejelentkezést, jelenti az új előfizető megérkezését az MSC-nek, amely feljegyzi ezt a tényt, és tájékoztatja az előfizető saját MSC-jét a felhasználó pillanatnyi tartózkodási helyéről. Normál működés során a telefon nagyjából 15 percenként bejelenti magát.

Hívás kezdeményezéséhez a felhasználónak be kell kapcsolnia a mobiltelefont, be kell billentyűznie a hívott fél számát, és meg kell nyomnia a „küldés" gombot. A telefon ekkor a hozzáférési csatornán elküldi a hívott fél számát, valamint a saját azonosítóját. Amennyiben ezen a csatornán ütközés történik, a telefon később újra próbálkozik. Amikor a bázisállomás megkapja a kérést, tájékoztatja róla az MSC-t. Ha a hívó az MSC üzemeltetőjének (vagy egyik partnerének) egyik előfizetője, akkor az MSC keres egy üres csatornát a hívás számára. Amennyiben talál egy üres csatornát, a számát visszaküldi a vezérlési csatornán. A mobiltelefon ekkor automatikusan a kiválasztott beszédcsatornára vált, és addig vár, amíg a hívott fél felveszi a telefont.

A bejövő hívások másképp működnek. Először is, minden olyan telefon folyamatosan figyeli a hívási csatornát, amelyen éppen nincsen hívás folyamatban, hogy érzékelhesse a neki szánt üzeneteket. Amikor egy mobiltelefonra valaki hívást kezdeményez (akár vezetékes telefonról, akár egy másik mobiltelefonról), egy olyan csomag érkezik a hívott fél saját MSC-jéhez, amely a hívott fél hollétét hivatott kideríteni. Az MSC ezután egy másik csomagot küld a telefon pillanatnyi cellájában elhelyezett bázisállomásnak, amely erre egy „14-es egység, jelentkezz!"-hez hasonló adást küld szét a hívási csatornán. A hívott telefon „Jelen"-nel válaszol a hozzáférési csatornán. A bázisállomás ekkor valami olyasmit mond, hogy „14-es egység, hívásod van a 3-as csatornán.". Amikor ezt az üzenetet megkapja, a hívott telefon átvált a 3-as csatornára, és csengetési hanggal jelez a hívott félnek (esetleg egy olyan dallammal, amelyet a tulajdonos valakitől születésnapi ajándékba kapott).

2.7.2. Második generációs (2G) mobiltelefonok: digitális beszédátvitel

A mobiltelefonok első generációja analóg volt, de a második generáció már digitális. A digitális átvitelre való áttérésnek számos előnye van. Nagyobb kapacitást biztosít azáltal, hogy lehetővé teszi a hangjelek digitalizálását és tömörítését. Ez a biztonságot is javítja azáltal, hogy a hang- és vezérlőjelek titkosíthatók. Ezenkívül megakadályozza a hamisnév-használatot és a lehallgatást, függetlenül attól, hogy az szándékos lehallgatás, vagy az RF-terjedés miatti egyéb hívások visszhangja. Végül lehetővé tesz új szolgáltatásokat, mint amilyen például a szöveges üzenetküldés.

Mint ahogy az első generáció idején nem volt nemzetközi szabványosítás, ugyanúgy a második generáció idején sem történt nemzetközi szabványosítás. Számos különböző rendszert hoztak létre, és ezek közül hármat alkalmaztak széleskörűen. A **D-AMPS** (**Digital Advanced Mobile Phone System – digitális AMPS**) az AMPS digitális változata, amely az AMPS-sel együtt létezik, és TDM-et használ több hívás azonos frekvenciájú

csatornára helyezéséhez. Ezt az IS-54 nemzetközi szabvány írja le, és annak jogutódja az IS-136. A **GSM** (**Global System for Mobile Communications – globális mobilkommunikációs rendszer**) lett a domináns rendszer, habár elterjedése az Egyesült Államokban lassú volt, de most gyakorlatilag mindenütt a világon használják. A D-AMPS-hez hasonlóan a GSM az FDM és TDM kombinációjára épül. A **CDMA**-t (**Code Division Multiple Access – kódosztásos többszörös hozzáférés**) az IS-95 nemzetközi szabvány írja le. Ez teljesen különböző rendszer, és sem az FDM-re, sem a TDM-re nem épül. Annak ellenére, hogy a CDMA nem lett domináns 2G-rendszer, ez képezi a 3G-rendszer alapját.

A marketinges irodalom néha a **PCS** (**Personal Communication System – személyi kommunikációs rendszer**) névvel illeti a második generációs (vagyis digitális) rendszereket. Eredetileg a kifejezés 1900 MHz-en működő mobiltelefont jelentett, de ezt a megkülönböztetést manapság már ritkán teszik meg.

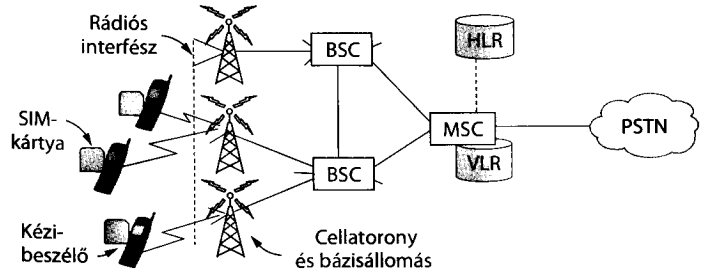
Ezután a GSM leírása következik, mivel ez a domináns 2G-rendszer. A következő részben a CDMA-val részletesen is foglalkozunk a 3G-rendszerek leírásakor.

GSM – a globális mobilkommunikációs rendszer

A GSM az 1980-as években kelt életre, amikor egyetlen európai 2G-szabványt próbáltak kialakítani. A feladatot egy francia távközlési csoporthoz rendelték, amelynek a neve Group Specialé Mobile (GSM). Az első GSM-rendszerek telepítése 1991-ben kezdődött és gyors sikert ért el. Hamarosan nyilvánvalóvá vált, hogy a GSM nem csak Európában lesz sikeres, teret hódított Ausztráliában is, így a GSM-et átnevezték, hogy világszerte vonzóbbá váljon.

A GSM és a többi, jövőben tanulmányozandó mobiltelefon-rendszer is megtartotta az 1G-rendszer cellaalapú kialakítását, a cellák közötti frekvencia-újrafelhasználást, valamint az átadással megvalósított mobilitást az előfizető mozgása során. Csak a részletek különböznek. A következő részben röviden megtárgyaljuk a GSM néhány főbb tulajdonságát. A GSM szabványa azonban több mint 5000 (sic!) nyomtatott oldalt tesz ki, melynek nagy része a rendszer műszaki részleteivel foglalkozik, különös tekintettel az adók és vevők szinkronizációjára és a vevők olyan kialakítására, amely lehetővé teszi a többutas jelterjedés kezelését. Ezekkel a továbbiakban nem foglalkozunk.

A 2.46. ábra mutatja, hogy a GSM-architektúra hasonló az AMPS-architektúrához, de az összetevők neve eltérő. A mobilkészülék maga két részre van osztva, a kézibeszélőre és



2.46. ábra. *GSM mobilhálózati architektúra*

egy kivehető chipre, amely előfizetői és számlainformációt tartalmaz. Ezt a chipet **SIM-kártyának** hívják (**Subscriber Identity Modul – előfizető-azonosító modul**). A SIM-kártya aktiválja a készibeszélőt és tartalmazza azokat a titkos adatokat, amelyek lehetővé teszik, hogy a készülék és a hálózat azonosítsa egymást, és hogy a párbeszédet titkosítsák. A SIM-kártya kivehető és áthelyezhető másik kézibeszélőbe, hogy átváltoztassa azt a kézibeszélőt az előfizető hálózat szerinti mobiltelefonjává.

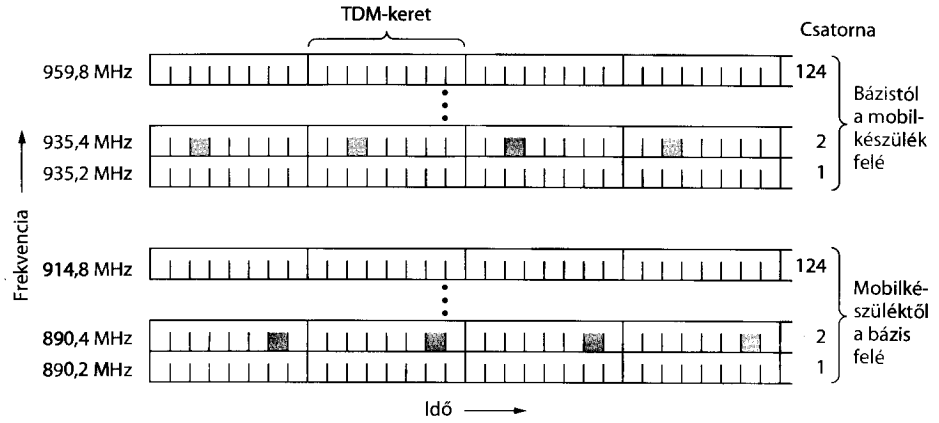
A mobilkészülék a bázisállomással **rádiós interfészen** keresztül kommunikál, amelyet mindjárt ismertetünk. Minden cella-bázisállomás **BSC**-hez (**Base Station Controller – bázisállomás-vezérlő**) csatlakozik, amely vezérli a cellák rádió-erőforrásait, valamint kezeli a kézibeszélőt. A BSC MSC-hez csatlakozik (ahogy az AMPS-ben is), amely a hívásokat vezérli és csatlakozik a nyilvános kapcsolt telefonhálózathoz (Public Switched Telephone Network, PSTN).

A hívások vezérléséhez az MSC-nek tudnia kell, hogy a mobilkészülékek pillanatnyilag hol találhatók. Az MSC egy adatbázist tart fenn az általa kezelt cellákhoz tartozó közeli mobilkészülékekről. Ezt az adatbázist **VLR-nek** (**Visitor Location Register – látogató-elhelyezkedési regiszter**) hívják. A mobilhálózatban is található egy adatbázis, amely megadja az összes mobilkészülék utolsó ismert helyét. Ezt **HLR-nek** (**Home Location Register – otthon-elhelyezkedési regiszter**) hívják. Ez az adatbázis vezérli a bejövő hívásokat a megfelelő helyre. Mindkét adatbázist naprakészen kell tartani, ahogy a mobilkészülék celláról cellára változtatja a helyét.

Most a rádiós interfészt írjuk le részletesen. A GSM különböző frekvenciatartományokban működik világszerte, 900, 1800 és 1900 MHz-en. Több frekvenciasáv kerül lefoglalásra, mint ahogy az AMPS esetében is, több felhasználó támogatása érdekében. A GSM frekvenciaosztásos kettőzésű cellás rendszer, mint az AMPS is. Azaz minden mobilkészülék egy frekvencián ad, és egy másik, nagyobb frekvencián vesz (GSM esetén 55 MHz-cel, AMPS esetén pedig 80 MHz-cel nagyobb). Az AMPS-sel ellentétben a GSM-nél azonban egy frekvenciapár van felosztva időszeletekre időosztásos multiplexeléssel. Ily módon ezt több mobilkészülék osztja meg.

Több mobiltelefon kezelése érdekében a GSM-csatornák szélesebbek, mint az AMPS-csatornák (itt 200 kHz, ott 30 kHz). 1200 kHz-es csatorna látható a 2.47. ábrán. A 900 MHz-es tartományban működő GSM-rendszernek 124 pár szimplex csatornája van. Minden szimplex csatorna 200 kHz széles és nyolc párhuzamos összeköttetést támogat időosztásos multiplexeléssel. Minden, éppen aktív állomás egy időszeletet kap egy csatornapáron. Elméletileg minden cellában 992 csatornát lehetne fenntartani, de ezek jelentős része nem érhető el, mivel csak így kerülhetők el a szomszédos cellákkal való frekvenciaütközések. A 2.47. ábrán a nyolc besatírozott időszelet ugyanahhoz az összeköttetéshez tartozik, mindkét irányba négy darab. Az adás és a vétel azért nem történik ugyanabban az időszeletben, mert a GSM-rádiók nem tudnak egyszerre adni és venni, és a két üzemmód közötti váltáshoz is szükségük van valamennyi időre. Ha a 890,4/935,4 MHz-en és a 2-es időszeletben működő mozgó állomás adni akar a bázisállomásnak, akkor az alsó négy besatírozott időszeletben (és az időben ezek után következő időszeletekben) teheti ezt meg, minden időszeletbe valamennyi adatot téve addig, amíg minden adatot el nem küldött.

A 2.47. ábrán látható TDM-időszeletek egy komplex keretezési hierarchia részét képezik. Mindegyik TDM-időszeletnek megvan a maga sajátságos felépítése, az időszeletek csoportjai pedig multikereteket formálnak, amelyeknek szintén jellegzetes a struktúrája. Ennek a hierarchiának egy egyszerűsített változata látható a 2.48. ábrán. Itt láthatjuk, hogy minden TDM-időszelet 148 bites adatkeretekből épül fel, amely 577 μs-ig foglalja a csatornát (és tartalmaz egy 30 μs-os védőidőt minden időszelet után). Mindegyik adatkeret három 0 bittel kezdődik és végződik. Ezek a keretek elkülönítését segítik. Az adatkeretben található még két 57 bites *Információs* mező, amelyekhez tartozik egy vezérlőbit. Ez a bit jelzi, hogy a következő *Információs* mező hangot vagy adatokat tartalmaz. Az *Információs* mezők között van egy 26 bites *Szinkron* mező, amelynek segítségével a vevő az adó kerethatáraihoz szinkronizálódhat.

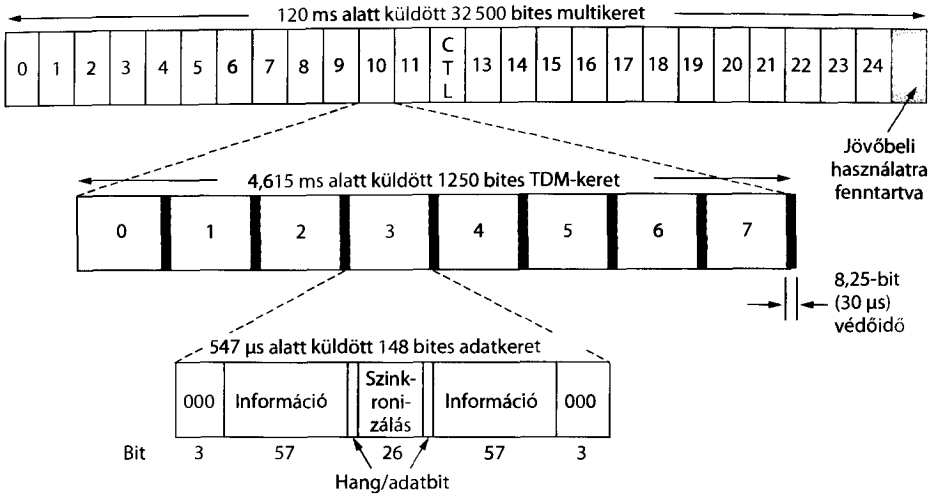


2.47. ábra. *GSM, amely 124 frekvenciacsatornát használ, amelyek közül mindegyik egy 8 időszeletes TDM-rendszert használ*

Egy adatkeret elküldéséhez 547 μs-ra van szükség, de az adó csak 4,615 ms-onként küldhet egy-egy keretet, mivel hét másik állomással osztozik a csatornán. A csatornák teljes kapacitása 270,833 kb/s, amely nyolc felhasználó között oszlik szét. Azonban – ahogy az AMPS-nél is –, a többletbitek itt is felemésztik a sávszélesség nagy részét, mindössze végül is 24,7 kb/s sebességet hagyva a hibajavítás előtti felhasználói adatoknak. Hibajavítás után 13 kb/s marad beszédátvitelre. Ez lényegesen kevesebb, mint a tömörítetlen hangjelekhez használt 64 kb/s PCM a vezetékes telefonhálózatokban. A mobileszközön végzett tömörítés kis minőségvesztéssel el tudja érni ezt a szintet.

Ahogy a 2.48. ábrán látható, nyolc adatkeret alkot egy TDM-keretet, és 26 TDM-keretből áll össze egy 120 ms hosszú multikeret. A multikeretben a 12-es szeletet vezérlésre használják, míg a 25-ös rést fenntartják későbbi használatra, így csak 24 szelet használható a felhasználók forgalmának továbbítására.

A 2.48. ábrán bemutatott 26 szeletet tartalmazó multikeret mellett használnak még egy 51 szeletet tartalmazó multikeretet is (ez nincs az ábrán). Ezeknek a szeleteknek egy része olyan vezérlési csatornákat tartalmaz, amelyeket a rendszer felügyeletére használnak. A **körözvény-vezérlési csatorna** (**broadcast control channel**) egy, a bázisállomás által generált folytonos adatfolyam, amely a bázisállomás azonosítóját és a csatorna állapotinformációját tartalmazza. Az összes mobilállomás figyeli ennek a csatornának a jelszintjét annak megállapítására, hogy mikor léptek át egy újabb cellába.



2.48. ábra. *A GSM keretezési struktúra egy részlete*

A **megkülönböztetett vezérlési csatorna** (**dedicated control channel**) szolgál a helymeghatározás, a regisztráció, valamint a hívásfelépítés lebonyolítására. Gyakorlatilag minden bázisállomás kezel egy adatbázist (VLR), amelyben nyilvántartja az aktuálisan fennhatósága alá eső mobilállomásokat. Az adatbázis karbantartásához szükséges információt a megkülönböztetett vezérlési csatornán továbbítják.

Végül van egy **közös vezérlési csatorna** (**common control channel**), amely három logikai alcsatornából tevődik össze. Ezek közül az első a **felhívási csatorna** (**paging channel**), amelyen keresztül a bázisállomás jelzi a beérkező hívásokat. Az összes mobilállomás folyamatosan figyeli ezt a csatornát, olyan hívások után kutatva, amelyekre válaszolniuk kell. A második a **véletlen hozzáférésű csatorna** (**random access channel**), amely lehetővé teszi a felhasználók számára azt, hogy időszeletet kérjenek a kijelölt vezérlési csatornán. A kijelölt vezérlési csatorna időszeletének felhasználásával az állomás egy hívást tud felépíteni. Az így lefoglalt időszeletről értesítés a harmadik alcsatornán, a **hozzáférés-engedélyező csatornán** (**access grant channel**) érkezik az állomáshoz.

És végül, a GSM és az AMPS abban is különbözik, ahogyan az átadást kezelik. AMPS esetén az átadás teljesen az MSC felügyelete alá tartozik, a mobileszközök segítségét nem veszi igénybe. A GSM időszeleteinél az idő nagy részében a mobilkészülék nem ad és nem is vesz. Ezeket az üres időszeleteket a mobilkészülékek más közeli bázisállomások jelminőségének mérésére használhatják. Ezt meg is teszik és elküldik az információt a BSC-nek. A BSC ennek segítségével meg tudja határozni, hogy a mobil mikor hagy el egy cellát, és mikor lép be a másikba, hogy el tudja végezni az átadást. Ezt a módszert **MAHO-nak** (**Mobile Assisted HandOff – átadás mobiltelefon segítségével**) nevezték el.

2.7.3. Harmadik generációs (3G) mobiltelefonok: digitális beszéd- és adatátvitel

A mobiltelefonok első generációja analóg, a második generációja pedig digitális beszédátvitellel működött. A harmadik generáció (vagy **3G**) teljesen digitális beszéd- és adatátvitelt használ.

Több tényező is hajtja előre az ipart. Először is, a vezetékes hálózaton az adatforgalom mennyisége máris túllépte a beszédforgalomét, és továbbra is exponenciálisan nő, ezzel szemben a beszédforgalom lényegében változatlan. Sok ipari szakértő azt reméli, hogy az adatforgalom hamarosan átveszi a főszerepet a beszédtől a mozgó eszközökön is. Másodszor, a telefon-, szórakoztató- és számítástechnikai ipar mind digitális megoldásokra váltott, és gyorsan közelítenek egymáshoz. Sok ember kezd epekedni egy olyan könnyű, hordozható eszköz láttán, mint amilyen a telefon, a zenelejátszó, a videolejátszó, az e-levelezési terminál, a webinterfész, a játékgép és számos más eszköz, amely a világszerte használható, nagy sávszélességű vezeték nélküli internetkapcsolattal rendelkezik.

Az Apple iPhone jó példa az ilyen típusú 3G-eszközre. Ezzel az emberek folyamatosan csatlakozni tudnak a vezeték nélküli adatszolgáltatásokhoz. Az AT&T vezeték nélküli adatkötegek mennyisége meredeken növekszik az iPhone-ok népszerűségével. A probléma az, hogy az iPhone 2.5G-hálózatot használ (továbbfejlesztett 2G-hálózat, de nem igazi 3G) és nincs elegendő adatkapacitás a felhasználók igényeinek kielégítéséhez. A 3G-mobiltelefon célja, hogy elegendő vezeték nélküli sávszélességet biztosítson a felhasználói igények kielégítéséhez.

Az ITU már 1992-ben megpróbált kissé pontosítani ezen az álmon, és kiadott egy tervet az odajutáshoz, amelyet **IMT-2000**-nek neveztek el. Az IMT feloldása **International Mobile Telecommunications** (**nemzetközi mobil telekommunikáció**). Az IMT-2000 hálózat feltehetően a következő alapvető szolgáltatásokat kínálja a felhasználóinak:

1. kiváló minőségű beszédtovábbítás,
2. üzenetküldés (az e-levelezés, a fax, az SMS, a csevegés stb. kiváltására),
3. multimédia (zene lejátszása, mozgóképek, filmek, tv-adások stb. megjelenítése),
4. internet-hozzáférés (szörfölés a weben, a hangot és mozgóképet is tartalmazó oldalakat is ideértve).

A további szolgáltatások között előfordulhat a videokonferencia, a távoli jelenlét (telepresence), a csoportos játékok játszása és az m-kereskedelem (a fizetéshez majd csak meg kell lobogtatnunk a telefonunkat a bolt pénztáránál). Mindezeken túl az összes felsorolt szolgáltatás elvileg az egész világon azonnal (és bármikor) elérhető lesz (olyan helyeken, amelyeken nincs földi továbbítású hálózat, az eszközök automatikusan egy műholdas kapcsolatra váltanak), garantált szolgáltatásminőséggel.

Az ITU-nak azért célja, hogy az egész világ egy egységes IMT-2000 megoldást alkalmazzon, mert így a gyártóknak csak egyetlen eszközt kell építeniük, amelyet azután a világon bárhol eladhatnak, és amelyet a vásárlók is bárhol használni tudnak (hasonlóan a CD-lejátszókhoz és a számítógépekhez, de ellentétben a mobiltelefonokkal és a tévék-

kel). Az egységes műszaki megoldás a hálózatüzemeltetők életét is jelentősen megkönnyítené, és még több embert ösztönözne arra, hogy igénybe vegye a szolgáltatásaikat. Az üzletnek nem tesznek jót az olyan „formátumháborúk", mint az a harc, amelyet a Betamax vívott a VHS-sel az első videolejátszók megjelenésekor.

Ahogy utóbb kiderült, ez túl optimista elképzelés volt. A 2000-es szám 3 dolgot jelent: (1) az év, amikorra a bevezetést tervezték, (2) a tervezett működési frekvencia (MHz), valamint (3) a szolgáltatás által biztosítandó sávszélesség (kb/s). Ezek közül egyik sem valósult meg. 2000-re semmi sem valósult meg. Az ITU javasolta, hogy minden kormányzat tartsa fenn a 2 GHz-es frekvenciát, hogy az eszközök zökkenőmentesen barangolhassanak az országok között. De egyedül Kína tartotta fenn ezt a sávszélességet. Végül azt is felismerték, hogy a 2 Mb/s jelenleg nem valósítható meg azon felhasználók esetén, akik *túl gyorsan* változtatják a helyüket (amiatt, hogy az átadás nem végezhető el elég gyorsan). Sokkal reálisabb a 2 Mb/s a helyhez kötött felhasználók esetén (ami fej-fej mellett fog haladni az ADSL-lel), a 384 kb/s a gyalogos, illetve a 144 kb/s az autóban ülő felhasználók csatlakoztatásához.

A kezdeti sikertelenség óta több dolog is megvalósult. Számos IMT-javaslatot dolgoztak ki és némi rostálás után ezek közül a két legfontosabb javaslat maradt meg. Az első, a **WCDMA (Wideband Code Division Multiple Access – széles sávú kódosztásos többszörös hozzáférés)** az Ericsson javaslata volt, és az Európai Unió támogatta, amely **UMTS (Universal Mobile Telecommunications System – univerzális mobiltávközlési rendszer)** néven vált ismertté. A másik versenyző a **CDMA2000**, amely a Qualcomm javaslata volt.

A két rendszerben több a közös, mint a különbség, mivel mindkettő a széles sávú CDMA-ra épül. A WCDMA 5 MHz-es csatornákat, a CDMA2000 pedig 1,25 MHz-es csatornákat használ. Ha az Ericsson és a Qualcomm mérnökeit egy terembe összegyűjtenék, és megkérnék őket, hogy tervezzenek egy közös rendszerkialakítást, az valószínűleg menne is nekik. A baj az, hogy a valódi probléma nem mérnöki, hanem (szokás szerint) inkább politikai természetű. Európa olyan rendszert akart, amelyet a GSM-mel össze lehet kapcsolni, az Egyesült Államok pedig egy olyan rendszert, amely kompatibilis egy széleskörűen használt régi rendszerével (az IS-95-tel). Mindkét oldal a saját vállalatát támogatta (az Ericsson központja Svédországban van, a Qualcomm kaliforniai cég). Végül az Ericsson és a Qualcomm számos bírósági perbe bonyolódott a különböző CDMA-szabadalmak miatt.

Világszerte a mobil előfizetők 10-15%-a már a 3G-technológiát használja. Észak-Amerikában és Európában a mobil-előfizetők körülbelül 1/3-a használ 3G-t. Japán az elsők között vette át a technikát, és jelenleg Japánban majdnem az összes mobiltelefon 3G-s. Ezek a számok magukban foglalják az UMTS és CDMA2000 telepítését egyaránt, de továbbra is a 3G körül zajlanak az események, miközben a piac kibontakozik. A zavar növeléséhez az UMTS lett az egyetlen 3G-szabvány több inkompatibilis opcióval, a CDMA2000-et is beleértve. Ez a változás a különböző táborok egyesítésére tett erőfeszítés volt, ez azonban a műszaki eltéréseket csak tünetileg kezelte, amely elvonja a figyelmet a folyamatban lévő erőfeszítésekről. Az UMTS-t fogjuk használni a WCDMA-ra, a CDMA2000-től való megkülönböztetés érdekében.

Tárgyalásunk középpontjában a CDMA cellás hálózatokban történő alkalmazása lesz, mivel mindkét rendszert ez a képesség különbözteti meg a többitől. A CDMA sem

nem FDM, sem nem TDM, hanem a kettő kombinációja, amelyben minden felhasználó ugyanazon a sávszélességen küld egyszerre. Amikor a CDMA ötletét először felvetették, az ipar körülbelül ugyanúgy válaszolt rá, mint ahogyan Izabella királynő válaszolt Kolumbusznak, amikor azt javasolta, hogy az ellenkező irányba hajózva jussanak el Indiába. Mégis, egyetlen vállalat, a Qualcomm kitartásának köszönhetően a CDMA sikeres 2G-rendszer lett, és olyan szintre fejlődött, hogy ez képezi a 3G-s mobiltelefon-hálózatok alapját is.

Ahhoz, hogy a CDMA működjön a mobiltelefonoknál, az előző részben leírt alap CDMA-technikánál többre van szükség. Az előző részben leírtuk a szinkron CDMA-t, amelyben a töredéksorozatok ortogonálisak. Ez a kialakítás akkor működik, ha az összes felhasználó a töredéksorozat elejétől szinkronban van, amikor a bázisállomás elkezd adni a mobileszközök felé. A bázisállomás egyszerre tudja elkezdeni a töredéksorozatok adását, így a jelek ortogonálisak lesznek és elkülöníthetők. A független mobiltelefonok átvitelét azonban nehéz szinkronizálni. Nem megfelelő körültekintés esetén az adások különböző időpontokban érkeznek a bázisállomásra, az ortogonalitás garanciája nélkül. Ahhoz, hogy a mobileszközök szinkronizáció nélkül küldhessenek a bázisállomáshoz, a kódsorozatoknak minden eltolásnál ortogonálisnak kell lenniük, nem csak a kezdeti igazításnál.

Az általános esethez nem találhatók teljesen ortogonális sorozatok, a hosszú álvéletlensorozatok elég jó közelítést jelentenek. Ezek nagy valószínűséggel gyenge **keresztkorrelációban** vannak egymással az összes eltolásnál. Ez azt jelenti, hogy ha az egyik sorozatot összeszorozzuk a másikkal, majd összegezzük a skaláris szorzat kiszámításához, akkor az eredmény kicsi lesz, illetve 0, amennyiben a sorozatok ortogonálisak voltak. (A véletlensorozatoknak mindig különbözniük kell egymástól. Szorzatuknak véletlen jelet kell előállítaniuk, amelynek összege kis érték lesz.) Ez lehetővé teszi, hogy a vevő kiszűrje a nem kívánt adásokat a kapott jelből. Az ál-véletlensorozatok **autokorrelációja** nagy valószínűséggel szintén gyenge, a nulla eltolás kivételével. Ez azt jelenti, hogy ha egy sorozatot megszorzunk saját magának a késleltetett másolatával, majd ezeket összeadjuk, akkor az eredmény kicsi lesz, kivéve, ha a késleltetés 0. (A késleltetett véletlensorozat másik véletlensorozatnak tűnik, és visszatértünk a keresztkorreláció esetéhez). Ez lehetővé teszi, hogy a vevő a kapott jelben lévő kívánt adás elejéhez kapcsolódjon.

Az ál-véletlensorozatok használata lehetővé teszi, hogy a bázisállomás CDMA-üzeneteket kapjon nem szinkronizált mobileszközöktől. A CDMA-val kapcsolatban azonban eddig olyan implicit feltételezéssel éltünk, hogy a mobileszközök jelszintjei megegyeznek a vevőével. Ha ezek nem egyeznek meg, akkor gyenge keresztkorreláció az erős jellel elnyomhatja az erős autokorellációt a gyenge jellel. Ezért a mobileszközökön szabályozni kell az átviteli jelszintet a versengő jelek közötti interferencia minimalizálása érdekében. Ez az interferencia korlátozza a CDMA-rendszerek kapacitását.

A bázisállomásnál fogott jelek teljesítménye attól függ, hogy a különböző adók milyen távolságban vannak, illetve függ az átvitt jel teljesítményétől. A bázisállomástól különböző távolságban lehet számos hordozható állomás. Jó heurisztikus megoldás a kapott jelteljesítmény kiegyenlítéséhez, ha minden hordozható állomás olyan teljesítménnyel sugároz a bázisállomás felé, mint az attól fogott jelek teljesítményének inverze. Más szavakkal egy olyan állomás, amelyik gyenge jeleket fog a bázisállomástól, sokkal nagyobb teljesítménnyel sugároz, mint egy olyan, amelyik erős jeleket fog. A bázisállomás pedig akár határo-

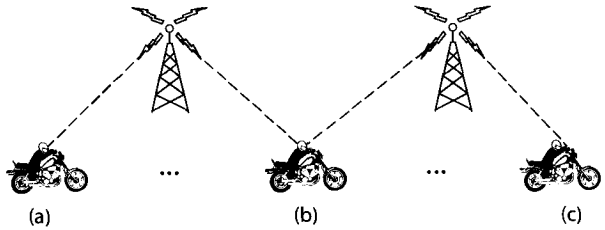
zott utasításokat is adhat a hordozható állomásoknak, hogy növeljék, csökkentsék vagy tartsák a sugárzott jelek teljesítményét. A visszajelzés gyakori (másodpercenként 1500), mivel a jó jelteljesítmény-szabályozás fontos az interferencia minimalizálása érdekében.

A korábban leírt alap CDMA-séma másik továbbfejlesztése, hogy lehetővé teszi a felhasználók számára, hogy különböző sebességgel küldjenek adatokat. A trükk természetesen a CDMA-ban van megvalósítva azáltal, hogy a töredékek (chips) küldési sebessége rögzített, és a felhasználók töredéksorozataihoz különböző hossz van rendelve. A WCDMA-ban például a töredéksebesség 3,84 Mtöredék/s és a kódok kiterjedése 4 és 256 töredék között változik. 256 hosszú töredékkód esetén körülbelül 12 kb/s marad hibajavítás után, és ez a kapacitás elegendő a hanghíváshoz. 4 hosszúságú töredékkód esetén a felhasználói adatsebesség közel van az 1 Mb/s-hoz. Közepes hosszúságú kódok használata közepes sebességet eredményez. Több Mb/s-os sebességhez a mobileszköznek több 5 MHz-es csatornát kell egyszerre használnia.

Feltesszük, hogy megbirkóztunk a bevezetési problémákkal, és most rátérünk a CDMA előnyeinek ismertetésére. Három fő előnye van. Az első, hogy a CDMA javítani tudja a kapacitást azzal, hogy képes kihasználni a kis periódusokat, amikor néhány adó nem ad. Intelligens hanghívások esetén az egyik fél csöndben van, miközben a másik beszél. Általában a vonal csak az idő 40%-ában foglalt. Azonban a szünetek rövidek lehetnek, és nem jósolhatók meg előre. TDM- vagy FDM-rendszerek esetén nem lehetséges újból időszeleteket vagy frekvenciacsatornákat elég gyorsan hozzárendelni, hogy ezek a kis csendes periódusok hatékonyan kihasználhatók legyenek. A CDMA-ban azonban azáltal, hogy egy felhasználó nem ad, csökken a másik felhasználókkal való interferencia lehetősége, és valószínű, hogy egy adott időpontban a felhasználók egy része nem ad egy elfoglalt cellában. Így a CDMA kihasználja a várt csendes periódusok előnyeit arra, hogy több egyidejű hívást tegyen lehetővé.

Második előny, hogy a CDMA esetén minden cella ugyanazt a frekvenciát használja. A GSM-mel és az AMPS-sel ellentétben az FDM-nél nem kell szétválasztani a különböző felhasználók adását. Ez kiküszöböli a bonyolult frekvenciatervezési feladatokat és javítja a kapacitást. Ezáltal a bázisállomás egyszerűen tud több **irányított** vagy **szektorantennát** használni a körsugárzó antenna helyett. Az irányított antennák a kívánt irányban figyelik a jelet, és más irányokban csökkentik a jelet, ezáltal az interferenciát is. Ez növeli a kapacitást. Három általános szektorkialakítás létezik. A bázisállomásnak nyomon kell követnie a mobileszközt, ahogy szektorról szektorra vándorol. Ez a nyomkövetés CDMA esetén egyszerű, mivel az összes szektor használja az összes frekvenciát.

A CDMA harmadik előnye, hogy elősegíti a **puha átadást** (**soft handoff**), amelynél az új bázisállomás már az előtt felveszi a kapcsolatot a mobileszközzel, mielőtt az előző bázisállomás kijelentkezne. Ezáltal a folytonosság nem szakad meg. A puha átadást a 2.49. ábra mutatja. Ezt CDMA-val egyszerű megvalósítani, mivel az összes cella használja az összes frekvenciát. Ennek alternatívája a **kemény átadás** (**hard handoff**), amikor a bázisállomás azelőtt dobja el a hívást, mielőtt az új bázisállomás felvenné azt. Ha az új bázisállomás nem képes összeköttetést létesíteni a telefonnal (például mert nincsen szabad frekvencia), akkor a hívás hirtelen megszakad. A felhasználók ezt általában észreveszik, de ennek ellenére a jelenlegi kialakítás mellett ez időnként elkerülhetetlen. Az FDM-kialakítások a kemény átadást használják azért, hogy kiküszöböljék a mobileszköz által két frekvencián történő egyidejű adás és vétel költségét.



2.49. ábra. *Puha átadás (a) előtt (b) közben és (c) után*

Sokat írtak már a 3G-ről. Többségük dicséri, hogy a szeletelt kenyér óta ez a legnagyobb felfedezés. Közben néhány szolgáltató óvatos lépéseket tesz a 3G irányába egy **2,5G**-nek nevezett rendszerrel, amit talán pontosabban 2,1G-nek kellene hívni. Az egyik ilyen rendszer az **EDGE** (**Enhanced Data rates for GSM Evolution – megnövelt adatsebességek a GSM evolúciójához**), amely pontosan ugyanolyan, mint a GSM, csak több bitet visz át egy szimbólumban. Sajnálatos módon azonban ha egy szimbólumban több bit van, akkor több hiba is jut egy szimbólumra, ezért az EDGE kilenc különböző eljárást használ a modulációra és a hibajavításra, amelyek abban különböznek egymástól, hogy mekkora sávszélességet kell elkülöníteni a megnövekedett sebesség miatt szükséges hibajavításra. Az EDGE egy lépés a fejlődés útján a GSM-től és a WCDMA felé. Ehhez hasonlóan, az operátorok számára is adott a fejlődési útvonal az IS-95-ről CDMA2000-re történő frissítéshez.

Bár a 3G-hálózatok fejlesztése még nem fejeződött be, egyesek már kész megoldásnak tekintik. Ezek a kutatók már a 4G-rendszeren dolgoznak **LTE** (**Long Term Evolution – hosszú távú fejlődés**) néven. A 4G ajánlott szolgáltatásai közül néhány: nagy sávszélesség, jelenlét mindenütt (csatlakozás bárhol), zökkenőmentes integráció más vezetékes és vezeték nélküli IP-hálózatokkal, a 802.11 hozzáférési pontokat is beleértve, adaptív erőforrás- és frekvenciamenedzsment, valamint kiváló minőségű szolgáltatás a multimédia számára. További információ Astely és mások [2009], valamint Larmo és mások [2009] munkáiban található.

Közben már rendelkezésre állnak vezeték nélküli hálózatok 4G szintű teljesítőképességgel. A legfontosabb példa a 802.16, amely WiMAX néven is ismeretes. A mobil-WiMAX áttekintését Ahmadi [2009] munkája tartalmazza. Ha azt mondjuk, hogy az iparág folyamatos változásban van, az túl enyhe kifejezés. Tekintsünk vissza az elmúlt néhány évre, hogy mi minden történt.

2.8. Kábeltelevízió

Mind a vezetékes, mind a vezeték nélküli telefonhálózatokat elég nagy részletességgel megtárgyaltuk már. Tisztán látható, hogy mindkettőnek jelentős szerepe lesz a jövő hálózataiban. Egy új főszereplő kezd azonban feltűnni a vezetékes hálózatok más lehetséges megvalósításainak színterén: a kábeltelevíziós hálózatok. Sokan már ma is kábelen kapják a telefon- és internetszolgáltatást, és a kábelhálózatok üzemeltetői nagy erőbedo-

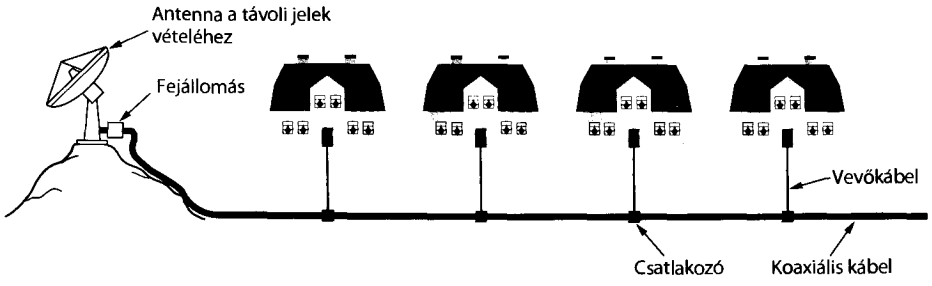
bással dolgoznak a piaci részesedésük növelésén. A következő szakaszokban a kábeltelevíziós rendszereket a hálózati alkalmazások szemszögéből fogjuk megvizsgálni, és összehasonlítjuk a most megtárgyalt telefonrendszerekkel. További információért lásd Donaldson és Jones [2001], Dutta-Roy [2001], továbbá Fellows és Jones [2001] művét.

2.8.1. Közösségi antennás televízió

A kábeltelevízió ötlete az 1940-es évek végén született meg arra a célra, hogy jobb vételt biztosítson a külvárosokban és a hegyek között élő embereknek. A rendszer eredetileg egy dombtetőn elhelyezett nagy antennából, egy erősítőből és egy koaxiális kábelből állt. Az antenna összegyűjtötte a tv-jeleket, amit a **fejállomásnak** (**headend**) nevezett erősítő felerősített, a koaxiális kábel pedig továbbított a házakhoz, a 2.50. ábrán is látható módon.

A korai években a kábeltelevíziót **közösségi antennás televíziónak** (**Community Antenna Television, CATV**) hívták, és akkoriban ez még nagyon családias üzletág volt. Bárki telepíthetett ilyen szolgáltatást a környékén, aki egy kicsit is értett az elektronikához, és a költségeket is fedezni tudta a többi beszálló felhasználó segítségével. Ahogyan az előfizetők száma egyre nőtt, további kábeleket illesztettek az eredeti kábelhez, és szükség szerint további erősítőket is telepítettek. Az átvitel egyirányú volt a fejállomástól a felhasználók felé. 1970-re már több ezer független rendszer működött.

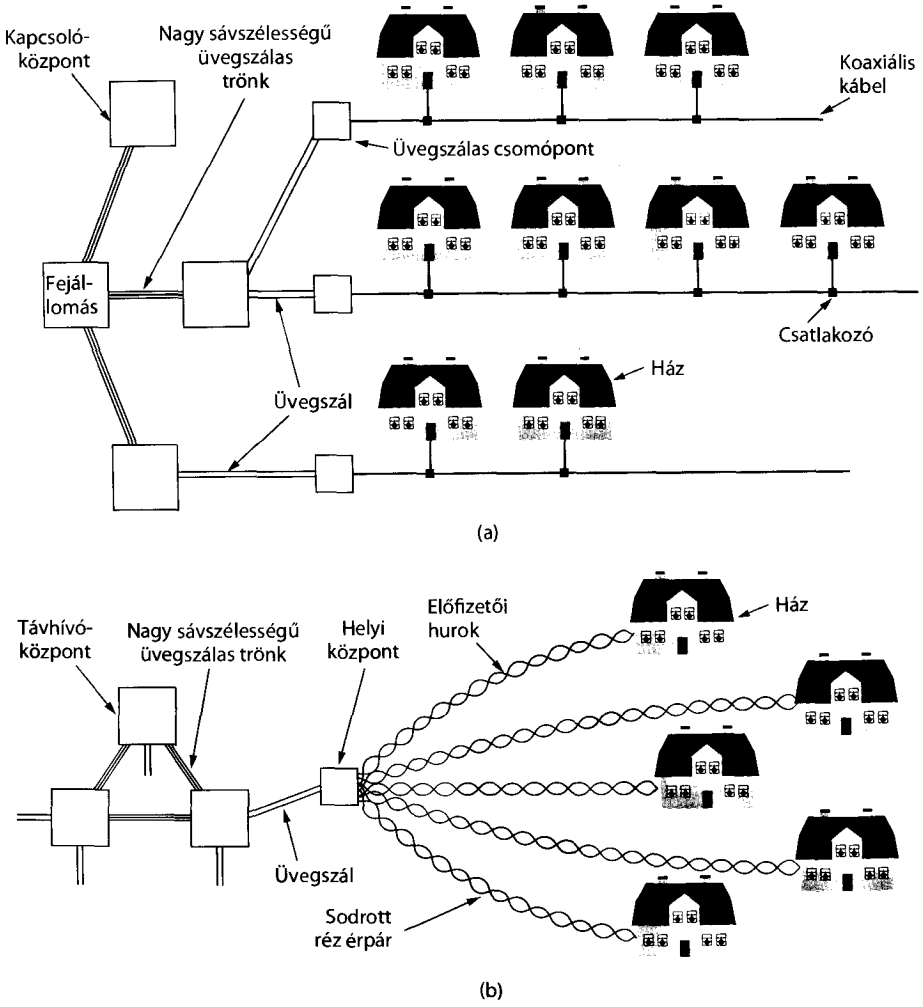
1974-ben a Time, Inc. új csatornát indított el, az HBO-t (Home Box Office – „házi mozi"), amely újfajta tartalmat (csak filmeket) kínált, és kizárólag kábelen terjesztették. További, kizárólag kábeles csatornák is követték, amelyek híreket, sportot, főzést és sok más témát kínáltak. Ez a fejlődés az iparban két változást eredményezett. Az első az, hogy a nagyvállalatok elkezdték felvásárolni a már működő kábelhálózatokat, és új kábeleket is lefektettek, hogy így szerezzenek több előfizetőt. Másodszor, felmerült az igény, hogy az új kábeles csatornák elosztásának elősegítésére olyan különböző rendszereket kapcsoljanak össze, amelyek gyakran távoli városokban voltak. A kábeles vállalatok az általuk kiszolgált városok között kábeleket kezdtek lefektetni, hogy egyetlen nagy rendszerré egyesíthessék azokat. Ez az eset hasonló ahhoz, ami a távközlési iparban történt 80 évvel korábban, amikor azért kötötték össze az addig izolált helyi központokat, hogy lehetővé tegyék a távolsági hívásokat.



2.50. ábra. *Egy korai kábeltelevíziós rendszer*

2.8.2. Internet a kábelhálózaton

Az évek múlásával a kábeltévérendszer egyre nőtt, és az egyes városok között futó kábeleket nagy sávszélességű üvegszálakra cserélték le, a telefonhálózat ezzel párhuzamos eseményeihez hasonlóan. Egy olyan rendszert, amely üvegszálakat alkalmaz a nagy távolságok áthidalására, és koaxiális kábeleket vezet a házakhoz, **HFC-** (**Hybrid Fiber Coax – üvegszál-koax hibrid**) rendszernek nevezünk. A rendszer fényvezető és villamos részei közötti csatolást megvalósító elektrooptikai átalakítóknak **üvegszálas csomópont** (**fiber node**) a neve. Egy fényvezető csomópont több koaxiális kábelt is táplálhat, mivel a üvegszál sávszélessége sokkal nagyobb a koaxénál. A 2.51.(a) ábrán egy modern HFC-rendszer egy részlete látható.



2.51. ábra. *(a) Kábel-tv. (b) A feljavított telefonhálózat*

Az elmúlt években sok kábelhálózat-üzemeltető cég döntött úgy, hogy beszáll az internetszolgáltatási üzletbe, sőt gyakran ezzel egyszerre a telefonszolgáltatási üzletbe is. A kábel- és a telefonhálózatok műszaki különbségei azonban arra is hatással vannak, hogy mi mindent kell megtenni a fenti célok eléréséhez. A legelső dolog az, hogy az összes egyirányú erősítőt kétirányú erősítőre kell cserélni a teljes rendszerben a feltöltés és letöltés támogatása érdekében. Amíg ez zajlott, a korai internet-hozzáférést biztosító kábeles rendszerek a kábeltelevíziós hálózatot használták a letöltéshez, és telefonhálózaton keresztüli betárcsázó összeköttetést a feltöltéshez. Ez egy intelligens kerülő megoldás volt, de sokkal gyengébb hálózat volt, mint amilyen lehetett volna.

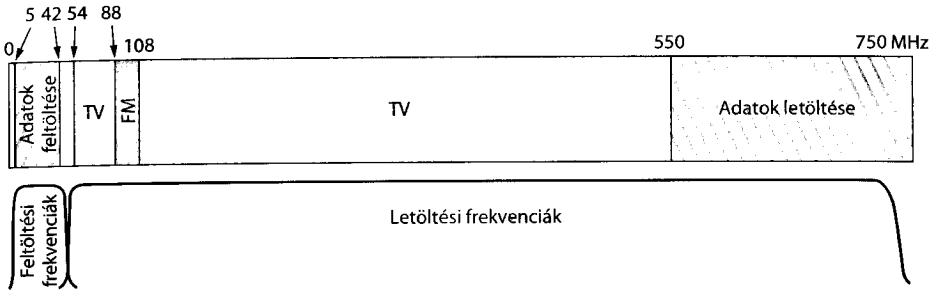
A 2.51.(a) ábrán látható HFC-rendszer és a 2.51.(b) ábrán látható telefonhálózat között azonban egy olyan további különbség is van, amelyet sokkal nehezebb megszüntetni. Az egyes lakókörzetekben egy tv-kábelt sok ház használ megosztva, míg a telefonhálózatban mindenki rendelkezik saját előfizetői hurokkal. Amikor a kábelrendszert tv-adások szórására használják, ez a megosztás nem játszik szerepet. Minden programot ugyanazon a kábelen szórnak, és teljesen mindegy, hogy 10-en vagy 10 000-en nézik az adást. Amikor a megosztott kábelt internet-hozzáférésre használják, sokat számít, hogy 10 felhasználó van vagy 10 000. Ha az egyik felhasználó úgy dönt, hogy letölt egy hatalmas állományt, akkor előfordulhat, hogy ezzel a többi felhasználótól veszi el a sávszélességet. Minél több a felhasználó, annál többen versenyeznek a sávszélességért. A telefonrendszernél nem találkozunk ezzel a tulajdonsággal: az ADSL-vonalakon egy nagy állomány letöltése nem csökkenti a szomszédok sávszélességét. Az érem másik oldala azonban az, hogy a koax sávszélessége jóval meghaladja a sodrott érpárokét, ezért szerencsés az, akinek a szomszédai nem használják túl sokat az internetet.

A kábeles szolgáltatók úgy orvosolták ezt a problémát, hogy több darabra osztották fel a hosszú kábeleket, és mindegyik szakaszt közvetlenül egy üvegszálas csomóponthoz kötöttek. A fejállomás és az üvegszálas csomópontok között a sávszélesség lényegében végtelen, így ha nincs túl sok felhasználó az egyes kábelszakaszokon, akkor a forgalom is kezelhető marad. Egy manapság tipikus kábelszakasz 500–2000 házat lát el, de ahogyan egyre több és több felhasználó fizet elő a kábeles internetszolgáltatásra, a forgalom túl naggyá válhat, így további felosztásra és még több üvegszálas csomópontra lesz szükség.

2.8.3. A spektrum kiosztása

Ha a szolgáltatók kihajítanák a tv-csatornákat, és kizárólag internet-hozzáférésre használnák a kábeles infrastruktúrát, az valószínűleg meglehetősen sok ideges előfizetőt jelentene nekik, így a kábelüzemeltetők kétkedéssel tekintenek erre a lehetőségre. Ezenkívül a legtöbb város igen szigorúan szabályozza, hogy mi mehet a kábeleken, így a kábelrendszerek üzemeltetői ezt akkor sem tehetnék meg, ha nagyon akarnák. Mindezek következtében meg kell oldaniuk azt, hogy a televízió és az internet megférjen egymás mellett ugyanazon a kábelen.

A megoldás a frekvenciaosztásos multiplexelés használatára épül. Az észak-amerikai kábeltévé-csatornák az 54 és 550 MHz közötti tartományt foglalják el (az FM-rádió 88 és 108 MHz közötti sávját kihagyva). Ezek a csatornák 6 MHz szélesek, amiben benne vannak a védősávok is, és egy hagyományos analóg televíziócsatornát vagy több digitális



2.52. ábra. *Egy internet-hozzáféréshez használt tipikus kábeltévérendszer frekvenciakiosztása*

televíziócsatornát tudnak átvinni. Európában a sáv alsó határa általában 65 MHz, és a csatornák 6–8 MHz szélesek a PAL és a SECAM által megkövetelt nagyobb felbontási képesség miatt, de a kiosztási elrendezés ettől eltekintve hasonló. A sáv legalsó részét nem használják. A modern kábelek már jóval 550 MHz fölött is képesek működni, gyakran 750 MHz-ig vagy még nagyobb frekvenciáig is. A megoldás az volt, hogy a feltöltési csatornáknak az 5–42 MHz-es (Európában valamivel nagyobb) sávot jelölték ki, és a spektrum felső végén levő frekvenciákat használják a letöltésekhez. A kábelek spektrumát a 2.52. ábra szemlélteti.

Mivel a televízió jelei mind lefelé haladnak, felfelé lehetséges olyan erősítőket alkalmazni, amelyek csak az 5–42 MHz-es tartományban működnek, lefelé pedig olyan erősítőket, amelyek csak az 54 MHz feletti frekvenciákon működnek, ahogyan ez az ábrán is látható. Ezzel a megoldással aszimmetrikussá tesszük a rendszer sávszélességét a két különböző irányban, mivel nagyobb frekvenciatartomány van a tv-csatornák felett, mint alattuk. Másrészt viszont a forgalom nagy része valószínűleg amúgy is lefelé haladna, így a kábeles szolgáltatókat ez a tény egyáltalán nem keseríti el. Amint azt már korábban láttuk, a telefontársaságok általában annak ellenére is aszimmetrikus DSL-szolgáltatást nyújtanak, hogy erre semmilyen műszaki okuk nincsen.

Az erősítők fejlesztésén kívül az üzemeltetőknek a fejállomást is fel kell fejleszteniük, buta erősítőből olyan intelligens digitális számítógéprendszerré, amely nagy sávszélességű üvegszálakkal csatlakozik egy ISP-hálózathoz. Gyakran a nevet is tovább „fejlesztik", és az új fejállomásokat inkább **CMTS**-nek (**Cable Modem Termination System – kábelmodem-véglezáró rendszer**) nevezik. A könyv hátralevő részében tartózkodni fogunk ettől a névtovábbfejlesztő tevékenységtől, és a hagyományos „fejállomás" szóhoz fogunk ragaszkodni.

2.8.4. Kábelmodemek

Az internet-hozzáféréshez egy kábelmodemre is szükség van. Ez olyan eszköz, amelyen két interfész található: egy a számítógép és egy a kábelhálózat felé. A kábeles hozzáférésű internet első éveiben minden hálózatüzemeltetőnek saját gyártmányú kábelmodemje volt, amelyet a kábeles cég technikusa telepített a felhasználónál. Ennek ellenére hamar nyilvánvalóvá vált, hogy egy nyílt szabvány versenyhelyzetet teremtene a kábelmode-

mek piacán, és ezzel lecsökkentené az árakat, vagyis ösztönözné a szolgáltatás terjedését. Mindezeken felül az, hogy a felhasználó boltban megvehetné a kábelmodemet, és saját maga telepíthetné (ahogyan a vezeték nélküli hozzáférési pontokat is), kiiktathatná a drága helyszíni kiszállásokat.

Mindezek következtében a nagyobb kábelszolgáltatók egy CableLabs nevű vállalkozásba tömörültek, hogy kidolgozzanak egy kábelmodemes szabványt, és hogy ellenőrizzék a kész termékek szabványosságát. Ez a szabvány, a **DOCSIS** (**Data Over Cable Service Interface Specification – kábelszolgáltatáson keresztül történő adattovábbító interfészek specifikációja**) mostanában kezdi leváltani az egyedi kialakítású modemeket. A DOCSIS 1.0 változat 1997-ben jött ki, és hamarosan követte a DOCSIS 2.0, 2001-ben. Ennek megnövekedett a feltöltési sebessége a szimmetrikus szolgáltatások – mint amilyen például az IP-telefon – jobb támogatása érdekében. A szabvány legújabb változata a DOCSIS 3.0, amely 2006-ban jelent meg. Ez nagyobb sávszélességet használ mindkét irányú sebesség növelése érdekében. Az európai változatot **EuroDOCSIS-nek** nevezték el. Nem minden kábelszolgáltatónak tetszik azonban a szabványos modemek megjelenése, mivel közülük sokan kerestek nagy pénzeket azzal, hogy a tehetetlen felhasználóiknak bérbe adták a modemeket. Egy nyílt szabvány és a több tucat gyártó, amelyeknek a kábelmodemjeit boltban meg lehet vásárolni, véget vet ennek a nagy hasznot hajtó tevékenységnek.

A modem és a számítógép közötti interfész teljesen magától értetődően adódik. Általában Ethernet, illetve egyes esetekben USB. A másik vég (a modem és a kábelhálózat közötti interfész) sokkal bonyolultabb, mivel egyaránt használ FDM-et, TDM-et és CDMA-t a kábel sávszélességének előfizetők közötti megosztásához.

Amikor a kábelmodemet csatlakoztatják az elektromos hálózathoz és áram alá kerül, pásztázni kezdi a letöltési csatornákat egy olyan különleges csomag után kutatva, amelyben a fejállomás időnként kiteszi a kábelre a rendszer paramétereit az újonnan bekapcsolt modemek részére. Miután megtalálta ezt a csomagot, az új modem az egyik feltöltési csatornán bejelenti a jelenlétét. A fejegység a válaszában kijelöli a modem feltöltési és letöltési csatornáit, ezen a kiosztáson azonban később változtatni is lehet, ha ezt a fejegység a terhelés kiegyenlítéséhez szükségesnek tartja.

A 6 MHz-es vagy 8 MHz-es csatornák használata az FDM része. Minden kábelmodem egy feltöltési és egy letöltési csatornán küld adatokat, vagy DOCSIS 3.0 esetén több csatornán. Az általános sémában a 6 (vagy 8) MHz-es letöltési csatornák modulálásra kerülnek QAM-64-gyel, vagy ha a kábelminőség kivételesen jó, akkor QAM-256-tal. 6 MHz-es csatornával és QAM-64-gyel körülbelül 36 Mb/s-ot kapunk. Ha levesszük a többletterhelést okozó fejrészeket, akkor a nettó felhasználói adatsebesség körülbelül 27 Mb/s, QAM-256 esetén pedig körülbelül 39 Mb/s. Az európai értékek ennél 1/3-dal nagyobbak.

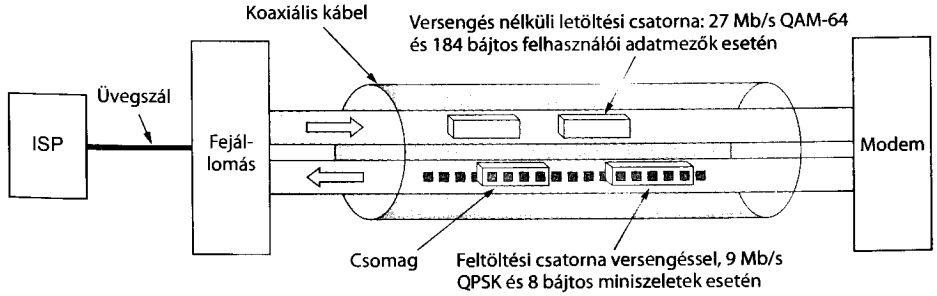
Feltöltés esetén több RF-zaj van, mivel a rendszert eredetileg nem adatokhoz tervezték, és a sok előfizetőtől származó zaj a fejállomásra koncentrálódik, így konzervatívabb séma kerül alkalmazásra. Ez a QPSK-tól a QAM-128-ig terjed, ahol a szimbólumok egy része hibajavításra kerül felhasználásra Trellis-kódmodulációval. Azáltal, hogy kevesebb bit van szimbólumonként a feltöltésben, a feltöltési és a letöltési sebesség közötti aszimmetria nagyobb, mint amit a 2.52. ábra sugall.

A TDM ezután több előfizető között megosztja a feltöltési sávszélességet. Ellenkező esetben az adásaik ütköznének a fejállomáson. Az idő felosztására kerül **miniszeletekre** (**minislot**), és a különböző felhasználók külön miniszeletben küldenek. Ahhoz, hogy ez működjön, a modem ezután megállapítja a fejegységtől való távolságát. Ehhez egy különleges csomagot küld el, és leméri, hogy mennyi idő múlva kap választ. Ezt a folyamatot **távolságbecslésnek** (**ranging**) hívják. A megfelelő időzítés miatt fontos, hogy a modem ismerje a távolságot a fejegységig. Minden felfelé haladó csomagnak bele kell férnie egy vagy több egymást követő miniszeletbe. A fejállomás rendszeresen bejelenti, amikor új miniszeletcsoport kezdődik, de ezt a bejelentést a kábelen való terjedési idő különbségei miatt a modemek nem egyszerre hallják. Mivel azonban minden modem ismeri a fejegységtől mért távolságát, ki tudja számolni, hogy mikor volt az első miniszelet tényleges kezdete. A miniszeletek hossza az egyes hálózatokon különböző. Egy tipikus miniszeletben 8 bájtnyi felhasználói adat található.

Az inicializálás folyamán a fejállomás minden modemhez hozzárendel egy olyan miniszeletet, amelyben a feltöltési sávszélesség-igényét bejelentheti. Amikor egy számítógép el akar küldeni egy csomagot, átadja a csomagot a modemnek, amely ezután a csomag továbbításához szükséges számú miniszeletet igényel. Amennyiben a fejállomás a kérést elfogadja, nyugtázó csomagot tesz a letöltési csatornára, amelyben megírja a modemnek, hogy mely miniszeleteket foglalta le a csomagjának. A modem ezután a kijelölt miniszeletekben elküldi a csomagot, és ha további csomagokat akar átvinni, a fejléc egyik mezőjében igényelhet további miniszeleteket.

Ugyanahhoz a miniszelethez több modem van rendelve, ami versenyhelyzethez vezet. Ez kétféleképpen kezelhető. Az első, hogy a CDMA megosztja a miniszeleteket az előfizetők között. Ez megoldja a versenyhelyzet problémáját, mivel az összes CDMA-kódsorozattal rendelkező előfizető egyszerre küldhet, ha csökkentett sebességgel is. A második lehetőség, hogy a CDMA nem kerül felhasználásra. Ebben az esetben a modem véletlen ideig vár, majd újból próbálkozik. Minden egymást követő kudarc után a véletlen várakozási idő lehetséges hossza megkétszereződik. (A hálózatokat már valamennyire ismerő olvasó felismerheti, hogy ez az algoritmus lényegében az időszeletelt ALOHA a kettes exponenciális visszalépés algoritmussal. Az Ethernetet azért nem lehet a kábelrendszereken használni, mert az állomások nem érzékelik a közeget. Ezekre a kérdésekre még visszatérünk a 4. fejezetben.)

A letöltési csatornákat a rendszer másképpen kezeli, mint a feltöltési csatornákat. Ennek egyik oka, hogy csak egy küldő van (a fejállomás), így nem alakulhat ki versenyhely-



2.53. ábra. *A feltöltési és letöltési csatornák tipikus részletei Észak-Amerikában*

zet és semmi szükség nincs a miniszeletekre, amelyek tulajdonképpen csak a statisztikus időosztásos multiplexelést valósítják meg. A másik ok az, hogy a lefelé haladó forgalom általában sokkal nagyobb a felfelé haladónál, ezért lefelé 204 bájtos rögzített méretű csomagokat használnak. Ennek egy része egy Reed–Solomon-hibajavító kód és némi egyéb többletteher, így a felhasználói adatok részére 184 bájt marad. Ezekre a számokra az MPEG-2 kódolású digitális televíziózással való kompatibilitás miatt esett a választás, hogy a tv- és a letöltési csatornák kialakítása azonos lehessen. Logikailag az összeköttetések a 2.53. ábrán látható módon épülnek fel.

2.8.5. A kábeles és az ADSL-összeköttetések összehasonlítása

Melyik jobb, az ADSL- vagy a kábeles kapcsolat? Ez olyan, mintha azt kérdeznénk, hogy melyik operációs rendszer, melyik nyelv vagy vallás a jobb. A válasz attól függ, hogy kinek tesszük fel a kérdést. Hasonlítsuk tehát össze az ADSL- és a kábeles kapcsolat néhány tulajdonságát! Az ADSL sodrott érpárat használ. A koax átviteli kapacitása több százszor jobb a sodrott érpárénál. A kábelrendszerekben viszont nem áll a teljes vonalkapacitás a felhasználó rendelkezésére, mert a kábel sávszélességének elég nagy részét televíziós programokra és más haszontalan dolgokra fordítják.

A gyakorlatban nehéz általánosítani a felhasználó számára rendelkezésre álló *kapacitással* kapcsolatban. Az ADSL-szolgáltatók határozottan megmondják, hogy mekkora sávszélességet adnak (például 1 Mb/s letöltésre, 256 kb/s feltöltésre), és a sebesség ennek általában a 80%-át folyamatosan el is éri. A kábeles szolgáltatók általában nem állítanak semmit a sávszélességről, mivel a tényleges kapacitás attól függ, hogy hány felhasználó tevékenykedik az adott kábelszakaszon. Néha jobb, mint az ADSL, néha viszont rosszabb is lehet. Ami azonban nagyon idegesítővé válhat, az a kiszámíthatatlansága. Az, hogy az egyik pillanatban remek minőségű szolgáltatást kapunk, nem garantálja azt, hogy a következő pillanatban is hasonlóan jó szolgáltatást fogunk kapni, mivel lehet, hogy a város legnagyobb sávszélesség-leszívója éppen most kapcsolta be a számítógépét.

Ahogyan egy ADSL-rendszer egyre több előfizetőt gyűjt, ezek egyre növekvő száma nem zavarja a már meglevő előfizetőket, mivel minden felhasználónak a többiektől független összeköttetése van. A kábelrendszerben az egyes felhasználók által tapasztalt teljesítmény egyre csökken, ahogyan egyre többen fizetnek elő az internetszolgáltatásra. Erre az egyetlen orvosság az, ha a kábeles szolgáltató több szakaszra bontja a forgalmas kábeleket, és mindegyiket közvetlenül egy fényvezető csomóponttal köti össze. Ez viszont időbe és pénzbe kerül, így az üzleti érdekük azt diktálja, hogy ne tegyék ezt.

Mellesleg egy másik olyan rendszerről is ejtettünk már szót, amely a kábelhez hasonló osztott csatornát használt: ez a mobiltelefon-hálózat. Ebben a rendszerben is a felhasználók egy csoportja (akiket jogosan nevezhetnénk cellatársaknak is) osztozik egy rögzített mennyiségű sávszélességen. Általában ezt TDM-mel és FDM-mel rögzített méretű darabokra osztják fel az aktív felhasználók között, mivel a beszédforgalom nagyon kiegyenlített. Az adatforgalom számára azonban ez a merev felosztás nagyon kevéssé hatékony, mivel az adatot forgalmazó felhasználók gyakran szakaszosan forgalmaznak, így a nekik lefoglalt sávszélesség kárba vész. Ugyanúgy, mint a kábeltévénél, dinamikusabb megoldást használnak a megosztott sávszélesség lefoglalására.

Az *elérhetőség* szempontjából az ADSL- és a kábelhálózat különbözik egymástól. Mindenkinek van telefonja, de nem minden felhasználó lakik olyan közel a helyi központhoz, hogy ADSL-t lehessen nála telepíteni. Másrészt viszont, nem mindenkinél van kábelhálózat, de ha van, és a szolgáltató internet-hozzáférést is biztosít, akkor bárkinél telepíthető. Az üvegszálas csomóponttól vagy a fejállomástól való távolsága nem számít. Azt is fontos megjegyezni, hogy a kábelhálózatoknak kevés céges előfizetőjük van, mivel eredetileg tv-műsorok szórására épültek.

Mivel az ADSL két pont közötti átviteli közegen valósul meg, természetéből fakadóan *biztonságosabb* a kábelhálózatnál. Egy kábelrendszer bármelyik felhasználója egyszerűen elolvashatja az összes csomagot, amely a kábelen halad. Emiatt minden jobb szolgáltató mindkét irányban titkosítja a teljes forgalmat. Mindazonáltal, ha a szomszéd megkapja a titkosított üzeneteinket, az még mindig kevésbé biztonságos, mintha semmit sem kapna meg.

A telefonhálózat általában megbízhatóbb a kábelhálózatnál. Például rendelkezik tartalék áramellátással, és így áramkimaradás esetén is működik. Ha a kábelrendszer láncában bármelyik erősítő tápellátása leáll, az összes letöltési irányban lévő felhasználó kapcsolata azonnal megszakad.

Végül, a legtöbb ADSL-szolgáltatónál *több internetszolgáltató közül választhatunk*, sőt egyes helyeken erre törvény is kötelez. A kábelszolgáltatóknál ez nem mindig van így.

A végkövetkeztetés az, hogy az ADSL- és a kábelszolgáltatás sokkal inkább hasonlítanak egymásra, mint amennyire különböznek egymástól. Összemérhető szolgáltatást nyújtanak, és ahogyan a kettő közötti verseny egyre jobban kiéleződik, az áraik is egyre jobban közelítenek egymáshoz.

2.9. Összefoglalás

A fizikai réteg minden hálózat alapja. A természet két olyan alapvető korlátozást kényszerít rá minden csatornára, amelyek meghatározzák a sávszélességét. Ez a két korlát a Nyquist-korlát, amely a zajmentes csatornákon érvényes, és a Shannon-korlát, amely a zajos csatornákon érvényes.

Az átviteli közegek lehetnek vezetékesek vagy vezeték nélküliek. A legfőbb vezetékes közegek a sodrott érpár, a koaxiális kábel és az üvegszál. A vezeték nélküli közegek között találjuk a földfelszíni rádiót, a mikrohullámokat, az infravörös fényt, a levegőben terjedő lézernyalábokat és a műholdakat.

A digitális modulációs eljárások biteket küldenek át a vezetékes vagy vezeték nélküli közegen analóg jel formájában. A vonali kódok alapsávon működnek és a jelek pedig áthelyezhetők egy áteresztősávba a vivőjel amplitúdójának, frekvenciájának és fázisának modulálásával. A csatornák megoszthatók a felhasználók között idő-, frekvencia- és kódosztásos multiplexeléssel.

A legtöbb nagy kiterjedésű hálózatban kulcsfontosságú elem a telefonhálózat. A fő alkotóelemei az előfizetői hurkok, a trönkök és a kapcsolók. Az ADSL 40 Mb/s-ig terjedő sebességeket kínál úgy, hogy az előfizetői hurkot sok, egymástól függetlenül modulált alvivőre osztja fel. Ez lényegesen meghaladja a telefonmodemek sebességét.

A trönkök digitális adatokat visznek át, amelyeket egyrészt WDM-mel multiplexelik a felhasználók közötti sok nagy kapacitású adatkapcsolat egyetlen üvegszálon keresztüli biztosításához, másrészt TDM-mel annak érdekében, hogy minden egyes nagy sebességű adatkapcsolatot megosszanak a felhasználók között. Mind a vonalkapcsolás, mind a csomagkapcsolás egyaránt fontos.

A vezetékes telefonhálózat nem alkalmas a mozgó alkalmazások kiszolgálására. A mobiltelefonokat mostanában széleskörűen használják a beszédtovábbításban, és hamarosan az adatszolgáltatások területén is elterjednek; már a harmadik generációnál tartanak. Az első generáció (1G) analóg volt, és alapja az AMPS volt. A 2G már digitális, a GSM jelenleg a legszélesebb körben telepített mobiltelefon-rendszer a világon. A 3G digitális és alapja a széles sávú CDMA, WCDMA-val és a CDMA2000-rel is mostanában telepítik.

A hálózatok elérésének egy másik lehetséges eszköze a kábeltelevíziós rendszer, amely fokozatosan alakult át koaxkábeles rendszerről hibrid üvegszálas-koax rendszerré, illetve televíziósról televíziósra és internetesre. A rendszer elméletileg nagyon nagy sávszélességet is biztosíthat, de a gyakorlatban a tényleges sávszélesség nagyban függ a felhasználóktól, mivel a sávszélesség köztük oszlik meg.

2.10. Feladatok

1. Adjuk meg az $f(t) = t$ függvény Fourier-együtthatóit $(0 \le t \le 1)$!

2. Egy 4 kHz-es zajmentes csatornát 1 ms-onként mintavételezünk. Mekkora a maximális adatsebesség? Hogyan változik a maximális sávszélesség, ha a csatorna zajos, 30 dB-es jel/zaj viszony mellett?

3. A televíziós csatornák 6 MHz sávszélességűek. Hány bit továbbítható rajtuk másodpercenként, ha négyszintű digitális jeleket használunk? Feltételezhetjük, hogy a csatornák zajmentesek.

4. Mekkora az elérhető maximális adatsebesség, ha bináris jeleket továbbítunk egy 3 kHz-es csatornán 20 dB jel/zaj viszony mellett?

5. Mekkora jel/zaj viszony szükséges ahhoz, hogy egy T1-vivőt egy 50 kHz-es vonalon továbbítsunk?

6. Melyek az üvegszálalapú átvitel előnyei a rézzel szemben, illetve van-e valamiféle hátránya a használatának?

7. Mekkora sávszélesség van egy 0,1 mikronos spektrumban, ha a hullámhossz 1 mikrométer?

8. A számítógép képernyőjéről üvegszálon keresztül szeretnénk képeket továbbítani. A képernyő mérete 2560 × 1600 pixel, és minden pixel 24 bites. Másodpercenként 60 képünk van. Mekkora sávszélesség kell ehhez, illetve hány mikrométeres hullámhossz szükséges ennél a sávnál 1,3 mikron esetén?

9. Igaz-e a Nyquist-tétel a kiváló minőségű, egymódusú üvegszálra is, vagy csak a rézvezetékre?

10. A rádióantennák vételi jellemzői akkor a legjobbak, ha az átmérőjük megegyezik a rádióhullámok hullámhosszával. A szokásos antennák átmérője 1 cm és 5 m közé esik. Milyen frekvenciatartománynak felel ez meg?

11. Egy 1 mm széles lézersugárral becélozunk egy 1 mm széles detektort, amely tőlünk 100 méterre, egy háztetőn van. Mekkora szögeltérése lehet (fokban) a lézersugárnak, hogy még eltalálja a detektort?

12. Az Iridium-rendszer 66 alacsonyröppályás műholdját hat láncra osztották el a Föld körül. A műholdak által használt magasságban a periódusidő 90 perc. Átlagosan mennyi idő telik el két átadás között azt feltéve, hogy az adó nem mozog?

13. Számoljuk ki egy csomag végpontok közötti átviteli idejét GEO (keringési magasság: 35 800 km), MEO (keringési magasság: 18 000 km) és LEO (keringési magasság: 750 km) műholdak esetében.

14. Mennyi a késleltetés az Iridium-rendszeren egy Északi-sarkról a Déli-sarkra indított hívás esetén 10 milliszekundumos műholdak közötti váltási idővel és 6371 km-es Föld sugárral számolva?

15. Mi a minimálisan szükséges sávszélesség a B b/s átvitel eléréséhez, ha az átvitelhez NRZ vagy MLT-3 vagy Manchester-kódolást használunk? Fejtse ki a válaszát!

16. Bizonyítsa be, hogy 4B/5B kódolást használva egy átmenet történik legalább 4 bitenként!

17. Hány végpontot lehetett megkülönböztetni az 1984 előtti időszakban, amikor minden egyes végpont háromjegyű régiókóddal és a szám első három jegyével volt azonosítva? A régiókód első jegye 2 és 9 közötti értékeket, a második 0-át vagy 1-et, míg az utolsó tetszőleges értéket tartalmazhatott. A szám első két jegye minden esetben 2 és 9 közötti, a harmadik tetszőleges értéket tartalmazott.

18. Egy egyszerű távbeszélőrendszer két helyi és egy olyan távhívóközpontból áll, amelyhez mindkét helyi központ egy 1 MHz-es duplex trönkön keresztül kapcsolódik. Egy átlagos telefonkészüléken egy 8 órás munkanapon 4 hívást bonyolítanak le. Egy hívás átlagosan 6 percig tart. A hívások 10%-a távhívás (tehát a távhívóközponton

keresztül zajlik le). Maximálisan hány telefont tud kezelni egy helyi központ, ha 4 kHz-es áramköröket feltételezünk?

19. Egy helyi telefontársaságnak 10 millió előfizetője van. Minden készüléke rézvezetékkel kapcsolódik a telefonközponthoz. A vezetékek átlagos hossza 10 km. Menynyit ér az előfizetői hurkokban található réz? Tegyük fel, hogy a vezetékek keresztmetszete kör, átmérője pedig 1 mm. A réz sűrűsége 9 g/cm^3, és kilogrammonként 3 dollárt adnak érte.

20. Egy olajvezeték szimplex, fél-duplex vagy duplex rendszer, esetleg ezek közül egyik sem?

21. A mikroprocesszorok ára olyan alacsonyra zuhant, hogy manapság már lehetséges minden egyes modembe beleépíteni egyet. Milyen hatással van ez a telefonvonal hibáinak kezelésére?

22. A 2.23. ábrán látható modemhez hasonlóan egy másik modem csillagképmintázatának adatpontjai a következők: (1, 1), (1, −1), (−1, 1), (−1, −1). Mekkora adatsebesség érhető el egy ilyen modemmel 1200 baud esetén?

23. Mennyi a maximálisan elérhető adatsebesség egy V.32-es szabványú modemmel 1200 baud esetén hibajavítás nélkül?

24. Hány különböző frekvenciát használ egy teljesen duplex QAM-64 modem?

25. Tíz, egyenként 4000 Hz-et igénylő jelet FDM-mel egyetlen csatornára multiplexelünk. Legalább mekkora sávszélesség szükséges a multiplexelt csatornán? Felteheti, hogy a védősávok 400 Hz szélesek.

26. Miért 125 μs a PCM-rendszer mintavételi periódusideje?

27. Hány százalékos rátartással rendelkezik egy T1-vivő; azaz mennyi jut valójában az 1,544 Mb/s-os adatsebességből a felhasználónak? Hogyan kapcsolódik ez a többletterhelési százalékokhoz az OC-1 vagy OC-768 vonalakon?

28. Hasonlítsuk össze annak a két 4 kHz-es zajmentes csatornának a maximális adatsebességét, ahol az egyik analóg kódolással mintánként 2 bitet továbbít, a másik pedig a T1 PCM-rendszerét használja!

29. Ha egy T1 vivőfrekvenciás rendszerben csúszás van, és a vevő kiesik a szinkronból, akkor a keretek első bitje segítségével megpróbál újraszinkronizálódni. Átlagosan hány keretet kell megvizsgálnia az újraszinkronizálódáshoz, ha a sikertelen újraszinkronizálódás valószínűsége 0,001?

30. Mi a különbség – ha van ilyen – egy *modem* demodulátor része és egy *kodek* kódoló része között? (Végül is mindkettő analóg jeleket alakít át digitális jelekké.)

31. A SONET órajelének pontossága kb. 10^{-9}. Mennyi idő alatt csúszik el az óra egy bitidőnyit? Mi következik a számításokból?

32. Mennyi az átviteli ideje egy 1 GB-os fájlnak vonalkapcsolt módon a 2.17. ábra szerint egy földi átjátszóval, felfelé 1 Mb/s, lefelé 7 Mb/s sávszélességgel és 1,2 másodperces kapcsolatfelépülési idővel számolva?

33. Adja meg az átviteli időt az előző problémához csomagkapcsolt átvitel esetén, ha a csomagok mérete 64 KB, a fejléc mérete 32 bájt, a kapcsolási késleltetés a műhold és az átjátszó között 10 mikroszekundum.

34. A 2.40. ábrán látható OC-3 felhasználói adatsebesség 148,608 Mb/s. Mutassuk meg, hogyan jön ki ez az érték a SONET OC-3 paramétereiből!

35. Az STS-1-nél kisebb adatsebességek támogatására a SONET fenntart egy virtuális becsatlakozói (virtual tributary, VT) rendszert. A VT egy olyan részleges adatcsomag, amelyet más részcsomagokkal együtt lehet egy STS-1 keretbe betenni, hogy így az adatok kitöltsék a keretet. A VT1.5 3 oszlopot, a VT2 4 oszlopot, a VT3 6 oszlopot és végül a VT6 12 oszlopot használ egy STS-1 keretből. Melyik VT alkalmas az alábbiak illesztésére?
(a) DS-1 szolgáltatás (1,544 Mb/s)
(b) Európai CEPT-1 szolgáltatás (2,048 Mb/s)
(c) DS-2 szolgáltatás (6,312 Mb/s)

36. Mekkora sávszélesség áll rendelkezésre egy OC-12c kapcsolat felhasználója számára?

37. Adott három n csomópontból álló csomagkapcsolt hálózat. Az első egy olyan csillagtopológiájú hálózat, amelyben van egy központi kapcsoló. A második hálózat egy (kétirányú) gyűrű, míg a harmadik egy olyan teljesen összekapcsolt hálózat, amelyben minden csomópont minden csomóponttal össze van kötve. Melyik a legjobb, az átlagos, illetve a legrosszabb átviteli útvonal az átlépések száma szempontjából?

38. Hasonlítsuk össze egy x bites, k darab csomóponton átjutott üzenet késleltetését egy vonalkapcsolt és egy (alig terhelt) csomagkapcsolt hálózatban! Ha a kapcsolatfelépítés ideje s másodperc, a terjedési idő csomópontonként d másodperc, a csomagméret p bit és az adatsebesség b b/s, akkor milyen feltételek esetén lesz a csomagkapcsolt hálózat késleltetése kisebb? Továbbá milyen feltételek mellett élvez előnyt a csomagkapcsolt hálózat a vonalkapcsolttal szemben?

39. Tegyük fel, hogy x bitnyi felhasználói adatot csomagok sorozataként továbbítunk egy csomagkapcsolt hálózatban úgy, hogy a csomagok k darab csomóponton mennek keresztül. Tegyük fel továbbá, hogy minden csomag p adatbitet és h fejrészbitet

tartalmaz, ahol $x \gg p+h$. Az adatvonalak átviteli sebessége b b/s, a vonali késleltetés pedig elhanyagolható. A p mely értékénél lesz a teljes késleltetés a legkisebb?

40. Egy tipikus, hatszögletű cellákkal dolgozó mobiltelefon-rendszerben tiltott a frekvenciák szomszédos cellákban való újrahasznosítása. Ha 840 frekvencia áll a rendelkezésünkre, mennyit használhatunk egy adott cellában?

41. A cellák tényleges elhelyezkedése csak ritkán olyan szabályos, mint a 2.45. ábrán látható celláké. Általában még az egyes cellák alakja is szabálytalan. Adjon egy lehetséges okot erre a szabálytalanságra!

42. Becsülje meg durván, hogy hány 100 m átmérőjű PCS-mikrocellára lenne szükség San Francisco lefedéséhez (120 km^2)!

43. Néha, amikor egy mobilfelhasználó átmegy az egyik cellából a másikba, hirtelen megszakad a folyamatban levő beszélgetése, annak ellenére, hogy minden adó és vevő tökéletesen működik. Miért?

44. Tegyük fel, hogy *A*, *B* és *C* párhuzamosan 0-s biteket küldenek egy, a 2.28.(a) ábrán látható töredéksorozatokat használó CDMA-rendszeren. Mi az eredő töredéksorozat?

45. Most nézzük egy másik szemszögből a CDMA-töredéksorozatok ortogonalitási tulajdonságát. Egy sorozatpárban minden bit vagy megegyezik, vagy nem. Fejezze ki az ortogonalitási tulajdonságot az egyezésekkel és az eltérésekkel!

46. Egy CDMA vevő a következő töredékeket veszi: (−1 +1 −3 +1 −1 −3 +1 +1). Feltéve, hogy a 2.28.(a) ábrán megadott töredéksorozatokat használjuk, mely állomások adtak, és milyen biteket küldtek az egyes állomások?

47. A 2.28. ábrán 4 adóállomáshoz tartozó töredéksorozat látható. Hogyan változik a sorozat négy újabb állomás hozzáadásával?

48. A legalacsonyabb szinten a telefonrendszer topológiája egy csillag, amelyben egy környékről minden előfizetői hurok egy helyi központhoz fut be. Ezzel ellentétben a kábeltévé-hálózat egyetlen hosszú kábelből áll, amely a környék összes háza mellett elmegy. Tegyük fel, hogy egy majdani tv-kábel nem rézkábel, hanem 10 Gb/s-os üvegszál lesz. Lehetne ezzel a kialakítással szimulálni a telefonhálózat fenti modelljét, amelyben mindenkinek saját kapcsolata van a helyi központ felé? Amennyiben igen, hány darab egytelefonos házat lehetne felfűzni egyetlen üvegszálra?

49 Egy kábeltévé-társaság azt határozza el, hogy internet-hozzáférést fog szolgáltatni a kábelrendszerén egy 5000 házból álló területen. A vállalat koaxiális kábelt használ, és úgy osztja ki a frekvenciasávokat, hogy 100 Mb/s-os letöltési sávszélesség jusson az egyes kábelekre. A vásárlók átcsábítására a cég úgy határoz, hogy legalább

2 Mb/s-os letöltési sávszélességet garantál minden háznak bármely pillanatban. Írja le, hogy mit kell tennie a vállalatnak ahhoz, hogy ezt garantálhassa!

50. Hány Mb/s-ot különít el a letöltési és a feltöltési forgalomnak egy, a 2.52. ábra sávszélesség-kiosztását felhasználó kábelrendszer? Válaszához a szövegben megadott információt is használja fel!

51. Milyen gyorsan tud egy kábelmodemes felhasználó adatokat fogadni, ha a hálózaton nincs egyéb forgalom a következő interfészekkel?
(a) 10 Mb/s Ethernet
(b) 100 Mb/s Ethernet
(c) 54 Mb/s Wireless

52. A többszörös STS-1 adatfolyamok (vagy becsatlakozók) multiplexelése fontos szerepet játszik a SONET-ben. Egy 3:1 arányú multiplexer a bemenő STS-1 becsatlakozókat egyetlen STS-3-as kimeneti folyamba multiplexeli. Ez a multiplexelés bájtról bájtra történik, vagyis az első három kimeneti bájt rendre az 1., 2. és 3. becsatlakozóé. A következő három kimeneti bájt rendre az 1., 2. és 3. becsatlakozó második bájtja és így tovább. Írjon egy programot a fenti multiplexer szimulálására! A programjának öt folyamatból kell állnia. A fő folyamat négy másik folyamatot hoz létre, egyet minden egyes STS-1 becsatlakozóhoz és egyet a multiplexerhez. Minden becsatlakozó folyamat egy STS-1 keretként 810 bájtot olvas be egy bemeneti állományból. A becsatlakozók a kereteiket bájtról bájtra küldik el a multiplexer-folyamatnak. A multiplexer-folyamat veszi ezeket a bájtokat, és egy STS-3 keretet ad ki a kimenetén (szintén bájtonként), aminek jelen esetben a standard outputra kell megérkeznie. A folyamatok közötti kommunikáció megvalósításához használjon csővezetékeket (pipe-ot)!

53. Írjon egy programot a CDMA megvalósításához. Tételezzük fel, hogy a töredéksorozat hossza 8, és az adóállomások száma négy. A program három folyamathalmazból áll: négy adófolyamat (t0, t1, t2 és t3), egy összekapcsoló folyamat, valamint négy vevőfolyamat (r0, r1, r2 és r3). A főprogram, amely összekapcsoló folyamatként is működik, először beolvassa a négy töredéksorozatot (bipoláris jelölés) a szabványos bemenetről, és a 4 bites sorozatot (1 átviendő bit adófolyamatonként), és elágazik négy pár adó- és vevőfolyamatra. Minden adó/vevőfolyamat pár (t0,r0; t1,r1; t2,r2; t3,r3) hozzárendelésre kerül egy töredéksorozathoz, és minden adófolyamathoz hozzárendelésre kerül 1 bit (az első bit a t0-hoz, a második bit a t1-hez és így tovább). Következőnek minden adófolyamat kiszámítja az átvinni kívánt jelet (8 bit sorozata) és elküldi azt az összekapcsoló folyamatnak. Miután mind a négy adófolyamattól megérkezett a jel, az összekapcsoló folyamat egyesíti a jeleket és elküldi az egyesített jelet a négy vevőfolyamatnak. Minden vevőfolyamat ezután kiszámítja az általa kapott bitet és kiírja azt a szabványos kimenetre. A folyamatok közötti kommunikáció megvalósításához használjon csővezetéket (pipe).

3. Az adatkapcsolati réteg

Ebben a fejezetben a 2. – adatkapcsolati – réteg tervezésének alapelveit fogjuk tanulmányozni. A vizsgálódás tárgya az lesz, hogy hogyan lehet megbízható, hatékony adategységek, ún. keretek átvitelére (tehát nem a fizikai rétegnél látott egyedülálló bitekre) alkalmas kommunikációt megvalósítani két szomszédos gép között. A szomszédosságon azt értjük, hogy a két gép fizikailag össze van kötve egy olyan kommunikációs csatornával, amely elméletileg vezetékként működik (például koaxiális kábel, telefonvonal vagy vezeték nélküli csatorna). Az alapvető tulajdonság, ami egy csatornát „vezetékszerűvé" tesz, az, hogy a rajta továbbított bitek a küldés sorrendjében érkeznek meg.

Elsőre azt gondolná az ember, hogy ez a probléma olyan egyszerű, hogy nincs is mit tanulmányozni – az *A* gép csak a vezetékre teszi a biteket, a *B* pedig veszi azokat. Sajnos, a kommunikációs áramkörök időnként hibáznak, ráadásul véges az adatátviteli sebességük, és nem nulla késleltetéssel továbbítják a biteket. Ezekből a korlátokból fontos következtetéseket lehet levonni az adatátvitel hatékonyságára vonatkozóan. Az alkalmazott kommunikációs protokolloknak figyelembe kell venniük az összes ilyen tényezőt. Ezek a protokollok képezik e fejezet tárgyát.

Az adatkapcsolati szinten előforduló fő tervezési szempontok megismerése után elkezdjük tanulmányozni a réteg protokolljait: megnézzük a hibák természetét, kialakulásuk okait és hogy hogyan lehet jelezni, illetve javítani őket. Ezután egy sor egyre növekvő bonyolultságú, több és több, az adatkapcsolati rétegben jelenlevő problémát megoldó protokollt vizsgálunk meg. A fejezetet végül az adatkapcsolati protokollokhoz tartozó példákkal zárjuk.

3.1. Az adatkapcsolati réteg tervezési szempontjai

Az adatkapcsolati réteg a fizikai réteg szolgáltatásait használja fel, hogy biteket küldjön át a csatornán.

A réteg számos feladat elvégzésére alkalmas, melyek közül néhány:

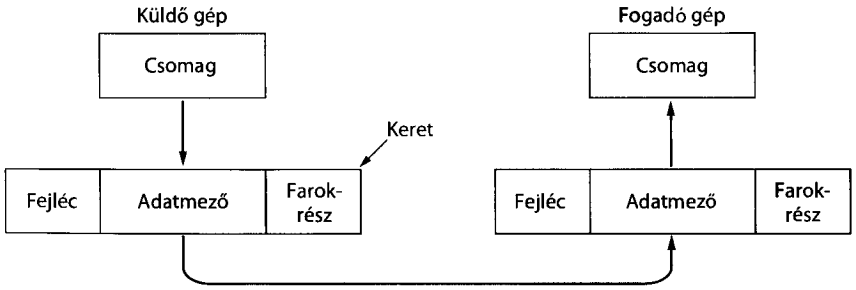
1. Jól definiált szolgáltatási interfész biztosítása a hálózati rétegnek.

2. Az átviteli hibák kezelése.

3. Az adatforgalom szabályozása, hogy a lassú vevőket ne árasszák el a gyors adók.

A fenti célok eléréséhez az adatkapcsolati réteg a hálózati rétegtől kapott csomagokat **keretekbe (frame)** ágyazza be az átvitel idejére. Minden keret tartalmaz egy keretfejrészt, egy adatmezőt a csomag számára, valamint egy keretfarokrészt. Ezeket a 3.1. ábra szemlélteti. A keretek kezelése az alapja mindannak, amit az adatkapcsolati réteg véghezvisz. A következő szakaszokban az imént említett feladatokat részletesen meg fogjuk vizsgálni.

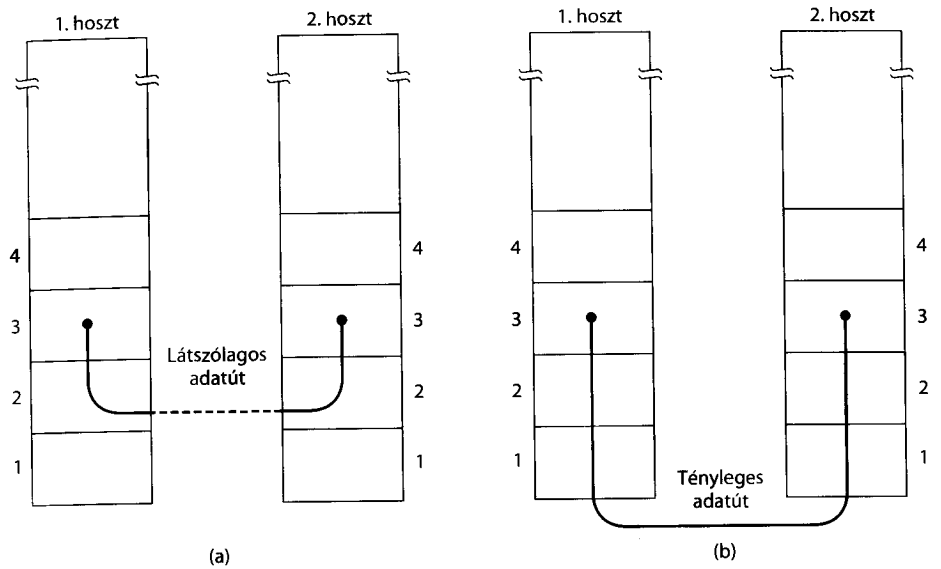
Annak ellenére, hogy ez a fejezet kifejezetten az adatkapcsolati rétegről és az adatkapcsolati protokollokról szól, sok itt megvizsgált alapelv a szállítási és más protokollokban is jelen van, hiszen a megbízhatóság átfogó cél, s mint ilyen, csak a rétegek együttműködésével valósítható meg. Ilyenek például a hibakezelés és a forgalomszabályozás. A gyakorlatban megvalósított hálózatokban ezek a feladatok sokszor csak a felsőbb rétegekben kapnak helyet, és kimaradnak az adatkapcsolati rétegből. Az alapelvek azonban teljes mértékben rétegfüggetlenek, és minden rétegben egyformán működnek, ezért nem igazán számít, hogy hol tanulmányozzuk őket. Ezek az elvek gyakran az adatkapcsolati rétegben érvényesülnek a legegyszerűbb és legtisztább formájukban, így talán ez a legalkalmasabb hely, ahol mindezeket részleteiben is megvizsgálhatjuk.



3.1. ábra. *A csomagok és a keretek közötti kapcsolat*

3.1.1. A hálózati rétegnek nyújtott szolgáltatások

Az adatkapcsolati réteg feladata az, hogy szolgáltatásokat nyújtson a hálózati rétegnek. A legfőbb szolgáltatás az adatok átvitele az adógép hálózati rétegétől a vevőgép hálózati rétegéig. Az adógépen van egy funkcionális egység – nevezzük folyamatnak – a hálózati rétegben, amely átad néhány bitet az adatkapcsolati rétegnek, hogy az továbbítsa a célhoz. Az adatkapcsolati réteg feladata az, hogy továbbítsa a biteket a címzett géphez, hogy ott azokat át lehessen adni a hálózati rétegnek, ahogyan a 3.2.(a) ábrán látható. A valódi átvitel a 3.2.(b) ábra szerint megy végbe, de egyszerűbb két adatkapcsolati rétegbeli folyamatot elképzelni, melyek adatkapcsolati protokollal kommunikálnak. Emiatt ebben a fejezetben hallgatólagosan a 3.2.(a) ábra szerinti modellt fogjuk használni.



3.2. ábra. *(a) Látszólagos (virtuális) kommunikáció. (b) Tényleges kommunikáció*

Az adatkapcsolati réteget különféle szolgáltatások megvalósítására készíthetik fel. A ténylegesen megvalósított szolgáltatások rendszerről rendszerre változhatnak. Három ésszerű, általánosan megvalósított lehetőség:

1. Nyugtázatlan összeköttetés nélküli szolgáltatás.

2. Nyugtázott összeköttetés nélküli szolgáltatás.

3. Nyugtázott összeköttetés-alapú szolgáltatás.

Vizsgáljuk meg ezeket sorjában!

Nyugtázatlan összeköttetés nélküli szolgáltatás esetén a forrásgép egymástól független kereteket küld a célgép felé, amely nem nyugtázza a keretek megérkezését. Az Ethernet jó példa ilyen típusú protokollra. Semmiféle kapcsolatot nem építenek fel előzetesen, illetve nem bontanak le az átvitel után. Ha egy keret a vonali zaj miatt elveszik, nem történik kísérlet a hiba felfedezésére és helyreállítására az adatkapcsolati rétegben. Ez a szolgáltatási osztály abban az esetben megfelelő, ha a hibaarány nagyon alacsony, így a hibák javítása a felsőbb rétegekre hagyható, valamint valós idejű forgalom esetén (például beszédátvitel), amikor a későn érkező adat rosszabb, mint a hibás adat.

A következő lépés a megbízhatóság irányába a nyugtázott összeköttetés nélküli szolgáltatás. Ilyen szolgáltatás esetén sincs felépített kapcsolat, de minden egyes elküldött keret megérkezését nyugtázza a címzett állomás, így a küldő értesül arról, hogy a keret megérkezett-e, vagy sem. Ha egy keret nem érkezik meg meghatározott időn belül, újra lehet küldeni. Ez a szolgáltatás megbízhatatlan csatornák (például vezeték nélküli rendszerek) esetén hasznos. A 802.11 Wi-Fi is ilyen típusú szolgáltatást alkalmaz.

Talán érdemes hangsúlyozni, hogy a nyugtázás megvalósítása az adatkapcsolati rétegben sohasem elvárás, csak optimalizáció. A hálózati réteg mindig küldhet egy csomagot és megvárhatja a távoli partner által küldött nyugtát. Ha a nyugta az időzítő lejárta előtt nem érkezik meg, a küldő újraküldheti az egész üzenetet. A probléma ezzel a stratégiával az, hogy nem hatékony. Az adatkapcsolatok rendszerint hardveres okokból általában szigorúan korlátolt hosszúságú keretekkel és ismert terjedési késleltetéssel rendelkeznek. Ezeket a paramétereket a hálózati réteg nem ismeri. Ha például küldünk egy nagy csomagot, amely 10 keretből áll, és ezek közül átlagosan kettő elvész, nagyon sokáig tarthat, amíg a csomag hibátlanul megérkezik. Ha minden keretet egyenként nyugtázunk, és szükség esetén újraküldünk, sokkal gyorsabban jut át az egész üzenet. Megbízható csatornákon, mint például az üvegszál, szükségtelen bonyolult adatkapcsolati protokoll használata, de vezeték nélküli átviteli csatornákon a csatorna megbízhatatlansága miatt nagyon is megéri.

Visszatérve a szolgáltatásokhoz, a legkifinomultabb szolgáltatás, amit az adatkapcsolati réteg a hálózati rétegnek nyújthat: az összeköttetés-alapú szolgáltatás. Ezt alkalmazva a forrás- és a címzett számítógép felépít egy összeköttetést, mielőtt az adatátvitelt megkezdenék. Minden elküldött keret sorszámozott, és az adatkapcsolati réteg garantálja, hogy a keretek valóban meg is érkezzenek, továbbá, hogy minden keret pontosan egyszer és a megfelelő sorrendben érkezzen meg. Az összeköttetés-alapú szolgáltatás így megbízható bitfolyamot biztosít a hálózati réteg folyamatai számára. Ez a fajta szolgáltatás kifejezetten előnyös olyan megbízhatatlan csatornákon, mint a műholdas összeköttetések vagy nagy távolságú telefonvezetékek. Ilyen esetekben, ha nyugtázott összeköttetés nélküli szolgáltatást használnánk, azzal járna, hogy az elveszett nyugták miatt a kereteket többször újra kellene küldeni és venni, ami jelentős sávszélesség-veszteséget okozna.

Amikor összeköttetés-alapú szolgáltatást alkalmazunk, az átvitel három jól elkülöníthető fázisra bontható. Az első fázisban az összeköttetés felépül; mindkét oldalon inicializálódnak azok a változók és számlálók, melyek ahhoz szükségesek, hogy számon tarthassuk, hogy mely keretek érkeztek meg és melyek nem. A második fázisban egy vagy több keret tényleges továbbítása történik. A harmadik – egyben utolsó – fázisban az összeköttetést lebontjuk, felszabadítva a változókat, puffereket és egyéb erőforrásokat, melyeket a kapcsolat karbantartásához használtunk.

3.1.2. Keretezés

Ahhoz, hogy az adatkapcsolati réteg szolgáltatást nyújthasson a hálózati rétegnek, a fizikai réteg szolgáltatását kell igénybe vennie. A fizikai réteg nem tesz mást, mint a kapott bitsorozatot megpróbálja továbbítani a célhoz. Ha a csatorna zajos – ahogy a legtöbb vezeték nélküli és néhány vezetékes adatkapcsolat is ilyen – a fizikai réteg, hogy a hibák számát csökkentse, redundanciát ad a kimenő jelekhez, de az érkező bitsorozat hibamentességét a fizikai réteg nem garantálja. A vett bitek száma lehet kevesebb, azonos vagy több, mint az elküldötteké, és a bitek értéke is különbözhet az eredetitől. Az adatkapcsolati réteg feladata, hogy jelezze, illetve – ha szükséges – kijavítsa a hibákat.

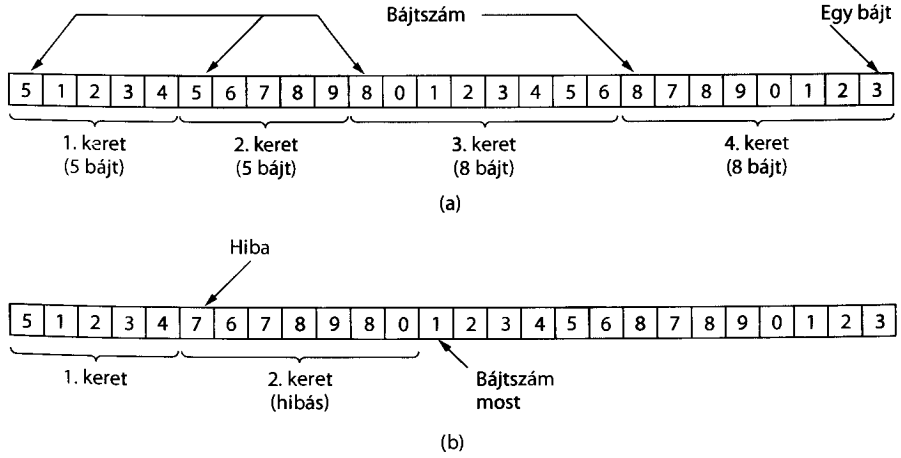
A szokásos megoldás az, hogy az adatkapcsolati réteg különálló keretekre tördeli a bitfolyamot, és a keretekhez ún. **ellenőrző összeget** számít. (Az ellenőrzőösszeg-képző algoritmusokat a fejezet későbbi részében tárgyaljuk.) Amikor a keret megérkezik a célhoz, az ellenőrző összeget újra kiszámolja az adatkapcsolati réteg. Ha ez különbözik attól, amit a keret tartalmaz, a réteg tudja, hogy hiba történt, és lépéseket tesz ennek kezelésére (például eldobja a rossz keretet és hibajelzést küld vissza).

A bitfolyam keretekre tördelése sokkal bonyolultabb feladat, mint amilyennek első ránézésre tűnik. Egy jó megoldásnak lehetővé kell tenni a keretek kezdetének könnyű felismerését elenyésző csatorna-sávszélesség használata mellett. Ebben a szakaszban négy módszert vizsgálunk meg:

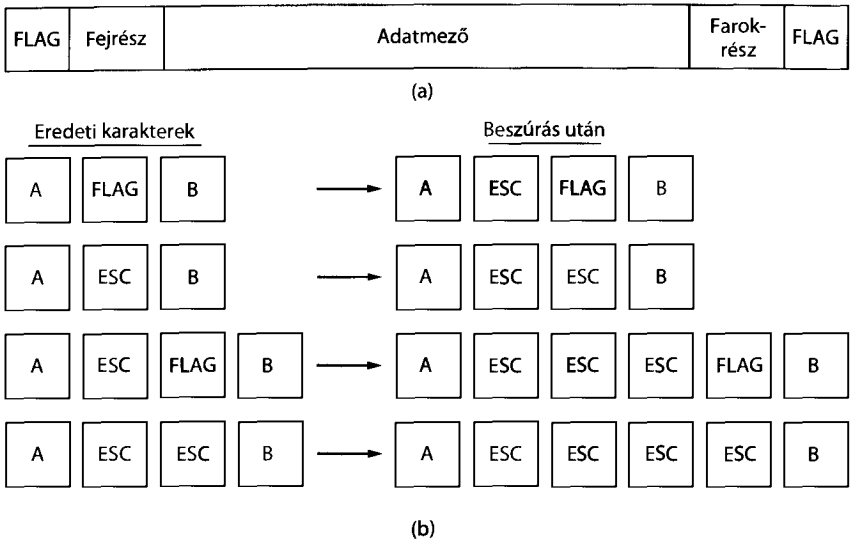
1. Bájtszámlálás.
2. Kezdő- és végkarakterek használata bájtbeszúrással.
3. Kezdő- és végjelek használata bitbeszúrással.
4. Fizikai rétegbeli kódolás megsértése.

Az első keretezési módszer a keretben levő bájtok számának megadására egy, a keret fejrészében levő **bájtszámmezőt** használ. Amikor a címzett állomás adatkapcsolati rétege megkapja a keretben levő bájtok számát, tudni fogja, hogy mennyi bájtnak kell érkeznie, és így azt is, hogy hol van a keret vége. Ennek a technikának az alkalmazása a 3.3.(a) ábrán látható négy keretre, melyek mérete 5, 5, 8 és 8 bájt.

Ezzel az algoritmussal az a baj, hogy egy átviteli hiba elronthatja a bájtszámmezőt. Például, ha egy 5-ös bájtszám a 3.3.(b) ábra második keretében 7-té válik egyetlen bithiba eredményeképpen, a célállomás kiesik a szinkronból, és képtelen lesz megtalálni a következő keret elejét. Még ha az ellenőrző összeg hibás is, és a címzett tudja, hogy a keret rossz, akkor sincs mód arra, hogy megmondjuk, hol kezdődik a következő keret. Szintén nem segít a keret újraküldésének kérése, hiszen a címzett nem tudja, hogy hány



3.3. ábra. *Egy bájtfolyam (a) hiba nélkül, (b) egy hibával*



3.4. ábra. *(a) Egy jelzőbájtokkal határolt keret. (b) Négy bájtsorozat bájtbeszúrás előtt és után*

bájtot kell átlépnie ahhoz, hogy az újraküldött keret elejéhez érjen. Az előzőek miatt a bájtszámlálásos módszert ma már ritkán használják.

A második keretezési eljárás úgy kerüli meg a hibák utáni újraszinkronizálás problémáját, hogy minden keret elejét és végét egy-egy különleges bájttal jelzi. Régebben a keretek elejét és végét különböző bájtok jelezték, de az elmúlt években a legtöbb protokoll már ugyanazt a bájtot használja erre a célra. Ezt **jelzőbájtnak** (**flag byte**) nevezték el, a 3.4.(a) ábrán FLAG-ként tüntettük fel. Két egymást követő jelzőbájt közül az egyik a keret végét, a másik a következő keret elejét jelzi. Ha a vevő bármikor kiesik a szinkronból, akkor csak a jelzőbájtot kell megkeresnie ahhoz, hogy megtalálja az éppen futó keret végét.

Ennél a módszernél komoly gond jelentkezik, amikor bináris adatokat, például képeket vagy zenét kell átvinnünk. Könnyen előfordulhat, hogy a jelzőbájt bitmintája megjelenik az adatok között is, ez pedig általában belezavar a keretezésbe. A gond orvoslásának egyik módja az, hogy a küldő fél adatkapcsolati rétege egy különleges *kivételbájtot* (*escape byte, ESC*) helyez minden olyan jelzőbájt elé, amely „véletlenül" került az adatmezőbe, így minden jelzőbájt megkülönböztethető az adatokban előforduló azonos értékű bájtoktól az előtte álló kivételbájt (ESC) megléte, vagy hiánya alapján. A vevő oldal adatkapcsolati rétege eltávolítja ezt a kivétel bájtot, mielőtt az adatokat a hálózati rétegnek továbbítaná. Ezt a módszert **bájtbeszúrásnak** (**byte stuffing**) nevezik.

A következő kérdés természetesen az, hogy mi történik abban az esetben, amikor a felhasználó adatai egy kivételbájtot tartalmaznak. A válasz az, hogy ezt is megjelölik egy kivételbájttal. Így minden egyedüli kivételbájt egy *kivételbájt-sorozat* (*escape sequence*) része, ahol minden kettős karakter azt jelenti, hogy egy kivételbájt volt a felhasználói adatok között. A 3.4.(b) ábra ezt az eljárást mutatja be néhány bájtsorozaton. A beszúrt bájtok eltávolítása után leszállított bájtsorozat minden esetben pontosan megegyezik

az eredeti bájtsorozattal. A kerethatárok a kivételbájtok eltávolítása nélkül továbbra is megtalálhatók két, egymás melletti jelzőbájt keresésével.

A 3.4. ábrán feltüntetett bájtbeszúrási módszer egy enyhén leegyszerűsített változata annak, amelyet a **PPP-protokoll** (**Point-to-Point Protocol – kétpontos protokoll**) használ. A PPP az a protokoll, amelyet a legtöbb otthoni számítógép az internetszolgáltatójával való kommunikációhoz használ, és később még lesz róla szó ebben a fejezetben.

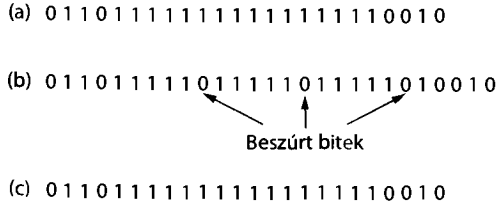
A harmadik módszer túllép a bájtbeszúrás azon hátrányán, hogy az tipikusan 8 bites bájtokat használ. Ez az új módszer lehetővé teszi, hogy tetszőleges számú bit legyen egy keretben, és az alkalmazott karakterkódok is tetszőleges számú bitet tartalmazzanak. A módszert az egykor nagyon népszerű **HDLC** (**Highlevel Data Link Control – magas szintű adatkapcsolati vezérlés**) protokollhoz fejlesztették ki. A módszer az alábbiak szerint működik: minden keret egy speciális, **jelző-** (**flag**) **bájtnak** nevezett bitmintával kezdődik. Ez a 01111110 (hexadecimálisan 0x7E). Amikor az adó adatkapcsolati rétege öt egymást követő 1-es bitet talál az adatok között, automatikusan beszúr egy 0-t a kimenő bitfolyamba. Ez a **bitbeszúrás** (**bit stuffing**) analóg a bájtbeszúrással, amelyben egy ESC-t szúrtunk be a kimenő bájtfolyamba az adatok között levő jelzőbájt elé. Mivel a módszer a legkevesebb átmenettel jár, ezért a fizikai réteg könnyebben szinkronban marad. Éppen ezért alkalmazzák e módszert az USB- (Universal Serial Bus – univerzális soros sin) szabványban.

Amikor a vevő öt egymást követő 1-es bitet talál, melyet egy 0-s követ, automatikusan törli a 0-s bitet. Ahogyan a karakterbeszúrás teljesen átlátszó a hálózati réteg számára mindkét számítógépben, a bitbeszúrás is az. Ha a felhasználói adat tartalmazza a jelzőbájt bitmintáját (01111110), ez 011111010-ként továbbítódik, de a vevő memóriájában már 01111110 jelenik meg. A 3.5. ábra egy példát mutat a bitbeszúrásra.

A bitbeszúrásos módszer segítségével egyértelműen felismerhetők a kerethatárok: ha a vevő szem elől téveszti a határokat, semmi mást nem kell tennie, mint a bemeneti bitfolyamban a jelző mintát keresnie, hiszen a minta csak kerethatárokon fordulhat elő, az adatok között sohasem.

Mind a bájtbeszúrásnál, mind a bitbeszúrásnál megjelenik az a mellékhatás, hogy a keret hossza a szállított adatok tartalmától függ. Például ha az adatmező nem tartalmaz jelzőbájtot, 100 bájtnyi adat nagyjából egy 100 bájtos keretben átvihető, míg egy kizárólag jelzőbájtokból álló adatmező a kivételbájtokkal körülbelül 200 bájtos keretben továbbítható. Ebből a szempontból a bitbeszúrás kedvezőbb, mert a keretméret növekedése nagyjából 12,5%, hiszen minden bájthoz egy bitet adtunk hozzá.

Az utolsó keretezési eljárás a fizikai réteg kódolásainak tulajdonságán alapszik. A 2. fejezetben láttuk, hogy a bitek fizikai jelekké kódolása gyakran redundanciát tar-



3.5. ábra. *Bitbeszúrás. (a) Az eredeti adat. (b) Az átviteli vonalon megjelenő adat. (c) A vevő memóriájában megjelenő, a beszúrt bitek törlése utáni adat*

talmaz, amely azt jelenti, hogy bizonyos jelek nem fordulhatnak elő hagyományos adat kódolása esetén. Például a 4B/5B séma 4 adatbitet kódol 5 jellel a megfelelő mennyiségű jelátmenet előállítása érdekében. Ez azt jelenti, hogy a lehetséges 32 jelből 16-ot nem használnak, így ezek közül néhány fenntartott jel felhasználható a keret elejének és végének jelzésére. Gyakorlatilag tehát „kódsértést" használtunk keretezésre. A módszer szépsége, hogy mivel ezek a jelek nem használatosak kódolás esetén, könnyű őket megtalálni, és nincs szükség az adatmező módosítására, bájtok vagy bitek beszúrására.

Végül megjegyezzük, hogy sok adatkapcsolati protokoll a nagyobb biztonság érdekében a fenti módszerek valamely kombinációját alkalmazza. Egy gyakori megoldásra láthatunk példát az Ethernet és a 802.11 protokollokban, ahol a keret egy jól definiált **előtaggal (preamble)** kezdődik. Az előtag megfelelően hosszú is lehet (802.11 esetén 72 bit), hogy segítse a vevőt felkészülni az adatok fogadására. Az előtagot egy hosszkód- (azaz bájtszám-) mező követi a fejlécben, melyet a keret végének meghatározására használnak.

3.1.3. Hibakezelés

Miután megoldottuk a keretek kezdetének és végének jelzését, szembekerülünk a következő problémával: hogyan bizonyosodjunk meg arról, hogy minden keret ténylegesen megérkezik-e a címzett állomás hálózati rétegéhez, és hogy helyes sorrendben érkezik-e meg. Egyelőre tételezzük fel, hogy a vevő meg tudja állapítani, hogy a beérkezett keret helyes vagy hibás adatokat tartalmaz (a 3.2. fejezetben majd áttekintjük a hiba detektálására és javítására használatos kódolásokat). Nyugtázatlan összeköttetés nélküli szolgáltatáshoz megfelelő lehet, ha a küldő folyamatosan küldi a kereteket, függetlenül attól, hogy azok helyesen érkeztek-e meg, viszont megbízható, összeköttetés-alapú szolgáltatásoknál ez már nem elég.

A biztonságos átvitel megvalósításának általános módja az, hogy az adónak valamilyen visszacsatolást biztosítunk arról, hogy mi történik a vonal másik végén. Tipikusan az adó megköveteli a vevőtől, hogy speciális vezérlőkereteket küldjön vissza, melyek pozitív vagy negatív nyugtát hordoznak a bejövő keretekről. Ha a küldő pozitív nyugtát kap egy keretről, tudja, hogy a keret rendben megérkezett. A negatív nyugta ellenben azt jelenti, hogy valami nincs rendben, és a keretet újra kell adni.

További komplikáció származik abból, hogy hardverhibák (például zaj) egy keret teljes eltűnését okozhatják. Ebben az esetben az adó egyáltalán nem reagál, mivel nincs is mire reagálnia. Hasonlóan, ha a nyugtázó keret veszik el, a küldő nem tudja folytatni tevékenységét. Tisztán látszik, hogy az a protokoll, amelyben a küldő a keret elküldése után vár a pozitív vagy negatív nyugtára, örökre felfüggesztődne, ha egy keret egyszer teljesen elveszne a hibásan működő hardver miatt.

Ezt a lehetőséget az adatkapcsolati rétegben időzítők bevezetésével küszöbölik ki. Amikor az adó továbbít egy keretet, általában egy időzítőt is elindít. Az időzítő úgy van beállítva, hogy lejártáig legyen elég idő arra, hogy a keret elérje a célt, ott feldolgozásra kerüljön, és a nyugta visszatérjen az adóhoz. Normális esetben a keret helyesen megérkezik, és a nyugta visszaér, mielőtt az időzítő lejárna. Ekkor az időzítő törlődik.

Ha viszont a keret vagy a nyugta elvész, az időzítő lejár, és jelzi az adónak, hogy valószínűleg hiba történt. A nyilvánvaló megoldás: egyszerűen újra elküldeni a keretet.

Ha azonban a kereteket többször továbbítjuk, fennáll a veszélye annak, hogy a vevő többször veszi ugyanazt a keretet, és többször adja át a hálózati rétegnek. Hogy ezt megakadályozzuk, általában a kimenő kereteknek sorszámot adunk, hogy a vevő meg tudja különböztetni az újraadott kereteket az eredetiektől.

Az adatkapcsolati réteg feladatának fontos része az, hogy az időzítőket és a számlálókat úgy kezelje, hogy biztosítani tudja a keretek pontosan egyszeri (nem több és nem kevesebb) megérkezését a címzett állomás hálózati rétegéhez. E fejezet egy későbbi részében részletesen tanulmányozni fogjuk egy sor, egyre bonyolultabb példán keresztül, hogy hogyan lehet ezt megvalósítani.

3.1.4. Forgalomszabályozás

Egy másik fontos tervezési kérdés, amely megjelenik az adatkapcsolati rétegben (és a felsőbb rétegekben is) az, hogy mit tegyünk azzal az állomással, amelyik rendszeresen gyorsabban akarja adni a kereteket, mint ahogy a vevő azokat fogadni tudná. Ez a szituáció könnyen előállhat akkor, ha az adó egy gyors (vagy kevéssé terhelt) számítógép, a vevő pedig egy lassú (vagy erősen leterhelt) gép. Gyakori helyzet például, hogy egy okostelefon weboldalt kér le egy gyors szervertől, amely teljes sebességgel kiszolgálja a kérést, és a szerencsétlen telefont teljesen elárasztja adatokkal. Még ha az átvitel hibamentes is, a vevő egy bizonyos ponttól kezdve nem lesz képes kezelni a folyton érkező kereteket, és néhányat el fog veszíteni.

Világos, hogy valamit tenni kell, hogy megakadályozzuk ennek kialakulását. A gyakorlatban két megközelítés terjedt el. Az első a **visszacsatolás-alapú forgalomszabályozás (feedback-based flow control)**. Ebben a megoldásban a vevő információt küld vissza a feladónak, amelyben engedélyt ad neki további adatok küldésére, vagy legalábbis tájékoztatja saját pillanatnyi állapotáról. A második a **sebességalapú forgalomszabályozás (rate-based flow control)**. Itt a protokollba be van építve egy sebességkorlát, amelyet minden küldőnek minden adattovábbítás során tiszteletben kell tartania. Ez a megoldás a fogadó részéről semmilyen visszacsatolást nem igényel.

Ebben a fejezetben a visszacsatolás-alapú forgalomszabályozás módszereivel fogunk foglalkozni elsősorban azért, mert a sebességalapú forgalomirányítás csak a szállítási rétegben fordul elő (lásd 5. fejezet). A visszacsatolás-alapú forgalomszabályozási módszerek léteznek mind az adatkapcsolati, mind a felsőbb rétegekben. A visszacsatolás-alapú forgalomszabályozás manapság sokkal gyakoribb, mivel az adatkapcsolati réteget megvalósító hardverek elég gyorsak ahhoz, hogy ne veszítsenek kereteket. Például az adatkapcsolati réteg hardver megvalósításáról, a **hálózati csatoló kártyáról (Network Interface Card, NIC)** néha azt mondják, hogy „vonali sebességgel fut", mivel a beérkező kereteket olyan gyorsan tudja feldolgozni, ahogy azok a vonalról a vevőbe érkeznek. Bármilyen túltöltés ekkor már nem adatkapcsolati probléma, így a túltöltéssel felsőbb rétegeknek kell foglalkozniuk.

Különféle forgalomszabályozási elgondolások ismertek, de legtöbbjük ugyanazt az alapelvet alkalmazza. A protokoll jól definiált szabályokat tartalmaz arra vonatkozóan, hogy a következő keretet mikor küldheti el az adóállomás. Ezek a szabályok általában megtiltják egy keret elküldését, amíg a vevő – akár implicit, akár explicit módon – enge-

délyt nem ad rá. Például, amikor egy kapcsolat felépül, a vevő mondhatja a következőt: „Most küldhetsz nekem *n* keretet, de ha ezeket elküldted, ne küldj többet addig, amíg nem szólok." A részleteket hamarosan áttekintjük.

3.2. Hibajelzés és hibajavítás

Amint a második fejezetben láttuk, a kommunikációs csatornáknak eltérő karakterisztikáik vannak. Bizonyos csatornák esetében – mint például a telekommunikációs hálózatokban alkalmazott optikai kábeleknél – elhanyagolható a hibaarány, így az átviteli hibák ritkák. Ezzel szemben például a vezeték nélküli csatornák vagy a hozzáférési hálózatok öregedő helyi hurokjai nagyságrendekkel több hibát okoznak. Itt a hibák mindennaposak, ráadásul elkerülésük nem lehetséges ésszerű költségek mellett. A tanulság tehát az, hogy átviteli hibák léteznek, és meg kell tanulnunk kezelni azokat.

A hálózattervezők két alapvető stratégiát dolgoztak ki a hibák kezelésére. Mindkét módszer redundáns információkat csatol az adatokhoz. Az egyik módszer az, hogy minden elküldött adatblokkhoz annyi redundáns információt mellékelünk, amennyiből a vevő ki tudja következtetni, hogy mik voltak az eredetileg elküldött adatok. A másik módszerben csak annyi redundanciát iktatunk az adatok közé, amennyi a vevőnek lehetővé teszi, hogy a hiba tényét kikövetkeztesse. A vevő ebben a megoldásban nem tudja, milyen hiba történt, ezért újraküldést kér. Az előbbi stratégia **hibajavító kódokat (error-correcting code)** használ, az utóbbi pedig **hibajelző kódokat (error-detecting code)**. A hibajavító kódok használatát gyakran **előre irányuló hibajavításnak (Forward Error Correction, FEC)** is nevezik.

Mindkét módszernek megvan a saját alkalmazási területe. A fényvezető szálakon és más, nagymértékben megbízható csatornákon olcsóbb, ha hibajelző kódot használunk, és egyszerűen újraküldjük a ritkán előforduló hibás blokkokat. Ezzel szemben, a vezeték nélküli összeköttetéseken és más olyan csatornákon, amelyek sokat hibáznak, jobb, ha minden blokkba annyi redundanciát építünk, amennyiből a vevő már ki tudja találni, hogy mi volt az eredeti blokk. Az újraküldésre ebben az esetben nem jó támaszkodni, mivel az maga is hibás lehet.

A hibajavító és hibajelző kódolásokkal kapcsolatban kulcsfontosságú megvizsgálni a lehetséges hibatípusokat, mivel egyik sem képes mindenfajta hibát kezelni. Gondoljunk bele, hogy maguk a redundáns bitek is hibásan érkezhetnek meg, nem csak az adatbitek, ami megnehezíti az adatok védelmét. Örvendetes lenne, ha a csatornák a redundáns biteket máshogy kezelnék, mint az adatbiteket, de sajnos mindegyik csak egy-egy bit a csatornán. Ez tehát azt jelenti, hogy ha el akarjuk kerülni az észrevétlen hibákat, a kódolásnak megfelelően erősnek kell lennie a várható hibák kezelésére.

Az egyik modell szerint a hibákat a termikus zaj időnkénti extrém magas értékei okozzák, melyek a hasznos jelhez adódva különálló, egybites hibákat okoznak. Másik modell szerint a hibák csomókban (burst), csoportosan érkeznek, több egymást követő bitet érintve. Ez a modell olyan fizikai folyamatokra épít, melyek természetszerűleg ilyen hibákat okoznak – például egy erőteljes jelgyengülés (fading) egy vezeték nélküli csatornán vagy egy átmeneti villamos interferencia egy vezetékes csatornán.

Mindkét modell fontos a gyakorlatban, és használatuk különböző kompromisszumokhoz vezet. A csoportosan jövő hibáknak megvan az előnyük és a hátrányuk is a különálló, csupán egy bitet érintő hibákkal szemben. Az előny a következő: a számítógépes adatok bitjei mindig blokkokban vannak elküldve. Tételezzük fel, hogy a blokkméret 1000 bit, és átlagosan 0,001 hiba van bitenként. Ha a hibák függetlenek lennének, a legtöbb blokk tartalmazna hibát. Ha a hibák 100-as csoportokban jönnek, 100 közül átlagosan csak egy vagy két blokkot érintenek. A csoportos hibák hátránya az, hogy sokkal nehezebb jelezni és kijavítani őket, mint a különállókat.

Természetesen másfajta hibák is léteznek. Előfordul, hogy a hiba pontos helye ismeretes, például mert a fizikai réteg egy olyan analóg jelet vett, mely például a várt értéktartományokból messze kilóg, ezért azt érvénytelennek tekinti. Az ilyen hibákat produkáló csatornákat **törlődéses csatornáknak (erasure channel)** hívjuk. Ilyen típusú hibákat könnyebb javítani, mint egy-egy bit megváltozását, hiszen ha az érték el is veszett, legalább a hiba helyét tudjuk. Sajnos ritkán van lehetőség a törlődéses csatorna használatára, és így előnyeinek kiaknázására.

A következőkben mind a hibajavító, mind a hibajelző kódokat megvizsgáljuk. Két dolgot azonban nagyon fontos észben tartanunk. Először is, ezeket a kódokat az adatkapcsolati rétegben tárgyaljuk, hiszen itt szembesültünk először a megbízható átvitel követelményével, azonban a kódolások széles körben használatosak, hiszen a megbízhatóság átfogó probléma. Hibajavító kódokat találunk a fizikai rétegben (főleg zajos csatornákon), valamint felsőbb rétegekben is, főleg valós idejű média- és tartalommegosztás területén. Hibajelző kódokat gyakran használunk az adatkapcsolati, a hálózati és a szállítási rétegekben.

A másik fontos, említést érdemlő dolog, hogy a hibajavító és hibajelző kódok elmélete az alkalmazott matematika tárgyterülete, így – hacsak nem vagyunk otthon a Galois-testekben vagy a ritka mátrixok tulajdonságaiban – érdemes megbízható forrásból származó kódokat használnunk, saját kódok készítése helyett. Gyakorlatilag ezt teszik a protokollszabványok is: újra és újra ugyanazok a hibajavító és hibajelző kódok bukkannak fel bennük. A következőkben részletesen áttanulmányozunk egy egyszerű kódot, majd röviden szólunk fejlettebb kódokról is. Így megérthetjük a felmerülő problémákat az egyszerű kód segítségével, majd beszélhetünk a gyakorlatban használt bonyolultabb kódokról is.

3.2.1. Hibajavító kódok

Négy különböző hibajavító kódot fogunk megvizsgálni. Ezek a következők:

1. Hamming-kódok,
2. bináris konvolúciós kódok,
3. Reed–Solomon-kódok,
4. alacsony sűrűségű paritásellenőrző kódok.

Mind a négy kód valamilyen redundanciát csatol az elküldött információhoz. A keretek általában m adatbitből, vagyis üzenetbitből (message bit) és r redundáns bitből, vagyis ellenőrző bitből (check bit) állnak. A **blokk-kódokban** (**block code**) az r ellenőrző bitek kiszámítása csak és kizárólag a hozzájuk tartozó m adatbitek függvényeként történik, pontosan úgy, mintha egy nagy táblából kikeresnénk az m adatbithez tartozó r ellenőrző bitet. Ha az m adatbitet közvetlenül, változtatás (előzetes kódolás) nélkül küldjük el az ellenőrző bitekkel együtt, akkor **szisztematikus kódokról** (**systematic code**) beszélünk. **Lineáris kódokban** az r ellenőrző bit az m adatbit lineáris függvénye, melyre gyakori példa a KIZÁRÓ VAGY (XOR) vagy a modulo 2 összeadás. Ez azt jelenti, hogy a kódolás folyamata mátrixszorzással vagy egyszerű logikai áramkörökkel végezhető. Ebben a szakaszban olyan kódokat tárgyalunk, amelyek – ha másként nem jelöljük – lineáris, szisztematikus blokk-kódok.

A keret teljes hossza legyen n (vagyis $n=m+r$). Az ilyen kódot (n,m) kódnak nevezzük. Egy ilyen, adat- és ellenőrző bitekből álló n bites egységet gyakran n bites **kódszónak** (**codeword**) is neveznek. A **kódsebesség** (**code rate**) vagy egyszerűen sebesség a kódszó azon része, amely a nem redundáns információt tartalmazza (tehát m/n). A kódsebesség a gyakorlatban tág határok között változik. Zajos csatornára akár 1/2 is lehet, így a vett információ fele redundancia, míg egy jó minőségű csatornára megközelítheti az 1-et, hiszen kevés ellenőrző bitet csatolnak egy hosszú üzenethez.

Hogy a hibák kezelését megérthessük, először közelről meg kell vizsgálnunk, hogy egy hiba tulajdonképpen micsoda. Bármely két kódszó, például az 10001001 és az 10110001 esetén meghatározható, hogy a két szó hány egymásnak megfelelő bitje különbözik. Ebben az esetben most 3 bit különbözik. Ahhoz, hogy megszámoljuk a két kódszóban azonos helyeken előforduló különböző biteket, csak egy KIZÁRÓ VAGY (XOR) műveletet kell végezni a két kódszón, majd megszámolni az eredményben előforduló 1-eseket. Például:

```
10001001
10110001
00111000
```

Az olyan helyek számát, amelyeken a két kódszóban különböző bitek állnak, a két kódszó **Hamming-távolságának** [Hamming, 1950] nevezzük. A jelentősége abban áll, hogy ha két kódszó Hamming-távolsága d, akkor d darab egy bitet érintő hiba kell ahhoz, hogy az egyik kódszó a másikba menjen át.

Megadva az ellenőrző bitek kiszámításának módját, meg lehet alkotni a legális kódszavak teljes listáját, és ebből a listából ki lehet keresni azt a két kódszót, melyeknek legkisebb a Hamming-távolsága. Ez a távolság a teljes kód Hamming-távolsága.

A legtöbb adatátviteli alkalmazásban mind a 2^m lehetséges adatüzenet legális, de az ellenőrző bitek kiszámítási módja miatt nem fordul elő mind a 2^n lehetséges kódszó. Valójában r ellenőrző bit esetén csupán a lehetséges üzenetek töredéke, $2^m/2^n$-ed vagy $1/2^n$-ed része lesz legális kódszó. Ez a különbség az, amellyel az üzenet beágyazódik a kódszavak terébe, amely lehetővé teszi, hogy a vevő kijelezze és kijavítsa az átvitel során keletkezett hibákat.

Az, hogy egy blokk-kód hibajelző vagy hibajavító tulajdonságú-e, a kód Hamming-távolságától függ. Ahhoz, hogy d hibát jelezni tudjunk, $d+1$ Hamming-távolságú kód

kell, mert egy ilyen kódban d bithiba nem tudja a kódszót egy másik érvényes kódszóba vinni. Amikor a vevő egy érvénytelen kódszót lát, tudja, hogy átviteli hiba történt. Hasonlóan: ahhoz, hogy d hibát ki tudjunk javítani, $2d+1$ Hamming-távolságú kód kell, mert így az érvényes kódszavak olyan távol vannak, hogy még d bit megváltozásakor is közelebb lesz az eredeti kódszó a hibáshoz, mint bármely másik érvényes kódszóhoz, így az egyértelműen meghatározható, feltételezve, hogy nagyobb számú bithibáknak kicsi a valószínűsége.

Egyszerű példaként a hibajavító kódra, vegyünk egy kódot, ami csak négy érvényes kódszót tartalmaz:

0000000000, 0000011111, 1111100000 és 1111111111

Ennek a kódnak a Hamming-távolsága 5, ami azt jelenti, hogy kétbitnyi hibát tud javítani. Ha a 0000000111 kódszó érkezik, a vevő tudja, hogy az eredetinek 0000011111-nek kellett lennie. Azonban, ha hárombitnyi hiba változtatta a 0000000000-t 0000000111-re, akkor a kódszó nem megfelelően lesz kijavítva. Ezzel szemben a hibát jelezni tudjuk, ha mindegyik hibára számítunk, ugyanis egyik kódszó sem legális, tehát hiba történt az átvitelben. Vegyük észre, hogy ebben a példában nincs lehetőségünk mind a kétbites hibákat javítani, mind a négybites hibákat jelezni, hiszen ehhez a vett kódszavakat kétféleképpen kellene értelmeznünk.

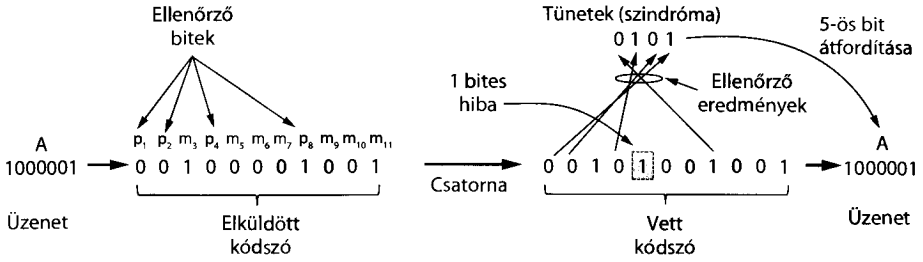
A példánkban a legközelebbi legális kódszó megtalálása csak részletes vizsgálattal végezhető el. Általános esetben, amikor minden egyes legális kódszót át kell néznünk ehhez, meglehetősen hosszú időt vehet igénybe a keresés. A gyakorlatban alkalmazott kódokat ezért úgy tervezték, hogy segítséget nyújtsanak az eredeti kódszó gyors megtalálására.

Képzeljük el, hogy egy olyan kódot akarunk tervezni m üzenet- és r ellenőrző bittel, amely minden egybites hibát ki tud javítani. A 2^m érvényes üzenet mindegyikéhez van n darab, tőle 1 távolságra levő érvénytelen kódszó. Ezeket úgy kaphatjuk meg, hogy az üzenetből képzett n bites kódszó minden egyes bitjét egyenként invertáljuk. Így a 2^m érvényes üzenet mindegyikéhez $n+1$ bitmintát kell hozzárendelnünk. Mivel a bitminták teljes száma 2^n, igaznak kell lennie az $(n+1)2^m \leq 2^n$ egyenlőtlenségnek. Felhasználva, hogy $n=m+r$ a feltétel

$$(m+r+1) \leq 2^r \tag{3.1}$$

Megadva m-et kapunk egy alsó korlátot arra, hogy hány ellenőrző bit szükséges ahhoz, hogy az egybites hibákat ki tudjuk javítani.

Az elméleti alsó korlát valóban elérhető a Hammingnek köszönhető [Hamming, 1950] eljárással. A **Hamming-kódokban** a kódszó bitjeit balról jobbra, 1-gyel kezdődően megszámozzuk. Azok a bitek, melyek sorszáma 2 egész számú hatványa (például 1, 2, 4, 8, 16 stb.), ellenőrző bitek. A maradék bithelyeket (3, 5, 6, 7, 9 stb.) az üzenet bitjeivel töltjük ki. A 3.6. ábra a 7 adatbitet és 4 ellenőrző bitet tartalmazó (11,7) Hamming-kódra mutat példát. Mindegyik ellenőrző bit a bitek valamilyen csoportjának (beleértve önmagát is) a paritását állítja be párosra (vagy páratlanra). Egy bit számos paritásszámítási csoportba tartozhat. Ahhoz hogy megállapítsuk, hogy a k-adik pozícióban levő adatbit



3.6. ábra. *Példa a (11, 7) Hamming-kód egybites hibajavítására*

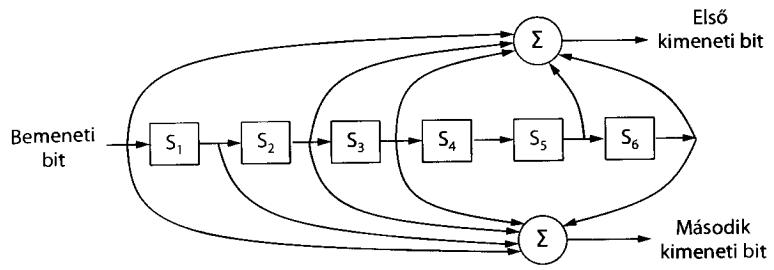
melyik ellenőrző bit kiszámításában vesz részt, írjuk fel k-t 2 hatványainak összegeként. Például $11 = 1+2+8$ és $29 = 1+4+8+16$. Egy bitet azok a paritásbitek ellenőriznek, amelyek sorszáma szerepel az így képzett összeg tagjai között (például a 11. bitet az 1., 2. és a 8. bit ellenőrzi). A példában az ASCII-kódban ábrázolt „A" betűre számoljuk ki az ellenőrző biteket páros paritással.

Ez a felépítés 3 Hamming-távolságú kódot eredményez, így minden egybites hibát javítani és minden 2 bites hibát jelezni tud. Az adat- és ellenőrző bitek számozása a kódszó vétele utáni ellenőrzés során válik világossá. Amikor egy kódszó megérkezik, a vevő elvégzi az ellenőrző bitek kiszámítását, beleszámítva a vett ellenőrző biteket is. Ezeket *ellenőrző eredményeknek* nevezzük. Ha minden bit hibátlan, akkor – páros paritás esetén – az ellenőrző eredmények nullát adnak. Ebben az esetben a kódszót érvényesnek tekintjük.

Ha az ellenőrző eredmények értékei nem nullák, akkor hiba történt az átvitel során. Az ellenőrző eredmények halmaza alkotja a **hiba tünetcsoportot** vagy **hiba szindrómát** (**error syndrome**), ami segítséget nyújt a hiba megtalálására és kijavítására. A 3.6. ábrán látszik, hogy egy egybites hiba történt az átvitelben, így az ellenőrző eredmények sorra 0, 1, 0 és 1 a $k = 8, 4, 2$ és 1 bitekre. Ebből a 0101 szindrómát kapjuk, amely decimálisan $4+1=5$. A kód felépítése miatt ez azt jelenti, hogy az ötödik bit hibás. A hibás bit (ami természetesen adat- és ellenőrző bit is lehet) átfordítása után, majd az ellenőrző bitek eldobása után az eredeti üzenetet, az „A" betű kódját kapjuk.

A Hamming-távolságok sokat segítenek megérteni a blokk-kódokat. A Hamming-kódokat gyakran használják hibajavító memóriákban, azonban a hálózatokban erősebb kódokat használnak. Másodjára a **konvolúciós kódokat** tekintjük át. Ez az egyetlen olyan tárgyalt kódunk, ami nem a blokk-kódok közé tartozik. A konvolúciós kódok esetén a kódoló bemeneti bitek sorozatából kimeneti biteket generál, így nem beszélhetünk az üzenet természetes méretéről vagy kódolási határokról, mint a blokk-kódolók esetében. A kimenet mind az aktuális, mind az előző bemeneti bitektől függ, tehát a kódoló memóriát tartalmaz. Azon korábbi bemeneti bitek számát, melytől az aktuális kimenet függ, a kód **kényszerhosszának** (**constraint length**) nevezzük, és a továbbiakban k-val jelöljük. A konvolúciós kódokat a kényszerhosszukkal és kódsebességükkel jellemezzük.

Konvolúciós kódokat széles körben alkalmazzák olyan, üzemben lévő hálózatokon, mint például a GSM mobilkommunikációs hálózatokban, műholdas kommunikációs rendszerekben, és a 802.11 szabványban. Az egyik legnépszerűbb konvolúciós kódot láthatjuk a 3.7. ábrán. A kód a NASA konvolúciós kódja, $r = 1/2$ és $k = 7$ paraméterekkel.



3.7. ábra. *A 802.11. szabványban alkalmazott NASA bináris konvolúciós kód*

A kódot először az 1977-ben indult Voyager űrmissziókban használták, azóta viszont sok esetben hasznosították, például a 802.11. szabványban is.

Ahogy a 3.7. ábrán látjuk, minden bemeneti bit a bal oldalon két kimeneti bitet képez a jobb oldalon, amelyek a bemenet és a belső állapotok KIZÁRÓ VAGY (XOR) kapcsolatai. Mivel a kód bitekkel dolgozik, és lineáris műveleteket végez, ezért bináris, lineáris konvolúciós kódnak nevezzük. A kódsebesség 1/2, ugyanis 1 bemeneti bit 2 kimeneti bitet eredményez. A kód nem szisztematikus, ugyanis egyik kimeneti bit sem egyezik meg egyszerűen a bemeneti bittel.

A belső állapotot hat regiszterben tároljuk. Amikor egy új bemeneti bit megérkezik, a regiszterek értékeit jobbra léptetjük. Például, ha a bemenet 111, és a kezdeti állapot 0, akkor a belső állapot balról jobbra 100000, 110000 és 111000 lesz rendre, miután az első, második és harmadik bemeneti bit megérkezik. A kimeneti bitek 11, 10 majd 01 értékűek lesznek. Összesen 7 léptetésnek kell megtörténnie, hogy egy bemenet hatása teljesen elmúljon a kimeneten, tehát a kód kényszerhossza $k = 7$.

A konvolúciós kódok dekódolása úgy történik, hogy megkeressük azt a legvalószínűbb bemeneti bitsorozatot, amely a megfigyelt kimeneti (akár hibás) bitsorozatot előállította. Kis k értékekre leggyakrabban a Viterbi által kifejlesztett algoritmust használják [Forney, 1973]. Az algoritmus folyamatosan figyeli a vett bitsorozatot, és sorra előállítja azt a lehetséges bemeneti sorozatot, mely a legkisebb hibával állította volna elő a megfigyelt bitsorozatot. Az algoritmus végén így előáll az a bemeneti sorozat (és így a legvalószínűbb eredeti üzenet), amely a legkisebb hibával állítja elő a megfigyelt bitsorozatot.

A konvolúciós kódokat a gyakorlatban azért szeretik, mert könnyű figyelembe venni a dekódolás során a bitek 0 vagy 1 értékének bizonytalanságát. Tegyük fel például, hogy –1 V felel meg a logikai 0 értéknek, és +1 V a logikai 1 értéknek, és a vétel során 2 bitre 0,9 V-ot és –0,1 V-ot kapunk. Ahelyett, hogy egyből leképeznénk őket 0-ra vagy 1-re, úgy szeretnénk kezelni a 0,9 V-ot, hogy „nagy valószínűséggel 1", és a –0,1 V-ot, hogy „talán 0", és a sorozatot egészében javítani. A Viterbi-algoritmus megfelelő kiterjesztése meg tud birkózni ezzel a bizonytalansággal, és megbízhatóbb hibajavítást képes végezni. Ezt a megközelítést, amikor egy bit bizonytalanságát ily módon veszik figyelembe, **bizonytalan döntésű dekódolásnak** (**soft-decision decoding**) hívjuk. Ezzel ellentétben, a bitek értékének azonnali eldöntését 0 vagy 1 értékre **biztos döntésű dekódolásnak** (**hard-decision decoding**) nevezzük.

A harmadik típusú megvizsgálandó hibajavító kódokat **Reed–Solomon-kódoknak** nevezzük. A Reed–Solomon-kódok – ahogy a Hamming-kódok is – lineáris blokk-kó-

dok, és gyakran szisztematikusak is. Míg azonban a Hamming-kódok különálló biteken működnek, addig a Reed–Solomon-kódok m bites szimbólumokon. Mivel a Reed–Solomon-kódolás komoly matematikai megfontolásokon alapszik, ezért egy analógiával mutatjuk be működését.

A Reed–Solomon-kódok azt a matematikai tényt hasznosítják, hogy minden n-ed fokú polinomot egyértelműen meghatároz $n+1$ pont. Például egy $ax+b$ formában felírható egyenest két pont egyértelműen meghatároz. További pontok az adott egyenesen redundanciát alkotnak, mely segíthet a hibák javításában. Képzeljük el, hogy van két pontunk, melyek egy egyenest reprezentálnak, és elküldjük ezt a két pontot, valamint két további, az egyenesen fekvő pontot. Ha valamely pont hibásan érkezik meg, akkor is helyre tudjuk állítani az összes pontot, ha egy egyenest illesztünk a pontokra. Négyből három pont egy egyenesen fog elhelyezkedni, míg az egyetlen hibás pont nem. Ha megtaláltuk az egyenest, akkor egyben a hibát is kijavítottuk.

A Reed–Solomon-kódok valójában véges mezők feletti polinomokként definiálhatók, de hasonlóképpen működnek. Ha a szimbólumok m bitesek, akkor a kódszavak 2^m-1 szimbólum hosszúak. Egy gyakori választás, $m=8$ esetén a szimbólumok bájt hosszúságúak, és így a kódszavak 255 bájt nagyságúak. A (255, 233) kód széles körben használatos; 233 adat szimbólumhoz 32 redundáns szimbólumot rendel. A kód hibajavítással történő dekódolása Berlekamp és Massey által kifejlesztett algoritmussal végezhető, mely közepes hosszúságú kódokra képes hatékonyan elvégezni az illesztés feladatát [Massey, 1969].

A Reed–Solomon-kódokat a gyakorlatban rendszeresen alkalmazzák, köszönhetően a fejlett hibajavítási képességeinek, főleg csoportos hibák esetén. Használják DSL-vonalakon, műholdas kommunikációkban, és a mindenütt jelenlévő CD-ken, DVD-ken és Blu-Ray lemezeken. Mivel a kód m bites szimbólumokra épül, ezért az egybites hibák és az m-bites hibacsomó ugyanúgy egy szimbólumnyi hibának számítanak. Ha $2t$ redundáns szimbólumot adunk az adatszimbólumokhoz, akkor a Reed–Solomon-kódok t hibát tudnak kijavítani akármelyik átvitt szimbólumon. Ez azt jelenti, hogy például a (255, 233) kód esetén, amely 32 redundáns szimbólumot használ, akár 16 szimbólumnyi hibát is ki tud javítani. Egymást követően küldött 8 bites szimbólumok esetén egy 128 bites hibacsomó is kijavítható. A helyzet abban az esetben még érdekesebb, ha a hiba törlődéses típusú (például karcolás egy CD lemezen olvashatatlanná tesz pár szimbólumot), hiszen ez esetben akár $2t$ hiba is kijavítható.

A Reed–Solomon-kódokat gyakran használják más kódokkal – például konvolúciós kódokkal – kombinálva, ugyanis a konvolúciós kódok hatékonyak elszigetelt bithibák javításában, ugyanakkor hibacsomók javítására nem alkalmasak, főleg, ha sok hiba van a vett bitfolyamban. Ha ehhez hozzáadunk Reed–Solomon-kódolást is, akkor a Reed–Solomon-kód felszámolja a hibacsomókat, ugyanis erre kifejezetten alkalmas. A teljes kódolás így jó védelmet biztosít mind csoportos, mind elszigetelt bithibák ellen.

Az utolsó tárgyalt hibajavító kódok az **LDPC-** (**Low-Density Parity Check – kis sűrűségű paritásellenőrző kódok**) kódok. Az LDPC-kódok lineáris blokk-kódok, melyeket Robert Gallagher talált ki, és a doktori értekezésében tárgyalt [Gallagher, 1962]. Mint általában minden doktori tézist, ezt is hamar elfeledték, és csupán 1995-ben fedezték fel újra, amikor a számítástechnika fejlődése lehetővé tette gyakorlati használatát.

Az LDPC-kódokban minden kimeneti bitet a bemeneti bitek egy részéből képeznek. Így a kód egy mátrixszal jellemezhető, ami nagyon kevés egyest tartalmaz. Innen ered a kód neve. A vett kódszavak dekódolása egy közelítő algoritmussal történik, mely az iterációk során a vett adatot egyre jobban közelíti egy legális kódszóhoz, ami végül a hiba kijavításához vezet.

LDPC-kódok hasznosak nagy blokkméretű adatokhoz, és kitűnő hibajavító képességeik túlszárnyalják sok másik kódét (például azokét is, melyeket eddig részletesebben is megnéztünk). Éppen ezért nagyon gyorsan beépítik az új protokollokba. Több protokollnak része, például a digitális videoszórás szabványa, a 10 gigabites Ethernet, villamos kisfeszültségű hálózaton történő adattovábbítás, valamint a legújabb 802.11 szabvány részét képezik. Várhatóan több más protokoll is tartalmazni fogja a jövőben.

3.2.2. Hibajelző kódok

A hibajavító kódokat széles körben alkalmazzák a vezeték nélküli összeköttetéseken, amelyek a rézvezetékekkel és a fényvezető szálakkal szemben közismerten zajosak és hajlamosak a hibázásra. Hibajavító kódok használata nélkül ezeken az összeköttetéseken nagyon nehéz lenne bármit is átvinni. Ezzel ellentétben, a jó minőségű rézvezetékeken és a fényvezető szálakon a hibaarány annyira kicsi, hogy a hibák felderítése és az újraadás általában hatékonyabb módszer a ritkán előforduló hibák kezelésére.

Három különböző hibajelző kódot fogunk megvizsgálni, melyek mind lineáris, szisztematikus blokk-kódok:

1. paritásbit képzése,
2. ellenőrző összeg képzése,
3. ciklikus redundancia ellenőrzés (Cyclic Redundancy Check, CRC).

Hogy lássuk, miért lehet hatékonyabb a hibajelzés, mint a hibajavítás, nézzük az első kódot, mely egy **paritásbitet** (**parity bit**) csatol az adatokhoz. A paritásbitet úgy választjuk meg, hogy a kódszóban lévő egyesek száma páros (vagy páratlan) legyen. A páros paritásbit meghatározása történhet az adatbitek modulo 2 összegének kiszámításával, vagy KIZÁRÓ VAGY (XOR) kapcsolatuk képzésével. Például az 1011010 bitekhez páros paritás esetén egy 0 bitet csatolunk, így az eredmény 10110100 lesz. Páratlan paritásbit esetén 10110101-et kapnánk. Egybites paritás esetén a kód Hamming-távolsága 2, mivel minden egybites hiba rossz paritáskóddal ellátott kódszót hoz létre. A paritásbit tehát egybites hibákat tud jelezni.

Egyszerű példaként vegyünk egy olyan csatornát, amelyen a hibák izoláltak, és a bithiba-arány 10^{-6}. Ez meglehetősen kicsinek tűnik, de elfogadható közelítés egy olyan hosszú kábel esetén, melyen hibajelzésre van szükség. Tipikus LAN-kapcsolatok hibaaránya 10^{-10}. A blokkméret legyen 1000 bit. Az 1000 bites blokk hibajavító kódolásához a (3.1) egyenletből tudjuk, hogy 10 ellenőrző bit szükséges; egy megabit adathoz 10 000 ellenőrző bitre lenne szükség. Ahhoz, hogy egyetlen 1 bites hibát jelezni tudjunk, blokkonként egyetlen paritásbit elegendő. 1000 blokkonként egy bit hiba miatt egy további blokkot (1001 bitet) kell átvinnünk a hiba kijavítására. A hibajelzés és újraküldés mód-

szerével az összes többletként átvitt bit egy megabit adatra vonatkoztatva mindössze 2001 bit, míg a Hamming-kód esetén 10 000 bit.

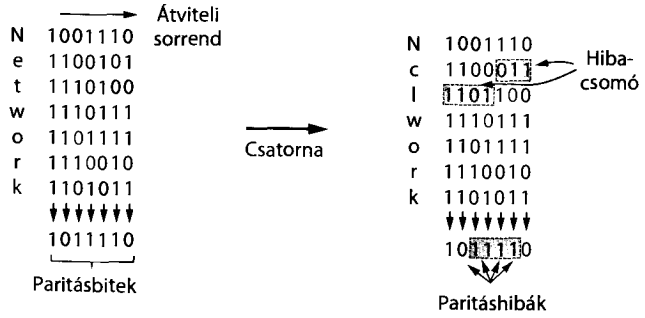
Ennek a megoldásnak az egyik hátránya az, hogy a blokkot egyetlen paritásbittel egészítjük ki a blokkban lévő egyszeres hiba biztonságos kijelzésére. Ha egy hosszú hibacsomó erősen eltorzítja blokkot, a hiba jelzésének az esélye csupán 0,5, ami alig elfogadható. Az esély jelentősen növelhető, ha minden blokkot egy n bit szélességű, k bit hosszúságú mátrixnak tekintve küldünk el. Ha most a sorokhoz kiszámítunk és elküldünk egy paritásbitet, maximum k bitnyi hibát tudunk megbízhatóan jelezni, feltételezve, hogy minden sorban maximum egy bithiba történt.

A hibacsomók ellen azért tudunk még küzdeni: a paritásbiteket más sorrendben is kiszámíthatjuk, mint ahogyan az adatokat elküldjük. Ezt a módszert **összefésülésnek (interleaving)** nevezik. Ebben az esetben először kiszámítjuk a paritásbiteket mind az n oszlophoz, majd elküldjük a k sornyi adatbitet a szokásos módon, soronként felülről lefelé, a sorokban a biteket balról jobbra. Végül az n paritásbitet – mint utolsó sort – küldjük. A módszert a 3.8. ábra mutatja $n=7$ és $k=7$ méretű mátrixra.

Az összefésülés gyakori technika, hogy a különálló hibák felfedezésére (javítására) szolgáló kódolást alkalmassá tegyük hibacsomók jelzésére (javítására). A 3.8. ábrán egy olyan $n=7$ bites hibacsomót láthatunk, melynek bitjei különböző oszlopokat érintenek. (A hibacsomó nem jelenti azt, hogy az összes bit rossz, csak azt, hogy legalább az első és az utolsó az. A 3.8. ábrán éppen 4 bit változott meg a 7-ből.) Mivel legfeljebb 1 bit változott meg minden oszlopban, az oszlopok paritásbitjei jelezni fogják a hibát. Ez a módszer tehát n paritásbitet csatol kn bites adatblokkokhoz, hogy egy legfeljebb n bit hosszú hibacsomót jelezzen.

Egy $n+1$ hosszúságú hibacsomó továbbra is észrevétlen marad, ha az első és az utolsó bit invertálódik, és az összes többi bit helyes. Ha a blokk erősen eltorzul egy hosszú vagy több rövid hibacsomó miatt, annak az esélye, hogy az n oszlop közül bármelyikben véletlenül jó a paritásbit, 0,5. Így annak a valószínűsége, hogy egy blokkot elfogad a vevő, amikor nem kellene: 2^{-n}.

A következő hibajelző kód, az **ellenőrző összeg (checksum)** képzése szorosan kapcsolódik a paritásbitek csoportjához. Az „ellenőrző összeg" szó gyakran az üzenethez csatolt ellenőrző bitek egy csoportját jelenti, függetlenül a kiszámításuk módjától. A paritásbitek egy csoportja jó példa az ellenőrző összegre. Mindazonáltal léteznek más,



3.8. ábra. *A paritásbitek összefésülése csoportos hibák jelzésére*

erősebb ellenőrző összegek, melyeket az üzenetben lévő adatbitek mozgó összegével képeznek. Az ellenőrző összeget általában az üzenet végére helyezik, mint az összegző függvény eredményét. Így a hiba kijelezhető a teljes vett kódszó – beleértve az adatbiteket és az ellenőrző összeget – összegzésével. Ha az eredmény nullát ad, az átvitelben nem történt hiba.

Jó példa ellenőrző összegre a 16 bites IP ellenőrző összeg, mely az IP-protokoll részeként minden internetet megjárt csomagban megtalálható [Braden és mások, 1988]. Itt az ellenőrző összeg az üzenet 16 bitre felosztott szavainak összege. Mivel a módszer – a paritásbit-képzéssel ellentétben – 16 bites szavakkal, és nem bitekkel dolgozik, olyan hibákat is felfedez, melyek a paritásbitet nem módosítják. Például, ha két szó legkisebb helyi értékű bitjei 0-ról 1-re változnak, a paritásellenőrzés nem fogja a hibát kimutatni, míg a 16 bites ellenőrző összeghez adott további két 1-es megváltoztatja az összeget, így a hiba felfedhető.

Az IP ellenőrző összeget egyes-komplemens aritmetikával számítják, nem modulo 2^{16}-os összeadással. Az egyes-komplemens aritmetikában egy negatív szám a neki megfelelő pozitív szám bitenkénti negáltja. A modern számítógépek kettes-komplemens aritmetikát használnak, ahol egy negatív szám a neki megfelelő pozitív szám egyes-komplemense megnövelve eggyel. Kettes-komplemenst alkalmazó gép esetén az egyes-komplemens összeg megegyezik a modulo 2^{16} összeggel, ha ez utóbbihoz minden legnagyobb helyi értéken túlcsorduló bitet a legkisebb helyi értékhez hozzáadjuk. Ez az algoritmus az ellenőrző összeg által az adatok egységesebb kezelését biztosítja. Máskülönben a két legnagyobb helyi értéken lévő bitet össszeadhatjuk, amely túlcsordul, és elveszik anélkül, hogy az ellenőrző összeget megváltoztatta volna. Az egyes-komplemensnek van még egy előnye, ugyanis a nullát kétféleképpen is jelöli: lehet csupa 0-ból, és lehet csupa 1-ből álló szám. Ez lehetővé teszi, hogy az egyiket (például a csupa 0-t) arra használjuk, hogy az ellenőrző összeg hiányát jelölje anélkül, hogy erre egy újabb mezőt bevezetnénk.

Évtizedekig azt tételezték fel, hogy az ellenőrző összeggel ellátandó keretek véletlen biteket tartalmaznak. Az ellenőrző összeget készítő algoritmusok analízise teljes egészében ezen a feltételezésen alapult. A valós adatok vizsgálata azt mutatja, hogy ez a feltételezés elég alaptalan [Partridge és mások, 1995]. Következésképpen, bizonyos körülmények között a felderítetlen hibák sokkal gyakoribbak, mint azt korábban gondoltuk volna.

Az IP ellenőrző összeg kifejezetten hatékony és egyszerű, de bizonyos esetekben kevés védelmet nyújt éppen azért, mert csak egy egyszerű összeg. Nem jelzi például zérus adatbitek eltűnését és beékelődését, sem az üzenet részeinek megcserélődését, és kevés védelmet nyújt üzenetek összefonódásai ellen, amikor két csomag különböző részei együvé válnak. Persze ilyen hibákat ritkán produkálnak véletlen folyamatok, de a hibás hardverelemek éppen ilyen és hasonló hibákat okozhatnak.

Az ellenőrző összegek területén jobb választás **Fletcher ellenőrző összege** [Fletcher, 1982]. Az algoritmus figyelembe veszi a szó elhelyezkedését is, ugyanis az összeghez az adat és az adat pozíciójának szorzatát adja. Ez erősebb védelmet nyújt az adat pozíciójának megváltozása ellen.

Meglehet, hogy az előző két megoldás néha megfelelő a felsőbb rétegekben, a gyakorlatban azonban egy harmadik, sokkal erősebb módszert alkalmaznak széleskörűen az adatkapcsolati rétegben: **CRC-t (Cyclic Redundancy Check – ciklikus redundancia**

ellenőrzés) vagy más néven a **polinom kódot** (**polynomial code**). A polinom kódok azon alapulnak, hogy a bitsorozatokat polinomok reprezentációjának tekintjük, melyekben csupán a 0 és 1 együtthatók szerepelnek. Egy k bites keretet tekintsünk egy k tagú polinom együtthatóinak x^{k-1}-től x^0-ig. Az ilyen polinomot $k-1$-ed fokúnak nevezzük. A legnagyobb helyi értékű (bal szélső) bit az x^{k-1} tag együtthatója; a következő bit az x^{k-2} tagé és így tovább. Például a 110001 bitsorozat 6 bites, így egy 6 tagú polinomot reprezentál 1, 1, 0, 0, 0 és 1 együtthatókkal: $1x^5+1x^4+0x^3+0x^2+0x^1+1x^0$.

A polinom aritmetika a modulo 2 moduláris aritmetikán alapul az absztrakt algebra mező elméletének megfelelően. Nincs átvitel sem az összeadásnál, sem a kivonásnál, mindkét művelet a KIZÁRÓ VAGY (XOR) művelettel azonos. Például:

10011011	00110011	11110000	01010101
+11001010	+11001101	−10100110	−10101111
01010001	11111110	01010110	11111010

Az osztást ugyanúgy kell elvégezni, mint a binárist, kivéve, hogy a kivonás – mint fent – modulo 2 értendő. Az osztó „megvan" az osztandóban, ha az osztandó ugyanannyi bitet tartalmaz, mint az osztó.

Amikor a polinom-kódot alkalmazzuk, az adónak és a vevőnek előre meg kell egyeznie egy **generátor polinomban** (**generator polynomial**). Jelöljük ezt $G(x)$-szel. A generátor legmagasabb és legkisebb helyi értékű bitjének 1-nek kell lennie. Ahhoz, hogy egy $M(x)$ polinomnak megfelelő m bites keret ellenőrző összegét kiszámíthassuk, a keretnek hosszabbnak kell lennie, mint a generátor polinom. Az ötlet az, hogy úgy fűzzünk ellenőrző összeget a kerethez, hogy az így kapott keret által reprezentált polinom osztható legyen $G(x)$-szel. Amikor a vevő megkapja a keretet, megpróbálja elosztani $G(x)$-szel. Ha van maradék, akkor hiba volt az átvitel során.

Az ellenőrző összeg kiszámításának algoritmusa a következő:

1. Legyen r a $G(x)$ foka. Fűzzünk r darab 0 bitet a keret legkisebb helyi értékű végéhez, így az $m+r$ bitet fog tartalmazni és megfelel az $x^rM(x)$ polinomnak.

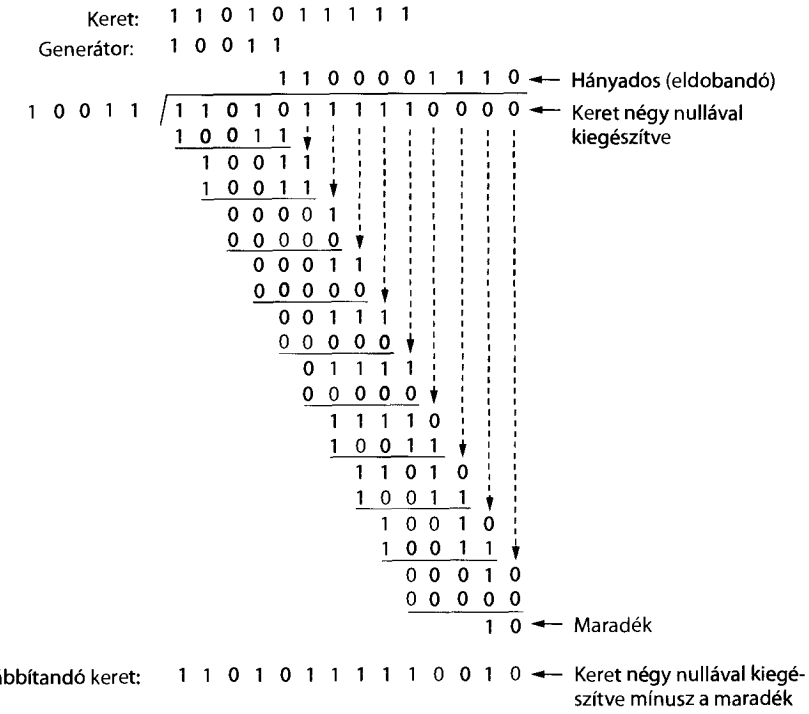
2. Osszuk el az $x^rM(x)$-hez tartozó bitsorozatot a $G(x)$-hez tartozó bitsorozattal modulo 2 aritmetika szerint.

3. Vonjuk ki a maradékot (mely mindig r vagy r-nél kevesebb bitet tartalmaz) az $x^rM(x)$-nek megfelelő bitsorozatból modulo 2-es kivonással. Az eredmény az ellenőrző összeget tartalmazó, továbbítandó keret. Nevezzük ennek a polinomját $T(x)$-nek.

A 3.9. ábra illusztrálja a számítást egy 1101011111-et tartalmazó keretre és $G(x)=x^4+x+1$-re.

Tisztán látszik, hogy $T(x)$ osztható (modulo 2 aritmetika szerint) $G(x)$-szel. Bármilyen osztási problémánál, ha az osztandót csökkentjük a maradékkal, az eredmény osztható lesz az osztóval. Például ha 10-es számrendszerben elosztunk 210278-at 10941-gyel, a maradék 2399. Kivonva 2399-et 210278-ból, az eredmény (207879) osztható 10941-gyel.

Most pedig elemezzük a módszer erejét! Milyen hibákat fogunk tudni jelezni? Képzeljük el, hogy átviteli hiba történt, így a $T(x)$-nek megfelelő bitsorozat helyett $T(x)+E(x)$-nek



3.9. ábra. *Példa CRC kiszámítására*

megfelelő érkezik. Minden 1-es bit $E(x)$-ben egy invertálódott bitnek felel meg. Ha k darab 1-es bit van $E(x)$-ben, k darab egybites hiba történt. Egy hibacsomó egy kezdeti 1-gyel, 0-k és 1-ek keverékével, és egy záró 1-gyel jellemezhető. $E(x)$ többi bitje ebben az esetben 0.

Az ellenőrző összeggel ellátott keretet a vevő elosztja $G(x)$-szel; azaz kiszámítja $[T(x)+E(x)]/G(x)$-et. $T(x)/G(x)$ egyenlő 0-val, így a számítás eredménye csak az $E(x)/G(x)$. Azok a hibák, melyek történetesen olyan polinomnak felelnek meg, amelyek $G(x)$ többszörösei, átcsúsznak az ellenőrzésen; minden más hibát felismer a vevő.

Ha egy egybites hiba történt, $E(x)=x^i$, ahol i a hibás bit sorszáma. Ha $G(x)$ kettő vagy több tagból áll, soha nem fogja osztani $E(x)$-et, így minden egybites hibát jelezni fogunk.

Ha két izolált egybites hiba történt, $E(x)=x^i+x^j$, ahol $i>j$. Másképpen ez az $E(x)=x^j(x^{i-j}+1)$ alakba írható. Ha feltételezzük, hogy $G(x)$ nem osztható x-szel, akkor ahhoz, hogy minden kettős hibát jelezni tudjunk, $G(x)$-nek nem oszthatja x^k+1-et semmilyen az $i-j$ maximális értékénél (azaz a maximális kerethossznál) kisebb k-ra. Egyszerű, alacsony fokszámú polinomok ismertek hosszú keretek védelmére. Például az $x^{15}+x^{14}+1$ nem osztja x^k+1-et semmilyen 32768-nál kisebb k-ra.

Ha páratlan számú bit hibás, $E(x)$ páratlan számú tagot tartalmaz (például x^5+x^2+1, de például az x^2+1 nem ilyen). Elég érdekes, hogy nincs olyan páratlan tagot tartalmazó polinom, amely osztható lenne $x+1$-gyel a modulo 2-es rendszerben. $G(x)$-et úgy megválasztva, hogy osztható legyen $x+1$-gyel, az összes páratlan számú invertált bitet tartalmazó hibát felismerhetjük.

Végül, a legfontosabb az, hogy egy r ellenőrző bittel ellátott polinom-kód minden legfeljebb r hosszúságú hibacsomót jelezni tud. Egy k hosszúságú hibacsomó $x^i(x^{k-1}+\ldots+1)$-gyel reprezentálható, ahol i azt mutatja, hogy a hibacsomó a keret jobb szélétől milyen messze kezdődik. Ha $G(x)$ tartalmaz x^0 tagot, nem osztható x^i-nel, így ha a zárójeles kifejezés fokszáma kisebb, mint $G(x)$-é, a maradék sohasem lehet 0.

Ha a hibacsomó hossza $r+1$, a $G(x)$-szel való osztás maradéka akkor, és csak akkor lehet 0, ha a hibacsomó $G(x)$-nek felel meg. A hibacsomó definíciója miatt az első és az utolsó bit mindenképpen 1, így az, hogy a hiba megfelel-e $G(x)$-nek, az $r-1$ közbenső biten múlik. Ha minden kombinációt egyenlően valószínűnek veszünk, akkor annak a valószínűsége, hogy egy ilyen hibás keretet a vevő elfogad: $1/2^{r-1}$.

Az is megmutatható, hogy amikor egy $r+1$ bitnél hosszabb vagy több rövidebb hibacsomó lép fel, akkor annak a valószínűsége, hogy egy hibás keret észrevétlen marad $1/2^r$, feltételezve, hogy minden bitminta egyformán valószínű.

Néhány polinom nemzetközi szabvánnyá vált. Az Ethernet példáját követve az IEEE 802 által használt polinom:

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$$

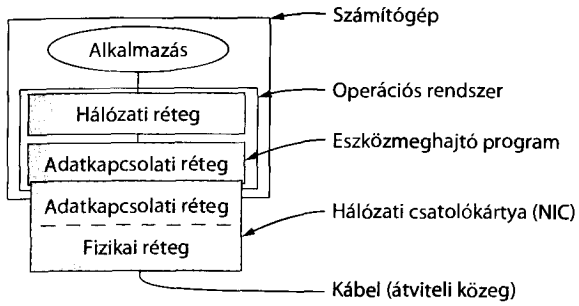
Egyéb jó tulajdonságai mellett például arra is képes, hogy minden 32 bites vagy annál rövidebb hibacsomót, valamint minden páratlan számú bitet érintő hibacsomót jelezzen. Bár az 1980-as évek óta előszeretettel használják, ez nem jelenti azt, hogy ez lenne a legjobb választás. Kimerítő keresés után megtalálták a legjobb polinomokat [Castagnoli és mások, 1993; Koopman, 2002], melyek Hamming-távolsága tipikus üzenetméretekre 6, míg az IEEE-szabvány CRC-32 polinomjának Hamming-távolsága mindössze 4.

Bár az ellenőrző összeg kiszámításához szükséges algoritmus bonyolultnak tűnhet, Peterson és Brown [1961] megmutatta, hogy szerkeszthető egy egyszerű, léptető regiszteres áramkör az ellenőrző összeg hardverben történő kiszámítására és ellenőrzésére. A gyakorlatban majdnem mindig ezt a hardvert használják. Számos hálózati szabvány használ CRC-t, többek között minden LAN (például Ethernet, 802.11) és a kétpontos adatkapcsolatok (például packets over SONET).

3.3. Elemi adatkapcsolati protokollok

A protokollok témakörébe való bevezetésként három, egyre növekvő bonyolultságú protokollt fogunk megnézni. Az érdeklődő olvasók számára ezekhez és további protokollokhoz egy szimulátor áll rendelkezésre a weben (lásd a bevezetőt). Mielőtt megnéznénk a protokollokat, hasznos nyilvánvalóvá tenni néhány, a kommunikáció modelljét megalapozó feltételezést.

Először is feltételezzük, hogy a fizikai réteg, az adatkapcsolati réteg és a hálózati réteg független folyamatok, melyek üzenetek oda-vissza küldözgetésével kommunikálnak egymással. Egy gyakori megvalósítás látható a 3.10. ábrán. A fizikai réteg folyamatai és az adatkapcsolati réteg néhány folyamata egy hozzájuk rendelt (dedikált) hardveren fut, melyet **hálózati csatoló kártyának (Network Interface Card, NIC)** hívnak. Az adat-



3.10. ábra. *A fizikai, az adatkapcsolati és a hálózati réteg megvalósítása*

kapcsolati réteg folyamatának és a hálózati réteg folyamatának többi része a fő CPU-n fut, az operációs rendszer részeként. Az adatkapcsolati réteg szoftvere gyakran **eszközillesztő** vagy **eszközmeghajtó program (device driver)** formájában található meg. Persze más megvalósítások is lehetségesek (például három folyamat egyetlen kitüntetett chipen belül, melyet **hálózati gyorsítónak (network accelerator)** hívnak, vagy a három folyamat a fő CPU-n fut szoftverrel definiált sebességgel). Az implementáció lényegében évtizedről évtizedre változik a technikai haladásnak megfelelően. Bármelyik esetben a három réteg különálló folyamatnak tekintése fogalmilag világosabbá teszi a tárgyalást, és a rétegek függetlenségének hangsúlyozását is szolgálja.

Egy másik kulcsfontosságú feltételezés az, hogy az A gép megbízható, összeköttetésalapú szolgáltatás alkalmazásával akar a B gépnek egy hosszú adatfolyamot küldeni. Később azt az esetet is átgondoljuk, amikor egyidejűleg B is akar adatot küldeni A-nak. A-ról feltételezzük, hogy végtelen utánpótlása van elküldendő adatokból, és sohasem kell várakoznia az adatok előállítására. Amikor A adatkapcsolati rétege adatot kér, a hálózati réteg mindig azonnal teljesíteni tudja a kérést. (Ezt a megkötést később szintén elvetjük.)

Azt is feltesszük, hogy a gépek nem fagynak le. Ez azt jelenti, hogy ezek a protokollok a kommunikációs hibákkal foglalkoznak, nem pedig a lefagyott és újraindított számítógépek által okozott problémákkal.

Ami az adatkapcsolati réteget illeti, a hálózati réteg által az interfészen keresztül hozzá eljuttatott csomag tiszta adat, aminek minden bitjét továbbítani kell a célgép hálózati rétegéhez. Az, hogy a hálózati réteg a csomag egy részét fejrésznek értelmezi, nem tartozik az adatkapcsolati rétegre.

Amikor az adatkapcsolati réteg megkap egy csomagot, azt egy adatkapcsolati fejrésszel és farokrésszel kiegészítve ágyazza be egy keretbe (lásd 3.1. ábra). Így egy keret egy beágyazott csomagból, némi vezérlési információból (a fejrészben) és egy ellenőrző összegből áll (a farokrészben). A keretet az adatkapcsolati réteg ezután továbbítja a másik gép adatkapcsolati rétegének. A továbbiakban fel fogjuk tenni, hogy léteznek megfelelő könyvtári eljárások a keretek átadására a fizikai rétegnek (küldés: *to_physical_layer*), valamint azok átvételére a fizikai rétegtől (fogadás: *from_physical_layer*). Ezek az eljárások számítják ki, ellenőrzik és illesztik a keretbe az ellenőrző összeget (melyet jórészt egyébként hardverben valósítanak meg), így az általunk ebben a fejezetben fejlesztett algoritmusoknak nem kell bajlódniuk vele. Használható például az ebben a fejezetben korábban ismertetett CRC-algoritmus.

Kezdetben a vevőnek nincs mit tennie. Csak üldögél, várva, hogy valami történjen. A fejezet példaprotokolljaiban a *wait_for_event(&event)* eljáráshívással jelezzük, hogy az adatkapcsolati réteg vár valamire. Ez az eljárás csak akkor tér vissza, ha valami történt (például egy keret érkezett). A visszatéréskor az *event* változó mondja meg, hogy mi volt az esemény. A lehetséges események halmaza a különféle ismertetett protokolloknál különböző, és mindegyik protokollhoz külön fogjuk meghatározni. Megjegyezzük, hogy egy valóságosabb helyzetben az adatkapcsolati réteg nem ücsörög egy végtelen ciklusban eseményre várva, ahogyan javasoltuk, hanem egy megszakítást kezel, aminek a beérkezésekor félbeszakítja addigi tevékenységét, és a beérkező keret feldolgozását kezdi el. Ennek ellenére az egyszerűség kedvéért nem veszünk tudomást az adatkapcsolati rétegben folyó párhuzamos tevékenységekről, és feltételezzük, hogy a szoftver dolga kizárólag a mi egyetlen csatornánk kezelése.

Amikor egy keret érkezik a vevőhöz, a hardver kiszámítja az ellenőrző összeget. Ha az helytelen (azaz átviteli hiba történt), az adatkapcsolati réteget tájékoztatja erről (*event = cksum_err*). Ha a keret sértetlenül érkezik meg, az adatkapcsolati réteg erről szintén tájékoztatást kap (*event = frame_arrival*), így az megkaphatja a keretet a fizikai rétegtől a *from_physical_layer* eljárás használatával. Amint a vevő adatkapcsolati rétege megkapta a sértetlen keretet, ellenőrzi a vezérlőinformációkat a fejrészben, és ha minden rendben van, a csomag részt továbbadja a hálózati rétegnek. A keret fejrésze semmilyen körülmények között sem kerül a hálózati réteghez.

Jó oka van annak, hogy a hálózati rétegnek sohasem szabad átadni a keret fejrészének semmilyen részét: a hálózati és az adatkapcsolati protokollt teljesen szét kell választani. Amíg a hálózati réteg semmit sem tud az adatkapcsolati protokollról vagy a keretformátumról, ezek anélkül változhatnak meg, hogy bármilyen változtatást kellene tenni a hálózati réteg szoftverében. Pontosan ez történik, amikor egy új hálózati csatoló kártyát telepítünk egy számítógépre. A hálózati és az adatkapcsolati réteg között merev csatolást biztosítva, nagyon leegyszerűsödik a szoftver tervezése, mert így a különböző rétegekben levő kommunikációs protokollok egymástól függetlenül fejleszthetők.

A 3.11. ábra néhány deklarációt mutat be (C nyelven), amelyek több, később ismertetendő protokollban is megtalálhatók. Öt adatstruktúrát definiáltunk: *boolean*, *seq_nr*, *packet*, *frame_kind* és *frame*. A *boolean* egy felsorolt típus, és a *true* és a *false* értékeket veheti fel. A *seq_nr* egy kis egész szám, melyet a keretek megszámozására használunk, hogy meg tudjuk őket különböztetni. Ezek a sorszámok 0 és *MAX_SEQ* közötti értéket vehetnek fel, *MAX_SEQ*-t is beleértve, mely minden egyes protokollban definiálva van, amelyiknek szüksége van rá. A *packet* az az információegység, amit az azonos gépen levő hálózati és adatkapcsolati réteg között kerül átadásra, vagy a kommunikáló hálózati réteg párok cserélnek ki egymással. A mi modellünkben ez mindig *MAX_PKT* számú bájtot tartalmaz, de a valóságnak jobban megfelelne, ha változó hosszúságú lenne.

A *frame* négy mezőből áll: *kind*, *seq*, *ack* és *info*, melyekből az első három vezérlőinformációt tartalmaz, az utolsó pedig a valódi továbbítandó adatot. Ezeket a vezérlőmezőket együttesen a **keret fejrészének (frame header)** nevezzük.

A *kind* mező azt mondja meg, hogy van-e adat a keretben, néhány protokoll ugyanis különbséget tesz azok között a keretek között, amelyek kizárólag vezérlőinformációt tartalmaznak, illetve amelyek adatot is. A *seq* és az *ack* mezőket sorszámozáshoz, illetve


```
#define MAX_PKT 1024                          /* megadja a keretek hosszát bájtban */

typedef enum {true, false} boolean;           /* logikai típus */
typedef unsigned int seq_nr;                  /* sorszámok vagy nyugta számok */
typedef struct {unsigned char data[MAX_PKT];} packet;   /* csomag definíció */
typedef enum {data, ack, nak} frame_kind;     /* keret fajta definíció */

typedef struct {                              /* ebben a rétegben kereteket továbbítunk */
  frame_kind kind;                            /* milyen fajta keret? */
  seq_nr seq;                                 /* sorszám */
  seq_nr ack;                                 /* nyugta száma */
  packet info;                                /* a hálózati rétegbeli csomag */
} frame;

/* Vár egy esemény bekövetkezésére; visszaadja a típusát az event-ben. */
void wait_for_event(event_type *event);

/* Lehoz egy csomagot a hálózati rétegtől a csatornán való továbbításra. */
void from_network_layer(packet *p);

/* Az érkező keretből átadja az információt a hálózati rétegnek. */
void to_network_layer(packet *p);

/* Átveszi az érkező keretet a fizikai rétegtől és r-be másolja. */
void from_physical_layer(frame *r);

/* Átadja a keretet a fizikai rétegnek továbbításra. */
void to_physical_layer(frame *s);

/* Elindítja az órát és engedélyezi a timeout eseményt. */
void start_timer(seq_nr k);

/* Leállítja az órát és letiltja a timeout eseményt. */
void stop_timer(seq_nr k);

/* Elindítja a segéd időzítőt és engedélyezi az ack_timeout eseményt. */
void start_ack_timer(void);

/* Leállítja a segéd időzítőt és letiltja az ack_timeout eseményt. */
void stop_ack_timer(void);

/* Engedélyezi a hálózati rétegnek, hogy network_layer_ready eseményt okozzon. */
void enable_network_layer(void);

/* Megtiltja a hálózati rétegnek, hogy network_layer_ready eseményt okozzon. */
void disable_network_layer(void);

/* Az inc makro soron belül (in-line) lesz kifejtve: körkörösen növeli k-t. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

3.11. ábra. *Néhány definíció, amely a későbbi protokollokhoz szükséges. Ezek a definíciók a protocol.h fájlban találhatók*

Végül, a legfontosabb az, hogy egy r ellenőrző bittel ellátott polinom-kód minden legfeljebb r hosszúságú hibacsomót jelezni tud. Egy k hosszúságú hibacsomó $x^i(x^{k-1}+\ldots+1)$-gyel reprezentálható, ahol i azt mutatja, hogy a hibacsomó a keret jobb szélétől milyen messze kezdődik. Ha $G(x)$ tartalmaz x^0 tagot, nem osztható x^i-nel, így ha a zárójeles kifejezés fokszáma kisebb, mint $G(x)$-é, a maradék sohasem lehet 0.

Ha a hibacsomó hossza $r+1$, a $G(x)$-szel való osztás maradéka akkor, és csak akkor lehet 0, ha a hibacsomó $G(x)$-nek felel meg. A hibacsomó definíciója miatt az első és az utolsó bit mindenképpen 1, így az, hogy a hiba megfelel-e $G(x)$-nek, az $r-1$ közbenső biten múlik. Ha minden kombinációt egyenlően valószínűnek veszünk, akkor annak a valószínűsége, hogy egy ilyen hibás keretet a vevő elfogad: $1/2^{r-1}$.

Az is megmutatható, hogy amikor egy $r+1$ bitnél hosszabb vagy több rövidebb hibacsomó lép fel, akkor annak a valószínűsége, hogy egy hibás keret észrevétlen marad $1/2^r$, feltételezve, hogy minden bitminta egyformán valószínű.

Néhány polinom nemzetközi szabvánnyá vált. Az Ethernet példáját követve az IEEE 802 által használt polinom:

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+1$$

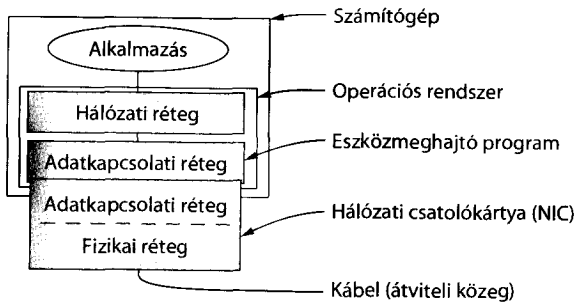
Egyéb jó tulajdonságai mellett például arra is képes, hogy minden 32 bites vagy annál rövidebb hibacsomót, valamint minden páratlan számú bitet érintő hibacsomót jelezzen. Bár az 1980-as évek óta előszeretettel használják, ez nem jelenti azt, hogy ez lenne a legjobb választás. Kimerítő keresés után megtalálták a legjobb polinomokat [Castagnoli és mások, 1993; Koopman, 2002], melyek Hamming-távolsága tipikus üzenetméretekre 6, míg az IEEE-szabvány CRC-32 polinomjának Hamming-távolsága mindössze 4.

Bár az ellenőrző összeg kiszámításához szükséges algoritmus bonyolultnak tűnhet, Peterson és Brown [1961] megmutatta, hogy szerkeszthető egy egyszerű, léptető regiszteres áramkör az ellenőrző összeg hardverben történő kiszámítására és ellenőrzésére. A gyakorlatban majdnem mindig ezt a hardvert használják. Számos hálózati szabvány használ CRC-t, többek között minden LAN (például Ethernet, 802.11) és a kétpontos adatkapcsolatok (például packets over SONET).

3.3. Elemi adatkapcsolati protokollok

A protokollok témakörébe való bevezetésként három, egyre növekvő bonyolultságú protokollt fogunk megnézni. Az érdeklődő olvasók számára ezekhez és további protokollokhoz egy szimulátor áll rendelkezésre a weben (lásd a bevezetőt). Mielőtt megnéznénk a protokollokat, hasznos nyilvánvalóvá tenni néhány, a kommunikáció modelljét megalapozó feltételezést.

Először is feltételezzük, hogy a fizikai réteg, az adatkapcsolati réteg és a hálózati réteg független folyamatok, melyek üzenetek oda-vissza küldözgetésével kommunikálnak egymással. Egy gyakori megvalósítás látható a 3.10. ábrán. A fizikai réteg folyamatai és az adatkapcsolati réteg néhány folyamata egy hozzájuk rendelt (dedikált) hardveren fut, melyet **hálózati csatoló kártyának (Network Interface Card, NIC)** hívnak. Az adat-



3.10. ábra. *A fizikai, az adatkapcsolati és a hálózati réteg megvalósítása*

kapcsolati réteg folyamatának és a hálózati réteg folyamatának többi része a fő CPU-n fut, az operációs rendszer részeként. Az adatkapcsolati réteg szoftvere gyakran **eszközillesztő** vagy **eszközmeghajtó program (device driver)** formájában található meg. Persze más megvalósítások is lehetségesek (például három folyamat egyetlen kitüntetett chipen belül, melyet **hálózati gyorsítónak (network accelerator)** hívnak, vagy a három folyamat a fő CPU-n fut szoftverrel definiált sebességgel). Az implementáció lényegében évtizedről évtizedre változik a technikai haladásnak megfelelően. Bármelyik esetben a három réteg különálló folyamatnak tekintése fogalmilag világosabbá teszi a tárgyalást, és a rétegek függetlenségének hangsúlyozását is szolgálja.

Egy másik kulcsfontosságú feltételezés az, hogy az A gép megbízható, összeköttetésalapú szolgáltatás alkalmazásával akar a B gépnek egy hosszú adatfolyamot küldeni. Később azt az esetet is átgondoljuk, amikor egyidejűleg B is akar adatot küldeni A-nak. A-ról feltételezzük, hogy végtelen utánpótlása van elküldendő adatokból, és sohasem kell várakoznia az adatok előállítására. Amikor A adatkapcsolati rétege adatot kér, a hálózati réteg mindig azonnal teljesíteni tudja a kérést. (Ezt a megkötést később szintén elvetjük.)

Azt is feltesszük, hogy a gépek nem fagynak le. Ez azt jelenti, hogy ezek a protokollok a kommunikációs hibákkal foglalkoznak, nem pedig a lefagyott és újraindított számítógépek által okozott problémákkal.

Ami az adatkapcsolati réteget illeti, a hálózati réteg által az interfészen keresztül hozzá eljuttatott csomag tiszta adat, aminek minden bitjét továbbítani kell a célgép hálózati rétegéhez. Az, hogy a hálózati réteg a csomag egy részét fejrésznek értelmezi, nem tartozik az adatkapcsolati rétegre.

Amikor az adatkapcsolati réteg megkap egy csomagot, azt egy adatkapcsolati fejrésszel és farokrésszel kiegészítve ágyazza be egy keretbe (lásd 3.1. ábra). Így egy keret egy beágyazott csomagból, némi vezérlési információból (a fejrészben) és egy ellenőrző összegből áll (a farokrészben). A keretet az adatkapcsolati réteg ezután továbbítja a másik gép adatkapcsolati rétegének. A továbbiakban fel fogjuk tenni, hogy léteznek megfelelő könyvtári eljárások a keretek átadására a fizikai rétegnek (küldés: *to_physical_layer*), valamint azok átvételére a fizikai rétegtől (fogadás: *from_physical_layer*). Ezek az eljárások számítják ki, ellenőrzik és illesztik a keretbe az ellenőrző összeget (melyet jórészt egyébként hardverben valósítanak meg), így az általunk ebben a fejezetben fejlesztett algoritmusoknak nem kell bajlódniuk vele. Használható például az ebben a fejezetben korábban ismertetett CRC-algoritmus.

Kezdetben a vevőnek nincs mit tennie. Csak üldögél, várva, hogy valami történjen. A fejezet példaprotokolljaiban a *wait_for_event(&event)* eljáráshívással jelezzük, hogy az adatkapcsolati réteg vár valamire. Ez az eljárás csak akkor tér vissza, ha valami történt (például egy keret érkezett). A visszatéréskor az *event* változó mondja meg, hogy mi volt az esemény. A lehetséges események halmaza a különféle ismertetett protokolloknál különböző, és mindegyik protokollhoz külön fogjuk meghatározni. Megjegyezzük, hogy egy valóságosabb helyzetben az adatkapcsolati réteg nem ücsörög egy végtelen ciklusban eseményre várva, ahogyan javasoltuk, hanem egy megszakítást kezel, aminek a beérkezésekor félbeszakítja addigi tevékenységét, és a beérkező keret feldolgozását kezdi el. Ennek ellenére az egyszerűség kedvéért nem veszünk tudomást az adatkapcsolati rétegben folyó párhuzamos tevékenységekről, és feltételezzük, hogy a szoftver dolga kizárólag a mi egyetlen csatornánk kezelése.

Amikor egy keret érkezik a vevőhöz, a hardver kiszámítja az ellenőrző összeget. Ha az helytelen (azaz átviteli hiba történt), az adatkapcsolati réteget tájékoztatja erről (*event = cksum_err*). Ha a keret sértetlenül érkezik meg, az adatkapcsolati réteg erről szintén tájékoztatást kap (*event = frame_arrival*), így az megkaphatja a keretet a fizikai rétegtől a *from_physical_layer* eljárás használatával. Amint a vevő adatkapcsolati rétege megkapta a sértetlen keretet, ellenőrzi a vezérlőinformációkat a fejrészben, és ha minden rendben van, a csomag részt továbbadja a hálózati rétegnek. A keret fejrésze semmilyen körülmények között sem kerül a hálózati réteghez.

Jó oka van annak, hogy a hálózati rétegnek sohasem szabad átadni a keret fejrészének semmilyen részét: a hálózati és az adatkapcsolati protokollt teljesen szét kell választani. Amíg a hálózati réteg semmit sem tud az adatkapcsolati protokollról vagy a keretformátumról, ezek anélkül változhatnak meg, hogy bármilyen változtatást kellene tenni a hálózati réteg szoftverében. Pontosan ez történik, amikor egy új hálózati csatoló kártyát telepítünk egy számítógépre. A hálózati és az adatkapcsolati réteg között merev csatolást biztosítva, nagyon leegyszerűsödik a szoftver tervezése, mert így a különböző rétegekben levő kommunikációs protokollok egymástól függetlenül fejleszthetők.

A 3.11. ábra néhány deklarációt mutat be (C nyelven), amelyek több, később ismertetendő protokollban is megtalálhatók. Öt adatstruktúrát definiáltunk: *boolean*, *seq_nr*, *packet*, *frame_kind* és *frame*. A *boolean* egy felsorolt típus, és a *true* és a *false* értékeket veheti fel. A *seq_nr* egy kis egész szám, melyet a keretek megszámozására használunk, hogy meg tudjuk őket különböztetni. Ezek a sorszámok 0 és *MAX_SEQ* közötti értéket vehetnek fel, *MAX_SEQ*-t is beleértve, mely minden egyes protokollban definiálva van, amelyiknek szüksége van rá. A *packet* az az információegység, amit az azonos gépen levő hálózati és adatkapcsolati réteg között kerül átadásra, vagy a kommunikáló hálózati réteg párok cserélnek ki egymással. A mi modellünkben ez mindig *MAX_PKT* számú bájtot tartalmaz, de a valóságnak jobban megfelelne, ha változó hosszúságú lenne.

A *frame* négy mezőből áll: *kind*, *seq*, *ack* és *info*, melyekből az első három vezérlőinformációt tartalmaz, az utolsó pedig a valódi továbbítandó adatot. Ezeket a vezérlőmezőket együttesen a **keret fejrészének** (**frame header**) nevezzük.

A *kind* mező azt mondja meg, hogy van-e adat a keretben, néhány protokoll ugyanis különbséget tesz azok között a keretek között, amelyek kizárólag vezérlőinformációt tartalmaznak, illetve amelyek adatot is. A *seq* és az *ack* mezőket sorszámozáshoz, illetve


```
#define MAX_PKT 1024                                  /* megadja a keretek hosszát bájtban */

typedef enum {true, false} boolean;                   /* logikai típus */
typedef unsigned int seq_nr;                          /* sorszámok vagy nyugta számok */
typedef struct {unsigned char data[MAX_PKT];} packet; /* csomag definíció */
typedef enum {data, ack, nak} frame_kind;             /* keret fajta definíció */

typedef struct {                                      /* ebben a rétegben kereteket továbbítunk */
  frame_kind kind;                                    /* milyen fajta keret? */
  seq_nr seq;                                         /* sorszám */
  seq_nr ack;                                         /* nyugta száma */
  packet info;                                        /* a hálózati rétegbeli csomag */
} frame;

/* Vár egy esemény bekövetkezésére; visszaadja a típusát az event-ben. */
void wait_for_event(event_type *event);

/* Lehoz egy csomagot a hálózati rétegtől a csatornán való továbbításra. */
void from_network_layer(packet *p);

/* Az érkező keretből átadja az információt a hálózati rétegnek. */
void to_network_layer(packet *p);

/* Átveszi az érkező keretet a fizikai rétegtől és r-be másolja. */
void from_physical_layer(frame *r);

/* Átadja a keretet a fizikai rétegnek továbbításra. */
void to_physical_layer(frame *s);

/* Elindítja az órát és engedélyezi a timeout eseményt. */
void start_timer(seq_nr k);

/* Leállítja az órát és letiltja a timeout eseményt. */
void stop_timer(seq_nr k);

/* Elindítja a segéd időzítőt és engedélyezi az ack_timeout eseményt. */
void start_ack_timer(void);

/* Leállítja a segéd időzítőt és letiltja az ack_timeout eseményt. */
void stop_ack_timer(void);

/* Engedélyezi a hálózati rétegnek, hogy network_layer_ready eseményt okozzon. */
void enable_network_layer(void);

/* Megtiltja a hálózati rétegnek, hogy network_layer_ready eseményt okozzon. */
void disable_network_layer(void);

/* Az inc makro soron belül (in-line) lesz kifejtve: körkörösen növeli k-t. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

3.11. ábra. *Néhány definíció, amely a későbbi protokollokhoz szükséges. Ezek a definíciók a protocol.h fájlban találhatók*

nyugtákhoz használjuk; később részletesebben ismertetjük a használatukat. A keret *info* mezője egyetlen csomagot tartalmaz; egy vezérlőkeret az *info* mezőt nem használja. Egy a valóságnak jobban megfelelő megvalósításban változó hosszúságú *info* mezőt kellene használnunk, és a vezérlőkereteknél ezt a mezőt el is kellene hagynunk.

Fontos, hogy tisztában legyünk a csomag és a keret egymáshoz való viszonyával. A hálózati réteg úgy építi fel a csomagot, hogy vesz egy üzenetet a szállítási rétegtől és hozzáteszi a hálózati réteg fejrészét. Ezt a csomagot átadja az adatkapcsolati rétegnek, hogy az beletegye egy kimenő keret *info* mezőjébe. Amikor a keret megérkezik a célhoz, az adatkapcsolati réteg kiveszi a csomagot a keretből és átadja a hálózati rétegnek. Ilyen módon a hálózati réteg úgy működhet, mintha a gépek közvetlenül csomagokat tudnának cserélni.

Számos eljárást is felsoroltunk a 3.11. ábrán. Ezek könyvtári rutinok, melyek részletei megvalósításfüggők, és belső működésükkel itt nem foglalkozunk a továbbiakban. A *wait_for_event* eljárás egy végtelen ciklusban ücsörög, várva, hogy valami történjen, mint ahogyan korábban már említettük. A *to_network_layer* és a *from_network_layer* eljárásokat az adatkapcsolati réteg arra használja, hogy csomagokat adjon át a hálózati rétegnek, illetve vegyen át a hálózati rétegtől. Vegyük észre, hogy a *from_physical_layer* és a *to_physical_layer* eljárásokat keretek átadására használjuk az adatkapcsolati és a fizikai réteg között, míg a *to_network_layer* és a *from_network_layer* eljárásokat csomagok átadására az adatkapcsolati réteg és a hálózati réteg között. Más szóval, a *to_network_layer* és a *from_network_layer* a 2. és 3. rétegek közötti interfésszel foglalkozik, míg a *from_physical_layer* és a *to_physical_layer* az 1. és 2. réteg közöttivel.

A legtöbb protokollban megbízhatatlan csatornát tételezünk fel, amely alkalomadtán egész kereteket elveszíthet. Ahhoz, hogy ilyen csapásokból képes legyen felépülni, a küldő adatkapcsolati rétegnek minden keret elküldésekor el kell indítania egy belső időzítőt vagy órát. Ha bizonyos, előre meghatározott időn belül nem érkezik válasz, az óra lejár, és az adatkapcsolati réteg egy megszakítás jelzést kap.

A protokolljainkban úgy oldjuk ezt meg, hogy a *wait_for_event* eljárás visszatérhet *event = timeout* eseménnyel. A *start_timer* és *stop_timer* eljárásokat az időzítő be-, illetve kikapcsolására használjuk. Az időtúllépés csak akkor következhet be, ha az időzítő jár, és mielőtt a *stop_timer*-t meghívnánk. A *start_timer* meghívható akkor is, ha az időzítő jár; egy ilyen hívás egyszerűen lenullázza az órát, hogy a következő időtúllépés egy teljes időzítési intervallum után következzen be (hacsak közben az időzítőt nem nullázzuk le újra, vagy nem kapcsoljuk ki).

A *start_ack_timer* és a *stop_ack_timer* eljárásokat a segéd időzítő vezérlésére használjuk. Ez az időzítő állít elő nyugtákat bizonyos körülmények között.

Az *enable_network_layer* és a *disable_network_layer* eljárásokat azok a kifinomultabb protokollok használják, melyekben már nem tételezzük fel, hogy a hálózati rétegnek mindig van küldeni való csomagja. Amikor az adatkapcsolati réteg engedélyezi a hálózati réteget, a hálózati réteg megszakítást küldhet, ha van elküldeni való csomagja. Ezt az *event = network_layer_ready*-vel jelezzük. Amikor a hálózati réteg le van tiltva, nem okozhat ilyen eseményt. Ügyelve arra, hogy mikor engedélyezi és tiltja le a hálózati rétegét, az adatkapcsolati réteg megakadályozhatja, hogy a hálózati réteg elárassza csomagokkal, amelyek tárolására nincs puffer területe.

A keretsorszámok mindig a 0–*MAX_SEQ* (zárt) intervallumon belül vannak, ahol *MAX_SEQ* a különböző protokollokhoz különböző. Gyakran szükséges, hogy egy sorszámot 1-gyel továbbléptessünk körkörösen (azaz *MAX_SEQ*-t 0 követi). Az *inc* makró ezt a léptetést végzi. Azért definiáltuk makróként, mert a kritikus végrehajtási útvonalon belül soron belül kifejtve (in-line) használjuk. Ahogyan a könyv későbbi részében látni fogjuk, az a tényező, mely korlátozza a hálózat teljesítőképességét, gyakran a protokoll végrehajtása, így ha az olyan egyszerű műveleteket, mint ez, makróként definiáljuk, ez nem befolyásolja a kód olvashatóságát, de javítja a teljesítőképességét.

A 3.11. ábrán levő deklarációk közül mindegyik a következőkben ismertetendő protokollban megtalálható. Helytakarékosságból és a kényelmesebb hivatkozás miatt, kiemeltük őket és együtt soroltuk fel, de fogalmilag magukkal a protokollokkal összevonva kell érteni őket. A C nyelvben ezt az összevonást a definíciók speciális fejrész fájlba (ebben az esetben a *protocol.h*) helyezésével, és a C előfeldolgozó #include lehetőségének használatával érjük el, így a definíciók a fordításkor a protokoll fájlokban megjelennek.

3.3.1. Egy utópikus szimplex protokoll

Bevezető példaként egy olyan protokollt veszünk, amelyik annyira egyszerű, amennyire csak lehet, ugyanis nem aggódik amiatt, hogy bármi gond felmerülhet. Az adatokat csak egy irányba továbbítjuk. Mind az adó, mind a vevő hálózati rétege mindig készen áll. A feldolgozási időtől eltekinthetünk. Végtelen pufferterület áll rendelkezésre. És a legjobb: az adatkapcsolati rétegek közötti kommunikációs csatorna sohasem rontja vagy veszíti el a kereteket. Ezt a gondosan valószerűtlen protokollt, melyre építkezünk, „utópiá"-nak fogjuk becézni, megvalósítása a 3.12. ábrán látható.

A protokoll két különálló eljárásból épül fel: egy küldő eljárásból és egy vevő eljárásból. A küldő a forrásgép adatkapcsolati rétegében fut, a vevő pedig a címzett gépében. Semmilyen sorszámozást vagy nyugtázást nem használ, így a *MAX_SEQ*-ra nincs szükség. Az egyetlen lehetséges eseményfajta a *frame_arrival* (azaz egy sértetlen keret érkezése).

A küldő egy végtelen while ciklus, amely olyan gyorsan pumpálja kifelé a vonalra az adatokat, ahogyan csak tudja. A ciklusmag három teendőből áll: szerezni egy csomagot a (mindig szolgálatkész) hálózati rétegtől, összerakni egy keretet az *s* változó segítségével, és útjára indítani a keretet. Ez a protokoll csak a keret *info* mezőjét használja, mivel a többi mező a hibajavításhoz és a forgalomszabályozáshoz kell, és itt nincsenek hibák vagy forgalomszabályozási megkötések.

A vevő ugyanilyen egyszerű. Kezdetben vár, hogy valami történjen. Az egyetlen lehetőség egy sértetlen keret érkezése. Végül is megérkezik a keret, és a *wait_for_event* eljárás visszatér az *event*-et *frame_arrival*-re állítva (melyet egyébként nem vesz figyelembe). A *from_physical_layer* meghívása eltávolítja a keretet a hardver pufferéből, és beleteszi az *r* változóba. Végül az adatrészt továbbadja a hálózati rétegnek, és az adatkapcsolati réteg dolga végeztével várakozni kezd a következő keretre: felfüggeszti magát, amíg a keret meg nem érkezik.

Az „utópia" protokoll a valóságtól elrugaszkodott, hiszen sem forgalomszabályozást, sem hibakezelést nem tartalmaz. A feldolgozás menete alapján nagyon hasonlít a

```
/* Az 1. protokoll (utópia) csak egyirányú adatátvitelről gondoskodik: az adótól a vevő felé.
   A kommunikációs csatornát hibamentesnek tételezzük fel. A vevőről azt tételezzük fel,
   hogy képes a bemenő adatokat végtelen gyorsan feldolgozni. Következésképp az adó egy
   ciklusban pumpálja ki az adatokat a vonalra olyan gyorsan, ahogyan csak tudja. */

typedef enum {frame_arrival} event_type;
#include „protocol.h"

void sender1(void)
{
  frame s;                          /* puffer a kimenő keretnek */
  packet buffer;                    /* puffer a kimenő csomagnak */

  while (true) {
    from_network_layer(&buffer);    /* szerezz valami küldenivalót */
    s.info = buffer;                /* másold s-be a továbbításhoz */
    to_physical_layer(&s);          /* ereszd útjára */
  }                                 /* Holnap és holnap és holnap:
                                       tipegve vánszorog létünk
                                       a kimért idő végső szótagjáig ...
                                       - Macbeth V. felv., 5. szín
                                       ford.: Szabó Lőrinc */
}

void receiver1(void)
{
  frame r;
  event_type event;                 /* a wait_for_event tölti ki, de itt nem használja */

  while (true) {
    wait_for_event(&event);         /*az egyetlen lehetőség a frame_arrival */
    from_physical_layer(&r);        /* szerezd meg a bejövő keretet */
    to_network_layer(&r.info);      /* add tovább az adatot a hálózati rétegnek */
  }
}
```

3.12. ábra. *Egy korlátozás nélküli szimplex protokoll*

nyugtázatlan, összeköttetés nélküli szolgáltatásra, mely e problémák megoldását felsőbb rétegekre bízza. Ennek ellenére egy nyugtázatlan, összeköttetés nélküli szolgáltatás is tipikusan tartalmaz legalább hibajelzést.

3.3.2. Szimplex megáll-és-vár protokoll hibamentes csatornához

A következőkben azt a problémát fogjuk megoldani, hogy az adó a vevőt olyan sebességgel árassza el keretekkel, hogy az képtelen feldolgozni azokat. Mivel ez a probléma a gyakorlatban rendszerint jelen van, ezért nagyon fontos, hogy megelőzzük. A kommunikációs csatornát azonban továbbra is hibamentesnek feltételezzük, és az adatforgalom is még egyirányú.

Az egyik megoldás, hogy a vevőt olyan gyorsra tervezzük, hogy képes legyen szünet nélkül, szorosan egymást követő kereteket feldolgozni (vagy ami ezzel egyenértékű, hogy az adatkapcsolati réteget megfelelően lassúra tervezzük, hogy a vevő bírja a tempót). Ebben az esetben a vevőnek megfelelő méretű pufferekkel és feldolgozási kapacitással kell rendelkeznie, hogy vonali sebességgel tudjon futni, és a beérkező kereteket megfelelő sebességgel tudja a hálózati rétegnek továbbadni. Ez nyilvánvalóan egy „legrosszabb eset" típusú megoldás. A megoldás hozzárendelt célhardvert igényel, és meglehetősen erőforrás pazarló lehet, ha az adatkapcsolat kihasználtsága többnyire kicsi. Ráadásul a túl gyors adóval való foglalkozás problémáját áttolja máshová, jelen esetben a hálózati rétegre.

Egy sokkal általánosabb megoldás erre a problémára az, hogy a vevő visszacsatolást biztosít a küldő állomás felé. Miután a vevő átadta a csomagot a hálózati rétegének, visszaküld egy kis álkeretet, amely valójában engedély a küldő állomásnak arra, hogy továbbítsa a következő keretet. Miután az állomás egy keretet elküldött, a protokoll szerint várakoznia kell addig, amíg a kis ál- (azaz nyugta-) keret meg nem érkezik. A késleltetés ilyen módon történő megvalósítása jó példa egy forgalomszabályozási protokollra.

Azokat a protokollokat, melyekben a küldő egy keret elküldése után nyugtára vár, mielőtt továbbmenne, **megáll-és-vár** (**stop-and-wait**) protokollnak nevezik. A 3.13. ábrán a szimplex megáll-és-vár protokollra látható példa.

Bár ebben a példában az adatáramlás egyirányú: csak a küldőtől a vevő felé folyik, keretek mindkét irányban utaznak. Következésképpen a két adatkapcsolati réteg közötti kommunikációs csatornának alkalmasnak kell lennie kétirányú információátvitelre. A protokollból következik az áramlás irányának szigorú váltakozása: először a küldő állomás küld egy keretet, azután a vevő küld egy keretet, azután a küldő küld még egy keretet, azután a vevő még egyet és így tovább. Ehhez egy fél-duplex fizikai csatorna is megfelelő.

Úgy, mint az 1. protokollban, a küldő állomás itt is azzal kezdi, hogy lekér egy csomagot a hálózati rétegtől, ennek segítségével összeállít egy keretet, és útjára küldi. Csakhogy most – nem úgy, mint az 1. protokollban – a küldőnek várakoznia kell, amíg egy nyugtakeret meg nem érkezik, mielőtt újabb ciklusba kezdene, és lekérné a következő csomagot a hálózati rétegtől. A küldő adatkapcsolati rétegnek meg sem kell néznie az érkező keretet, mivel úgyis csak egy lehetőség van, hiszen a bejövő keret mindig egy nyugta.

Az egyetlen különbség a *receiver1* és a *receiver2* között az, hogy miután a csomagot kézbesítette a hálózati rétegnek, a *receiver2* egy nyugtakeretet küld vissza a küldő állomásnak a várakozó hurokba való újbóli belépés előtt. Mivel a küldőnek csak a keret visszaérkezése fontos, az nem, hogy mit tartalmaz, a vevőnek nem szükséges semmilyen különleges információt beletenni.

3.3.3. Szimplex megáll-és-vár protokoll zajos csatornához

Most vegyük figyelembe azt a normális esetet, hogy a kommunikációs csatorna hibázhat. A keretek megsérülhetnek vagy teljesen elveszhetnek. Azt azért feltételezzük, hogy ha egy keret megsérül az átvitel során, a vevő hardvere ezt felismeri, amikor kiszámítja

```
/* A 2. protokoll (megáll-és-vár) is egyirányú adattovábbításról gondoskodik az adótól a vevő
   felé. A kommunikációs csatornát az 1. protokollhoz hasonlóan szintén hibamentesnek
   feltételezzük. Ebben az esetben azonban a vevőről feltételezzük, hogy véges puffer
   kapacitással és véges számítási sebességgel rendelkezik, így a protokollnak kifejezetten el
   kell kerülnie, hogy az adó gyorsabban érkező adatokkal árassza el a vevőt, mint ahogy az le
   tudja kezelni. */

typedef enum {frame_arrival} event_type;
#include „protocol.h"

void sender2(void)
{
  frame s;                          /* puffer a kimenő keretnek */
  packet buffer;                    /* puffer a kimenő csomagnak */
  event_type event;                 /* frame_arrival az egyetlen lehetőség */

  while (true) {
    from_network_layer(&buffer);    /* szerezz valami küldenivalót */
    s.info = buffer;                /* másold s-be a továbbításhoz */
    to_physical_layer(&s);          /* viszlát kicsi keret */
    wait_for_event(&event);         /* ne folytasd, amíg nem hallod, hogy gyerünk */
  }
}

void receiver2(void)
{
  frame r, s;                       /* pufferek kereteknek */
  event_type event;                 /* frame_arrival az egyetlen lehetőség */

  while (true) {
    wait_for_event(&event);         /*az egyetlen lehetőség a frame_arrival */
    from_physical_layer(&r);        /* szerezd meg a bejövő keretet */
    to_network_layer(&r.info);      /* add tovább az adatot a hálózati rétegnek */
    to_physical_layer(&s);          /* küldj egy ál-keretet a küldő állomás */
                                    /* felébresztésére */
  }
}
```

3.13. ábra. *Egy szimplex megáll-és-vár protokoll*

az ellenőrző összeget. Ha a keret úgy sérül meg, hogy az ellenőrző összeg ennek ellenére helyes – ami rendkívül ritkán fordul elő –, ez a protokoll (és az összes többi is) hibázhat (azaz egy hibás keret kézbesítődik a hálózati rétegnek).

Első pillantásra úgy tűnhet, hogy a 2. protokoll egy kis változtatással jó lenne: be kellene építeni egy időzítőt. A küldő állomás küldhetne egy keretet, de a vevő csak akkor küldene nyugtakeretet, ha az adat helyesen érkezett meg. Ha sérült keret érkezik a vevőhöz, el kellene dobni. Egy idő után a küldőnek lejárna az időzítője, és újra elküldené a keretet. Ezt a folyamatot addig ismételnénk, míg végül a keret sértetlenül megérkezik.

A fenti sémának van egy végzetesen gyenge pontja. Gondolkodjunk a problémáról, és mielőtt továbbmegyünk, próbáljuk kitalálni, hogy mi a baj!

Ahhoz, hogy lássuk, mi lehet a baj, emlékezzünk rá, hogy az adatkapcsolati réteg folyamatainak a kötelessége, hogy hibamentes, átlátszó kommunikációt biztosítsanak a hálózati rétegek folyamatai között. Az *A* gép hálózati rétege egy csomagsorozatot ad az adatkapcsolati rétegnek, amelynek biztosítania kell, hogy megegyező csomagsorozatot kapjon a hálózati réteg a *B* gépen az adatkapcsolati rétegtől. Különös tekintettel arra, hogy a *B* gép hálózati rétegének nincs módja tudomást szerezni arról, hogy egy csomag elveszett vagy megkettőződött, az adatkapcsolati rétegnek kell garantálnia, hogy az átviteli hibák semmilyen kombinációja – tekintet nélkül arra, hogy milyen ritkán fordulnak elő – se okozhassa, hogy egy keret duplán érkezzen meg a hálózati réteghez.

Gondoljuk át a következő eseménysort:

1. Az *A* hálózati rétege átadja az 1. csomagot az adatkapcsolati rétegnek. A csomag rendben megérkezik *B*-hez, és átadódik *B* hálózati rétegének. *B* visszaküld egy nyugtakeretet *A*-nak.

2. A nyugtakeret teljesen elveszik. Egyáltalán nem érkezik meg soha. Az élet sokkal egyszerűbb lenne, ha a csatorna csak az adatkereteket tenné tönkre és veszítené el, a vezérlőkereteket nem, de szomorúan azt kell mondanunk, hogy a csatorna nem nagyon tud különbséget tenni.

3. *A* adatkapcsolati rétegének végül lejár az időzítője. Mivel nem érkezett nyugta, azt feltételezi (hibásan), hogy az adatkerete elveszett vagy megsérült, így újra elküldi az 1. csomagot tartalmazó keretet.

4. A megkettőzött keret szintén tökéletesen megérkezik *B* adatkapcsolati rétegéhez, az pedig – minden rossz szándék nélkül – továbbadja az ottani hálózati rétegnek. Ha *A* egy állományt küld *B*-nek, az állomány egy része megkettőződik (azaz *B* által az állományról készített másolat hibás lesz, és a hibát senki nem jelezte ki). Egyszóval, a protokoll hibázni fog.

Nyilvánvalóan arra van szükség, hogy valahogyan képessé tegyük a vevőt arra, hogy meg tudja különböztetni az először látott kereteket az újraküldöttektől. Ennek kézenfekvő módja az, hogy az adó egy sorszámot tesz minden elküldött keret fejlécébe. Ekkor a vevő ellenőrizheti minden érkező keret sorszámát, hogy megállapítsa, hogy új keret érkezett-e, vagy csak egy megkettőzött, amit el kell dobni.

Mivel a protokollnak hibamentesnek kell lennie, és a sorszámmezőnek pedig a fejlécben elegendően kicsinek kell lennie ahhoz, hogy az adatkapcsolat hatékonyan működjön, felmerül a kérdés: hány bit szükséges a fejlécben a sorszám tárolására? A sorszám részére a fejlécben a protokolltól függően lehet 1 bit, néhány bit, 1 bájt vagy akár több bájt. A lényeg mindenesetre az, hogy a sorszámmezőnek elegendően nagynak kell lennie ahhoz, hogy a protokoll hibátlanul működjön.

A protokollban az egyetlen nem egyértelmű helyzet az *m*. és a közvetlenül azt követő, $(m + 1)$. keret között áll fenn. Ha az *m*. keret elveszett vagy megsérült, a vevő nem fogja nyugtázni, így a küldő tovább próbálkozik az elküldésével. Ha egyszer hibátlanul megérkezik, a vevő visszaküld egy nyugtát a küldő állomásnak. Ez az a pont, ahol a potenciális

hiba fellép. Attól függően, hogy a nyugtakeret rendben visszajut a küldő állomáshoz vagy sem, az adó az (m + 1). vagy az m. keretet próbálja meg elküldeni.

Az esemény, mely kiváltja az (m + 1). keret elküldését, az m. keretre vonatkozó nyugta megérkezése. Ez azonban azt is magában foglalja, hogy az (m – 1). keret is rendben megérkezett, továbbá az (m – 1). keret nyugtája is megérkezett a küldő állomáshoz, másképp a küldő nem is fogott volna hozzá az m. keret adásához, és pláne békén hagyta volna az (m + 1). keretet is. Következésképpen az egyetlen bizonytalanság egy keret és az azt közvetlenül megelőző vagy azt követő keret között van, nem pedig a keretet megelőző és a keretet követő között.

A fentiek miatt egy 1 bites (0 vagy 1) sorszám elegendő. A vevő minden pillanatban pontosan tudja, hogy milyen sorszámot vár. Amikor egy helyes sorszámot tartalmazó keret érkezik, azt elfogadja, és átadja a hálózati rétegnek, majd a várt sorszámot modulo 2 szerint lépteti (azaz a 0-ból 1 lesz, az 1-ből pedig 0). A vevő minden hibás sorszámot tartalmazó beérkező keretet – másolatnak tekintve – eldob. Az utolsó érvényes nyugtát azonban újra elküldi a vevő, így az adó tudni fogja, hogy a keret megérkezett.

A 3.14. ábrán egy példa látható ilyenfajta protokollra. Azokat a protokollokat, amelyekben a küldő állomás pozitív nyugtára vár, mielőtt továbblépne a következő adategységre, gyakran **PAR**-nak (**Positive Acknowledgement with Retransmission – pozitív nyugtázás újraküldéssel**) vagy **ARQ**-nak (**Automatic Repeat reQuest – automatikus ismétléskérés**) nevezik. A 2. protokollhoz hasonlóan, ez is csak egy irányba továbbítja az adatokat.

A 3. protokoll abban különbözik elődeitől, hogy az adónak és a vevőnek is van olyan változója, amelynek emlékszik az értékére, mialatt az adatkapcsolati réteg várakozási állapotban van. Az adó a következő elküldendő keret sorszámát tárolja a *next_frame_to_send*-ben; a vevő a következő venni kívánt keret sorszámát a *frame_expected*-ben. Mindkét eljárásnak van egy rövid inicializációs fázisa, mielőtt belép a végtelen ciklusba.

Az adóállomás egy keret továbbítása után elindít egy időzítőt. Ha már ment az időzítő, akkor nullázódik, hogy megint egy teljes időzítési intervallum álljon rendelkezésre. Az időzítési intervallumot úgy kell megválasztani, hogy elég ideje legyen a keretnek eljutni a vevőhöz, a vevőnek feldolgozni azt a legrosszabb esetben is, és a nyugtakeretnek visszaérni az adóállomáshoz. Csak ha ez az idő eltelt, akkor feltételezhetjük biztonsággal, hogy vagy a továbbított keret vagy annak nyugtája elveszett, és csak ekkor küldhetünk egy másik példányt belőle. Ha az időzítési intervallumok túl rövidek, akkor a küldő felesleges kereteket is elküld. Ezek a többletforgalmat jelentő keretek nem javítanak a protokoll helyességén, viszont rontják teljesítményét.

Egy keret továbbítása és az időzítő elindítása után az adóállomás várakozni kezd, hátha történik valami izgalmas. Három lehetőség van: egy nyugtakeret érkezik sértetlenül, egy sérült nyugtakeret támolyog be vagy az időzítő lejár. Ha egy érvényes nyugta jön be, az adó lekéri a következő csomagot a hálózati rétegétől, és beleteszi a pufferbe, felülírva az előző csomagot, valamint lépteti a sorszámot is. Ha sérült keret vagy egyáltalán semmi sem érkezik, sem a puffer, sem a sorszám nem változik, így a következő ciklusban egy újabb példányát tudjuk elküldeni az előbbi keretnek. Mindegyik esetben a puffer tartalmát (mind a következő csomagnál, mind a másolatnál) küldjük el.

Amikor egy érvényes keret érkezik a vevőhöz, megvizsgáljuk a sorszámát, hogy meglássuk, duplikátum-e. Ha nem az, akkor elfogadjuk, továbbadjuk a hálózati rétegnek,


```
/* A 3. protokoll (PAR) egyirányú adatfolyamot biztosít megbízhatatlan csatornán. */

#define MAX_SEQ 1                                   /* 1-nek kell lennie a 3. protokollhoz */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include „protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;                        /* a következő kimenő keret sorszáma */
  frame s;                                          /* ideiglenes változó */
  packet buffer;                                    /* puffer a kimenő csomagnak */
  event_type event;

  next_frame_to_send = 0;                           /* inicializáld a kimenő sorszámot */
  from_network_layer(&buffer);                      /* kérd le az első csomagot */
  while (true) {
    s.info = buffer;                                /* készíts egy keretet a továbbításhoz */
    s.seq = next_frame_to_send;                     /* tedd bele a keretbe a sorszámot */
    to_physical_layer(&s);                          /* indítsd útnak */
    start_timer(s.seq);                             /* ha a válasz túl sokáig tart, legyen
                                                       időtúllépés */
    wait_for_event(&event);                         /* frame_arrival, cksum_err, timeout */
    if (event == frame_arrival) {
      from_physical_layer(&s);                      /* szerezd meg a nyugtát */
      if (s.ack == next_frame_to_send) {
        stop_timer(s.ack);                          /* állítsd le az időzítőt */
        from_network_layer(&buffer);                /* szerezd meg a következő elküldeni valót */
        inc(next_frame_to_send);                    /* inkrementáljuk a next_frame_to_send-et */
      }
    }
  }
}
void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
    wait_for_event(&event);                         /*lehetőségek: frame_arrival, cksum_err */
    if (event == frame_arrival) {                   /* egy érvényes keret érkezett */
      from_physical_layer(&r);                      /* szerezd meg az újonnan érkezett keretet */
      if (r.seq == frame_expected) {                /* ez az, amire vártál */
        to_network_layer(&r.info);                  /* add tovább az adatot a hálózati rétegnek */
        inc(frame_expected);                        /* inkrementáljuk a legközelebb várt sorszámot */
      }
      s.ack = 1 - frame_expected;                   /* mondd meg, hogy melyik keretet kell nyugtázni */
      to_physical_layer(&s);                        /* küldj nyugtát */
    }
  }
}
```

3.14. ábra. *Egy pozitív nyugtázás újraküldéssel protokoll*

és egy nyugtát küldünk. A duplikátumokat és a sérült kereteket nem adjuk át a hálózati rétegnek, viszont ezek után is küldünk nyugtát a legutoljára beérkezett hibátlan keretről, hogy az adó mihamarabb lehetőséget kapjon a továbbhaladásra, a következő csomag továbbítására, vagy a másolat elküldésére.

3.4. Csúszóablakos protokollok

Az eddigi protokollokban adatkeretek csak az egyik irányba haladtak. A legtöbb gyakorlati helyzetben viszont mindkét irányba kell adatokat átvinni. A duplex adatátvitel elérésének egyik módja az, hogy vesszük az előzőekben megismert protokollok közül az egyiknek két példányát úgy, hogy egyiket szimplex adatkapcsolatként az egyik irányba, a másikat szimplex adatkapcsolatként a másik irányba működtetjük. Mindkét adatkapcsolat így megvalósít egy „előremenő" (forward) csatornát (adatok számára) és egy „visszajövő" (reverse) csatornát (nyugták számára). A visszairányú csatorna sávszélessége mindkét esetben majdnem teljesen kihasználatlanul marad.

Jobb ötlet ugyanazt az adatkapcsolatot használni az adatok számára mindkét irányra. Végül is a 2. és 3. protokollban már mindkét irányba továbbítottunk kereteket, és a visszairányú csatornának normális körülmények között ugyanakkora kapacitása van, mint az előreirányúnak. Ebben a modellben az *A*-tól *B* felé tartó adatkeretek összekeverednek az *A*-tól *B* felé menő nyugtakeretekkel. A vevő a beérkező keret *kind* mezője alapján tudja megmondani, hogy a keret adat- vagy nyugtakeret-e.

Bár az adat- és a vezérlőkeretek átlapolódása egyetlen adatkapcsolaton történő továbbítás során nagy fejlődés ahhoz képest, mintha két különálló fizikai kapcsolat lenne, még van további fejlődési lehetőség. Amikor egy adatkeret megérkezik, ahelyett, hogy azonnal küldene egy külön vezérlőkeretet, a vevő türtőzteti magát, és megvárja, hogy a hálózati réteg átadja neki a következő csomagot. A nyugtát hozzácsatolja a kimenő adatkerethez (a fejrész *ack* mezőjét használva). Valójában a nyugta így ingyen utazik a következő kimenő adatkerettel. Az a módszer, amikor a kimenő nyugtákat átmenetileg késleltetjük, hogy rá tudjuk akasztani a következő kimenő adatkeretre, **ráültetésként** (**piggybacking**) ismert.

A ráültetés alkalmazásának fő előnye a különálló nyugtakeretekhez képest a csatorna rendelkezésre álló sávszélességének jobb kihasználása. Az *ack* mező a keret fejrészében csak néhány bitbe kerül, míg egy külön kerethez kellene saját fejrész, a nyugta maga és ellenőrző összeg. Ráadásul a kevesebb elküldött keret a vevő oldalán csökkenti a feldolgozási terhelést is. A következő protokollban, amit megvizsgálunk, a ráültetett nyugta mező csak egy bitet foglal el a keret fejrészében. Egyéb esetekben is ritkán foglal többet néhány bitnél.

A ráültetés azonban olyan komplikációt okoz, ami a különálló nyugtáknál nem jelentkezett. Meddig várakozzon az adatkapcsolati réteg a csomagra, amelyre ráülteti a nyugtát? Ha az adatkapcsolati réteg tovább vár, mint a küldő időzítési intervalluma, a küldő állomás a keretet újraküldi, meghiúsítva a nyugta célját. Ha az adatkapcsolati réteg jós lenne, és meg tudná jósolni a jövőt, meg tudná mondani, hogy mikor fog bejönni a következő csomag a hálózati rétegtől, és attól függően, hogy milyen hosszú

a várható várakozás, el tudná dönteni, hogy várjon-e, vagy azonnal küldjön egy külön nyugtakeretet. Természetesen az adatkapcsolati réteg nem tudja megjósolni a jövőt, így valamilyen *ad hoc* elgondolásra kell hagyatkoznia, mint például, hogy várjon rögzített számú milliszekundumot. Ha az új csomag hamar megérkezik, a nyugtát ráültetjük; ha viszont nem érkezik új csomag az időzítési intervallum végéig, az adatkapcsolati réteg elküld egy önálló nyugtakeretet.

A következő három protokoll a csúszóablakos (**sliding window**) protokollok osztályába tartozó, kétirányú protokoll. A három protokoll hatékonyság, bonyolultság és pufferigény kérdésében tér el egymástól, ahogy azt később meg is fogjuk tárgyalni. Ezekben, mint a többi csúszóablakos protokollban is, minden kimenő keret tartalmaz egy sorszámot, amely 0 és egy meghatározott legnagyobb érték között lehet. A maximum általában $2^n - 1$, hogy a sorszámot pontosan be lehessen illeszteni egy n bites mezőbe. A megáll-és-vár csúszóablakos protokoll $n = 1$-et használ, amellyel a sorszámokat a 0-ra és az 1-re korlátozza, de a fejlettebb változatok tetszőleges n-et is használni tudnak.

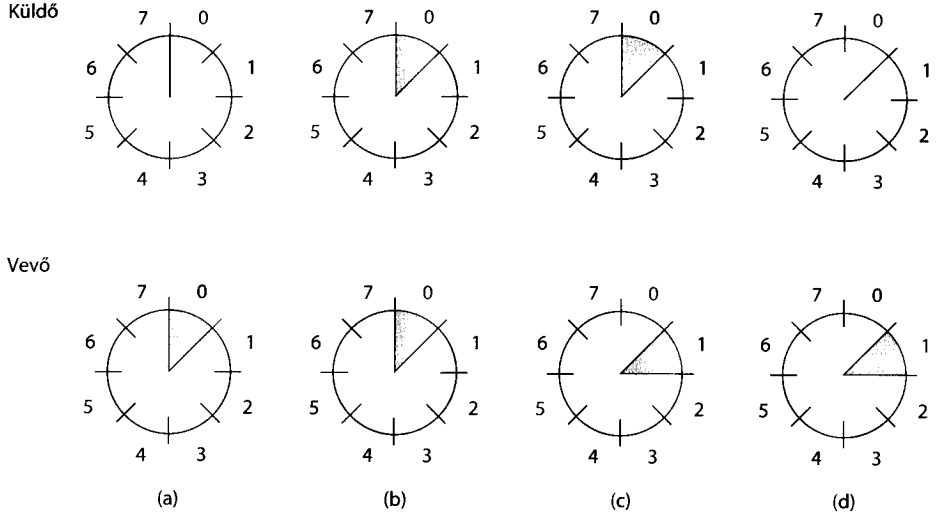
Minden csúszóablakos protokoll lényege az, hogy az adóállomás folyamatosan karbantart egy sorszámhalmazt, amely az elküldhető kereteknek felel meg. Azt mondjuk, hogy ezek a keretek az **adási ablakba** (**sending window**) esnek. Ehhez hasonlóan a vevő is karbantart egy **vételi ablakot** (**receiving window**), amely azon keretek halmazának felel meg, amelyeket befogadhat. Az adó és a vevő ablakainak nem kell azonos alsó és felső határral rendelkeznie, sőt a méretének sem kell megegyeznie. Néhány protokollban az ablakok rögzített méretűek, másokban azonban az idő előrehaladtával nőhetnek vagy csökkenhetnek, ahogyan a kereteket az állomások küldik és veszik.

Habár ezek a protokollok nagyobb szabadságot biztosítanak az adatkapcsolati rétegnek abban, hogy milyen sorrendben küldi és fogadja a kereteket, azért még határozottan nem mondtunk le arról az elvárásról, hogy a protokoll a csomagokat ugyanolyan sorrendben továbbítsa a címzett gép hálózati rétegének, mint amilyen sorrendben a küldő adatkapcsolati rétege azokat megkapta. Azt az elvárásunkat sem változtattuk meg, hogy a fizikai kommunikációs csatorna „vezetékszerű" legyen, vagyis minden keretet a küldés sorrendjében kell továbbítania.

A küldő ablakába eső sorszámok azokat a kereteket jelképezik, amelyeket már az adó elküldött, vagy amelyek elküldhetők, de a vevő még nem nyugtázta. Amikor egy új csomag érkezik a hálózati rétegtől, az megkapja a következő legmagasabb sorszámot, és az ablak felső széle eggyel előre ugrik. Amikor egy nyugta érkezik, akkor az ablak alsó széle lép egyet előre. Ezzel a módszerrel az ablak a még nem nyugtázott keretek listáját tartja folyamatosan karban. A módszerre a 3.15. ábra mutat egy példát.

Mivel azok a keretek, amelyek a küldő ablakában vannak, végső soron elveszhetnek vagy megsérülhetnek az átvitel során, a küldő állomásnak az összes ilyen keretet a memóriájában kell tartani az esetleges újraküldések miatt. Ezért, ha a maximális ablakméret n, a küldőnek n pufferre van szüksége a nyugtázatlan keretek megtartásához. Ha az ablak eléri a maximális méretét, a küldő adatkapcsolati rétegnek erőszakkal le kell kapcsolnia a hálózati réteget, amíg egy puffer fel nem szabadul.

A vevő adatkapcsolati rétegének ablaka azokhoz a keretekhez tartozik, amelyeket az adatkapcsolati réteg elfogadhat. Minden keret, amely az ablakon belülre esik, a vevő pufferébe kerül. Amikor egy olyan keret érkezik, melynek a sorszáma egyenlő az ablak alsó szélével, a vevő elfogadja, és átadja a hálózati rétegnek és az ablakot eggyel elforgatja. Az



3.15. ábra. *Egy 1-es méretű csúszóablak 3 bites sorszámmal. (a) Kezdetben. (b) Az első keret elküldése után. (c) Az első keret vétele után. (d) Az első nyugta vétele után*

olyan kereteket, amelyek az ablakon kívül esnek, eldobja. Az előbbi esetekben egy soron következő nyugtát is továbbít az adónak, hogy az folytatni tudja működését. Megjegyezzük, hogy az 1 méretű ablak azt jelenti, hogy az adatkapcsolati réteg csak sorrendben fogadja el a kereteket, de nagyobb ablakoknál ez nem így van! A hálózati réteget azonban mindig a megfelelő sorrendben táplálja adatokkal, tekintet nélkül arra, hogy mekkora az adatkapcsolati réteg ablakának mérete.

A 3.15. ábrán egy példa látható az 1-es maximális ablakméretre. Kezdetben egyetlen keret sincs kinn, így az adó ablakának alsó és felső szélei egyenlők, de ahogy az idő telik, a helyzet az ábrán látható módon változik. A vételi ablak mérete – az adási ablakkal ellentétben – minden esetben marad a kezdeti értéken, és előre lép, ahogy a következő keret megérkezik és továbbítódik a hálózati réteghez.

3.4.1. Egybites csúszóablakos protokoll

Az általános esettel való megbirkózás előtt vizsgáljunk meg egy olyan csúszóablakos protokollt, amelynek maximum 1 nagyságú ablaka lehet. Egy ilyen protokoll a megáll-és-vár technikát alkalmazza, mivel a küldő állomás elküld egy keretet és megvárja ennek nyugtáját, mielőtt a következőt elküldené.

A 3.16. ábra egy ilyen protokollt ábrázol. Hasonlóan a többihez, ez is néhány változó definiálásával kezdődik. A *next_frame_to_send* azt mutatja meg, hogy éppen melyik keretet próbálja az adóállomás elküldeni. Ehhez hasonlóan, a *frame_expected* azt mutatja meg, hogy a vevő melyik keretet várja. Mindkét változónál csak 0 és az 1 érték jöhet szóba.

Rendszerint a két adatkapcsolati réteg közül az egyik korábban kezd adni. Ezért csak az egyik adatkapcsolati réteg programjának kellene tartalmaznia a *to_physical_layer* és


```
/* A 4. protokoll (csúszóablakos) kétirányú. */

#define MAX_SEQ 1                       /* 1-nek kell lennie a 4. protokollhoz */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include „protocol.h"

void protocol4(void)
{
  seq_nr next_frame_to_send;            /* csak 0 vagy 1 lehet */
  seq_nr frame_expected;                /* csak 0 vagy 1 lehet */
  frame r, s;                           /* pufferek keretnek */
  packet buffer;                        /* az éppen küldendő csomag */
  event_type event;
  next_frame_to_send = 0;               /* következő keret a kimenő folyamban */
  frame_expected = 0;                   /* a várt érkező keret száma */
  from_network_layer(&buffer);          /* kérj le egy csomagot a hálózati rétegtől*/
  s.info = buffer;                      /* készítsd elő a kezdő keretet az elküldéshez */
  s.seq = next_frame_to_send;           /* tedd bele a sorszámot a keretbe */
  s.ack = 1 - frame_expected;           /* ráültetett nyugta */
  to_physical_layer(&s);                /* továbbítsd a keretet */
  start_timer(s.seq)                    /* indítsd el az időzítőt */
  while (true) {
    wait_for_event(&event);             /* frame_arrival, cksum_err vagy timeout */
    if (event == frame_arrival) {       /* egy keret érkezett sértetlenül */
      from_physical_layer(&r);          /* szerezd meg */
      if (r.seq == frame_expected) {    /* kezeljük a bejövő keretfolyamot */
        to_network_layer(&r.info);      /* add tovább a csomagot a hálózati rétegnek */
        inc(frame_expected);            /* inkrementáljuk a legközelebb várt sorszámot */
      }
      if (r.ack == next_frame_to_send) { /* kezeld a kimenő keretfolyamot */
        stop_timer(r.ack);              /* állítsd le az időzítőt */
        from_network_layer(&buffer);    /* kérd le a következő csomagot a hálózati
                                           rétegtől */
        inc(next_frame_to_send);        /* inkrementáljuk a next_frame_to_send-et */
      }
    }
    s.info = buffer;                    /* készítsd el a kimenő keretet */
    s.seq = next_frame_to_send;         /* tedd bele a keretbe a sorszámot */
    s.ack = 1 - frame_expected;         /* a legutóbb vett keret sorszáma */
    to_physical_layer(&s);              /* továbbítsd a keretet */
    start_timer(s.seq);                 /* indítsd el az időzítőt */
  }
}
```

3.16. ábra. *Egy egybites csúszóablakos protokoll*

a *start_timer* eljáráshívásokat a fő cikluson kívül. A kezdő gép lekéri az első csomagot a hálózati rétegétől, egy keretet épít belőle, és elküldi azt. Amikor ez a keret (vagy bármelyik) megérkezik, a vevő adatkapcsolati rétege – pontosan úgy, mint a 3. protokollban – megnézi, hogy duplikátum-e. Ha a keret az, amelyiket várta, átadja a hálózati rétegnek, és a vevő ablakát feljebb csúsztatja.

A nyugta mező a legutolsó hibátlanul vett keret sorszámát tartalmazza. Ha ez a szám megegyezik annak a keretnek a sorszámával, amit az adó próbál elküldeni, az adó tudja,

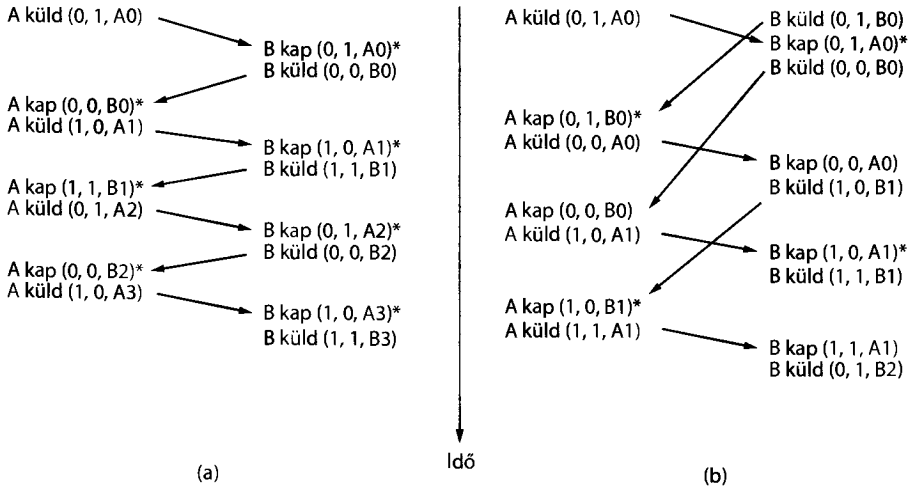
hogy végzett a *buffer*-ben tárolt csomaggal, és lekérheti a következőt a hálózati rétegtől. Ha a sorszám nem egyezik, folytatnia kell a próbálkozást ugyanazzal a kerettel. Minden alkalommal, amikor egy keret érkezik, egyet el is küld.

Most vizsgáljuk meg a 4. protokollt, hogy lássuk, mennyire ellenálló hibás eseménysorozatok esetén! Tételezzük fel, hogy *A* a 0-s keretét próbálja elküldeni *B*-nek, és *B* szintén a 0-s keretét próbálja elküldeni *A*-nak. Tegyük fel, hogy *A* elküldi a keretet *B*-nek, de az időzítési intervalluma egy kicsit rövidebb, mint kellene. Ezért több ízben lejárhat az időzítője, így egy sor azonos keretet küldhet, mindet *seq* = 0-val és *ack* = 1-gyel.

Amikor az első érvényes keret megérkezik *B*-be, az elfogadja és a *frame_expected*-et 1-re állítja. Az összes ezt követő keretet elutasítja, mert *B* most olyan keretet vár, melynek a sorszáma 1, nem pedig 0. Továbbá, mivel minden duplikátumnál *ack* = 1 és *B* még 0-s nyugtára vár, *B* nem fog lekérni újabb csomagot a hálózati rétegétől.

Miután mindegyik elutasítandó duplikátum megérkezett, *B* küld *A*-nak egy keretet *seq* = 0-val és *ack* = 0-val. Végül is ezek közül az egyik megérkezik *A*-hoz, és *A* elkezdi küldeni a következő csomagot. Az elveszett keretek vagy idő előtti időtúllépések semmilyen kombinációja nem tudja azt okozni, hogy a protokoll akár megkettőzött csomagokat küldjön a hálózati rétegnek, akár kihagyjon egy csomagot, vagy akár holtpontba kerüljön, tehát a protokoll megfelelően működik.

Hogy megmutassuk, milyen körmönfont események történhetnek egy protokollal, kiemelünk egy sajátos helyzetet: ha mindkét oldal egyszerre küldi el a kezdeti csomagot. Ez a szinkronizációs probléma látható a 3.17. ábrán. Az (a) részben a protokoll normális működése figyelhető meg. A (b) rész a sajátos helyzetet illusztrálja. Ha *B* megvárja *A* első keretét, mielőtt a sajátjai közül egyet elküldene, az eseménysorozat olyan, mint az (a) részen látható, és minden keretet elfogadunk.



3.17. ábra. *Két eseménysorozat a 4. protokollhoz. (a) Normális működés. (b) Hibás működés. A jelölés: (sorszám, nyugtaszám, csomagsorszám). A csillag azt jelzi, hogy az adatkapcsolati réteg továbbította a keretet a hálózati rétegnek*

Ha azonban *A* és *B* egyszerre indítja el a kommunikációt, az első kereteik keresztezik egymást, és az adatkapcsolati réteg a (b) részen látható helyzetbe kerülhet. Az (a) ábrán minden keret érkezése új csomagot hoz a hálózati rétegnek, nincsenek duplikátumok. A (b) ábrán a keretek fele duplikátumot tartalmaz, még akkor is, ha nincsenek átviteli hibák. Hasonló helyzetek alakulhatnak ki idő előtti időtúllépések eredményeként, még akkor is, ha az indulásnál egyértelműen csak az egyik állomás kezd adni. Ha többszörös korai időtúllépés következik be, a keretek háromszor vagy még többször is elküldhetők, ami értékes sávszélesség vesztést eredményez.

3.4.2. Az n visszalépést alkalmazó protokoll

Az eddigiekben azzal a hallgatólagos feltételezéssel éltünk, hogy egy keret vevőhöz való megérkezéséhez és a nyugta visszaérkezéséhez együttesen szükséges idő elhanyagolható. Néha ez a feltételezés egyáltalán nem helytálló. Ezekben a szituációkban a hosszú körülfordulási idő fontos következményeket von maga után a sávszélesség kihasználtságára nézve. Példának okáért, vegyünk egy 50 kb/s-os műholdas csatornát 500 ms-os körülfordulási idővel. Képzeljük el, hogy a 4. protokollt szeretnénk használni 1000 bites keretek küldésére a műholdon keresztül. A $t = 0$ időpillanatban az adóállomás elkezdi küldeni az első keretet. A $t = 20$ ms-ban már a teljes keretet elküldte. Nem előbb, mint $t = 270$ ms érkezik meg a teljes keret a vevőhöz, és nem előbb, mint $t = 520$ ms érkezik vissza a nyugta az adóállomáshoz a legjobb esetben (nincs várakozás a vevőben, és rövid a nyugtakeret). Ez azt jelenti, hogy az adó az idő 500/520 részében, azaz 96%-ában blokkolva van (tehát a rendelkezésre álló sávszélességnek csupán a 4%-át használja ki). Tisztán látszik, hogy a hosszú átviteli idő, a nagy sávszélesség és a rövid keretek kombinációja végzetes a hatékonyságra nézve.

A fent leírt problémát azon szabály következményének tekinthetjük, amely megköveteli az adóállomástól, hogy megvárja az elküldött keret nyugtáját, mielőtt egy másik keretet küldene. Ha lazítunk ezen a korlátozáson, sokkal nagyobb hatékonyságot érhetünk el. A megoldás alapvetően arra épül, hogy megengedjük az adónak, hogy a blokkolódás előtt 1 helyett legfeljebb *w* darab keretet elküldjön. Elegendően nagy *w* megválasztásával az adó folyamatosan küldhet kereteket, hiszen a már elküldött keretekre a nyugták azelőtt megérkeznek, mielőtt az ablak betelne, tehát az adó nem blokkolódik.

A megfelelő *w* megválasztásához tudnunk kell, hogy az adótól a vevőig történő terjedés során hány csomag fér el a kommunikációs csatornán egy időben. Ezt a kapacitást bit mértékegységben meghatározhatjuk a csatorna sávszélességének és egyirányú késleltetésének a szorzatával, más néven a **sávszélesség-késleltetés szorzattal (bandwidth-delay product)**. Ha ezt a mennyiséget elosztjuk a keret bitekben mért hosszával, akkor megkapjuk, hány keret lehet kinn a csatornán. Nevezzük ezt a mennyiséget *BD*-nek. Ebben az esetben a *w*-t $2BD + 1$ értékűre kell választani. A *BD* kétszerese azt mutatja, hogy hány keret lehet a csatornán, ha az adó folyamatosan adja a kereteket, figyelembe véve, hogy a nyugta megérkezésére egy körülfordulásnyi idő szükséges. A „+1" veszi figyelembe azt a tényt, hogy a nyugta elküldése csak egy teljes keret beérkezése után lehetséges.

Az előző példánkban, ahol a sávszélesség 50 kb/s és az egyirányú késleltetés 250 ms, a sávszélesség-késleltetés szorzat 12,5 kilobit, amely 1000 bit hosszúságú keretekkel szá-

molva 12,5 keretnek felel. A $2BD+1$ kifejezés értéke tehát 26 keret. Tegyük fel, hogy az adó kezdetben a 0. keretet, majd minden 20 ms múlva egy újabb keretet küld. Mire végez 26 keret küldésével, a $t=520$ ms időpontban éppen megérkezik a 0. keret nyugtája. Így tehát minden 20 ms múlva érkezik egy újabb nyugta, amely engedélyt ad az adónak a további keretek küldésére. A továbbiakban 25 vagy 26 nyugtázatlan keret mindig a csatornán lesz. Más szavakkal, az adó maximális ablakmérete 26.

Kisebb ablakméretekre a kapcsolat kihasználtsága kisebb lesz mint 100%, mivel az adó néha blokkolt állapotba kerül. Az adatkapcsolat kihasználtsága az idő azon része, amelyben az adó nincs blokkolt állapotban:

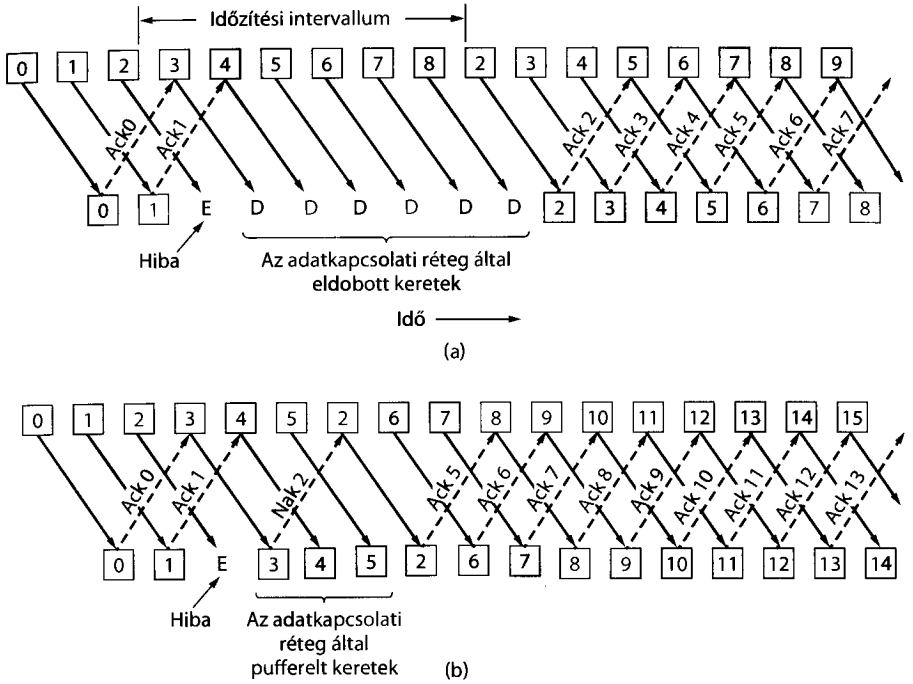
$$\text{Kihasználtság} \leq \frac{w}{1+2BD}$$

Mivel ez az érték a keret feldolgozási idejét elhanyagolja, és a nyugták hosszát 0-nak veszi (ugyanis azok általában rövidek), ezért csupán egy felső határt állapít meg. Az egyenlet viszont jól mutatja, hogy ha a sávszélesség-késleltetés szorzat nagy, akkor szükségszerűen az ablakméretet is nagyra kell választani. Ha a késleltetés nagy, az adó hamar elfogyasztja az ablakméretét – még közepes sávszélesség esetén is –, ahogy a műholdas csatorna esetén láttuk. Hasonlóan, ha a sávszélesség nagy, akkor az adó szintén gyorsan kiüríti az adási ablakot – akár közepes késleltetés esetén is –, hacsak nem használ nagy ablakméretet (például egy 1 Gb/s adatkapcsolat 1 ms késleltetéssel 1 megabites szorzattal rendelkezik). A megáll-és-vár protokoll $w=1$ értéke még egy keretnyi késleltetési idő esetén is kevesebb mint 50% kihasználtságot eredményez.

Azt a módszert, amely több csomagot is mozgásban tart a csatornán, **csővezetékezésnek (pipelining)** nevezik. A csővezetékezés alkalmazása megbízhatatlan kommunikációs csatornán néhány komoly kérdést vet fel. Először is: mi történik, ha egy hosszú folyam közepén egy keret megsérül vagy elveszik? Mielőtt az adó egyáltalán észrevenné, hogy valami nincs rendben, nagyszámú további keret érkezik meg a vevőhöz. Amikor egy sérült keret érkezik a vevőhöz, nyilvánvalóan el kell dobnia, de mit csináljon az azt követő hibátlan keretekkel? Ne feledkezzünk meg róla, hogy a vevő adatkapcsolati rétegének mindenképpen a helyes sorrendben kell a hálózati rétegnek átadnia a csomagokat. A hiba kezelésére csővezetékezés esetén két alapvető megközelítés létezik, melyeket a 3.18. ábrán mutatunk be.

Az egyik megközelítést **n visszalépéses (go-back-n)** eljárásnak nevezik. Ekkor a vevő eldobja az összes keretet, amely a hibás után érkezik, és nem küld nyugtát róluk, ahogy az a 3.18.(a) ábrán látható. Ez a stratégia az 1 hosszúságú vételi ablaknak felel meg. Más megközelítésből, az adatkapcsolati réteg elutasít minden keretet, kivéve a soron következőt, amit a hálózati rétegnek át kell adnia. Ha az adó ablaka betelik mielőtt az időzítő lejár, a csővezeték elkezd kiürülni. Végül is az adónak lejár az időzítője, és újraküldi az összes nyugtázatlan keretet, kezdve a sérült vagy elveszett kerettel. Ez a megközelítés nagy sávszélességet pazarolhat el, ha nagy a hibaarány.

Amikor a vevő ablakának mérete nagy, akkor a 0-s és 1-es sorszámú kereteket a vevő helyesen veszi és nyugtázza. A 2-es sorszámú keret azonban megrongálódott vagy elveszett. Az adó ezt nem tudva, folytatja a további keretek küldését, amíg a 2-es keret időzítése le nem jár. Ekkor visszamegy a 2-es kerethez, és onnan kezdi újra az átvitelt, a



3.18. ábra. *Csővezetékezés és hibakezelés. Egy hiba hatása, (a) ha a vételi ablak mérete 1 (n-nel történő visszalépés esetén), (b) ha a vételi ablak mérete 1-nél nagyobb (szelektív ismétlés esetén)*

2-es, 3-as, 4-es stb. sorszámú kereteket újra elküldve. A 0-s és 1-es sorszámú kereteket a vevő helyesen veszi és nyugtázza. A 2-es sorszámú keret azonban megrongálódott vagy elveszett. Az adó ezt nem tudva, folytatja a további keretek küldését, amíg a 2-es keret időzítése le nem jár. Ekkor visszamegy a 2-es kerethez, és onnan kezdi újra az átvitelt, a 2-es, 3-as, 4-es stb. sorszámú kereteket újra elküldve.

A csővezetékezett keretek esetén alkalmazható másik általános hibakezelési stratégia a **szelektív ismétlés (selective repeat)**. Ennek a módszernek a használatakor a vevő a rosszul vett kereteket eldobja, de az ezután érkező jó kereteket tárolja egy pufferben. Amikor az adó időzítése lejár, csak a legrégebbi nyugtázatlan keretet küldi el újra. Ha ez a keret helyesen megérkezik, akkor a vevő a helyes sorrendben adja tovább a hálózati rétegnek az összes, addig pufferelt keretet is. A szelektív ismétlés módszere egyenértékű azzal, hogy a vevő ablakmérete nagyobb mint 1. Ez a megközelítés nagy ablakméretekre nagy mennyiségű memóriát igényel az adatkapcsolati rétegben.

A szelektív ismétlést gyakran alkalmazzák együtt azzal a megoldással, hogy a vevő **negatív nyugtát (negative acknowledgement, NAK)** küld, amikor hibát észlel, például hibás ellenőrző összeg vagy nem soron következő keret esetén. A negatív nyugták még azelőtt kikényszerítik az újraküldést, hogy a megfelelő időzítő lejárna, ezért javítják a rendszer hatásfokát.

A 3.18.(b) ábrán a 0-s és az 1-es sorszámú keret ismét helyesen megérkezik, de a 2-es sorszámú keret elveszett. Amikor a 3-as keret megérkezik a vevőhöz, az adatkapcsolati rétege

```
/* Az 5. protokoll (n visszalépéses) megengedi, hogy több keret legyen úton. Az adó leadhat
   MAX_SEQ darab keretet anélkül, hogy nyugtára várna. Ráadásul nem feltételezzük a
   hálózati rétegről, hogy mindig van új csomagja. Ehelyett a hálózati réteg network_layer_
   ready eseményt okoz, amikor van elküldendő kerete.*/

#define MAX_SEQ 7
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include „protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* true-t ad vissza, ha a <= b < c körkörösen; egyébként false-t */
   if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
        return(true);
     else
        return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
/* Építsünk fel és küldjünk el egy adatkeretet. */
   frame s;                                          /* átmeneti változó */

   s.info = buffer[frame_nr];                        /* tedd be a csomagot a keretbe */
   s.seq = frame_nr;                                 /* tedd be a sorszámot a keretbe */
   s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);      /* ültessük rá a nyugtát */
   to_physical_layer(&s);                            /* továbbítsd a keretet */
   start_timer(frame_nr);                            /* indítsd el az időzítőt */
}

void protocol5(void)
{
   seq_nr next_frame_to_send;          /* MAX_SEQ > 1; a kimenő keretekhez használjuk */
   seq_nr ack_expected;                /* a legrégebbi keret ami még nincs nyugtázva */
   seq_nr frame_expected;              /* a következő várt keret a bejövő folyamban */
   frame r;                            /* átmeneti változó */
   packet buffer[MAX_SEQ + 1];         /* pufferek a kimenő folyamhoz */
   seq_nr nbuffered;                   /* a használatban levő kimeneti pufferek */
   seq_nr i;                           /* a puffertömb indexelésére használjuk */
   event_type event;

   enable_network_layer();          /* engedélyezd a network_layer_ready eseményeket */
   ack_expected = 0;                   /* a következőként várt bejövő nyugta */
   next_frame_to_send = 0;             /* következő kimenő keret */
   frame_expected = 0;                 /* a várt érkező keret száma */
   nbuffered = 0;                      /* kezdetben egyetlen csomag sincs pufferelve */

while (true) {
     wait_for_event(&event);           /* négy lehetőség: lásd az event_type-ot fent */
     switch(event) {
        case network_layer_ready:      /* a hálózati rétegnek van elküldeni való csomagja */
                                       /* fogadj, ments el és továbbíts egy új keretet. */
```

3.19. ábra. *Egy n visszalépéses eljárást használó csúszóablakos protokoll*


```
        from_network_layer(&buffer[next_frame_to_send]);   /* kérj le új csomagot */
        nbuffered = nbuffered + 1;         /* növeld az adó ablakát */
        send_data(next_frame_to_send, frame_expected, buffer); /* továbbítsd a keretet */
        inc(next_frame_to_send);           /* léptesd az adó ablakának felső szélét */
        break;

      case frame_arrival:                  /* egy adat-, vagy vezérlőkeret érkezett */
        from_physical_layer(&r);           /* szerezd meg a bejövő keretet a fizikai rétegtől */

        if (r.seq == frame_expected) {
                                           /* a kereteket csak sorrendben fogadd el. */
          to_network_layer(&r.info);       /* add át a csomagot a hálózati rétegnek */
          inc(frame_expected);             /* léptesd a vevő ablakának alsó szélét */
        }

        /* az n. nyugta magában foglalja az n – 1.-t, n – 2.-t, stb. Ellenőrizd ezt. */
        while (between(ack_expected, r.ack, next_frame_to_send))
        {
                                           /*kezeld a ráültetett nyugtát. */
          nbuffered = nbuffered - 1;        /* eggyel kevesebb van pufferelve */
          stop_timer(ack_expected);        /* keret érkezett sértetlenül; állítsuk le az időzítőt */
          inc(ack_expected);               /* húzd összébb az adó ablakát */
        }
        break;

      case cksum_err: break;               /* egyszerűen hagyd figyelmen kívül a hibás */
                                           /* kereteket */

      case timeout:                        /* baj van; küldd újra az összes kint levő keretet */
        next_frame_to_send = ack_expected;          /* kezdd itt az újraküldést */
        for (i = 1; i <= nbuffered; i++) {
          send_data(next_frame_to_send, frame_expected, buffer);   /* küldd újra */
                                                                   /* 1 keretet */
          inc(next_frame_to_send);         /* készülj fel a következő küldésére */
        }

    }

    if (nbuffered < MAX_SEQ)
      enable_network_layer();
    else
      disable_network_layer();
  }
}
```

3.19. ábra. *Egy n visszalépéses eljárást használó csúszóablakos protokoll (folytatás)*

észreveszi, hogy kimaradt egy keret, ezért elküld egy NAK-ot a 2-esről, de puffereli a 3-ast. Amikor a 4-es és 5-ös keret megérkezik, az adatkapcsolati réteg ezeket is puffereli ahelyett, hogy továbbadná őket a hálózati rétegnek. Előbb-utóbb a 2-es NAK visszaér az adóhoz, amely azonnal újraküldi a 2-es keretet. Amikor ez megérkezik, az adatkapcsolati réteg már rendelkezik a 2-es, a 3-as, a 4-es és az 5-ös kerettel, így azokat a megfelelő sorrendben

tovább is adhatja a hálózati rétegnek. Ezenfelül nyugtázhatja az összes keretet az 5-ösig bezárólag, amint az ábrán is látható. Ha a NAK elveszik, az adó időzítése a 2-es keretre előbb-utóbb lejár és magától ismét elküldi azt (és csak azt), de ez akár sokkal később is lehet.

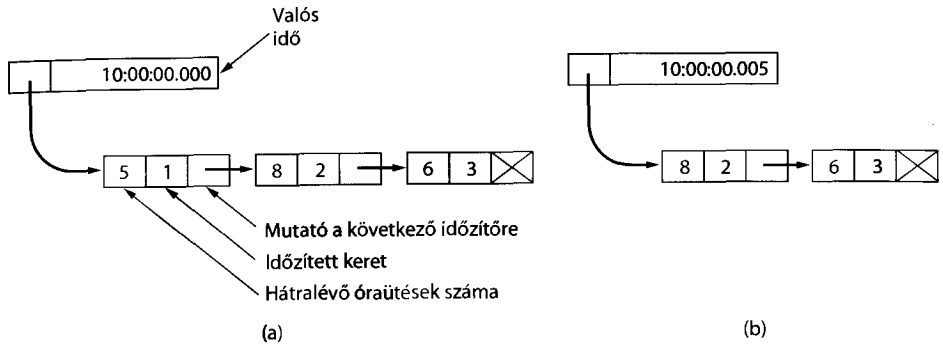
Ez a két megközelítés lehetőséget ad arra, hogy kompromisszumot kössünk a sávszélesség-kihasználás és az adatkapcsolati rétegbeli memória mérete között. Attól függően, hogy melyik erőforrás az értékesebb, az egyik vagy a másik megközelítés használható. A 3.19. ábrán (lásd 258-259. oldal) egy olyan n visszalépéses protokoll látható, amelyben az adatkapcsolati réteg csak sorrendben fogadja el a kereteket; egy hibás keretet követően az összes keretet eldobja. Ebben a protokollban – első ízben – szakítottunk azzal a feltételezéssel, hogy a hálózati réteg végtelen sok csomagot tud az adatkapcsolati réteg rendelkezésére bocsátani. Amikor a hálózati rétegnek van egy csomagja, amit el akar küldeni, ez egy *network_layer_ready* eseményt okozhat. Annak érdekében azonban, hogy forgalomszabályozási megkötéseket tegyünk az adási ablakra vagy a nyugtázatlan keretek számára, az adatkapcsolati rétegnek meg kell tudnia akadályozni a hálózati réteget abban, hogy az több munkával zaklassa. Az *enable_network_layer* és a *disable_network_layer* könyvtári rutinok végzik ezt a feladatot.

A tetszőleges pillanatban maximálisan kint lévő keretek száma nem egyezik meg a sorszámok maximális méretével. Az n visszalépéses protokollra legfeljebb *MAX_SEQ* keret lehet kint, annak ellenére, hogy *MAX_SEQ* + 1 különböző sorszám van (ezek: 0, 1, 2, ..., *MAX_SEQ)*. Látni fogjuk, hogy a szelektív ismétlés protokollra még szorosabb megkötés tartozik. Hogy megértsük, miért szükséges ez a korlátozás, vegyük a következő eseménysort *MAX_SEQ* = 7 mellett.

1. Az adóállomás elküldi a kereteket a 0.-tól a 7.-ig.
2. A 7. keretre végül is visszajön az adóállomáshoz egy ráültetett nyugta.
3. Az adóállomás újabb nyolc keretet küld, megint 0 és 7 közötti sorszámokkal.
4. Most még egy ráültetett nyugta érkezik a 7. keretre.

A kérdés: a második sorozathoz tartozó nyolc keret mind megérkezett sikeresen, vagy mind a nyolc elveszett (a hibás keretet követő keretek eldobását is keretvesztésnek számítva). A vevőnek mindkét esetben a 7. keretre kellene nyugtát küldenie. Az adóállomás semmilyen módon nem tudja a kérdést eldönteni. Emiatt a kint levő keretek száma legfeljebb *MAX_SEQ* lehet.

Bár az 5. protokoll nem puffereli az érkező kereteket egy hibás keret után, mégsem ússza meg teljesen a pufferelés problémáját. Mivel lehet, hogy az adóállomásnak újra kell küldenie az összes nyugtázatlan keretet egy későbbi időpontban, meg kell tartania az összes elküldött keretet addig, amíg biztosan nem tudja, hogy a vevő elfogadta azokat. Amikor nyugta jön az *n*. keretre, az $n-1$., $n-2$. stb. szintén automatikusan nyugtázódik. Az ilyen típusú nyugtázás **halmozott nyugtázás (cumulative acknowledgement)** néven ismert. Ez a tulajdonság különösen fontos akkor, amikor néhány előző nyugtahordozó keret elveszik vagy tönkremegy. Mindig, amikor egy nyugta beérkezik, az adatkapcsolati réteg ellenőrzi, hogy fel lehet-e szabadítani valamelyik puffert. Ha fel



3.20. ábra. *Több időzítő szimulációja szoftverből. (a) A sorba kötött időzítők. (b) Az első időzítő lejárta utáni helyzet*

lehet szabadítani puffereket (azaz van hely az ablakban), a korábban blokkolt hálózati rétegnek engedélyezni lehet, hogy további *network_layer_ready* eseményeket okozzon.

Ennél a protokollnál feltesszük, hogy mindig van forgalom az ellenkező irányba is, vagyis van mire ráültetni a nyugtákat. A 4. protokoll esetében nincsen szükség erre a feltételezésre, mivel az minden egyes keret vételekor elküld egy keretet, még akkor is, ha valamivel korábban azt a keretet már elküldte. A következő protokollban egy elegáns módszert mutatunk az egyirányú forgalom problémájának megoldására.

Mivel az 5. protokollban több kint levő keret van, logikailag több időzítőre van szükség: minden kint levő kerethez egyre. Minden keret időzítője az összes többiétől függetlenül jár le. Az összes időzítő könnyedén szimulálható szoftverből egy olyan hardveridőzítővel, amelyik periodikusan megszakításokat okoz. A járó időzítők egy láncolt listát alkotnak, melyben minden csomópont azt mutatja, hogy mennyi óraütés van hátra, amíg az időzítő lejár, és hogy melyik kerethez tartozik az adott időzítő. A csomópont ezenkívül tartalmaz még egy mutatót is a következő csomópontra.

Illusztrációként arra, hogy hogyan lehet megvalósítani az időzítőket, vegyük a 3.20.(a) ábra példáját. Tételezzük fel, hogy az óra minden 1 ms-ban üt egyet. Kezdetben a valós idő 10:00:00.000, és három függőben levő időzítés van: 10:00:00.005-kor, 10:00:00.013-kor és 10:00:00.019-kor. Mindig, amikor a hardveróra üt, a valós időt aktualizáljuk, és az ütésszámlálót a lista fejében csökkentjük eggyel. Amikor az ütésszámláló nulla lesz, az időzítő időtúllépést okoz, és a csomópontot eltávolítjuk a listából, ahogyan a 3.20.(b) ábrán látható. Bár ez a szervezés azt kívánja, hogy a listát végignézzük, amikor a *start_timer*-t vagy a *stop_timer*-t meghívjuk, óraütésenként nem kell sok mindent csinálni. Az 5. protokollban mindkét rutinnak egy paramétert adtunk át, mely azt jelzi, hogy melyik kerethez tartozik az időzítő.

3.4.3. Szelektív ismétlést alkalmazó protokoll

Az n visszalépéses protokoll jól működik akkor, ha a hibák ritkán fordulnak elő. Ha azonban a vonal gyenge minőségű, nagy sávszélességet pazarol el a keretek újraküldé-

```
/* A 6. protokoll (szelektív ismétlés) elfogadja a nem soron következő kereteket is, de a
   csomagokat sorrendben továbbítja a hálózati rétegnek. Minden úton levő kerethez tartozik
   egy időzítő. Amikor az időzítő lejár, csak a hozzá tartozó keretet küldjük újra, nem az összes
   kintlévőt, mint az 5. protokollban. */

#define MAX_SEQ 7                       /* 2^n – 1-nek kell lennie */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout}
event_type;
#include „protocol.h"
boolean no_nak = true;                  /* még nem küldtünk negatív nyugtát */
seq_nr oldest_frame = MAX_SEQ + 1;      /* a kezdeti érték csak a szimulátor számára érdekes */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Ugyanaz, mint a between az 5. protokollban, de rövidebb és nehezebben érthető */
   return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected, packet
buffer[])
{
/* Építsünk fel és küldjünk el egy adat-, nyugta- vagy negatív nyugtakeretet. */
   frame s;                                          /* átmeneti változó */

   s.kind = fk;                                      /* kind == data, ack vagy nak */
   if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
   s.seq = frame_nr;                                 /* csak adatkereteknél van jelentése */
   s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
   if (fk == nak) no_nak = false;                    /* keretenként egy negatív nyugta */
   to_physical_layer(&s);                            /* továbbítsd a keretet */
   if (fk == data) start_timer(frame_nr % NR_BUFS);
   stop_ack_timer();                                 /* nincs szükség külön nyugtakeretre */
}

void protocol6(void)
{
   seq_nr ack_expected;                /* az adó ablakának alsó széle */
   seq_nr next_frame_to_send;          /* az adó ablakának felső széle + 1 */
   seq_nr frame_expected;              /* a vevő ablakának alsó széle */
   seq_nr too_far;                     /* a vevő ablakának felső széle +1 */
   int i;                              /* index a pufferparkhoz */
   frame r;                            /* átmeneti változó */
   packet out_buf[NR_BUFS];            /* pufferek a kimenő folyamhoz */
   packet in_buf[NR_BUFS];             /* pufferek a bejövő folyamhoz */
   boolean arrived[NR_BUFS];           /* bejövő keretek bittérképe */
   seq_nr nbuffered;                   /* hány kimeneti puffer van jelenleg használatban */
   event_type event;

   enable_network_layer();             /* inicializálás */
   ack_expected = 0;                   /* a következőként várt bejövő nyugta */
```

3.21. ábra. *Egy szelektív ismétlést alkalmazó csúszóablakos protokoll*


```
   next_frame_to_send = 0;             /* következő kimenő keret */
   frame_expected = 0;
   too_far = NR_BUFS;
   nbuffered = 0;                      /* kezdetben egyetlen csomag sincs pufferelve */
   for (i = 0; i < NR_BUFS; i++) arrived[i] = false;

   while (true) {
      wait_for_event(&event);          /* öt lehetőség: lásd az event_type-ot fent */
      switch(event) {
         case network_layer_ready:     /* fogadj, ments el és továbbíts egy új keretet. */
            nbuffered = nbuffered + 1; /* növeld az ablakot */
            from_network_layer(&out_buf[next_frame_to_send % NR_BUFS]);
                                       /* kérj le új csomagot */
            send_frame(data, next_frame_to_send, frame_expected, out_buf);
                                       /* továbbítsd a keretet */
            inc(next_frame_to_send);   /* léptesd az adó ablakának felső szélét */
            break;

         case frame_arrival:                       /* egy adat- vagy vezérlőkeret érkezett */
            from_physical_layer(&r);               /* kérd le a bejövő keretet a fizikai rétegtől */
            if (r.kind == data) {
                                                   /* Egy sértetlen keret érkezett. */
               if (r.seq != frame_expected && no_nak )
                  send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
               if (between(frame_expected, r.seq, too_far) && (arrived[r.seq % NR_BUFS] == false)) {
                     /* A kereteket bármilyen sorrendben el lehet fogadni. */
                  arrived[r.seq % NR_BUFS] = true;          /* jelöld meg a puffert teliként */
                  in_buf[r.seq % NR_BUFS] = r.info;         /* tedd az adatot a pufferbe */
                  while (arrived[frame_expected % NR_BUFS]) {
                        /* add át a kereteket és léptesd az ablakot */
                     to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                     no_nak = true;
                     arrived[frame_expected % NR_BUFS] = false;
                     inc(frame_expected);  /* léptesd a vevő ablakának alsó szélét */
                     inc(too_far);         /* léptesd a vevő ablakának felső szélét */
                     start_ack_timer();    /* hogy lássuk, hogy szükséges-e külön nyugta */
                  }
               }
            }
            if ((r.kind==nak) && between(ack_expected,(r.ack+1)%(MAX_SEQ+1),
            next_frame_to_send))
               send_frame(data, (r.ack + 1) % (MAX_SEQ + 1), frame_expected, out_buf);

            while (between(ack_expected, r.ack, next_frame_to_send)) {
               nbuffered = nbuffered – 1;                 /* kezeld a ráültetett nyugtát */
               stop_timer(ack_expected % NR_BUFS);        /* ép keret érkezett */
               inc(ack_expected);                         /* léptesd az adó ablakának alsó szélét */
            }
            break;
         case cksum_err:
            if (no_nak) send_frame(nak, 0, frame_expected, out_buf);   /* sérült keret */
            break;
```

3.21. ábra. *Egy szelektív ismétlést alkalmazó csúszóablakos protokoll (folytatás)*

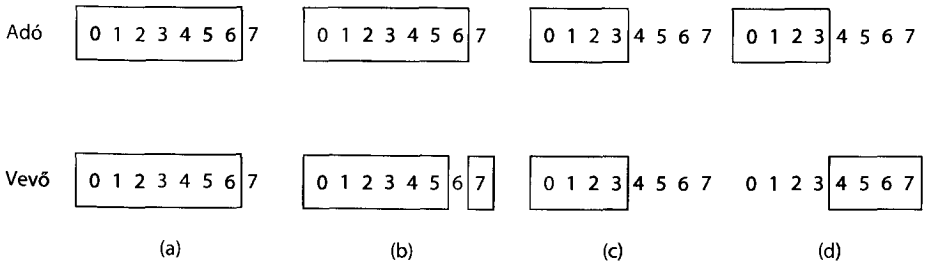
```
        case timeout:
          send_frame(data, oldest_frame, frame_expected, out_buf); /* lejárt az időzítőnk */
          break;
        case ack_timeout:
          send_frame(ack, 0, frame_expected, out_buf);    /* a nyugta időzítő lejárt, küldj */
                                                          /* nyugtát */
      }
      if (nbuffered < NR_BUFFS) enable_network_layer(); else disable_network_layer();
    }
}
```

3.21. ábra. *Egy szelektív ismétlést alkalmazó csúszóablakos protokoll (folytatás)*

sére. Egy másik stratégia – a szelektív ismétlés – szerint megengedjük a vevőnek, hogy elfogadja és pufferelje a kereteket, melyek egy elveszett vagy megsérült keretet követnek.

Ebben a protokollban az adó karbantart egy ablakot a kint lévő keretek sorszámairól, a vevő pedig egy ablakot az elfogadható sorszámokból. Az adó ablakának mérete 0-ról indul és valamilyen előre definiált maximumig növekszik. A vevő ablaka ezzel ellentétben rögzített méretű, és egy előre meghatározott értékkel egyenlő. A vevőnek minden, az ablakában levő sorszámhoz fenntartott puffere van. Mindegyik pufferhez tartozik egy bit (*arrived*), mely azt mutatja, hogy a puffer tele van-e vagy üres. Amikor egy keret megérkezik, a *between* függvénnyel ellenőrzi a sorszámát, hogy beleesik-e az ablakba. Ha beleesik, és a keretet még nem vette korábban, akkor elfogadja és tárolja. Ezt mindig megteszi, tekintet nélkül arra, hogy a keret azt a csomagot tartalmazza-e, amit a hálózati réteg vár, vagy sem. Természetesen, a keretet meg kell őrizni az adatkapcsolati rétegben, és nem szabad átadni a hálózati rétegnek addig, amíg az összes alacsonyabb sorszámú keretet nem továbbította a hálózati rétegnek a helyes sorrendben. Az ezt az algoritmust használó protokoll a 3.21. ábrán látható (262-264. oldal).

A keretek nem sorrendben történő vétele további megkötéseket ad a sorszámokhoz, szemben azokkal a protokollokkal, melyek a kereteket csak sorrendben fogadják el. A problémát legkönnyebben egy példával illusztrálhatjuk. Tételezzük fel, hogy 3 bites sorszámunk van, így az adóállomás legfeljebb hét keretet továbbíthat, mielőtt várakoznia kellene nyugtára. Kezdetben az adó és a vevő ablakai olyanok, mint az a 3.22.(a) ábrán látható. Az adóállomás most továbbítja a kereteket a 0-tól a 6-ig. A vevő ablaka



3.22. ábra. *(a) Kezdeti állapot héthosszúságú ablakkal. (b) Hét keret elküldése és vétele után, de még a nyugtázás előtt. (c) Kezdeti állapot négyhosszúságú ablakkal. (d) Négy keret elküldése és vétele után, de még a nyugtázás előtt*

megengedi bármely keret elfogadását, melynek a sorszáma 0 és 6 közé esik, a határokat is beleértve. Mind a hét keret rendben megérkezik, így a vevő nyugtázza azokat, és lépteti az ablakát, hogy lehetővé tegye a 7., 0., 1., 2., 3., 4. és az 5. keret vételét. A beállt állapot a 3.22.(b) ábrán látható. Mind a hét puffert üresnek jelöljük meg.

Most érkeztünk el ahhoz a ponthoz, amikor beüt a mennykő egy villám formájában, mely a telefonoszlopot veszi célba, és kipucolja a vonalról az összes nyugtát. A katasztrófa ellenére a protokollnak továbbra is helyesen kell működnie. Az adóállomásnak előbb-utóbb lejár az időzítője, és újraadja a 0. keretet. Amikor ez a keret megérkezik a vevőhöz, a vevő ellenőrzi, hogy beleesik-e a vételi ablakba. Szerencsétlenségünkre a 3.22.(b) ábrán a 0. keret az új ablakon belül van, így új keretként elfogadja. A vevő egy ráültetett nyugtát küld a 6. keretről, hiszen a 0. és a 6. közötti kereteket vette.

Az adóállomás örül, amikor megtudja, hogy mindegyik továbbított kerete rendben megérkezett, így lépteti az ablakát, és azonnal elküldi a 7., 0., 1., 2., 3., 4. és 5. keretet. A 7. keretet a vevő elfogadja, és a benne levő csomagot rögtön átadja a hálózati rétegnek. Közvetlenül ezután a vevő adatkapcsolati rétege megnézi, hogy van-e már érvényes 0. kerete, rájön, hogy van, és továbbadja a benne levő öreg pufferelt csomagot a hálózati rétegnek, mintha az új csomag lenne. Ennek következtében a hálózati réteg helytelen csomagot kap, tehát a protokoll hibázik.

A probléma lényege az, hogy miután a vevő léptette az ablakát, az új érvényes sorszámok tartománya átlapolódik a régiekével. Ennek következtében az ezt követő keretköteg lehet akár duplikátum (ha az összes nyugta elveszett), akár új keretek kötege (ha az összes nyugta megérkezett). A szegény vevő sehogyan sem tud különbséget tenni a két eset között.

A dilemmából kivezető megoldás azon alapul, hogy biztosnak kell lennünk abban, hogy miután a vevő léptette az ablakát, nincs átfedés az eredeti ablakkal. Ahhoz, hogy ne legyen átfedés, a maximális ablakméret legfeljebb a sorszámok tartományának fele lehet. Ezt a helyzetet látjuk a 3.22.(c) és a 3.22.(d) ábrákon. 3 bit esetén a sorszámok 0-tól 7-ig terjednek. Így mindössze 4 nyugtázatlan keret lehet kint tetszőleges időpillanatban. Ha a vevő éppen elfogadta a 0.-tól 3.-ig tartó kereteket, és léptette az ablakát, hogy lehetővé tegye a 4.-től 7.-ig tartó keretek elfogadását, egyértelműen megmondható, hogy a következő keretek újraküldések (0.-tól 3.-ig) vagy újak (4.-től 7.-ig). Az ablakméret a 6. protokollnál általában $(MAX\_SEQ + 1)/2$.

Érdekes kérdés az, hogy a vevőben hány puffernek kell lennie. A vevő semmilyen körülmények között nem fogad el olyan kereteket, amelyek sorszáma kisebb, mint az ablak alsó széle, vagy nagyobb, mint a felső. Következésképpen a szükséges pufferek száma az ablak méretével egyenlő, nem a sorszámok tartományával. A fenti, 3 bites sorszámokat tartalmazó példában négy puffer kell 0-tól 3-ig számozva. Amikor az *i*. keret megérkezik, az (*i* mod 4). pufferbe kell tenni. Vegyük észre, hogy bár az *i*. és az (*i* + 4) mod 4. ugyanazért a pufferért „verseng", soha nincsenek az ablakban egyszerre, mert ehhez legalább 5 hosszúságú ablak kellene.

Ugyanezért az időzítők száma is a pufferek számával egyenlő, nem pedig a sorszámmező méretével. Valójában minden pufferhez tartozik egy időzítő. Amikor az időzítő lejár, a puffer tartalmát az adó újraküldi.

A 6. protokoll enyhíti azt az implicit feltételezést, miszerint a csatorna alaposan le van terhelve. Ezt a feltételezést még az 5. protokollban tettük, amikor a visszairányban rá-

ültetett nyugtákat használtunk. Ha a visszairányú forgalom kicsi, a nyugta hosszú ideig várakozhat, ami problémákat okozhat. Szélsőséges esetben, ha nagy a forgalom az egyik irányban és nincs forgalom a másikban, a protokoll blokkolódik, amikor az adó ablaka eléri a maximumát.

A probléma megoldása érdekében egy segédidőzítőt indítunk el a *start_ack_timer*-rel, miután egy sorrendben következő adatkeret megérkezik. Ha nem jelentkezik visszairányú forgalom, mielőtt az időzítő lejár, egy külön nyugtakeretet küldünk. A segédidőzítő által okozott megszakítás az *ack_timeout* esemény. Ezzel a megoldással most már egyirányú forgalom is lehetséges, mert a visszairányú adatkeretek (melyekre a nyugták ráültethetők) hiánya már nem akadály. Csak egyetlen segédidőzítő van, és ha a *start_ack_timer*-t meghívjuk, miközben az időzítő jár, a hívásnak semmilyen hatása nem lesz. Az időzítő nem áll vissza kezdeti állapotba, és nem állítódik be újra, hiszen a célja éppen az, hogy valamilyen rendszeres, a protokoll működéséhez szükséges nyugtázási sebességet biztosítson.

Lényeges, hogy a segédidőzítő időzítési intervalluma jelentősen rövidebb legyen a keretek időzítéséhez használt időzítőénél. Ez a feltétel ahhoz szükséges, hogy biztosak lehessünk benne, hogy egy rendben vett keret nyugtája megérkezik, mielőtt az adó időzítője lejárna és újraadná a keretet.

A 6. protokoll hatékonyabb stratégiát alkalmaz a hibák kezelésére, mint az 5. protokoll. Amikor a vevőnek oka van gyanítani, hogy hiba történt, egy negatív nyugtakeretet (NAK) küld vissza az adóállomásnak. Egy ilyen keretkérés a NAK-ban megjelölt keret újraadására. Két eset van, amikor a vevő gyanakodhat: egy sérült vagy egy a várttól különböző keret érkezett (potenciális keretvesztés). Ahhoz, hogy elkerüljük, hogy ugyanarra az elveszett keretre több újraküldési igény jelentkezzen, a vevőnek nyomon kell követnie, hogy küldött-e már NAK-ot egy adott keretre. A *no_nak* változó értéke a 6. protokollban igaz, ha még nem küldtünk NAK-ot a *frame_expected*-re. Ha a NAK tönkremegy vagy elveszik, ez semmi kárt nem okoz, hiszen az adóállomásnak végül is lejár az időzítője, és mindenképpen újraküldi a keretet. Ha rossz keret érkezik, miután egy NAK-ot elküldtünk és az elveszett, a *no_nak* igaz lesz, és elindítjuk a segédidőzítőt. Amikor lejár, egy nyugtát küldünk, hogy újraszinkronizáljuk az adóállomást a vevő aktuális állapotához.

Néhány helyzetben a keret célba jutásához, ottani feldolgozásához és a nyugta visszaéréséhez szükséges idő (közel) konstans. Ezekben a helyzetekben az adóállomás az időzítési intervallumát beállíthatja úgy, hogy éppen csak egy kicsit legyen hosszabb, mint az a várható idő, ami egy keret elküldése és nyugtájának vétele között telik el. A negatív nyugtázás ebben az esetben nem túl hasznos.

Néhány esetben azonban ez az idő tág határok közt változik. Például, ha a visszairányú forgalom esetleges, akkor a nyugtázáshoz szükséges idő rövid, ha van visszairányú forgalom, és ez az idő hosszú, ha nincs visszairányú forgalom. Az adó ekkor azzal a problémával szembesül, hogy vagy rövidre állítja ezt az intervallumot (kockáztatva a felesleges újraküldéseket), vagy hosszúra állítja (és sokáig várakozik egy hiba után). Mindkét választás sávszélesség-pazarló. Általában, amikor a nyugtázási időtartam szórása nagy magához az időtartamhoz képest, az időzítést „tág"-ra lehet állítani, és a NAK-ok észrevehetően felgyorsíthatják az elveszett vagy megsérült keretek újraküldését.

Szorosan kapcsolódva az időtúllépések és NAK-ok problémájához, felmerül azon kérdés eldöntése, hogy melyik keret okozta az időtúllépést. Az 5. protokollban ez mindig az *ack_expected*, mert ez mindig a legidősebb. A 6. protokollban nincs triviális módja annak, hogy meghatározzuk, hogy melyik okozta az időtúllépést. Tegyük fel, hogy a 0.-tól a 4.-ig tartó kereteket küldtük el, ami azt jelenti, hogy a kint levő keretek listája 0 1 2 3 4, a legidősebbtől a legfiatalabbig. Most képzeljük el, hogy a 0. időzítője lejár, az 5-et (egy új keretet) továbbítjuk, az 1. időzítője lejár, a 2. időzítője lejár, és a 6.-at (egy másik új keret) továbbítjuk. Ezen a ponton a kint levő keretek listája 3 4 0 5 1 2 6 a legidősebbtől a legfiatalabbig. Ha egy ideig az összes bejövő forgalom (például ráültetett nyugtát hordozó keretek) elvész, a hét kint levő keretnek ebben a sorrendben jár le az időzítője.

Hogy ne bonyolítsuk tovább a már eddig is elég bonyolult példát, nem mutatjuk be az időzítők adminisztrációját. Ehelyett csak azt feltételezzük, hogy az *oldest_frame* változót időtúllépéskor beállítjuk, hogy jelezze, hogy melyik keret időzítése járt le.

3.5. Példák adatkapcsolati protokollokra

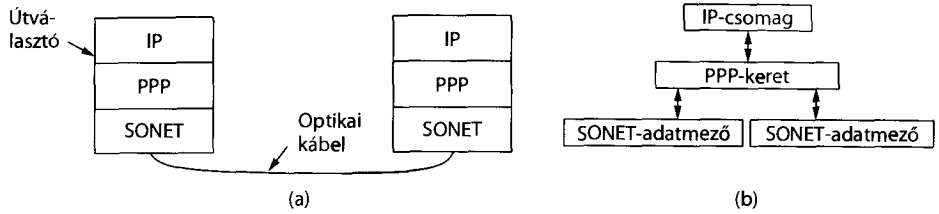
Széles körben használnak LAN-okat egyetlen épületen belül számítógépek összekapcsolására, a legtöbb nagy kiterjedésű hálózati infrastruktúra azonban kétpontos kapcsolatokra épül. A 4. fejezetben tárgyaljuk a LAN-okat. Itt most olyan kétpontos adatkapcsolati protokollokat fogunk vizsgálni, melyek az interneten két általános szituációban fordulnak elő. Az egyik esetben csomagokat küldenek át SONET optikai kapcsolatokon nagy kiterjedésű hálózatokban. Ezeket a kapcsolatokat széles körben használják például az internetszolgáltatók (ISP) hálózatában két különböző helyszínen található útválasztó összekapcsolására.

A második eset az internet szélén használatos, a telefonhálózatok előfizetői szakaszaiban megtalálható ADSL-kapcsolatok formájában. Ezeken keresztül több millió egyén és vállalat kapcsolódik az internetre.

Az internet is kétpontos kapcsolatokat igényel ilyen célokra ugyanúgy, ahogy a betárcsázós modemek (dial-up modem), a bérelt vonalak, a kábelmodemek és mások is. Egy szabványos protokollt, a **PPP-t (Point-to-Point Protocol – kétpontos protokoll)** használják csomagok küldésére az ilyen kapcsolatokon keresztül. A PPP-t az RFC 1661 definiálja, majd átdolgozták a RFC 1662-ben és a további RFC-kben is (Simpson, 1994a, 1994b). A SONET is és az ADSL is PPP-t használ, de mindkettő más módon.

3.5.1. Csomagok küldése SONET-en keresztül

A SONET – amelyről már ejtettünk szót a 2.6.4. szakaszban – a nagy kiterjedésű hálózatokban alkalmazott optikai kapcsolatot megvalósító fizikai rétegbeli protokoll, melyet leggyakrabban a kommunikációs hálózatok gerinchálózataiban használnak, beleértve a telefonhálózatokat is. Egy jól definiált sebességű bitfolyamot biztosít, például az OC-48 2,4 Gb/s sebességet határoz meg. Ezt a bitfolyamot adott darabszámú bájtot tartalmazó blokkokba (payload) rendezi, amelyeket 125 μs-onként ismétel, függetlenül attól, hogy van-e elküldendő felhasználói adat, vagy nincs.



3.23. ábra. *Csomagok küldése SONET-en keresztül. (a) Egy protokollkészlet (b) Adategységek viszonya*

Ahhoz, hogy csomagokat továbbítsunk ezeken a kapcsolatokon, valamilyen keretezési eljárásra lesz szükségünk, hogy megkülönböztessük az esetleges csomagokat attól a folyamatos bitfolyamtól, amelyben a csomagok továbbítódnak. Ennek biztosítására az IP-útválasztókon PPP-t futtatnak, ahogy azt a 3.23. ábrán láthatjuk.

A PPP egy korábbi, egyszerűbb protokoll továbbfejlesztése, amelyet **SLIP-nek (Serial Line Internet Protocol – soros vonali internetprotokoll)** neveznek. A PPP-protokollt hibakezelésre, kapcsolatok konfigurálására, több protokoll támogatására, hitelesítésre és több más célra használják. A lehetőségek széles tárházával rendelkezve három fő képessége van:

1. Olyan keretezési módszer, amely egyértelműen ábrázolja a keret végét és a következő keret kezdetét. A keretformátum megoldja a hibajelzést is.

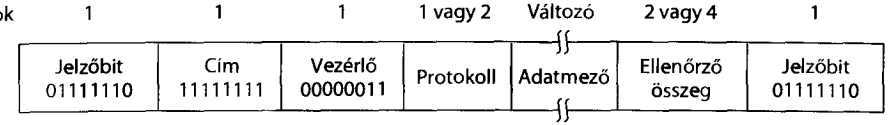
2. Adatkapcsolati protokoll a vonalak felélesztésére, tesztelésére, az opciók megbeszélésére és a vonalak elegáns elengedésére, amikor már nincs rájuk szükség. Ezt a protokollt **LCP-nek (adatkapcsolat-vezérlő protokoll – Link Control Protocol)** nevezik.

3. Egy olyan megoldás a hálózati rétegbeli opciók megbeszélésére, amely független az alkalmazott hálózati rétegbeli protokolltól. A választott módszer az, hogy különböző **NCP ((Network Control Protocol – hálózatvezérlő protokoll)** van mindegyik támogatott hálózati réteghez.

A PPP keretszerkezetét a tervezők a **HDLC (High-level Data Link Control – magas szintű adatkapcsolati vezérlés)** keretszerkezetéhez nagyon hasonlónak választották, mivel nem volt semmi okuk arra, hogy újra feltalálják a kereket.

A legfőbb különbség a PPP és a HDLC között az, hogy a PPP bájtalapú, a HDLC pedig bitalapú. Ez például abban nyilvánul meg, hogy a PPP bájtbeszúrást használ, és minden keret egész számú bájtot tartalmaz. A HDLC bitbeszúrást alkalmaz, és megenged, mondjuk, 30,25 bájtos kereteket is.

Van azonban egy másik fontos különbség is: a HDLC megbízható átvitelt biztosít csúszóablakkal, nyugtákkal és időzítőkkel úgy, ahogy azt korábban tanulmányoztuk. A PPP is képes megbízható átvitelt nyújtani zajos csatornákon, mint amilyenek például a vezeték nélküli hálózatok. A részleteket az RFC 1663-ban találjuk. A gyakorlatban azonban ezt ritkán használják. Helyette majdnem mindig a „számozatlan üzemmód" használatos az interneten, hogy összeköttetés nélküli, nyugtázatlan szolgáltatást nyújtson.



3.24. ábra. *A PPP teljes keretformátuma a számozatlan módú működéshez*

A PPP-keretek szerkezetét a 3.24. ábra mutatja be. Minden PPP-keret a szabványos HDLC-jelzőbájttal (0x7E, 01111110) kezdődik. Bájtbeszúrást alkalmaznak a 0x7D kivételbájttal, ha a jelzőbájt előfordul az *Adat* mezőben. A következő bájt az eredeti bájt és a 0x20 szám KIZÁRÓ VAGY (XOR) kapcsolata, ami az 5. bitet megfordítja. Például a 0x7E bájtból bájtbeszúrás után a 0x7D 0x5E lesz. Ez azt jelenti, hogy a keret eleje és vége egyszerűen megtalálható a 0x7E bájtra való kereséssel, hiszen az nem fordulhat elő máshol. A vétel során a vett keretben a 0x7D bájtokat kell keresni, majd azokat eltávolítani, és a következő bájt és a 0x20 szám KIZÁRÓ VAGY kapcsolatát kell venni. Keretek közt mindössze egy jelzőbájt elegendő, több jelzőbájt a csatorna kitöltésére használható, ha nincs küldendő keret.

A keret kezdetét mutató jelzőbájt után következik a *Cím* mező, amelyik mindig a bináris 11111111 értékre van állítva, hogy jelezze, hogy minden állomásnak vennie kell a keretet. Ennek az értéknek a használatával elkerülhetjük az adatkapcsolati címek hozzárendelésének problémáját.

A *Cím* mezőt a *Vezérlő* mező követi, melynek alapértéke 00000011. Ez az érték a számozatlan keretet jelzi.

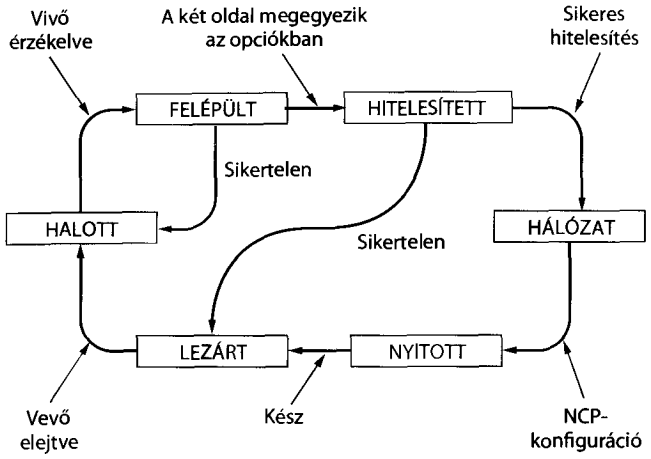
Mivel a *Cím* és a *Vezérlő* mezők mindig ugyanazok az alapbeállításban, az LCP biztosítja a szükséges mechanizmust a két résztvevő számára, hogy kihagyják ezeket a mezőket, és így megtakarítsanak 2 bájtot keretenként.

A negyedik mező a *Protokoll* mező. Ennek a feladata az, hogy megmutassa, hogy milyen csomag van az *Adat* mezőben. A 0 bittel kezdő értékeket az IP 4. és a 6. verziójának, valamint más hálózati rétegbeli protokolloknak (mint az IPX és az AppleTalk) definiálták. Az 1-es bittel kezdődő értékeket használják a PPP konfigurációs protokolljai, mint amilyen az LCP és a támogatott hálózati protokollokhoz tartozó NCP-k. A *Protokoll* mező hosszának alapértéke 2 bájt, de ez levihető 1 bájtra az LCP segítségével történő megegyezés alapján. A tervezők láthatólag nagyon óvatosak voltak, és feltételezték, hogy egyszer talán több mint 256 lehetséges protokoll lesz.

Az *Adat* mező változó hosszúságú, de legfeljebb a megegyezett maximumnak megfelelő méretű. Ha a hosszban nem egyeznek meg a felek a vonal beállítása alatt az LCP segítségével, az alapértelmezés 1500 bájt. Ha szükség van rá, az adat mezőt kitöltő bájtok követhetik.

Az *Adat* mező után az *Ellenőrző összeg* mező jön, amely rendszerint 2 bájtos, de a felek megegyezhetnek 4 bájtos ellenőrző összegben is. A 4 bájtos ellenőrző összeg valójában ugyanaz a 32 bites CRC, aminek a generátor polinomját a 3.2.2. szakasz végén megadtunk. A 2 bájtos ellenőrző összeg is egy szabványos CRC.

A PPP egy keretezési eljárás, amely különféle protokollok csomagjait képes továbbítani különböző fizikai rétegek felett. Ahhoz, hogy a PPP-t a SONET felett használjuk, az RFC 2615 ajánlásait kell követnünk [Malis és Simpson, 1999]. 4 bájtos ellenőrző összeg



3.25. ábra. *Egy PPP-kapcsolat felépítésének és lebontásának állapotdiagramja*

használatos, mivel ez az elsődleges eszköze a hiba felfedezésének a fizikai, adatkapcsolati és hálózati réteg felett. A *Cím*, *Vezérlő* és *Protokoll* mező tömörítése nem ajánlott, mivel a SONET kapcsolatok így is viszonylag nagy sebességgel működnek.

Van azonban egy szokatlan képessége is a PPP-nek. A PPP adatmezőjét tördelik (scrambling) (ahogy a 2.5.1. szakaszban leírtuk), mielőtt a SONET adatmezőjébe illesztik. A tördelés során az adatmezőt egy hosszú, ál-véletlen sorozattal KIZÁRÓ VAGY (XOR) kapcsolatba hozzák, mielőtt továbbítják. Mindezt azért teszik, mert a SONET-nek a bitfolyamban gyakori bitváltásra van szüksége a szinkronizáció fenntartására. Ezek a váltások természetes módon kerülnek a beszédjelekbe, de az adatkommunikációban a felhasználó választja meg az elküldendő információt, és akár egy sok nullával feltöltött csomagot is elküldhet. Tördeléssel elhanyagolhatóvá tehető az esélye annak, hogy a felhasználó problémát tudjon okozni egy sok 0-t tartalmazó csomag elküldésével.

Mielőtt PPP-kereteket tudnánk küldeni SONET-vonalakon, a PPP-kapcsolatot fel kell építeni, és konfigurálni kell. A 3.25. ábrán ábrázoltuk azokat a fázisokat, melyeken a kapcsolat keresztülmegy, miközben felélesztjük, használjuk és elengedjük.

A kapcsolat kezdő állapota *HALOTT*, ami azt jelenti, hogy nincs fizikai rétegbeli összeköttetés. Miután a fizikai összeköttetés felépült, a kapcsolat a *FELÉPÜLT* állapotba kerül. Ezen a ponton a PPP-felek az *Adat* mezőben szállított LCP-csomagokat cserélnek, hogy a fentebb említett PPP-lehetőségekből válasszanak a kiépítendő kapcsolatra vonatkozólag. A kezdeményező fél felajánl opciókat, majd a válaszoló fél részben vagy egészben elfogadja vagy elutasítja azokat. A válaszoló alternatív opciókat is felajánlhat.

Ha az LCP-opciók megbeszélése sikeres, akkor a kapcsolat eléri a *HITELESÍTETT* állapotot. Ekkor a két fél ellenőrizheti egymás identitását. Ha a hitelesítés sikeres, akkor eljutunk a *HÁLÓZAT* állapotba, és egy sor NCP-csomagot küldenek a felek, hogy a hálózati réteget beállítsák. Nehéz általánosságban bármit is mondani az NCP-protokollokról, mivel mindegyik speciális valamely hálózati protokollra, és olyan konfigurációs kéréseket tesz lehetővé, melyek specifikusak arra a protokollra nézve. Például az IP esetén a legfontosabb ilyen lehetőség a kapcsolat két végpontjához IP-címek hozzárendelése.

A *NYITOTT* állapotot elérve lehetőség van a felhasználói adatok áramlására. Ebben az állapotban IP-csomagok továbbítása történik PPP-keretekben SONET-vonalakon keresztül. Ha az adatátvitel véget ért, akkor a kapcsolat a *LEZÁRT* állapotba kerül, majd innen a *HALOTT* állapotba ugrik a fizikai rétegbeli összeköttetés megszakadásakor.

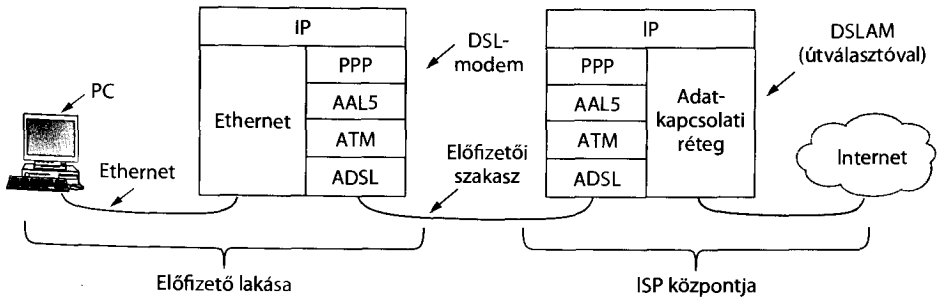
3.5.2. ADSL – aszimmetrikus digitális előfizetői szakasz

Az ADSL segítségével sok millió lakossági előfizető kapcsolódik az internethez megabit/s nagyságrendű sebességgel ugyanazon az előfizetői szakaszon keresztül, amelyet a hagyományos telefonrendszer szolgáltatásához használnak. A 2.5.3. szakaszban leírtuk, hogyan lehet telepíteni egy DSL-modemet az előfizetői oldalon. A modem biteket küld az előfizetői szakaszon keresztül egy, a telefontársaságok helyi központjaiban lévő DSLAM (DSL Access Multiplexer – DSL hozzáférési multiplexer) nevű eszköznek. A következőkben áttekintjük, hogyan lehet csomagokat átvinni ADSL-kapcsolatokon.

A 3.26. ábrán egy áttekintő képet láthatunk az ADSL-lel használt protokollokról és eszközökről. Különböző hálózatokban különböző protokollokat használnak, így mi bemutatásra a legnépszerűbb esetet választottuk ki. Egy számítógép a lakásban – mondjuk egy PC – IP-csomagokat küld a DSL-modemnek az adatkapcsolati réteg felhasználásával, amilyen például az Ethernet is. A DSL-modem aztán elküldi ezeket az IP-csomagokat az előfizetői szakaszon keresztül a DSLAM részére egy olyan protokoll segítségével, amelyet most fogunk megvizsgálni. A DSLAM (vagy az implementációtól függően a hozzákapcsolt útválasztó) az IP-csomagokat kiveszi, és belépteti az ISP-szolgáltató hálózatába, hogy azok az interneten keresztül a címzetthez jussanak.

A protokollkészlet (nevezik protokollveremnek is), amelyet a 3.26. ábrán az ADSL-kapcsolat felett láthatunk, alulról az ADSL fizikai réteggel kezdődik. A réteg egy digitális modulációs eljáráson alapul, melyet ortogonális frekvenciaosztásos nyalábolásnak vagy más néven diszkrét többvivős modulációnak neveznek, ahogy a 2.5.3. szakaszban láttuk. A verem tetejéhez közel, az IP alatt találjuk a PPP-t. Ez ugyanaz a PPP, amit a 3.5.1. szakaszban tanulmányoztunk. Ugyanúgy működik az adatkapcsolat létesítésére, konfigurálására és IP-csomagok átvitelére.

Az ADSL és a PPP között helyezkedik el az ATM és az AAL5. Ezek olyan protokollok, melyekkel eddig még nem találkoztunk. Az **ATM-et (Asynchronous Transfer**



3.26. ábra. *ADSL-protokollkészlet*

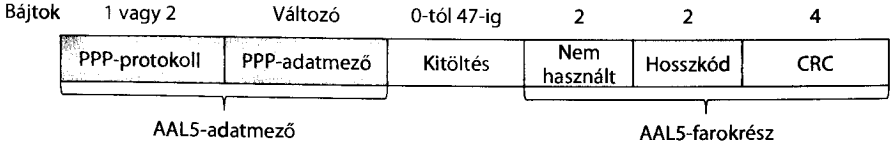
Mode – aszinkron átviteli mód) az 1990-es évek elején tervezték, és hihetetlen méretű hírveréssel indították. Olyan hálózati megoldást ígért, amely a világ telekommunikációs problémáit véglegesen megoldja, amely a beszédet, az adatot, a kábeltelevíziót, a távírót, a postagalambot, a madzaggal egymáshoz erősített konzervdobozokat, a tam-tam dobokat és minden mást egyetlen integrált rendszerbe olvasztja, és megold mindenkinek mindent. Sajnos ez nem történt meg. Az ATM hibái nagyrészt hasonlóak voltak az OSI-protokollok hibáihoz, vagyis a rossz időzítés, a rossz technológia, a rossz implementáció és a rossz politika. Mindazonáltal az ATM sokkal sikeresebb volt, mint az OSI. Ugyan a világot nem hódította meg, de még mindig széles körben használják hozzáférési hálózatokban, amilyen a DSL, és WAN-kapcsolatokon a telefonhálózatok belsejében.

Az ATM olyan adatkapcsolati réteg, amely rögzített hosszúságú adategységek, ún. **cellák** átvitelén alapul. Az „aszinkron" jelző a nevében azt jelzi, hogy a cellákat nem kell folyamatosan küldeni, mint ahogy a biteket a szinkronvonalakon, például a SONET-en keresztül. A cellákat csak akkor kell átvinni, ha van valami elküldendő információnk. Az ATM összeköttetés-alapú technológia. Minden cella fejrészében van egy **virtuális áramköri** (**virtual circuit**) azonosító, amelyet az ATM-eszközök a cellák előre kiépített útvonalon történő továbbításánál használnak.

A cellák 53 bájt hosszúak, amiből 48 bájt a felhasználói adatoké, 5 bájt pedig a fejléc. A kis celláknak köszönhetően az ATM rugalmasan fel tudja osztani a rendelkezésre álló sávszélességet a különböző felhasználók között. Ez a képessége abban az esetben hasznos, amikor például adatot és valós idejű beszédet szeretnénk átvinni egyetlen adatkapcsolaton, elkerülve azt, hogy hosszú adatcsomagok nagy szórást okozzanak a beszéd mintáinak késleltetésében. A cella méretének szokatlan megválasztása (összehasonlítva például a sokkal természetesebbnek ható 2 hatványaival) is mutatja, hogy a politika milyen mértékben beleszólt az ATM tervezésébe. A 48 bájtos adatmező az Európa által szorgalmazott 32 bájtos cella és az Egyesült Államok által javasolt 64 bájtos cella közötti kompromisszum eredménye. Egy rövid összefoglalót olvashatunk az ATM-ről Siu és Jain [1995] munkájában.

Ahhoz, hogy adatokat küldjünk egy ATM-hálózaton keresztül, először az adatokat le kell képezni cellák sorozatára. A leképezést az ATM adaptációs rétege végzi egy „darabolás és összeállítás" (segmentation and reassembly) folyamat során. Számos adaptációs réteget definiáltak a különböző szolgáltatásokhoz, amelyek a periodikus beszédmintáktól egészen az adatcsomagokig terjed. A legfontosabb adaptációs réteg az adatcsomagok szállításához használt **AAL5** (**ATM Adaptation Layer 5 – 5. ATM adaptációs réteg**).

A 3.27. ábra egy AAL5-keretet mutat. Fejléc helyett egy farokrészt tartalmaz, amely megadja a keret hosszát és tartalmaz egy hibajelzésre alkalmas 4 bájtos CRC-t. A CRC természetesen ugyanaz, mint amit a PPP és az IEEE 802 LAN-ok használnak (így az Ether-



3.27. ábra. *PPP-adatokat hordozó AAL5-keret*

net is). Wang és Crowcroft [1992] megmutatták, hogy ez elég erős nem szokványos hibák (például cellasorrend változás) jelzésére. Az AAL5-keret adatmezője is tartalmaz kitöltést (padding), hogy a teljes hosszt 48 bájt többszörösére egészítse ki annak érdekében, hogy a keretet hiánytalanul cellákra lehessen osztani. Címekre nincs szükség a keretben, hiszen a cellák virtuális áramköri azonosítója gondoskodik a cellák célba juttatásáról.

Most, hogy megismerkedtünk az ATM-mel, már csak azt kell áttekintenünk, hogyan használja a PPP az ATM-et az ADSL esetében. Ezt ismét egy szabvány írja le, amelyet **PPPoA**-nak (**PPP over ATM**) neveznek. Ez a szabvány nem is igazán egy protokoll (ezért sem látjuk a 3.26. ábrán), hanem inkább egy specifikáció, amely azt adja meg, hogyan kell dolgozni mind PPP-keretekkel, mind AAL5 keretekkel. Leírása az RFC 2364-ben található [Gross és mások, 1998].

Az AAL5 adatmezőjébe csupán a PPP-protokollmező és PPP-adatmező került, ahogy azt a 3.27. ábrán látjuk. A protokollmező a DSLAM felé jelzi, hogy az adatmező egy IP-csomag vagy egy másik protokoll, például az LCP-csomagja. A szolgáltató oldalán tudják, hogy a cellák PPP-információt tartalmaznak, mivel egy ATM virtuális áramkör épült fel ennek érdekében.

Az AAL5-kereten belül PPP keretezésre nincs szükség, mivel semmilyen célja nem volna. Az ATM és az AAL5 ugyanis már biztosít keretezést, így bármilyen további keretezés értelmetlen volna. A PPP CRC ugyancsak felesleges, hiszen az AAL5 tartalmazza ugyanazt a CRC-t. Ez a hibajelzés kiegészíti az ADSL fizikai rétegének Reed–Solomon hibajavító kódolását és 1 bájtos CRC-jét, amelyet a rejtve maradt hibák jelzésére használnak. Ez a megoldás sokkal kifinomultabb hibakezelést biztosít, mint amikor csomagokat SONET-vonalon keresztül küldenek, ugyanis az ADSL-csatorna sokkal zajosabb.

3.6. Összefoglalás

Az adatkapcsolati réteg feladata az, hogy a fizikai réteg által kínált nyers bitfolyamot átalakítsa keretfolyammá, amit a hálózati réteg használ. A keretfolyam különböző megbízhatóságot nyújthat az összeköttetés nélküli, nyugtázatlan szolgáltatástól kezdve egészen a megbízható, összeköttetés-alapú szolgáltatásig.

Változatos keretezési eljárások használatosak, ideértve a bájtszámlálást, bájtbeszúrást és a bitbeszúrást. Az adatkapcsolati protokollok biztosíthatnak hibavédelmet a sérült keretek jelzésére vagy javítására és az elveszett keretek újraküldésére. Ahhoz, hogy megakadályozzuk, hogy a gyors adó elárassza a lassú vevőt, az adatkapcsolati protokoll biztosíthat forgalomszabályozást is. A csúszóablakos mechanizmust széles körben alkalmazzák a hibavédelem és a forgalomszabályozás egyszerű megoldására. Amikor az ablakméret 1 keret, a protokoll megáll-és-vár típusú.

A hibajavító és hibajelző kódok különböző matematikai módszerek segítségével redundanciát kapcsolnak az üzenetekhez. Hibajavításra széles körben használják a konvolúciós kódokat és a Reed–Solomon-kódokat, de az alacsony sűrűségű paritásellenőrző kódok népszerűsége is rohamosan növekszik. A hibajelzésre leggyakrabban ciklikus redundancia ellenőrzést és ellenőrző összeget használnak. Ezek a kódok az adatkapcsolati rétegben használatosak, de a fizikai és a felsőbb rétegekben is alkalmazhatók.

Egy sor protokollt vizsgáltunk meg, melyek megbízható adatkapcsolatot biztosítanak. Ezek a protokollok nyugtákat és újraküldéseket vagy – valószerűbb feltételezéssel – ARQ-t (Automatic Repeat reQuest) használnak. A vizsgálódást hibamentes csatornán kezdtük, egy olyan vevővel, amely bármilyen keretet elfogad, majd beépítettük a forgalomszabályzást, a hibakezelést sorszámokkal és a megáll-és-vár algoritmust. Bevezettük végül a csúszóablakos megoldást a kétirányú kommunikáció érdekében, majd a ráültetett nyugtázás koncepcióját. A két utóbbi protokoll a csővezetékezés segítségével több keretet is a vonalon tart, hogy elkerülje az adó blokkolódását nagy terjedési késleltetések esetén. A vevő két megoldás közül választhat: vagy eldob minden keretet, kivéve a soron következőt, vagy a nem soron következő kereteket tárolja, és negatív nyugtát küld a sávszélesség hatékony kihasználása végett. Az előző stratégia az n visszalépéses protokoll, az utóbbi a szelektív ismétlés eljárás.

Az internet – adatkapcsolati protokollként – elsősorban PPP-t használ kétpontos (pont–pont típusú) vonalakon. A PPP összeköttetés nélküli, nyugtázatlan szolgáltatást biztosít, és jelzőbájtokat használ a keretek elválasztására, valamint CRC-t a hibajelzésre. Csomagok továbbítására használják különböző kapcsolatokon, például nagy kiterjedésű hálózatok SONET-kapcsolatain és ADSL-kapcsolatokban a hozzáférési hálózaton.

3.7. Feladatok

1. Egy felsőbb rétegbeli üzenetet 10 keretre osztunk, és mindegyiknek 80% esélye van arra, hogy sértetlenül megérkezzen. Ha az adatkapcsolati réteg semmilyen hibavédelmet nem végez, átlagosan hányszor kell az üzenetet elküldeni ahhoz, hogy az egész eljusson a vevőhöz?

2. Egy adatkapcsolati protokollban a karaktereket a következő módon kódolják:
   A: 01000111; B: 11100011; FLAG: 01111110; ESC: 11100000
   Írja le (binárisan), hogy milyen bitsorozat kerül továbbításra az A B ESC FLAG négykarakteres keret elküldésekor, a következő keretszervezési eljárások használata esetén:
   (a) Bájtszámlálás.
   (b) Jelzőbájtok bájtbeszúrással.
   (c) A keret elejét és végét jelzőbájtok jelölik, bitbeszúrással.

3. Egy a szövegben ismertetett bájtbeszúrási algoritmust alkalmazó adatfolyam közepén az „A B ESC C ESC FLAG FLAG D" adattöredék érkezik. Mi a kimenet a bájtbeszúrás után?

4. Mennyi a maximális többletráfordítása (overhead) a bájtbeszúrás algoritmusnak?

5. Az egyik osztálytársa, Fukar, arra hívta fel a figyelmét, hogy pazarló megoldás minden keretet egy jelzőbájttal befejezni, és a következő keretet egy újabb jelzőbájttal

elkezdeni. Szerinte egyetlen jelzőbájt ugyanolyan alkalmas lenne erre a feladatra, így egy bájt megtakarítható lenne. Egyetért vele?

6. Az adatkapcsolati rétegnek a 0111101111101111110 bitfüzért kell átvinnie. Mi a bitbeszúrás után ténylegesen átvitelre kerülő füzér?

7. El tud-e képzelni olyan körülményeket, amelyek között egy nyílthurkú protokoll (például egy Hamming-kód) előnyösebb lehet azoknál a visszacsatolás típusú protokolloknál, amelyeket ebben a fejezetben ismertünk meg?

8. Egyetlen paritásbit által nyújtottnál nagyobb biztonságot akarunk elérni, így olyan hibaészlelő sémát alkalmazunk, amelyben két paritásbit van: az egyik a páros, a másik a páratlan bitek ellenőrzésére. Mekkora e kód Hamming-távolsága?

9. 16 bites üzeneteket továbbítunk Hamming-kódolással. Hány ellenőrző bitre van szükség ahhoz, hogy a vevő oldalán biztosíthassuk az egybites hibák felderítését és javítását? Írja le az átvitt bitmintázatot arra az esetre, amikor az üzenet 1101001100110101! Használja azt a feltevést, hogy a Hamming-kód páros paritást használ!

10. Egy olyan 12 bites Hamming-kód érkezik a vevőhöz, amelynek a hexadecimális értéke 0xE4F. Mi volt az eredeti érték hexadecimális formája? Tegyük fel, hogy legfeljebb 1 bit hibásodott meg.

11. A hibák észlelésének egy lehetséges módja az, hogy az adatokat n sorból és k oszlopból álló blokkokban visszük át úgy, hogy minden egyes sorhoz és oszlophoz paritásbiteket adunk. A jobb alsó sarok egy olyan paritásbit, amely mind az oszlopát, mind a sorát ellenőrzi. Vajon ez az elrendezés tudja-e jelezni az összes egybites hibát? A kettősöket? A hármas hibákat? Mutassa meg, hogy ez az elrendezés nem képes bizonyos 4-bites hibákat jelezni!

12. Tegyük fel, hogy az adatokat 1000 bites blokkokban továbbítjuk. Mennyi az a maximális hibaarány, ami alatt a hibajelzés és az újraküldés (1 paritásbit blokkonként) jobb, mint Hamming-kód használatával? Tegyük fel, hogy a bithibák függetlenek egymástól, és az újraküldés alatt nem történik bithiba!

13. Egy n soros, k oszlopos bitblokk vízszintes és függőleges paritásbiteket használ hibajelzésre. Tételezzük fel, hogy pontosan 4 bit állítódik át az átviteli hibák következtében. Vezessünk le egy képletet annak a valószínűségére, hogy a hiba észrevétlen marad.

14. A 3.7. ábra konvolúciós kódolóját használva, mi lesz a kimeneti sorozat, ha a bemeneti sorozat 10101010 (balról jobbra), és a kódoló kezdeti belső állapota tisztán 0?

15. Tegyük fel, hogy az 1001 1100 1010 0011 üzenetet IP ellenőrző összeggel továbbítjuk (4-bites szavakkal)! Mennyi az ellenőrző összeg értéke?

16. Mekkora lesz a keletkező maradék, ha $x^7 + x^5 + 1$-et elosztjuk az $x^3 + 1$ generátor-polinommal?

17. Az 10011101 bitfolyamot visszük át a szövegben ismertetett szabványos CRC-módszerrel. A generátor-polinom $x^3 + 1$. Írja le a ténylegesen átvitelre kerülő bitfüzért! Tegyük fel, hogy balról a harmadik bit az átvitel folyamán invertálódik, mutassa meg, hogy ez a hiba felderítésre kerül a vevő oldalán! Adjon egy példát olyan bithibákra ebben a bitfolyamban, amelyet a vevő nem vesz észre!

18. Egy 1024 bites üzenetet küldünk, ami 992 adatbitet és 32 CRC-bitet tartalmaz. A CRC-t az IEEE 802 szabványban meghatározott 32-ed fokú CRC-polinommal számítjuk. Mutassa meg a következő esetekre, hogy a vevő észreveszi-e a megadott hibát vagy nem:
(a) Egybites hiba.
(b) Két különálló bithiba.
(c) 18 darab különálló bithiba.
(d) 47 darab különálló bithiba.
(e) 24 bites hibacsomó.
(f) 35 bites hibacsomó.

19. Az ARQ-protokoll tárgyalásánál a 3.3.3. szakaszban bemutattunk egy olyan esetet, amikor a vevő ugyanannak a keretnek két példányát is elfogadta, mivel elveszett egy nyugta. Elképzelhető-e, hogy a vevő egy keret több példányát is elfogadja úgy, hogy semmilyen keret (sem üzenet, sem nyugta) nem veszik el?

20. Egy csatornának 4 kb/s-os sebessége és 20 ms-os terjedési késleltetése van. Milyen keretméret-tartományra ad a megáll-és-vár stratégia legalább 50%-os hatékonyságot?

21. Előfordulhat a 3. protokollban, hogy az adó olyankor indítja el az időzítést, amikor az már fut? Amennyiben igen, hogyan kerülhet erre sor? Ha nem, miért lehetetlen?

22. Egy 3000 km hosszú T1-es trönköt az 5. protokoll segítségével 64 bájtos keretek átvitelére használunk. Ha a terjedési sebesség 6 μs/km, hány bitesnek kell lennie a sorszámnak?

23. Képzeljünk el egy olyan csúszóablakos protokollt, amely annyi sorszámbitet használ, hogy körbefordulás soha nem fordul elő. Milyen összefüggéseknek kell fennállniuk a két ablak négy széle és az ablakok mérete között, ha az ablakméret állandó, valamint a vevő és a fogadó oldalán megegyezik?

24. Ha az 5. protokoll *between* eljárása az $a \le b \le c$ feltételt ellenőrizné az $a \le b < c$ feltétel helyett, annak lenne valamilyen hatása a protokoll helyességére és hatásfokára? Indokolja válaszát!

25. Amikor a 6. protokollban egy adatkeret megérkezik, a vevő ellenőrzi, hogy a sorszám különbözik-e a várttól, és hogy a *no_nak* értéke igaz-e. Amennyiben mindkét feltétel teljesül, a vevő egy NAK-ot küld, egyébként pedig egy kiegészítő időzítést indít el. Tegyük fel, hogy az else-ágat kihagyjuk a kódból! Van ennek a változtatásnak hatása a protokoll helyességére?

26. Tegyük fel, hogy a 6. protokoll vége felé szereplő három kifejezéses while-hurkot kivesszük a kódból! Ez a változtatás csak a protokoll hatékonyságát érintené, vagy a helyességét is? Indokolja válaszát!

27. Egy távoli bolygó és a Föld távolsága megközelítőleg 9×10^{10} m. Mekkora a csatorna kihasználtsága, ha megáll-és-vár protokollt használunk a keretek továbbítására egy 64 Mb/s sebességű kétpontos kapcsolaton? Tegyük fel, hogy a keretméret 32 KB, és a fény sebessége 3×10^8 m/s!

28. Az előző feladatban tételezzük fel, hogy csúszóablakos protokollt használunk! Mekkora adási ablakméretre lesz a kihasználtság 100%? A protokoll feldolgozási idejét mind az adó, mind a vevő oldalon elhanyagolhatjuk.

29. A 6. protokollban a *frame_arrival* eljárás kódja tartalmaz egy olyan részt, amely a NAK-okat kezeli. Ez a rész akkor kerül meghívásra, ha a bejövő keret egy NAK, és egy további feltétel is teljesül. Adjon meg egy olyan helyzetet, amelyben ennek a második feltételnek kulcsfontosságú szerepe van!

30. Tekintsük a 6. protokoll működését egy 1 Mb/s-os hibamentes vonalon. A maximális keretméret 1000 bit. Az új csomagok egy másodperces időközönként generálódnak. Az időzítési intervallum 10 ms. Ha elhagynánk a speciális nyugtázási időzítéseket, akkor szükségtelen időtúllépések következnének be. Hányszor kellene elküldeni egy átlagos üzenetet?

31. A 6. protokollban $MAX\_SEQ = 2^n - 1$. Míg ez a feltétel nyilvánvalóan kívánatos a fejrészbitek hatékony használata miatt, azt nem mutattuk meg, hogy egyben alapvető fontosságú is. Vajon a protokoll helyesen működne-e $MAX\_SEQ = 4$ esetén?

32. 1000 bites kereteket küldünk egy 1 Mb/s-os műholdas csatornán, amelynek a késleltetése 270 ms a Földtől a geostacionárius műholdig. A nyugtákat mindig ráültetjük az adatkeretekre. A fejrészek nagyon rövidek. Hárombites sorszámokat használunk. Mennyi a maximálisan elérhető csatornakihasználtság
(a) a megáll-és-vár,
(b) az 5. protokoll és
(c) a 6. protokoll esetében?

33. Számítsuk ki, hogy a sávszélesség mekkora hányada megy veszendőbe az overhead (fejrészek és újraadások) miatt a 6. protokoll használata esetén, egy erősen terhelt 50 kb/s-os műholdas csatornán, melyen az adatkeretek 40 fejrész- és 3960 adatbitet

tartalmaznak! Tegyük fel, hogy a késleltetés a földtől a műholdig 270 ms! ACK keretek soha nem fordulnak elő. A NAK-keretek 40 bitesek. Az adatkeretek hibaaránya 1%-os, a NAK-kereteké elhanyagolható. A sorszámok 8 bitesek.

34. Vegyünk egy 64 kb/s-os műholdas csatornát, melyet 512 bájtos adatkeretek küldésére használunk az egyik irányban, és nagyon rövid nyugták jönnek vissza a másik irányban. Mennyi a maximális átviteli sebesség az 1, 7, 15 és a 127 ablakméretekhez? A Föld és a műhold közötti terjedési idő 270 ms.

35. Egy 100 km hosszú kábel T1-es adatsebességgel működik. A terjedési sebesség a kábelen a fénysebesség 2/3-a. Hány bit fér a kábelre?

36. Adjon legalább egy indokot arra, hogy a PPP miért bájtbeszúrást használ bitbeszúrás helyett, hogy az adatmezőben kizárja a jelzőbájt összekeverését egy azonos értékű adatbájttal!

37. Mennyi a minimális többletadat (overhead) egy IP-csomag PPP-vel való elküldésekor? Csak a PPP által okozott rezsit vegyük figyelembe, az IP fejrészét ne! Mekkora a maximális többletadat (overhead)?

38. Egy 100 bájtos IP-csomagot küldünk át egy előfizetői szakaszon ADSL-protokollvermet használva. Hány ATM-cellát továbbítunk? Röviden írja le a tartalmukat!

39. Ennek a laborgyakorlatnak az a célja, hogy a szövegben ismertetett szabványos CRC-eljárás felhasználásával megvalósítsunk egy hibajelző megoldást. Írjon két programot, egy *generátort* (generator) és egy *ellenőrzőt* (verifier)! A *generátor* program egy n bites üzenetet olvas be 0-k és 1-esek füzéreként, a standard inputon megadott ASCII-szövegsorból. A második sor egy k bites polinom, szintén ASCII-kódolásban. A program a standard outputra teszi a kimenetét, egy olyan $n+k$ darab 0-ból és 1-esből álló ASCII-sort, amely az elküldendő üzenetet jelképezi. Ezután kiteszi a kimenetére a polinomot is, minden változtatás nélkül. Az ellenőrző program beolvassa a generátor program kimenetét és a kimenetén egy üzenettel jelzi, hogy az helyes volt-e. Végül, írjon egy olyan *változtató* (alter) programot, amely az első sorban egy, a paraméterétől függő bitet invertál (a paraméter a bit száma, ahol a bal szélső bit az 1-es), de a két sor összes többi részét helyesen másolja le. Ha azt gépeli be, hogy

```
generator <file | verifier,
```

akkor azt kell tapasztalnia, hogy az üzenet helyes. Ha azonban azt gépeli be, hogy

```
generator <file | alter arg | verifier,
```

akkor egy hibaüzenetet kell kapnia.

4. A közeg-hozzáférési alréteg

A hálózatok két kategóriába sorolhatók: vannak, melyek kétpontos összeköttetést, és vannak, amelyek adatszóró csatornát használnak. A 2. fejezet tárgyalta a kétpontos összeköttetéseket, ez a fejezet az adatszóró hálózatokkal, és az ilyen hálózatokon használható protokollokkal foglalkozik.

Minden adatszóró hálózat esetén a kulcskérdés az, hogy versenyhelyzetben hogyan állapítható meg, melyik állomás nyerje el a csatorna használatának jogát. Ahhoz, hogy a problémát tisztábban láthassuk, vegyünk egy példát: képzeljünk el egy telefonos konferenciabeszélgetést, amelyben hat ember vesz részt, akik külön telefonkészülékeket használnak. A konferencia során minden készülék között van kapcsolat, így mindenki hallja a másikat, és bárkihez tud szólni. Ilyen szituációkban sűrűn fordul elő az, hogy amikor valaki befejezi a mondandóját, egyszerre ketten vagy többen is megszólalnak, ami teljes kavarodáshoz vezet. Ez a zűrzavar nem jelentkezne egy személyes találkozón, hiszen ekkor külső jelekkel, például kézfeltartással jelezhetik felszólalási szándékukat. Viszont akkor, ha csak egyetlen csatorna áll rendelkezésre, sokkal nehezebb annak eldöntése, hogy ki használhatja a csatornát következőként. A feladat megoldására rengeteg protokoll ismert, és a fejezet ezekről próbál áttekintést nyújtani. A szakirodalomban az adatszóró csatornákra (broadcast channel) sokszor hivatkoznak **többszörös hozzáférésű csatorna** (**multiaccess channel**), illetve **véletlen hozzáférésű csatorna** (**random access channel**) megnevezéssel is.

Az adatkapcsolati réteg egy alrétegéhez, a **MAC**-alréteghez (**Medium Access Control – közeg-hozzáférési alréteg**) tartoznak azok a protokollok, amelyek az adatszóró csatorna használatának vezérléséért felelősek. A MAC-alréteg különösen fontos szerepet tölt be a LAN-hálózatokban, különösen a vezeték nélküli LAN-hálózatokban, mert ezek természetükből adódóan adatszóró csatornák. Ezzel szemben – a műholdas rendszerek kivételével – a WAN-hálózatok kétpontos összeköttetésekből állnak össze. Mivel a többszörös hozzáférésű csatornák és a LAN-hálózatok ilyen szoros kapcsolatban állnak, ebben a fejezetben általában véve a LAN-hálózatokról lesz szó, néhány olyan témával együtt, melyek nem tartoznak szorosan a MAC-alréteghez. Mindezzel együtt a fejezet fő témája a csatorna vezérlése lesz.

Technikailag a MAC-alréteg az adatkapcsolati réteg alsó részét képezi, így logikailag a 3. fejezetben részletezett kétpontos protokollok előtt kellett volna ismertetni. Mindazonáltal a legtöbb ember számára egyszerűbb a több résztvevő számára készült proto-

kollokat megérteni azután, ha már megismerte a két résztvevő számára készült protokollokat, ezért kis mértékben eltértünk a szigorúan vett alulról felfelé haladó bemutatási sorrendtől.

4.1. A csatornakiosztás problémája

Ez a fejezet elsősorban azzal foglalkozik, hogyan lehet egy adatszóró csatornát a versengő felhasználók között kiosztani. Maga a csatorna lehet a rádiós spektrum egy szelete, vagy egy földrajzi régióban egy egyszerű rézvezeték vagy optikai kábel, amelynek több csatlakozási pontja van. Ez csak részletkérdés. A csatorna minden esetben fizikai összeköttetést biztosít a hozzá csatlakozó felhasználók között. Bármelyik felhasználó, aki a csatornát használja, zavarni fogja a többieket, akik ugyanúgy használni szeretnék a csatornát.

Először a statikus csatornakiosztási sémák löketes forgalomnál jelentkező hátrányait vizsgáljuk meg. Majd a rákövetkező szakaszokban lefektetjük a dinamikus csatornakiosztási sémák modellezésének kulcsfontosságú alapelveit.

4.1.1. Statikus csatornakiosztás

Hagyományosan egy közös csatorna, mint amilyen egy telefontrönk, több versengő felhasználó közötti felosztása úgy zajlik, hogy a csatorna kapacitását több részre bontjuk a 2.5. szakaszban leírt valamelyik nyalábolási módszerrel, például frekvenciaosztásos nyalábolással (Frequency Division Multiplexing, FDM). Ha N felhasználó van, a sávszélességet N darab egyenlő méretű sávra osztják, majd minden felhasználóhoz hozzárendelik az egyik frekvenciasávot. Mivel minden felhasználónak saját frekvenciasávja van, nem zavarhatják egymást. Az FDM egyszerű és hatékony kiosztási mechanizmus olyan esetekben, amikor kevés, fix számú felhasználó van, és ezek közül mindegyik állandó méretű vagy nagy forgalmi igénnyel rendelkezik. Az FM-rádióállomások rendszere jó példa a vezeték nélküli hálózatokra. Minden állomás megkapja az FM-frekvenciasáv egy részét, és saját jeleinek üzenetszórására használja az idő nagy részében.

Azokban az esetekben azonban, amikor a küldő felek száma nagy és folyamatosan változik, vagy az adatforgalom löketes jellegű, felmerül néhány probléma az FDM alkalmazásával. Ha a frekvenciaspektrumot N részre osztottuk, viszont N-nél kevesebb felhasználó szeretne egyszerre kommunikálni, az értékes spektrum jókora hányada kárba vész. Ha viszont N-nél több felhasználó szeretne kommunikálni, szabad sávok hiányában néhányat a rendszernek vissza kell utasítania még akkor is, ha a frekvenciasávval rendelkező felhasználók közül néhányan nem is küldenek vagy fogadnak adatokat.

A meglevő csatorna statikus alcsatornákra bontása természetesen még akkor sem lehet hatékony megoldás, ha feltételezzük, hogy a felhasználók számát valahogy konstans N értéken tartjuk. Az alapvető probléma az, hogy amíg a felhasználók nem forgalmaznak, a számukra kijelölt frekvenciatartomány egyszerűen elveszik, mivel ők nem használják, mások pedig nem használhatják. A legtöbb számítógépes rendszer esetében a statikus csatornakiosztás nem jó választás, mert az adatforgalom szélsőségesen löketes jellegű (a maximális és átlagos forgalom aránya gyakran 1000:1), így a csatornák többsége az idő túlnyomó részében kihasználatlan marad.

A statikus FDM alacsony hatékonysága könnyen belátható egy, a sorbanállási elméleten alapuló egyszerű számítással. Számítsuk ki egy keret elküldésének várható késleltetését (T). A csatorna kapacitása legyen C b/s. Tételezzük fel, hogy a véletlenszerűen érkező keretek átlagosan λ keret/másodperc intenzitással érkeznek, és a keretek átlagos hossza $1/\mu$ bit. Ezen paraméterek mellett a csatorna kiszolgálási intenzitása μC keret/s. Az ismert sorbanállási elmélet alapján az eredmény

$$T = \frac{1}{\mu C - \lambda}$$

(Az érdeklődőknek elmondom, hogy ez az eredménye egy „M/M/1" sornak. A képlet alkalmazásának a feltételei, hogy a keretek közötti várakozási idő és a kerethosszak exponenciális eloszlásúak legyenek, más szóval a keretek érkezése legyen Poisson-folyamat.)

Ha például a csatorna kapacitása $C = 100$ Mb/s, az átlagos kerethossz $1/\mu = 10\,000$ bit és a keretek érkezési intenzitása pedig $\lambda = 5000$ keret/s, akkor $T = 200$ μs. Vegyük észre, hogy ha figyelmen kívül hagynánk a sorbaállásból fakadó késleltetést, és csak azt néznénk, hogy mennyi ideig tart egy 10 000 bites keretet elküldeni egy 100 Mb/s-os hálózaton, akkor eredményül (helytelenül) 100 μs-ot kapnánk. Ez az eredmény csak akkor lenne helyénvaló, ha nem folyna versengés a csatornáért.

Most vágjuk a csatornát N darab független alcsatornára, amelyek közül mindegyik C/N b/s kapacitással rendelkezik. Ekkor az átlagos érkezési intenzitás mindegyik alcsatornán λ/N lesz. Újraszámolva a T-t a következő eredményt kapjuk.

$$T_N = \frac{1}{\mu(C/N) - (\lambda/N)} = \frac{N}{\mu C - \lambda} = NT \tag{4.1}$$

Az igények által a rendszerben eltöltött átlagos idő osztott csatorna alkalmazása esetén N-szer nagyobb annál, mintha az összes keret valamilyen varázslatos módon egy nagy központi sorba rendeződött volna. Az eredmény analóg azzal az elgondolással, hogy egy bank előterében elhelyezett pénzkiadó automaták hatékonyabbak, ha egy közös várakozási sor van az automatákhoz, mintha minden automatához külön sorban kell várakozni.

Pontosan ugyanazok az érvek érvényesek más statikus csatornakiosztási megoldásoknál is, mint az FDM-nél. Ha időosztásos nyalábolást (Time Division Multiplexing, TDM) használnánk és minden egyes felhasználóhoz statikusan hozzárendelnénk minden N. időszeletet, és az egyik felhasználó nem használná ki a számára fenntartott időszeletet, akkor az egyszerűen üres maradna. Ez akkor is ugyanígy fennállna, ha a hálózatot fizikailag több részre osztanánk. Az előző példánál maradva, ha egy 100 Mb/s-os hálózat helyett 10 darab 10 Mb/s-os hálózatot használnánk, és mindegyikhez statikusan egy felhasználót rendelnénk, akkor az átlagos késleltetés 200 µs-ról 2 ms-ra nőne.

Mivel a hagyományos statikus hozzárendelési módszerek közül egy sem képes elfogadható hatásfokkal működni löketes forgalom esetén, vizsgáljunk most meg néhány dinamikus megoldást!

4.1.2. Dinamikus csatornakiosztás

Mielőtt rátérnénk a jelen fejezetben tárgyalt rengeteg csatornakiosztási módszerek vizsgálatára, megéri egy kis időt szakítani a csatornakiosztás problémájának pontos megfogalmazására. A munkánk során a következő öt kulcsfontosságú feltételezéssel fogunk élni.

1. **Független forgalom.** A modell N független **állomást (station)** feltételez (például számítógépet, telefont), melyeken egy program vagy egy felhasználó továbbítandó kereteket generál. Annak a valószínűsége, hogy Δt idő alatt generálódik egy keret, $\lambda \Delta t$, ahol λ egy konstans (az új keretek érkezési intenzitása). Ha egy állomás generált egy keretet, akkor blokkolt állapotban marad mindaddig, amíg a keretet sikeresen nem továbbította.

2. **Egyetlen csatorna.** Mindenféle kommunikációhoz egyetlen csatorna vehető igénybe. Minden állomás képes ezen adatokat továbbítani, illetve erről adatokat fogadni. Az állomásokat ugyanolyan képességűnek feltételezzük, bár a protokollok különböző szerepeket és ezzel prioritásokat rendelhetnek hozzájuk.

3. **Megfigyelhető ütközések.** Ha két keretet egyszerre továbbítanak, akkor azok időben átlapolódnak, és az eredményül kapott jel értelmezhetetlenné válik. Az ilyen eseményt **ütközésnek (collision)** nevezzük. Az ütközéseket minden állomás képes észlelni. Az ütközésbe került kereteket később újra kell küldeni. Az ütközéseken kívül semmilyen egyéb hiba nem léphet fel.

4. **Folytonos idő vagy diszkrét idő (slotted time).** Az időt folytonosnak vehetjük, ha a keretek továbbítása bármelyik időpillanatban megkezdődhet. Másik esetben az idő diszkrét intervallumokra (időszeletekre) van osztva. A keretek továbbítása mindig csak az időszeletek elején kezdődhet meg. Egy időszelet 0, 1 vagy több keretet tartalmazhat, és ennek megfelelően nevezzük az időszeletet üresnek, sikeresnek vagy ütközésesnek.

5. **Vivőjel-érzékelés (carrier sense) vagy nincs vivőjel-érzékelés.** Vivőjel-érzékelést feltételezve az állomások meg tudják állapítani, hogy a csatorna foglalt-e, mielőtt megpróbálnák használni. Egyetlen állomás sem próbálja meg használatba venni a csatornát addig, amíg foglaltnak érzékeli. Ha nincs vivőjel-érzékelés, akkor az állomások nem tudják megvizsgálni a csatornát, mielőtt megpróbálnák használatba venni, így egyszerűen elkezdenek adni. Csak ezután tudják eldönteni, hogy az átvitel sikeres volt-e vagy sem.

Néhány magyarázat a fenti feltételezésekhez azok sorrendjében. Az első feltételezés azt mondja, hogy a keretérkezések függetlenek, mind az állomások között, mind egy állomáson belül, és a keretek előre megjósolhatatlan ütemezéssel, de állandó intenzitással keletkeznek. Ez a feltételezés éppenséggel nem kifejezetten jó a hálózati forgalom modellezésére, mivel ismert tény, hogy a csomagok löketekben érkeznek különféle időskálákon [Paxson és Floyd, 1995; és Leland és mások, 1994]. Akárhogyis, a **Poisson-modelleknek** is nevezett modellek hasznosak, mert matematikailag könnyen kezelhetők. Ezek a modellek segítenek a protokollanalízisben, és betekintést engednek abba, hogy a protokollok hogyan teljesítenek különböző működési feltételeknél, valamint jó összehasonlítási alapot nyújtanak a különböző protokollok számára.

Az egyetlen csatorna feltételezése a modell lényege, azaz nincs semmilyen más külső kommunikációs lehetőség. Az állomások nem képesek feltenni a kezüket, hogy a tanár felszólítsa őket, szóval valami jobb megoldással kell előállnunk.

A következő három feltétel az adott rendszer kialakításától függ, és minden egyes vizsgált protokollnál kiemeljük, hogy az alábbi feltételezések közül melyek igazak.

Az ütközések feltételezése szintén alapvető. Az állomásoknak érzékelniük kell az ütközéseket, hogy újraküldhessék a kereteket, mivel nem hagyhatják elveszni. Vezetékes hálózatokban az állomások hardverét lehet úgy tervezni, hogy az állomások azonnal érzékeljék az ütközést. Ezáltal az állomások egy keret küldését még a vége előtt megszakíthatják, ezzel a kapacitásvesztést elkerülhetik. Vezeték nélküli hálózatok esetében az ütközések érzékelése sokkal nehezebb. Az ütközésekre csak abból következtetnek, hogy a várt nyugta nem érkezett meg. A jel és a fogadó hardver tulajdonságaitól függően az is elképzelhető, hogy egy ütközésbe került keretet sikeresen fogadjanak. Ez egyáltalán nem tipikus eset, ezért feltételezzük, hogy minden ütközést elszenvedett keret elvész. Látni fogunk olyan protokollokat, amelyek fő tervezési célja az ütközések bekövetkeztének elkerülése.

Két alternatív feltételezés az időről azért van, mert szeletelt időkezeléssel javíthatunk a rendszer teljesítőképességén. Ámbár, ennek van egy feltétele, amely nem minden esetben teljesül. Megköveteli, hogy az állomások egy központi órát kövessenek vagy egymáshoz szinkronizálják a tevékenységeiket, hogy az időt diszkrét intervallumokra osszák. Mindkét típusú rendszert tárgyalni és vizsgálni fogjuk. Természetesen egy adott rendszerre ezek közül csak az egyik állhat fenn.

Továbbá, egy hálózat lehet vivőjel-érzékeléses vagy vivőjel-érzékelés nélküli. Általában a vezetékes hálózatok vivőjel-érzékeléses hálózatok. A vezeték nélküli hálózatok nem mindig tudják hatékonyan használni a vivőjel-érzékelést, ugyanis nem minden állomás esik bele az összes többi állomás hatósugarába. Továbbá a vivőjel-érzékelés egyáltalán nem használható, ha a két állomás nem tud közvetlenül kommunikálni egymással. Erre példa a kábelmodem esete, amikor az állomások a fejállomáson keresztül kommunikálnak egymással. Itt jegyezném meg, hogy a „vivőjel" (carrier) szó egy csatornán egy jelre utal, és semmi esetre sem a távközlési szolgáltatót (például a telefontársaságot)[1] jelenti.

A félreértések elkerülése végett fontos megjegyezni, hogy egyetlen többszörös hozzáférésű protokoll sem biztosít megbízható átvitelt. Még ha az ütközéseket figyelmen kívül hagyjuk is, a vevő különböző okok miatt hibásan másolhatja be a keretet. Az adatkapcsolati réteg más részei vagy a felsőbb rétegek felelősek a megbízhatóságért.

1 Az angol nyelvben a carrier szónak két jelentése van: (1) vivőjel, (2) távközlési szolgáltató. A szerző e kettősséget kívánja egyértelműsíteni. *(A lektor megjegyzése)*

4.2. Többszörös hozzáférésű protokollok

Többszörös hozzáférésű csatornák kiosztására több algoritmus is ismert. A következő fejezetben megismerkedünk néhány jellegzetes protokollal az érdekesebbek közül, és példákat mutatunk be a használatukra.

4.2.1. ALOHA

Az első MAC-protokoll története az érintetlen Hawaii-szigeteken kezdődött az 1970-es évek elején. Ebben az esetben az „érintetlent" úgy kell értelmezni, hogy „működő telefonhálózattal nem rendelkező". Ez problémát okozott Norman Abramsonnak, a Hawaii Egyetem kutatójának és kollégáinak, akik más szigeteken lévő felhasználókat szerettek volna csatlakoztatni a Honoluluban található központi számítógéphez. Nem volt kivitelezhető az az elképzelés, hogy saját kábeleket fektessenek a Csendes-óceán fenekére, ezért más megoldás után néztek.

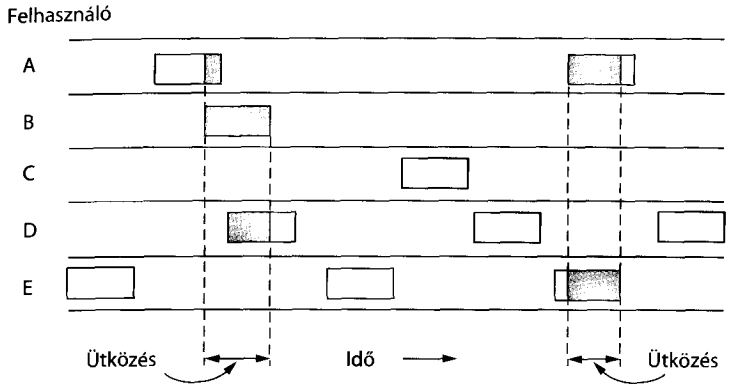
Az általuk talált megoldás rövid hatótávolságú rádiót használt, amelyben minden felhasználói terminál közösen használta ugyanazt a felirányú (upstream) frekvenciasávot a központi számítógéphez irányuló keretek továbbítására. Tartalmazott továbbá egy új, elegáns módszert a csatornakiosztás problémájának megoldására. Munkájukat azóta sok kutató fejlesztette tovább [Schwartz és Abramson, 1985]. Bár Abramson ALOHA-rendszernek nevezett munkáját földi telepítésű rádiós üzenetszóráshoz készítette, az alapötlete bármely olyan rendszerre alkalmazható, amelyben koordinálatlan felhasználók egyetlen megosztott csatorna hozzáférési jogáért versengenek.

Az ALOHA két változatát fogjuk vizsgálni: az egyszerű (pure) és az időszeletelt (slotted) ALOHA-t. Ezek abban különböznek, hogy az időt folytonosnak veszik-e, mint az egyszerű ALOHA esetében, vagy diszkrét időszeletekre osztottnak, amelyekbe minden keretnek el kell férnie.

Egyszerű ALOHA

Az ALOHA-rendszer alapötlete egyszerű: engedjük a felhasználót adni, amikor csak van továbbítandó adata. Természetesen ütközések lesznek, és ezek a keretek el fognak veszni. A küldőnek meg kell tudnia állapítani, hogy ez történt-e. Az ALOHA-rendszerben miután egy állomás elküldte a keretét a központi számítógépnek, a központi számítógép a vett keretet adatszórással visszaküldi minden állomásnak. Emiatt a küldő állomás figyeli a központi számítógép adatszórását azért, hogy megállapítsa, vajon a kerete átjutott-e. Más rendszerekben, például vezetékes LAN-ok esetén, a küldő képes lehet az ütközés érzékelésére a küldés alatt.

Ha a keret megsérült, a küldő egyszerűen véletlenszerű ideig várakozik, majd ismét elküldi a keretet. A várakozási időnek véletlenszerűnek kell lennie, különben ugyanazok a keretek ütköznének újra és újra szabályos időközönként. Azokat a rendszereket, amelyekben a közös csatorna használata konfliktushelyzetek kialakulásához vezethet, **versenyhelyzetes (contention)** rendszereknek nevezzük.



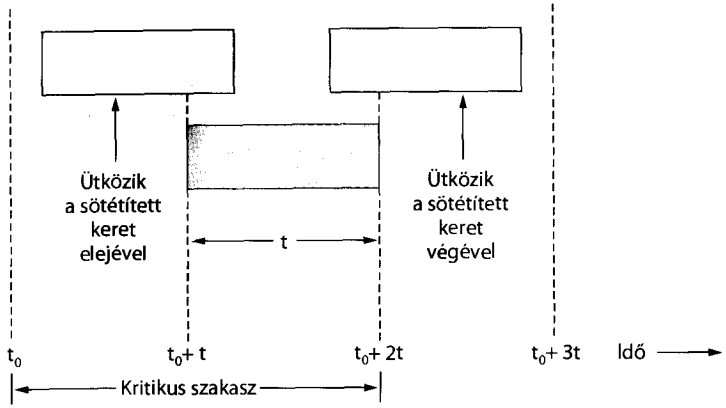
4.1. ábra. *Az egyszerű ALOHA-rendszerben a keretek küldése tetszőleges időpillanatban kezdődhet meg*

Az ALOHA-rendszerben a keretgenerálás vázlatát a 4.1. ábra mutatja. A kereteket egyforma hosszúnak vettük, mivel az ALOHA-rendszerek átbocsátóképessége egyforma keretméretek esetén maximális.

Amikor ugyanabban az időpillanatban két keret is megpróbálja elfoglalni a csatornát, ütközés lép fel (ahogy a 4.1. ábrán látható), és mindkét csomag megsérül. A két keret akkor is használhatatlanná válik (azaz hibás lesz az ellenőrző összege), ha az egyik első bitje éppen hogy ütközik a másik utolsó bitjével, így később mindkét keretet újra kell majd küldeni. Az ellenőrző összeg nem különbözteti meg (és nem is feladata) a teljes ütközést a részlegestől. Ami hibás, az hibás.

Nagyon érdekes kérdés a következő: milyen egy ALOHA-csatorna hatékonysága. Vagyis az elküldött keretek mekkora hányada képes épségben maradni ilyen kaotikus körülmények között? Először képzeljünk el egy végtelen számú felhasználói csoportot, ahol mindenki a terminálja (állomása) előtt ül. Egy felhasználó mindig két állapot közül az egyikben van: vagy gépel, vagy várakozik. Kezdetben minden felhasználó gépel. Amikor egy sor elkészült, a felhasználó befejezi a gépelést és várja a visszajelzést. Ekkor az állomás egy keretbe helyezve továbbítja a sort a központi számítógépnek a megosztott csatornán, miközben figyeli a csatornát, hogy sikeresen átjutott-e a csomag. Ha sikeres volt az átvitel, a felhasználó visszajelzést kap erről, majd tovább gépel. Ha az átvitel nem sikerül, a felhasználó tovább várakozik, mialatt az állomás újra meg újra elküldi a keretet addig, amíg a csatornán egyszer sikeresen át nem jut.

Nevezzük „keretidőnek" azt az időtartamot, amely egy szabványos, fix hosszúságú keret átviteléhez szükséges (azaz a keret hosszát osztva az adatsebességgel)! Tegyük fel, hogy a felhasználók végtelen populációja az új kereteket Poisson-eloszlás szerint állítja elő, keretidőnként átlagosan N keretet. (A végtelen populáció feltételezése azért szükséges, hogy a felhasználók blokkolódása esetén se csökkenjen N értéke.) Ha $N > 1$, akkor a felhasználói közösség nagyobb intenzitással állítja elő a kereteket, mint ahogyan azt a csatorna kezelni képes, így majdnem minden keret ütközést fog elszenvedni. Elfogadható áteresztőképesség $0 < N < 1$ tartományban alakulhat ki.



4.2. ábra. *A sötétített keret ütközésveszélyes szakasza*

Az új keretek elküldése mellett az állomásoknak a régi, ütközést szenvedett keretek újraküldését is el kell végezni. Tegyük még fel, hogy az új és régi keretek együttes elküldési kísérleteinek valószínűsége ugyancsak Poisson-eloszlású, és keretidőnkénti várható értéke G keret. Egyértelmű, hogy $G \geq N$. Kis terhelés (vagyis $N \approx 0$) esetén csak kevés ütközés fordul elő, így az újraadások száma is kicsi, tehát $G \approx N$. Nagy terheléskor sok az ütközés, így $G > N$. Bármekkora is a terhelés, az S áteresztőképességet mindig a G aktuális terhelés, és a sikeres átvitel valószínűségének (P_0) szorzata adja meg. Tehát $S = GP_0$, ahol P_0 annak a valószínűsége, hogy egy keret nem szenved ütközést az átvitel során.

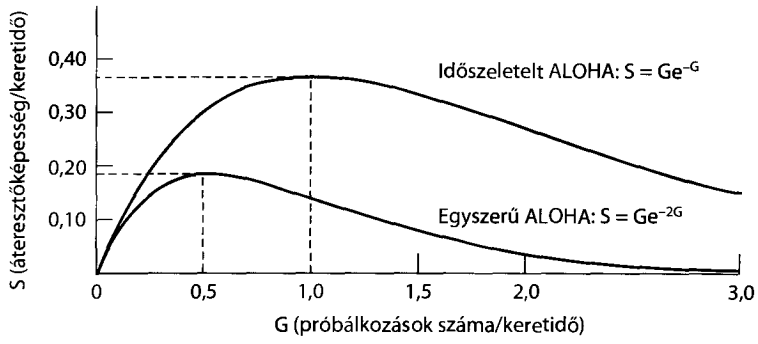
Egy keret akkor nem szenved ütközést, ha elküldésének első pillanatától kezdve egy keretideig nem próbálkozik más állomás keretküldéssel. Ezt szemlélteti a 4.2. ábra. Milyen körülmények között érkezhet meg sértetlenül az ábrán besötétített keret? Legyen t az egy keret elküldéséhez szükséges idő. Ha t_0 és $t_0 + t$ időpontok között valamelyik felhasználó keretet küldött, akkor annak vége ütközni fog a sötétített keret elejével. Igazság szerint, a sötétített keret sorsa már azelőtt megpecsételődött, mielőtt az első bitjét elküldték volna, de mivel az egyszerű ALOHA-rendszerben az állomások nem figyelik a csatornát az adás megkezdése előtt, nem tudhatják, hogy egy keret már úton van. Továbbá, azok a keretek, amelyek küldését a $t_0 + t$ és $t_0 + 2t$ intervallumban kezdik meg, a sötétített keret végével fognak ütközni.

Annak valószínűsége, hogy egy olyan keretidő alatt, amelyben G keretet várunk, k keret keletkezik a feltételezett Poisson-eloszlás szerint:

$$\Pr[k] = \frac{G^k e^{-G}}{k!} \tag{4.2}$$

így annak valószínűsége, hogy nem keletkezik keret: e^{-G}. Két keret hosszúságú időintervallumban a keletkezett keretek várható száma $2G$. Annak valószínűsége tehát, hogy a teljes kritikus szakasz során nem keletkeznek újabb keretek, $P_0 = e^{-2G}$. Az $S = GP_0$ egyenlőséget felhasználva az áteresztőképesség:

$$S = Ge^{-2G}$$



4.3. ábra. *Az ALOHA-rendszerek áteresztőképessége a forgalmi igény függvényében*

Az áteresztőképesség és a forgalmi igény kapcsolatát a 4.3. ábra szemlélteti. Az áteresztőképesség $G = 0{,}5$-nél éri el a maximumát, az $1/(2e)$ értéket, amely megközelítőleg 0,184. Ez annyit jelent, hogy az elérhető legjobb csatornakihasználtság legfeljebb 18%-os lehet. Ez az eredmény nem túl biztató, de egy olyan módszerrel szemben, amelyben mindenki akkor adhat, amikor csak akar, aligha várható el 100%-os kihasználtság.

Időszeletelt ALOHA

Nem sokkal az ALOHA megjelenése után Roberts [1972] olyan módszert publikált, amellyel egy ALOHA-rendszer kapacitása megduplázható. Javaslata szerint az időt **diszkrét szeletekre** (**slot**) kell osztani, amelyek hossza a keretidőkhöz igazodik. Az eljárás megköveteli viszont, hogy a felhasználók megegyezzenek az időintervallumok határainak pontos helyében. A szinkronizálás egyik lehetséges módja, hogy van egyetlen speciális állomás, amely akárcsak egy óra, ütemező jelet bocsát ki minden időintervallum kezdetén.

Roberts módszerében, amely **időszeletelt** (vagy réselt) **ALOHA** (**slotted ALOHA**) néven lett ismert, Abramson **egyszerű ALOHA** (**pure ALOHA**) rendszerével szemben, az állomások nem kezdhetnek el adni bármikor, amikor egy sor begépelése befejeződött, hanem meg kell várniuk a következő időszelet kezdetét. Ezáltal a folyamatos idejű ALOHA diszkrét idejűvé alakul. Ez megfelezi a keretek kritikus szakaszát. Ennek igazolásához, nézzünk a 4.2. ábrára, és képzeljük el a lehetséges ütközéseket. Annak a valószínűsége, hogy a tesztkeretünkkel azonos időszeletben ne legyen egyéb forgalom e^{-G}, ami az

$$S = Ge^{-G} \tag{4.3}$$

összefüggéshez vezet. Ahogy a 4.3. ábrán láthatjuk, az időszeletelt ALOHA $G = 1$-nél éri el áteresztőképességének maximumát $S = 1/e$ azaz megközelítőleg a 0,368 értéket, amely kétszer akkora, mint az egyszerű ALOHA esetén. Ha a rendszer $G = 1$-gyel működik, akkor egy üres időszelet előfordulási valószínűsége (4.2)-ből következően 0,368. Az időszeletelt ALOHA-val elérhető legjobb kihasználtság mellett az időszeletek 37%-a üres,

37%-a sikeres és 26%-a ütközéses lesz. Magasabb G értékek mellett az üres időszeletek száma ugyan csökken, viszont exponenciálisan megnő az ütközéses időszeletek száma. A növekedés ütemének szemléltetése céljából végezzünk el egy egyszerű számítást egy tesztkeret elküldésével kapcsolatban. Annak valószínűsége, hogy elkerüli az ütközést, megegyezik annak valószínűségével, hogy más állomás nem ad az adott időszeletben, vagyis e^{-G}. Az ütközés valószínűsége tehát $1-e^{-G}$. Pontosan k kísérletet követelő átvitel valószínűsége (vagyis a sikeres átvitelt $k-1$ ütközés előzte meg):

$$P_k = e^{-G}(1-e^{-G})^{k-1}$$

Ebből következik, hogy egy sor begépelése után az átviteli kísérletek várható száma, E:

$$E = \sum_{k=1}^{\infty} kP_k = \sum_{k=1}^{\infty} ke^{-G}(1-e^{-G})^{k-1} = e^{G}$$

E-nek G-től való exponenciális függése azt eredményezi, hogy a csatorna terhelésének kis növekedése is drasztikusan csökkentheti a csatorna teljesítményét.

Talán nem magától értetődő, hogy miért is annyira jelentős az időszeletelt ALOHA. Ezt a megoldást már az 1970-es években kidolgozták, aztán néhány korai, kísérleti rendszerben használták is, de később szinte teljesen feledésbe merült. Amikor aztán az internet kábelen keresztül történő elérését feltalálták, a mérnökök hirtelen azzal a problémával találták szembe magukat, hogyan osszanak meg egy közös csatornát több egymással versengő felhasználó között. Ekkor, mintegy a szemétből előbányászva, elővették az időszeletelt ALOHA-t, és a világ meg volt mentve. Később a probléma újra megjelent, amikor több RFID-címke küld adatot ugyanannak az RFID-olvasónak. Az időszeletelt ALOHA, egy csipetnyit hozzákeverve más módszerekből, megint megmentőként érkezett. Gyakran előfordult, hogy teljesen jó protokollokat mellőztek üzletpolitikai okok (például valamelyik nagy társaság azt akarta, hogy mindenki az ő útját járja) vagy az örökké változó műszaki trendek miatt. Aztán évekkel később egy értelmes ember mégis észrevette, hogy egy rég feledésbe merült protokoll megoldja az aktuális problémáját. Éppen ezért ebben a fejezetben mi is tanulmányozni fogunk számos olyan elegáns protokollt, amelyeket jelenleg nem használnak széles körben, de a jövő alkalmazásaiban még minden további nélkül népszerűek lehetnek, feltéve, hogy most elég sok hálózattervező megismeri azokat. Emellett természetesen sok olyan protokollal is foglalkozunk, mely ma is használatos.

4.2.2. Vivőjel-érzékeléses többszörös hozzáférésű protokollok

Időszeletelt ALOHA használatával az elérhető legjobb csatornakihasználtság $1/e$. Ez az alacsony hatékonyság nem meglepő, mivel az állomások tetszés szerint adhatnak anélkül, hogy a többi állomás tevékenységét figyelembe vennék. Ez elkerülhetetlenül sok ütközéssel jár. A helyi hálózatokban (LAN-okban) azonban az állomások általában érzékelhetik más állomások tevékenységét, így viselkedésüket azokhoz igazíthatják. Ennek köszönhetően ezek a hálózatok $1/e$-nél sokkal jobb csatornakihasználtságot érhetnek el. Ebben a szakaszban néhány olyan protokollt mutatunk be, amelyek megnövelik a csatorna teljesítőképességét.

Azokat a protokollokat, amelyekben az állomások figyelik a csatornán folyó forgalmat, és ennek megfelelően cselekszenek, **vivőjel-érzékeléses protokollnak** (**carrier sense protocols**) vagy **csatornafigyelő protokollnak** nevezik. A kutatók számos ilyen protokollt javasoltak, melyeket már régen részletesen kielemeztek, például lásd Kleinrock és Tobagi [1975] munkáját. Az alábbiakban a vivőjel-érzékeléses protokollok több verzióját is bemutatjuk.

Perzisztens és nemperzisztens CSMA

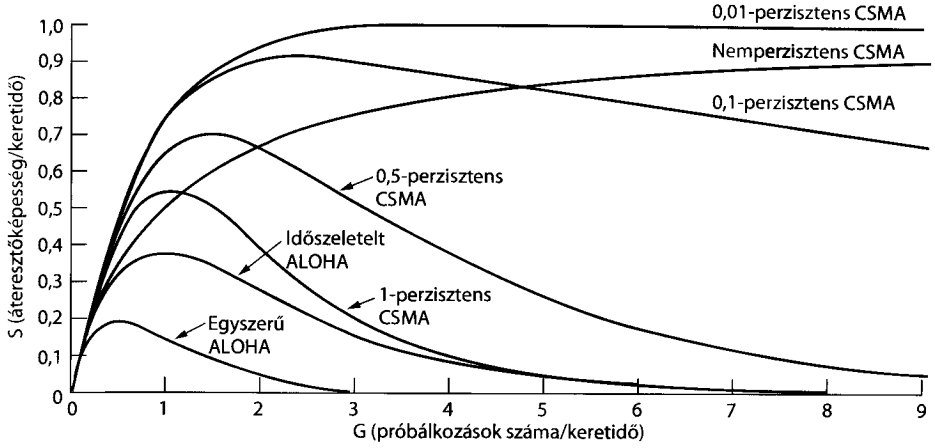
Az első vivőjel-érzékeléses protokoll az **1-perzisztens CSMA** (**Carrier Sense Multiple Access – vivőjel-érzékeléses többszörös hozzáférés**). Ez egy kicsit körülményes név a legegyszerűbb CSMA-sémának. Amikor egy állomás adni készül, először belehallgat a csatornába, hogy eldönthesse, használja-e azt éppen egy másik állomás. Ha a csatorna szabad, az állomások elküldik a kereteiket. Különben, ha a csatorna foglalt, akkor addig vár, amíg az ismét szabad nem lesz. Ekkor az állomás elküld egy keretet. Ha ütközés következik be, akkor az állomás véletlen hosszúságú ideig vár, majd újból elölről kezdi az egészet. A protokollt 1-perzisztensnek nevezik, mivel a várakozó állomás 1 valószínűséggel adni kezd, amint szabadnak érzékeli a csatornát.

Azt várnánk, hogy ez a séma az egyszerre küldés ritka esetét kivéve elkerüli az ütközéseket, de valójában nem ez a helyzet. Ha két állomás lesz adásra kész, miközben egy harmadik a csatornát használja, mindketten udvariasan várnak az adás végéig, és akkor mindkettő pontosan egyszerre fogja elkezdeni az adását, ami ütközést okoz. Ha kevésbé lennének türelmetlenek, akkor kevesebb ütközés lenne.

Egy kicsit pontosabban fogalmazva, a terjedési késleltetésnek jelentős befolyása van az ütközésekre. Van esélye annak, hogy miután egy állomás adni kezd, egy másik állomás éppen adásra kész állapotba fog kerülni és érzékeli a csatorna állapotát. Ha az első állomás által küldött jel nem éri el a másodikat, akkor az utóbbi szabadnak érzékeli a csatornát, és szintén adni kezd, így ütközés következik be. Ennek az esélye függ attól, hogy hány keret fér el a csatornán, vagyis hogy mekkora a csatorna **sávszélesség-késleltetés szorzata**. Ha a keretnek csak egy kis töredéke fér el a csatornán – és ez a helyzet a legtöbb LAN esetén –, mivel kicsi a terjedési késleletetés, az ütközés bekövetkeztének az esélye kicsi. Minél nagyobb a sávszélesség-késleltetés szorzat, annál fontosabbá válik ez a hatás, és annál rosszabb lesz a protokoll teljesítőképessége.

Mindezek ellenére, ennek a protokollnak nagyobb a teljesítőképessége, mint az egyszerű ALOHA-nak, mert mindkét állomás van annyira illedelmes, hogy tartózkodik a harmadik állomás által küldött keret zavarásától. Pontosan ugyanez érvényes az időszeletelt ALOHA-ra is.

A második vivőjel-érzékeléses protokoll a **nemperzisztens CSMA** (**nonpersistent CSMA**). Ebben a protokollban tudatosan arra törekedtek, hogy az állomások ne legyenek annyira mohók, mint az előző protokollnál voltak. Mint ahogy az előbb, itt is az állomás figyeli a csatornát, amikor küldeni akar, és ha senki sem forgalmaz, akkor maga kezd el adni. Ha azonban a csatorna már foglalt, nem folytatja folyamatosan a csatorna figyelését,



4.4. ábra. *Véletlen hozzáférésű protokollok összehasonlítása a terhelés függvényében mért csatornakihasználtság alapján*

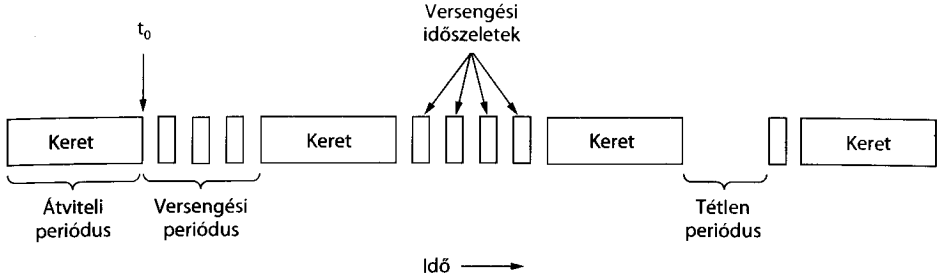
hogy a forgalom megszűntével azonnal megkezdje az adást, hanem véletlen hosszúságú ideig várakozik, és ekkor újrakezdi az algoritmust. Ebből következően, ez az algoritmus jobb kihasználtságot, de nagyobb késleltetéseket okoz, mint az 1-perzisztens CSMA.

Az utolsó protokoll a **p-perzisztens CSMA** (**p-persistent CSMA**). Időszeletelt csatornát alkalmaz, és a következőképpen működik. Amikor egy állomás adásra kész állapotba kerül, megvizsgálja a csatornát. Ha szabad, akkor p valószínűséggel forgalmazni kezd, azaz $q = 1 - p$ valószínűséggel visszalép szándékától a következő időszeletig. Ha a következő időszeletben a csatorna még mindig szabad, akkor ismét p, illetve q valószínűséggel ad vagy visszalép. Ez a folyamat addig folytatódik, amíg a keret elküldésre nem kerül vagy egy másik állomás forgalmazni nem kezd. Az utóbbi eset ugyanolyan hatású, mintha ütközés következett volna be (azaz véletlen hosszúságú ideig vár, majd elölről kezdi az algoritmust). Ha az állomás már kezdetben érzékeli a csatorna foglaltságát, akkor vár a következő időszeletig, és csak ott kezdi a fent leírt algoritmust. Az IEEE 802.11 szabvány a p-perzisztens CSMA egy finomított változatát használja, amit a 4.4. szakaszban fogunk tárgyalni.

A 4.4. ábra nem csak e három protokoll, hanem az egyszerű és időszeletelt ALOHA-protokollok áteresztőképességét is szemlélteti a forgalmi igények függvényében.

CSMA ütközésérzékeléssel

A perzisztens és nemperzisztens CSMA-protokollok egyértelmű előrelépést jelentenek az ALOHA-rendszerhez képest, hiszen az állomások nem kezdenek el adni, ha a csatornát foglaltnak érzékelik. Ennek ellenére, ha két állomás szabadnak érzékeli a csatornát és mindketten egyszerre elkezdenek adni, akkor a küldött jeleik ütközni fognak. További fejlődés, hogy az állomások gyorsan érzékelik az ütközést és azonnal félbeszakítják adásukat (az adás végigvitele helyett), mert a keretek már visszavonhatatlanul megsérültek. Ez a stratégia időt és sávszélességet takarít meg.



4.5. ábra. *A CSMA/CD mindig a következő állapotok közül az egyikben lehet: versengési, átviteli vagy tétlen*

Ezt a protokollt **CSMA/CD-nek** (**Carrier Sense Multiple Access with Collision Detection – ütközésérzékeléses CSMA**) nevezik, és ez képezi a klasszikus Ethernet LAN-ok alapját, tehát érdemes némi időt szánni a részletes megismerésére. Fontos tisztában lenni azzal, hogy az ütközés érzékelése egy analóg folyamat. Az állomás hardverének az adás folyamán figyelnie kell a csatornát. Ha a visszaolvasott jelé különbözik a kiadott jeltől, akkor tudja, hogy ütközés keletkezett. A feltétel az, hogy a vett jel ne legyen nagyon kicsi a küldött jelhez képest (ezt nehéz biztosítani vezeték nélküli környezetben, mivel a vett jelek 1 000 000-szor gyengébbek lehetnek a küldött jeleknél), és hogy olyan moduláció kell, amely lehetővé teszi az ütközések érzékelését (például két 0 volt feszültségű jel ütközését szinte lehetetlen érzékelni).

A CSMA/CD- és sok más LAN-protokoll a 4.5. ábrán látható fogalmi modellt használja. A t_0-val jelölt ponton egy állomás befejezi keretének küldését. Ezen a ponton minden olyan állomás, amelyik kész kerettel rendelkezik, megkísérelheti azt elküldeni. Ha kettő vagy több állomás egyszerre kezd el adni, akkor ütközés jön létre. Ha egy állomás ütközést érzékel, félbehagyja az adást, véletlen hosszúságú ideig vár, majd újrapróbálkozik (feltételezve, hogy egyetlen másik állomás sem kezdett bele az adásba ezalatt). Ennek következtében a CSMA/CD-modellünk váltakozó versengési és átviteli periódusokból áll, kiegészítve tétlen periódusokkal, amikor egyik állomás sem forgalmaz (például nincs továbbítandó adat).

Most pedig ismerkedjünk meg a versengés algoritmusának részleteivel! Tegyük fel, hogy két állomás pontosan ugyanabban a t_0 időpillanatban kezd el adni. Mennyi idő telik el, amíg észlelik, hogy ütköztek? Ennek a kérdésnek a megválaszolása döntő jelentőségű a versengési periódus hosszának meghatározásához, és ebből következően a késleltetés és az áteresztőképesség kiszámításához.

Az ütközés érzékeléséhez szükséges minimális idő az, ami alatt a jel az egyik állomástól a másikig eljut. Ezen információ alapján azt hihetnénk, hogy egy állomás, amely a kábel teljes terjedési idejének megfelelő ideig nem észlel ütközést, biztos lehet abban, hogy megszerezte a kábel használati jogát. A „megszerezte" kifejezésen azt értjük, hogy az összes többi állomás tud a folyó átvitelről, és így nem zavarják meg azt. Ez a következtetés azonban hibás.

Tekintsük a következő, legrosszabb eshetőséget! Jelöljük a két legtávolabb levő állomás között a jel terjedési idejét τ-val! A t_0 időpillanatban az egyik állomás elkezd adni.

$t_0 + \tau - \varepsilon$ időpontban, tehát egy pillanattal azelőtt, hogy a jel megérkezhetett volna a legtávolabbi állomáshoz, az is forgalmazni kezd. Természetesen szinte azonnal észreveszi az ütközést és leáll, de az ütközés által okozott kis zaj nem jut vissza az első állomáshoz $2\tau - \varepsilon$ időn belül. Vagyis legrosszabb esetben egy állomás csak akkor lehet biztos abban, hogy megszerezte a csatornát, ha már 2τ ideje forgalmaz ütközés nélkül.

Ennek tudatában úgy gondolhatunk a CSMA/CD versengésre, mintha egy 2τ hosszúságú időszeletelt ALOHA-rendszer lenne. Egy 1 km hosszúságú koaxiális kábelen $\tau \approx 5$ μs. A CSMA/CD abban különbözik az időszeletelt ALOHA-tól, hogy azokat az időszeleteket, amelyekben csak egyetlen állomás ad (amelyekben a csatorna foglalt), a keret további részei követik. Ha a keretidő jóval hosszabb a terjedési időnél, akkor ez a különbség nagyban javítja a teljesítőképességet.

4.2.3. Ütközésmentes protokollok

Bár a CSMA/CD esetén nincs ütközés, ha egy állomás már egyértelműen megszerezte a csatornát; azt megelőzően, a versengési szakaszban azonban még mindig előfordulhatnak ütközések. Ezek az ütközések hátrányosan érintik a rendszer teljesítőképességét, különösen akkor, ha a sávszélesség-késleltetés szorzat nagy, úgy mint amikor a kábel hosszú (azaz τ nagy), a keretek pedig rövidek. Az ütközések miatt nemcsak a sávszélesség csökken, de a keretek küldési ideje is változó lesz, ami nem ideális valós idejű forgalom számára, mint amilyen az IP-hálózaton keresztül történő hangátvitel (VoIP). A CSMA/CD sem mindenhol alkalmazható.

Ebben a szakaszban olyan protokollokat mutatunk be, melyek ütközés nélkül teszik lehetővé a csatornáért folytatott verseny lebonyolítását, még a versengési periódus folyamán is. Legtöbbjüket nem használják a jelenlegi nagyobb rendszerekben, de ilyen gyorsan változó területen mégis hasznos lehet, ha van néhány kiváló tulajdonságokkal bíró protokollunk az esetleges jövőbeli rendszerek számára.

A protokollok bemutatásánál feltételezzük, hogy pontosan N állomás van, és hogy ezeket az állomásokat 0-tól $N-1$-ig terjedő egyedi címmel látták el. Nem okoz problémát, ha néhány állomás az idő egy részében inaktív. Feltételezzük azt is, hogy a terjedési késleltetés elhanyagolható. Az alapvető kérdés az marad, melyik állomás használhatja a csatornát egy-egy sikeres átvitel befejeztével. Továbbra is a 4.5. ábrán bemutatott diszkrét versengési időszeletekkel dolgozó modellt fogjuk használni.

Helyfoglalásos protokoll

Az első bemutatandó ütközésmentes protokollban, az **alapvető bittérkép-eljárásban** (**basic bit-map method**) az ütköztetési periódus pontosan N időszeletből áll. Ha a 0-s állomás adni szeretne, akkor 1-es bitet küld a 0-s (első) versengési időszeletben. Ez alatt az időszelet alatt a többi állomás nem használhatja a csatornát. A 0-s állomástól függetlenül, az 1-es állomásnak szintén megvan a lehetősége, hogy az 1-es (második) időszelet jelzőbitjét 1-re állítsa, ha van kész kerete. Általánosan a j-edik állomás a j-edik időszeletben jelezheti egy 1-es bittel, ha van elküldésre váró kerete. Az N darab időszelet letelte után mindegyik

4.6. ábra. *Az alapvető bittérkép-protokoll*

állomás pontosan tudja, hogy mely állomások szeretnének forgalmazni. Ekkor számsorrendben megkezdhetik a keretek továbbítását, mint ahogy azt a 4.6. ábra is szemlélteti.

Mivel az állomások megegyeztek, hogy a sorban mikor melyikük következik, sohasem jöhet létre ütközés. Miután az utolsó adni kívánó állomás is elküldte a keretét – mely eseményt minden állomás könnyedén észlelhet –, egy újabb N időszeletes versengési periódus veszi kezdetét. Ha egy állomás szerencsétlenségére éppen akkor lesz adásra kész, amikor már éppen elmúlt a számára fenntartott versengési időszelet, akkor kénytelen az aktuális körben csendben maradni, és megvárni, amíg a versengési periódus számára fenntartott időszelete ismét körbeér.

Azokat a protokollokat, amelyekben – ehhez hasonlóan – a forgalmazási igényt a tényleges adatátvitel előtt kell adatszórással (broadcast) jelezni, **helyfoglalásos protokollnak** (**reservation protocol**) nevezik, mert előre lefoglalják maguknak a csatornát és megelőzik az ütközést. Röviden vizsgáljuk meg e protokoll teljesítményét. Az egyszerűség kedvéért az időt az 1 bit hosszú versengési időszeletekben mérjük, és egy-egy adatkeretet d hosszúságúnak tételezünk fel.

Ha a terhelés kicsi, akkor továbbítandó adatkeretek hiányában a versengési bittérkép fog újra meg újra ismétlődni a csatornán. Nézzük a helyzetet egy alacsony (például 0 vagy 1) azonosítójú állomás szemszögéből. Amikor küldésre kész állapotba kerül, az „aktuális" rés általában valahol a bittérkép közepén fog járni. Átlagosan tehát $N/2$ időszeletet kell egy ilyen állomásnak várnia az aktuális kör végéig, majd N időszeletet, amíg a következő versengési periódus befejeztével megkezdheti a forgalmazást.

A magasabb sorszámú állomások helyzete már kedvezőbb. Ezeknek általában a versengési periódus felét ($N/2$ időszeletet) kell csak várniuk az adás megkezdése előtt. Ezeknek az állomásoknak csak nagyon ritkán kell kivárniuk a következő versengési periódust. Mivel az alacsony sorszámú állomásoknak átlagosan $1{,}5N$ időszeletet, a magas sorszámúaknak pedig $0{,}5N$ időszeletet kell várniuk, az összes állomásra számított átlagos várakozási idő N időszelet lesz.

Alacsony terhelés mellett tehát egyszerűen számítható a csatorna hatékonysága. Keretenként a d adatbitre N többletbit (overhead) jut, így a hatékonyság $d/(d+N)$.

Nagy terhelés esetén, amikor mindegyik állomásnak mindig van küldenivalója, az N hosszúságú versengési periódus N adatkeret között oszlik meg, így minden keretre csak 1 többletbit adódik. Ebből a csatorna hatékonyságára $d/(d+1)$ adódik. Egy keret átlagos késleltetése két részből tevődik össze: az adott állomáson belüli sorbanállási késleltetésből, valamint további $(N-1)d + N$ időszeletnyi késleltetésből, amelyet a belső sor elejére kerülve fog várni a tényleges elküldésig. Ez utóbbi intervallum abból áll, hogy mennyi időt kell várnia arra, hogy az összes többi állomás elküldhesse saját keretét és még egy bittérképből.

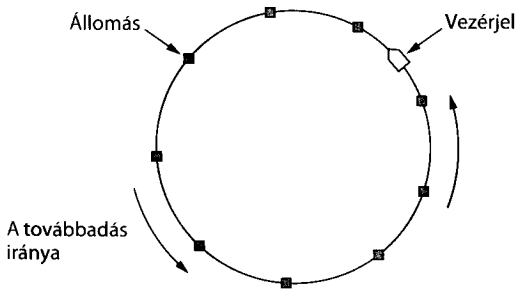
Vezérjeles gyűrűprotokoll

A bittérkép-protokoll lényege, hogy lehetővé teszi minden állomás számára azt, hogy egy előre meghatározott sorrend szerint elküldjön egy keretet. Egy másik módszer erre az, hogy egy rövid üzenetet, amit **vezérjelnek** (**tokennek**) hívnak, továbbadnak az egyik állomástól a másikig ugyanabban az előre meghatározott sorrendben. A vezérjel jelképezi az adásra való feljogosítást. Ha egy állomásnak van küldésre várakozó kerete, amikor a vezérjelet megkapja, akkor elküldheti a keretet, mielőtt továbbadná a vezérjelet a következő állomásnak. Ha az állomásnak nincs küldenivaló kerete, egyszerűen továbbadja a vezérjelet.

A **vezérjeles gyűrűprotokoll** (**token ring protocol**) esetén a hálózat topológiája határozza meg az állomások adásának a sorrendjét. Az állomások egymásután egy gyűrűbe vannak kapcsolva. A vezérjel továbbadása a következő állomáshoz egyszerűen abból áll, hogy egy állomás fogadja az egyik irányból a vezérjelet és továbbadja a másik irányba, ahogy ez a 4.7. ábrán látható. A keretek is a vezérjel haladásának irányában továbbítódnak. Ennek megfelelően a keretek körbehaladnak a gyűrű mentén, és elérik a címzett állomást. Hogy azonban egy adatkeret ne körözzön a hálózatban a végtelenségig (úgy, mint a vezérjel), valamelyik állomásnak el kell távolítania a gyűrűről. Ez lehet az az állomás, amelyik eredetileg küldte a keretet, vagy az, amelyik a keret szándékolt célállomása.

Érdemes megjegyezni, hogy nem szükséges, hogy a hálózat fizikailag gyűrűt alkosson ahhoz, hogy megvalósítható legyen a vezérjel továbbadása (token passing). Az állomásokat összekötő csatorna lehet egy sín is. Minden állomás ezt a sínt használja, hogy a vezérjelet a meghatározott sorrend szerinti következő állomásnak küldje. A vezérjel birtoklása ugyanúgy feljogosítja az állomást, hogy keretet küldjön a sínen, mint eddig. Ezt a protokollt **vezérjeles sínnek** (**token bus**) hívják.

A vezérjel-továbbadó protokoll teljesítőképesége hasonló, mint a bittérkép-protokollnak, bár az egy perióduson belül a versengési időszeletek és a keretek sorrendje itt összekeveredett. Miután egy állomás elküldte a keretét, meg kell várnia, hogy mind az N állomás (magát is beleértve) továbbadja a szomszédjának a vezérjelet, és hogy a többi $N-1$ állomás elküldje a keretét, amennyiben van egyáltalán elküldendő kerete. Egy apró különbség az, hogy mivel a gyűrűben minden pozíció azonos, nincs eltérés a kis és nagy sorszámú azonosítójú állomások között. A vezérjeles gyűrű esetén is minden állomás csak a szomszéd állomásig küldi a vezérjelet, mielőtt a protokoll szerinti következő lépésre sor kerül. Nem szükséges a vezérjelnek az összes állomást elérnie, mielőtt a protokoll szerinti következő lépés bekövetkezik.



4.7. ábra. *Vezérjeles gyűrű*

A vezérjeles gyűrű protokollok, nagyjából változatlan formában MAC-protokollokként jelentek meg. Egy korai vezérjeles gyűrű protokoll (a „Token Ring"-nek nevezett és IEEE 802.5 szabványban specifikált protokoll) népszerű volt az 1980-as években, és a klasszikus Ethernet alternatívájaként szerepelt. Az 1990-es években egy sokkal gyorsabb vezérjeles gyűrű protokollt az **FDDI**-t (**Fiber Distributed Data Interface – fényvezetőszálas osztott adatinterfész**) a kapcsolt Ethernet ütötte ki. A 2000-es években az **RPR**-nek (**Resilient Packet Ring – ellenálló csomagkapcsolt gyűrű**) nevezett vezérjeles gyűrűt IEEE 802.17 néven szabványosították, hogy egységesítsék az internetszolgáltatók által használt különféle nagyvárosi gyűrűhálózatokat. Kíváncsian várjuk, mit hoznak a 2010-es évek.

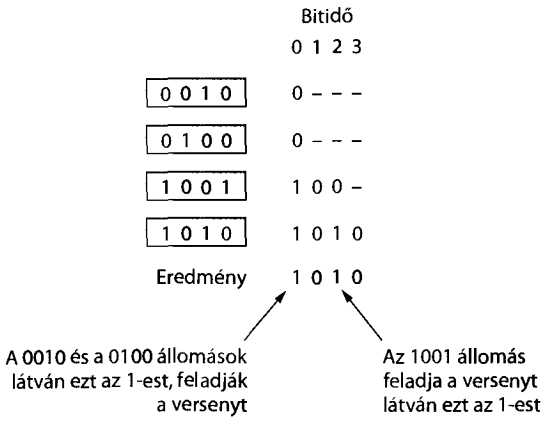
Bináris visszaszámlálás protokoll

A bittérkép-, valamint a vezérjel-továbbadó protokoll egyik hátránya az, hogy a versengési periódus hossza állomásonként 1 bittel nő, így rosszul skálázható már néhány ezer állomást tartalmazó hálózatokra is. Jobb eredményeket érhetünk el, ha bináris állomáscímeket és egy olyan csatornát használunk, amely kombinálja az átviteleket. Ez esetben a forgalmazni kívánó állomás adatszórással elkezdi mindenkinek szétküldeni a bináris címét, a legnagyobb helyi értékű bittel kezdve. Az összes állomás címének azonos hosszúságúnak kell lennie. Az elküldött címek ugyanabban a pillanatban elküldött azonos helyi értékű bitjeit a csatorna egymással logikai VAGY kapcsolatba hozza. Ezt a protokollt **bináris visszaszámlálásnak** (**binary countdown**) nevezzük. Ezt használják a Datakit-ben is, melyet Fraser [1987] tárgyal bővebben. A protokoll hallgatólagosan feltételezi, hogy az átviteli késleltetések elhanyagolhatók, azaz a leadott biteket minden állomás lényegében azonnal érzékeli.

A konfliktusok elkerülése érdekében szükség van egy kiegészítő szabályra is: amint egy állomás észleli, hogy 1-essel lett felülírva egy olyan, magasabb helyi értékű címbitpozíció, ahol a saját címében 0-s van, fel kell adnia a próbálkozást. Például ha a 0010, 0100, 1001 és 1010 című állomások szeretnék használni a csatornát, akkor az első bitidőben sorrendben 0-st, 0-st, 1-est és 1-est küldenek. Ezek logikai VAGY kapcsolata 1-est eredményez. A 0010 és a 0100 állomások látván az 1-est, és megtudván, hogy a versenyben magasabb című állomás is részt vesz, feladják a versenyt. Az 1001 és 1010 állomások tovább folytatják a versengést.

A következő bit 0-s, így mindkét állomás versenyben marad. Az ezt követő bit azonban 1-es, így az 1001 című állomás feladja a versengést. A győztes tehát az 1010 állomás lesz, mivel övé a legnagyobb cím. Miután megnyerte a „licitálást", továbbíthat egy keretet, amely után újabb verseny kezdődik a forgalmazás jogáért. A protokoll működését a 4.8. ábra illusztrálja. Megfigyelhetjük azt a sajátosságot, hogy a magasabb című állomásoknak a prioritásuk is magasabb az alacsonyabb című állomásokénál, ami lehet jó is, rossz is, az adott környezettől függően.

A csatornakihasználtság ilyen protokoll mellett $d/(d+\log_2 N)$. Abban az esetben viszont, ha okosan választjuk meg a keretek felépítését, és az első mező éppen a küldő



4.8. ábra. *A bináris visszaszámlálás protokoll. A „-" azt jelzi, hogy az állomás nem forgalmaz*

címét tartalmazza, még ez a $\log_2 N$ bit sem vész kárba, így a csatorna kihasználtsága 100% is lehet!

A bináris visszaszámlálás jó példája az olyan egyszerű, elegáns és hatékony protokolloknak, melyek újrafelfedezésre várnak. Remélhetőleg egy nap ez is megtalálja majd új helyét.

4.2.4. Korlátozott versenyes protokollok

Eddig két alapvető csatorna-megszerzési stratégiát tárgyaltunk adatszóró hálózatok esetén: a versenyhelyzetes (mint amilyen a CSMA) és az ütközésmentes protokollokat. Mindkét stratégiát megítélhetjük két fontos teljesítménymérő szám, a kis terhelés mellett fellépő késleltetés, illetve a nagy terhelés mellett fennálló csatornakihasználtság alapján. Kis terhelés esetén a versenyhelyzetes módszerek (azaz az egyszerű és az időszeletelt ALOHA) a kedvezőbbek kis késleltetésük miatt (mert ritkán fordulnak elő ütközések). Ahogy nő a terhelés, a versenyhelyzetes protokollok egyre kevésbé vonzók, mivel egyre növekszik a csatorna megszerzésével eltöltött idő. Az ütközésmentes protokollokra ennek éppen az ellenkezője igaz. Kis terhelés mellett viszonylag nagy a késleltetésük, de ahogy a terhelés növekszik, a csatorna kihasználtsága egyre javul (mert a csatorna megszerzésével töltött idő rögzített hosszúságú).

Nyilvánvaló, hogy szerencsés lenne ötvözni a versenyhelyzetes és ütközésmentes protokollok legjobb tulajdonságait, és olyan új protokollt tervezni, amely kis terhelés esetén versenyhelyzetes technikát használna a kis késleltetés érdekében, illetve nagy terhelés mellett ütközésmentes technikát alkalmazna a csatorna jó kihasználása érdekében. Ilyen, **korlátozott versenyes protokollok (limited contention protocol)** már léteznek, és ezekkel zárjuk a vivőjel-érzékeléses protokollok tanulmányozását.

Az összes eddig tanulmányozott versenyhelyzetes protokoll szimmetrikus volt, vagyis az állomások p valószínűséggel próbálták megszerezni a csatornát, ahol a p minden állomásra azonos értékű volt. Érdekes, hogy a teljes rendszer teljesítményének növelése érdekében néha elég, ha olyan protokollt használunk, amely az állomásokhoz különböző valószínűségeket rendel.

Mielőtt áttérnénk az aszimmetrikus protokollok vizsgálatára, röviden tekintsük át a szimmetrikus protokollok teljesítményviszonyait. Tegyük fel, hogy a csatorna megszerzéséért minden résben k állomás verseng, és mindegyik p valószínűséggel adhat. Annak valószínűsége, hogy egy állomás sikeresen megszerzi a csatornát egy adott időszeletben az az, hogy csak egy állomás ad, p valószínűséggel, és a többi $k-1$ állomás közül egyik sem akar adni egyenként $1-p$ valószínűséggel. Ez az érték $kp(1-p)^{k-1}$. Az optimális p megtalálása érdekében deriváljuk a kifejezést p szerint, az eredményt nullával egyenlővé tesszük, majd megoldjuk p-re az egyenlőséget. Ezt végrehajtva, p-re az $1/k$ értéket kapjuk, amelyet ha behelyettesítünk az eredeti kifejezésbe, a következő összefüggést kapjuk:

$$\Pr[\text{siker optimális } p \text{ mellett}] = \left(\frac{k-1}{k}\right)^{k-1} \tag{4.4}$$

A valószínűség-függvény grafikonját a 4.9. ábrán láthatjuk. Kis állomásszám mellett a csatorna megszerzésének valószínűsége jó, de már öt állomás esetén is az esélyek az aszimptotikus $1/e$ érték közelébe zuhannak le.

A 4.9. ábra alapján nyilvánvaló, hogy egy állomás csatorna-megszerzési esélyeit növelni csak a versenyhelyzetek számának csökkentésével lehet. A korlátozott versenyes protokoll pontosan ezt teszi. Először is az állomásokat (nem feltétlenül diszjunkt) csoportokra osztják. A 0-s résért csak a 0-s csoport tagjai versenghetnek. Ha valamelyikük nyer, akkor megszerzi a csatornát, és elküldi a keretét. Ha viszont ütközés fordult elő, vagy a rés kihasználatlan maradt, akkor máris az 1-es csoport tagjai versengenek az 1-es résért, és így tovább. Az állomások megfelelő csoportokra osztásával a résekre jutó versenyhelyzetek száma csökkenthető, így a rések a 4.9. ábra bal oldalához hasonló karakterisztikával működhetnek.

A trükk abban van, ahogyan az állomásokat a résekhez rendeljük. Mielőtt az általános esetet megvizsgálnánk, nézzünk meg néhány speciális esetet! Az egyik szélső eset az, amikor mindegyik csoport csak egy állomást tartalmaz. Az ilyen eset biztosítja, hogy ne legyen ütközés, hiszen résenként legfeljebb egy állomás versenghet a csatornáért. Ilyen protokollt már láttunk korábban (például bináris visszaszámlálás). A következő speciá-

4.9. ábra. *A csatorna megszerzésének valószínűsége szimmetrikus versenyhelyzetes protokoll esetén*

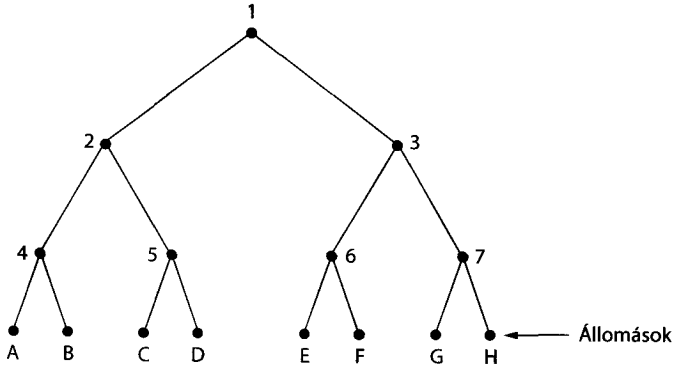
lis esetben csoportonként két állomás van. Annak esélye, hogy ezek egy résben egyszerre akarjanak adni p^2, amely kis p-re elhanyagolható. Az azonos réshez tartozó állomások számának növelésével nő az ütközések valószínűsége, viszont a bittérkép mérete csökken anélkül, hogy állomások elvesztenék adási lehetőségüket. A határeset az, amikor az összes állomás egy csoportba kerül, ez az időszeletelt ALOHA. Olyan dinamikus állomás-hozzárendelésre lenne szükségünk, amely egy réshez kis terhelés esetén sok, míg nagy terhelés esetén csak néhány (esetleg csak egy) állomást rendelne.

Adaptív fabejárási protokoll

A hozzárendelés egyik legegyszerűbb módja az, ahogy a II. világháború alatt az amerikai hadseregben a katonák szifiliszes fertőzöttségét vizsgálták [Dorfman, 1943]. Röviden ezt úgy végezték, hogy a hadsereg N katonájától vért vettek, amelyek mindegyikéből egy kis részt egyetlen kémcsőbe öntöttek, és ezt vizsgálták antitestek után kutatva. Ha nem találtak, akkor az összes abba a csoportba tartozó katona egészséges volt. Ha azonban találtak antitestet, akkor az N katonát két csoportba osztották, és a vérmintákból két újabb keveréket állítottak elő. Az egyik keverék az első $N/2$ katonától származik, a második a többitől. A folyamatot addig ismételték, amíg meg nem találták a fertőzött katonát.

Az algoritmus számítógépes változatához [Capetanakis, 1979] kényelmes, ha az állomásokat a 4.10. ábrának megfelelően egy bináris fa leveleinek képzeljük el. Egy sikeres keretátvitelt követően az első versengési résben, a 0-s résben, az összes állomás szabadon versenghet a csatorna megszerzéséért. Ha az egyik állomásnak sikerül, akkor minden rendben van. Ha ütközés következik be, akkor az 1-es résben már csak a 2. csomópont alatti részfa állomásai versenyezhetnek. Ha az egyikük megszerzi a csatornát, akkor a keretét követő rés a 3. csomópont alatti állomások számára lesz fenntartva. Ha viszont két vagy több 2. csomópont alatti állomás is forgalmazni szeretett volna az 1-es résben, akkor a bekövetkező újabb ütközés miatt a 2-es résben a 4. csomópont alatti részfa állomásai következhetnek.

Lényegében tehát, ha a 0-s időszelet alatt ütközés következik be, akkor a küldésre kész állomások felkutatása érdekében megkezdődik a fa mélységi bejárása. Az 1 bit hosszú-



4.10. ábra. *Nyolc állomásból álló fa*

ságú időszeletek a fa egyes csomópontjaihoz vannak rendelve. Ha ütközés következik be, akkor a keresés rekurzívan az adott csomópont bal és jobb gyermekcsomópontjánál folytatódik. Ha egy bitrés kihasználatlan marad, vagy csak pontosan egy állomás küld benne, akkor annak a csomópontnak a keresése befejeződik, hiszen alatta nincs több küldésre kész állomás. (Ha ugyanis több is lett volna, akkor ütközésnek kellett volna bekövetkeznie.)

Amikor a rendszer terhelése nagy, aligha éri meg a 0-s rést az 1. csomóponthoz rendelni, hiszen csak abban a valószínűtlen esetben nem következne be ekkor ütközés, ha legfeljebb csak egyetlen állomás rendelkezne küldésre kész kerettel. Hasonló megfontolásból lehetne érvelni a 2. és 3. csomópont átugrására is. A kérdést általánosan fogalmazva: melyik szinten érdemes elkezdeni a keresést? Világos, hogy minél nagyobb a terhelés, annál mélyebben érdemes kezdeni a keresést a fában. Tegyük fel, hogy az állomásoknak q értékű jó becslése van arra, hogy hány kész állomás van éppen (például az addigi forgalom megfigyeléséből következtetve).

A fa gyökerétől számozzuk meg az egyes szinteket! A 4.10. ábrán az 1. csomópont a 0-s szint, a 2. és 3. csomópont az 1-es szint stb. Vegyük észre, hogy az i-edik szint minden csomópontjához az alattuk levő állomások 2^{-i} része tartozik! Ha a q adásra kész állomás a fában egyenletesen elosztva helyezkedik el, akkor egy i-edik szinten levő csomópont alatt várhatóan $2^{-i}q$ darab van. Mindenféle megfontolás nélkül azt várhatjuk, hogy a keresést azon a szinten optimális elkezdeni, ahol a résenként versengő állomások száma átlagosan 1, azaz azon a szinten, ahol $2^{-i}q = 1$. Az egyenletet megoldva az $i = \log_2 q$ értéket kapjuk.

Bertsekas és Gallager [1992] az alapalgoritmus számos továbbfejlesztett változatát állította elő és vizsgálta meg. Például tekintsük azt az esetet, amikor csak a G és H állomás akar adni. Az 1-es csomópontnál ütközés következik be, így a 2-esre kerül a sor, amikor üres marad a csatorna. Teljesen felesleges lenne a 3-as csomópont vizsgálata, hiszen biztos, hogy ütközés fog bekövetkezni, mivel tudjuk, hogy az 1-es csomópont alatt kettő vagy több kész állomás is van, viszont a 2-es alatti részfában egy sincs, így ezeknek mind a 3-as alatt kell lenniük. A 3-as csomópont vizsgálata kimarad, és helyette a 6-os következik. Mikor kiderül, hogy ez alatt a csomópont alatt sincs adásra kész állomás, kihagyható a 7-es tesztelése, és azonnal a G állomásra kerülhet a sor.

4.2.5. Vezeték nélküli LAN-protokollok

Egy olyan rendszer, amelyben hordozható számítógépek rádión keresztül kommunikálnak, már vezeték nélküli LAN-nak nevezhető. Egy ilyen LAN az adatszóró hálózatok egy példája. Ezek a hálózatok bizonyos mértékben különböző tulajdonságokkal rendelkeznek a vezetékes LAN-okhoz képest, ami miatt különböző MAC-protokollokat használnak. Ebben a részben ezeket a protokollokat fogjuk megvizsgálni. A 4.4. szakaszban részletezzük a 802.11 (Wi-Fi) szabványt.

A vezeték nélküli LAN tipikus alkalmazása például, amikor a hálózat egy irodaházban előre meghatározott terv szerint elhelyezett hozzáférési pontokatból (access point, AP) áll. A hozzáférési pontokat rézvezetékes vagy üvegszálas hálózat köti össze egymással, és kapcsolatot létesítenek a velük kommunikáló állomásokkal. Ha a hozzáférési pontok és a

hordozható eszközök (például laptopok) adóteljesítményét úgy állítják be, hogy 10 méteres nagyságrendű hatósugaruk legyen, akkor a közeli szobák olyanok lesznek, mint egy külön cella, a teljes épület pedig, mint egy celluláris telefonrendszer, amiről a 2. fejezetben már beszéltünk, azzal a különbséggel, hogy minden cellának csak egyetlen csatornája van. A cella összes állomása és a hozzáférési pont is közösen használja ezt a csatornát. Az ilyen csatornák sávszélessége tipikusan néhány Mb/s-től 600 Mb/s-ig terjed.

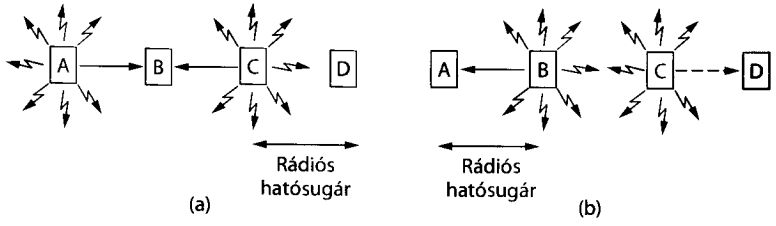
Már korábban megjegyeztük, hogy a vezeték nélküli rendszerek általában nem tudják az ütközéseket a történésük idejében érzékelni. Egy állomás vett jele lehet nagyon gyenge, akár milliószor gyengébb, mint az elküldött jel. Egy ilyet érzékelni olyan, mint tűt keresni egy szénakazalban. Ehelyett, nyugtákat használunk az ütközések és az egyéb hibák utólagos felderítésére.

Van egy még ennél is fontosabb különbség a vezetékes és vezeték nélküli LAN-ok között. A rádióadó korlátozott hatósugara miatt előfordulhat, hogy a vezeték nélküli LAN-on az állomások nem képesek kereteket küldeni az összes többi állomásnak, illetve kereteket venni az összes többi állomástól. Vezetékes LAN esetén, ha egy állomás egy keretet elküld, az összes többi állomás megkapja. Ennek a tulajdonságnak a hiánya a vezeték nélküli hálózatoknál különféle bajok forrása.

Azzal az egyszerűsítő feltételezéssel fogunk élni, hogy minden rádióadónak van valamilyen rögzített hatósugara, amit egy kör alakú lefedettségi területtel jelölünk, amelyen belül egy másik állomás érzékelni és venni tudja az állomás adását. Lényeges tisztában lenni azzal, hogy a lefedettségi területek közel sem olyan szabályosak, mert a rádiójelek terjedése a környezettől függ. Falak és egyéb akadályok, amelyek csillapítják és visszaverik a jeleket, megváltoztathatják a különböző irányokba eső hatótávolságot. Az egyszerű kör alakú modell azonban a céljainknak megfelel.

Vezeték nélküli LAN használatának naiv megközelítése lenne, ha megpróbálnánk a CSMA-t alkalmazni. Figyeljünk, és csak akkor kezdjünk el adni, ha nem észlelünk egyéb forgalmat. A probléma ezzel az, hogy ez a protokoll nem igazán jó megoldás a vezeték nélküli esetre, mivel a vételkor a vevőnél, és nem az adónál levő zavarás (interferencia) számít. Hogy megértsük a problémát, tekintsük a 4.11. ábrát, amelyen négy vezeték nélküli állomás látható. Számunkra most érdektelen, hogy melyek a hozzáférési pontok, és melyek a hordozható készülékek. A rádiós hatósugár akkora, hogy *A* és *B* egymás hatósugarán belül vannak, így egymást zavarhatják. *C* szintén zavarhatja *B*-t és *D*-t, de *A*-t már nem.

Először vizsgáljuk meg, mi történik akkor, amikor *A* és *C* forgalmaz *B*-nek, ahogyan azt a 4.11.(a) ábra szemlélteti! Ha *A* ad és *C* rögtön belehallgat a csatornába, akkor nem



4.11. ábra. *Egy vezeték nélküli LAN. (a)* A *és* C *rejtett állomások, amikor a* B*-nek adnak. (b)* B *és* C *megvilágított állomások, amikor* A*-nak és* D*-nek adnak*

hallhatja *A* adását, hiszen *A* kívül van a hatósugarán. Emiatt *C* tévesen azt a következtetést vonhatja le, hogy elkezdhet *B*-nek adni. Ha *C* mégis elkezd adni, akkor interferencia lép fel *B*-nél, és tönkreteszi az *A* által küldött keretet. (Feltételezzük, hogy nem használunk CDMA-típusú sémát, ami több csatornát nyújtana, így az ütközések megváltoztatják a jelet és tönkreteszik mindkét keretet.) Egy olyan MAC-protokollra van szükségünk, amely megelőzi ezt a fajta ütközést, mert ez sávszélesség veszteséget okoz. Azt a problémát, amikor egy állomás nem képes érzékelni egy potenciális versenytársát, mivel az túl messze van tőle, a **rejtett állomás problémájának (hidden terminal problem)** nevezik.

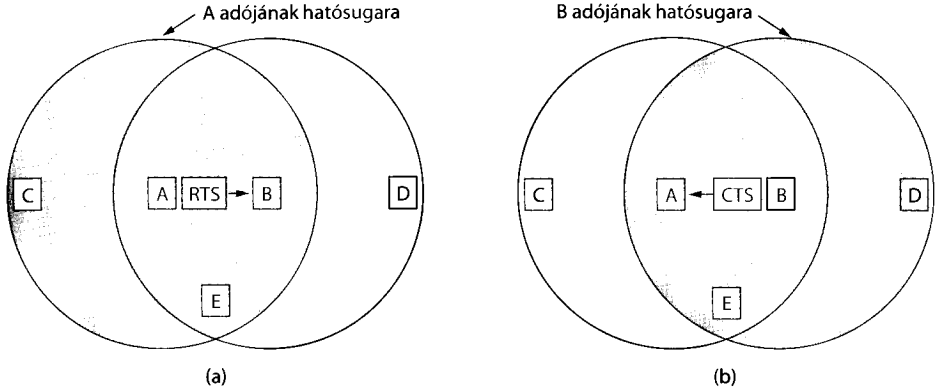
Vizsgáljunk most meg egy másik szituációt, amikor *B* ad *A*-nak, ugyanakkor *C* akar *D*-nek adni, ahogy ez a 4.11.(b) ábrán látszik. Amikor *C* megvizsgálja a csatornát, hallva a folyamatban levő átvitelt, tévesen arra a következtetésre jut, hogy nem adhat *D*-nek (szaggatott vonallal jelölve). Valójában, ez az adás csak a *B* és *C* közötti szakaszon tenné lehetetlenné a keretek vételét, de egyik címzett vevő sem ott található. Egy olyan MAC-protokollra van szükségünk, amely megelőzi az ilyen típusú késlekedést, mert ez sávszélesség-vesztéssel jár. Ezt a problémát a **megvilágított állomás problémájának (exposed terminal problem)** nevezik.

A nehézség az, hogy az adás megkezdése előtt az állomás valójában arra lenne kíváncsi, hogy a vevő környezetében van-e rádiósugárzás. A CSMA csupán azt képes megmondani a vivő érzékelésével, hogy az adó környezetében van-e jelforgalom vagy sem. Vezeték esetén minden jel eljut minden állomáshoz, ezért ez a megkülönböztetés nem létezik. Az egész rendszerben azonban minden pillanatban csak egyetlen átvitel történhet. Viszont a kis hatósugarú rádióhullámokat használó hálózatok esetében egyszerre több átvitel is lebonyolítható, ha azok célállomásai különbözők, és kívül esnek egymás hatósugarán. Erre a párhuzamosságra szükség is van, ahogy a cella mérete egyre növekszik, pontosan úgy, mint ahogy egy társasági összejövetelen sem kell megvárni, hogy a teremben mindenki elhallgasson, mielőtt valaki beszélni kezd. Több beszélgetés is folyhat egy nagy teremben mindaddig, amíg azt nem ugyanahhoz a személyhez intézik.

A vezeték nélküli LAN-ok fent említett problémáit kezelő korai és meghatározó protokoll a **MACA (Multiple Access with Collision Avoidance – többszörös hozzáférés ütközések elkerülésével)** [Karn, 1990]. A protokoll mögött rejlő alapötlet az, hogy az adónak rá kell vennie a vevőt, hogy adjon ki egy rövid keretet, amelyet a hatósugarában tartózkodó állomások érzékelnek, és nem kezdenek adni a következő (hosszabb) adatkeret időtartama alatt. A vivőjel-érzékelés helyett ezt a módszert alkalmazzák.

A MACA-protokollt a 4.12. ábra szemlélteti. Vizsgáljuk meg, hogyan küld *A* egy keretet *B*-nek. *A* azzal kezdi, hogy a 4.12.(a) ábrának megfelelően küld egy **RTS (Request To Send – adási engedély kérése)** keretet *B*-nek. Ez a rövid üzenet (mindössze 30 bájt) tartalmazza a soron következő adatkeret hosszát. Ekkor *B* a 4.12.(b) ábrán jelzett módon egy **CTS (Clear To Send – adásra kész)** üzenettel válaszol. A CTS-keret szintén tartalmazza az adatkeret hosszát (az RTS-keretből másolja ki *B*). Amint *A* megkapja a CTS-keretet, azonnal adni kezd.

Most nézzük meg, hogyan reagálnak azok az állomások, amelyek akaratlanul is veszik ezek közül valamelyik keretet. Bármelyik állomás, amelyik veszi az RTS-keretet, nyilvánvalóan közel van *A*-hoz, így legalább annyi ideig csendben kell maradnia, amíg a CTS-keret konfliktus nélkül visszaérkezik az *A* állomáshoz. Bármely állomás, amelyhez eljut a



4.12. ábra. *A MACA-protokoll. (a)* A *RTS-üzenetet küld* B*-nek. (b)* B *egy CTS-üzenettel válaszol* A*-nak*

CTS-keret, nyilvánvalóan közel van a *B*-hez, így csendben kell maradnia a CTS-t követő adatkeret átvitelének időtartama alatt, amelynek hosszát a CTS-keretből derítheti ki.

A 4.12. ábrán *C* belül van *A* hatósugarán, de kívül esik *B* hatósugarán, így foghatja az RTS-keretet *A*-tól, de nem hallhatja a *B*-től érkező CTS-keretet. Mivel nem jutott el hozzá a CTS-keret, szabadon forgalmazhat az adatkeret átvitelének időtartama alatt. Ezzel ellentétben a *D* állomás csak *B* hatósugarába esik bele, így nem foghatja az RTS-, csak a CTS-keretet. A CTS-keretet megkapva értesül arról, hogy közel van ahhoz az állomáshoz, amelyik nemsokára egy adatkeretet szeretne fogadni, így nem forgalmazhat addig, amíg az adatkeret küldése várhatóan be nem fejeződik. Az *E* állomás mindkét vezérlőüzenetet megkapja, így *D*-hez hasonlóan kénytelen csendben maradni az adatkeret továbbításának befejezéséig.

Az óvintézkedések ellenére is létrejöhet azonban ütközés. Például előfordulhat, hogy *B* és *C* egyszerre küld RTS-üzenetet *A*-nak. Ezek ütközni fognak és elvesznek. Ütközés esetén a sikertelen küldő állomás (amelyik a meghatározott időkorláton belül nem kapott válaszul egy CTS-keretet) véletlenszerű ideig várakozik, majd próbálkozik újra.

4.3. Ethernet

Ezzel befejeztük a csatornakiosztási protokollok elméleti tárgyalását, ideje tehát megnéznünk, hogyan valósulnak meg ezek az elvek a létező rendszerekben. Számos személyi, helyi és nagyvárosi hálózat megvalósítást szabványosítottak az IEEE 802 név alatt. Ezek közül néhány fennmaradt, de a többség nem, ahogy azt az 1.38. ábrán láthattuk. Vannak, akik hisznek a reinkarnációban, és úgy vélik, hogy maga Charles Darwin tért vissza az IEEE Szabványügyi Egyesületének tagjaként, hogy elbánjon az életképtelenekkel. A túlélők közül a legfontosabbak a 802.3 (Ethernet) és a 802.11 (vezeték nélküli LAN). A Bluetooth (vezeték nélküli PAN) széles körben használt, de már a 802.15-ön kívül van szabványosítva. A 802.16 (vezeték nélküli MAN) sorsáról még korai lenne jóslatokba bocsátkozni. Könyvünk 6. kiadása már bizonyára tartalmazni fogja a választ.

Valós rendszerek tanulmányozását az Ethernettel kezdjük, amely alighanem a világ legelterjedtebb számítógép-hálózat típusa. Két fajta Ethernet létezik: a **klasszikus Ethernet (classic Ethernet)**, amely a többszörös hozzáférés problémájára a fejezetben tanult technikákat alkalmazza, és a **kapcsolt Ethernet (switched Ethernet)**, amely kapcsolóknak nevezett eszközöket használ számítógépek összekötésére. Fontos megjegyezni, hogy annak ellenére, hogy nagyon különbözőek, mindkettőt Ethernetként szokták hívni. A klasszikus Ethernet jött létre előbb, melynek a sebessége 3 és 10 Mb/s között volt. Az Ethernetből alakult ki a kapcsolt Ethernet, amelynek sebessége 100, 1000 és 10 000 Mb/s. Ebben a sorrendben gyors (fast) Ethernetnek, gigabites Ethernetnek és 10 gigabites Ethernetnek hívjuk. Manapság gyakorlatilag csak kapcsolt Ethernetet használnak.

Az Ethernet megvalósulási formáit időrendi sorrendben tárgyaljuk, és kitérünk a kifejlesztésükre is. Mivel az Ethernet és a 802.3 egy apróbb különbséget leszámítva (ezt hamarosan tárgyaljuk) megegyezik, sokan ugyanazt értik az „Ethernet" és a „802.3" kifejezés alatt. Mi sem fogjuk őket megkülönböztetni. Az Ethernetről bővebben is olvashatunk Spurgeon [2000] művében.

4.3.1. A klasszikus Ethernet fizikai rétege

Az Ethernet története nagyjából egy időben kezdődik az ALOHA-val, amikor egy Bob Metcalfe nevű egyetemi hallgató megkapta a BSc oklevelét[2] az MIT-n (Massachusetts Institute of Technology), majd átment a Harvardra, hogy doktori fokozatot szerezzen.[3] A tanulmányai alatt nagy hatással volt rá Abramson munkássága. A téma annyira felkeltette az érdeklődését, hogy miután ledoktorált úgy határozott, hogy egy nyarat Abramsonnál dolgozik Hawaii-n, mielőtt elkezd dolgozni a Xerox PARC-nál (Palo Alto Research Center, Palo Altó-i kutatóközpont). Mikor a nyár végén munkába állt, azt látta, hogy az ottani kutatók éppen azt tervezik és építik meg, amit később személyi számítógépnek kereszteltek el. Ezek a gépek azonban elszigeteltek voltak. Használva Abramson munkásságából szerzett ismereteit, kollégájával, David Boggs-szal, megtervezte és megvalósította[4] az első helyi hálózatot [Metcalfe és Boggs, 1976], amelyhez egy hosszú, vastag koaxiális kábelt használt, és amely 3 Mb/s adatsebességgel működött.

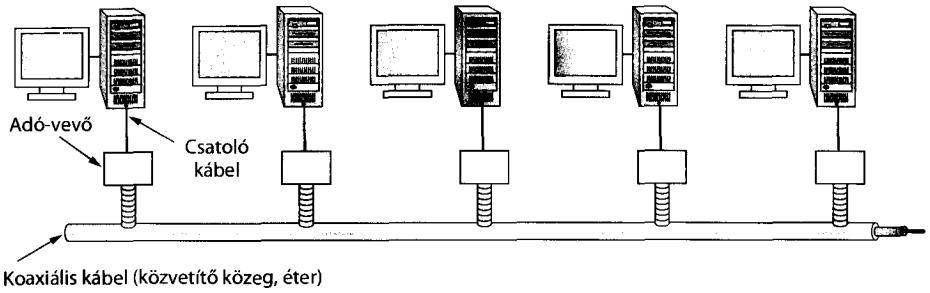
A rendszert **Ethernetnek** („éterháló") nevezték a világmindenséget betöltő anyagi közeg, az *éter* után, amelyről úgy gondolták, hogy azon keresztül terjed az elektromágneses sugárzás. (Amikor a 19. századi brit[5] fizikus, James Clerk Maxwell felfedezte, hogy az elektromágneses sugárzás hullámegyenletekkel leírható, a tudósok feltételezték, hogy a teret valamilyen éternek nevezett anyagi közegnek kell kitöltenie, amiben a sugárzás terjed. A fizikusok csak Michelson és Morley híres, 1887-es kísérlete után fedezték fel, hogy az elektromágneses sugárzás vákuumban is terjed.)

2 Robert Metcalfe 1969-ben az MIT-n valójában két BSc fokozatot szerzett: egyet Electrical Engineering, és egy másikat Industrial Management területen. *(A lektor megjegyzése)*

3 A Harvard Egyetemen 1970-ben MSc, majd 1973-ban PhD fokozatot szerzett Computer Sciences területen. *(A lektor megjegyzése)*

4 1973. november 11-én működött először egy így megtervezett hálózat. *(A lektor megjegyzése)*

5 Maxwell skót származású matamatikus-fizikus volt. *(A fordító megjegyzése)*



4.13. ábra. *A klasszikus Ethernet felépítése*

A Xerox Ethernet olyannyira sikeres volt, hogy 1978-ban a DEC, az Intel és a Xerox elkészítették egy közös, 10 Mb/s-os Ethernet-szabványt, amely a **DIX- szabvány** nevet kapta. 1983-ban, egy kis változtatással a DIX-szabványból jött létre az IEEE 802.3 szabvány. Sajnos a Xeroxnak már korábban is voltak olyan jelentékeny, fejlődést elindító felfedezései (mint például a személyi számítógép), amelyekből azután nem tudott piaci tőkét kovácsolni. Ezt a történetet írja meg Smith és Alexander [1988] *Fumbling the Future (A jövő elherdálása)* című könyvében. Amikor a Xerox csak az Ethernet szabványosítására mutatott érdeklődést, Metcalfe saját céget alapított, a 3Com-ot, hogy Ethernet-csatolókat adjon el személyi számítógépekhez. A 3Com azután ezeket milliós nagyságrendben adta el.

A klasszikus Ethernet hosszú kábelként kígyózott végig az épületen, amelyhez minden számítógép csatlakozott. Ez az architektúra látható a 4.13. ábrán. Az első változat, a közkedvelt **vastag Ethernet (thick Ethernet)**, amely megjelenésében egy sárga kerti locsolótömlőre emlékeztetett, amelyen a csapok lehetséges csatlakoztatási pontjait 2,5 méterenként megjelölték. (A 802.3 szabvány nem *írja elő,* hogy a kábelnek sárgának kell lennie, de *javasolja.*) Ezt követte a **vékony Ethernet (thin Ethernet)**, amelyet könnyebb volt hajlítani és könnyebb volt hozzá gépeket csatlakoztatni az ipari szabvány BNC-csatlakozókkal. A vékony Ethernet jóval olcsóbb volt és könnyebb volt telepíteni, de csak 185 méteres szegmensekből állhatott (a korábbi 500 m helyett), valamint csak 30 eszközt lehetett hozzá csatlakoztatni (a korábbi 100 helyett).

Minden Ethernet-verzió rendelkezik egy legnagyobb megengedett szegmensenkénti kábelhosszal (erősítetlen hossz), amely távolságra a jel terjed. Hogy nagyobb hálózatokat lehessen kialakítani, több kábelszegmenst **ismétlőkkel (repeater)** kell összekapcsolni. Az ismétlők fizikai rétegben működő eszközök, amelyek fogadják, erősítik (azaz regenerálják, újra előállítják) és mindkét irányba kiküldik a jelet. Szoftverszempontból a kábelszegmensek ismétlőkkel összekapcsolt sorozata nem különbözik egy egyszerű kábeltől (leszámítva az ismétlők által behozott kismértékű késleltetést).

Ezeken a kábeleken keresztül az információt a Manchester-kódolás használatával továbbítják, amiről már tanultunk a 2.5. szakaszban. Egy Ethernet-hálózat több kábelszegmenst és több ismétlőt tartalmazhat, de két adó-vevő nem lehet messzebb egymástól 2,5 km-nél, valamint bármelyik két adó-vevő között legfeljebb csak négy ismétlő lehet. Erre a korlátozásra azért van szükség, hogy a MAC-protokoll, amellyel a következő szakaszban foglalkozunk, megfelelően működjön.

4.3.2. A klasszikus Ethernet MAC-alréteg protokollja

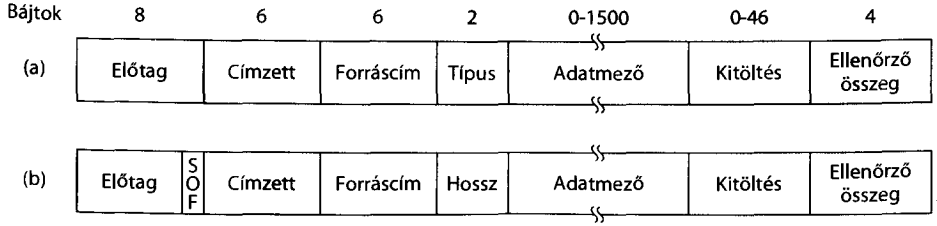
A használt keretformátumot a 4.14. ábra mutatja be. Először egy 8 bájtos *Előtag (Preamble)* jön, mely az 10101010 mintát tartalmazza (kivéve az utolsó bájtot, amelyben az utolsó 2 bit 11). Ezt az utolsó bájtot, amit *Keret kezdete (Start of Frame)* határolónak neveznek a 802.3 szabványban. Ennek a mintának a Manchester-kódolása egy 10 MHz-es, 6,4 μs időtartamú négyszögjelet állít elő, ami lehetővé teszi, hogy a vevő az adóhoz igazítsa az óráját. Az utolsó két 1-es bit jelzi a vevőnek, hogy a keret többi része kezdődik.

Ezután két címmező jön, egyik a célcím, másik a forráscím. Mindkettő 6 bájt hosszú. A célcím első bitje közönséges címek esetén 0, csoportcímek esetén viszont 1 értékű. A csoportcímek több állomás egyetlen címmel való megcímzését teszik lehetővé. Amikor egy keretet csoportcímmel küldünk el, akkor azt a csoport minden tagja veszi. Az állomások egy meghatározott csoportjának való keretküldést **többesküldésnek (multicast)** nevezik. A különleges, csupa 1-esekből álló cím az **adatszórás (broadcast)** esetén használatos. A célcímben csupa 1-est tartalmazó kereteket az összes állomás veszi. A többesküldés válogatósabb, de rendelkezik csoportmenedzsmenttel, ami meghatározza, hogy mely állomások tartoznak a csoportba. Az adatszórás azonban nem különbözteti meg az állomásokat, ezért nem is igényel semmiféle csoportkezelést.

Az állomások forráscímének érdekessége, hogy globálisan egyediek, melyeket az IEEE jelöl ki azért, hogy a világon ne fordulhasson elő két azonos cím. Az alapgondolat az, hogy 48 bitet használva már a világ bármely két állomása megcímezheti egymást. Ennek eléréséhez a címmezők első 3 bájtját az **OUI (Organizationally Unique Identifier – szervezetenkénti egyedi azonosító)** teszi ki. Ennek a mezőnek az értékeit az IEEE határozza meg a gyártó megjelölésére. A gyártók 2^{24} címből álló blokkokat kapnak. A gyártó az utolsó 3 bájt kiválasztása után egy hálózati csatolóba beleprogramozza a teljes címet, még annak eladása előtt.

A következő mező a *Típus (Type)* vagy a *Hossz (Length)* mező, attól függően, hogy a keret Ethernet vagy IEEE 802.3. Az Ethernet a *Típus* mezőt használja, amely azt határozza meg, hogy a vevőnek mit kell tennie a kerettel. Több hálózati rétegbeli protokoll is működhet egy gépen egyszerre, az operációs rendszernek pedig tudnia kell, hogy melyiknek kell átadni a keretet. A *Típus* mező tehát azt adja meg, hogy melyik folyamatnak kell átadni a keretet. Például, a 0x0800 típus azt jelenti, hogy a hordozott adat egy IPv4-csomag.

Az IEEE 802.3 bölcsen úgy döntött, hogy ez a mező a keret hosszát fogja szállítani, mert eddig az Ethernet-keretek hosszát úgy állapították meg, hogy belenéztek az adat-



4.14. ábra. *Keretformátumok. (a) Ethernet (DIX). (b) IEEE 802.3*

mezőbe, ami a rétegezés megsértése, ha egyáltalán volt rétegezés. Ebből természetesen az következett, hogy a vevő sehogy nem tudta kitalálni, hogy mit kezdjen a bejövő kerettel. Ezt a problémát egy újabb, az **LLC (Logical Link Control – logikai kapcsolatvezérlés)** protokoll fejlécének az adatmezőbe történő beszúrásával oldották meg. Ez a fejléc felhasznál 8 bájtot, hogy a 2 bájtos protokoll típus információt szállítsa.

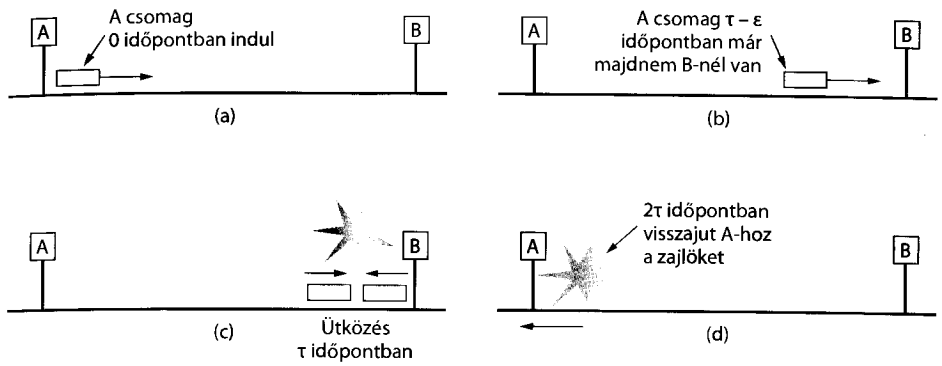
Mire a 802.3-at közzé tették, sajnos már olyan sok DIX Ethernet-hardver és -szoftver volt használatban, hogy a gyártók és felhasználók nemigen lelkesedtek a *Típus*-ról a *Hossz* mezőre való átállásért. 1997-ben aztán az IEEE is bedobta a törölközőt, és mindkét megoldásra áldását adta. Szerencsére minden 1997 előtt használt *Típus* érték nagyobb volt 1500-nál, ezért ezt állapították meg legnagyobb adathossznak. Most az a szabály, hogy a 0x0600-nál (1536) kisebb vagy egyenlő számokat *Hossz*-ként, az 0x0600-nál (1536) nagyobbakat pedig *Típus*-ként kell értelmezni. Így az IEEE is elmondhatja magáról, hogy mindenki az ő szabványát használja, a többiek pedig folytathatják azt, amit eddig csináltak (nem kell bajlódniuk az LLC-vel) anélkül, hogy bűntudatot kellene érezniük.

Ezután jön a legfeljebb 1500 bájt hosszúságú adatmező. Ezt a határt többé-kevésbé önkényesen állapították meg az Ethernet-szabvány mérföldkővé válásakor, leginkább arra alapozva, hogy az adó-vevőnek elegendő memóriával kell rendelkeznie egy teljes keret tárolásához, márpedig a memória 1978-ban drága volt. Egy magasabb felső határ több memóriát, és emiatt drágább adó-vevőt igényelt volna.

Nemcsak a maximális, hanem a minimális kerethossz is rögzítve van. Egy 0 hosszúságú adatmezőnek is lehet ugyan értelme, mégis problémákat okozhat. Amikor egy adó-vevő ütközést érzékel, csonkolja az aktuális keretet, ami azt jelenti, hogy kóbor bitek és keretdarabkák bármikor jelen lehetnek a kábelen. Annak érdekében, hogy az érvényes kereteket könnyebben meg lehessen különbözteti a szeméttől, az érvényes Ethernet-kereteknek a célcímtől az ellenőrző összegig (beleértve e két mezőt is) legalább 64 bájt hosszúnak kell lenniük. Ha tehát egy keret adatrésze 46 bájtnál rövidebb, akkor a *Kitöltés (Pad)* mezőt kell használni a keret minimális méretének eléréséhez.

A minimális kerethosszúság előírását más (sokkal fontosabb) indok is szükségessé teszi. Rövid keretek engedélyezése esetén előfordulhatna, hogy egy állomás még azelőtt befejezné a keretének küldését, mielőtt annak első bitje elérné a kábel legtávolabbi végét, ahol az még egy másik kerettel ütközhet. A problémát a 4.15. ábra illusztrálja. A 0 időpillanatban a hálózat egyik végén elhelyezkedő *A* állomás elküld egy keretet. Jelöljük azt az időtartamot, amíg ez a csomag elér a hálózat másik végéig, τ-val. Éppen azelőtt, hogy a keret elérte volna a vezeték másik végét (azaz $\tau - \varepsilon$ pillanatban), a legtávolabbi állomás, *B* szintén adni kezd. Amikor *B* észleli, hogy az általa vett jel erőssége nagyobb, mint amit maga sugárzott, rájön, hogy ütközés történt. Abbahagyja az adását és egy 48 bit hosszú zajlöketet (noise burst) állít elő, hogy a többi állomást is figyelmeztesse az ütközésre. Más szóval, beletömköd egy bitsorozatot (jam) az éterbe, hogy biztos legyen abban, hogy az eredeti adó észreveszi az ütközést. Az eredeti küldő fél az adás megkezdése után csak körülbelül 2τ idő elteltével érzékeli a zajlöketet, amelynek hatására szintén leáll a forgalmazással. Ezután véletlenszerű ideig vár, majd újból próbálkozik.

Ha egy állomás nagyon rövid keretet próbál elküldeni, elképzelhető, hogy bekövetkezik egy ütközés, de az átvitel befejeződik, mielőtt a zajlöket a 2τ idő elteltével az adóhoz visszaérkezik. Az adó ekkor azt a téves következtetést vonja le, hogy a keretet sikeresen



4.15. ábra. *Az ütközés érzékelése 2τ időt is igénybe vehet*

küldte el. Az ilyen helyzetek elkerülése érdekében minden keretnek olyan hosszúnak kell lennie, hogy elküldése legalább 2τ időt igényeljen. Ily módon az átvitel még biztosan tartani fog, amikor a zajlöket visszaérkezik az adóhoz. Egy (a 802.3 specifikációja alapján) maximális, azaz 2500 méter hosszú, négy ismétlőt tartalmazó, 10 Mb/s-os LAN-on a körülfordulási idő értékét (beleértve a négy ismétlőn való áthaladás idejét is) a legrosszabb esetet feltételezve mintegy 50 μs-ban rögzítették. Következésképpen a legrövidebb engedélyezett keret átvitelének is legalább ennyi ideig kell tartania. 10 Mb/s átviteli sebesség esetén egy bit 100 ns hosszú, vagyis 500 bites az a legrövidebb keret, mely garantáltan működni fog. A biztonság kedvéért ezt a számot felkerekítették 512 bitre, azaz 64 bájtra.

Az utolsó mező az *Ellenőrző összeg (Checksum)*. Ez egy 32 bites CRC, amiről a 3.2. szakaszban volt szó. Valójában ugyanazt a generátorpolinomot használja, mint amit ott megadtunk, és ami felbukkan a PPP-nél, az ADSL-nél és még más összeköttetéseknél is. Ez a CRC egy hibajelző kód, amit a keret helyes vételének megállapítására használnak. Mivel csak hibajelzésre használható, a keretet eldobják, ha hibát érzékeltek.

CSMA/CD kettes exponenciális visszalépéssel

A klasszikus Ethernet 1-perzisztens CSMA/CD algoritmust használ, amit már tanulmányoztunk a 4.2. szakaszban. Ez a megnevezés azt takarja, hogy ha az állomásoknak van küldendő keretük, akkor közegérzékelést végeznek és elküldik a keretet, amint a közeget használatlannak érzik. A küldés alatt figyelik a csatornát. Ha ütközést érzékelnek, egy rövid zavarjellel megszakítják a küldést, és egy véletlen hosszú időtartam után újraküldik.

Most nézzük meg, hogyan számítódik a véletlen hosszú időtartam, amikor ütközés történik, mivel ez egy új metódus. Még mindig a 4.5. ábrán látható modellt használjuk. Egy ütközés után az időt diszkrét szeletekre osztva képzelhetjük el, ahol az időszeletek olyan hosszúak, mint amennyi idő legrosszabb esetben ahhoz kell, hogy egy jel visszaérhessen az éteren keresztül (vagyis 2τ). Az Ethernet-szabvány által megengedett legnagyobb hosszhoz igazodva, az időszeleteknek a hosszát 512 bit-időre, azaz 51,2 μs-ra választották meg.

Az első ütközés után minden állomás véletlenszerűen vagy 0, vagy 1 időszeletnyit várakozik, mielőtt újra próbálkozna. Ha egy ütközés után a két állomás ugyanazt a véletlen számot sorsolja ki, akkor ismét ütközni fognak. A második ütközés után véletlenszerűen 0-t, 1-et, 2-t vagy 3-at sorsolnak, és ennyi időszeletnyit várakoznak. Ha harmadszor is ütköznek (ennek valószínűsége 0,25), akkor a 0 és a 2^3-1 közé eső intervallumból választják ki, hogy mennyi időszeletnyit várakoznak.

Általánosan igaz, hogy az i-edik ütközés után a véletlen szám a 0 és 2^i-1 közötti intervallumból kerül kiválasztásra, és az állomások ennek megfelelő számú időszeletet hagynak ki. Mindazonáltal a 10. ütközés után a tartomány már nem nő tovább, hanem az 1023 marad a felső korlátja. A 16. ütközés után a vezérlő bedobja a törülközőt, és hibajelzést küld a számítógépnek. A további hibajavítás már a felsőbb rétegek feladata.

Ezt az algoritmust **kettes exponenciális visszalépésnek (binary exponential backoff)** nevezik. Azért erre az algoritmusra esett a választás, mert dinamikusan képes az adni kívánó állomások számához igazodni. Ha a véletlenszám-generálás felső határa minden ütközés esetén 1023 lenne, akkor két állomás újbóli ütközésének valószínűsége valóban elhanyagolható volna, de a várakozási idő várható értéke több száz rés körül alakulna, amely megengedhetetlenül nagy késleltetéseket okozna. Másfelől viszont, ha az állomások örökösen csak a 0 és az 1 közül választanának csak, akkor 100 egyszerre adni kívánó állomás keretei addig ütköznének, amíg végre 99 állomás 1-et, és a maradék egy 0-t választana. Ez akár évekig is eltarthatna. Azáltal, hogy a véletlenszám-generálás intervalluma az egymást követő ütközések hatására exponenciálisan nő, az algoritmus biztosítja azt, hogy kevés ütköző állomás esetén viszonylag kis késleltetés következzen be, ugyanakkor nagyszámú állomás esetén az ütközés még belátható időn belül feloldódjon. A visszalépés 1023-ra való visszavágásának a célja, hogy ez a korlát ne nőjön túl nagyra.

Ha nincs ütközés, a küldő feltételezi, hogy a keret sikeresen megérkezett. Azaz, sem a CSMA/CD, sem az Ethernet nem küld nyugtákat. Ez a módszer megfelelő rézvezetékes és fényvezetőszálas csatornák esetén, amelyek kis bithibaarányt biztosítanak. Ha mégis hiba történik, akkor azt a CRC-nek kell észlelnie, és a felsőbb rétegeknek kell helyreállítania. Amint látni fogjuk, a vezeték nélküli csatornák, amelyeknek rosszabb a bithibaaránya, nyugtázást használnak.

4.3.3. Az Ethernet teljesítőképessége

Vizsgáljuk most meg a klasszikus Ethernet teljesítőképességét nagy és állandó terhelés mellett! Tegyük fel, hogy k állomás folyamatosan adásra kész állapotban van. A bináris exponenciális visszalépés algoritmusának teljes vizsgálata nagyon bonyolult, ezért Metcalfe és Boggs [1976] példáját követve, minden résben állandó újraküldési valószínűséget feltételezünk. Ha minden egyes állomás p valószínűséggel ad egy versengés során, akkor annak valószínűsége (A), hogy valamelyik állomás meg is tudja szerezni a csatornát:

$$A = kp(1-p)^{k-1} \tag{4.5}$$

A akkor maximális, ha $p = 1/k$ és ha $k \to \infty$, akkor $A \to 1/e$-hez. Annak valószínűsége, hogy a versengési intervallum pontosan j időszeletet tartalmaz, $A(1-A)^{j-1}$, így tehát a versengésenkénti rések számának középértéke:

$$\sum_{j=0}^{\infty} jA(1-A)^{j-1} = \frac{1}{A}$$

Mivel minden rés időtartama 2τ, ezért a versengési intervallum átlagos hossza $w = 2\tau/A$. Optimális p-t feltételezve a versengési rések számának középértéke soha nem nagyobb mint e, így w legfeljebb $2\tau\varepsilon \approx 5{,}4\tau$ lehet.

Ha egy átlagos keret elküldéséhez P másodpercre van szükség, akkor sok küldeni kívánó állomás esetén:

$$\text{Csatornahatékonyság} = \frac{P}{P + 2\tau/A} \tag{4.6}$$

Itt látható, hogy a két állomás közti maximális kábelhosszúság hol játszik szerepet a teljesítőképesség alakulásában. Minél hosszabb a kábel, annál hosszabb a versengési intervallum hosszúsága is, ezért határoz meg az Ethernet-szabvány maximális kábelhosszt.

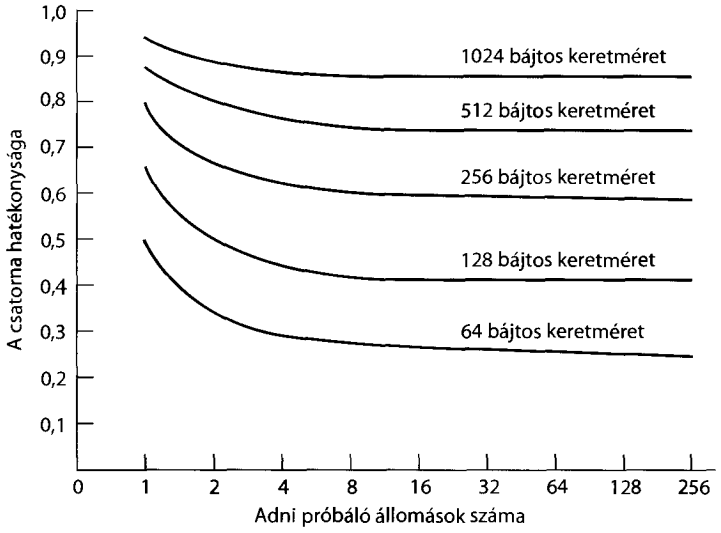
Tanulságos a (4.6) egyenlőséget az F kerethossz, a B hálózati sávszélesség, az L kábelhosszúság és a c jelterjedési sebesség segítségével az optimális e keretenkénti versengési rés esetére átalakítani. $P = F/B$ teljesülése esetén a (4.6) egyenlet így alakul:

$$\text{Csatornahatékonyság} = \frac{1}{1 + 2BLe/cF} \tag{4.7}$$

Amikor a nevező második tényezője nagy, a hálózat hatékonysága kicsi. Konkrétan, ha a hálózati sávszélesség és a távolság nő (BL szorzat), akkor ez csökkenti az egy adott keretméretre számított hatékonyságot. Sajnos azonban a legtöbb hálózatihardver-kutatás éppen ennek a szorzatnak a növelésére irányul. Nagy távolságokon nagy sávszélességet akarnak elérni (például üvegszálas MAN-ok), ez alapján a klasszikus Ethernet nem a legalkalmasabb az ilyen alkalmazások számára. Az Ethernet megvalósítására más módszereket látunk majd a következő szakaszban.

A 4.16. ábrán a (4.7) egyenlőség alapján $2\tau = 51{,}2$ μs és 10 Mb/s-os adatátviteli sebesség mellett az adni kész állomások függvényében rajzoltuk fel a csatornahatékonyság görbéjét. 64 bájtos résidő mellett nem meglepő, hogy a 64 bájtos keretek nem hatékonyak. Másfelől, 1024 bájtos kereteket és versengési intervallumonként e darab (amely csak aszimptotikusan közelíthető) 64 bájtos rést feltételezve, a hatékonyság 0,85, míg a versengési periódus 174 bájt hosszú lesz. Ez az érték sokkal jobb, mint az időszeletelt ALOHA 37%-os hatékonysága.

Valószínűleg megéri megemlíteni, hogy az Ethernettel (és más hálózatokkal) kapcsolatban rengeteg elméleti teljesítményelemzés létezik. A legtöbb eredmény (jókora) fenntartással kezelendő, két ok miatt. Először is, majdhogynem az összes elméleti munka Poisson-forgalmat feltételez. Ahogy azonban a kutatók elkezdték a valódi forgalmi adatokat vizsgálni, kiderült, hogy a hálózatok forgalma ritkán Poisson-forgalom. Ehe-



4.16. ábra. *Az Ethernet hatékonysága 10 Mb/s-os sebesség és 512 bites résidő esetén*

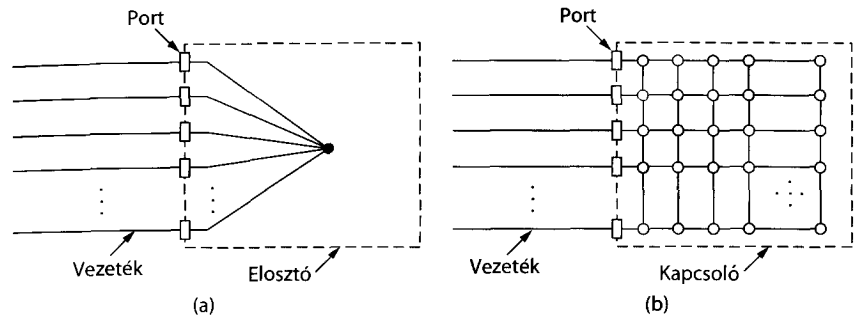
lyett, önhasonló és löketes a különböző időintervallumokban [Paxson és Floyd, 1994; Willinger és mások, 1995]. Ez azt jelenti, hogy a hosszú időintervallumokra számított átlagértékek nem egyenlítik (nem simítják) ki a forgalmat. Továbbá, megkérdőjelezhető modellt használva sok vizsgálat az „érdekes", irreálisan magas terhelés melletti teljesítőképesség vizsgálatára fókuszál. Boggs és társai [1988] kísérletileg bebizonyították, hogy az Ethernet a valóságban jól működik, még viszonylag magas terhelés mellett is.

4.3.4. Kapcsolt Ethernet

Az Ethernet szinte azonnal – a klasszikus Ethernet egyetlen hosszú kábelt tartalmazó architektúrájának megjelenését követően – továbbfejlődött. A kábel töréseivel vagy az érintkezési hibák megtalálásával összefüggő probléma egy új vezetékezési mintát kényszerített ki, amelyben minden állomás saját kábellel csatlakozik egy központi **elosztóhoz** (**hub**). Az elosztó egyszerűen villamos kapcsolatot létesít a hozzá csatlakozó vezetékek között, mintha összeforrasztották volna azokat. Ez a kiépítés látható a 4.17.(a) ábrán.

A vezetékek általában a telefontársaságok sodrott érpárjai voltak, mivel a legtöbb irodaépület már be volt kábelezve ezzel, és rendszerint rengeteg tartalék vonal állt rendelkezésre. Ez az újrahasznosítás nagy nyereség volt, de a megengedett legnagyobb kábelhosszt a legközelebbi elosztótól mérve 100 méterre csökkentette (illetve 200 méterre, ha a jó minőségű 5-ös kategóriájú sodrott érpárt használták). Egy állomás hozzáadása vagy eltávolítása ebben a kiépítésben egyszerűbb lett, és a kábeltörés is könnyebben érzékelhetővé vált. A meglévő kábelezés újrahasznosításával, a könnyebb karbantarthatóságával a sodrott érpáros, elosztós kiépítés gyorsan az Ethernet domináns formájává vált.

Az elosztók azonban nem növelték meg az átviteli kapacitást, mert logikailag a klasszikus Ethernet egyetlen hosszú kábeles kiépítésével egyezett meg. Ahogy újabb és újabb



4.17. ábra. *(a) Elosztó. (b) Kapcsoló*

állomások csatlakoznak a klasszikus Ethernethez, minden állomás egyre csökkenő részt kap a rögzített kapacitásból. Esetenként a LAN telítődhet. Ennek elkerülésre az egyik megoldás az, hogy növeljük a sebességet, mondjuk 10 Mb/s-ról 100 Mb/s-ra, 1 Gb/s-ra vagy még tovább. Ahogy azonban a multimédiás tartalmak és a szerverek teljesítménye növekszik, még az 1 Gb/s-os Ethernet is telítődhet.

Szerencsére a megnövekedett forgalom kezelésére van egy másik megoldás is: a kapcsolt Ethernet. A rendszer lelke a 4.17.(b) ábrán látható **kapcsoló** (**switch**). Ez egy nagy sebességű hátlapot (backplane) tartalmaz, ami összekapcsolja az összes portot. Kívülről egy kapcsoló éppen úgy néz ki, mint egy elosztó. Mindkettő tipikusan 4–48 portot tartalmazó doboz, melynek portjai egy-egy RJ-45 foglalatot tartalmaznak a sodrott érpár részére. Mindegyik kábel egy-egy számítógéphez csatlakoztatja a kapcsolót vagy az elosztót, ahogy a 4.18. ábrán látható. Egy kapcsolónak megvannak ugyanazok az előnyös tulajdonságai, mint az elosztónak. Egyszerű az állomások hozzáadása és eltávolítása: a vezetéket egyszerűen be kell dugni (vagy ki kell húzni) a csatlakozó aljzatba. Könnyű a hibák helyének meghatározása is, mert a hibás kábel vagy port általában csak egyetlen állomást érint. Továbbra is van egy közös komponens, amely meghibásodhat, ez maga a kapcsoló. Ha azonban az összes állomás kiesik, akkor a rendszergazdák tudják, hogyan hárítsák el a hibát: kicserélik magát a kapcsolót.

A kapcsolón belül azonban teljesen más történik, mint az elosztón belül. A kapcsolók csak azokra a portokra teszik ki a kereteket, amelyekre azokat a kereteket szánták. Amikor egy kapcsoló port egy Ethernet-keretet kap egy állomástól, megvizsgálja az Ethernet-címeket, hogy meg tudja állapítani, hogy melyik porton kell kimennie. Ez a lépés megköveteli a kapcsolóktól, hogy képesek legyenek meghatározni, hogy melyik porthoz melyik cím van hozzárendelve. Ezt a folyamatot a 4.8. szakaszban mutatjuk be annak az általános esetnek a tárgyalásánál, amikor kapcsolók más kapcsolókkal vannak összeköttetésben. Egyelőre tételezzük fel, hogy a kapcsoló ismeri a keret célportját. Ezután a kapcsoló továbbítja a keretet a nagy sebességű hátlapján keresztül a célporthoz. A hátlap tipikusan több Gb/s sebességű, és egyedi protokollt használ, amelyet nem szükséges szabványosítani, mert teljes egészében el van rejtve a kapcsoló belsejében. A célport ezután továbbítja a keretet a vezetéken, amely így eléri a kívánt állomást. Egyetlen másik port sem tud a keret létezéséről.

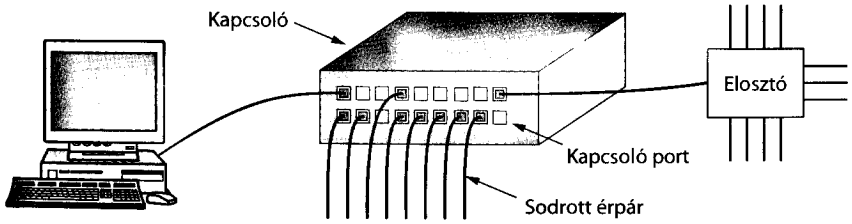
Mi történik, ha ugyanabban a pillanatban több mint egy állomás akar keretet kül-

deni? Ismét megjegyezzük, hogy a kapcsolók különböznek az elosztóktól. Az elosztók esetében az összes állomás egy **ütközési tartományt** (**collision domain**) képez. Ezeknek feltétlen a CSMA/CD-algoritmust kell használniuk adásaik ütemezésére. Kapcsoló esetén, minden porthoz saját ütközési tartomány tartozik. Abban az általános esetben, amikor a kábel duplex, mind az állomás, mind a port egyszerre tud keretet küldeni a kábelre anélkül, hogy aggódni kellene más portok és állomások miatt. Így az ütközések lehetetlenné váltak és a CSMA/CD alkalmazása nem szükséges. Ha azonban a kábel fél-duplex, az állomásnak és portnak versengenie kell az adás jogáért a CSMA/CD-vel a szokásos módon.

A kapcsoló két módon is jobb teljesítményt nyújt az elosztóknál. Először is, mivel nincsenek ütközések, az átviteli kapacitás hatékonyabban kihasználható. Másodszor, és ez a fontosabb érv, kapcsoló használatával több állomás tud egyszerre keretet továbbítani. Ezek a keretek elérik a kapcsoló portjait és áthaladnak a kapcsoló hátlapján, hogy a megfelelő kimeneti portokon keresztül hagyják el a kapcsolót. Mivel azonban lehetséges, hogy két keretet ugyanarra a kimenő portra küldtek ugyanabban a pillanatban, ezért a kapcsolónak pufferrel kell rendelkeznie, hogy a bejövő keretet ideiglenesen tárolni tudja, amíg az a kimenő portra küldhető lesz. Összegezve, ezek a fejlesztések olyan nagy teljesítménybeli előnyt jelentenek, amire az elosztó nem képes. A teljes rendszer átbocsátóképessége gyakran egy nagyságrenddel nagyobb lesz, a portok számától és a forgalom mintájától függően.

A biztonság szempontjából is előnyös, hogy megváltozott, hogy mely portokon keresztül mennek ki a keretek. A legtöbb LAN-illesztő rendelkezik azzal a képességgel, hogy ún. **válogatás nélküli üzemmódban** (**promiscuous mode**) üzemeljen, amikor is *minden* keretet továbbít a számítógépnek, és nem csak azokat, amelyeket annak az állomásnak címeztek. Egy elosztóval minden hozzákapcsolt számítógép láthatja az összes többi számítógép között menő forgalmat. A kémek és a minden lében kanál emberek nagyon szeretik ezt a funkciót. Ez a korlátozás nagyobb elszigeteltséget eredményez, ami által a forgalom nem tud könnyen kijutni és rossz kezekbe kerülni. Mindenesetre jobb, ha titkosítjuk az adatforgalmat, amennyiben a biztonságra ténylegesen szükség van.

Mivel a kapcsoló csak annyit vár el, hogy szabványos Ethernet-keretek érkezzenek a bemeneti portjaira, néhány portját koncentrátorként lehet használni. A 4.18. ábrán a jobb felső portra nem egy különálló számítógép, hanem egy 12 portos elosztó csatlakozik. Az elosztóra érkező keretek a szokott módon versengenek a közegért, beleértve az ütközéseket és a kettes visszalépéses algoritmus használatát is. Azok a keretek, amelyek sikeresen átjutottak az elosztón, folytatják útjukat a kapcsoló felé, ahol ugyanolyan bá-



4.18. ábra. *Egy Ethernet-kapcsoló*

násmódban részesülnek, mint bármilyen más beérkező keret. A kapcsoló nem tudja, hogy meg kellett harcolniuk az odajutásért. Amikor már a kapcsolóban vannak, továbbításra kerülnek a nagy sebességű hátlapon keresztül a megfelelő kimeneti vonal felé. Az is elképzelhető, hogy a megfelelő cél az elosztóhoz kapcsolódik, ebben az esetben a keret már meg is érkezett, és a kapcsoló eldobja. Az elosztók egyszerűbbek és olcsóbbak ugyan a kapcsolóknál, de a kapcsolók árának zuhanása miatt mára már veszélyeztetett fajnak számít. A modern hálózatok főként kapcsolt Ethernetet használnak, azonban a korábbi elosztók még mindig megtalálhatók.

4.3.5. Gyors Ethernet

Ahogy a kapcsolók egyre népszerűbbé váltak, a 10 Mb/s sebességű Ethernet egyre nagyobb nyomás alá került. A 10 Mb/s kezdetben valóságos mennyországnak tűnt a felhasználók számára éppúgy, ahogy a kábelmodemek is csodának számítottak a telefonmodemek felhasználóinak szemében. Az újdonság varázsa azonban hamar elszállt. Úgy tűnik, mintha Parkinson törvényének („A munka mindig kitölti az elvégzéséhez rendelkezésre álló időt") egyfajta következményeként igaz lenne az is, hogy az adat mindig kitölti az átviteléhez rendelkezésre álló sávszélességet.

Sok alkalmazás 10 Mb/s-nál nagyobb sávszélességet igényelt, emiatt sokszor jó néhány 10 Mb/s-os LAN-t kötöttek össze ismétlők, elosztók és kapcsolók útvesztőjével. A rendszergazdák pedig sokszor már úgy érezték, hogy az egész rendszert csak rágógumi és csirkeháló tartja egyben. Még Ethernet-kapcsolók használatakor is korlátozta a számítógép rendelkezésére álló sávszélességet az a kábel, ami a gépet összekötötte a kapcsoló portjával.

Így történt, hogy az IEEE 1992-ben újból összehívta a 802.3 bizottságot, hogy készítsen szabványt egy gyorsabb LAN-ra. Az egyik javaslat az volt, hogy tartsák meg a 802.3 szabványt olyannak amilyen, csak tegyék gyorsabbá. A másik indítvány szerint viszont az egészet teljesen át kellett volna alakítani és olyan újabb szolgáltatásokkal felruházni, mint például a valós idejű forgalom és a digitális beszédátvitel, és csak a régi nevet tartsák meg (üzleti okokból). Némi huzavona után a bizottság az első javaslat mellett döntött. A stratégia szerint a munkát be fogják fejezni, még mielőtt a technológia megváltozna, és így elkerülik a teljesen új tervezéssel járó előreláthatatlan problémákat. Az új tervezetnek kompatibilisnek kellene lennie a létező Ethernet LAN-okkal. A vesztes indítvány mögött álló emberek erre azt tették, amit a számítógépipar bármely más önérzetes résztvevője tett volna a helyükben: félrevonultak, megalapították saját bizottságukat, és elkészítették saját LAN-szabványukat, a 802.12-t. Az eredmény szánalmas bukás volt.

A munkával hamar végeztek is (legalábbis egy szabványosítási bizottsághoz képest), és az eredményt, a **802.3u** szabványt 1995 júniusában elfogadta az IEEE. A 802.3u technikailag nem új szabvány, hanem a meglevő 802.3 kiegészítése (a hátrafelé kompatibilitás hangsúlyozása érdekében). Ezt a stratégiát gyakran használják. Szinte mindenki – így mi is – **gyors Ethernet** (**fast Ethernet**) néven hivatkozik rá a 802.3u helyett.

A gyors Ethernet alapötlete egyszerű: tartsunk meg minden régi keretformátumot, interfészt és eljárási szabályt, a bitidőt, de csökkentsük 100 ns-ról 10 ns-ra. Műszaki szempontból lehetséges lett volna a 10 Mb/s-os klasszikus Ethernet megőrzése, és még

Név	Kábel	Max. szegmens	Előnyök
100Base-T4	Sodrott érpár	100 m	3-as kategóriájú UTP-t használ
100Base-TX	Sodrott érpár	100 m	Duplex 100 Mb/s (5. kat. UTP)
100Base-FX	Fényvezető szál	2000 m	Duplex 100 Mb/s, nagy távolság

4.19. ábra. *A gyors Ethernet eredeti kábelezése*

ekkor is időben lehetett volna észlelni az ütközéseket, ha a maximális kábelhosszt tizedére csökkentették volna. A sodort érpáros kábelezés előnyei azonban annyira meggyőzők voltak, hogy a gyors Ethernetet teljes egészében erre a megoldásra alapozták. Ezért minden gyors Ethernet-rendszer elosztókat és kapcsolókat használ; a vámpírcsatlakozós csatolókábelek vagy a BNC-csatlakozók nem megengedettek.

Néhány döntést azonban még így is meg kellett hozni. A legfontosabb ezek közül az volt, hogy milyen kábelfajtát támogassanak. A verseny egyik résztvevője a 3-as kategóriájú sodrott érpár volt. A mellette szóló érv az volt, hogy a nyugati világban gyakorlatilag minden irodát legalább négy, 3-as kategóriájú (vagy jobb) sodrott érpár kötött össze a 100 méteren belül lévő telefonos kábelrendezővel. Esetenként két ilyen kábel is létezett. Ily módon a 3-as kategóriájú sodrott érpárok segítségével az asztali számítógépeket anélkül lehetne gyors Ethernet-hálózatba csatlakoztatni, hogy az egész épületet újra kellene kábelezni, ami sok szervezet számára óriási előnyt jelent.

A 3-as kategóriájú sodrott érpár fő hátránya az, hogy képtelen a 100 Mb/s sebesség nyújtására 100 méter hosszú kábelen, márpedig ez a maximális távolság a számítógépek és az elosztó között a 10 Mb/s-os elosztók esetén. Ezzel szemben az 5-ös kategóriájú sodrott érpár számára a 100 méter nem jelent akadályt, a fényvezető szál pedig még ennél is távolabb mehet. Végül azt a kompromisszumot választották, hogy megengedik mindhárom lehetőséget, ahogy ezt a 4.19. ábra is szemlélteti, de a 3-as kategóriájú megoldást fel kellett javítani úgy, hogy elérje a szükséges szállítási kapacitást.

A 3-as kategóriájú UTP-séma, melyet **100Base-T4**-nek neveznek, 25 MHz-es[6] jelzési sebességet használt, ami csak 25%-kal több az Ethernet-szabvány 20 MHz-énél[7] (ne felejtsük el, hogy a 2.5. szakaszban tárgyalt Manchester-kódolás két órajel-periódust igényel minden egyes bithez a másodpercenként elküldött 10 millió bitből). A szükséges adatsebesség eléréséhez a 100Base-T4 négy sodrott érpárt igényel. A négy sodrott érpár közül egy mindig az elosztó felé, egy mindig az elosztó felől, a maradék kettő pedig átkapcsolható módon, az aktuális átvitel irányába szállítja az adatokat. A 100 Mb/s-os sávszélesség mindkét irányban való elérésének érdekében egy meglehetősen bonyolult sémát használnak mindegyik sodrott érpáron. Ez azzal jár, hogy három feszültségszinttel ternális digiteket kell tobábbítani. Ezt a kódolást valószínűleg nem az eleganciájáért fogják díjazni, ezért a részleteket mellőzzük. Akárhogy is, miután már évtizedek óta négy sodrott érpár fut a telefonkábelekben, a legtöbb irodának lehetősége van a meglévő kábelezést használni. Természetesen ez az irodai telefon feladását vonja maga után, de ez bizonyára nem nagy ár a gyorsabb elektronikus levelekért cserébe.

6 Helyesebb lenne 25 MBaud-ot mondani. *(A lektor megjegyzése)*

7 Helyesebb lenne 20 MBaud-ot mondani. *(A lektor megjegyzése)*

A 100Base-T4 fokozatosan kikerült a használatból, amikor sok irodaépületet újrakábeleztek 5-ös kategóriájú UTP-kábellel, hogy a **100Base-TX** Ethernetet használják, ami mára piacvezető lett. Ez a megvalósítás egyszerűbb, mivel a kábel képes a 125 MHz-es órajelet kezelni. Állomásonként mindössze két sodrott érpárt használnak, egyet az elosztó felé, egyet pedig az elosztó felől. Sem a közvetlen bináris kódolást (azaz NRZ), sem a Manchester-kódolást nem alkalmazzák. Helyettük a **4B/5B** kódolást használják, amit a 2.5. szakaszban mutattunk be. 4 adatbitet kódolnak 5 bitre és küldik át 125 MHz-en, így szolgáltatva a 100 Mb/s-os adatsebességet. Ez a séma egyszerű, ugyanakkor van a szinkronizációhoz megfelelő számú jelátmenete és viszonylag jól kihasználja a vezeték sávszélességét is. A 100Base-**TX** rendszer duplex: az állomások egy időben adhatnak az egyik sodrott érpáron és vehetnek a másikon 100 Mb/s sebességgel.

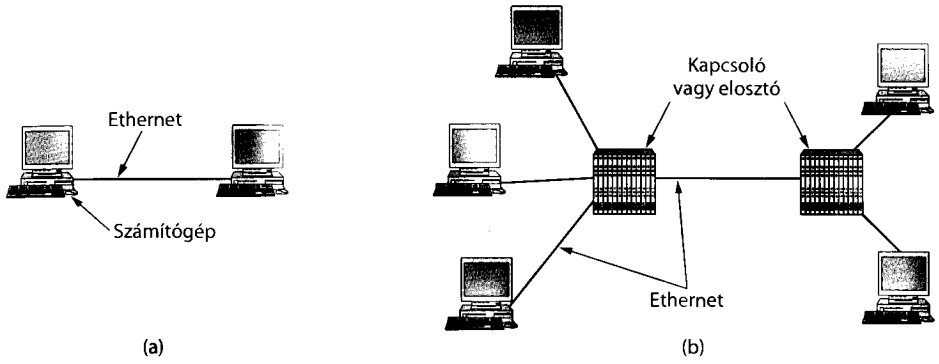
Az utolsó lehetőség, a **100Base-FX** két többmódusú fényvezető szálat használ, mindkét irányban egyet-egyet, így ez is 100 Mb/s-os duplex átvitelt biztosít mindkét irányban. Ebben a kiépítésben egy állomás és a kapcsoló közötti távolság pedig akár 2 km is lehet.

A gyors Ethernet megengedi, hogy az összeköttetés akár elosztóval, akár kapcsolóval történjen. Hogy biztosítsák a CSMA/CD-algoritmus további működését, a legkisebb keretméret és a legnagyobb kábelhossz közötti kapcsolatot fenn kell tartani, a hálózat 10 Mb/s-ról 100 Mb/s-ra való gyorsításakor. Tehát, vagy a 64 bájtos legkisebb keretméretet kell arányosan megnövelni, vagy a 2500 méteres legnagyobb kábelhosszat arányosan kell lerövidíteni. A könnyű választás az volt, hogy a két tetszőleges állomás közötti megengedett legnagyobb kábelhosszt rövidítik le egy 10-es osztóval, mert az elosztók miatt a 100 méteres kábelek már úgyis ezen a korláton belül vannak. A 2 km-es 100Base-FX kábelek azonban túl hosszúak ahhoz, hogy megengedjék egy 100 Mb/s-os normális Ethernet ütközési eljárású elosztó működését. Ezeket a kábeleket elosztó helyett kapcsolóval kell összekötni és duplex módban kell üzemeltetni, így nem lesznek ütközések.

A felhasználók rövidesen elkezdték a gyors Ethernet használatát, de nem dobták ki a régi számítógépeikben lévő 10 Mb/s-os Ethernet-kártyáikat. Ennek következtében gyakorlatilag minden gyors Ethernet-kapcsoló képes vegyesen 10 Mb/s-os és 100 Mb/s-os állomásokat is kezelni. A frissítés megkönnyítésére a szabvány egy **automatikus egyezkedés** (**autonegotiation**) nevű mechanizmust ajánl, amivel két állomás automatikusan megegyezhet az optimális sebességről (10 vagy 100 Mb/s) és a duplexitásról (duplex vagy fél-duplex). Ez jól működik a legtöbb esetben. Köztudott azonban, hogy duplexitás illeszkedési hiba (duplex mismatch) nevű problémához vezet, ha egy kapcsolat egyik vége alkalmazza az automatikus egyezkedést, míg a másik vége nem, ugyanakkor ez utóbbi duplex módba van állítva [Shalunov és Carlson, 2005]. A gyors Ethernet-termékek többsége ki is használja ezt a képességet saját maga konfigurálására.

4.3.6. Gigabites Ethernet

Még alig száradt meg a tinta a gyors Ethernet-szabványon, amikor a 802-es bizottság már elkezdett dolgozni egy még gyorsabb Ethernet tervén, gyorsan rá is ragasztották a **gigabites Ethernet** (**gigabit Ethernet**) nevet. Az IEEE 1999-ben a 802.3ab név alatt hagyta jóvá a legnépszerűbb formáját. Alább megvizsgáljuk a gigabites Ethernet néhány kulcsfontosságú képességét. További információkkal Spurgeon [2000] munkája szolgál.



4.20. ábra. *(a) Ethernet két állomással. (b) Ethernet több állomással*

A bizottság céljai a gigabites Ethernet tervezésénél lényegében megegyeztek a gyors Ethernet tervezésénél alkalmazottakkal: tízszereződjön meg a teljesítmény, de maradjon kompatibilis az összes meglevő Ethernet-szabvánnyal. A gigabites Ethernetnek lényegében nyugtázatlan datagramszolgáltatást kellett nyújtania, mind egyesküldésnél, mind adatszórásnál; használnia kellett a már meglevő 48 bites címzési sémát, és meg kellett őriznie a régi keretformátumot a minimális és maximális keretméretekkel együtt. A végső szabvány meg is felelt ezeknek a céloknak.

Mint ahogy a gyors Ethernet, a gigabites Ethernet is minden kiépítésében kétpontos (point-to-point) kapcsolatokat használ. A legegyszerűbb gigabites Ethernet-kiépítést a 4.20.(a) ábra szemlélteti, ahol két számítógép van közvetlenül összekötve egymással. Gyakoribb azonban az az eset, hogy egy kapcsoló vagy elosztó köt össze több számítógépet, esetleg további kapcsolókat vagy elosztókat, ahogy azt a 4.20.(b) ábra mutatja. Bármelyik kiépítést is nézzük azonban, minden egyes Ethernet-kábel végén pontosan két eszköz található, se több, se kevesebb.

A gyors Ethernethez hasonlóan, a gigabites Ethernet is két különböző működési módot támogat: a duplex és a fél-duplex működést. A duplex mód a „normális" eset, mely lehetővé teszi, hogy mindkét irányban menjen forgalom egyazon időben. Ezt akkor használják, amikor egy központi kapcsolót kötnek össze a periférián lévő számítógépekkel (vagy más kapcsolókkal). Ebben az elrendezésben minden vonalat pufferelnek, így bármely számítógép és kapcsoló tetszése szerinti időben küldheti el a kereteit. Az adónak nem kell figyelnie a csatornát, hogy használja-e azt éppen más is, mert a versengés kizárt. Egy olyan vonalon, mely egy számítógépet és egy kapcsolót köt össze, csak az adott számítógép küldhet adatokat a kapcsoló felé, és az átvitel még akkor is sikeres lesz, ha a kapcsoló éppen keretet küldött a számítógépnek, hiszen a vonal duplex. Mivel nincs versengés, ezért CSMA/CD-protokollt sem használnak, így a maximális kábelhosszt a jel erőssége határozza meg, nem pedig az, hogy legrosszabb esetben mennyi ideig tart egy zajlöketnek visszajutnia az adóig. A kapcsolóknak módjukban áll keverni és egyeztetni a sebességeket. Automatikus egyezkedésre is van lehetőség, akárcsak a gyors Ethernetnél, de most a 10, 100 és 1000 Mb/s-os sebességek közül tudnak választani.

A fél-duplex működési módot akkor használják, ha a számítógépek nem egy kapcsolóhoz, hanem egy elosztóhoz csatlakoznak. Az elosztó nem puffereli a beérkező

kereteket, hanem belül az összes vonalat villamosan összeköti, a klasszikus Ethernet többpontú kábeleit utánozva. Ebben a módban ütközések is történhetnek, ezért a szabványos CSMA/CD-protokollra is szükség van. Mivel egy 64 bájtos (azaz a megengedett legrövidebb) keretet a klasszikus Ethernetnél 100-szorta gyorsabban lehet elküldeni, a maximális távolságnak 100-szor kisebbnek kell lennie, azaz 25 méteresnek, hogy megmaradjon az az alapvető tulajdonság, hogy az adó adása még a legrosszabb esetben is tartson addig, amíg a zajlöket visszaér hozzá. Egy 2500 méter hosszú kábel esetén az 1 Gb/s sebességgel működő adó már rég végezne egy 64 bájtos keret adásával, amikor a keret még a kábel tizedén se haladt végig, nem beszélve a visszaútról.

A hossz ilyetén szabályozása elég fájdalmas volt ahhoz, hogy két új képességet tettek bele a szabványba, hogy 200 méter hosszúságúra növeljék a legnagyobb kábelhosszt, ami valószínűleg a legtöbb irodának megfelel. Az első képességet **vivőjel-kiterjesztésnek** (**carrier extension**) nevezik. Ez lényegében arra utasítja a hardvert, hogy illessze a saját kitöltő bitsorozatát a rendes keret után, hogy a keret hossza elérje az 512 bájtot. Mivel ezt a kitöltést az adó hardvere illeszti be, és a vevő hardvere távolítja el, a szoftver nem is tud róla, vagyis a meglevő szoftvert nem kell megváltoztatni. A hátránya az, hogy 512 bájt átviteléhez szükséges kapacitást használunk fel 46 bájtnyi felhasználói adat (ennyi a 64 bájtos keret adatrésze) átviteléhez, így a vonal hatásfoka mindössze 9%-os lesz.

A második képesség a **keretfűzés** (**frame bursting**). Ez lehetővé teszi, hogy az adó egyetlen adás során több, egymás után fűzött keretet vigyen át. Amennyiben a teljes löket hossza kisebb mint 512 bájt, akkor itt is a hardver végzi a kitöltést. Ha kellően sok keret vár továbbításra, akkor kiemelkedő hatékonysága miatt ezt a sémát szokták előnyben részesíteni a vivőjel-kiterjesztéssel szemben.

Őszintén szólva, nehéz elképzelni egy olyan szervezetet, amely modern számítógépeket vásárol gigabites Ethernet-kártyákkal, majd régimódi elosztókkal köti össze a gépeket, hogy a klasszikus Ethernetet szimulálja, annak ütközéseivel együtt. A gigabites Ethernet-illesztőkártyák elég drágák voltak, de az áruk esett, ahogy az eladott mennyiség nőtt. A számítógépiparban a hátrafelé kompatibilitás fogalma még mindig szent, ezért a bizottságnak sem volt más választása, minthogy elfogadja. Manapság a számítógépeket olyan Ethernet-csatolóval forgalmazzák, amely képes 10, 100 és 1000 Mb/s-os működésre, és amely mindezekkel kompatibilis is.

A gigabites Ethernet támogatja a rézből és a fényvezető szálból készült vezetékeket is, amint azt a 4.21. ábra felsorolása is mutatja. Az 1 Gb/s-os vagy azt megközelítő se-

Név	Vezeték	Max. szegmens	Előnyök
1000Base-SX	Fényvezető szál	550 m	Többmódusú fényvezető szál (50 vagy 62,5 μm átmérő)
1000Base-LX	Fényvezető szál	5000 m	Egy- (10 μm átmérő) vagy többmódusú (50 vagy 62,5 μm átmérő) fényvezető szál
1000Base-CX	2 pár STP	25 m	Árnyékolt sodrott érpár
1000Base-T	4 pár UTP	100 m	Szabványos 5-ös kategóriájú UTP

4.21. ábra. *A gigabites Ethernet kábelezése*

besség megköveteli, hogy minden nanoszekundumban egy bit kódolása és elküldése megtörténjen. Ezt a mutatványt kezdetben rövid, árnyékolt rézkábelekkel (az 1000Base-CX verzió) és fényvezető szálakkal sikerült teljesíteni. A fényvezető szálak esetében két hullámhossz engedélyezett, ezért két különböző verziójuk van: a 0,85 mikronos hullámhossz (rövid, 1000Base-SX) és az 1,3 mikronos hullámhossz (hosszú, 1000Base-LX).

A rövid hullámhosszú jelzés olcsó LED-ekkel megvalósítható. Többmódusú fényvezető szálat használ, és épületen belüli összeköttetések kiépítésénél hasznos, mivel az 50 mikron átmérőjű fényvezető szál akár 500 m hosszan is futhat. A hosszú hullámhosszú jelzés megköveteli a drágább lézereket. Amikor viszont egymódusú (10 mikron átmérőjű) fényvezető szállal kombinálják, a kábel hossza akár 5 km is lehet. Ez a felső határ lehetővé teszi az épületek közötti hozzárendelt (dedikált), nagy távolságú kétpontos (point-to-point) összeköttetések megvalósítását, mint amilyen például egy egyetemi gerinchálózat. A szabvány későbbi változatai még hosszabb kapcsolatokat is megengednek egymódusú fényvezető szálon.

Hogy biteket a gigabites Etherneten keresztül lehessen küldeni, a 2.5. szakaszban bemutatott **8B/10B** kódolást vették kölcsön egy másik, Fibre Channelnek nevezett hálózati megoldástól. A séma 8 adatbitet 10 bites kódszavakba foglalja, innen származik a 8B/10B név. A kódszavakat úgy választották meg, hogy kiegyensúlyozható legyen (azaz ugyanannyi 0-st és 1-est tartalmazzon) és elegendő jelátmenetet tartalmazzon az órák újraszinkronizálásához. Az NRZ-vel kódolt bitek küldése 25%-kal nagyobb sávszélességet igényel a kódolatlan bitek küldéséhez képest, ami nagy előrelépést jelent a Manchester-kódolás 100%-os többletéhez képest.

Mindenesetre, ezek a tulajdonságok új rézvezetéket vagy fényvezető szálakat igényeltek a gyorsabb jelzések támogatásához. Egyik sem hasznosítja a gyors Ethernettel telepített nagy mennyiségű 5-ös kategóriájú UTP-kábelt. Egy éven belül megérkezett az 1000Base-T, hogy kitöltse a rést, és azóta is ez a gigabites Ethernet legnépszerűbb formája. Úgy tűnik, az emberek nem szeretik újrakábelezni az épületeiket.

Még bonyolultabb jelzés szükséges, hogy az Ethernet 1000 Mb/s sávszélességgel fusson az 5-ös kategóriájú vezetékeken. Először is, a kábel mind a négy sodrott érpárját használják, ráadásul egyszerre mindkét irányba. A különböző irányú jelek szétválasztásához digitális jelfeldolgozást használnak. Mindegyik vezetéken a jelzésekhez 5 feszültségszinten visznek át 2 bitet 125 megaszimbólum/másodperc sebességgel. A bitek szimbólumokká történő leképezése nem magától értetődő. A nagyobb számú jelátmenet érdekében tördeléseket tartalmaz, ezután egy hibajavító kódolás jön, ami 4 értéket képez le 5 jelszintre.

Az 1 Gb/s-os sebesség meglehetősen gyorsnak számít. Például, ha egy vevő akár csak 1 ms ideig is valami mással van elfoglalva, és nem üríti a bemeneti puffert valamelyik vonalon, akkor akár 1953 keret is összegyűlhet a szünet alatt. Ugyanígy, ha egy gigabites Etherneten levő számítógép egy klasszikus Etherneten levő másik gépnek küldi át az adatokat, akkor nagy valószínűséggel puffertúlcsordulás lép fel. E két megfigyelésnek köszönhetően a gigabites Ethernet támogatja a forgalomszabályozást. A mechanizmus abból áll, hogy az egyik végpont egy speciális vezérlőkeretet küld a másiknak, melyben arra utasítja, hogy bizonyos ideig tartson szünetet. Ezek a PAUSE (SZÜNET) vezérlőkeretek 0x8808 típus kódú szokványos Ethernet-keretek. A szünet hosszát a minimális keretidő időegységében adják meg. A gigabites Ethernet esetében ez az időegység 512 ns, amely lehetővé teszi, hogy a szünetek 33,6 ms hosszúak legyenek.

A gigabites Ethernet még egy kiterjesztést vezetett be. Az **óriáskeret** vagy **jumbokeret** (**Jumbo frame**) használata lehetővé teszi az 1500 bájtnál hosszabb keretek használatát általában 9 KB felső határig. Ez a kiterjesztés nem szabványos. Azért nem ismerték el a szabvány részeként, mert a használatával a korábbi Ethernet-verziókkal inkompatibilissé válna. Ennek ellenére a legtöbb gyártó támogatja. A használatának alapvető oka, hogy az 1500 bájt túl kicsi egység a gigabites sebességhez. Nagyobb információtömbbel való foglalkozás csökkenti a keret sebességét és a vele járó adatfeldolgozást, mint amilyen például a processzornak küldött megszakítás, amely értesíti a processzort egy keret érkezéséről, vagy az egy Ethernet-keretbe nem férő üzenet feldarabolásáról majd újra összerakásáról.

4.3.7. 10 gigabites Ethernet

Mire a gigabites Ethernet szabványosítása befejeződött, a 802-es bizottság már elunta magát, és újra dolgozni akart. Az IEEE javaslatára aztán nekiláttak a 10 gigabites Ethernet kidolgozásának. A munkamenet követte a korábbi Ethernet-szabványok munkamenetének mintáját: szabványt dolgoztak ki fényvezető szálra és árnyékolt rézkábelre először 2002-ben, majd 2004-ben, amit a réz sodrott érpárt használó szabvány követett 2006-ban.

10 Gb/s óriási sebesség, 1000-szer gyorsabb az eredeti Ethernetnél. Vajon hol lehet erre szükség? A válasz az, hogy adatközpontok és kapcsolóközpontok belsejében, amelyek felső kategóriás útválasztókat, kapcsolókat és szervereket kötnek össze, valamint irodák közötti nagy távolságú és nagy sávszélességű trönköknél, amelyek lehetővé teszik, hogy egész nagyvárosi hálózatok Ethernetre és fényvezető szálra épüljenek. A nagy távolságú összeköttetésekhez fényvezető szálat használnak, míg a rövid összeköttetésekhez használhatnak rézvezetéket vagy fényvezető szálat.

A 10 gigabites Ethernet összes változata kizárólag a duplex működést támogatja. A CSMA/CD már kikerült a tervekből, és a szabvány a nagyon nagy sebességre képes fizikai réteg részleteire koncentrál. A kompatibilitás azonban még mindig fontos, ezért a 10 gigabites Ethernet-csatolók alkalmazzák az automatikus egyezkedést és a vonal mindkét vége által támogatott, a legnagyobb sebességre történő visszalépést.

A 10 gigabites Ethernet főbb változatait a 4.22. ábra mutatja. A többmódusú fényvezető szálat használják 0,85 μm (S, rövid) hullámhosszal közepes távolságokra, és az egymódusú fényvezető szálat használják 1,3 μm (L, hosszú) és 1,5 μm (E, kiterjesztett) hullámhosszal nagy távolságokra. 10GBase-ER akár 40 km hosszú is lehet, ami alkalmazhatóvá teszi nagy kiterjedésű hálózatokban. Mindegyik változat egy soros adatfolyamot küld, ami az adatbitek tördelésével és ezután **64B/66B** kódolásával jön létre. Ez a kódolás kisebb többletet eredményez, mint a 8B/10B kódolás.

Először a rézalapú verziót szabványosították, a 10GBase-CX4-et, ami 4 pár twinaxiális rézvezetékből álló kábelt használ. Minden vezetékpár 8B/10B kódolást alkalmaz és 3,125 Giga-szimbólum/másodperc sebességen fut, így adva ki 10 Gb/s-ot. Ez a változat olcsóbb, mint a fényvezető szál és korán a piacra került. A jövő kérdése azonban, hogy

Név	Vezeték	Max. szegmens	Előnyök
10GBase-SR	Fényvezető szál	300 m-ig	Többmódusú fényvezető szál (0,85 μm hullámhossz)
10GBase-LR	Fényvezető szál	10 km	Egymódusú fényvezető szál (1,3 μm hullámhossz)
10GBase-ER	Fényvezető szál	40 km	Egymódusú fényvezető szál (1,5 μm hullámhossz)
10GBase-CX4	4 pár twinax	15 m	Twinaxiális rézkábel
10GBase-T	4 pár UTP	100 m	6a-s kategóriájú UTP

4.22. ábra. *A gigabites Ethernet kábelezése*

vajon hosszú távon a közönséges sodrott rézpáron futó 10 gigabites Ethernet kiüti-e majd a nyeregből.

10GBase-T az a verzió, ami UTP-kábelt használ és 6a-s kategóriájú vezetékezést kíván, de kisebb távolságra megengedi a meglévő alacsonyabb kategóriájú (5-ös kategóriájúig) kábelek újrafelhasználását. Nem meglepő, hogy a fizikai réteg meglehetősen bonyolult, hogy sodrott érpáron elérje a 10 Gb/s sávszélességet. Csak néhány felsőszintű részletet vázolunk fel. Mind a 4 sodrott érpár használatban van, hogy 2500 Mb/s sebességgel tudjon mindkét irányban adni. Ezt a sebességet 800 megaszimbólum/másodperc sebességgel és szimbólumonkénti 16 feszültségszinttel éri el. A szimbólumok előállítása az adatok tördelésével, LDPC (Low Density Parity Check – alacsony sűrűségű paritásellenőrző) kódolással és a hibajavítás érdekében további hibajavító kódolással történik.

A 10 gigabites Ethernet még a helyét keresi a piacon, de a 802.3-es bizottság már továbblépett. 2007 végén az IEEE létrehozott egy csoportot, hogy szabványosítsa azt az Ethernetet, amely 40 Gb/s és 100 Gb/s adatsebességgel üzemel. Ez a továbbfejlesztés az Ethernetet az olyan nagyon nagy teljesítményű rendszerek versenytársává fogja tenni, mint amilyenek a gerinchálózatokon lévő nagy távolságú összeköttetések és az eszközök hátlapján lévő kis távolságú összeköttetések. Bár a szabvány még nincs kész, a gyártók saját változatú termékei már elérhetők.

4.3.8. Visszatekintés az Ethernetre

Az Ethernet már több mint három évtizede létezik, és még mindig nem akadt komoly vetélytársa, így valószínűleg még sokáig fennmarad. Kevés processzorarchitektúra, operációs rendszer vagy programozási nyelv mondhatja el magáról, hogy három évtizede uralja a maga területét. Az Ethernet tehát valamit nagyon eltalált. De vajon mit?

Hosszú életének legfőbb oka valószínűleg az, hogy egyszerű és rugalmas. Az „egyszerű" a gyakorlatban azt jelenti, hogy megbízható, olcsó és könnyű karbantartani. Amióta az elosztót és kapcsolót tartalmazó architektúrák elterjedtek, a hibák rendkívül ritkává váltak. Az emberek pedig kétszer is meggondolják, hogy lecseréljenek-e valamit, ami mindig is tökéletesen működött, különösen annak a tudatában, hogy a számítógépiparban hihetetlenül sok dolog működik nagyon gyatrán, és sok úgynevezett „továbbfejlesztés" szembetűnően rosszabb annál a rendszernél, amit felváltani hivatott.

Az „egyszerű" tehát olcsót is jelent. A sodrott érpáros kábelezés ára viszonylag kedvező ugyanúgy, ahogy a hardverkomponensek ára is. Lehetséges, hogy ez utóbbiak kezdetben drágák voltak az átmeneti időszak alatt, például az új gigabites Ethernet-csatolók vagy -kapcsolók, de ezek pusztán a meglévő hálózat kiegészítései (nem a helyettesítései), és az áruk gyorsan esik az eladott mennyiség növekedésével.

Az Ethernetet könnyű karbantartani. A meghajtókon (driver) kívül nem kell más szoftvert telepíteni, és nem nagyon kell kezelni konfigurációs táblázatot sem, amit el lehetne rontani. Az új hosztokat pedig egyszerűen csak csatlakoztatni kell, és már működnek is.

Fontos érv az is, hogy az Ethernet jól együttműködik a mára egyeduralkodóvá vált TCP/IP-vel. Az összeköttetés nélküli IP tökéletesen illeszkedik a szintén összeköttetés nélküli Ethernethez. Az IP sokkal kevésbé illik az olyan összeköttetés-alapú alternatívákhoz, mint amilyen az ATM is. Ez a különbözőség határozottan rontotta az ATM esélyeit.

Végül a valószínűleg leglényegesebb: az Ethernet a kritikus területeken is képes volt a fejlődésre. A sebessége több nagyságrenddel nőtt, megjelentek az elosztók és a kapcsolók, de mindezen újítások nem igényelték a szoftver megváltoztatását és gyakran a meglévő kábelezést is újra fel lehetett használni. Ha egy hálózatokkal kereskedő ügynök megjelenik egy nagy telephelyen, és azt mondja: „Ajánlom önnek ezt a fantasztikus új hálózatot! Nem is kell mást tennie, mint hogy kidobja az összes hardverét, és újraírja az összes szoftverét!" – nos, akkor máris bajban van.

Sok alternatív hálózati megoldás, amiről Ön valószínűleg nem is hallott, a bevezetésükkor gyorsabbak voltak, mint az Ethernet. Ilyenek az ATM, az FDDI (Fiber Distributed Data Interface – fényvezetőszálas osztott adatinterfész) és a Fibre Channel[8] (fényvezetőszálas csatorna). Az utóbbi két hálózat kettős gyűrűalapú optikai helyi hálózat volt, azonban egyik sem volt kompatibilis az Ethernettel. Mivel túl bonyolultak voltak, ami összetett chipekhez és magas árakhoz vezetett, nem is maradtak versenyben. A gyártóknak ismerniük kellett volna a KISS leckét (Keep It Simple, Stupid – tartsd meg egyszerűnek, butának). Az Ethernet pedig idővel a sebesség terén is utolérte ezeket, gyakran egyes megoldásaik átvételével, például a 4B/5B kódolást az FDDI-től és a 8B/10B kódolást a Fibre Channeltől. Eztán nem is maradt több előnyük, és csendesen ki is múltak vagy speciális feladatokra korlátozódtak.

Úgy tűnik, hogy az Ethernet alkalmazási területe bővülni fog az elkövetkező időkben. A 10 gigabites Ethernet megszabadult a CSMA/CD távolságkorlátjától. Sok fejlesztés irányul a **nagy távolságú Ethernetre** (**carrier-grade Ethernet**), ami lehetővé teszi a szolgáltatóknak, hogy Ethernet-alapú szolgáltatást ajánljanak fogyasztóiknak nagyvárosi és nagy kiterjedésű hálózatokhoz [Fouli és Maler, 2009]. Ez az alkalmazás Ethernet-kereteket szállít nagy távolságra fényvezető szálon keresztül, és jobb felügyeleti képességet igényel az operátorok támogatására, hogy megbízható, magas színvonalú szolgáltatást ajánljanak. Nagyon nagy sebességű hálózatokat kezdenek alkalmazni a

8 Azért lett „Fibre Channel" és nem „Fiber Channel", mert a dokumentum szerkesztője brit volt. *(A szerző megjegyzése)*

nagy útválasztók és szerverek hátlapján a komponensek összekötésére is. Mindkét fent említett alkalmazási terület ráadás ahhoz képest, hogy irodai számítógépek között kereteket lehet küldözgetni.

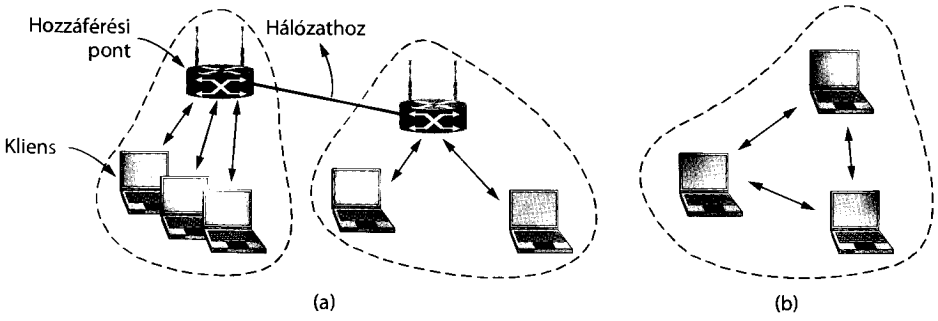
4.4. Vezeték nélküli LAN-ok

A vezeték nélküli LAN-ok egyre népszerűbbek lesznek, és otthonokban, irodákban, kávézókban, könyvtárakban, repülőtereken, állatkertekben és más közterületen jelennek meg, hogy számítógépeket, PDA-kat és okostelefonokat csatlakoztassanak az internethez. Két vagy több számítógép tud kommunikálni egymással a vezeték nélküli LAN-ok használatával úgy, hogy nem használják az internetet.

A fő vezeték nélküli LAN-szabvány a 802.11. Az 1.5.3. szakaszban adtunk már némi háttér-információt róla. Most itt az ideje, hogy közelebbről is szemügyre vegyük ezt a megoldást. A következő szakaszokban megvizsgáljuk a protokollkészletét, a fizikai réteg rádiós átviteli módszereit, a MAC-alréteg protokollját, a keretszerkezetet és a nyújtott szolgáltatásait. A 802.11-ről bővebb információval szolgál Gast [2005] munkája. Aki pedig a legközvetlenebb forrásra kíváncsi, az tanulmányozza magát az IEEE 802.11-2007-es szabványt.

4.4.1. A 802.11 felépítése és protokollkészlete

A 802.11 hálózatokat kétféleképpen lehet használni. A legnépszerűbb üzemmód célja az, hogy kliensgépeket (például laptopokat és okostelefonokat) más hálózatokhoz, például egy vállalati intranethez vagy az internethez csatlakoztasson. Ezt a módot a 4.23.(a) ábra szemlélteti. Az infrastruktúra üzemmódban minden kliensgép egy **hozzáférési ponthoz (Access Point, AP)** csatlakozik, amely viszont egy másik hálózathoz kapcsolódik. A kliensgép a hozzáférési ponton keresztül küldi és fogadja a csomagjait. Több hozzáférési pontot lehet összekötni egy **elosztórendszernek (distribution system)** nevezett vezetékes hálózattal, hogy egy kiterjedtebb 802.11 hálózatot alakuljon ki. Ebben



4.23. ábra. *A 802.11 felépítése (a) Infrastruktúra mód. (b) Ad hoc mód*

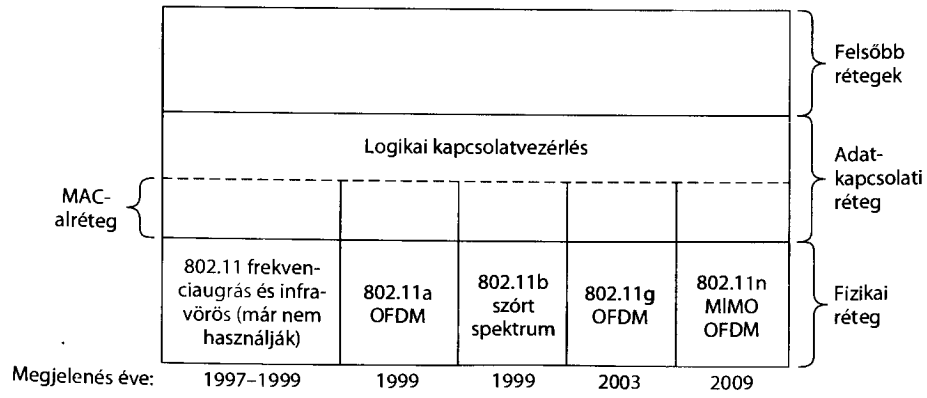
az esetben, a kliensgépek a hozzáférési pontjaikon keresztül küldhetnek kereteket más kliensgépeknek.

A másik üzemmód, amelyet a 4.23.(b) ábra mutat be, az **ad hoc hálózat**. Ebben az üzemmódban egy csoport számítógép úgy kapcsolódik egymáshoz, hogy hozzáférési pont nélkül közvetlenül tudnak kereteket küldeni egymásnak. Mivel az internet-hozzáférés a vezeték nélküli hálózatok legütősebb felhasználása, ezért az ad hoc hálózatok nem igazán népszerűek.

Az alábbiakban a protokollokat vesszük vizsgálat alá. Minden 802-es protokoll, beleértve a 802.11-et és az Ethernetet is, rendelkezik egy bizonyos közös felépítéssel. A 802.11 protokollkészletének egy részlete látható a 4.24. ábrán. A klienseknek és a hozzáférési pontoknak ugyanaz a protokollkészlete. A fizikai réteg nagyjából az OSI fizikai rétegének felel meg, az adatkapcsolati réteg viszont minden 802-es protokollban két vagy több alrétegre bomlik. A 802.11 esetében a MAC (Medium Access Control – közeg-hozzáférési alréteg) dönt a csatornakiosztásról, vagyis arról, hogy ki lesz a soron következő adó. Fölötte található az LLC (Logical Link Control – logikai kapcsolatvezérlés) alréteg, melynek az a feladata, hogy elrejtse a különböző 802-es változatok eltéréseit, és a hálózati réteg szempontjából megkülönböztethetetlenné tegye azokat. Ez egy fontos feladat lehetett volna, de manapság az LLC egy ragasztóréteg, amely azonosítja a 802.11-kereten belül szállított protokollt (például IP).

A 802.11 fizikai rétegét az 1997-es megjelenése óta számos átviteli megoldással bővítették. A kezdeti megoldások közül kettő, a tv-készülékek távirányítójához hasonlóan működő infravörös átvitel, és a 2,4 GHz-es sávban működő frekvenciaugrás (frequency hopping) már használaton kívülre került. A harmadik kezdeti átviteli megoldást, a 2,4 GHz-es sávban 1 vagy 2 Mb/s sávszélességű DSSS-t (Direct Sequence Spread Spectrum – közvetlen sorozatú szórt spektrum) kibővítették, hogy akár 11 Mb/s adatsebességgel is működjön, ennek eredményeként gyorsan nagy siker lett. Ezt most 802.11b-ként ismerik.

Hogy a vezeték nélküli hálózatok rajongóinak megadják a nagyon várt sebességnövelést, a 2.5.3. szakaszban bemutatott OFDM-en (Orthogonal Frequency Divison Multiplexing – ortogonális frekvenciaosztásos nyalábolás) alapuló új átviteli megoldásokat vezettek



4.24. ábra. *A 802.11 protokollkészlet egy részlete*

be 1999-ben és 2003-ban. Az elsőt 802.11a-nak nevezték és egy másik frekvenciasávot, az 5 GHz-es sávot használja. A második ragaszkodott a 2,4 GHz-es sávhoz és a kompatibilitáshoz. Ezt 802.11g-nek nevezik. Mindkettő legnagyobb adatsebessége 54 Mb/s.

Legutóbb 2009 októberében, 802.11n néven olyan átviteli megoldásokat véglegesítettek, amelyek egyidejűleg több antennát használnak adásra és vételre a sebesség növelése érdekében. Négy antennával és nagyobb sávszélességű csatornákkal a 802.11-es szabvány már az elképesztően nagy 600 Mb/s-ig definiál sebességeket.

Most röviden megvizsgáljuk az átviteli eljárásokat, de csak azokat, amelyek használatban vannak. Így az elavult 802.11-es átviteli módokat kihagyjuk. Műszaki szempontból ezek ugyan a fizikai réteghez tartoznak, így a 2. fejezetben kellett volna szót ejteni róluk, de olyan szorosan kötődnek általában a LAN-okhoz, és különösen a 802.11 LAN-hoz, hogy inkább itt tárgyaljuk ezeket.

4.4.2. A 802.11 fizikai rétege

Mindegyik átviteli eljárás lehetővé teszi egy MAC-keretnek az egyik állomásról a másikra a levegőben történő elküldését. A különbség az ehhez felhasznált műszaki megoldásban és az elérhető sebességben van. A megoldások részletes tárgyalása messze túlmutat könyvünk keretein, de néhány szót ejtünk minden megoldásról, amely összefüggésbe hozza a módszert a 2.5. szakaszban leírt anyaggal, és amely ellátja az érdeklődő olvasókat kulcsszavakkal a további tájékozódáshoz.

Valamennyi 802.11 módszer rövid hatótávolságú rádiókat használ jelek adására a 2.3.3. szakaszban már bemutatott 2,4 GHz-es vagy az 5 GHz-es ISM-sávban. Ezeknek a frekvenciasávoknak az az előnye, hogy külön engedély nélkül használhatók és ezért ingyen elérhetők azon adók számára, amelyek betartanak néhány korlátozást, mint például a sugárzott teljesítmény nem lehet nagyobb 1 W-nál (bár a vezeték nélküli LAN-adók leggyakrabban csak 50 mW-tal adnak). Sajnos ezt a tényt a garázsnyitók, vezeték nélküli telefonok, mikrohullámú sütők és számtalan más eszköz gyártója is ismeri. Ezek a termékek a laptopokkal versengenek ugyanazért a spektrumért. A 2,4 GHz-es sáv kezd zsúfoltabbá válni, mint az 5 GHz-es sáv, ezért az 5 GHz megfelelőbb néhány alkalmazás számára annak ellenére, hogy a nagyobb frekvencia miatt rövidebb a hatótávolsága.

Mindegyik átviteli módszer több átviteli sebességet határoz meg. A különböző adatsebességek mögött meghúzódó gondolat az, hogy mindig az aktuális feltételeknek legmegfelelőbb sebesség legyen használható. Ha gyenge a jel, egy kisebb sebesség; ha a jel tisztán vehető, akkor a legnagyobb sebesség legyen használható. Ezt a hangolást **sebességadaptálásnak** (**rate adaptation**) hívják. Mivel a sebességek között 10-szeres vagy nagyobb arány is lehet, a jó sebességadaptálás fontos a jó teljesítmény eléréséhez. Természetesen, mivel ez nem szükséges az eszközök együttműködéséhez, ezért a szabvány nem határozza meg, hogyan kell a sebességadaptációt végezni.

Az első átviteli módszer, amit megnézünk, a **802.11b**. Ez egy szórt spektrumú eljárás, amely 1, 2, 5,5 és 11 Mb/s adatsebességeket támogat, jóllehet a gyakorlatban a működési sebesség majdnem mindig 11 Mb/s. Ez hasonló a CDMA-rendszerhez, amit a 2.5. szakaszban vizsgáltunk, azzal a különbséggel, hogy a felhasználók csak egyetlen közös szórókódot használnak. A spektrumszórást az FCC követelményeinek kielégítésére használják, amelynek előírása szerint az ISM-sávban a jel teljesítményét szét kell szórni. A 802.11b által használt szórósorozat a **Barker-sorozat** (**Barker sequence**), amely rendelkezik azzal a tulajdonsággal, hogy alacsony az autokorrelációja kivéve, ha a sorozatok egymáshoz vannak igazítva. Ez a tulajdonság lehetővé teszi a vevőnek, hogy az átvitel kezdetéhez rögzítse magát. Az 1 Mb/s sebességhez a Barker-sorozatot BPSK-modulációval használják, amely 1 bitet 11 chipben visz át. A chipeket 11 millió chip/s sebességgel küldik. A 2 Mb/s sebességhez a Barker-sorozatot QPSK-modulációval használják, amely 2 bitet 11 chipben visz át. A nagyobb sebességek másként működnek. Ezeken a sebességeken a Barker-sorozat helyett egy **CCK**-nak (**Complementary Code Keying – kiegészítő kód billentyűzés**) nevezett módszert alkalmaznak a kódok létrehozásához. Az 5,5 Mb/s sebességnél 4 bitet küldenek egy 8 chipes kódszóban és 11 Mb/s sebességnél 8 bitet egy 8 chipes kódszóban.

A következő átviteli módszer a **802.11a**, amely 54 Mb/s sebességig támogatja az átvitelt az 5 GHz-es ISM-sávban. Azt hihetnénk, hogy a 802.11a korábbi, mint a 802.11b, azonban nem ez a helyzet. Annak ellenére, hogy a 802.11a munkacsoportot hamarabb hozták létre, a 802.11b szabványt előbb hagyták jóvá, és a termékei hamarabb érték el a piacot, mint a 802.11a termékek, részben az 5 GHz-es sáv jelentette nagyobb frekvenciák működtetésének nehézségei miatt.

A 802.11a módszer az **OFDM**-et (**Orthogonal Frequency Division Multiplexing – ortogonális frekvenciaosztásos multiplexelés**) használja, mert az OFDM hatékonyan használja ki a spektrumot, és ellenálló az olyan vezeték nélküli jelromlással szemben, mint a többutas terjedés. A biteket párhuzamosan 52 alvivőn küldik, amelyből 48 az adatbiteket szállítja és 4-et szinkronizációra használnak. Minden szimbólum 4 μs-ig tart és 1, 2, 4 vagy 6 bitet szállít. A biteket a hibajavítás érdekében először bináris konvolúciós kóddal kódolják, tehát a biteknek csak az 1/2-e, a 2/3-a vagy a 3/4-e nem redundáns. Különböző kombinációkkal a 802.11a nyolc különböző sebességgel képes működni, a 6 Mb/s-tól 54 Mb/s-ig terjedő tartományban. Ezek a sebességek jelentősen nagyobbak a 802.11b sebességértékeinél, valamint kevesebb az interferencia az 5 GHz-es sávban. Ennek ellenére a 802.11b hatótávolsága körülbelül hétszer nagyobb, mint a 802.11a szabványé, ami sok esetben fontosabb.

Bár a 802.11b-nek nagyobb volt a hatótávolsága, a 802.11b munkacsoport tagjai nem akarták hagyni, hogy az újonnan jött szabvány megnyerje a sebességversenyt. Szerencsére 2002 májusában az FCC eltörölte azt a rég fennálló szabályt, miszerint az USA-ban minden az ISM-sávban működő eszköznek használnia kell a spektrumszórást. Tehát nekiláttak a **802.11g** kidolgozásának, amit az IEEE 2003-ban jóváhagyott. Ez másolja a 802.11a modulációs eljárását, az OFDM-et, de a keskeny 2,4 GHz-es sávban működik a 802.11b-vel együtt, továbbá pontosan ugyanazokat az adatsebességeket nyújtja, mint a 802.11**a** (6 Mb/s-tól 54 Mb/s-ig), és természetesen kompatibilis az éppen közelben lévő 802.11b eszközökkel. Mivel ezek a különböző választási lehetőségek összezavarhatják a vevőket, az az általános, hogy minden csatolókártya támogatja a 802.11a/b/g-t.

Az IEEE bizottság nem volt annyira elégedett, hogy itt abbahagyja a munkát. Elkezdett dolgozni egy nagy átbocsátóképességű fizikai rétegen, amelyet **802.11n**-nek neveznek és 2009-ben fogadták el. A 802.11n esetében az volt a cél, hogy az átbocsátóképesség legalább 100 Mb/s legyen, miután a vezeték nélküli átvitel többletbitjeit eltávolították. Ez a cél megkövetelte, hogy a nyers sebességet legalább a négyszeresére növeljék. Hogy

ez lehetségessé váljon, a bizottság megkétszerezte a csatorna sávszélességét 20 MHz-ről 40 MHz-re, és csökkentette a keretezési többletterheket azáltal, hogy lehetővé tette azt, hogy a kereteket csoportban lehet együtt küldeni. Ennél fontosabb viszont az, hogy a 802.11n akár négy antennát is használhat annak érdekében, hogy négy információfolyamot küldjön egyszerre. A folyamok jelei a vevőnél interferenciát okoznak, de a **MIMO (Multiple Input Multiple Output – több bemenet-több kimenet)** kommunikációs módszerrel el lehet ezeket különíteni. Több antenna használata nagy sebességnövelést eredményez, vagy helyette nagyobb hatótávolságot és nagyobb megbízhatóságot. A MIMO, az OFDM-hez hasonlóan, az egyik azok közül az okos kommunikációs elképzelések közül, amelyek megváltoztatják a vezeték nélküli rendszerek tervezését, és amelyekről még valószínűleg sokat fogunk hallani a jövőben. A többantennás rendszerekbe való rövid bevezetésért forduljon Halperin és mások [2010] munkájához.

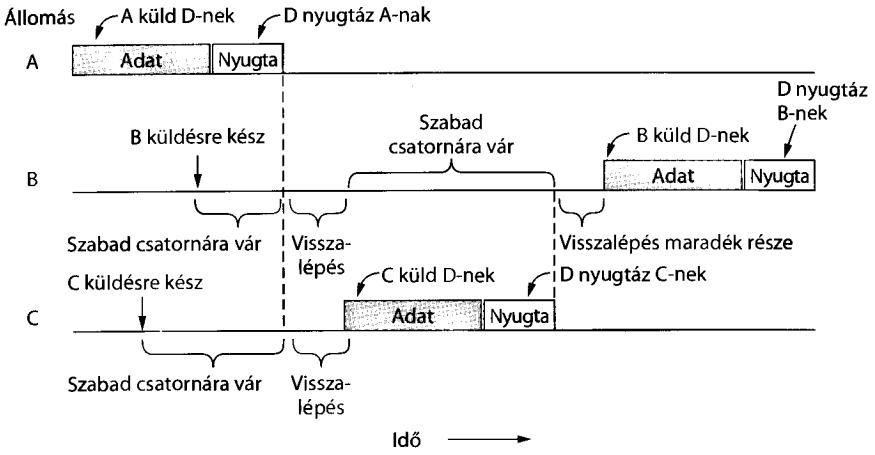
4.4.3. A 802.11 MAC-alrétegének protokollja

Térjünk most vissza a villamosmérnököktől az informatika földjére! A 802.11 MAC-alrétegének protokollja meglehetősen eltér az Ethernet MAC-protokolljától, a vezeték nélküli kommunikáció két alapvető jellegzetessége miatt.

Az első jellegzetesség az, hogy a rádiók majdnem mindig fél-duplexek, vagyis nem képesek egyidejűleg ugyanazon a frekvencián adni és zajlöketeket venni. A vett jel gyakran milliószor gyengébb a küldött jelnél, tehát egyszerre nem lehet hallani mindkettőt. Egy Ethernet-állomás egyszerűen csak addig vár, amíg a közegen csend lesz, aztán elkezd adni. Ha nem jut vissza hozzá zajlöket az első 64 bájt küldésének ideje alatt, akkor szinte biztos lehet benne, hogy a keret hibátlanul célba ért. A vezeték nélküli esetben ez az érzékelési mechanizmus nem működik.

Ehelyett a 802.11 a **CSMA/CA**-nak (**CSMA with Collision Avoidance – CSMA ütközéselkerüléssel**) nevezett protokollal próbálja meg elkerülni az ütközéseket. Ez a protokoll alapjaiban hasonló az Ethernet által használt CSMA/CD-hez, mert figyeli a csatornát mielőtt az adni kezd, és alkalmazza a kettes exponenciális visszalépés eljárást az ütközések után. Viszont egy állomás, amelynek van küldeni való kerete, véletlenszerű ideig vár (kivéve azt az esetet, amikor nem használta a csatornát előzőleg és a csatorna szabad). Nem vár viszont ütközésre. A várakozással töltött időszeletek számát a 0 és, mondjuk, a 15 közötti intervallumból választja, ha az OFDM fizikai réteget használja. Az állomás addig vár, amíg a csatorna szabad lesz. Ezt azzal érzékeli hogy nincs jel egy bizonyos rövid ideig (ezt a rövid időszakot DIFS-nek nevezik és lejjebb részletesen kifejtünk). Ezalatt visszaszámolja a tétlen időszeleteket, de szünetet tart, amikor valaki kereteket küld. Amikor a számláló elérte a 0-t, elküldi a kereteit. Ha a keret megérkezett, a célállomás rögtön egy rövid nyugtaüzenetet küld. A nyugta megérkezésének hiányát hibajelentésnek veszi, nem törődve azzal, hogy ütközésből vagy más okból történt hiba. Ebben az esetben a küldő megduplázza a visszalépés időtartamát, és újra próbálkozik, mint az Ethernet esetében, egészen addig, amíg a keret sikeresen meg nem érkezett vagy az újraküldések maximális számát el nem érte.

Egy példa időbeli lefutását szemléleti a 4.25. ábra. Az *A* állomás az első, amelyik keretet küld. Amíg *A* ad, *B* és *C* állomások adásra készek lesznek. Látják, hogy a csatorna



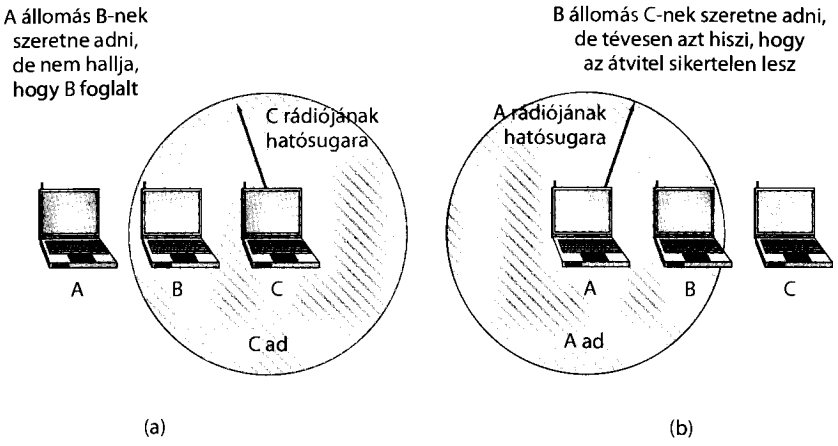
4.25. ábra. *Keret küldése CSMA/CA segítségével*

foglalt és várnak, hogy a csatorna szabad legyen. Viszont ahelyett, hogy rögtön keretet küldenének, amelyek ütköznének, *B* is és *C* is alkalmazza a visszalépéses eljárást. *C* egy rövid visszalépést sorsolt, és ezért az ad először. *B* leállítja a visszaszámlálását, amíg érzékeli, hogy *C* használja a csatornát. *B* nem sokkal később befejezi a visszalépését és elküldi a keretét.

Az Ethernethez képest két lényeges különbség van. Egyrészt, a visszalépés korai kezdése segít az ütközések elkerülésében. Ez az elkerülés megéri, mert az ütközések költségesek, ugyanis az egész keretet újra kell küldeni, még ha csak egy ütközés is történik. Másrészt, a nyugtákból kell az ütközésekre következtetni, mert az ütközést nem lehet érzékelni.

Ezt a működési módot **DCF**-nek (**Distributed Coordination Function – elosztott koordinációs funkció**) nevezik, mivel minden állomás a többitől függetlenül cselekszik, bármilyen központi irányítás nélkül. A szabvány egy opcionális működési módot is tartalmaz, amit **PCF**-nek (**Point Coordination Function – pont-koordinációs funkció**) neveznek, amely esetben egy hozzáférési pont vezérel minden tevékenységet a saját cellájában csakúgy, mint egy bázisállomás egy mobiltelefon-hálózatban. A PCF-et azonban a gyakorlatban nem használják, mert normális esetben nincs lehetőség arra, hogy másik közeli hálózathoz tartozó állomásokat visszatartsanak vetélkedő adatforgalom küldésétől.

A második jellegzetesség az, hogy különböző állomások hatótávolsága különböző lehet. A vezetékes rendszereket úgy alakították ki, hogy minden állomás hallja minden más állomás adását. A rádiófrekvenciás terjedés bonyodalmai miatt ez a megállapítás nem érvényes a vezeték nélküli állomásokra. Ezáltal előfordulhatnak olyan, már korábban említett szituációk, mint a 4.26.(a) ábrán újra vázolt rejtett állomás problémája. Mivel nincs minden állomás egymás hatótávolságán belül, ezért a cella egyik részében zajló adás a cella nem minden részében vehető. Ebben a példában a *C* állomás ad *B* állomásnak. Ha *A* állomás figyeli a csatornát, nem fog hallani semmit, és ezért hamisan arra következtet, hogy elkezdhet adni *B*-nek. Ez a döntés ütközést okoz.



4.26. ábra. *(a) A rejtett állomás problémája. (b) A megvilágított állomás problémája*

A megfordított helyzet, a megvilágított állomás problémája, amelyik a 4.26.(b) ábrán látható. Itt *B* szeretne adni *C*-nek, tehát figyeli a csatornát. Amikor hall egy adást, hibásan arra következtet, hogy nem küldhet *C*-nek annak ellenére, hogy *A* valójában (a nem ábrázolt) *D*-nek küldi adását. Ez a döntés elveszteget egy adási lehetőséget.

Hogy csökkentsék a többértelműséget azzal kapcsolatban, hogy melyik állomás ad éppen, a 802.11 meghatároz egy csatornafigyelési módszert, amely fizikai és virtuális érzékelést is alkalmaz. A fizikai érzékelés egyszerűen megvizsgálja a közeget, hogy van-e rajta érvényes jel. A virtuális érzékeléssel minden állomás feljegyzi, hogy a csatornát mikor használják, azáltal, hogy figyelemmel kíséri a **NAV**-ot (**Network Allocation Vector – hálózatkiosztási vektor**). Minden keret hordoz egy NAV mezőt, amely megmondja, hogy az a sorozat, amelynek ez a keret is a része, milyen hosszú ideig tart még. Az állomások, amelyek ezt a keretet hallják, tudják, hogy a csatorna foglalt lesz a NAV által jelzett ideig, függetlenül attól, hogy a fizikai érzékelés mit jelzett. Például egy adatkeret NAV mezője azt az időt is tartalmazza, ami a nyugta küldéséhez szükséges. Minden állomás, amely hallja az adatkeretet, várni fog a nyugtázás ideje alatt, függetlenül attól, hogy hallhatja-e a nyugtát.

Egy opcionális RTS/CTS mechanizmus használja a NAV-ot annak megelőzésére, hogy állomások egyszerre küldjenek kereteket egymás rejtett állomásaiként. Ezt ábrázolja a 4.27. ábra. Ebben a példában *A* szeretne küldeni *B*-nek. A *C* állomás az *A* vételkörzetében van (és esetleg a *B* vételkörzetében is, de ez most nem számít). A *D* állomás a *B* vételkörzetén belül, de az *A* vételkörzetén kívül helyezkedik el.

A protokoll működése azzal kezdődik, hogy *A* úgy dönt, adatokat szeretne küldeni *B*-nek. Először *A* egy RTS-keretet küld *B*-nek, hogy engedélyét kérje egy keret elküldéséhez. Ha *B* vette ezt a kérést, akkor válaszol egy CTS-kerettel, hogy jelezze, a csatorna szabad. A CTS vétele után *A* elküldi a keretét, és elindít egy ACK-időzítőt. Az adatkeret helyes vételét követően *B* egy ACK-kerettel válaszol, ezzel befejezve az üzenetváltást. Ha az *A* ACK-időzítője lejár, mielőtt az ACK megérkezne hozzá, akkor ezt egy ütközésnek veszik, és az egész protokoll megismétlődik egy visszalépéses eljárás után.

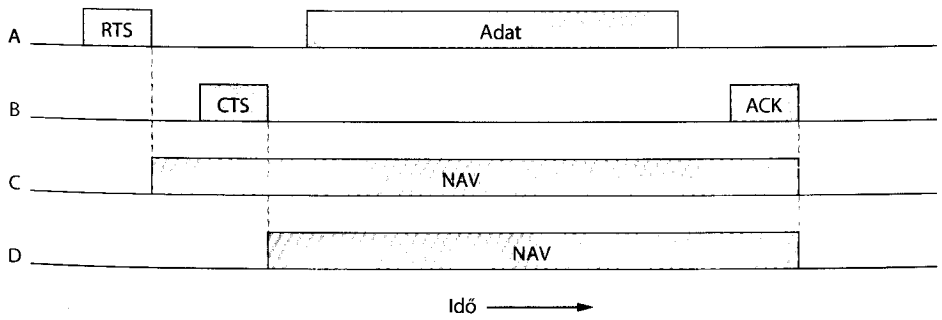
Vizsgáljuk most meg ezt az üzenetváltást a *C* és a *D* szemszögéből! A *C* az *A* vételkörzetén belül tartózkodik, tehát megkaphatja az RTS-keretet. Ha ez történik, akkor rájön, hogy valaki nemsokára adatokat fog küldeni. Az RTS-kérésben megadott információból kikövetkeztetheti, hogy meddig fog tartani az üzenetsor, beleértve a végső ACK-ot is. Így mindannyiuk érdekében eláll adási szándékától, amíg az üzenetváltás véget nem ér. Ezt azáltal éri el, hogy a NAV mezőjét frissíti, jelezve, hogy a csatorna foglalt, mint ahogy a 4.27. ábrán látható. A *D* nem hallja az RTS-t, de a CTS-t igen, tehát az is frissíti a NAV-ját. Ne felejtsük el, hogy a NAV-jeleket nem adják le: azok csupán belső emlékeztetőként szolgálnak, hogy az állomás bizonyos ideig csendben maradjon.

Annak ellenére, hogy az RTS/CTS-elméletben jónak látszik, ez azok közül az elképzelések közül való, amelyeknek a gyakorlati értéke nem túl nagy. Több ok is ismert, amiért ritkán használják. Ez nem jelent segítséget a rövid kereteknek (amelyeket az RTS-ek helyett küldenek), sem az AP-nek (amit mindenki hall, definíció szerint). Más esetekben csak lelassítja a működést. A 802.11 RTS/CTS-protokollja csak kissé más, mint a 4.2. szakaszban bemutatott MACA, mert mindenki, aki hallja az RTS-t vagy a CTS-t, csendben marad egy ideig, hogy a nyugta átjusson ütközés nélkül. Ezért ez nem segít a megvilágított állomások esetén, ellentétben a MACA-val, csak a rejtett állomások esetén. Leggyakrabban kevés rejtett állomás van, és a CSMA/CA már segít rajtuk azzal, hogy a bármilyen okból sikertelenül küldő állomásokat lelassítja azért, hogy nagyobb eséllyel sikerüljön adniuk.

A CSMA/CA fizikai és virtuális érzékeléssel kiegészítve adja a 802.11 protokoll magját. Van viszont néhány más mechanizmus, amit a 802.11-hez fejlesztettek ki. Mindegyiket a valós működés szükségletei kényszerítették ki, ezért röviden megnézzük ezeket.

Az első szükséglet a megbízhatóság. A vezetékes hálózatokkal ellentétben, a vezeték nélküli hálózatok zajosak és megbízhatatlanok, nem kis részben az olyan más eszközök által keltett interferencia miatt, mint amilyenek például a mikrohullámú sütők, amelyek ugyanúgy az engedélymentes ISM-sávot használják. A nyugták és újraküldések kis segítséget nyújtanak akkor, amikor egy keret elsőre történő átvitelének kicsi a valószínűsége.

A fő stratégia, amelyet a sikeres küldések számának növelésére használnak az, hogy csökkenteni kell az átviteli sebességet. A kisebb sebességek robusztusabb modulációt használnak, amely mellett nagyobb a valószínűsége annak, hogy adatok hiba nélkül érkezzenek meg egy adott jel/zaj arány mellett. Ha túl sok keret veszik el, az állomás



4.27. ábra. *A virtuális csatornaérzékelés használata CSMA/CA-val*

csökkentheti a sebességet. Ha a keretek kis veszteséggel továbbítódnak, akkor az állomás esetenként megpróbálja ellenőrizni, hogy egy nagyobb sebesség használható-e.

Egy keret sértetlen érkezésének esélye növelhető egy másik stratégiával: rövidebb keretek küldésével. Ha bármelyik bit meghibásodásának valószínűsége p, akkor egy n bites keret helyes vételének valószínűsége $(1-p)^n$. Ha például $p = 10^{-4}$, akkor egy teljes Ethernet-keret (12 144 bit) helyes vételének valószínűsége kisebb mint 30%. A legtöbb keret el fog veszni. De ha a keretek hossza csak a harmada (4048 bit), akkor kétharmaduk helyesen érkezik meg. Így a legtöbb keret átjut és kevesebb újraküldésre lesz szükség.

Rövidebb kereteket létre lehet hozni a hálózati rétegtől elfogadott legnagyobb csomagméret csökkentésével. Másként, a 802.11 lehetővé teszi a keretek kisebb **részekre, darabokra** (**fragment**) történő szabdalását, melyek közül mindegyik saját ellenőrző összeggel rendelkezik. A szabvány nem rögzíti a darabok méretét, de a hozzáférési pont paramétereként állítható. A részeket egyenként számozzák és nyugtázzák egy megállás-vár protokollal (azaz a küldő nem küldheti el a $k+1$-edik részt, amíg a k. részre meg nem érkezett a nyugta). Ha egy állomás megszerezte a csatornát, akkor több darabot tud egy lökетben küldeni. Ezek egymás után mennek, közöttük egy-egy nyugtával (és valószínűleg újraküldésekkel), amíg vagy az egész keret sikeresen megérkezik, vagy az átviteli próbálkozások száma eléri a megengedett maximumot. A NAV-mechanizmus csak a következő nyugtáig biztosítja a többi állomás csendben maradását, de egy másik mechanizmust (lásd lejjebb) használnak arra, hogy részek lökete elküldhető legyen anélkül, hogy más állomások közben keretet küldenének.

A második szükséglet, amit megvizsgálunk, az energiatakarékosság. Az akkumulátorok élettartama mindig is gondot jelent a vezeték nélküli eszközöknél. A 802.11 szabvány figyelmet szentel a teljesítménygazdálkodásnak, ezáltal a kliensek nem pazarolják a teljesítményt, amikor nincs olyan adatuk, amit adni vagy venni kellene.

Az alapvető energiatakarékossági mechanizmus a **jelzőfénykeretekre** (**beacon frame**) épül. A jelzőfénykereteket a hozzáférési pont periodikusan üzenetszórja (például másodpercenként 10 alkalommal). Ezek a keretek hirdetik a hozzáférési pont jelenlétét a klienseknek, és olyan rendszer paramétereket szállítanak, mint amilyen a hozzáférési pont azonosítója, a pontos idő, a következő jelzőfénykeretig tartó idő és biztonsági beállítások.

A kliensek beállíthatnak egy teljesítménygazdálkodás bitet a hozzáférési pontnak küldendő keretekben, hogy értesítsék arról, hogy **energiatakarékos módba** (**power save mode**) lépnek. Ebben az állapotban a kliens egy kicsit elszundíthat, és a hozzáférési pont fogja pufferelni a neki szánt forgalmat. A kliens, hogy ellenőrizze, van-e neki szánt forgalom, minden jelzőfénykeretre felébred és megvizsgálja a forgalmi térképet, amit a jelzőfénykeret részeként megkap. Ez a térkép tájékoztatja a klienst arról, hogy van-e pufferelt forgalma. Ha igen, akkor a kliens küld egy lekérő üzenetet a hozzáférési pontnak, ami ezután küldi a pufferelt forgalmat. A kliens ezután visszamehet aludni a következő jelzőfénykeretig.

Egy másik energiatakarékossági mechanizmust, amit **ASPD**-nek (**Automatic Power Save Delivery – automatikus energiatakarékos kézbesítés**) neveznek, 2005-ben csatolták a 802.11-hez. Ezzel az új mechanizmussal a hozzáférési pont puffereli a kereteket és küldi a kliensnek éppen az után, hogy a kliens egy keretet küldött a hozzáférési pontnak. A kliens ezután elmehet aludni egészen addig, amíg nem lesz újabb küldendő (vagy

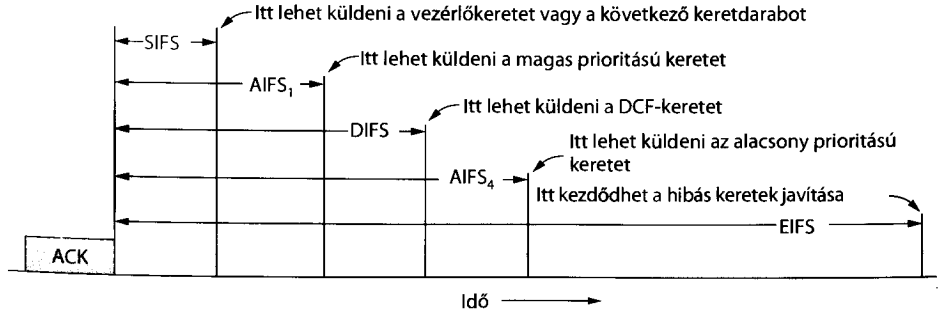
veendő) forgalma. Ez a mechanizmus jól működik a VoIP-hoz hasonló alkalmazásoknál, amelyeknél sűrűn küldenek kereteket mindkét irányba. Például egy VoIP vezeték nélküli telefon használhatja ezt a mechanizmust, hogy 20 ms-onként küldjön és vegyen kereteket, és közben aludjon. Ez a sűrűség jóval gyakoribb a jelzőfénykeretek 100 ms-os periódusánál.

A harmadik és egyben utolsó szükséglet, amelyet megvizsgálunk, a szolgáltatásminőség. Ha az előző példában a VoIP-forgalom P2P-forgalommal verseng, akkor a VoIP-forgalom fogja húzni a rövidebbet. A VoIP-forgalom késleltetést szenved a nagy sávszélességű P2P-forgalommal szemben, annak ellenére, hogy a VoIP sávszélessége kicsi. Ezek a késleltetések valószínűleg rontják a beszédhívás minőségét. A minőségromlás elkerülésére szeretnénk, hogy a VoIP elsőbbséget élvezzen a P2P-forgalommal szemben azáltal, hogy nagyobb a prioritása.

IEEE 802.11-nek van egy ügyes mechanizmusa, amelyet 2005-ben 802.11e név alatt kiegészítésként vezettek be annak érdekében, hogy lehetővé tegye ezt a fajta szolgáltatásminőséget. Úgy működik, hogy a CSMA/CA-t kiterjesztették gondosan definiált keretek közötti intervallumokkal. Egy keret elküldése után, mielőtt bármelyik állomás elküldhetne egy keretet, egy bizonyos tétlen időtartam van előírva annak érdekében, hogy ellenőrizzék, hogy a csatornát a továbbiakban senki nem használja. A trükk az, hogy különböző időintervallumokat definiáltak a különböző típusú keretek számára.

Öt időintervallumot mutat be a 4.28. ábra. A szabályos adatkeretek közötti intervallumot **DIFS**-nek (**DCF InterFrame Spacing – DCF-keretek közti időköz**) nevezik. Bármelyik állomás megkísérelheti megszerezni a csatornát, hogy egy új keretet küldjön, miután a közeg DIFS ideig tétlen volt. A szokásos versengési szabály érvényes és a kettes exponenciális visszalépés algoritmust szükséges alkalmazni ütközés esetén. A legrövidebb intervallum a **SIFS** (**Short InterFrame Spacing – rövid keretek közti időköz**). Ezt arra használják, hogy az egyik fél egy üzenetváltás során elsőnek kezdjen adni. Például lehetővé teszi, hogy a vevő nyugtát vagy más vezérlőkeret-sorozatokat küldjön, mint amilyen az RTS és a CTS, vagy az adó egy keretdarabokat tartalmazó löketet elküldhessen. Azzal, hogy csak SIFS intervallumnyi időt vár a következő rész küldése előtt, azt biztosítja, hogy más állomás egy keretével ne ugorhasson az üzenetváltás közepébe.

A két **AIFS**- (**Arbitration InterFrame Space – döntési keretek közti időköz**) intervallum két különböző prioritási szintre ad példát. A rövid intervallum, az $AIFS_1$, rövi-



4.28. ábra. *A keretek közti idő felosztása a 802.11-ben*

debb a DIFS-nél, de hosszabb a SIFS-nél. Ezt arra használhatja a hozzáférési pont, hogy a beszéd- és más nagy prioritású forgalmat a várakozási sor elejére hozzon. A hozzáférési pont rövidebb ideig fog várni a beszédforgalom küldése előtt, és emiatt azt a szabályos forgalom előtt küldi el. A hosszú intervallum, az $AIFS_4$ nagyobb mint a DIFS. Ezt háttérforgalomra lehet használni, amit lehet halogatni, amíg van szabályos forgalom. A hozzáférési pont hosszabb intervallumot fog várni, mielőtt elküldené a kereteit, amivel megadja a szabályos forgalomnak az előbb adás lehetőségét. A teljes szolgáltatásminőségi mechanizmus négy különböző prioritási szintet határoz meg, amelyeknek különböző visszalépési paraméterük, valamint különböző tétlenségi paraméterük van.

Az utolsó időközt, az **EIFS**-t (**Extended InterFrame Spacing – kiterjesztett keretek közti időköz**) csak olyan állomások használják, melyek épp egy hibás vagy ismeretlen keretet vettek, és a problémát kívánják jelezni. Ez azért történik, mert elképzelhető, hogy a vevőnek fogalma sincs arról, hogy mi történik, ebben az esetben pedig egy ideig célszerű várakoznia, hogy ne zavarjon meg egy másik, két állomás között folyó párbeszédet.

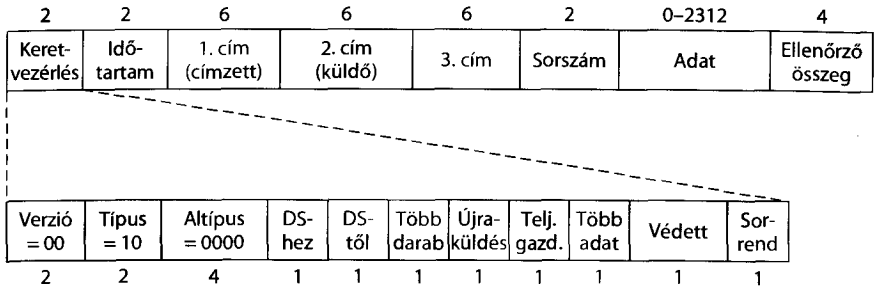
A szolgáltatásminőségi kiegészítések egy további része a **TXOP** (**Transmission Opportunity – átviteli lehetőség**). Az eredeti CSMA/CA mechanizmus az állomásoknak csak egy keret küldését engedélyezi egy adott időpontban. Ez a működés jó volt, amíg a sebességek meg nem emelkedtek. A 802.11a/g esetén előfordulhat, hogy az egyik állomás 6 Mb/s, míg a másik állomás 54 Mb/s sebességgel ad. Mindkettő egy keretet küldhet, de a 6 Mb/s sebességű állomásnak kilencszer annyi ideig tart (a rögzített méretű többletbiteket leszámítva) egy keret elküldése, mint az 54 Mb/s-os sebességű állomásnak. Ennek az egyenlőtlenségnek a szerencsétlen mellékhatása az, hogy a lassú adóval versengő gyors adó lelassul, nagyjából a lassú adó sebességére. Például, a rögzített méretű többletbitek elhanyagolásával, amikor a 6 Mb/s-os és az 54 Mb/s-os adók egyedül adnak, akkor megtartják a saját sebességüket, de amikor együtt adnak, akkor mindketten átlagban 5,4 Mb/s-ot kapnak. Ez komoly büntetés a gyors adó számára. Ezt a problémát **sebességanomáliának** (**rate anomaly**) nevezik [Heusse és mások, 2003].

A TXOP-val minden állomás ugyanannyi adásidőt kap, és nem ugyanannyi elküldhető keretet. Azok az állomások, amelyek gyorsabban adnak, az adásidejükben nagyobb átviteli sebességet tudnak elérni. A példánkban, amikor a két állomás egyszerre ad 6 Mb/s-os és 54 Mb/s-os sebességgel, az első adó 3 Mb/s-ot, a második adó 27 Mb/s-ot fog kapni.

4.4.4. A 802.11 keretszerkezete

A 802.11 szabvány három különböző keretosztályt definiál: adat-, vezérlő- és menedzsmentkereteket. Mindegyiknek saját fejrésze van, különböző mezőkkel, melyeket a MAC-alrétegben használnak. Ezenfelül vannak olyan fejrészek is, melyeket a fizikai réteg használ, de ezek többnyire az alkalmazott modulációs eljárással foglalkoznak, így ezeket most nem tárgyaljuk.

Példaként az adatkeret felépítését fogjuk megvizsgálni, amelyet a 4.29. ábra mutat. Az első mező a *Keretvezérlés (Frame Control)*, amelyet 11 almező alkot. Ezek közül az első a *Protokollverzió (Protocol Version)*, amely 00-ra van állítva. Ez lehetővé teszi, hogy a 802.11 protokollnak újabb változatai működhessenek egyszerre ugyanabban a cellában. Ezt követik a *Típus (Type)* (adat-, vezérlés- vagy menedzsment-) és az *Altípus (Subtype)*



4.29. ábra. *A 802.11 keretszerkezete*

mezők (például RTS vagy CTS). Egy szabályos adatkeret esetében (szolgáltatásminőség nélkül) bináris 10 és 0000 értékűek. A *DS-hez (To DS)* és a *DS-től (From DS)* biteket úgy állítják be, hogy jelezzék, hogy a keret a hozzáférési ponthoz csatlakozó, elosztórendszernek (distribution system) nevezett hálózat felé tart, vagy onnan érkezett. A *Több darab (More fragments)* bit azt jelenti, hogy még további darabok következnek. Az *Újraküldés (Retry)* bit egy előzőleg már elküldött keret újraküldését jelzi. A *Teljesítménygazdálkodás (Power management)* bit jelentése, hogy a küldő készenléti állapotba fog lépni. A *Több adat (More data)* bit arra utal, hogy az adónak további keretei vannak a vevő számára. A *Védett keret (Protected Frame)* bit azt adja meg, hogy a keret törzsét biztonsági okból titkosították. A következő szakaszban röviden megvizsgáljuk a biztonság kérdését. Végül a *Sorrend (Order)* bit azt közli a vevővel, hogy a felső réteg az érkező keretek sorozatát szigorúan sorrendben várja.

Az adatkeret második mezője az *Időtartam (Duration)* mező, amely azt adja meg mikroszekundumokban, hogy a keret és a hozzá tartozó nyugta mennyi ideig fogja lefoglalni a csatornát. Ez a mező mindenfajta keretben jelen van, így a vezérlőkeretek is tartalmazzák, és ez az, amelyet az állomások a NAV-mechanizmus vezérlésére használnak.

Ezután jönnek a címek. A hozzáférési pontnak küldött vagy tőle fogadott adatkeret három címmel rendelkezik, amelyek közül mindegyik szabványos IEEE 802 formátumú. Az első a forrás-, a második a célcím, amelyekre nyilván szükség van. De mire szolgál a harmadik cím? Ne feledjük, hogy a hozzáférési pont a keretek számára csak egy egyszerű átjátszóállomás, amint egy kliens és a hálózat egy másik pontja között utaznak, amely talán éppen egy távoli kliens vagy egy kapu az internet felé. A harmadik cím pontosan ezt a távoli végpontot adja meg.

A *Sorszám (Sequence)* mező sorszámozza a kereteket, így a duplikátumokat lehet detektálni. A 16 rendelkezésre álló bitből 4 a keretet, 12 pedig a keretrészt azonosítja, amely minden új átvitellel növelődik. Az *Adat (Data)* mező tartalmazza a legfeljebb 2312 bájtnyi felhasználói adatot. A felhasználói adat első bájtjai az **LLC**-ként (**Logical Link Control – logikai kapcsolatvezérlés**) ismertek. Ez a réteg képezi a ragasztót, amely a felső rétegbeli protokollt azonosítja (például IP), amihez a felhasználói adatot továbbítani kell. Végül a *Keretellenőrző összeg (Frame Check Sequence)* következik, ami megegyezik a többek között a 3.2.2. szakaszban is bemutatott 32 bites CRC-vel.

A menedzsmentkeretek felépítése ugyanaz, mint az adatkereteké, kiegészítve egy adatrészlettel, ami az *Altípustól* függően változik (például a jelzőfénykeretek paramété-

rei). A vezérlőkeretek rövidek. Mint minden keret, ezek is rendelkeznek egy *Keretvezérlés*, egy *Időtartam* és egy *Keretellenőrző összeg* mezővel. Viszont csak egyetlen címmezőjük van, és nincs felhasználói adatuk. A legfontosabb információk nagy részét az *Altípus (Subtpye)* mező hordozza (például ACK, RTS és CTS).

4.4.5. Szolgáltatások

A 802.11 szabvány meghatározza a szolgáltatásokat, hogy a kliensek, a hozzáférési pontok és a hozzájuk kapcsolódó hálózat az előírásoknak megfelelő vezeték nélküli LAN legyen. Ezek a szolgáltatások számos csoportba sorolhatók.

A **kapcsolódás (association)** szolgáltatást mozgó állomások használják, hogy csatlakozzanak a hozzáférési ponthoz. Erre rendszerint azt követően kerül sor, hogy egy állomás belépett a hozzáférési pont adáskörzetébe. Az állomás megérkezése után megtudja a hozzáférési pont azonosítóját és képességeit, vagy a jelzőfénykeretekből, vagy közvetlenül a hozzáférési pont megkérdezésével történhet. Ezek a képességek magukba foglalják a támogatott sebességeket, a biztonsági beállításokat, a teljesítménygazdálkodási képességeket, a szolgáltatásminőség támogatását és másokat. Az állomás kapcsolódási kérést küld a hozzáférési pontnak. A hozzáférési pont elfogadhatja, vagy elutasíthatja a kérést.

Az **újrakapcsolódás (reassociation)** lehetővé teszi, hogy az állomások kiválasszák az általuk előnyben részesített hozzáférési pontot. Ez a lehetőség egy kiterjedt 802.11 LAN-nak az egyik hozzáférési pontjától a másikig haladó állomások számára hasznos, mint az átadás (handover) a mobiltelefon-hálózatok esetén. Ha helyesen használják, akkor a cellaváltás során nem vesznek el adatok. (Persze az Ethernethez hasonlóan a 802.11 sem képes garanciákat nyújtani.) A **szétkapcsolást (disassocation)**, azaz a kapcsolat bontását mind az állomás, mind pedig a hozzáférési pont kezdeményezheti. Az állomásnak használnia kell ezt a szolgáltatást a kikapcsolása vagy a hálózatból való távozása előtt. A hozzáférési pont használhatja a karbantartás miatti lekapcsolást megelőzően.

Az állomásoknak **hitelesíteniük (authentication)** kell magukat, mielőtt a hozzáférési ponton keresztül kereteket küldenek. De a választott biztonsági sémától függően a hitelesítést különböző módokon kezelik. Ha egy 802.11 hálózat nyílt, akkor mindenki használhatja. Ellenkező esetben, az állomásnak igazolnia kell magát a hitelesítéshez. Az ajánlott séma a **WPA2 (Wi-Fi Protected Access 2 – Wi-Fi védett hozzáféréssel 2)**, amely a 802.11i szabványt valósítja meg. (Az alap WPA egy köztes séma, amely a 802.11-nek csak egy részét valósítja meg. Ezt átugorjuk, és rögtön a teljes sémát tárgyaljuk.) A WPA2 esetén a hozzáférési pont egy felhasználónév- és jelszóadatbázissal rendelkező hitelesítő szerverrel kommunikálhat, hogy megállapítsa, vajon az állomás használhatja-e a hálózatot. Alternatívaként egy előre megosztott kulcsot lehet beállítani, ami a hálózati jelszó szépen csengő neve. Több keret cserélődik ki az állomás és a hozzáférési pont között, kihívással és válasszal, hogy az állomás bebizonyítsa, hogy rendelkezik a megfelelő jogosultságokkal. Ez az üzenetváltás azonnal a kapcsolódás után történik.

A WPA előtt használt sémát **WEP**-nek (**Wired Equivalent Privacy – vezetékessel egyenértékű titkosság**) nevezik. Ebben a sémában, az előre megosztott kulccsal történő hitelesítés a kapcsolódás előtt történik. Ennek ellenére nem javasolt a használata a tervezési hibái miatt, amelyek a WEP-et könnyen feltörhetővé teszik. Ennek az első gyakorlati bemutatására akkor került sor, amikor Adam Stubblefield nyári gyakornok volt az AT&T-nél [Stubblefield és mások, 2002]. Egy hét alatt megírta és tesztelte a támadást, pedig az idő nagy része azzal telt, hogy a vezetőségtől engedélyt szerezzen a kísérletezéshez szükséges Wi-Fi-kártya vásárlására. A WEP feltöréséhez szükséges szoftverek ingyen beszerezhetők.

Amint keretek érkeznek a hozzáférési ponthoz, az **elosztás (distribution)** szolgáltatás határozza meg azt, hogyan kell a beérkező keretek forgalomirányítását elvégezni. Ha a címzett állomás a hozzáférési pont körzetében van, akkor a kereteket közvetlenül a rádiós összeköttetésen keresztül lehet kiküldeni; egyébként a vezetékes hálózaton kell továbbítani. Az **integráció (integration)** szolgáltatás kezeli a leképezést, ami akkor szükséges, ha egy keret a 802.11 LAN-ról ki akar menni, vagy egy másik hálózatról be akar jönni. A leggyakoribb eset, amikor a vezeték nélküli hálózat az internetre csatlakozik.

Az egész történet az adatátvitelről szól, tehát a 802.11 természetesen nyújt **adatkézbesítő (data delivery) szolgáltatást**. Ez a szolgáltatás lehetővé teszi, hogy az állomások adatot küldjenek és fogadjanak az ebben a fejezetben korábban leírt protokollokkal. Mivel azonban a 802.11 az Ethernet mintájára készült, és az átvitel az Ethernet esetében sem 100%-ig megbízható, ezért erre a 802.11 sem vállal garanciát. A hibák észlelésének és javításának terhe tehát a felsőbb rétegekre hárul.

A vezeték nélküli jelek üzenetszóró jellegűek. Azért, hogy a vezeték nélküli LAN-on küldött információ bizalmas maradjon, titkosítani kell. **Ezt a titoktartás (privacy)** szolgáltatással érik el, ami a titkosítás és megfejtés részleteit kezeli. A WPA2 titkosítási algoritmusa az **AES**-en (**Advanced Encryption Standard – fejlett titkosító szabvány**) alapul, amit az Egyesült Államokban 2002-ben fogadtak el. A titkosításhoz használt kulcsokat a hitelesítés alatt állapítják meg.

A különböző prioritású forgalom kezeléséhez **QoS-forgalomütemező (QoS traffic scheduling)** szolgáltatás áll rendelékezésre. A korábban bemutatott protokollt használja, hogy a beszéd- és videoforgalmat különleges bánásmódban részesítse a best-effort és a háttérforgalomhoz képest. Egy ezzel társított szolgáltatás szintén nyújt magasabb rétegbeli időzítő szinkronizálást. Ez lehetővé teszi az állomásoknak, hogy a tevékenységeiket koordinálják, ami hasznos lehet médiafeldolgozásnál.

Végezetül van még két szolgáltatás, amelyek a spektrumhasználat menedzselésében segítenek az állomásoknak. Az **adásiteljesítmény-szabályozó (transmit power control)** szolgáltatás ellátja az állomást információval a régióról régióra változó adási teljesítmény előírt korlátjairól. A **dinamikus frekvenciaválasztó (dynamic frequency selection)** szolgáltatás információval látja el az állomásokat, hogy el tudják kerülni az 5 GHz-es sávhoz tartozó frekvenciákon történő adást, amit egy közeli radar éppen használ.

A 802.11 szabvány ezekkel a szolgáltatásokkal rengeteg funkciót nyújt a közeli mobil klienseknek az internethez való csatlakoztatására. A 802.11 szabvány hatalmas siker, és folyamatosan újabb funkciókkal bővítik. A szabvány perspektíváiról és jövőjéről lásd Hiertz és mások munkáját [2010].

4.5. Széles sávú vezeték nélküli hálózatok

Eleget voltunk már a négy fal között. Itt az ideje, hogy kimenjünk a szabadba, ahol elég sok érdekes hálózat van az úgynevezett „utolsó kilométeren". A távbeszélőrendszer liberalizációja után sok országban a korábban monopolhelyzetben lévő szolgáltatók versenytársai is engedélyt kaptak helyi beszédcélú szolgáltatások és nagy sebességű internetszolgáltatások nyújtására. A kereslet kétségkívül jelentős. A gond csak az, hogy egy fényvezető szálat vagy koaxiális kábelt megfizethetetlenül drága lenne elvezetni több millió háztartáshoz. Mit tehet ilyenkor egy új piaci szereplő?

A megoldást a széles sávú vezeték nélküli hálózatok jelentik. Sokkal egyszerűbb és olcsóbb felállítani egy nagy antennát egy városszéli dombon, mint árkokat ásni és kábeleket fektetni. A távközlési társaságok tehát kísérletezni kezdtek annak érdekében, hogy több megabites vezeték nélküli beszéd-, internet-, hálózati video- stb. szolgáltatást nyújtsanak.

Az IEEE, hogy a piacot fellendítse, megalakította a széles sávú vezeték nélküli nagyvárosi hálózatok szabványosításával foglalkozó csoportot. A következő szabad szám a 802-es csoportban a **802.16** volt, így a szabvány ezt a számot kapta. Informálisan ezt a technológiát **WiMAX**-nak (**Worldwide Interoperability for Microwave Access – világméretű együttműködés a mikrohullámú hozzáférésért**) nevezik. Mi a 802.16-ot és a WiMAX-ot szinonimaként használjuk.

Az első 802.16 szabványt 2001 decemberében fogadták el. A korai változatok lehetővé tették vezeték nélküli helyi hurkok (wireless local loop) kialakítását rögzített pontok között, amelyek rálátnak egymásra. Ezt a kialakítást nem sokkal később megváltoztatták, hogy a kábel- és DSL-alapú internet-hozzáférés versenyképes alternatívájává váljon. 2003 januárjára a 802.16-ot újragondolták, és hogy ne igényelje az egymásra rálátást, a 2 GHz és 10 GHz közötti frekvenciákon OFDM-technikát használtak. Ez a változtatás sokkal egyszerűbbé tette az alkalmazását, de még mindig csak rögzített helyszínen volt használható. A 3G mobiltelefon-hálózatok megjelenése veszélyeztette a WiMAX-ot azzal, hogy nagy sebességet *és* mobilitást ígért. Válaszul 2005 decemberére a 802.16-ot ismét továbbfejlesztették úgy, hogy járművi sebességű mobilitást biztosítson. A jelenlegi IEEE 802.16-2009 szabvány célja az, hogy mobil széles sávú internet-hozzáférést nyújtson.

A többi 802-es szabványhoz hasonlóan a 802.16-ra is nagy hatással volt az OSI-modell, annak (al)rétegeivel, terminológiájával, szolgáltatási primitívjeivel és egyéb részeivel együtt. Sajnos az OSI-hoz hasonlóan ez a szabvány is elég bonyolult. Valójában a **WiMAX Fórumot** azért hozták létre, hogy határozza meg a szabvány együttműködésre képes részhalmazait annak érdekében, hogy a WiMAX piacképessé váljon. A következő szakaszokban rövid leírást adunk a 802.16 vezeték nélküli interfészének néhány legfontosabb tulajdonságáról, de leírásunk közel sem lesz teljes és sok részletre nem tér ki. A WiMAX-ról és a széles sávú vezeték nélküli kommunikáció általános kérdéseiről Andrews és mások [2007] művéből tájékozódhatunk.

4.5.1. A 802.16 összehasonlítása a 802.11-gyel és a 3G-vel

Ezen a ponton az olvasó azt kérdezheti magától: mi szükség van egy újabb szabványra. Miért nem elég a 802.11 vagy a 3G? Nos, a WiMAX a 802.11 és a 3G ötleteit egyesíti, és ez hasonlatossá teszi a 4G-technikához.

A 802.11-hez hasonlóan a WiMAX az eszközök internethez való vezeték nélküli megabit/másodperces sebességű csatlakoztatására készült, a kábeltévé és a DSL alternatívájaként. Az eszközök lehetnek mobilak vagy legalábbis hordozhatók. A WiMAX nem úgy indult, hogy kis sebességű adatforgalmat tettek a beszédátvitelre kialakított mobiltelefon-hálózatokra. A 802.16-ot úgy tervezték, hogy IP-csomagokat szállítson vezeték nélkül és hogy IP-alapú vezetékes hálózatokhoz a lehető legkisebb vesződséggel csatlakoztatható legyen. A csomagok, hogy az alkalmazások széles palettáját támogassák, szállíthatnak P2P-forgalmat, VoIP-hívásokat vagy média-adatfolyamot. Mint a 802.11, ez a szabvány is egyrészt az OFDM-technikán alapul, hogy jó legyen a teljesítőképessége az olyan vezeték nélküli jelromlás ellenére, mint amilyen a többutas jelgyengülés, másrészt a MIMO-technikán alapul, hogy nagy legyen az átbocsátóképessége.

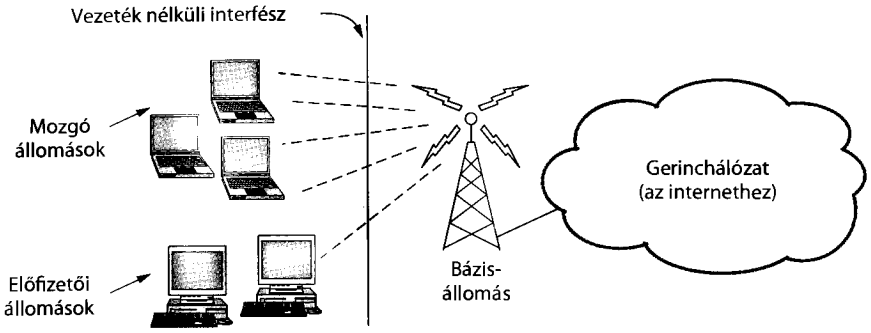
A WiMAX számos kulcsfontosságú területen nagyon hasonlít a 3G-re (és ezért nem hasonlít a 802.11-re). A legfontosabb műszaki probléma az, hogyan érhető el nagy kapacitás a spektrum hatékony kihasználásával úgy, hogy a lefedettségi területen sok előfizető kapjon nagy sávszélességet. A tipikus távolságok legalább 10-szer nagyobbak, mint egy 802.11-es hálózat esetén. Ennek következtében, a WiMAX-bázisállomások nagyobb teljesítményűek, mint a 802.11-es hozzáférési pontok (AP-k). A nagyobb távolságokra érkező gyengébb jelek kezelésére a bázisállomás nagyobb teljesítményt és jobb antennákat használ, és több adatfeldolgozást végez hibák kezelésére. Annak érdekében, hogy maximalizálja az átbocsátóképességet, egyrészt a bázisállomás az egyes felhasználóknak szóló átviteleket gondosan ütemezi, másrészt nem engedi a spektrumot a CSMA/CA-val használni, mert az az ütközésekkel pazarolhatja a kapacitást.

A WiMAX számára az engedélyhez kötött frekvenciasávok használata a legvalószínűbb eset, az USA-ban ez tipikusan a 2,5 GHz körül van. Az egész rendszert lényegesen jobban optimalizálták, mint a 802.11-et. Ez a komplexitás megérte az árát, figyelembe véve azt is, hogy engedélyhez kötött spektrumok használata nagyon drága. A 802.11-től eltérően az eredmény egy menedzselt és megbízható szolgáltatás jó szolgáltatásminőséggel megtámogatva.

Ezekkel a képességekkel, a 802.16 leginkább a 4G mobiltelefon-hálózatokhoz hasonlít, amelyek szabványosítása most van folyamatban **LTE (Long Term Evolution – hosszú távú fejlődés)** néven. Amíg a 3G mobiltelefon-hálózatok CDMA-ra épülnek, és beszéd- és adatátvitelt támogatnak, addig a 4G mobiltelefon-hálózatok a MIMO-n és az OFDM-en alapulnak, és az adatátvitelt célozzák meg úgy, hogy a beszédátvitel csak egy alkalmazás lesz a sok közül. Úgy tűnik, hogy a WiMAX és a 4G egymással ütköző pályán mozog, ha a technikát és az alkalmazási területet nézzük. Talán ez a konvergencia nem meglepő, hiszen az internet a legütősebb alkalmazás, az OFDM és a MIMO pedig a legismertebb technikák a spektrum hatékony használatára.

4.5.2. A 802.16 felépítése és protokollkészlete

A 802.16 felépítését a 4.30. ábra mutatja. A bázisállomások közvetlenül a szolgáltatói gerinchálózatra kapcsolódnak, ami viszont az internethez kapcsolódik. A bázisállomások az állomásokkal vezeték nélküli közegen keresztül kommunikálnak. Kétféle állomás létezik. Az előfizetői állomások rögzített helyen maradnak, például otthoni széles sávú



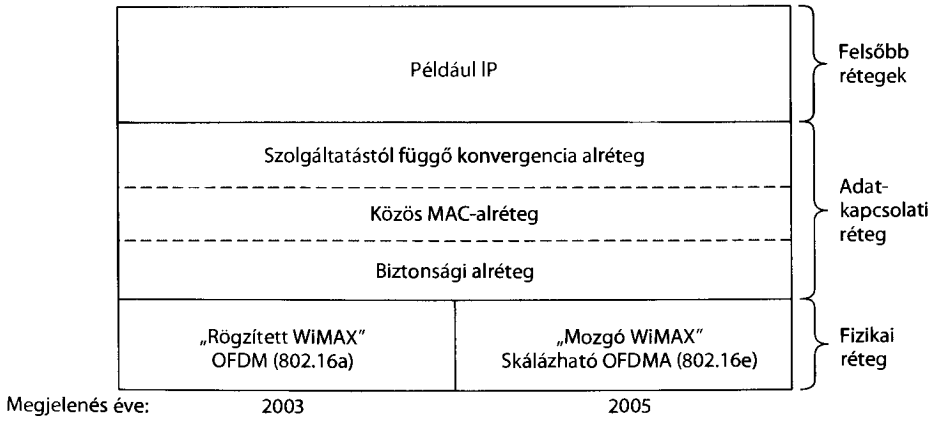
4.30. ábra. *A 802.16 felépítése*

internet-hozzáférés esetében. A mozgó állomások a helyváltoztatás közben is igénybe tudják venni a szolgáltatást, például egy WiMAX-eszközzel felszerelt autó esetén.

A 802.16 vezeték nélküli interfészen keresztül használt protokollkészletét a 4.31. ábra szemlélteti. Az általános szerkezete hasonló a 802 hálózatokéhoz, de több alrétege van. A legalsó réteg foglalkozik az átvitellel, amelyben csak a népszerű 802.16-os felhasználásokat ábrázoljuk, a rögzített és mozgó WiMAX-ot. Minden felhasználási formának különböző fizikai rétege van. Mindkét fizikai réteg – a 11 GHz frekvenciasáv alatt – engedélyhez kötött spektrumban működik, és mindkettő OFDM-et használ, de különböző módon.

A fizikai réteg feletti adatkapcsolati réteg három alrétegből áll. A legalsó a titoktartással (privacy) és a biztonsággal foglalkozik, ami a nyilvános szabadtéri hálózatokban még inkább kritikus tényező, mint a beltéri magánhálózatokban. Ez az alréteg felelős a titkosításért és a visszafejtésért, valamint a kulcsok kezeléséért.

A következő a közös MAC-alréteg. Ebben a részben helyezkednek el a legfontosabb protokollok, mint például a csatornakezelés. E modell szerint a bázisállomás irányítja a rendszert teljes egészében. Nagyon hatékonyan képes ütemezni a lefelé irányuló (azaz a bázisállomástól az előfizető felé haladó) csatornákat, és központi szerepet játszik a felfelé



4.31. ábra. *A 802.16 protokollkészlete*

irányuló (az előfizetőtől a bázisállomás felé haladó) csatornák kezelésében is. Ennek a MAC-alrétegnek a többi 802-es protokollal ellentétben van egy szokatlan tulajdonsága is, nevezetesen hogy teljesen összeköttetés-alapú, mégpedig azért, hogy megfelelő szolgáltatásminőségi garanciákat nyújtson a telefon- és a multimédia-alkalmazások részére.

A szolgáltatástól függő konvergencia-alréteg a többi 802-es protokoll logikai kapcsolatvezérlésért (LLC) felelős alrétegét hivatott helyettesíteni. Az a feladata, hogy csatlakozási felületet nyújtson a hálózati réteg számára. Ebben a szabványban különböző konvergencia-alrétegeket határoztak meg, hogy a 802.16 rendszer problémamentesen integrálható legyen különböző felsőbb rétegekkel. A leglényegesebb felsőbb csatlakozási lehetőség az IP-protokollhoz történő csatlakozás, de a szabvány csatlakozási lehetőséget határoz meg olyan protokollokra is, mint amilyen az Ethernet és az ATM. Mivel az IP összeköttetés nélküli szolgáltatást nyújt, a 802.16 MAC-alréteg pedig összeköttetés-alapú, ezért ennek az alrétegnek a feladata a címek és összeköttetések közötti leképezés.

4.5.3. A 802.16 fizikai rétege

A legtöbb WiMAX megvalósítás engedélyköteles spektrumot használ vagy a 3,5 GHz-es, vagy a 2,5 GHz-es frekvencia körül. Ahogy a 3G mobiltelefon-hálózatoknál, itt is egy elérhető spektrum meglelése a legnagyobb probléma. Ezért a 802.16 szabványt rugalmasra tervezték: 2 GHz-től 11 GHz-ig teszi lehetővé a működtetését. Különböző méretű csatornákat támogat, például 3,5 MHz-et a rögzített WiMAX-ra, és 1,25 MHz-től 20 MHz-ig mozgó WiMAX-ra.

Ezeken a csatornákon a 2.5.3. szakaszban bemutatott OFDM-módszert használják az átvitelre. A 802.16 protokoll OFDM-működését arra optimalizálták (ellentétben a 802.11-gyel), hogy a lehető legtöbbet hozzák ki az engedélyköteles spektrumból és a nagy távolságú átvitelből. A csatornát több alvivőre osztották, amelyek hosszabb jeltartási idővel rendelkeznek, hogy ellentételezzék a nagyobb vezeték nélküli jelalakromlást. A WiMAX paraméterei nagyjából 20-szor nagyobbak a 802.11 ide vonatkozó paramétereinél. Például, a mozgó WiMAX esetén 512 alvivő van egy 5 MHz-es csatornára és az egy szimbólum küldésére szánt idő minden alvivőn nagyjából 100 μs.

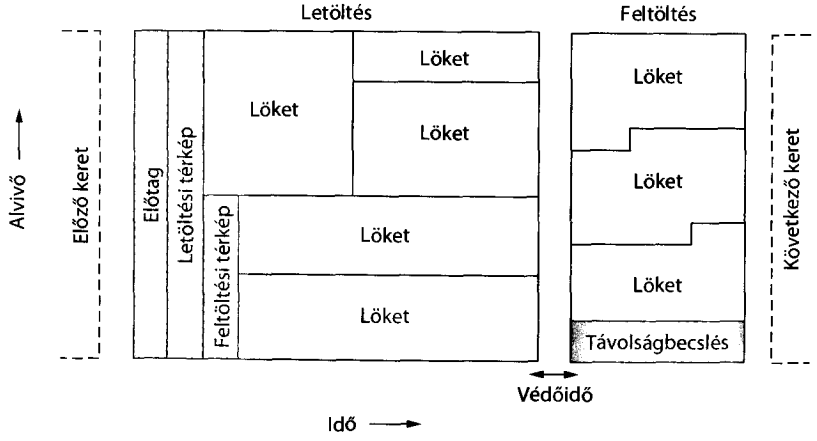
A szimbólumokat minden alvivőn valamelyik alábbi modulációs sémával küldik: QPSK-val, QAM-16-tal vagy QAM-64-gyel. (A modulációs sémákat a 2.5.3. szakaszban tárgyaltuk.) Amikor a mozgó vagy az előfizetői állomás közel van a bázisállomáshoz és a vett jel/zaj viszony (signal-to-noise ratio, SNR) magas, a QAM-64 használható szimbólumonkénti 6 bit küldésére. Az olyan távoli állomások elérésére, amelyekre a kis jel/zaj viszony jellemző, QPSK-t használnak, ami 2 bitet szállít szimbólumonként. Az adatot először hibajavítási céllal a 3.2.1. szakaszban bemutatott konvolúciós (esetleg jobb) kódolási sémával kódolják. Ezt a kódolást gyakran használják zajos csatornákon, hogy néhány bithibát elviseljen, és ne legyen szükség újraküldésekre. Ezeknek a modulációs és kódolási eljárásoknak valójában már ismerősnek kell tűnniük, mivel sok, már eddig tárgyalt hálózatban használják, beleértve a 802.11-et, a kábeltévét és a DSL-t. Az eredmény, hogy egy bázisállomás 5 MHz-es csatornánként és antennapáronként legfeljebb 12,6 Mb/s sebességgel tud működni a felhasználók felé, és 6,2 Mb/s sebességgel a bázisállomás felé.

Egyetlen dolog volt, amit a 802.16 tervezői nem szerettek, mégpedig a GSM- és a D-AMPS-hálózatok működésének egy bizonyos aspektusa. Mindkét rendszer ugyanúgy megegyező méretű frekvenciasávot használ feltöltési és letöltési irányban. Ezzel kimondatlanul is azt feltételezik, hogy mindkét irányban ugyanakkora a forgalom. A beszédforgalom az idő nagy részében szimmetrikus, de internet-hozzáférés esetén (és webböngészés alkalmával bizonyosan) a letöltés gyakran sokkal nagyobb, mint a feltöltés. Az arány sokszor 2:1, 3:1 vagy sok:1.

Így a tervezők egy flexibilis sémát választottak, hogy a csatornát felosszák az állomások között, amit **OFDMA**-nak (**Orthogonal Frequency Division Multiple Access – ortogonális frekvenciaosztásos többszörös hozzáférés**) neveznek. Ezzel az alvivők különböző csoportjai különböző állomásokhoz lehetnek rendelve, így több mint egy állomás tud adni vagy venni egyszerre. Ha ez egy 802.11-es hálózat lenne, akkor bármely adott pillanatban csak egy állomás használhatná az összes alvivőt. A sávszélesség-kiosztás rugalmasságának fokozása növelheti a rendszer teljesítményét, mivel egy adott alvivő elgyengülhet a többutas hatások miatt az egyik vevőnél, de tiszta marad egy másiknál. Az alvivőket annak az állomásnak lehet kiosztani, amelyik a legjobban ki tudja használni azt.

Az állomások az aszimmetrikus forgalmuknak megfelelően általában felváltva adnak és vesznek. Ezt az eljárást **TDD**-nek (**Time Division Duplexing – időosztásos kettőzés**) nevezik. A másik választható eljárást, amelyben egy állomás egyszerre ad és vesz (különböző alvivő frekvenciákon), **FDD**-nek (**Frequency Division Duplexing – frekvenciaosztásos kettőzés**) nevezik. A WiMAX mindkettő alkalmazását megengedi, de a TDD kedveltebb, mert egyszerűbb megvalósítani és rugalmasabb.

A 4.32. ábrán láthatunk példát egy keret szerkezetére, amely időben ismétlődik. Egy előtaggal (preamble) kezdődik, hogy szinkronizálja az összes állomást, eztán következik a lefelé irányú átvitel a bázisállomástól. Ennek az elején a bázisállomás térképeket küld el, amelyek tájékoztatják az állomásokat arról, hogy a keretben hogy vannak kiosztva a feltöltési és letöltési alvivők. A bázisállomás vezérli a térképeket, ezáltal keretről keretre különböző sávszélességeket tud kiosztani a különböző állomásoknak, igényeik szerint.



4.32. ábra. *Időosztásos kettőzéses OFDMA keretszerkezete*

Ezután a bázisállomás löketeket küld a különböző előfizetői és mozgó állomásoknak a térképen meghatározott alvivőn és időben. A lefelé irányuló átvitelek egy védőidővel végződnek, hogy az állomások vételről adásra válthassanak. Végül az előfizetői és mozgó állomások küldik a löketeiket a bázisállomásnak a térképen számukra fenntartott feltöltési pozícióban. Ezen feltöltési löketek közül egyet fenntartanak a **távolságbecslésre (ranging)**, ez az a folyamat, amelyben új állomások az időmérő eszközeiket a rendszerhez igazítják, és kezdeti sávszélességet kérnek, hogy a bázisállomáshoz csatlakozhassanak. Mivel itt még nincs összeköttetés kialakítva, az új állomások csak adnak, és remélik, hogy nem lesz ütközés.

4.5.4. A 802.16 MAC-alrétegének protokollja

Amint azt a 4.31. ábrán láthattuk, az adatkapcsolati réteg három alrétegre bontható. Mivel a titkosítást csak a 8. fejezetben tárgyaljuk, nehéz lenne most elmagyarázni, hogyan működik a biztonsági alréteg. Legyen elég annyit mondanunk, hogy titkosítást alkalmaznak annak érdekében, hogy minden átvitt adatot biztonságban tudjanak. Csak a keretek adatmezejét titkosítják, a fejrészeket nem. Ez azt jelenti, hogy egy lehallgató tudhatja, hogy ki beszél kihez, de azt már nem, hogy mit mondanak egymásnak.

Ha az olvasó tud már egyet és mást a titkosításról, akkor ajánljuk figyelmébe a biztonsági alréteg itt következő egy bekezdésnyi magyarázatát. Ha azonban még semmit sem tud a titkosításról, akkor a következő bekezdés valószínűleg nem fog túl sokat mondani (de talán érdemes lesz újra elolvasnia, miután befejezte a 8. fejezetet).

Amikor az előfizető és a bázisállomás kapcsolatba lépnek egymással, akkor kölcsönösen hitelesítik egymást az RSA nyilvános kulcsú titkosító eljárásával az X.509-es tanúsítványok segítségével. Magát az adatmezőt egy szimmetrikus kulcsú eljárás, konkrétan vagy AES (Rijndael), vagy titkosított blokkok láncolását használó DES alkalmazásával titkosítják. A sértetlenség ellenőrzését SHA-1-gyel oldják meg. Nem is volt olyan vészes, ugye?

Vessünk most egy pillantást a közös MAC-alréteg részre! A MAC-alréteg összeköttetés-alapú és egypont-többpont (point-to-multipoint) típusú, ami azt jelenti, hogy az egyetlen bázisállomás több előfizetői állomással kommunikál. Ennek a felépítésnek a nagy részét a kábelmodemektől vették át, amelyben az egyetlen kábelfejegység vezérli a sok előfizető lakásán vagy telephelyén lévő kábelmodemek átvitelét.

A lefelé irányuló csatorna működése eléggé magától értetődő. A bázisállomás vezérli a fizikai rétegbeli löketeket, amelyeket arra használnak, hogy adatokat vigyenek át az előfizetői állomásoknak. A MAC-alréteg egyszerűen berakja a kereteket ebbe a struktúrába. A többletbitek számának csökkentésére több lehetőség áll rendelkezésünkre. Például MAC-keretek küldhetők egyesével külön-külön vagy egy csoportot egybecsomagolva egyszerre.

A felfelé irányuló csatorna már bonyolultabb, hiszen annak eléréséért egymással össze nem hangolt előfizetők versengenek. A csatornakiosztás szorosan összefügg a szolgáltatásminőség kérdésével. A szabvány az alábbi négy szolgáltatásosztályt rögzíti:

1. Állandó adatsebességű szolgáltatás.

2. Valós idejű, változó adatsebességű szolgáltatás.

3. Nem valós idejű, változó adatsebességű szolgáltatás.

4. Garanciák nélküli (best-effort) szolgáltatás.

A 802.16-ban minden szolgáltatás összeköttetés-alapú. Minden összeköttetéshez egy szolgáltatási osztály kerül, amikor az összeköttetést kiépítik. Ez a megoldás eltér a 802.11-ben vagy az Ethernetben használt megoldástól, amelyek a MAC-alrétegben összeköttetés nélküliek.

Az állandó adatsebességű szolgáltatást a tömörítetlen beszéd átvitelére szánták. Ennek a szolgáltatásnak előre meghatározott mennyiségű adatot kell előre meghatározott időközönként elküldenie. Ezt úgy teszik lehetővé, hogy minden ilyen típusú összeköttetés számára fenntartanak bizonyos löketeket. A sávszélesség kiosztása után a löketek automatikusan rendelkezésre állnak anélkül, hogy egyenként kérni kellene azokat.

A valós idejű, változó adatsebességű szolgáltatás a tömörített multimédia és más, kevésbé intenzív valós idejű alkalmazások számára készült. Ezeknél az igényelt sávszélesség mennyisége minden pillanatban változhat. Ezt a szolgáltatást úgy valósítják meg, hogy a bázisállomás rögzített időközönként körbekérdezi az előfizetőket, hogy ezúttal mekkora sávszélességre van szükségük.

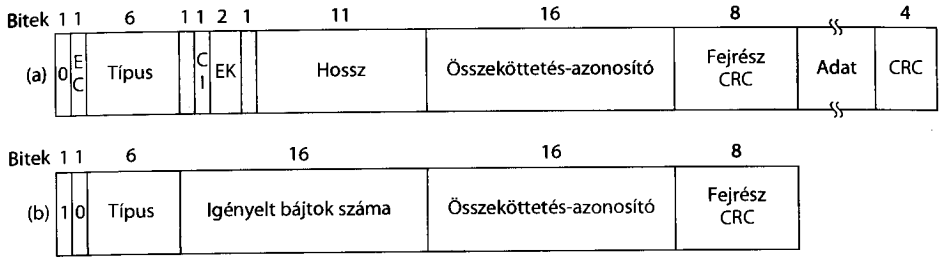
A nem valós idejű, változó adatsebességű szolgáltatás az olyan alkalmazások számára hasznos, amelyek jelentős mennyiségű adatot visznek át – például nagyméretű állományokat –, de nincs szükségük valós idejű átvitelre. Itt a bázisállomás gyakran, de nem szigorúan előírt időközönként kérdezi körbe az előfizetőket. Azok az összeköttetések, ahol van ilyen szolgáltatás, sávszélesség-igénylésre használhatják a garanciák nélküli szolgáltatást is, amit az alábbiakban mutatunk be.

A garanciák nélküli szolgáltatást szánták az összes fennmaradó célra. Itt nincs körbekérdezés, hanem az előfizetőnek kell versengenie a szolgáltatás többi előfizetőjével a sávszélességért. A sávszélességre vonatkozó igényeket azokban a löketekben lehet bejelenteni, melyek a feltöltési térkép szerint a versengésre rendelkezésre állnak. Az elfogadott kérésekről a következő, letöltési térképben küldenek értesítést. A visszautasított előfizetőknek később kell újra próbálkozniuk. Az ütközések számának minimalizálására az Ethernet kettes exponenciális visszalépéses algoritmusát használják.

4.5.5. A 802.16 keretszerkezete

Minden MAC-keret egy fejrésszel kezdődik. Ezt egy opcionális adatmező és egy opcionális ellenőrző összeg (CRC) követi, amint azt a 4.33. ábra mutatja. A vezérlőkeretekben, mint amilyen például a csatornában időszeleteket kérő keret is, nincs szükség adatmezőre. Meglepő módon az ellenőrző összeg is opcionális. Ez azért van, mert a fizikai rétegben van már hibajavítás, illetve azért, mert a valós idejű kereteket sohasem próbálják meg újraküldeni. Ha pedig úgysem lesz újraküldés, akkor mi szükség ellenőrző összegre? De ha mégis van ellenőrző összeg, akkor az egy szabványos IEEE 802 CRC, és nyugtákat és újraküldéseket alkalmaznak a megbízhatóság érdekében.

A következőkben röviden áttekintjük a 4.33.(a) ábrán látható fejrészmezőket. Az *EC (Encripted)* bit megadja, hogy az adatmező titkosítva van-e. A *Típus (Type)* mező a keret



4.33. ábra. *(a) Egy általános keret. (b) Egy sávszélesség-igénylő keret*

típusát azonosítja, és többnyire azt adja meg, hogy alkalmaztak-e csomagolást és darabolást. A *CI (Checksum Present)* mező a végső ellenőrző összeg meglétét vagy hiányát jelzi. Az *EK (Encription Key)* mező azt adja meg, hogy melyik titkosító kulcsot használták (ha volt ilyen). A *Hossz (Length)* mező a keret teljes hosszát mutatja, a fejrészt is beleértve. Az *Összeköttetés-azonosító (Connection Identifier)* megadja, hogy a keret melyik összeköttetéshez tartozik. Az utolsó mező, a *Fejrész CRC (Header CRC)* pedig egy, csak a fejrészre vonatkozó ellenőrző összeget tartalmaz, melyet az $x^8 + x^2 + x + 1$ polinom felhasználásával képeztek.

A 802.16 protokoll sokféle kerettel rendelkezik. Egy másik típusú keret látható a 4.33.(b) ábrán, melyet sávszélesség kérésére használnak. Ez 0-s helyett egy 1-es bittel kezdődik, és hasonlít az előző fejrészhez, azzal a különbséggel, hogy a második és harmadik bájt egy 16 bites számot tartalmaz, ami megadja, hogy mekkora sávszélességre van szükség az adott számú bájt elszállításához. A sávszélesség-igénylő keretek nem hordoznak sem adatmezőt, sem a teljes keretre vonatkozó ellenőrző összeget.

Jóval többet is mondhatnánk még a 802.16-ról, de erre nem ez a legmegfelelőbb hely. További információkat magában az IEEE 802.16-2009 szabványban érdemes keresnünk.

4.6. BLUETOOTH

Az L. M. Ericsson társaság 1994-ben kezdett érdeklődni az iránt, hogy mobiltelefonjait hogyan lehetne más eszközökkel (például laptopokkal) vezetékek nélkül kapcsolatba hozni. Négy másik vállalattal (IBM, Intel, Nokia és Toshiba) együtt rövidesen megalapította a SIG-et (Special Interest Group – különleges érdekcsoport, magyarul konzorcium). A csoport célja egy olyan vezeték nélküli szabvány kidolgozása volt, amely lehetővé teszi a számítástechnikai eszközök, kommunikációs eszközök és egyéb kiegészítők összekapcsolását rövid hatósugarú, kis teljesítményű, olcsó rádiós adó-vevők segítségével. A projekt a **Bluetooth** nevet kapta II. Harald Blaatand (a „Kékfogú", i. sz. 940–981) viking király után, akinek sikerült egyesítenie (magyarul elfoglalnia) Dániát és Norvégiát – szintén vezetékek nélkül.

A Bluetooth 1.0-t 1999 júliusában adták ki, azóta a SIG rá se nézett. Most már mindenféle elektronikus eszköz használja a Bluetooth-t, a mobiltelefonoktól és a laptopoktól a fejhallgatókig, nyomtatókig, billentyűzetekig, egerekig, játékkonzolokig, karórákig,

zenelejátszókig, navigációs eszközökig és így tovább. A Bluetooth-protokoll lehetővé teszi ezeknek az eszközöknek, hogy felderítsék egymást és egy **párosítás** (**pairing**) nevű folyamat keretében egymáshoz csatlakozzanak, és biztonságosan vigyenek át adatot.

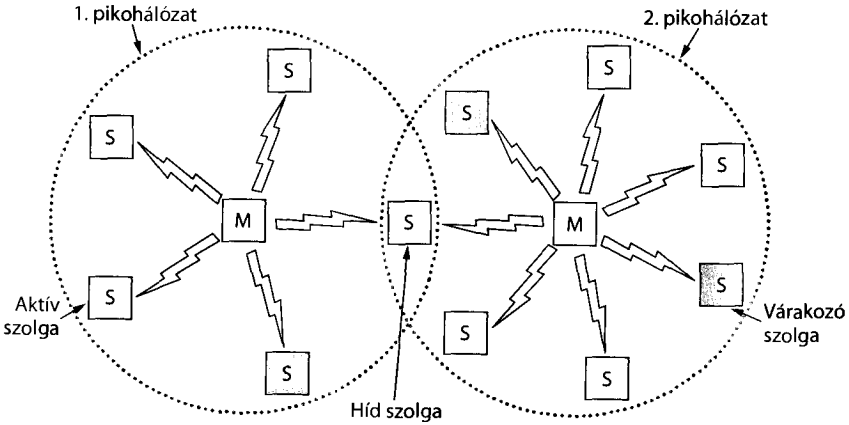
Ezek a protokollok az elmúlt évtizedben továbbfejlődtek. Miután a kezdeti protokollok kiforrottá váltak, 2004-ben a Bluetooth 2.0-t már nagyobb sebességgel definiálták. A 2009-ben megjelent Bluetooth 3.0-t már párosítani lehet a 802.11-gyel a nagyobb sebességű adatátvitel érdekében. A 2009 decemberében kiadott 4.0-ás verzió kis teljesítményű működést specifikált. Ez azoknak lesz kényelmes, akik nem szeretik rendszeresen cserélgetni az tápelemeket. A Bluetooth legfontosabb aspektusait a következő szakaszokban tárgyaljuk.

4.6.1. A Bluetooth felépítése

Tekintsük át először röviden a Bluetooth-rendszer elemeit és célját. A rendszer alapegysége a **pikohálózat** (**piconet**), mely egy mester (master) csomópontból és legfeljebb hét darab, 10 méteres távolságon belül levő aktív szolga (slave) csomópontból áll. Ugyanabban a (nagyméretű) helységben több pikohálózat is lehet egyszerre, sőt egy híd csomópont – amely több pikohálózatban is részt vesz – révén azok össze is köthetők, amint az a 4.34. ábrán is látható. Az egymással összekötött pikohálózatokat **szórt hálózatnak** (**scatternet**) is nevezik.

A pikohálózat hét aktív szolga csomópontja mellett legfeljebb 255 várakozó (parked) csomópont lehet a hálózatban. Ezek olyan eszközök, melyeket a mester alacsony teljesítményű állapotba vitt, hogy csökkentse az akkumulátoraik igénybevételét. Várakozó állapotban egy eszköz semmit sem tehet, leszámítva a mester aktivációs vagy jelzőfény jelére való válaszadást. Két köztes teljesítményállapot, a tartás (hold) és a szimatolás (sniff) szintén létezik, de ezekkel most nem foglalkozunk.

A mester/szolga felépítést az indokolja, hogy a tervezők 5 dollár alatti áron szerették volna megvalósítani egy komplett Bluetooth-lapka (chip) kivitelezését. Ennek a döntés-



4.34. ábra. *Két pikohálózat összekapcsolásával egy szórt hálózatot kapunk*

nek az a következménye, hogy a szolgák meglehetősen buták, és lényegében mindig csak azt teszik, amire a mester utasítja őket. A pikohálózat voltaképpen egy központosított TDM-rendszer, melyben a mester vezérli az órát, és eldönti, hogy melyik eszköz melyik időszeletben kommunikálhat. A kommunikáció mindig csak a mester és egy szolga között folyhat; a szolgák közvetlenül nem érintkezhetnek egymással.

4.6.2. Bluetooth-alkalmazások

A hálózati protokollok többnyire csak csatornákat adnak a kommunikáló entitások számára, és az alkalmazás tervezőire bízzák, hogy mire használják azokat. A 802.11 például nem határozza meg, hogy mit kell csinálniuk a felhasználóknak a hordozható számítógépeiken: e-leveleket olvasni, a weben szörfölni vagy éppen valami mást. A Bluetooth ellenben meghatároz konkrét támogatandó alkalmazásokat, és mindegyik számára külön protokollkészletet biztosít. A könyv írásának időpontjában 25 alkalmazás van, melyeket **profilnak** (**profile**) is neveznek. Sajnos ez a megközelítés nagyon nagyfokú bonyolultsághoz vezet. Ennek a részletezésétől mi most eltekintünk, de vetünk egy pillantást ezekre a profilokra, hogy jobban megértsük, hogy mit akar véghezvinni a Bluetooth SIG.

A profilok közül hatot különböző audio- és videoalkalmazásokra készítettek. Például, az interkom profil lehetővé teszi két telefon walkie-talkie-szerű összekapcsolását. A fejhallgató (headset) és a kéz nélküli (hands-free) profilok beszédhang kommunikációt biztosítanak a fejhallgató és a bázisállomása között, és kéz nélküli telefóniára használható például autóvezetés közben. Más profilok jó minőségű beszéd- és videofolyam átvitelére valók, mondjuk egy hordozható zenelejátszóról a fejhallgatókra vagy digitális kameráról tv-re.

Az emberi interfészeszköz (human interface device) profilja billentyűzetek és egerek számítógéphez csatlakozására valók. Más profilok arra szolgálnak, hogy egy mobiltelefon vagy másmilyen számítógép fogadhasson képeket fényképezőgéptől vagy küldhessen képeket nyomtatóra. Érdekesebb lehet az a profil, amellyel a mobiltelefont egy (Bluetooth-képes) tévé távirányítójaként lehet használni.

Megint más profilok lehetővé teszik egy hálózat kiépítését. A személyi hálózatok (personal area network) profilja a Bluetooth-eszközöknek nyújt lehetőséget, hogy ad hoc hálózatot építsenek vagy távolról elérjenek egy másik hálózatot, mint például egy 802.11-es hálózatot egy hozzáférési ponton keresztül. Az egész projektet eredetileg betárcsázós (dial-up) profil motiválta. Ez teszi lehetővé, hogy egy számítógép egy beépített modemet tartalmazó mobiltelefonhoz vezetékek használata nélkül kapcsolódjon.

Felsőbb rétegbeli információcserélő profilokat is definiáltak. A szinkronizációs profil arra szolgál, hogy egy mobiltelefonba át lehessen vinni az asztali számítógép adatait, mielőtt az elhagyja a lakást, majd visszatölteni az új adatokat hazaérkezéskor.

A többi profilt nem tárgyaljuk, de azt meg kell jegyeznünk még, hogy van néhány profil, amelyekre mintegy építőkockaként más profilokat építettek. Az alap hozzáférés (generic access) profilra épül az összes többi profil, mert ez nyújt lehetőséget biztonságos kapcsolatok felépítésére és karbantartására a mester és a szolgák között. Más alapprofilok objektumok kicserélésének és audio/video átvitel alapjait határozzák meg. A segédprofilokat széleskörűen használják olyan feladatokhoz, mint amilyen a soros vonal emulálása, ami különösen fontos sok hagyományos alkalmazásnál.

Vajon tényleg szükség volt arra, hogy ezeket az alkalmazásokat részletezzék, és külön protokollkészletet készítsenek mindegyikhez? Valószínűleg nem, de a szabvány különböző részeit több különböző munkacsoport dolgozta ki, mindegyik csak a saját konkrét problémájával foglalkozott, és egy saját profilt hozott létre. Vehetjük úgy is, hogy Conway törvénye lépett működésbe. (A *Datamation* magazin 1968. áprilisi számában írta le Melvin Conway azt a megfigyelését, hogy ha n embert bízunk meg egy fordítóprogram megírásával, akkor egy n-menetes fordítót kapunk; vagy még általánosabban fogalmazva: egy szoftver felépítése tükrözi annak a csoportnak a felépítését, amely létrehozta azt.) Minden bizonnyal két protokoll is elég lett volna a 25 helyett: egy az állományátvitel és egy a folyamszerű (streaming) valós idejű kommunikáció számára.

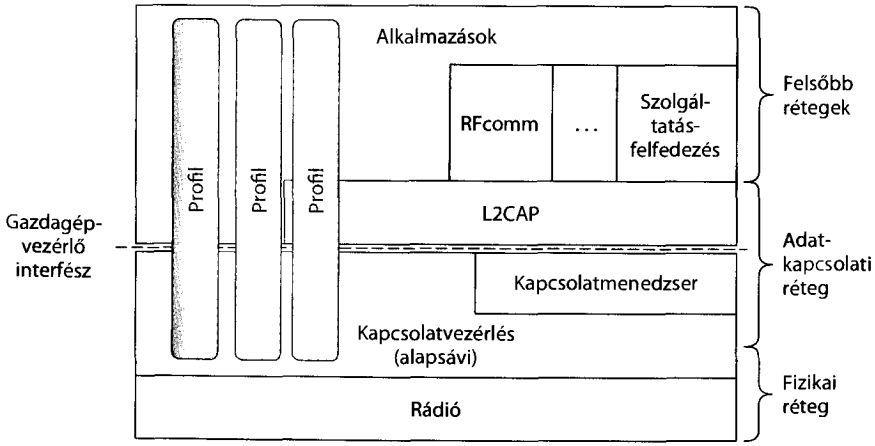
4.6.3. A Bluetooth protokollkészlete

A Bluetooth-szabványnak számos protokollja van, melyek laza rétegeket alkotnak, amint azt a 4.35. ábra mutatja. Az első megfigyelés, amit meg kell tennünk, hogy a felépítése nem követi sem az OSI-t, sem a TCP/IP-t, sem a 802-t, sem bármilyen más modellt.

Legalul van a fizikai rádiós réteg, ami nagyjából megfelel az OSI és a 802-es modellek fizikai rétegének. Ez a rádiós átvitellel és a modulációval foglalkozik. Itt az elgondolások jó része azzal a céllal kapcsolatos, hogy a rendszert minél olcsóbbá tegyék, vagyis hogy tömegtermelésben lehessen előállítani.

A kapcsolatvezérlő (vagy alapsávi réteg) valamelyest a MAC-alréteghez hasonlatos, de a fizikai rétegből is tartalmaz elemeket. Azzal foglalkozik, hogy a mester hogyan vezérli az időszeleteket, és hogy ezeket az időszeleteket hogyan csoportosítják keretekbe.

Ezután két protokoll következik, amelyek a kapcsolatvezérlő protokollt használják. A kapcsolatmenedzser az eszközök közötti logikai csatornák kiépítését kezeli, beleértve a teljesítménygazdálkodást, a párosítást és titkosítást, valamint a szolgáltatásminőséget



4.35. ábra. *A Bluetooth-protokoll architektúrája*

is. Ez még a gazdagép–vezérlő interfész (Host Controller Interface) alatt fekszik. Ez az interfész a megvalósítás könnyítésére való: tipikusan az interfész alatt elhelyezkedő protokollokat a Bluetooth-lapkán valósítják meg, és a felette elhelyezkedő protokollokat az eszközön valósítják meg, amely a Bluetooth-lapka (chip) gazdagépe.

A kapcsolati protokoll az interfész felett az **L2CAP** (**Logical Link Control Adaptation Protocol – logikai kapcsolatvezérlés adaptációs protokollja**) helyezkedik el. Ez változó hosszúságú üzeneteket helyez keretekbe és szükség esetén megbízhatóságot is nyújt. Sok protokoll használja az L2CAP-ot, mint az a két hasznos protokoll, amit az ábrán megjelenítettünk. A szolgáltatás-felfedezés (Service Discovery) protokoll a hálózaton elérhető szolgáltatás felderítésére használatos. Az RFcomm (Radio Frequency Communication – rádiófrekvenciás kommunikáció) protokoll azt a szabványos soros portot helyettesíti, mely a PC-ket köti össze a billentyűzettel, egérrel, modemmel és más eszközökkel.

A legfelső rétegben vannak az alkalmazások. A profilokat függőleges dobozok képviselik, mert közülük mindegyik meghatározza a protokollkészlet egy szeletét egy bizonyos felhasználásra. Az egyes profilok, mint például a fejhallgató profilja, rendszerint csak az adott alkalmazás számára szükséges protokollokat tartalmazzák, semmi mást. Például a profilok magukba foglalhatják az L2CAP-ot, ha csomagokat akarnak küldeni, de ki is hagyhatják, ha csak egy stabil hangmintafolyamuk van.

A következő szakaszokban a Bluetooth rádiós rétegét vesszük szemügyre, illetve a különböző kapcsolati protokollokat, mivel ezek nagyjából megfelelnek más, korábban vizsgált protokollkészletek fizikai és MAC-alrétegének.

4.6.4. A Bluetooth rádiós rétege

A rádiós réteg a biteket szállítja a mestertől a szolgáig vagy fordítva. Ez egy kis teljesítményű rendszer, 10 méteres hatósugárral, ugyanabban a 2,4 GHz-es ISM-sávban, mint a 802.11. A sáv 79 darab, egyenként 1 MHz-es csatornára van osztva. Hogy más hálózatokkal együtt tudjon élni az ISM-sávban, frekvenciaugrásos szórtspektrumot használ. Lehet akár 1600 ugrás másodpercenként az időszeletek felett 625 μs-os tartózkodási idővel (dwell time). A pikohálózat összes csomópontja – követve a résidőzítést és a mester által diktált álvéletlen ugrási sorozatot – egyszerre végzi a frekvenciaugrásokat.

Sajnos az derült ki, hogy a Bluetooth korai verziói és a 802.11 annyira zavarták egymást, hogy tönkretették egymás átviteleit. Néhány cég a Bluetooth betiltásával válaszolt erre, de végül is kitaláltak egy műszaki megoldást, mégpedig azt, hogy a Bluetooth-nak úgy kell alakítania az ugrási sorozatát, hogy kihagyja azokat a csatornákat, ahol van más rádiófrekvenciás jel. Ez a folyamat csökkenti a káros zavarást, és **adaptív frekvenciaugrásnak** (**Adaptive Frequency Hopping**) nevezik.

Három modulációs eljárást használnak a bitek csatornán való átvitelére. Az alapvető modulációs eljárásként frekvenciabillentyűzést alkalmaznak 1 bites szimbólumok mikroszekundumonkénti átvitelére, ami kiadja a bruttó 1 Mb/s-os adatsebességet. Megnövelt sebességeket vezettek be a Bluetooth 2.0 verziójánál. Ezek a sebességek fázisbillentyűzést használnak szimbólumonkénti 2 vagy 3 bit küldésére, ami bruttó 2 vagy 3 Mb/s sebességet jelent. A megnövelt sebességeket csak a keretek adatrészében használják.

4.6.5. A Bluetooth kapcsolati rétegei

A kapcsolatvezérlési (vagy alapsávi) réteg az, ami a Bluetooth-ban leginkább hasonlít a MAC-alrétegre. Ez keretekbe rendezi a nyers bitfolyamot, és definiálja a legfontosabb formátumokat. Legegyszerűbb formájában az egyes pikohálózatokban lévő mesterek 625 μs-os időszeletek sorozatát határozzák meg, ahol a mesterek adásai a páros, a szolgák adásai pedig a páratlan időszeletekben kezdődnek. Ez a séma a hagyományos időosztásos nyalábolás: a mesterek kapják meg az időszeletek egyik felét, a szolgák pedig a másikat. A keretek hossza 1, 3 vagy 5 időszelet lehet. Minden keretnek van többleteleme, mely 126 bitből áll a hozzáférési kód és a fejrész számára, plusz ugrásonként 250–260 μs beállási időből (settling time), hogy az olcsó rádiós áramkörök stabilizálódhassanak. Annak érdekében, hogy a keret felhasználói adata megbízható maradjon, titkosítható egy kulccsal, amelyet a mester és a szolga összekapcsolódásakor választanak ki. Ugrások csak a keretek között történnek, nem pedig egy keret alatt. Ezek eredményeként egy 5 időszeletes keret sokkal hatékonyabb, mint egy 1 időszeletes keret, mert a többletelem állandó, de több adat került elküldésre.

A kapcsolatmenedzser protokoll egy **kapcsolatnak** (**link**) nevezett logikai csatornát állít fel, hogy kereteket vigyen át a mester és a szolga eszközök között, amelyek felfedezték egymást. Ezután egy párosítási folyamat következik, hogy mielőtt a csatornát használják, megbizonyosodjanak arról, hogy a két eszköznek megengedett az egymással való kommunikáció. A régi párosítási módszer alapján mindkét eszközt ugyanazzal a négy számjegyű PIN-számmal (Personal Identification Number – személyi azonosító szám) kell felkonfigurálni. Az egyező PIN-számok alapján ismerte volna fel egy eszköz, hogy a megfelelő távoli eszközhöz csatlakozik. Fantáziátlan felhasználók és eszközök olyan alap PIN-számokat használtak, mint a „0000" és az „1234", ami azt jelentette, hogy a gyakorlatban ez a módszer nagyon kis biztonságot nyújtott.

Az új, **egyszerű, biztonságos párosítási módszer** lehetővé teszi, hogy a felhasználók megerősítsék azt, hogy mindkét eszköz ugyanazt a jelszót használja, vagy hogy megfigyeljék az egyik eszközön a jelszót, és bevigyék a másikba. Ez a módszer biztonságosabb, mert a felhasználóknak nem kell választaniuk vagy beállítaniuk egy jelszót. Nekik csupán egy hosszabb, az eszköz által létrehozott jelszót kell megerősíteniük. Természetesen ez nem használható olyan korlátozott be- és kiviteli képességű eszközökön, mint egy fejhallgató.

Amint a párosítás megtörtént, a kapcsolatmenedzser protokoll felépíti az adatkapcsolatot. Felhasználói adatok szállítására két fő kapcsolattípus létezik. Az első az **SCO**-kapcsolat (**Synchronous Connection Oriented – szinkron összeköttetés-alapú**). Ezt valós idejű adatok esetén, például telefonos kapcsolatoknál használják. Ehhez a kapcsolattípushoz mindkét irányban egy rögzített időszeletet rendelnek. Egy szolgának legfeljebb három SCO-kapcsolata lehet a mesterével. Minden egyes SCO-kapcsolat egy 64 kb/s-os PCM-beszédcsatornát képes átvinni. Az SCO-kapcsolatok időérzékeny természetéből adódóan a rajtuk átküldött kereteket sosem küldik újra. Ehelyett a megbízhatóság növelése érdekében hibajavítás alkalmazható.

A másik lehetőség az **ACL**-kapcsolat (**Asynchronous Connection-Less – aszinkron összeköttetés nélküli kapcsolat**). Ezt a kapcsolattípust a szabálytalan időközönként érkező csomagkapcsolt adatokhoz használják. Az ACL-forgalom „best-effort" alapon

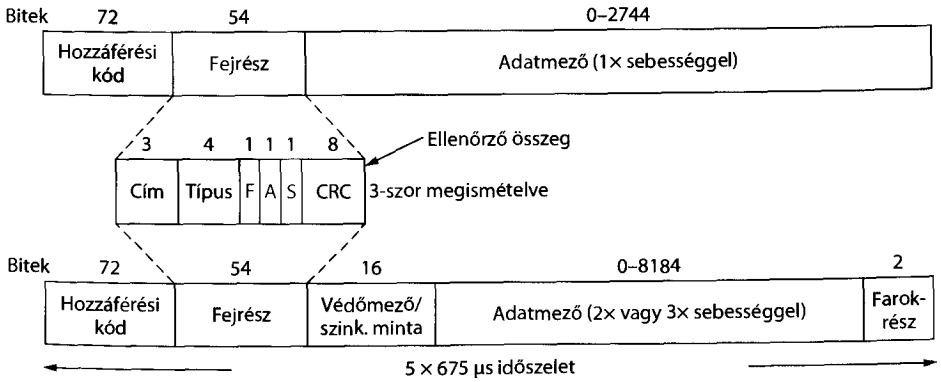
bonyolódik, garanciák tehát nincsenek: a keretek elveszhetnek, ilyenkor esetleg újra kell adni azokat. Egy szolgának legfeljebb egy ACL-kapcsolata lehet a mesterével.

Azt ACL-kapcsolaton küldött adat az L2CAP-rétegen keresztül érkezik meg. Ennek a rétegnek négy fő funkciója van. Először is ez fogadja a maximum 64 KB-os csomagokat a felsőbb rétegektől, és azokat az átvitel céljából keretekre tördeli. A másik oldalon a keretekből újra csomagokat állít elő. Másodszor, több csomagforrás esetén ez a réteg kezeli a nyalábolást (multiplexing) és a nyalábbontást (demultiplexing). A csomagok újbóli összerakása után az L2CAP-réteg dönti el, hogy melyik felsőbb rétegbeli protokollnak – például az RFcomm-nak vagy a szolgáltatás-felfedezésnek – kell átadni azokat. Harmadszor, az L2CAP-réteg kezeli a hibaellenőrzést és az újraküldéseket. Ez észleli a hibákat és küldi újra a nem nyugtázott csomagokat. Végül, az L2CAP betartatja a több kapcsolat közötti szolgáltatásminőségi követelményeket is.

4.6.6. A Bluetooth keretszerkezete

A Bluetooth többféle keretformátumot határoz meg, melyek közül a legfontosabbat két változatban a 4.36. ábra mutatja be. Ez egy hozzáférési kóddal kezdődik, amely általában a mestert azonosítja, így az egyszerre két mester adáskörzetében tartózkodó szolgák tudni fogják, hogy melyik forgalom honnan származik. Ezt egy 54 bites fejrész követi, ami tipikus MAC-alrétegbeli mezőket tartalmaz. Ha a keretet alapsebességgel küldik, akkor ezután jön az adatmező, amely maximum 2744 bit hosszúságú öt időszeletes átvitel esetén. Ugyanez a formátum érvényes akkor is, ha csak egy időszeletet használunk, de ekkor az adatmező csak 240 bites.

Ha a keretet megnövelt sebességgel küldik, akkor az adatrészbe 2-szer vagy 3-szor annyi bit fér bele, mert minden szimbólum 2 vagy 3 bitet szállít 1 bit helyett. Ezeket az adatokat megelőzi egy védőmező és egy szinkronizációs minta, amit a gyorsabb sebességre váltásra használnak. Azaz, a hozzáférési kód és a fejrész alapsebességgel halad, és csak az adatmező halad gyorsabban. A megnövelt sebességű keretek egy rövid farokrésszel végződnek.



4.36. ábra. *Egy jellegzetes Bluetooth-adatkeret alapsebesség (fölül) és megnövelt sebesség (alul) esetén*

Vessünk egy pillantást a közös fejrészre. A *Cím (Address)* mező a nyolc aktív eszköz közül azonosítja azt, amelyiknek a keretet szánták. A *Típus (Type)* mező azonosítja a keret típusát (ACL, SCP, lekérdezés vagy nulla) és az adatmezőben használt hibajavítást, valamint megadja, hogy hány időszeletre terjed ki a keret. A *Folyam (Flow)* bitet a szolga állítja be, ha a puffere megtelt és már nem tud több adatot fogadni. Ez a bit a forgalomszabályozás egy primitív formáját hozza létre. A *Nyugta (Acknowledgement)* bit segítségével egy ACK-t ültetnek rá a keretre. A *Sorszám (Sequence)* bittel számozzák meg a kereteket az újraadások felismerése céljából. Azért elég ide mindössze 1 bit, mert a megáll-és-vár protokollt használják. Ezután következik a fejrész 8 bites *Ellenőrző összege (Checksum)*. Az egész 18 bites fejrészt háromszor ismétlik meg, így jön ki a 4.36. ábrán látható 54 bites fejrész. A vételi oldalon egy egyszerű áramkör megvizsgálja az egyes bitek mindhárom másolatát. Ha ezek mind megegyeznek, a bitet elfogadják. Ha nem, a többség dönt. Ily módon 54 bites átviteli kapacitást használnak fel a 10 bites fejrész elküldésére. Ezt az indokolja, hogy zajos környezetben csak jókora redundancia segítségével lehet olcsó, kis teljesítményű (2,5 mW-os), csekély számítási kapacitású eszközökkel megbízható adatátvitelt megvalósítani.

Az ACL- és SCO-keretek adatmezejénél számos formátum használatos. Az alapsebességű SCO-keretek tanulmányozása egyszerű: az adatmező mindig 240 bites. Itt három különböző változatot definiáltak, ezek 80, 160 vagy 240 bites tényleges adatot tesznek lehetővé; a maradék a hibajavítást szolgálja. A legmegbízhatóbb változatnál (ahol 80 bites a tényleges adat), a tartalmat egyszerűen háromszor megismétlik, akárcsak a fejrésznél.

Kiszámolhatjuk ennek a keretnek a kapacitását. Mivel a szolga csak a páratlan időszeleteket használhatja, 800 időszeletet kaphat másodpercenként, éppúgy, mint a mester. 80 bites tényleges adat esetén a csatorna kapacitása mind a szolga, mind a mester részéről 64 kb/s. Ez a kapacitás pedig éppen elég egy duplex PCM-beszédcsatorna számára (ezért döntöttek a másodpercenkénti 1600 ugrás mellett). Azaz, hiába áll rendelkezésre 1 Mb/s átviteli kapacitás, egy tömörítetlen, 64 kb/s-os duplex beszédcsatorna teljesen telíti a pikohálózatot. A 13%-os hatékonyság az eredménye annak, hogy a kapacitás 41%-át beállási idővel, 20%-át fejrésszel és 26%-át ismétlő kódolással töltik. Ez a hiányosság emeli ki a megnövelt sebesség és a több mint 1 időszeletes keretek használt.

Sokkal többet is lehetne még mondani a Bluetooth-ról, de a könyv keretei ezt nem teszik lehetővé. Az érdeklődők számára a Bluetooth 4.0 specifikáció tartalmaz minden részletet.

4.7. RFID

Eddig megnéztünk már különböző MAC működéseket a LAN-októl fel a MAN-okig és le a PAN-okig. Utolsó példaként olyan legalsó kategóriás vezeték nélküli eszközöket vizsgálunk meg, amelyeket általában nem tekintenek a számítógép-hálózatok egyik fajtájának: az **RFID- (Radio Frequency IDentification – rádiófrekvenciás azonosítás)** címkéket és -olvasókat, amelyeket már az 1.5.4. szakaszban ismertettünk.

Az RFID-technika sok formában ismeretes: csipkártyákban, háziállatokba beültetve, útlevelekben, könyvtári könyvekben és még sok helyen. Azt a formát, amelyet meg fogunk nézni, egy **elektronikus termékkód (Electronic Product Code, EPC)** kutatása közben fejlesztettek ki, ami 1999-ben a Massachusetts Institute of Technology egyetem

Auto-ID Centerében kezdődött. Az EPC a vonalkódot helyettesíti. Az EPC nagyobb mennyiségű információt tud szállítani és elektronikus eszközökkel 10 méteres távolságig olvasható még akkor is, ha az olvasónak nincs rálátása a címkére. Ez a technika különbözik például az útlevelekben használt RFID-től, amit viszonylag közel kell helyezni az olvasóhoz, hogy az átvitel megtörténjen. A nagyobb távolságon átívelő kommunikáció képessége miatt az EPC relevánsabb a tanulmányainkhoz.

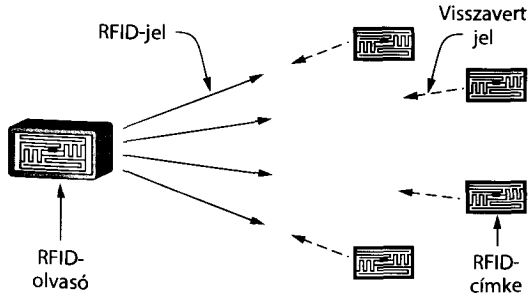
Az EPCglobalt azért alapították meg 2003-ban, hogy piacra dobják az Auto-ID Center által kifejlesztett RFID-technikát. Az erőfeszítéseik 2005-ben megerősítést kaptak, amikor a Walmart kötelezte a 100 legnagyobb beszállítóját, hogy RFID-címkékkel lássák el az összes szállítmányukat. Széles körű alkalmazását gátolta az olcsó nyomtatott vonalkódokkal való küzdelem nehézsége, de az új alkalmazásokban, mint például a jogosítványokban, egyre népszerűbbek. Ennek a technikának a második generációját mutatjuk be itt, amelyet nem hivatalosan **EPC Gen 2-nek** neveznek [EPCglobal, 2008].

4.7.1. Az EPC Gen 2 felépítése

Az EPC Gen 2 RFID-hálózat felépítését a 4.37. ábrán mutatjuk be. Két fő alkotóeleme van: a címkék és az olvasók. Az RFID-címkék kicsi, olcsó eszközök, amelyek egyedi 96 bites EPC-azonosítóval és az RFID-olvasó által olvasható pici memóriával vannak ellátva. A memória használható például annak a feljegyzésére, hogy egy tárgy útja során milyen helyeken járt.

A címkék általában úgy néznek ki, mint egy matrica, amilyet például egy áruházi polcon lévő farmernadrágra ragasztanak. A címke nagy részét a rányomtatott antenna teszi ki. A középső kis pötty az RFID integrált áramkör. Ritkább esetben az RFID-címkék egy tárgyba integrálva is megjelenhetnek, például egy jogosítványban. Egyik esetben sincsenek a címkék áramforrással ellátva és a közeli RFID-olvasó rádiós átviteléből kell teljesítményt nyerniük a működéshez. Ezt a címketípust „1-es osztályú" címkének nevezik, hogy ezzel is megkülönböztessék a fejlettebb címkéktől, amelyek áramforrással vannak ellátva.

Az olvasók képezik a rendszer intelligens elemét, a bázisállomásokhoz és a hozzáférési pontokhoz hasonlóan a mobil- és Wi-Fi-hálózatokban. Az olvasók sokkal többet tudnak, mint a címkék. Van saját áramforrásuk és gyakran több antennájuk is. Az olvasók



4.37. ábra. *Az RFID felépítése*

feladata a címkéknek szóló üzenetek küldése és a tőlük jövő üzenetek fogadása. Mivel hatótávolságon belül gyakran több címke tartózkodik, az olvasóknak kell megoldania a többszörös hozzáférés problémáját. Az is előfordulhat, hogy több olvasó verseng egymással ugyanazon a területen.

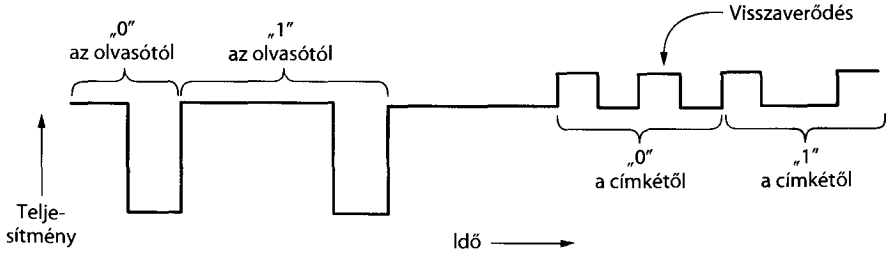
Az olvasó elsődleges dolga, hogy leltárba vegye a szomszédos címkéket, azaz felfedezze a közeli címkék azonosítóját. Egy fizikai rétegbeli protokoll és egy címkeazonosító protokoll végzi a leltározást, amelyeket a következő szakaszokban vázolunk.

4.7.2. Az EPC Gen 2 fizikai rétege

A fizikai réteg határozza meg, hogy az RFID-olvasó és -címke között a biteket hogyan küldjék. Nagyrészt olyan módszereket használ a vezeték nélküli jelek küldésére, amelyeket már láttunk korábban. Az USA-ban az átvitelek az engedélyhez nem kötött 902-928 MHz-es ISM-sávban történnek. Ez a sáv az UHF- (Ultra High Frequnecy – ultra nagy frekvencia) tartományba esik, ezért a címkékre UHF RFID-címkeként is hivatkoznak. Az RFID-olvasó legalább 400 ms-onként frekvenciaugrást végez, hogy a jelét szétszórja a csatornán, ezzel korlátozva a zavarást, és hogy megfeleljen az előírásoknak. Az RFID-olvasó és az RFID-címkék a bitek kódolására (a 2.5.2. szakaszban bemutatott) ASK- (Amplitude Shift Keying – amplitúdó billentyűzés) moduláció egy formáját használja. A bitek küldése fordulókra van osztva, ezért az adatkapcsolat fél-duplex.

Két fő különbség van más, eddig már tanult fizikai rétegekhez képest. Az első az, hogy az olvasó mindig küld jelet, függetlenül attól, hogy vajon az olvasó vagy a címke kommunikál. Természetesen az olvasó küld jelet, amikor biteket továbbít a címkének. Ahhoz, hogy a címke biteket küldjön az olvasónak, az olvasó egy rögzített vivőjelet küld, amely nem szállít biteket. A címkék begyűjtik ezt a jelet, hogy a működéshez energiát nyerjenek belőle; különben a címke nem lenne képes előbb jelet küldeni. Adatok küldéséhez a címke úgy változik, hogy az olvasótól jövő jelet – a tárgyakról visszaverődő radarjelekhez hasonlóan – visszaveri vagy elnyeli.

Ezt a módszert **visszaverődésnek** (**backscatter**) nevezik. Ez különbözik az összes többi vezeték nélküli helyzettől, amelyeket eddig láttunk, és amelyekben az adó és a vevő sohasem küld egyszerre ugyanabban a pillanatban. A visszaverődés egy kis energiájú módszer, amely lehetővé teszi, hogy a címke egy gyenge saját jelet hozzon létre, ami megjelenik az olvasónál. Ahhoz, hogy az olvasó dekódolja a bejövő jelet, ki kell szűrnie a saját



4.38. ábra. *Az olvasó által kibocsátott jelek és a címke visszavert jelei*

maga által küldött jelet. Mivel a címke jele gyenge, a címkék csak az olvasónak tudnak jelet küldeni kis sebességgel. A címkék nem tudják sem venni, sem érzékelni egymás jeleit.

A második különbség, hogy nagyon egyszerű modulációs formát használ, ami egy nagyon kis teljesítményű, nagyon olcsón legyártható címkén megvalósítható. Az olvasó két amplitúdószintet használ a címkéknek szóló adatküldéshez. Egy bit attól függően lesz 0-s vagy 1-es, hogy az olvasó milyen hosszan vár a kis teljesítményű periódus előtt. A címke méri a kis teljesítményű periódusok közötti időt, és összehasonlítja az előtagban küldött referenciaidővel. Ahogy az a 4.38. ábrán is látható, az 1-esek hosszabbak a 0-soknál.

A címke válaszai a címke adott időintervallumok alatti váltakozó visszaverődési állapotát tartalmazó impulzusjel-sorozatok. Egytől nyolcig terjedő impulzusjelet tartalmazó periódusok használhatók a 0-s vagy 1-es bitek kódolására attól függően, hogy milyen szintű megbízhatóságra van szükség. Az 1-es bitben kevesebb átmenet van, a 0-s bitben több, ahogy az a 4.38. ábrán bemutatott két impulzusjel periódusú kódolásból is látszik.

4.7.3. Az EPC Gen 2 címkeazonosító rétege

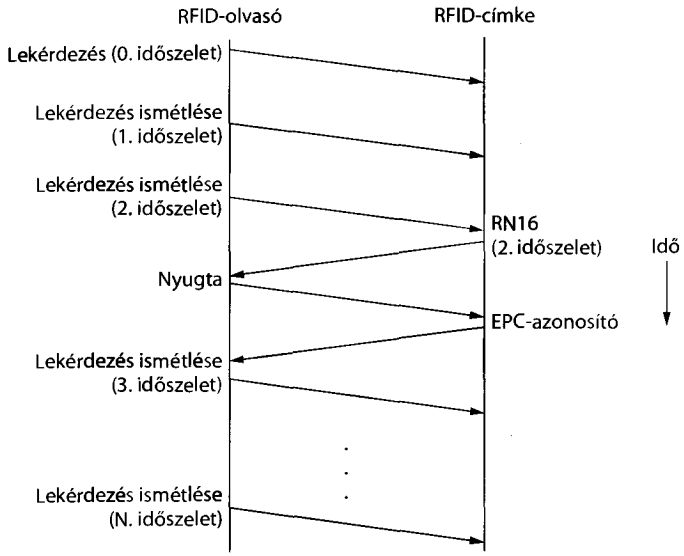
A közeli címkék leltározásához, az olvasónak minden címkétől fogadnia kell egy, a címkét azonosító üzenetet. Ez a szituáció egy többszörös hozzáférési problémának az az általános esete, melyben a címkék száma nem ismert. Az olvasó egy lekérdezést szórhatna, hogy megkérje az összes címkét, hogy küldjék el az azonosítójukat. Viszont azok a címkék, amelyek azonnal válaszolnának, ugyanúgy ütköznének, mint a klasszikus Ethernet állomásai.

Ebben a fejezetben már sok módszert láttunk a többszörös hozzáférés kezelésére. A jelenlegi helyzethez legközelebb álló protokoll, amelyben a címkék nem hallják egymás adását, az időszeletelt ALOHA, a tanult protokollok közül az egyik legkorábbi. Ezt a protokollt alakították át úgy, hogy a Gen 2 RFID esetén is használható legyen.

A címkék azonosítására szolgáló üzenetek sorrendjét a 4.39. ábra mutatja be. Az első időszeletben (0. időszelet), az olvasó küld egy *Lekérdezés (Query)* üzenetet, amivel elindítja a folyamatot. Minden *Lekérdezés ismétlése (QRepeat)* üzenet továbblép a következő időszeletre. Az olvasó elmondja a címkéknek, hogy mekkora időszelet-tartományon belül kezdhetik véletlenszerűen az átvitelüket. Az időszelet-tartomány használata szükségszerű, mert az olvasó szinkronizálja a címkéket, amikor elkezdi a folyamatot; az Ethernet állomásaitól eltérően a címkék nem ébrednek fel egy választásuk szerinti időpontban üzenet küldésével.

A címkék véletlenszerűen választanak egy időszeletet, amelyben válaszolnak. A 4.39. ábrán a címke a 2. időszeletben válaszol. A címkék azonban nem küldik el az azonosítójukat, amikor először válaszolnak, hanem egy rövid 16 bites véletlen számot küldenek egy *RN16* üzenetben. Ha nincs ütközés, az olvasó veszi az üzenetet és küld egy saját *Nyugta (ACK)* üzenetet. Ebben a fázisban a címke már megszerezte az időszeletet és elküldi az EPC-azonosítóját.

Ezt az üzenetváltást az indokolja, hogy az EPC-azonosítók túl hosszúak, így az ilyen üzenetek ütközése túl drága lenne. Ehelyett egy rövid üzenetváltást használnak annak vizsgálatára, hogy a címke biztonságosan használhatja-e az időszeletet azonosítójának az elküldésére. Miután a címke sikeresen elküldte az azonosítóját, ideiglenesen nem válaszol az új *Lekérdezés* üzenetekre, hogy a többi címkét is azonosítani lehessen.



4.39. ábra. *Példa a címke azonosítására szolgáló üzenetváltásra*

Az olvasó fő problémája, hogy behangolja az időszeletek számát úgy, hogy elkerülje az ütközéseket, de ne is használjon túl sokat, mert az a teljesítőképességet rontaná. Ez a behangolás analóg az Ethernet kettes exponenciális visszalépés algoritmusával. Ha az olvasó túl sok válasz nélküli időszeletet vagy túl sok ütközéses időszeletet lát, küldhet egy *Lekérdezés-behangolás (QAdjust)* üzenetet, hogy csökkentse vagy növelje azt az időszelet-tartományt, amelyben a címkék válaszolnak.

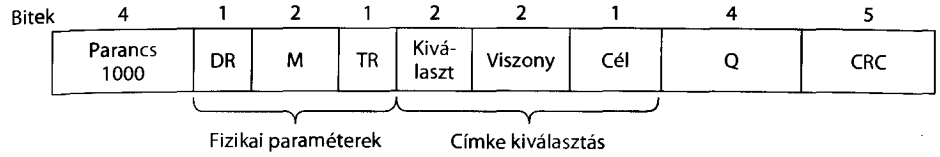
Az RFID-olvasó más műveletet is végre tud hajtani a címkéken. Például ki tudja választani a címkék egy részhalmazát, mielőtt elkezdi a leltározást, ezzel lehetővé téve azt, hogy mondjuk a címkézett farmernadrágok és ne a címkézett ingek válaszát gyűjtse össze. Az olvasó adatot is tud írni a címkékre, miután azonosította azokat. Ez a képesség az eladás helyének vagy más releváns információnak a feljegyzésére használható.

4.7.4. A címkeazonosítási üzenet formátumai

A *Lekérdezés* üzenet formátumát a 4.40. ábra mutatja, amely egyben példa is az olvasótól a címke felé küldött üzenetre. Az üzenet kompakt, azaz minden szükséges képességet magába foglal, mert a lefelé irányuló adatsebesség korlátozva van, 27 kb/s-tól 128 kb/s-ig. A *Parancs (Command)* mező hordozza az 1000 kódot, ami *Lekérdezés* üzenet azonosítója.

A következő jelzőbitek, a *DR*, *M* és a *TR* határozzák meg a fizikai réteg paramétereit az olvasó adásaihoz és a címkék válaszaihoz. Például beállítható a válasz sebessége 5 kb/s és 640 kb/s között. E jelzőbitek tárgyalását most mellőzzük.

Az ezután következő három mező a *Kiválaszt (Select)*, a *Viszony (Session)* és *Cél (Target)*. Ezek választják ki azt a címkét, amely majd válaszolni fog. Ahogy az olvasó ki tudja választani az azonosítók egy részét, úgy a címkék is követni tudnak négy párhuza-



4.40. ábra. *A Lekérdezés üzenetformátuma*

mos viszonyt, illetve hogy az adott viszonyban azonosítva vannak-e. Ezzel a módszerrel több olvasó tud működni átlapolódó lefedettségi területen azáltal, hogy különböző viszonyt használnak.

A következő paraméter a parancs legfontosabb eleme: a *Q* mező. Ez a mező határozza meg az időszeletek tartományát, amelyen belül a címkék válaszolni fognak: 0-tól $2^Q - 1$-ig. Végül, van még egy CRC, hogy az üzenet mezőit védje. Az 5 bites hosszúsága rövidebb, mint az eddig látott CRC-k, de a *Lekérdezés* üzenet is sokkal rövidebb, mint a legtöbb csomag.

A címkétől az olvasóig menő üzenetek egyszerűbbek. Mivel az olvasó vezérel, ezért tudja, hogy milyen üzenetre számítson válaszul az egyes átviteleire. A címkétől jövő válaszok egyszerűen olyan adatokat szállítanak, mint amilyen az EPC-azonosító.

Eredetileg a címkéket csak azonosítási célra szánták. Az idők folyamán azonban felnőttek, és ma már egy nagyon kis számítógépre emlékeztetnek. Néhány kutatási fázisban lévő címke szenzorral van felszerelve, és képes egyszerű programokat futtatni, hogy adatokat gyűjtsön be és dolgozzon fel [Sample és mások, 2008]. Egy Vízió erről a technikáról az „internet of things" („Dolgok internete"), amely a fizikai világban található tárgyakat kapcsolja az internethez [Welbourne és mások, 2009, valamint Gershenfeld és mások, 2004].

4.8. Kapcsolás az adatkapcsolati rétegben

Sok szervezet több LAN-nal is rendelkezik, és szeretnék ezeket összekötni. Nem lenne egyszerűbb, ha a LAN-okat össze tudnánk kapcsolni egy nagyobb LAN-ná? Valójában meg tudjuk ezt tenni, amikor az adatkapcsolatokat az úgynevezett **hidakkal (bridge)** alakítjuk ki. A 4.3.4. szakaszban bemutatott Ethernet-kapcsoló a híd modernebb neve. A klasszikus Ethernetnél és az Ethernet-elosztóknál szélesebb funkcionalitást nyújt, hogy több LAN-t egyszerűbben lehessen nagyobb és gyorsabb hálózattá összekapcsolni. A „híd" és a „kapcsoló" fogalmakat szinonimaként használjuk.

A hidak az adatkapcsolati rétegben működnek, és az adatkapcsolati réteg címeit vizsgálják meg a keretek továbbításához. Mivel az általuk továbbított keretek adatmezejét nem vizsgálják meg, képesek IP- vagy másfajta csomagokat kezelni (például AppleTalk-csomagokat). Az *útválasztók* ezzel szemben a csomagokban található címeket vizsgálják meg, és azok alapján végzik az útválasztást, tehát csak azzal a protokollal működnek, amelynek kezelésére tervezték.

Ebben a szakaszban megnézzük a hidak működését, illetve azt, hogyan lehet velük több fizikai LAN-t egy nagy logikai LAN-ná összefogni. Megnézzük ennek az ellenkező-

jét is, azaz hogyan lehet egy fizikai LAN-t több logikai LAN-ként kezelni, a **VLAN**-nak (**Virtual LAN – virtuális LAN**) nevezett technikával. Mindkét technika a hálózatok menedzsmentjének rugalmasságát segíti elő. Kapcsolók, hidak és más, ezekhez kötődő területek átfogó tárgyalásával Seifert és Edwards [2008], valamint Perlman [2000] munkája foglalkozik.

4.8.1. Hidak használata

Mielőtt megismerkednénk a hidak kialakításával, nézzünk meg néhány olyan gyakori helyzetet, amelyben hidakat használnak. Három okot említünk meg, hogy miért lehet szüksége egyetlen szervezetnek több LAN-ra is.

Először is, sok egyetemi tanszék és vállalati szervezeti egység rendelkezik saját LAN-nal azért, hogy összekössék saját személyi számítógépeiket, kiszolgálóikat és egyéb eszközeiket (például nyomtatóikat). Mivel a különböző szervezeti egységek céljai is eltérőek, elképzelhető, hogy különböző típusú LAN-okat építettek ki, függetlenül más egységekétől. Előbb vagy utóbb azonban szükség lesz az együttműködésre, azaz hidakat kell alkalmazni. Ebben a példában tehát a hálózatok tulajdonosainak autonómiája eredményezte a több LAN létrejöttét.

A második esetben a szervezet földrajzilag szétszórtan, egymástól akár nagy távolságokra lévő épületekben helyezkedhet el. Olcsóbb lehet, ha az épületekben különálló LAN-okat telepítenek, és ezeket hidak és néhány hosszú fényvezető szál segítségével kötik össze ahelyett, hogy a kábeleket egyetlen központi kapcsolóba vezetnék. Még ha a kábelek lefektetése egyszerű feladat is, de a hosszukra vonatkozóan korlátozás van (például sodrott érpáras gigabites Ethernet esetén 200 m). A hálózat működésképtelen lesz, ha hosszabb kábelt használnak, a túl nagy jelgyengülés vagy körülfordulási idő miatt. Az egyetlen megoldás, ha feldaraboljuk a LAN-t, és a részeket hidakkal kötjük össze, hogy a teljes hálózat által lefedhető távolságot megnöveljük.

A harmadik eset az, amikor a terhelés megosztása indokolja az egyetlen LAN több (kapcsolók által összecsatolt) logikai szegmensre bontását. Több nagy egyetemen például több ezer munkaállomás áll rendelkezésre hallgatói és oktatói célokra. Cégeknek is lehet többezres alkalmazotti létszáma. A rendszer hatalmas mérete egyszerűen lehetetlenné teszi, hogy az összes munkaállomást egyetlen LAN-hoz kapcsolják, ugyanis több a számítógép, mint ahány port van egy (bármilyen) Ethernet-elosztón, és több állomás van, mint amennyi egyetlen klasszikus Etherneten megengedett.

Még ha lehetséges is lenne az, hogy az összes munkaállomást összekapcsolják, a több állomás csatlakoztatása egyetlen Ethernet-elosztóhoz vagy klasszikus Ethernethez nem növelné meg a kapacitást. Minden állomás ugyanazt a rögzített méretű sávszélességet használja. Minél több állomás van, annál kevesebb az állomásonkénti átlagos sávszélesség.

Akárhogy is, két különálló LAN kétszer akkora kapacitású, mint egy LAN. A hidak lehetővé teszik a LAN-ok összekapcsolását, miközben a kapacitásukat megtartják. A kulcs az, hogy ne küldjenek forgalmat olyan portokra, ahova nem szükséges, így minden LAN teljes sebességgel tud futni. Ez a viselkedés növeli a megbízhatóságot is, mivel egyetlen LAN esetén egy meghibásodott állomás, amely folyamatosan szemetet küldözget, eldugíthatja a teljes LAN-t. Azzal, hogy eldöntjük mit továbbítsunk, és mit ne, a hidak úgy

viselkednek, mint egy épületben a tűzbiztos ajtók, megelőzve azt, hogy egy megbolondult állomás működésképtelenné tegye az egész rendszert.

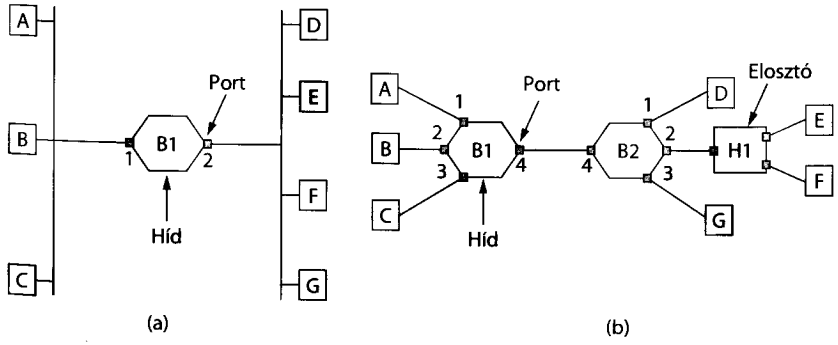
Hogy ezek az előnyök könnyen érvényre jussanak, ideális esetben a hidaknak tökéletesen átlátszóknak kell lenniük. Lehetségesnek kell lennie, hogy valaki elmenjen hidakat venni, csatlakoztassa a LAN-kábeleket a hidakhoz, és ezáltal minden azonnal tökéletesen működjön. Ne kelljen hardvert módosítani, ne kelljen szoftvert módosítani, ne kelljen címeket beállítani, ne kelljen útválasztó-táblázatokat vagy más paramétereket letölteni. Egyáltalán semmit se kelljen tenni, csak a kábeleket csatlakoztatni, majd az egészet magára lehessen hagyni. Továbbá, a létező LAN-ok működését egyáltalán nem befolyásolhatják a hidak. Ami az állomásokat illeti, nem lehet észrevehető különbség, hogy azok híddal összekapcsolt LAN vagy anélküli LAN részei. Az állomások mozgatásának ugyanolyan könnyűnek kell lenni a híddal összekapcsolt LAN-on belül, mint egyetlen LAN-on belül.

Elég meglepő, de lehetséges átlátszó hidak létrehozása. Erre két algoritmust használnak: a hátrafelé tanulás algoritmust, hogy megakadályozzák a forgalom olyan helyre küldését, ahova nem szükséges; és a feszítőfa algoritmust, hogy az összevissza kábelezés során esetlegesen kialakult hurkokat feltörjék. Nézzük meg ezeket az algoritmusokat, hogy rájöjjünk, hogyan megy végbe ez a varázslat.

4.8.2. Helyi hálózatok összekapcsolása

Egymáshoz kapcsolt hálózatok topológiájának két esetét mutatja a 4.41. ábra. A bal oldalon két többpontos LAN-t, mint amilyen a klasszikus Ethernet, egy hídnak nevezett speciális állomás kapcsol össze, amely mindkét LAN része. A jobb oldalon kétpontos kábelezésű, egy elosztót is tartalmazó LAN-ok vannak összekapcsolva. A hidak olyan eszközök, amelyekhez az állomások és az elosztó csatlakoznak. Ha a LAN megvalósítási technikája Ethernet, a hidat Ethernet-kapcsolónak nevezik.

A hidakat abban az időben fejlesztették ki, amikor a klasszikus Ethernet volt használatban, ezért gyakran ábrázolják többpontos kábellel, mint például a 4.41.(a) ábrán. Viszont minden topológia, amivel manapság találkozhatunk, kétpontos kábelekből és kapcsolókból áll. A hidak ugyanúgy működnek mindkét beállításban. Minden állomás,



4.41. ábra. *(a) Egy híd, amely két többpontos LAN-t kapcsol össze. (b) Hidak (és egy elosztó), amely hét kétpontos állomást kapcsol össze*

amely egy hídnak ugyanahhoz a portjához csatlakozik, ugyanabba az ütközési tartományba tartozik és minden port külön ütközési tartományt képvisel. Ha ugyanahhoz a porthoz több állomás csatlakozik (mint a klasszikus Ethernet esetén), vagy egy elosztó csatlakozik, amely több állomást csatlakoztat ugyanarra a portra, akkor a CSMA/CD-protokollt használják az esetleges ütközések feloldására és a keretek elküldésére.

Van viszont egy különbség abban, ahogy a híddal összekapcsolt LAN-okat építik. Egy híd többpontos LAN-ba építéséhez a hidat új állomásként kell csatlakoztatni mindkét többpontos LAN-hoz, ahogy a 4.41.(a) ábrán látható. Kétpontos LAN-ok híddal való összekapcsolásához, az elosztók csatlakoztathatók egy hídhoz, vagy inkább az elosztók helyettesíthetők híddal a teljesítőképesség növelése érdekében. A 4.41.(b) ábrán egy kivételével minden elosztót lecseréltek hidakra.

Különböző típusú kábeleket lehet egy hídhoz csatlakoztatni. Például a kábel, amely a *B1* és *B2* hidakat köti össze a 4.41.(b) ábrán, lehet egy hosszú fényvezető szál, míg az a kábel, amelyik az állomásokat köti össze a híddal, lehet rövid sodrott érpár. Ez az elrendezés különböző épültekben elhelyezkedő LAN-ok híddal történő összekapcsolásakor hasznos.

Most vizsgáljuk meg azt, hogy mi történik a híd belsejében. Minden híd válogatás nélküli üzemmódban (promiscuous mode) működik, azaz elfogad minden keretet, amely valamely portjára csatlakozó állomástól származik. Egy hídnak mindegyik keretről el kell döntenie, hogy vajon továbbítania vagy eldobnia kell, valamint ez előbbi esetben azt, hogy melyik porton küldje ki a keretet. A döntést a híd a célcím használatával hozhatja meg. Vegyük például a 4.41.(a) ábrán látható topológiát. Ha az *A* állomás küld keretet *B* állomásnak, akkor *B1* híd az 1-es portján fogja venni. A keretet rögtön eldobja minden további nélkül, mert már a megfelelő porton van. Ellenben a 4.41.(b) ábrán látható topológián, tegyük fel, hogy *A* küld egy keretet *D*-nek. A *B1* híd az 1-es portján fogja venni a keretet és a 4-es portján küldi ki. A *B2* híd ezután a keretet a 4-es portján fogja venni és az 1-es portján kiküldeni.

E séma megvalósításának egyszerű módja, hogy teszünk egy nagy (hash-struktúrájú) táblát a hídba. A tábla felsorolja az összes lehetséges célállomást, és mindegyikhez megadja a megfelelő kimeneti portot. Például a 4.41.(b) ábrán a *B1* híd táblája szerint *D* a 4-es porthoz tartozik, hiszen *B1*-nek csak annyit kell tudnia, hogy melyik portra továbbítsa a *D*-nek küldött kereteket. Valójában *B1*-et nem érdekli, hogy a keretet útja során még többször továbbítani kell, amikor az a *B2*-höz érkezik.

Amikor a hidakat először kapcsolják be, a tábláik üresek. Egyik híd sem tudja, hogy a célállomások merre helyezkednek el, így az elárasztásos algoritmust használják: minden bejövő keretet, amelynek címzettje ismeretlen, kiküldik az összes porton, kivéve azt, amelyikről érkezett. Ahogy telik az idő, a hidak lassan megtanulják, hogy merre találhatók a célállomások. Miután egy címzett állomás ismertté vált, a felé irányuló keretek csak a megfelelő porton kerülnek továbbításra, nem árasztják el a hálózatot.

Az az algoritmus, amit a hidak használnak, a **hátrafelé tanulás** (**backward learning**). Mint már említettük, a hidak válogatás nélküli üzemmódban működnek, így minden keretet látnak, amely a portjaikon megjelenik. Megvizsgálva a forráscímeket, megállapíthatják, hogy mely portokon mely állomások érhetők el. Például ha a 4.41.(b) ábrán a *B1* híd lát egy keretet a 3-as portján, amelyik a *C* állomástól származik, rájön, hogy *C* a 3-as portján keresztül érhető el, így készít a táblájában egy bejegyzést. *B1* minden olyan rákövetkező keretet, amely bármelyik portján *C*-nek érkezik, továbbítani fogja a 3-as portjára.

A hálózat topológiája változhat, ahogy állomásokat és hidakat üzembe vagy üzemen kívül helyeznek, vagy megváltoztatják az üzemelésük helyét. A dinamikusan változó topológia kezelése érdekében minden alkalommal, amikor létrejön egy táblabejegyzés, eltárolják a keret beérkezésének időpontját is. Amikor egy olyan keret érkezik, amelynek feladójáról helyes bejegyzés szerepel a táblában, az időinformációt az aktuális időponttal írják felül. Ilyen módon a tábla bejegyzéseihez rendelt időpontok megadják, hogy a híd mikor vett utoljára keretet az adott állomástól.

Egy folyamat a hídban időről időre végignézi a táblát, és törli onnan a néhány percnél régebbi bejegyzéseket. Ily módon elérhető, hogy kézi beavatkozás nélkül, néhány percen belül ismét visszaálljon egy olyan állomás normális működése, amelyet levettek a hálózatról, majd annak egy eltérő pontjára csatlakoztattak vissza. Az algoritmus azonban azt is maga után vonja, hogy ha egy állomás néhány percig csendben marad, a felé irányuló forgalom elárasztja a teljes rendszert addig, amíg az állomás maga nem küld végre egy keretet.

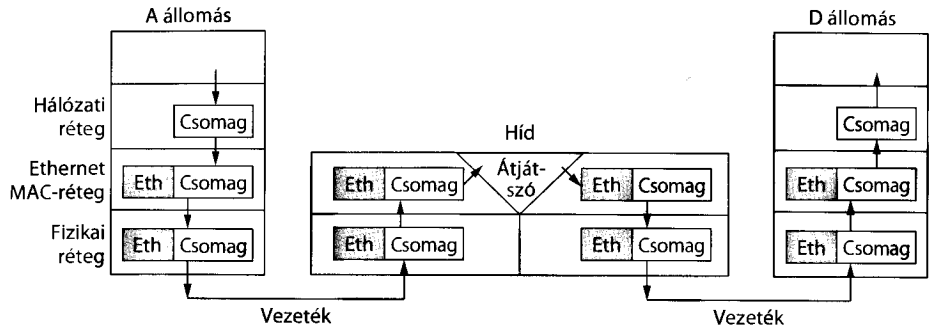
Egy beérkező keret útválasztása függ attól, hogy melyik porton érkezett (forrásport), valamint hogy mi a céljának a címe (célcím). Az eljárás a következő:

1. Ha a célcímhez tartozó port és a forrásport azonos, akkor a keretet el kell dobni.
2. Ha a célcímhez tartozó port és a forrásport különböző, akkor a keretet továbbítani kell a célporton.
3. Ha a célport ismeretlen, akkor elárasztást kell alkalmazni és a keretet a forrásport kivételével minden porton ki kell küldeni.

Lehet, hogy a kedves olvasó megdöbben azon, hogy fordulhat elő az első eset kétpontos adatkapcsolatok esetén. A válasz az, hogy úgy történhet meg, ha elosztókat használnak a számítógépek egy csoportjának a hídhoz való csatlakoztatására. A 4.41.(b) ábrán látható példában az *E* és *F* állomások a *H1* elosztóhoz csatlakoznak, ami pedig a *B2* hídhoz csatlakozik. Ha *E* küld egy keretet *F*-nek, az elosztó továbbítani fogja *B2*-nek is, nemcsak *F*-nek. Ez az, amit az elosztók csinálnak: összekötik az összes portot úgy, hogy egy porton bejövő keretet egyszerűen az összes többi porton kiküldi. A keret a *B2* 2-es portján fog megérkezni, ami már a címzett állomás felé néző kimenő port. A *B2* hídnak csak el kell dobnia a keretet.

Ahogy egy keret megérkezik, az algoritmust alkalmazni kell rá, ezért ez általában egy nagyon speciális célú VLSI-lapkán van megvalósítva. A lapka végzi a cím keresését és a tábla frissítését, mindezt néhány mikroszekundum alatt. Mivel a hidak csak a MAC-címeket vizsgálják egy keret továbbításakor, ezért lehetséges az, hogy a továbbítást elkezdjék, amint a célcímmező beérkezett, mielőtt a keret többi része megérkezik (feltéve természetesen, hogy a kimeneti vonal elérhető). Ez a megoldás csökkenti a keretnek azt a késleltetését, amely a hídon való áthaladásából ered, valamint csökkenti a keretek számát, amelyeket a híd pufferelni kényszerül. Erre **átvágó kapcsolóként** (**cut-through switching**) vagy **féreglyuk útválasztásként** (**wormhole routing**) hivatkoznak, és általában hardveresen valósítják meg.

A hidak működését most a protokollkészlet szemszögéből vizsgáljuk meg azért, hogy megértsük mit is jelent egy adatkapcsolati rétegbeli eszköznek lenni. Vegyünk egy



4.42. ábra. *Keret feldolgozása egy híd protokollkészletét tekintve*[9]

keretet, amelyet az *A* állomás küld *D* állomásnak a 4.41.(a) ábrán látható topológián, amelyben a LAN-ok Ethernetek. A keret át fog menni az egyik hídon. A feldolgozást a protokollkészlet figyelembevételével a 4.42. ábra mutatja be.

A csomag egy felsőbb rétegtől érkezik az Ethernet MAC-rétegbe. Kap egy Ethernet-fejrészt (és egy farokrészt is, de ezt az ábrán nem jelöltük). Ezt az egységet a MAC-réteg továbbadja a fizikai rétegbe, majd onnan kimegy, végig a kábelen, ahonnan a híd felszedi.

A híd fizikai rétege feladja a keretet az Ethernet MAC-rétegnek. Ez a réteg kiterjedtebb feldolgozási képességekkel rendelkezik, mint egy állomás Ethernet MAC-rétege. Ez továbbadja a keretet az átjátszónak (relay), ami szintén a MAC-réteg része. A híd átjátszó funkciója csak az Ethernet MAC-fejrészt használja, hogy megállapítsa, hogyan kezelje a keretet. Ebben az esetben, ez továbbítja a keretet annak az Ethernet MAC-rétegbeli portnak, amelyiken keresztül el lehet érni a *D* állomást, és a keret folytatja az útját.

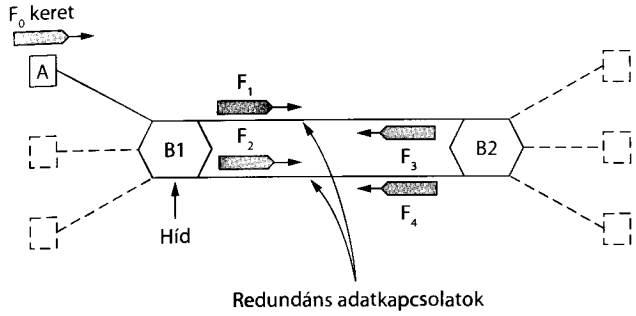
Általános esetben, egy adott rétegbeli átjátszó újraírhatja az adott rétegnek szánt fejrészt. Rövidesen a VLAN-oknál látunk erre példát. A híd semmilyen esetben sem nézhet bele a keret tartalmába és nem tudhatja meg, hogy IP-t szállít-e; ez teljesen érdektelen a híd adatfeldolgozása szempontjából, és megsértené a protokollrétegezést. Fontos még megjegyezni, hogy egy *k* portos hídnak *k* darab MAC- és fizikai rétege van. A *k* értéke a mi példánkban 2.

4.8.3. Feszítőfás hidak

A megbízhatóság növelése érdekében redundáns adatkapcsolatokat lehet használni a hidak között. A 4.43. ábrán látható példán két párhuzamos adatkapcsolat van a két híd között. Ez a felépítés biztosítja, hogy ha az egyik adatkapcsolatot elvágják, a hálózat nem szakad két számítógép-csoportra, amelyek nem tudnak egymással beszélgetni.

Ez a redundancia azonban okoz néhány új problémát, mert hurkokat hoz létre a topológiában. Jobban megérthetjük ezt egy példán keresztül, ha megvizsgáljuk, hogyan

9 Az ábrán a fizikai réteg adatszerkezete hibás, mert ott struktúra nélküli bitsorozatnak kellene lenni. *(A lektor megjegyzése)*



4.43. ábra. *Hidak két párhuzamosan futó adatkapcsolattal*

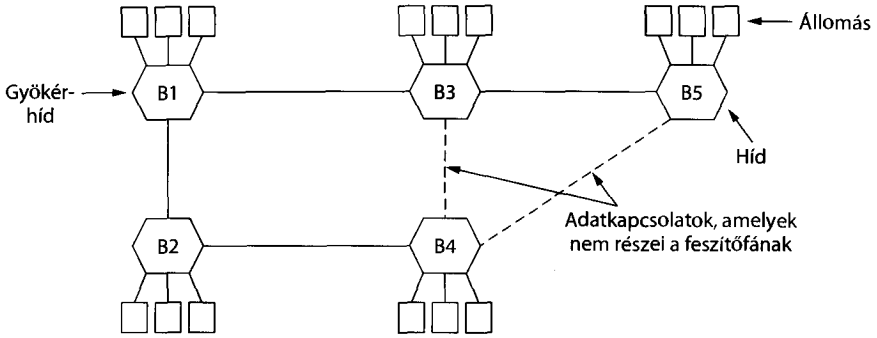
történik *A* kereteinek küldése egy korábban nem látott célállomásnak a 4.43. ábrán. Minden híd az ismeretlen célállomások kezelésénél megszokott szabályt követi, azaz a kerettel elárasztja a hálózatot. Nevezzük F_0-nak azt a keretet, amely *A*-tól jön és eléri a *B1* hidat. A híd ennek a keretnek a másolatait a többi portján kiküldi. Csak azokat a portokat vesszük figyelembe, amelyek *B1*-et és *B2*-t kötik össze (annak ellenére, hogy a keret a többi porton is kiküldésre kerül). Mivel két adatkapcsolat van *B1* és *B2* között, a keret két példánya fogja elérni a *B2*-t. Ezeket F_1-gyel és F_2-vel jelöltük a 4.43. ábrán.

Nem sokkal később, a *B2* híd veszi ezeket a kereteket. Viszont nem tudja (és nem is tudhatja), hogy ezek ugyanannak a keretnek a példányai, hanem azt gondolja, hogy két különböző keretet küldtek egymás után. Így a *B2* híd fogja F_1-et és a másolatait kiküldi a többi portján, ugyanígy tesz F_2-vel. Ezzel létrejön az F_3 és az F_4 keret, amelyeket a két adatkapcsolaton visszaküldenek *B1*-nek. Ezután *B1* lát két keretet ismeretlen célcímekkel, és újra lemásolja őket. Ez a ciklus a végtelenségig folytatódik.

A fenti nehézség úgy küszöbölhető ki, hogy a hidak kommunikálnak egymással, és lefedik a hálózat aktuális topológiáját egy olyan feszítőfával (spanning tree), amely eléri az összes hidat. Tulajdonképpen annyi a teendő, hogy bizonyos adatkapcsolatokat figyelmen kívül kell hagyni, hogy egy képzeletbeli, hurokmentes topológia jöjjön létre, amely az aktuális topológia egy részhalmaza.

Vegyük például a 4.44. ábrát, amelyen öt összekapcsolt híd és a hozzájuk csatlakozó állomások láthatók. Minden állomás csak egy hídhoz kapcsolódik. Vannak redundáns adatkapcsolatok a hidak között, tehát ha az összes adatkapcsolat használatban van, akkor a kereteket körbe-körbe fogják küldeni. Erre a topológiára tekinthetünk úgy, mint egy gráfra, amelyben a hidak képezik a csúcsokat és a kétpontos adatkapcsolatok az éleket. A gráfot egy feszítőfává csökkenthetjük, amely definíciószerűen hurokmentes azáltal, hogy a 4.44. ábrán látható szaggatott vonalakat kiejtjük. Ezt a feszítőfát használva pontosan egyetlen útvonal vezet bármelyik két állomás között. Miután a hidak megállapodtak a feszítőfában, az állomások közötti összes továbbítás ezt a fát követi. Mivel így minden forrás és cél között csak egyetlen útvonal létezik, nem jöhetnek létre hurkok.

A feszítőfa felépítéséhez a hidaknak egy elosztott algoritmust kell futtatniuk. Minden híd periodikusan konfigurációs üzenetet küld az összes portján a szomszédainak, és feldolgozza a más hidaktól vett üzeneteket a következők szerint. Ezeket az üzeneteket nem továbbítják, mert az a feladatuk, hogy egy fát építsenek, amit majd továbbításra lehet használni.



4.44. ábra. *Egy feszítőfa, amely öt hidat köt össze. A szaggatott vonalak nem részei a feszítőfának*

A hidaknak maguk közül ki kell választaniuk egyet, amely a feszítőfa gyökere lesz. A kiválasztás érdekében mindegyik állomás beletesz a konfigurációs üzenetébe egy, a MAC-címükön alapuló azonosítót, valamint annak a hídnak az azonosítóját, amelyet a gyökérnek gondolnak. A MAC-címeket a gyártók állítják be és az egész világon egyediek, ezért ezek az azonosítók is egyediek lesznek, és így kényelmes a használatuk. A hidak a legkisebb azonosítójú hidat választják gyökérnek. Miután a hírek elterjesztéséhez elég üzenetet cseréltek, a hidak megegyeznek, hogy közülük melyik legyen a gyökér. A 4.44. ábrán, a *B1* hídnak van a legkisebb azonosítója, ezért az lesz a gyökér.

A következő lépésben a gyökértől minden egyes hídhoz menő legrövidebb útvonalat kijelölő fa meghatározása történik. A 4.44. ábrán a *B2* és *B3* hidakat *B1*-től közvetlenül egy ugrással el lehet érni, ami a legrövidebb út. A *B4* híd két ugrással érhető el, vagy a *B2*-n vagy a *B3*-on keresztül. Hogy a döntetlent feloldják, a kisebb azonosítójú hídon keresztül menő utat választják, tehát a *B4* a *B2*-n keresztül érhető el. A *B5* híd két ugrással elérhető *B3*-on keresztül.

Hogy a hidak a legrövidebb útvonalakat megtalálják, a gyökértől való távolságukat belerakják a konfigurációs üzenetükbe. Minden híd megjegyzi a gyökérhez vezető legrövidebb utat. Ezután a hidak lekapcsolják azokat a portjaikat, amelyek nem részei a legrövidebb útnak.

Igaz, hogy a fa lefedi az összes hidat, de nem biztos, hogy az összes adatkapcsolat (vagy akár híd[10]) szerepel benne. Ez azért lehetséges, mivel néhány port kiiktatásával néhány adatkapcsolatot lenyesnek a hálózatról a hurkok elkerülésére. Miután a feszítőfa elkészült, az algoritmus tovább fut, hogy a hálózat topológiai változásait automatikusan észlelhesse, és a fát bármikor frissíthesse.

A feszítőfa kialakítására alkalmazott algoritmust Radia Perlman fejlesztette ki. Az volt a feladata, hogy oldja meg több LAN összekapcsolását hurkok nélkül. Kapott egy hetet, hogy megcsinálja, de ő egy nap alatt kidolgozta az algoritmust. Szerencsére maradt ideje arra, hogy verses formában is megírja [Perlman, 1985].

10 Ez egy önellentmondás. De ha a műszaki részleteket nézzük, a protokoll lehetővé teszi ezt. Ez a nem szándékolt működés akkor fordulhat elő, ha túl sok híd van a hálózatban, és két vagy több fa alakul ki, melyek nem fedik egymást. *(A fordító megjegyzése)*

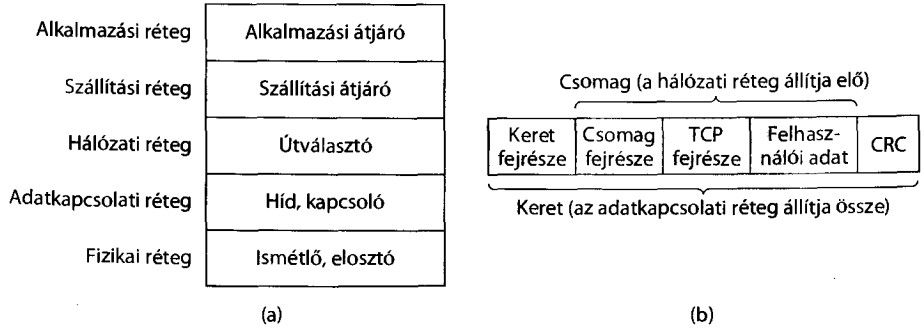
Úgy vélem többé nem látok
egy fánál remekebb gráfot.
A fák mind jellegzetesen
teljesen hurokmentesek.
Mivel a fák feszítenek,
csomag elér minden szegmenst.
A gyökeret úgy választják,
számaikat sorbarakják.
Gyökértől a legjobb utak
a fában érvényre jutnak.
Csinálok egy hálót, melyben
hidak feszítőfát lelnek.

A feszítőfa algoritmust IEEE 802.1D néven szabványosították, és már sok éve használatban van. 2001-ben újraírták, hogy topológiaváltozás után gyorsabban találjon egy új feszítőfát, melyet Perlman [2000] munkája részletez.

4.8.4. Ismétlők, elosztók, hidak, kapcsolók, útválasztók és átjárók

A könyvben ez idáig számos módját láthattuk annak, hogyan lehet kereteket és csomagokat egyik számítógépről a másikra átvinni. Szó volt az ismétlőkről, az elosztókról, a hidakról, a kapcsolókról, az útválasztókról és az átjárókról is. Mindegyik eszközt széles körben használják, de többé-kevésbé eltérnek egymástól. Mivel ilyen sokan vannak, talán érdemes egyszer együtt is megvizsgálni őket, hogy lássuk, miben hasonlítanak és miben különböznek.

A legfontosabb azt felismerni, hogy ezek az eszközök különböző rétegekben működnek, ahogy azt a 4.45.(a) ábra is szemlélteti. Azért lényeges, hogy melyik rétegről van szó, mert a különböző eszközök más-más információt használnak fel annak eldöntésére, hogyan kapcsoljanak. A tipikus felállás az, hogy a felhasználó előállít valamilyen adatot, amelyet egy távoli gépnek kíván küldeni. Ez az adat átkerül a szállítási rétegbe, ami hozzáad egy (például TCP) fejrészt, ami az így kapott adategységet továbbadja a hálózati ré-



4.45. ábra. *(a) Az egyes eszközök és a rétegek, ahol megtalálhatók. (b) Keretek, csomagok és fejrészek*

tegnek. A hálózati réteg is hozzáadja a saját fejrészét, ezzel egy hálózati rétegbeli (például IP) csomagot alakít ki. A 4.45.(b) ábrán az IP-csomagot szürke árnyékolással jelöltük. A csomag ezt követően az adatkapcsolati réteghez kerül, mely hozzáteszi saját fejrészét és ellenőrző összegét, majd az így kapott keretet átadja a fizikai rétegnek (például egy LAN-on keresztüli) átvitelre.

Tekintsük most a kapcsolóeszközöket, és nézzük meg, hogyan viszonyulnak a csomagokhoz és a keretekhez! Legalul, a fizikai rétegben találjuk az *ismétlőket*. Ezek analóg eszközök, melyek a hozzájuk csatlakoztatott kábelek jeleivel dolgoznak. Az ismétlő az egyik kábelszegmensen megjelenő jelet megtisztítja, felerősíti és átrakja a másikra. Az ismétlők nem ismerik a kereteket, a csomagokat vagy a fejrészeket. A bitek helyett csak azok voltban mért feszültségszintjeit értik. A klasszikus Ethernetet például úgy tervezték, hogy négy ismétlőt lehessen használni, amelyek megnövelik a jel teljesítményét azért, hogy a maximális kábelhosszt 500 méterről 2500 méterre lehessen kiterjeszteni.

Következő eszközünk az *elosztó (hub)*. Az elosztónak számos bemeneti vonala van, amelyekre egyszerűen villamosan lehet csatlakozni. A bármelyik vonalon beérkező keretek az összes többi vonalon kivitelre kerülnek. Ha két keret egyszerre érkezik, akkor ütközni fognak, pontosan úgy, mint egy koaxiális kábelen. Az elosztóba érkező vonalaknak ugyanazzal a sebességgel kell működniük. Az elosztók annyiban különböznek az ismétlőktől, hogy (általában) nem erősítik a beérkező jeleket, és olyan a kialakításuk, hogy több bemenetük lehet, ezek a különbségek azonban nem túl jelentősek. Az ismétlőkhöz hasonlóan az elosztók is fizikai rétegbeli eszközök, melyek nem vizsgálják vagy használják az adatkapcsolati címeket.

Lépjünk most feljebb az adatkapcsolati rétegbe, ahol a *hidakat* és a *kapcsolókat* találjuk A hidakat nemrég tanulmányoztuk. Egy híd kettő vagy több LAN-t köt össze. Az elosztókhoz hasonlóan a modern hidaknak is több portjuk van, rendszerint elég 4–48 darab adott típusú bemeneti vonal számára. Az elosztóktól eltérően, itt minden port el van szigetelve, hogy különálló ütközési tartományt alakítson ki. Ha egy port duplex kétpontos adatkapcsolat, akkor a CSMA/CD-algoritmusra nincs szüksége. Amikor megérkezik egy keret, a híd szoftvere kiolvassa a célcímet a keret fejrészéből, és egy táblázatból meghatározza, hogy hová kell küldeni az adott keretet. Az Ethernet esetében ez a cím a 4.14. ábrán látható 48 bites célcím. A híd csak a megfelelő portra teszi ki a keretet, és több keretet tud továbbítani egyszerre.

A hidak jobb teljesítményt nyújtanak az elosztóknál, és a portok elszigeteltsége azt jelenti, hogy a bemeneti vonalak különböző sebességen, és valószínűleg még különböző típusú hálózaton is képesek futni. Ennek gyakori példája az olyan híd, amelynek a portjaihoz 10, 100 és 1000 Mb/s-os Ethernet csatlakozik. A hídban szükség van pufferelésre, hogy fogadni tudjon egy keretet az egyik porton, és egy keretet továbbítani egy másikon. Ha a keretek gyorsabban érkeznek be, mint ahogy a kapcsoló újra ki tudná küldeni azokat, elfogyhat a pufferterület, és el kell dobnia a kereteket. Például, ha egy gigabites Ethernet teljes sebességgel zúdítja a bitjeit egy 10 Mb/s-os Ethernetre, a hídnak pufferelnie kell azokat, miközben reméli, hogy lesz elég memóriája hozzá. Ez a probléma akkor is fennáll, ha a portok megegyező sebességgel futnak, és több port küld ugyanarra a célportra kereteket.

A hidakat eredetileg különböző típusú LAN-ok összekapcsolására szánták, például egy Ethernetet és egy vezérjeles gyűrűt. Ez, bárhogy nézzük, soha nem működött jól a

LAN-ok közötti különbségek miatt. A különböző formátumok megkövetelik a másolást, és az újraformázást, ami processzoridőt vesz el, valamint új ellenőrző összeg számítását igényli, és lehetőséget ad rejtett hibákra is, amelyeket a híd memóriájában lévő rossz bitek eredményeznek. A különböző legnagyobb megengedett kerethosszúságok eltérése szintén komoly problémát jelent, melyre nincs hatékony megoldás. Alapvetően azokat a kereteket, amelyek túl nagyok ahhoz, hogy továbbítani lehessen, el kell dobni. Ennyit az átlátszóságról.

Két másik területen különbözhetnek még a LAN-ok, ezek a biztonság és a szolgáltatásminőség. Néhány LAN, például a 802.11, rendelkezik adatkapcsolati rétegbeli titkosítással, ellenben néhány LAN nem, ilyen például az Ethernet. Néhány LAN-nak van olyan szolgáltatásminőségi képessége, mint amilyen a prioritás (például a 802.11), ellenben néhánynak nincs, ilyen például az Ethernet. Ennek következtében, ha egy keret ezeken a LAN-okon át utazik, akkor ezek nem tudják nyújtani a küldő által elvárt biztonságot és szolgáltatásminőséget. Ezen okok miatt a modern hidak általában egyetlen típusú hálózaton működnek, és az útválasztókat (melyeket alább áttekintünk) használják a különböző típusú hálózatok összekapcsolására.

A *kapcsolók* a modern hidak másik elnevezése. A különbségeknek több köze van a marketinges, mint a műszaki dolgokhoz, de néhány eltérést fontos megjegyezni. A hidakat akkor fejlesztették ki, amikor a klasszikus Ethernet még használatban volt, ezért általában viszonylag kevés LAN-t kapcsolnak össze, és ezért viszonylag kevés portjuk van. Manapság a kapcsoló elnevezés a népszerűbb. A modern hálózatok kétpontos adatkapcsolatokat használnak, mint amilyen a sodrott érpár, így a számítógépeket egyesével közvetlenül a kapcsolóba dugják, és ezért a kapcsolóknak gyakran sok portjuk van. Végül, a kapcsoló kifejezést általános értelemben is használják. A híd funkcionalitása egyértelmű. A kapcsoló ellenben vonatkozhat egy Ethernet-kapcsolóra vagy egy teljesen más eszközre, amely továbbítási döntést hoz, például egy telefonközpontra.

Láthattuk eddig az ismétlőket és az elosztókat, melyek eléggé hasonlók, valamint a hidakat és a kapcsolókat, melyek még jobban hasonlítanak egymáshoz. Most továbblépünk az *útválasztókhoz*, melyek az összes eddigi eszköztől különböznek. Amikor egy csomag megérkezik egy útválasztóhoz, akkor a keret fej- és farokrészét eltávolítják, és a keret adatmezejében található csomagot (amit a 4.45. ábrán sötéttel jelöltünk) átadják az útválasztó szoftvernek. Ez a szoftver a csomag fejrészének segítségével választja ki a megfelelő kimeneti vonalat. IP-csomag esetén a csomag fejrésze nem a 48 bites IEEE 802-es címet, hanem a 32 bites (IPv4) vagy 128 bites (IPv6) címet tartalmazza. Az útválasztó szoftver nem látja a keretben levő címeket, és azt sem tudja, hogy a csomag egy LAN-on vagy egy kétpontos vonalon érkezett-e be. Az útválasztókat és az útválasztást az 5. fejezetben tárgyaljuk.

Egy réteggel feljebb találjuk a *szállítási átjárókat*. Ezek két olyan számítógépet kötnek össze, melyek eltérő összeköttetés-alapú szállítási protokollt használnak. Tegyük fel például, hogy egy összeköttetés-alapú TCP/IP-protokollt használó gép egy másik, összeköttetés-alapú SCTP-nek nevezett szállítási protokollt használó géppel szeretne kommunikálni. Ekkor a szállítási átjáró átmásolhatja a csomagokat az egyik összeköttetésről a másikra, miközben a szükséges módon átalakítja azok formátumát.

Végül, az alkalmazási átjárók az adatok formátumát és tartalmát is megértik, és képesek lefordítani az üzeneteket az egyik formátumról a másikra. Egy e-levél átjáró például

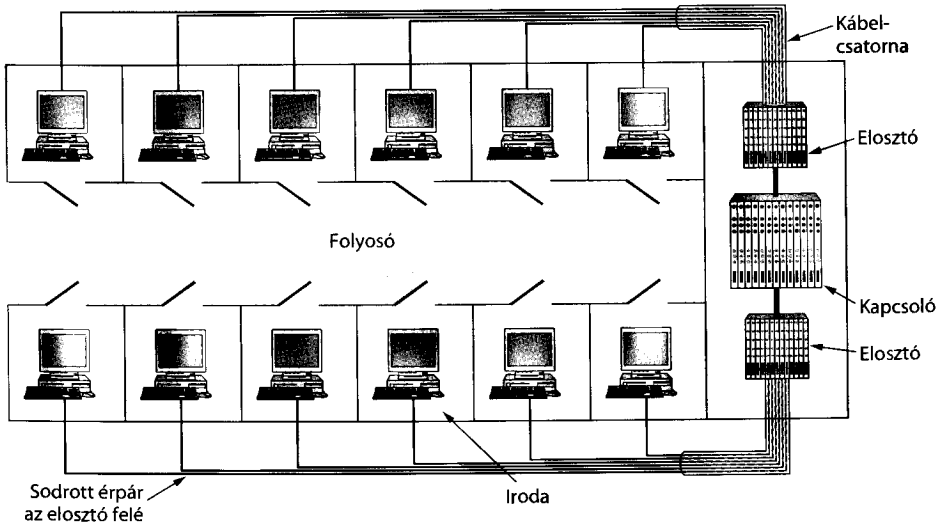
az internetes üzeneteket SMS-üzenetekre fordíthatja a mobiltelefonok számára. A kapcsolóhoz hasonlóan az átjáró is némileg általános kifejezés, mely utalhat egy felső rétegben futó továbbítási folyamatra.

4.8.5. Virtuális LAN-ok

A helyi hálózatok korai időszakában vastag sárga kábelek kígyóztak végig az irodaépületek kábelcsatornáin. Minden számítógép, amely mellett elhaladtak, ezekre csatlakozott. Senki nem foglalkozott azzal, hogy melyik gép melyik LAN-hoz tartozik. Az egymás melletti irodák összes felhasználója ugyanazt a LAN-t használta, akár összetartoztak, akár nem. Az elhelyezkedés fontosabbnak bizonyult a cég szervezeti felépítésénél.

Az 1990-es években a sodrott érpár és az elosztók megjelenésével mindez megváltozott. Az épületeket (jelentős kiadások árán) újrakábelezték, hogy megszabaduljanak az összes sárga kerti locsolótömlőtől, és minden irodából sodrott érpárokat vezettek a folyosók végén vagy a központi gépteremben lévő kábelrendezőkhöz, ahogy azt a 4.46. ábra is szemlélteti. Ha a kábelezésért felelős alelnök történetesen látnok volt, akkor 5-ös kategóriájú sodrott érpárokat telepítettek; de ha szűkmarkú volt, akkor a meglevő (3-as kategóriájú) telefonvezetékeket használták (aztán pár év múlva azokat is le lehetett cserélni, amikor a gyors Ethernet megjelent).

Manapság a kábelek megváltoztak és az elosztókból kapcsolók lettek, de a kábelezési minta maradt a régi. Ez a minta lehetővé tette, hogy a LAN-okat ne fizikai, hanem logikai úton alakítsák ki. Például, ha egy vállalat k darab LAN-t akar, akkor vásárolhat k darab kapcsolót. Ha kellő gondot fordítanak annak eldöntésére, hogy melyik csatlakozót melyik elosztóba dugják, akkor a LAN-ok tagjait az elhelyezkedéstől viszonylag függetlenül, a szervezeti felépítésnek megfelelően lehet megválasztani.



4.46. ábra. *Épület központi kábelezéssel, elosztókkal és egy kapcsolóval*

Egyáltalán számít az, hogy ki melyik LAN-on van? Végül is majdnem minden szervezetnél össze van kötve az összes LAN. A rövid válasz mégis az: igen, gyakran számít. A rendszergazdák több okból kifolyólag is szeretik, ha a LAN-okon lévő felhasználói csoportok nem az épület fizikai elrendezését, hanem az intézmény szervezeti rendjét tükrözik. Az első ok a biztonság. Egy LAN-on lehetnek webszerverek és más számítógépek, melyeket nyilvános használatra szánnak. Egy másik LAN-on lehetnek olyan számítógépek, amelyek a személyzeti osztály adatait tárolják, ezek nem kerülhetnek az osztályon kívülre. Ilyen helyzetben ésszerű dolog az osztály összes számítógépét egyetlen LAN-ra tenni, és megtiltani, hogy bármelyik szerverhez a LAN-on kívülről hozzáférjenek. A vállalati vezetés ráncolni szokta a homlokát, amikor azt hallja, hogy egy ilyen elrendezést lehetetlen kialakítani.

A második gondot a terhelés jelenti. Egyes LAN-okat sokkal intenzívebben használnak, mint másokat, és olykor kívánatos lehet az ilyeneket elkülöníteni. Ha például a kutatási osztályon dolgozók mindenféle remek kísérletet folytatnak, amelyek felett néha elvesztik az uralmukat és a LAN-juk telítődik, akkor a vezetőség tagjai nem fognak lelkesedni azért, hogy azzal a hálózati kapacitással segítsék ki a kollégákat, melyet éppen videokonferenciára használnának. Ez újra azt a benyomást kelti a vezetőségben, hogy gyorsabb hálózat telepítése szükséges.

A harmadik az adatszórásos forgalom kérdése. A hidak adatszórással[11] továbbítják a kereteket, amikor a címzett állomás elhelyezkedése nem ismert, és sok felsőbb rétegbeli protokoll is használja az adatszórást. Például ha egy felhasználó egy csomagot szeretne küldeni az x IP-címre, akkor honnan fogja tudni, hogy milyen MAC-címet rakjon a keretbe? Ezzel a kérdéssel az 5. fejezetben fogunk foglalkozni, de a válasz röviden az, hogy adatszórással szétküld egy keretet, mely azt a kérdést tartalmazza: „Kié az x cím?", majd várja a választ. Ahogy egy LAN-on a számítógépek száma növekszik, úgy nő az adatszórások száma is. Minden adatszórás nagyobb mértékben használja a LAN kapacitását, mint a szabályos keretforgalom, mert a forgalmat a LAN minden számítógépének el kell juttatni. Ha a LAN-okat sikerül a lehető legkisebb méretűnek megtartani, az adatszórásos forgalom hatása csökkenthető.

Az adatszórással függ össze az a probléma is, hogy néha az is előfordulhat, hogy egy hálózati illesztő meghibásodik vagy rosszul konfigurálják, és adatszórással végtelen keretfolyamot állít elő. Ha a hálózat tényleg szerencsétlen, akkor ezek a keretek válaszra ösztönöznek gépeket, amely egyre növekvő forgalomhoz vezet. Az ilyen **adatszórási vihar** (**broadcast** storm) eredménye az, hogy (1) ezek a keretek lekötik a LAN teljes kapacitását és (2) az összes összekapcsolt LAN-on lévő gépet megbénítja az adatszórt keretek feldolgozása és eldobása.

Első látásra úgy tűnhet, hogy az adatszórási viharok kiterjedését korlátozni lehetne a LAN-ok hidakkal vagy kapcsolókkal történő szétválasztásával, de ha az átlátszóság elérése a cél (azaz, hogy egy gépet át lehessen vinni a hídon egy másik LAN-ra anélkül, hogy azt bárki is észrevenné), akkor a hidaknak az adatszórással elküldött kereteket is továbbítaniuk kell.

11 Itt a szerző pontatlan, mert az adatszórást az különbözteti meg az elárasztástól, hogy a célcím csupa 1-es. De a hidak nem változtatják meg a fejlécet. Tehát a hidak ilyenkor elárasztják a hálózatot keretekkel. *(A fordító megjegyzése)*

Most, hogy láttuk, miért akarhatnak a vállalatok több, korlátozott kiterjedésű LAN-t kiépíteni, térjünk vissza a logikai és a fizikai topológia szétválasztásának problémájára. A szervezeti felépítést tükröző fizikai topológia építése munka- és költségigényes, még akkor is, ha központosított vezetékezést és kapcsolókat alkalmaznak. Például, ha ugyanannak a részlegnek két alkalmazottja különböző épületben dolgozik, könnyebb lehet őket különböző kapcsolókhoz csatlakoztatni, amelyek különböző LAN-okhoz tartoznak. Még ha nem is ez a helyzet, a vállalat egyik dolgozója átkerülhet az egyik részlegből a másikba anélkül, hogy kiköltözne az irodájából; vagy éppen átköltözik egy másik irodába anélkül, hogy átkerülne egy másik részlegbe. Ennek eredményeképpen előfordulhat, hogy a dolgozó nem a megfelelő LAN-on van addig, amíg a rendszergazda a dolgozó kábelét az egyik kapcsolóból a másikba át nem dugja. Továbbá lehetséges, hogy az egy részlegbe tartozó számítógépek száma nem jól illeszkedik a kapcsoló portjainak számához, esetleg néhány részleg túl kicsi, míg mások annyira nagyok, hogy több kapcsolóra van szükségük. Ez a kihasználatlan portok miatt a kapcsolók portjainak pazarlásához vezet.

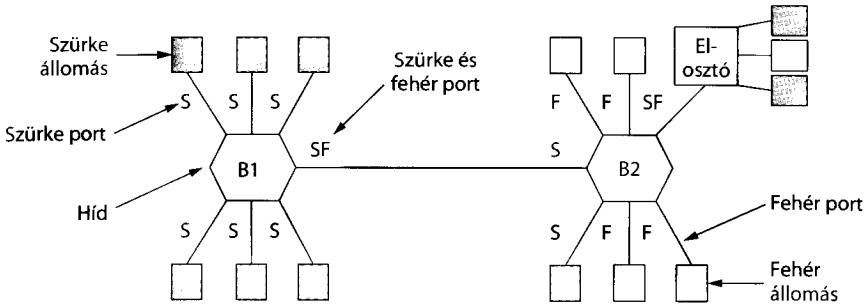
Sok vállalatnál állandóan szervezeti változások történnek, ami azt jelenti, hogy a rendszergazdák rengeteg időt töltenek azzal, hogy csatlakozókat húznak ki az egyik helyről, és máshová dugják azokat vissza. Ráadásul bizonyos esetekben az áthelyezés egyáltalán nem megoldható, mert a felhasználó gépéből kijövő sodrott érpár túl messze van a megfelelő elosztótól (például másik épületben van), vagy a kapcsolón rendelkezésre álló portok nem a megfelelő LAN-hoz tartoznak.

A hálózati gyártók a nagyobb rugalmasságot megcélzó vevői igények kielégítésére egy olyan megoldás kidolgozásába kezdtek, melynek segítségével az épületeket teljes egészében szoftveres úton lehet átkábelezni. A munka eredménye a **VLAN** (**Virtual LAN – virtuális LAN**) nevű elgondolás lett. Ezt még az IEEE 802-es bizottság is szabványosította és most már széles körben alkalmazzák sok szervezetnél. Vessünk hát egy pillantást erre. A VLAN-okról további információkkal szolgál Seifert és Edwards [2008] munkája.

A VLAN-ok a VLAN-képes kapcsolókon alapulnak. Egy VLAN-alapú hálózat kiépítésekor a rendszergazda eldönti, hogy hány VLAN-t fog használni, melyik gép melyik VLAN-on lesz, és hogy mi lesz az egyes VLAN-ok neve. A VLAN-okat gyakran (nem hivatalosan) színekről nevezik el, mivel így színes ábrát lehet nyomtatni, melyek a gépek fizikai elrendezését mutatják, és a piros LAN tagjait pirossal, a zöld tagjait zölddel jelölik és így tovább. Ily módon egyetlen képen látható mind a logikai, mind a fizikai elrendezés.

Példaként tekintsük a 4.47. ábrán látható kapcsolt LAN-t, ahol az S (szürke) VLAN-hoz kilenc, az F (fehér) VLAN-hoz pedig öt gép tartozik. A szürke VLAN-hoz tartozó számítógépek a két kapcsolón elszórva helyezkednek el, beleértve azt a két számítógépet is, amelyek egy elosztón keresztül csatlakoznak a kapcsolóhoz.

Ahhoz, hogy a VLAN-ok helyesen működjenek, konfigurációs táblázatokat kell feláll ítani a kapcsolókban. Ezek a táblázatok azt mondják meg, hogy az egyes VLAN-okat melyik port segítségével lehet elérni. Ha beérkezik egy keret, mondjuk a szürke VLAN-ról, akkor azt minden S jelű portra továbbítani kell. Ez egyaránt vonatkozik a hagyományos (azaz egyesküldéses) forgalomra, ahol a címzett állomás helyét a hidak még nem tanulták meg, valamint a többesküldéses és adatszórásos forgalomra is. Figyeljük meg, hogy egy portot több VLAN-színnel is fel lehet címkézni.



4.47. ábra. *Két VLAN (szürke és fehér) egy kapcsolt LAN-on*

Példaként tételezzük fel, hogy a 4.47. ábrán látható, *B1* hídhoz csatlakozó, egyik szürke állomás egy keretet küld egy olyan címzett állomásnak, amelyet még nem láttak korábban. A *B1* híd meg fogja kapni ezt a keretet, és látni fogja, hogy szürke VLAN-on lévő géptől érkezett, ezért azt minden S-sel címkézett (kivéve a bejövő) portjára árasztja. A keretet elküldi öt másik szürke állomásnak, amely a *B1*-hez kapcsolódik, valamint *B2*-nek a *B1* és *B2* közötti adatkapcsolaton. A *B2* hídon a keret, az előzőekhez hasonlóan, az összes S-sel jelölt porton továbbítódik. Ezzel a keret eljut még egy állomásig és az elosztóig (amely a keretet az összes állomásának továbbítja). Az elosztónak két címkéje is van, mert mindkét VLAN-hoz tartozó állomás csatlakozik hozzá. A kereteket nem küldik ki S-sel nem címkézett portokon, mert a híd tudja, hogy azokon a portokon keresztül nincs elérhető szürke VLAN-ba tartozó számítógép.

A példánkban, a keretet csak azért küldték el *B1* hídtól *B2* hídnak, mert a szürke VLAN-ba tartozó számítógépek kapcsolódnak *B2*-höz. A fehér VLAN-t megvizsgálva láthatjuk, hogy a *B2* híd *B1* irányába néző portja *nincs* F-fel felcímkézve. Ez azt jelenti, hogy a fehér VLAN egy kerete sem fog a *B2* hídtól a *B1* hídnak továbbítódni. Ez a helyes működés, mert fehér VLAN-hoz tartozó állomások nem kapcsolódnak a *B1*-hez.

Az IEEE 802.1Q szabvány

Ennek a sémának a megvalósításához az szükséges, hogy a hidak tudják, hogy a beérkező keretek melyik VLAN-hoz tartoznak. Enélkül az információ nélkül például a 4.47. ábrán látható *B2* híd nem tudhatná, hogy a *B1* híd felől érkező keretet a szürke vagy a fehér VLAN-on továbbítsa. Ha egy újfajta LAN-t terveznénk, elég könnyű lenne hozzáadni egy VLAN-mezőt a fejrészhez. De mit lehet tenni a leggyakoribb LAN-nal, az Ethernettel, amelynek nem volt semmilyen tartalék mezője egy VLAN-azonosító számára?

Az IEEE 802-es bizottsága 1995-ben szembesült ezzel a problémával. Hosszas viták után megtették az elképzelhetetlent: megváltoztatták az Ethernet-fejrészt. Az új formátumot a **802.1Q** IEEE-szabványban adták ki 1998-ban. Ez már tartalmaz egy VLAN-címkét, melyet most röviden bemutatunk. Nem meglepő, hogy nem volt teljesen triviális dolog valami olyasmit megváltoztatni, ami mindenhol olyan jól meghonosodott, mint az Ethernet-fejrész. Íme, néhány a felmerülő kérdések közül:

1. Ki kell majd dobni a több százmillió meglevő Ethernet-kártyát?

2. Ha nem, akkor ki fogja előállítani az új mezőket?

3. Mi lesz azokkal a keretekkel, amelyek már e nélkül is elérték a maximális keretméretet?

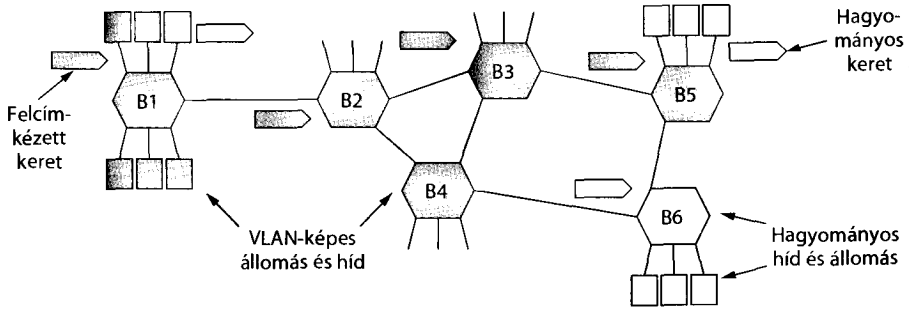
Persze a 802-es bizottság (nagyon is) tudatában volt ezeknek a problémáknak, és tudta, hogy mindegyikre megoldást kell találnia – amit meg is tett.

A megoldás kulcsa az a felismerés, hogy a VLAN-mezőket ténylegesen csak a hidak és a kapcsolók használják, *nem* pedig a felhasználók gépei. Így a 4.47. ábra esetében nem feltétlenül szükséges, hogy a végállomások felé vezető vonalakon is megjelenjenek az ilyen mezők; elég, ha a hidak között menő vonalakon szerepelnek. Tehát a VLAN-ok használatához a hidaknak kell VLAN-képesnek lenniük. Ez a tény teszi a tervet megvalósíthatóvá.

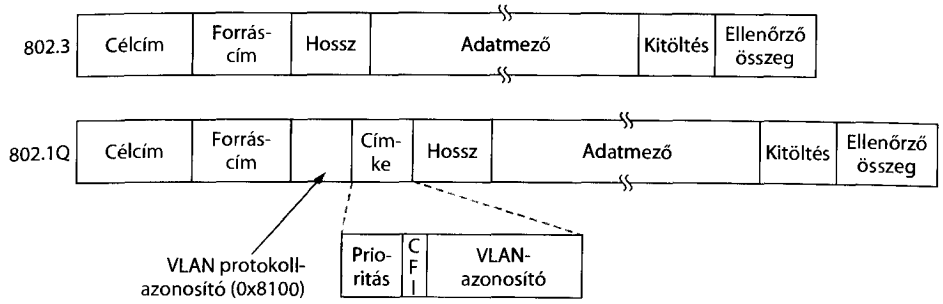
Ami a meglévő Ethernet-kártyák kidobására vonatkozó kérdést illeti: a válasz „nem". Emlékezzünk csak vissza arra, hogy a 802.3 bizottság még arra sem tudta rábírni az embereket, hogy a *Típus* mezőt *Hossz* mezőre változtassák. Ezek után elképzelhetjük, milyen reakció kísérte volna azt a bejelentést, hogy minden meglevő Ethernet-kártyát ki kell dobni. Mindenesetre az új Ethernet-kártyák már meg fognak felelni a 802.1Q előírásoknak, és helyesen töltik ki a VLAN-mezőket.

Mivel vannak számítógépek (és kapcsolók), amelyek nem ismerik a VLAN-t, a kerettel találkozó első VLAN-képes híd helyezi el a VLAN-mezőt, az útvonalon utolsóként szereplő pedig eltávolítja azt. A kevert topológiára hoz példát a 4.48. ábra. Ezen az ábrán a VLAN-képes számítógépek közvetlenül hozzák létre a felcímkézett (azaz 802.1Q) kereteket, és a továbbiakban a kapcsolók ezeket a címkéket használják. A sötéttel jelölt szimbólumok VLAN-képesek, míg az üresek nem.

A 802.1Q-val a kereteket az alapján színezik, hogy melyik porton érkeztek. Ahhoz, hogy ez a módszer működjön, az összes számítógépnek ugyanahhoz a VLAN-hoz kell tartoznia, ami csökkenti a rugalmasságot. Például a 4.47. ábrán ez a követelmény teljesül az összes portra, amelyhez csak egy számítógép csatlakozik, de nem teljesül a *B2* hídnak arra a portjára, amelyhez az elosztó csatlakozik.



4.48. ábra. *Hidakkal összekapcsolt LAN, amely csak részben áll VLAN-képes eszközökből. A sötéttel jelzett szimbólumok VLAN-képesek, az üresek nem*



4.49. ábra. *A (hagyományos) 802.3 és a 802.1Q keretformátumok*

Emellett a híd használhatja a felsőbb rétegbeli protokollokat a szín kiválasztására. Így az ugyanarról a portról érkező keretek különböző LAN-okra kerülhetnek attól függően, hogy IP-csomagokat vagy PPP-kereteket szállítanak.

Más módszerek is lehetségesek a szín meghatározására, de a 802.1Q nem támogatja ezeket. Egy példa erre: a MAC-cím alapján lehetne meghatározni a VLAN színét. Ez hasznos lehet, amikor a keretek egy közeli 802.11-es LAN-ról jönnek, amelyben a mozgásban lévő laptopok a mozgási helyüktől függően különböző portokon keresztül küldenek kereteket. Egy MAC-címet rögzített VLAN-ra képeznének le függetlenül attól, hogy melyik porton lépett be a LAN-ba. Ami pedig az 1518 bájtnál hosszabb keretek kérdését illeti, a 802.1Q egyszerűen felemelte a határt 1522 bájtra. Szerencsére, csak a VLAN-képes számítógépeknek és kapcsolóknak kell támogatniuk a hosszabb kereteket.

Vegyük most szemügyre a 4.49. ábrán látható 802.1Q formátumot! A keretben az egyetlen változást két darab 2 bájtos mező hozzáadása jelenti. Az első a *VLAN protokollazonosító (VLAN protocol ID)*, melynek értéke mindig 0x8100. Ez a szám nagyobb 1500-nál, ezért az Ethernet-kártyák nem hosszként, hanem típusként értelmezik. Az, hogy mit tesz egy hagyományos kártya egy ilyen kerettel, bizonytalan, mivel az ilyen keretek elküldését a hagyományos kártyák nem támogatják.

A második 2 bájtos mező három almezőt tartalmaz. Ezek közül a legfontosabb a *VLAN-azonosító (VLAN identifier)*, ami az alsó 12 bitet foglalja el. Erről szól az egész történet: ez adja meg a VLAN színét, amelyikhez a keret tartozik. A 3 bites *Prioritás (Priority)* mezőnek semmi köze a VLAN-okhoz, de mivel egy évtizedben úgyis csak egyszer módosítják az Ethernet-fejrészt, és ahhoz is három év és száz ember kell, akkor, ha már egyszer nekiálltak, miért ne raknának bele még valami jó dolgot. Ez a mező lehetővé teszi a szigorú és kevésbé szigorú követelményeket támasztó valós idejű, valamint a nem időérzékeny forgalmak megkülönböztetését, hogy jobb szolgáltatásminőséget lehessen elérni az Etherneten. Erre az Etherneten keresztül történő beszédátvitelnél van szükség (bár az igazat megvallva, az IP-nek is van egy hasonló mezője immár negyed százada, és azt sem használta soha senki).

Az utolsó mezőt igazából nem is *CFI*-nek *(Canonical Format Indicator – kanonikus formátumjelző)*, hanem *CEI*-nek *(Corporate Ego Indicator – vállalati érdekérvényesítésjelző)* kellene nevezni. A bitet eredetileg arra szánták, hogy megkülönböztessék vele a bitek sorrendjét a MAC-címekben (alsóvég vagy felsővég kódolású), de ez a használat

valahogy elsikkadt a viták során. Jelenléte ma már csak arra utal, hogy az adatmező egy nem módosítható 802.5-ös keretet tartalmaz, ami reményei szerint egy másik 802.5-ös LAN-t talál a céljánál, miközben a két hálózat között egy Etherneten halad át. Természetesen ennek az egész elrendezésnek semmi köze nincs a VLAN-okhoz. De hát a szabványosítási bizottságokban is csak úgy működik a politika, mint máshol: ha te megszavazod az én bitemet, én is megszavazom a tiédet.

Mint már említettük, ha egy címkézett keret érkezik egy VLAN-képes kapcsolóhoz, akkor a kapcsoló a VLAN-azonosító alapján keresi ki a táblázatból, hogy melyik porton kell a keretet kiküldeni. De honnan jön a táblázat? Ha kézzel kell összeállítani, mint annak idején a kézi konfigurációs hidakat, akkor ott vagyunk, ahol a part szakad. Az átlátszó hidak szépsége éppen abban van, hogy csatlakoztatás után rögtön működnek, és nem igényelnek semmilyen kézi beállítást. Nagy szégyen lenne elveszíteni ezt a tulajdonságot. Szerencsére a VLAN-képes hidak is képesek automatikusan konfigurálni magukat az elhaladó címkék megfigyelése alapján. Ha például egy 4-es VLAN-címkét hordozó keret a 3-as porton érkezik be, akkor a 3-as porton lévő egyik gép nyilvánvalóan a 4-es VLAN tagja. A 802.1Q szabvány, mely többnyire a 802.1D szabvány megfelelő részeire hivatkozik, kifejti, hogyan kell dinamikusan felépíteni a táblázatokat.

Mielőtt elhagynánk a VLAN-ok útválasztásának témáját, érdemes egy utolsó megfigyelést tennünk. Az internet és az Ethernet világában sokan fanatikus hívei az összeköttetés nélküli hálózatoknak, és hevesen elleneznek mindent, ami egy kicsit is emlékeztet az összeköttetésekre az adatkapcsolati vagy a hálózati rétegekben. A VLAN mégis valami olyasmit vezetett be, ami meglepően hasonlít az összeköttetésekhez. Egy helyesen működő VLAN-ban ugyanis minden keret egy új, speciális azonosítót hordoz, amit a kapcsoló táblázatában arra használnak, hogy kikeressék, hová kell küldeni a keretet. Pontosan ez az, ami az összeköttetés-alapú hálózatokban is történik. Az összeköttetés nélküli hálózatokban a célcím alapján végzik az útválasztást, nem pedig valamiféle összeköttetés-azonosító alapján. Az effajta, mélyben meglapuló összeköttetés-elvűségről az 5. fejezetben még többet is olvashatunk.

4.9. Összefoglalás

Egyes hálózatok csak egyetlen csatornával rendelkeznek, és minden kommunikáció ezt használja. Az ilyen hálózatokban a tervezés kulcskérdése az, hogy a csatornát hogyan osszuk ki a használatáért versengő állomások között. Az FDM és TDM kiosztási sémák egyszerűek és hatékonyak, amikor az állomások száma kicsi és rögzített, és a forgalom állandó. Mindkettőt széles körben használják ilyen körülmények között, például a telefontrönkök sávszélességének felosztásánál. Viszont amikor az állomások száma nagy és változó, vagy a forgalom eléggé löketes – a számítógép-hálózatokban ez az általános eset – az FDM és a TDM rossz választás.

Számos dinamikus csatornakiosztási algoritmust dolgoztak ki. Az időszeletes és az időszelet nélküli ALOHA-protokollt sok változatban használják valós hálózatokban, például kábelmodemeknél és RFID-nél. A csatorna állapotának érzékelésével az állomások elkerülhetik, hogy elkezdjenek adni, amikor már egy másik állomás ad. Ez a módszer a vivőjel-érzékelés, a CSMA számos változatához vezetett, amelyeket LAN-okban és MAN-okban használnak. Ez az alapja a klasszikus Ethernetnek és a 802.11-es hálózatoknak.

Ismeretes a protokolloknak egy olyan osztálya is, amely kiküszöböli, vagy legalábbis jelentősen csökkenti a versenyhelyzetet. A bittérképprotokoll, a gyűrűtopológiák és a bináris visszaszámlálás protokoll teljesen megszünteti a versengést. A fabejáró protokoll azzal csökkenti a versengés mértékét, hogy az állomásokat dinamikusan két különálló, különböző méretű csoportra osztja, és csak a csoporton belüli versengést engedi meg. Ideális esetben a csoport méretét úgy választják, hogy csak egyetlen küldésre kész állomás legyen, amikor a küldés lehetséges.

A vezeték nélküli LAN-ok járulékos problémái, hogy nehéz az ütköző átvitelek érzékelése, és hogy az állomások lefedettségi területe különbözik. A domináns vezeték nélküli LAN, az IEEE 802.11 esetén az állomások a CSMA/CA-t használják, hogy az első problémát kezeljék azzal, hogy rövid szüneteket hagynak az ütközések elkerülésére. Az állomások az RTS/CTS protokollal veszik fel a harcot a rejtett állomások ellen, amelyek a második probléma miatt keletkeznek. A laptopok és egyéb eszközök gyakran használják az IEEE 802.11-et a vezeték nélküli hozzáférési pontokhoz történő csatlakozáshoz, de használható az eszközök közötti kommunikációhoz is. A számos fizikai réteg közül bármelyik használható, beleértve a többcsatornás FDM-et, egy és több antennával és szórt spektrummal.

A 802.11-hez hasonlóan az RFID-olvasók és -címkék egy véletlen hozzáféréses protokollt használnak az azonosítójuk továbbítására. Más vezeték nélküli PAN-ok és MAN-ok máshogy működnek. A Bluetooth-rendszer fejhallgatókat és sok másféle perifériát csatlakoztat vezetékek nélkül számítógépekhez. Az IEEE 802.16 nagy kiterjedésű vezeték nélküli internet adatszolgáltatást nyújt mozgó és mozdulatlan számítógépeknek. Mindkét hálózat központosított, összeköttetés-alapú működésű, amelyekben a Bluetooth-mester és a WiMAX-bázisállomás döntik el, hogy mikor adhat vagy vehet egy állomás. A 802.16 számára ez a működés különböző szolgáltatásminőségeket támogat az olyan valós idejű forgalomhoz, mint amilyenek a telefonhívások és az olyan interaktív forgalomhoz, mint amilyen a web böngészése. A Bluetooth esetében a mesterbe helyezve a bonyolult működést elérték, hogy a szolgaeszközök olcsók legyenek.

A vezetékes LAN-ok domináns formája az Ethernet. A klasszikus Ethernet a csatornakiosztásra CSMA/CD-t használt egy locsolócső méretű sárga kábelen, amely géptől gépig kígyózott. A felépítése megváltozott és a sebessége 10 Mb/s-ról 10 Gb/s-ra növekedett, és folytatja a növekedést. Manapság kétpontos adatkapcsolatokat, mint amilyen a sodrott érpár, csatlakoztatnak elosztókhoz és kapcsolókhoz. Modern kapcsolóknál és duplex adatkapcsolatoknál nincs versenyhelyzet az adatkapcsolatokon, és a kapcsolók párhuzamosan tudnak különböző portokon keresztül kereteket továbbítani.

A LAN-okkal teli épületekben a LAN-ok összekapcsolására kellett valamilyen megoldás. Csatlakoztatás után rögtön működő kapcsolókat alkalmaznak erre a célra. A hidakat a visszafelé tanulás és feszítőfa algoritmussal látják el. Mivel ezeket a képességeket a modern kapcsolókba is beépítik, ezért a híd és a kapcsoló felcserélhető kifejezések. A kapcsolt hálózatok menedzsmentjét segíti elő a VLAN azzal, hogy lehetővé teszi a fizikai topológia különböző logikai topológiákra való bontását. A VLAN-szabvány, az IEEE 802.1Q, egy új keretformátumot vezet be az Ethernet-keretekre.

4.10. Feladatok

1. Ennél a feladatnál a 4. fejezet egyik képletét kell használnia. A megoldás előtt adja meg a használt képletet is! Az átvitelre szánt keretek véletlenszerű időközönként jelennek meg egy 100 Mb/s-os csatornán. Ha egy keret megérkezésekor a csatorna foglalt, akkor a keret egy várakozási sorba kerül. A keretek hosszának eloszlása exponenciális, 10 000 bites várható értékkel. Adja meg egy átlagos keret által elszenvedett késleltetést (beleértve a sorbaállás és az átvitel idejét is) a következő érkezési sebességek esetére!
   (a) 90 keret/s.
   (b) 900 keret/s.
   (c) 9000 keret/s.

2. Egy N állomásból álló csoport egyetlen, 56 kb/s-os, egyszerű ALOHA-csatornán osztozik. Az állomások 1000 bites kereteiket átlagosan 100 s-onként küldik el még akkor is, ha az előző kereteiket sem tudták elküldeni (például az állomások pufferelnek). Mekkora lehet N maximális értéke?

3. Vegyük a késleltetést egyszerű és időszeletelt ALOHA esetén, ha a terhelés kicsi! Melyik a kisebb? Adjon magyarázatot!

4. Egy nagyszámú ALOHA-felhasználókból álló populáció, az eredeti és az újraadásokat együtt számolva, 50 kérést bocsát ki másodpercenként. Az időszeletek 40 ms hosszúak.
   (a) Mi az első kísérlet sikerességének valószínűsége?
   (b) Mi a valószínűsége pontosan k ütközésnek, majd utána a sikernek?
   (c) Mi az adási kísérletek számának várható értéke?

5. Egy végtelen populációjú időszeletelt ALOHA-rendszerben egy állomásnak átlagosan 4 rést kell várnia egy ütközés és az azt követő újraküldés között. Rajzolja fel a rendszer késleltetésgörbéjét az áteresztőképesség függvényében!

6. Mekkora a hossza a versengési időszeletnek CSMA/CD esetén, ha (a) egy 2 km hosszú ikervezetékes kábelt használunk (a jelterjedési sebesség a vákuumbeli jelterjedési sebesség 82%-a), és ha (b) egy 40 km hosszú, többmódusú fényvezető szálat használunk (a jelterjedési sebesség a vákuumbeli jelterjedési sebesség 65%-a)?

7. Mennyit kell várakoznia az s állomásnak legrosszabb esetben, mielőtt leadhatja a keretét egy olyan LAN-on, mely az alapvető bittérképprotokollt használja?

8. Magyarázza meg, hogy egy bináris visszaszámlálást használó protokoll esetén a kis sorszámú állomásokat hogyan éheztetik ki, azaz hogyan nem hagyják, hogy adáshoz jussanak!

9. Az adaptív fabejáró protokoll alkalmazásával (1-től számozott) tizenhat állomás verseng egy csatorna használatáért. Ha az összes olyan állomás, amelynek prímszáma van, egyszerre kerül adásra kész állapotba, akkor mennyi bit-résre van szükség a versengés feloldásához?

10. Adott öt vezeték nélküli állomás, A, B, C, D és E. A állomás az összes többi állomással tud kommunikálni. B kommunikálni tud A-val, C-vel és E-vel. C kommunikálni tud A-val, B-vel és D-vel. D kommunikálni tud A-val, C-vel és E-vel. E kommunikálni tud A-val, D-vel és B-vel.
    (a) Ha A ad B-nek, milyen más kommunikáció folyhat még?
    (b) Ha B ad A-nak, milyen más kommunikáció folyhat még?
    (c) Ha B ad C-nek, milyen más kommunikáció folyhat még?

11. Hat állomás kommunikál A-tól F-ig a MACA-protokoll segítségével. Lehetséges-e az, hogy két átvitel történik egyszerre? Indokolja válaszát!

12. Egy hétemeletes irodaház minden szintjén 15 szomszédos iroda van. Minden iroda elülső oldalán van egy terminálaljzat, így azok a függőleges síkon rácsozatot alkotnak. Az aljzatok mind függőlegesen, mind vízszintesen 4 m távolságban vannak egymástól. Ha feltesszük, hogy bármelyik két aljzat között kihúzható egyenes vonalban kábel, akkor hány méter kábelre van szükség az összes aljzat bekötéséhez, ha a használt hálózat:
    (a) Egy csillag összeállítás, egyetlen útválasztóval a közepén?
    (b) Egy klasszikus 802.3 LAN?

13. Mekkora a jelsebessége (baud rate) egy klasszikus 10 Mb/s-os 802.3 LAN-nak?

14. Vázolja fel a 0001110101 bitsorozat Manchester-kódját egy klasszikus Etherneten!

15. Egy 1 km hosszú, 10 Mb/s-os CSMA/CD LAN-on (nem 802.3) a terjedési sebesség 200 m/μs. Ebben a rendszerben ismétlők nem megengedettek. Az adatkeretek 256 bit hosszúak, amely magába foglalja a fejrész, az ellenőrző összege és az egyéb, nem adat jellegű információk 32 bitjét is. Egy sikeres átvitelt követően az első rés a vevőnek van fenntartva azért, hogy egy 32 bites nyugtakeretet küldhessen a feladónak. Feltételezve, hogy nincsenek ütközések, mekkora a hasznos (fejléc nélküli) adatátviteli sebesség?

16. Két CSMA/CD-állomás egy keret elküldésével próbálkozik. Mindketten versenyeznek a csatorna használatáért a bináris exponenciális visszalépéses algoritmus használatával egy ütközés után. Mi a valószínűsége annak, hogy a versenyhelyzet k kör után feloldódik, és mi a versengési periódusonként szükséges körök várható száma?

17. Egy olyan IP-csomagot szeretnénk Etherneten átvinni, melynek hossza az összes fejrésszel együtt 60 bájt. Szükség van-e kitöltésre az Ethernet-keretben, ha nem használunk LLC-t, és ha igen, hány bájtnyi kitöltés szükséges?

18. Ethernet-hálózatok esetén a kereteknek legalább 64 bájt hosszúnak kell lennie, hogy az adó-vevő még biztosan adjon akkor is, ha a vezeték távoli részein következik be ütközés. A gyors Ethernet minimális kerethossza szintén 64 bájt, pedig a biteket tízszer gyorsabban adja. Hogyan lehetséges, hogy ugyanazt a minimális kerethosszt használják?

19. Egyes könyvek nem 1500, hanem 1522 bájtnak említik az Ethernet-keretek maximális méretét. Tévednek vajon? Indokolja válaszát!

20. Hány keretet képes kezelni a gigabites Ethernet másodpercenként? Jól gondolja meg válaszát, és vegye figyelembe az összes fontosabb esetet! *Tipp*: lényeges, hogy *gigabites* Ethernet!

21. Mondjon két olyan hálózatot, melyekben szorosan egymást követhetik a keretek! Miért érdemes ezt a képességet biztosítani?

22. A 4.27. ábrán négy állomás: *A*, *B*, *C* és *D* látható. Az utolsó két állomás közül melyik van közelebb *A*-hoz, és miért?

23. Adjon meg egy olyan példát, amely megmutatja, hogy a 802.11 protokollban használt RTS/CTS kicsit másképp működik, mint a MACA-protokollban!

24. Adott egy vezeték nélküli LAN egy hozzáférési ponttal és 10 kliensállomással. Négy állomás 6 Mb/s sebességű, négy állomás 18 Mb/s sebességű és az utolsó két állomás 54 Mb/s sebességű. Mi az egyes állomások által tapasztalt adatsebesség, ha mind a 10 állomás egyszerre ad és
(a) TXOP-t nem használnak?
(b) TXOP-t használnak?

25. Tegyük fel, hogy egy 11 Mb/s-os 802.11b LAN szorosan egymás után pakolt 64 bájtos kereteket visz át egy olyan rádiós csatornán, melynek bithibaaránya 10^{-7}. Átlagosan hány keret fog megsérülni másodpercenként?

26. Egy 802.16-os hálózat 20 MHz-es sávszélességgel rendelkezik. Hány bitet lehet elküldeni másodpercenként egy előfizetői állomásnak?

27. Mondjon két okot, ami miatt a hálózatok hibajavító kódolást használhatnak a hibajelzés és újraadás helyett!

28. Soroljon fel két hasonlóságot és két különbséget a WiMAX és a 802.11 között!

29. A 4.34. ábrán azt láthatjuk, hogy egy Bluetooth-eszköz két pikohálózatban is lehet egyszerre. Indokolja-e valami azt, hogy egy eszköz ne lehessen mester egy időben mindkét hálózatban?

30. Mi a legnagyobb megengedett mérete az adatmezőnek egy 3 időszeletes alapsebességű Bluetooth-keret esetén? Válaszát indokolja!

31. A 4.24. ábra több fizikai rétegbeli protokollt ábrázol. Ezek közül melyik áll legközelebb a Bluetooth fizikai rétegének protokolljához? Mi a legnagyobb különbség a két protokoll között?

32. A 4.6.6. szakaszban említettük, hogy egy 1 időszeletes, alapsebességű, ismétléses kódolású keret nagyjából 13%-os hatékonyságú. Mekkora lesz a hatékonysága egy 5 időszeletes, alapsebességű, ismétléses kódolású keretnek?

33. A 802.11 frekvenciaugrásos szórt spektrumú változatában a jelzőfénykeretek (beacon) tartalmazzák a tartózkodási időt. Mit gondol, vajon a Bluetooth hasonló jelzőfénykeretei szintén tartalmazzák a tartózkodási időt? Fejtse ki válaszát!

34. Tegyük fel, hogy 10 RFID-címke van egy RFID-olvasó körül. Mi a Q legjobb értéke? Mi annak a valószínűsége, hogy egy címke ütközés nélkül tud egy adott időszeletben válaszolni?

35. Soroljon fel az RFID-rendszerek biztonsági aggályai közül néhányat!

36. Egy gyors Ethernethez tervezett kapcsolónak olyan hátlapja van, mely adatok 10 Gb/s sebességű mozgatására képes. Hány keretet képes kezelni ez az eszköz másodpercenként a legrosszabb esetben?

37. Írja le röviden a tárol-és-továbbít és az átvágó elvű kapcsolók közti különbséget!

38. Adott egy kiterjedt LAN, amelyet a *B1* és *B2* hidak kapcsolnak össze a 4.41.(b) ábrán látható módon. Tételezzük fel, hogy mindkét híd hash-táblája üres. Sorolja fel az összes portot, amelyen egy csomagot továbbítanak, az alábbi adatátvitelek sorozata esetén:
(a) *A* küld egy csomagot *C*-nek.
(b) *E* küld egy csomagot *F*-nek.
(c) *F* küld egy csomagot *E*-nek.
(d) *G* küld egy csomagot *E*-nek.
(e) *D* küld egy csomagot *A*-nak.
(f) *B* küld egy csomagot *F*-nek.

39. A tárol-és-továbbít elvű kapcsolóknak van egy előnyük az átvágó elvű kapcsolókkal szemben a sérült keretek vonatkozásában. Fejtse ki ezt az előnyt!

40. A 4.8.3. szakaszban említettük, hogy néhány hídnak nem szükséges a feszítőfa részének lennie. Vázolja azt a forgatókönyvet, amikor egy hídnak nem kell a feszítőfa részének lennie!

41. A VLAN-ok működéséhez a hidakban konfigurációs táblázatokra van szükség. Mi a helyzet, ha a 4.47. ábra VLAN-jai elosztókat használnak kapcsolók helyett? Akkor az elosztókban is konfigurációs táblázatokra lesz szükség? Miért igen vagy miért nem?

42. A 4.48. ábrán a jobb oldali hagyományos távoli körzetben lévő kapcsoló VLAN-képes. Hagyományos kapcsolót is lehetne itt használni? Ha igen, hogyan működne ez a megoldás? Ha nem, miért nem?

43. Írjon programot, mely a CSMA/CD-protokoll Ethernet feletti működését szimulálja abban az esetben, ha *N* állomás áll adásra készen, miközben egy keret átvitele folyamatban van! A programja adjon jelentést azokról az időpontokról, amikor az egyes állomások sikeresen megkezdik a keretük elküldését! Felteheti, hogy minden időszeletben (51,2 μs-onként) történik egy óraütés, és az ütközések érzékelése valamint az ütközési sorozat elküldése egy időszeletnyi időbe telik. Az összes keret hossza a megengedett maximális kerethossz.

5. A hálózati réteg

A hálózati réteg feladata, hogy a csomagokat a forrástól egészen a célig eljuttassa. Ehhez a csomagnak esetleg több útválasztón is keresztül kell haladnia. Ez a feladat láthatóan elkülönül az adatkapcsolati réteg feladatától, amely ennél szerényebb: azaz keretek továbbítása a vonal egyik végétől a másikig. Ezért a hálózati réteg a legalacsonyabb réteg, amely két végpont közti átvitellel foglalkozik.

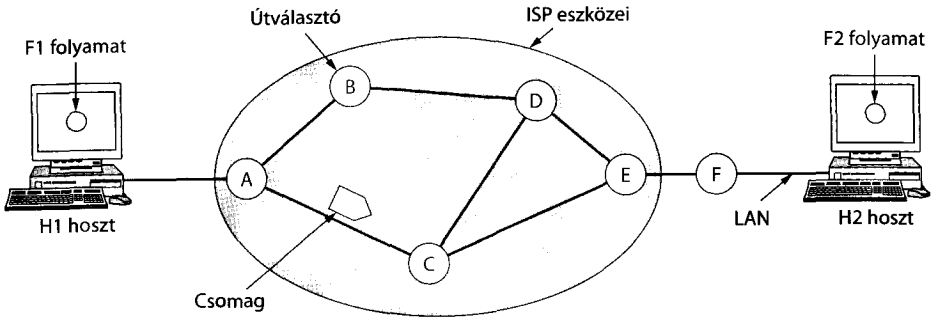
E célok elérése érdekében a hálózati rétegnek ismernie kell a hálózat (vagyis az útválasztók és az adatkapcsolatok halmazának) topológiáját, és megfelelő útvonalakat kell találnia azon keresztül, még nagy kiterjedésű hálózatok esetén is. Arra is ügyelnie kell, hogy úgy válassza ki az útválasztókat, hogy elkerülje néhány kommunikációs vonal és útválasztó túlterhelését, míg mások tétlenül maradnak. Végül a hálózati rétegre hárul azoknak a problémáknak a megoldása is, amelyek akkor merülnek fel, amikor a forrás és a cél különböző hálózatokhoz tatoznak. Ebben a fejezetben megtárgyaljuk és illusztráljuk mindezeket a kérdéseket, elsősorban az internet és annak hálózati réteg protokollja, az IP példáján.

5.1. A hálózati réteg tervezési kérdései

A következő szakaszokban néhány olyan kérdést tekintünk át, amelyekkel a hálózati réteg tervezőjének meg kell birkóznia. Ezek között találjuk a szállítási rétegnek nyújtott szolgáltatást és a hálózat belső tervezését.

5.1.1. Tárol-és-továbbít típusú csomagkapcsolás

Mielőtt azonban rátérnénk a hálózati réteg részleteinek megtárgyalására, érdemes áttekintenünk azt a környezetet, amelyben a réteg protokolljai működnek. Ezt szemlélteti az 5.1. ábra. A rendszer legfőbb elemei a sötét ellipszisben található internet-szolgáltatói (ISP) berendezések (átviteli vonalakkal összekötött útválasztók), és az ellipszisen kívül ábrázolt felhasználói berendezések. A *H1* hoszt egy bérelt vonalon keresztül közvetlen összeköttetésben áll az internetszolgáltató *A* útválasztójával. Ez a hoszt lehet egy otthoni számítógép, amely egy DSL-modemhez kapcsolódik. Ezzel szemben a *H2* hoszt egy



5.1. ábra. *A hálózati réteg protokolljainak környezete*

olyan LAN-hoz csatlakozik, amelyik esetleg lehet egy olyan irodai Ethernet, amelynek *F* útválasztóját a felhasználó birtokolja és üzemelteti. Ez az útválasztó is bérelt vonalon keresztül kapcsolódik a szolgáltatói berendezésekhez. Az *F* útválasztót az ellipszisen kívül ábrázoltuk, mivel nem tartozik a szolgáltatóhoz. Fejezetünk szempontjából a felhasználói területen lévő útválasztókat is a szolgáltatói hálózat részének tekintjük, hiszen ugyanazokat az algoritmusokat használják, mint a szolgáltató útválasztói (és minket most leginkább az algoritmusok érdekelnek).

A berendezések működése a következő. A hosztok az elküldeni kívánt csomagokat a saját LAN-on vagy a szolgáltató felé vezető kétpontos (point-to-point) kapcsolaton keresztül a legközelebbi útválasztóhoz továbbítják. Az útválasztó tárolja a csomagot, amíg az teljes egészében be nem érkezik, hogy ki lehessen számítani az ellenőrző összeget. Ezután a csomag mindig a soron következő útválasztóhoz kerül, míg el nem éri a címzett hosztot. Ezt hívják tárol-és-továbbít (store-and-forward) típusú csomagkapcsolásnak, amint azt az előző fejezetekben már láthattuk.

5.1.2. A szállítási rétegnek nyújtott szolgáltatások

A hálózati réteg a hálózati réteg és a szállítási réteg közötti interfészen nyújtja szolgáltatásait a szállítási rétegnek. Fontos ismernünk, hogy milyen jellegűek ezek a szolgáltatások. A hálózati réteg tervezésénél a következő vezérelveket tartották szem előtt:

1. A szolgáltatásoknak függetleneknek kell lenniük az útválasztók kialakításától.
2. A szállítási réteg elől el kell takarni a jelenlevő útválasztók számát, típusát és topológiáját.
3. A szállítási réteg rendelkezésére bocsátott hálózati címeknek egységes számozási rendszert kell alkotniuk, még LAN-ok és WAN-ok esetén is.

Ezeknek a céloknak a figyelembevételével, a hálózati réteg tervezői nagy szabadságot élveznek a szállítási rétegnek nyújtandó szolgáltatások részletes specifikációinak elkészítése során. Ám ez a szabadság gyakran válik két szembenálló csoport indulatos csatározásává. A vita középpontjában az áll, hogy vajon a hálózati réteg összeköttetés-alapú vagy összeköttetés nélküli szolgáltatást nyújtson-e.

Az egyik tábor (amelyet az internet közössége képvisel) véleménye az, hogy az útválasztók dolga csupán a bitek ide-oda mozgatása. Nézetük szerint (amelyet valódi, működő számítógép-hálózattal való 40 éves tapasztalat támaszt alá), a hálózat eredendően megbízhatatlan, függetlenül annak tervezésétől. Ezért a hosztoknak a megbízhatatlanságot tényként kell elfogadniuk, és a hibavédelmet (vagyis a hibajelzést és hibajavítást) és a forgalomszabályozást maguknak kell elvégezniük.

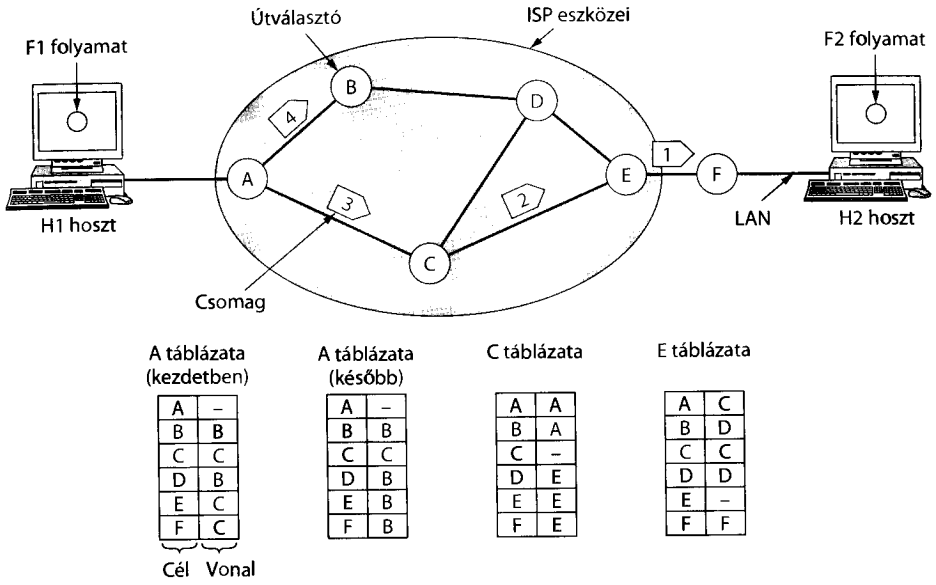
Ez a nézet gyorsan ahhoz a következtetéshez vezet, hogy a hálózati szolgáltatásnak összeköttetés nélkülinek kell lennie, a SEND PACKET és RECEIVE PACKET primitíveken kívül alig kell valami más. Hangsúlyozottan nincs szükség a csomagok sorrendi kezelésére és forgalomszabályozásra, mert ezt a hosztok amúgy is mindenképpen megteszik, és valószínűleg kevés nyereség származik abból, ha ezeket kétszer hajtjuk végre. Erre példa a **végponttól végpontig tervezési alapelv (end-to-end argument)**, amely nagyban befolyásolja az internet formáját [Saltzer és mások, 1984]. Továbbá, minden csomagnak hordoznia kell a teljes célcímet, mivel mindegyik elküldött csomag az előző csomagoktól függetlenül kerül továbbításra (ha egyáltalán voltak ilyenek).

A másik tábor (amelyet a telefontársaságok képviselnek) véleménye az, hogy a hálózatnak megbízható, összeköttetés-alapú szolgáltatást kell nyújtania. Állításuk szerint a világméretű telefonhálózattal szerzett 100 éves sikeres gyakorlat jó alapot ad ehhez. Ebből a nézőpontból a szolgáltatásminőség a meghatározó tényező, márpedig ezt a hálózatban kiépített összeköttetések nélkül nagyon nehéz elérni, különösen az olyan valós idejű forgalmak esetén, mint amilyen a hang vagy a mozgóképek továbbítása.

Az eltelt több évtized ellenére ez a probléma nagyon is aktuális. Az eleinte széles körben használt adathálózatok, mint például a 70-es években alkalmazott X.25, illetve a 80-as években alkalmazott kerettovábbító (frame relay), összeköttetés-alapúak voltak. Az ARPANET és az internet korai alkalmazása óta azonban az összeköttetés nélküli hálózati rétegek népszerűsége jelentősen megnőtt. Az IP-protokoll a siker töretlenül jelen lévő szimbóluma. Az IP népszerűségét az összeköttetés-alapú ATM-technika sem tudta megtörni, amelyet arra fejlesztettek ki, hogy az IP-protokoll helyére lépjen. Ehelyett az ATM az, amelyet jelenleg csak nagyon szűk körben használnak, az IP pedig a telefonos hálózatokat is átveszi. Ugyanakkor érdemes megjegyezni, hogy az internet is kezd lépést tartani az egyre fontosabbá váló szolgáltatásminőségi garanciákkal, pontosabban, kezd magára venni olyan tulajdonságokat is, amelyek eddig csak az összeköttetés-alapú szolgáltatásokat jellemezték, amint azt hamarosan látni fogjuk. Összeköttetés-alapú technika például az MPLS (MultiProtocol Label Switching), amelynek leírását az 5.1.4. szakasz tartalmazza, illetve a VLAN, amelyet a 4. fejezet mutat be. Mindkét technikát széles körben használják.

5.1.3. Összeköttetés nélküli szolgáltatás megvalósítása

Miután láttuk, hogy a hálózati réteg kétfajta szolgáltatást tud nyújtani a felhasználónak, ideje megnéznünk, hogyan működik belül. A felkínált szolgáltatás típusától függően kétfajta szerveződés lehetséges. Az összeköttetés nélküli szolgáltatás esetében a hálózat-



5.2. ábra. *Útválasztás datagramalapú hálózatban*

ba érkező csomagok egyenként és egymástól függetlenül kerülnek továbbításra; előzetes összeköttetés-felépítésre nincs szükség. Ebben az összefüggésben a csomagokat gyakran **datagramoknak (datagrams, DG)**, a hálózatot pedig **datagramalapú hálózatnak (datagram network)** is nevezik, a távirat (telegram) kifejezés mintájára. Ha összeköttetés-alapú szolgáltatást használunk, a forrás és a cél útválasztó között előre ki kell építeni egy útvonalat, mielőtt egyetlen adatcsomagot is elküldenénk. Ezt a kapcsolatot **virtuális áramkörnek (virtual circuit, VC)** nevezzük a telefonhálózat fizikai áramköreinek mintájára, a hálózat neve pedig ebben az esetben **virtuálisáramkör-alapú hálózat (virtual circuit subnet)**. Ebben a szakaszban a datagramalapú, a következőben pedig a virtuálisáramkör-alapú hálózatokat vizsgáljuk.

Nézzük meg tehát, hogy működik egy datagramalapú hálózat. Tegyük fel, hogy az 5.2. ábra *F1* folyamata egy hosszú üzenetet szeretne küldeni az *F2* folyamat számára. *F1* átadja az üzenetet a szállítási rétegének azzal az utasítással, hogy továbbítsa azt a *H2* hoszt *F2* folyamatának. A szállítási réteg kódja a *H1* hoszton fut, tipikusan az operációs rendszeren belül. Ez az üzenet elejéhez fűz egy szállítási fejrészt és továbbítja azt a hálózati rétegnek, amelyik valószínűleg szintén egy eljárás az operációs rendszeren belül.

Tegyük fel, hogy az üzenet négyszer olyan hosszú, mint a maximális csomagméret, tehát a hálózati rétegnek[1] négy csomagra kell bontania, és mindegyiket sorban az útválasztóhoz továbbítania valamilyen kétpontos protokoll, például a PPP felhasználásával. Ezen a ponton lép be a képbe a szolgáltató. A hálózat minden útválasztójának van egy

1 Az üzenet csomagokra tördelése és a csomagok összerakása üzenetté valójában nem a hálózati, hanem a szállítási rétegben történik. *(A lektor megjegyzése)*

belső táblázata, amely minden lehetséges cél esetére megadja, hogy merrefelé kell továbbítani a csomagokat. A táblázat bejegyzései olyan kettősök, amelyek a címzett útválasztó-azonosítóját és a címzetthez vezető kimeneti vonal azonosítóját tartalmazzák. Csak közvetlen kapcsolatban álló összeköttetéseket lehet használni. Az 5.2. ábrán például az útválasztónak csak két kimeneti vonala van – egy a *B* és egy a *C* felé –, így minden bejövő csomagot a két útválasztó közül valamelyiknek kell továbbküldeni, még akkor is, ha a címzett egy ezektől különböző útválasztó. Az *A* kezdeti útválasztó táblázatát a képen a „kezdetben" megjelölésű oszlop mutatja. Négy csomag útját követhetjük végig az ábrán.

Az *A* útválasztó a beérkezett 1., 2. és 3. csomagot rövid ideig tárolja, miután azok beérkeztek a bejövő adatkapcsolaton és kiszámítja az ellenőrző összegüket. Ezután a táblázatának megfelelően továbbítja a *C*-hez egy új keretben. Az 1. csomag ezután az *E*-hez, majd az *F*-hez kerül. *F*-hez érve beágyazódik egy keretbe, és a LAN-on keresztül eljut *H2*-höz. A 2. és a 3. csomag ugyanezt az utat járja be.

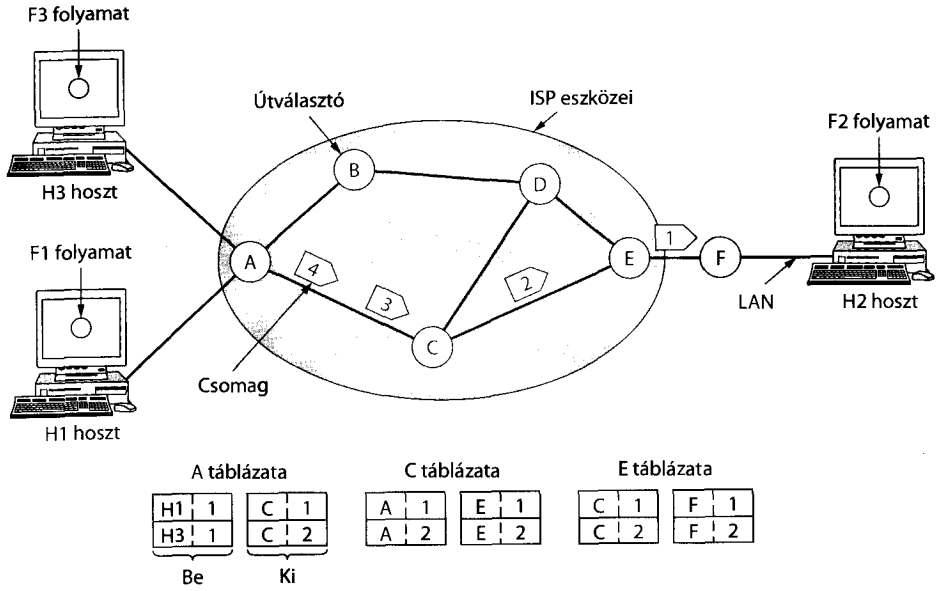
A 4. csomaggal azonban valami más történik. Az *A*-tól a *B* útválasztóhoz kerül, annak ellenére, hogy az *F* volt a címzett. Valami miatt az *A* úgy döntött, hogy a 4. csomagot más úton továbbítja, mint az első hármat. Értesülhetett például egy, az *ACE* út mentén lévő torlódásról, és módosíthatta az útválasztó táblázatát, amint azt az ábra „később" megjelölésű oszlopa mutatja. Azt az algoritmust, mely a táblázatok karbantartását végzi és meghozza az útválasztó döntéseket, **útválasztó algoritmusnak (routing algorithm)** nevezzük. Ezen algoritmusok képezik fejezetünk egyik legfontosabb tárgyát. Ahogy látni fogjuk, számos különböző algoritmus létezik.

Az internetprotokoll (IP, Internet Protocol), amely a teljes internet alapját képezi, meghatározó példája az összeköttetés nélküli hálózati szolgáltatásnak. Minden csomag tartalmazza a címzett IP-címet, amelyet az útválasztók használnak az egyes csomagok egyesével történő továbbításához. Az IPv4-csomagok címe 32 bites, az IPv6-csomagok címe pedig 128 bites. Az IP-t részletesen tárgyaljuk a fejezet későbbi részében.

5.1.4. Összeköttetés-alapú szolgáltatás megvalósítása

Az összeköttetés-alapú szolgáltatáshoz szükségünk van egy virtuálisáramkör-alapú hálózatra. Nézzük, hogyan is működik ez! A virtuális áramkörök alapötlete, hogy elkerülik azt, hogy minden egyes csomag számára újra és újra útvonalat kelljen választani, szemben az 5.2. ábrán látottakkal. Ehelyett, már az összeköttetés felépítésekor kiválasztanak a küldő és a címzett hoszt között egy utat, amelyet az útválasztók az összeköttetés kiépítése keretében tárolnak a táblázataikban. Ezt az utat használják azután a kapcsolat teljes forgalmának lebonyolítására, pontosan úgy, ahogy a telefonhálózat esetében is.[2] Amikor az összeköttetés megszűnik, a virtuális áramkör is bomlik. Az összeköttetés-alapú szolgáltatás esetén minden csomag tartalmaz egy azonosítót, amely megmondja, hogy a csomag melyik virtuális áramkörhöz tartozik.

2 A telefonhálózatra való hivatkozás nem szerencsés, mivel ott fizikai áramkör, míg az összeköttetés-alapú csomagkapcsolásnál logikai csatorna, virtuális áramkör alakul ki a két távoli felhasználó között és szolgál a teljes forgalom lebonyolítására. *(A lektor megjegyzése)*



5.3. ábra. *Útválasztás virtuálisáramkör-alapú hálózatban*

Példaként tekintsük az 5.3. ábrán látható helyzetet. Itt a *H1* hoszt kialakított egy összeköttetést *H2*-vel. Ezt jelzi az útválasztó táblázatok első bejegyzése. Az *A* útválasztó táblázatának első sora szerint, ha egy 1-es azonosítót hordozó csomag érkezik a *H1* felől, akkor azt a *C* útválasztó felé kell továbbítani 1-es azonosítóval. Hasonlóképpen, *C* első bejegyzése az *E*-hez továbbítja a csomagot, szintén 1-es összeköttetés-azonosítóval.

Most nézzük meg, mi történik, ha *H3* is szeretne összeköttetést létesíteni *H2*-vel. Összeköttetés-azonosítónak az 1-et választja (mivel ez kezdeményezi az összeköttetést és ez az egyetlen összeköttetése), és arra utasítja a hálózatot, hogy hozza létre a virtuális áramkört. Ez eredményezi a második sort a táblázatokban. Vegyük észre, hogy ez ütközéshez vezet, mert bár *A* könnyen meg tudja különböztetni a *H1*-ből és a *H3*-ból érkező, egyaránt 1-es azonosítójú csomagokat, *C* már nem képes erre. Ezért *A* új összeköttetés-azonosítót rendel a második összeköttetés kimenő forgalmához. Az ilyen konfliktushelyzetek elkerüléséhez tehát az szükséges, hogy az útválasztók képesek legyenek megváltoztatni az összeköttetés-azonosítókat a kimenő csomagokban.

Egyes esetekben ezt a működést **címkekapcsolásnak (label switching)** is nevezik. Összeköttetés-alapú hálózati szolgáltatásra példa a **többprotokollos címkekapcsolás (MultiProtocol Label Switching, MPLS)**. Ezt használják az ISP-hálózatok az interneten. Ebben az esetben az IP-csomagba beágyaznak egy 20 bites összeköttetés-azonosítót vagy összeköttetés-címkét tartalmazó MPLS-fejrészt. Az MPLS gyakran rejtve van az ügyfelek elől, mint amikor az ISP hosszú távú összeköttetéseket alakít ki nagy forgalomhoz, de egyre inkább használják akkor is, ha a szolgáltatás minősége fontos, illetve egyéb ISP forgalomkezelési feladatokhoz is alkalmazzák. Az MPLS-ről további részletek a fejezet későbbi részében találhatók.

5.1.5. A virtuálisáramkör- és a datagramalapú hálózatok összehasonlítása

Mind a virtuális áramköröknek, mind a datagramoknak megvannak a maguk támogatói és ellenzői. Mi most megkíséreljük összefoglalni mindkét oldal érveit. A főbb témák megtalálhatók az 5.4. ábrán, bár a szőrszálhasogatók bizonyára az ábra minden részére tudnának ellenpéldát találni.

A hálózaton belül sokfajta kompromisszum lehetséges a virtuális áramkörök és a datagramok között. Az egyik kompromisszum az összeköttetés-felépítési és a címfeldolgozási idő közt kereshető. A virtuális áramkörök használata megkövetel egy összeköttetés-felépítési fázist, amely idő- és erőforrás-igényes. Ha azonban az összeköttetés felépült, akkor könnyű eldönteni, mi legyen a csomagokkal: az útválasztó az áramkör számát indexként használja a táblázatokhoz a csomag továbbítása érdekében. Datagramalapú hálózatban nincs szükség az összeköttetés felépítésére, azonban sokkal bonyolultabb az a keresőeljárás, amely a célcímhez tartozó bejegyzés meghatározásához szükséges.

Másik probléma, hogy a datagramalapú hálózatokban használt célcím hosszabb a virtuális áramkörökben használt áramkörszámnál, mivel a célcím globális jelentéssel rendelkezik. Ha a csomagok hossza rövid, akkor a minden csomagban jelenlevő teljes célcím jelentős mértékű többletterhet jelent, és ez sávszélesség-veszteséggel jár.

További kérdés, hogy mennyi helyet foglalnak el a táblázatok az útválasztók memóriájában. Egy datagramalapú hálózatban minden lehetséges címzett számára fenn kell tartani egy bejegyzést, míg a virtuálisáramkör-alapú hálózatban csak az egyes áramkörök

Kérdés	Datagramalapú hálózat	Virtuálisáramkör-alapú hálózat
Áramkör-felépítés	Nem szükséges	Megkövetelt
Címzés	Minden csomag tartalmazza a teljes forrás- és célcímet	Minden csomag egy rövid virtuálisáramkör-számot tartalmaz
Állapotinformáció	Az útválasztó nem tartalmaz állapotinformációt	Minden virtuális áramkör összeköttetésenként helyet igényel az útválasztó táblázatában
Útválasztás	Minden csomagot egyedileg irányítanak	Az útvonalat akkor választják ki, amikor a virtuális áramkör felépül; minden csomag ezt az útvonalat követi
Az útválasztó meghibásodásainak hatása	Semmi, eltekintve az összeomlás során elveszett csomagoktól	Minden virtuális áramkör megszakad, amely a meghibásodott útválasztón keresztülhaladt
Szolgáltatásminőség	Bonyolult	Könnyű, ha elég erőforrást lehet előre lefoglalni mindegyik virtuális áramkör számára
Torlódáskezelés	Bonyolult	Könnyű, ha elég erőforrást lehet előre lefoglalni mindegyik virtuális áramkör számára

5.4. ábra. *A datagramalapú és a virtuálisáramkör-alapú hálózatok összehasonlítása*

számára kell egy-egy bejegyzés. Megjegyzendő ugyanakkor, hogy a virtuális áramkörök ezen előnye sem ennyire egyértelmű, mert az összeköttetést felépítő csomagokat is irányítani kell, ezek pedig ugyanúgy tartalmazzák a célcsomagok címét, mint a datagramok.

A virtuális áramkörök a szolgáltatásminőségi garanciák és a hálózaton belüli torlódáskezelés területén rendelkeznek némi előnnyel, mert az erőforrásokat (puffereket, sávszélességet, processzoridőt) előre, az összeköttetés felépítésekor le tudják foglalni. Mire a csomagok megérkeznek, a szükséges sávszélesség és útválasztó-kapacitás már rendelkezésre fog állni. Datagramalapú hálózatokban a torlódások elkerülése bonyolultabb kérdés.

A tranzakciófeldolgozó rendszereknél (mint például amikor az üzletek hitelkártyás vásárlásokat ellenőriznek) egy virtuális áramkör felállításához és kitörléséhez szükséges többletteher könnyen felülmúlhatja az áramkör valódi hasznát. Ha a forgalom nagy része várhatóan ilyen típusú lesz, a hálózaton belüli kapcsolt virtuális áramkörök használatának nincs sok értelme. Másrészt viszont az olyan hosszú ideig működő virtuális áramkörök, mint amilyen a VPN két cég központja között, amelyeket kézzel hoznak létre és hónapokig vagy évekig fennállnak, hasznosak lehetnek.

A virtuális áramkörök azonban sebezhetők is. Ha egy útválasztó összeomlik, és elveszti memóriájának tartalmát, még ha egy másodperccel később újraindul is, az összes rajta keresztülhaladó virtuális áramkört meg kell szakítani. Ezzel szemben, ha egy datagramalapú útválasztó megy tönkre, csak azok a felhasználók szenvednek kárt, amelyeknek a csomagjai éppen ebben az útválasztóban álltak sorban, sőt talán nem is mindegyik, mivel a küldő valószínűleg rövid időn belül újraküldi azokat. Egy kommunikációs vonal elvesztése végzetes a rajta keresztülhaladó virtuális áramkörök számára, de könnyen ellensúlyozható datagramok használata esetén. A datagramok azt is lehetővé teszik az útválasztóknak, hogy kiegyenlítsék a hálózat terhelését, mivel az útvonalak egy hosszabb csomagsorozat közben is módosíthatók.

5.2. Útválasztó algoritmusok

A hálózati réteg fő feladata, hogy a csomagokat a forrásgéptől a célgépig irányítsa. A legtöbb hálózatban a csomagoknak ehhez több útválasztón kell keresztülhaladni, több ugrást kell megtenni. Az egyetlen figyelemre méltó kivétel az adatszóró hálózatok esete, de az útválasztás itt is téma lehet, ha a forrás és a cél nem ugyanazon hálózaton van. Azok az algoritmusok, amelyek az útvonal meghatározását szolgálják, és azok az adatstruktúrák, amelyeket az algoritmusok használnak, a hálózati réteg tervezésének egyik legfontosabb területét alkotják.

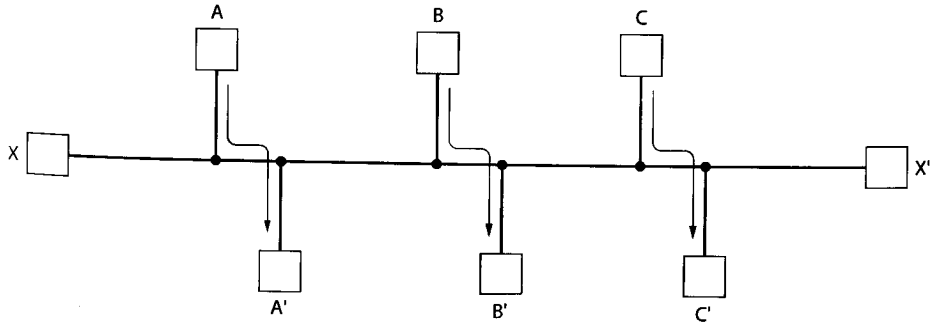
Az **útválasztó algoritmus** (**routing algorithm**) a hálózati réteg szoftverének azon része, amely azért a döntésért felelős, hogy egy bejövő csomag melyik kimeneti vonalon kerüljön továbbításra. Ha a hálózat belül datagramokat használ, ezt a döntést újra meg újra meg kell hozni minden beérkező adatcsomagra, mivel lehet, hogy a legjobb útvonal a legutóbbi meghatározás óta változott. Ha a hálózat belül virtuális áramköröket használ, akkor útválasztó döntéseket csak új virtuális áramkör felépítésekor kell meghozni. Ezután az adatcsomagok már csak az előzőleg kialakított útvonalat követik. Ezt az utóbbi esetet néha **viszony-útválasztásnak** (**session routing**) is nevezik, mivel az útvonal érvényben marad a teljes felhasználói viszony (mint például a VPN-en keresztüli bejelentkezés) alatt.

Néha célszerű megkülönböztetnünk az útválasztást, azaz a választandó útvonalról való döntést és a továbbítást, ami a csomag beérkezésekor történik. Képzeljük azt, hogy az útválasztó két folyamatot tartalmaz. Az egyik kezeli a beérkező csomagokat: mindegyik számára kikeres egy kimeneti vonalat az útválasztó táblázatokból. Ez a folyamat a **továbbítás** (**forwarding**). A másik folyamat az útválasztó táblázatok feltöltéséért és karbantartásáért felelős – itt kerülnek be a képbe az útválasztó algoritmusok.

Függetlenül attól, hogy az útvonalakat minden egyes csomagra egymástól függetlenül választják ki vagy csak új összeköttetés létrejöttekor, vannak bizonyos tulajdonságok, amelyek kívánatosak egy útválasztó algoritmus esetében. Ezek a következők: helyesség, egyszerűség, robusztusság, stabilitás, igazságosság, optimalitás és hatékonyság. A helyesség és az egyszerűség magától értetődő, a robusztusság azonban elsőre talán kevésbé nyilvánvaló. Amikor egy nagyobb hálózatot üzembe helyeznek, elvárják, hogy az évekig rendszerszintű hibáktól mentesen működjön. Ez alatt az idő alatt mindenféle hardver- és szoftverhiba adódik. Hosztok, útválasztók és vonalak többször is tönkremennek és megjavulnak, és a topológia is sokszor fog változni. Az útválasztó algoritmusnak képesnek kell lennie arra, hogy megbirkózzon a topológiában és a forgalomban bekövetkező változásokkal anélkül, hogy az összes hoszton futó összes munkát meg kellene szakítani. Képzeljük el, hogy mekkora probléma lenne az, ha a hálózatot mindig újra kellene indítani, ahányszor csak valamelyik útválasztó összeomlik.

A stabilitás szintén az útválasztó algoritmus fontos célja. Léteznek algoritmusok, amelyek soha nem közelítik meg az egyensúlyi állapotot (egy fix útvonalhalmazt), bármilyen hosszan fussanak is. Egy stabil algoritmus eléri az egyensúlyt, és meg is őrzi azt. Ennek gyorsan kell történnie, mivel a kommunikáció megszakadhat, mire az útválasztó algoritmus elérné az egyensúlyi állapotot.

Az igazságosság és hatékonyság ugyancsak nyilvánvalónak tűnik – bizonyára senkinek nem lenne kifogása ellenük –, de amint látható, ezek gyakran egymásnak ellentmondó célok. Erre a konfliktusra látható egy egyszerű példa az 5.5. ábrán. Tegyük fel, hogy elegendő forgalom van A és A', B és B', illetve C és C' között ahhoz, hogy telítődjenek a vízszintes adatkapcsolatok. A teljes adatfolyam maximalizálása érdekében az X és X' közti forgalmat teljesen be kellene szüntetni. Sajnos elképzelhető, hogy X és X'



5.5. ábra. *Hálózati konfliktus az igazságosság és az optimalitás között*

ezt nem így látja. Ezért kompromisszumot kell kötni a globális hatékonyság és az egyes összeköttetések azonos elbírálása között.

Mielőtt megpróbálnánk kompromisszumot keresni az igazságosság és hatékonyság között, el kell határoznunk, mit is akarunk optimalizálni. A hálózati forgalom hatékony küldésénél az egyik nyilvánvaló jelölt az átlagos csomagkésleltetés minimalizálása, de ugyanígy a teljes hálózati átbocsátóképesség maximalizálása is. Ez a két cél is ütközik, mivel ha bármely sorbanállási rendszert a teljesítőképessége határán működtetünk, az hosszú sorbanállási késleltetést von maga után. Kompromisszumként sok hálózatban a csomagok által megtett utat, illetve az ugrások számát próbálják meg minimalizálni, mivel ez csökkenti a csomagonként felhasznált sávszélességet, és ezáltal az átbocsátóképesség is javul.

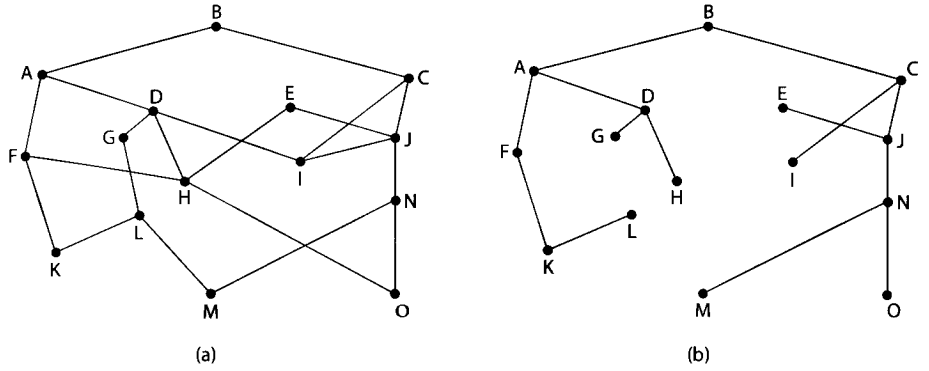
Az útválasztó algoritmusok két nagy osztályba sorolhatók: adaptív és nem adaptív algoritmusok. A **nem adaptív algoritmusok** (**nonadaptive algorithms**) döntéseikben nem támaszkodnak mérésekre vagy becslésekre az aktuális forgalomról és topológiáról. Ehelyett az *I*-től *J*-ig használandó útvonalat (minden *I*-re és *J*-re) előre, offline módon számolják ki, és a hálózat indításakor letöltik az útválasztókba. Ezt az eljárást néha **statikus útválasztásnak** (**static routing**) is szokták nevezni. Mivel a statikus útválasztás nem reagál a hibákra, főként olyan helyzetekben hasznos, ahol az útválasztás egyértelmű. Például az 5.3. ábrán látható *F* útválasztó a hálózatra küldött csomagokat a célcímtől függetlenül az *E* útválasztónak küldi.

Ezzel ellentétben az **adaptív algoritmusok** (**adaptive algorithms**) úgy változtatják útválasztó döntéseiket, hogy azok tükrözzék a topológiában és néha a forgalomban is történt változásokat. Ezek a **dinamikus útválasztó** algoritmusok abban különböznek egymástól, hogy honnan kapják az információt (például helyileg, a szomszédos útválasztóktól vagy az összes útválasztótól), mikor változtatják meg az útvonalakat (például amikor a topológia változik, vagy minden ΔT másodpercben, amikor a terhelés változik), és milyen metrikát használnak az optimalizáláshoz (például a távolságot, az ugrások számát vagy a becsült áthaladási időt).

A következő szakaszokban különböző útválasztó algoritmusokat fogunk tárgyalni. Az algoritmusok egy csomagnak a forrástól a cél felé történő küldésén kívül kézbesítési modelleket is magukban foglalnak. Bizonyos esetekben a cél az, hogy a csomagot több címzettnek, az összes címzettnek vagy egy adott címzettnek küldjük. Az itt leírt összes algoritmus a topológia alapján hoz döntést. A forgalomalapú döntések leírását az 5.3. alfejezet tartalmazza.

5.2.1. Az optimalitási elv

Mielőtt beleásnánk magukat az egyes algoritmusokba, segítségünkre lehet, hogy az optimális útvonalakról egy általános megállapítás tehető, a hálózat topológiájától és a forgalomtól függetlenül. Ez a megállapítás az **optimalitási elv** (**optimality principle**) néven ismeretes [Bellman, 1957]. Az állítás szerint, ha a *J* útválasztó az *I* útválasztótól *K* útválasztó felé vezető optimális útvonalon helyezkedik el, akkor a *J*-től *K*-ig vezető útvonal ugyanerre esik. Hogy ezt belássuk, nevezzük az útvonal *I*-től *J*-ig tartó részét r_1-nek és az útvonal másik részét r_2-nek. Ha létezne egy r_2-nél jobb, *J*-től *K*-ig tartó útvonal, ezt összefűzhetnénk r_1-gyel, hogy az *I*-től *K*-ig tartó útvonalon is javítsunk. Ez viszont ellentmond állításunknak, miszerint r_1r_2 optimális.



5.6. ábra. *(a) Egy hálózat. (b) Egy nyelőfa a* B *útválasztóhoz*

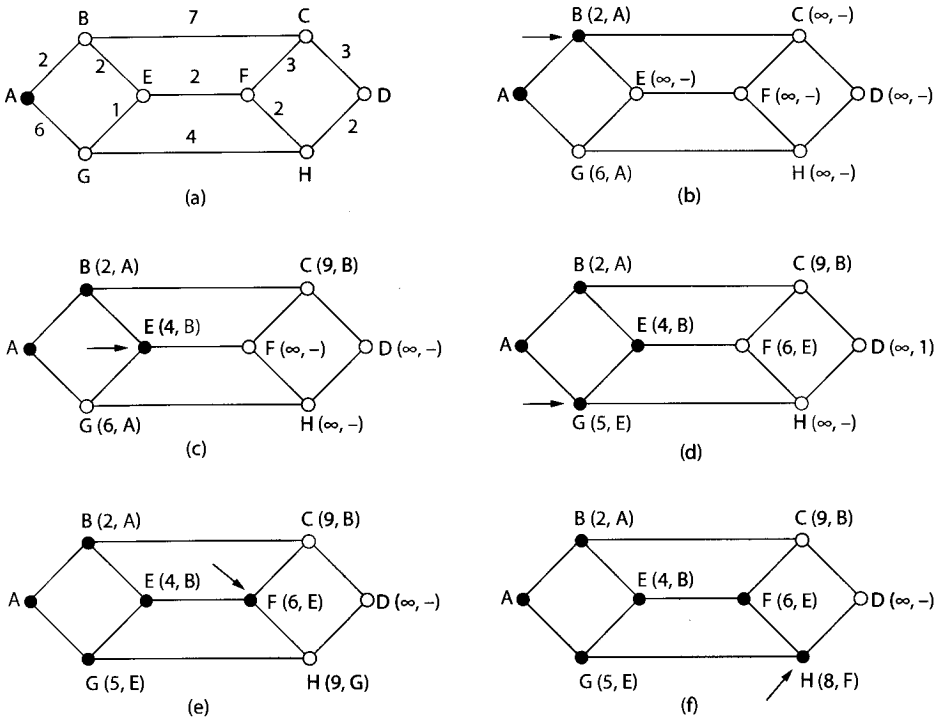
Az optimalitási elv egyenes következményeként beláthatjuk, hogy az összes forrásból egy adott célba tartó optimális útvonalak egy fát alkotnak, amelynek gyökere a cél. Az ilyen fákat **nyelőfának** (**sink tree**) nevezzük. Egy ilyen látható az 5.6. ábrán, ahol a távolság mértéke az ugrások száma. Vegyük észre, hogy a nyelőfa nem feltétlenül egyedi; más fák is létezhetnek ugyanolyan úthosszakkal. Minden útválasztó algoritmus célja a nyelőfák felderítése és használata az összes útválasztó számára.

Vegyük figyelembe, hogy a nyelőfa nem feltétlenül egyedi, létezhetnek azonos úthosszal más fák is. Ha megengedjük, hogy az összes lehetséges útvonal kiválasztható legyen, akkor a fa egy általánosabb adatszerkezetté válik, amelyet irányított aciklikus gráfnak, **DAG**-nak (**Directed Acycle Graph – irányított aciklikus gráf**) hívunk. A DAG nem tartalmaz hurkokat. Az egyszerűség kedvéért a nyelőfát, mint egy kényelmes jelölést mindkét típusra alkalmazzuk. Mindkét eset azon a műszaki feltételezésen alapul, hogy az útvonalak nem zavarják egymást, azaz az egyik útvonalon lévő forgalmi dugó nem okozza a másik útvonal eltérülését.

Mivel a nyelőfa valóban fa, nem tartalmaz hurkot, ezért minden csomag véges és korlátos számú ugráson belül kézbesítésre kerül. A gyakorlatban az élet nem ilyen könnyű. Adatkapcsolatok és útválasztók tönkremehetnek és megjavulhatnak működés közben, ezért különböző útválasztóknak különböző elképzeléseik lehetnek a pillanatnyi topológiáról. Szintén csendben elmentünk azon kérdés mellett, hogy vajon az útválasztóknak egyénileg kell-e beszerezni az információt, amelyre a nyelőfaszámítás épül, vagy ezt az információt valami más módszerrel gyűjtik össze. Ezekre a kérdésekre hamarosan visszatérünk. Mindazonáltal az optimalitási elv és a nyelőfa egy mérőszámot adnak, amelyhez más útválasztó algoritmusokat viszonyíthatunk.

5.2.2. Legrövidebb útvonal alapján történő útválasztás

Kezdjük az útválasztó algoritmusok tanulmányozását egy egyszerű módszerrel, amely kiszámítja az optimális útvonalakat, és a hálózat teljes képét megadja. Azt szeretnénk, hogy az osztott útválasztó algoritmus még akkor is megtalálja ezeket az útvonalakat, ha nem minden útválasztó ismeri a hálózat összes részletét.



5.7. ábra. *Az első hat lépés az* A*-tól* D *felé vezető legrövidebb út kiszámításában. A nyilak a munkacsomópontot jelzik*

Az ötlet az, hogy építsük fel a hálózat egy gráfját, ahol minden útválasztó egy csomópontnak, és minden él egy kommunikációs vonalnak (vagy adatkapcsolatnak) felel meg. Két adott útválasztó közötti útvonal kiválasztásához az algoritmus egyszerűen a köztük levő legrövidebb utat keresi meg a gráfban.

A **legrövidebb út** (**shortest path**) fogalma némi magyarázatra szorul. Egy út hosszát mérhetjük például a megtett ugrások számával. Ezzel a mértékkel az 5.7. ábrán szereplő *ABC* és *ABE* út ugyanolyan hosszú. Egy másik mérték lehet a földrajzi távolság kilométerekben, amikor is *ABC* nyilvánvalóan sokkal hosszabb, mint *ABE* (feltéve, hogy az ábra léptékhelyes).

Ám az ugrások száma és a földrajzi távolság mellett sok egyéb mérték lehetséges. Például mindegyik él súlya lehetne egy szabványos tesztcsomagra vonatkozó átlagos sorbanállási és átviteli késleltetés, amelyet óránkénti próbafuttatásokkal mérnénk. Ezzel az élsúlyozással a legrövidebb út a leggyorsabb út lesz, a legkevesebb kilométerű vagy legkevesebb élből álló helyett.

A legáltalánosabb esetben az élsúlyok a távolság, a sávszélesség, az átlagos forgalom, a kommunikációs költség, az átlagos sorhosszúság, a mért késleltetés és más értékek függvényeként számíthatók ki. A súlyozó függvény változtatásakor az algoritmus a „legrövidebb" utat a számos feltétel közül valamelyik vagy ezek kombinációja szerint számolná ki.

Számos algoritmus ismert egy gráf két csomópontja közti legrövidebb út kiszámítására. Az alábbi Dijkstrától [1959] származik, és a hálózatban a forráscsomópont és az összes többi csomópont mint nyelőcsomópont közötti legrövidebb utakat határozza meg. Minden csomópontot felcímkézünk (zárójelben) a forráscsomóponttól való legrövidebb ismert út mentén mért távolságával. A távolságértékek nem lehetnek negatív értékek, mivel olyan valós mennyiségeken alapulnak, mint amilyen a sávszélesség vagy a késleltetés. Kezdetben nem ismertek ilyen utak, ezért minden csomópont címkéje végtelen. Ahogy az algoritmus halad előre és találja meg az utakat, a címkék módosulhatnak, jelezve az egyre jobb utakat. Egy címke lehet ideiglenes vagy állandó. Kezdetben minden címke ideiglenes. Amikor kiderül, hogy a címke a legrövidebb lehetséges utat jelzi, állandóvá tesszük, és ezek után már nem módosítjuk.

A címkéző algoritmus bemutatásához nézzük az 5.7.(a) ábrán szereplő irányítatlan, súlyozott gráfot, ahol a súlyok például távolságot jeleznek. Az *A*-tól *D*-ig vezető legrövidebb utat akarjuk megtalálni. Azzal kezdjük, hogy az *A* csomópontot állandónak jelöljük meg, amit a kör besötétítésével jelzünk. Ezután sorban végigvizsgáljuk az *A*-val (a munkacsomóponttal) szomszédos csomópontokat, és újracímkézzük az *A*-tól való távolságukkal. Amikor ezeket a csomópontokat újracímkéztük, akkor azzal a csomóponttal is felcímkézzük ezeket, amelyiktől a vizsgálatot végeztük, hogy később rekonstruálhassuk a végső utat. Ha több legrövidebb útvonal létezik *A* és *D* között, és az összeset meg akarjuk találni, akkor az összes megvizsgált csomópontot fel kell jegyezni, amely egy adott csomópontot azonos hosszúságú úttal tud elérni.

Miután *A* minden szomszédját megvizsgáltuk, az egész gráfban levő ideiglenesen felcímkézett csomópontok közül a legkisebb címkéjűt állandóvá tesszük, amint az az 5.7.(b) ábrán látszik. Ez lesz az új munkacsomópont.

Most *B*-től kezdjük, és az összes szomszédját megvizsgáljuk. Ha *B* címkéjének és a *B* és a vizsgálandó csomópont közti él súlyának összege kisebb, mint a vizsgálandó csomópont címkéje, akkor rövidebb utat találtunk, és a csomópontot újracímkézzük.

Miután a munkacsomópont összes szomszédját megvizsgáltuk, és ha lehetett, az ideiglenes címkéket újra cseréltük, az egész gráfból kikeressük a legkisebb értékű ideiglenes címkéjű csomópontot. Ezt állandóvá tesszük, és ez lesz a következő körben a munkacsomópont. Az 5.7. ábrán az algoritmus első hat lépése látható.

Hogy lássuk, miért működik az algoritmus, nézzük az 5.7.(c) ábrát. Ezen a ponton éppen *E*-t tettük állandóvá. Tegyük fel, hogy volna *ABE*-nél rövidebb út, mondjuk *AXYZE*. Két lehetőség van: vagy már állandóvá tettük a *Z* csomópontot, vagy még nem. Ha igen, akkor az *E*-t már megvizsgáltuk (abban a körben, ami előtt *Z*-t állandóvá tettük), tehát az *AXYZE* nem került el a figyelmünket és így nem lehet a legrövidebb út.

Most nézzük azt az esetet, amikor *Z* még mindig ideiglenesen van felcímkézve. *Z* címkéje vagy nagyobb, vagy egyenlő *E*-vel, amikor is *AXYZE* nem lehet *ABE*-nél rövidebb út, vagy *E*-énél kisebb, amikor is *Z*, és nem *E* lesz előbb állandó, lehetővé téve, hogy *E*-t *Z*-ből vizsgálhassuk.

Ezt az algoritmust az 5.8. ábrán adjuk meg. Az *n* és *dist* globális változók írják le a gráfot; ezek még a *shortest_path* eljárás meghívása előtt inicializálásra kerülnek. A program és a fentebb leírt algoritmus között az egyetlen különbség, hogy az 5.8. ábrán a legrövidebb utat a végcsomóponttól, *t*-től számítjuk, a forráscsomópont, *s* helyett.

Mivel egy irányítatlan gráfban a *t*-től *s*-ig vezető legrövidebb út ugyanaz, mint az *s*-től

t-ig vezető, nem számít, melyik végén kezdjük (hacsak nincs több legrövidebb út, amikor is a keresés megfordítása egy másikat derítene fel). A visszafelé keresésnek az az oka, hogy minden csomópontot a megelőző, és nem a következő csomóponttal címkéztünk fel. Amikor a végső utat a kimeneti változóba, a *path*-ba másoljuk, az útvonal megfordul. A két visszafordítás kioltja egymást, és a válasz a helyes sorrendben kerül kiadásra.

```
#define MAX_NODES 1024                  /* a csomópontok számának maximuma */
#define INFINITY 1000000000             /* egy szám, amely nagyobb, mint bármely
                                           leghosszabb út */
int n, dist[MAX_NODES][MAX_NODES]       /* dist[i][j] a távolság i-től j-ig */

void shortest_path(int s, int t, int path[])
{ struct state {                        /* a munka tárgyát képező út */
    int predecessor;                    /* előző csomópont */
    int length;                         /* a forrástól eddig számolt hossz */
    enum {permanent, tentative} label;  /* a címke állapota */
} state[MAX_NODES];

int i, k, min;
struct state *p;

for (p = &state[0]; p <&state[n]; p++) {        /* állapotinicializálás */
    p->predecessor = -1;
    p->length = INFINITY;
    p->label = tentative;
}
state[t].length = 0; state[t].label = permanent;
k = t;                                  /* k a kezdeti munkacsomópont */
do {                                    /* Van jobb út k felől? */
    for (i = 0; i < n; i++)             /* ennek a gráfnak n csomópontja van */
      if (dist[k][i] ! = 0 && state[i].label == tentative) {
        if (state[k].length + dist[k][i] < state[i].length) {
            state[i].predecessor = k;
            state[i].length = state[k].length + dist[k][i];
        }
      }

    /* Keressük meg a legkisebb címkéjű ideiglenesen felcímkézett csomópontot! */
    k = 0; min = INFINITY;
    for (i = 0; i <n; i++)
        if (state[i].label == tentative && state[i].lenth < min) {
            min = state[i].length;
            k = i;
        }
    state[k].label = permanent;
  ) while (k != s);

  /* Másoljuk az utat a kimeneti tömbbe! */
  i = 0; k = s;
  do {path[i++] = k; k = state[k].predecessor;} while (k >= 0);
}
```

5.8. ábra. *Dijkstra algoritmusa egy gráfon keresztülhaladó legrövidebb út kiszámítására*

5.2.3. Elárasztás

Az útválasztó algoritmus megvalósításakor minden útválasztónak helyi ismeretek alapján kell döntéseket hoznia, nem pedig a hálózat teljes képe alapján. Egy egyszerű helyi módszer az **elárasztás (flooding)**, amelyben minden bejövő csomagot minden kimenő vonalon kiküldünk, kivéve azon, amelyiken beérkezett.

Az elárasztás nyilván nagyszámú kettőzött csomagot eredményez, valójában végtelen számút, hacsak nem teszünk intézkedéseket a folyamat visszafogására. Ilyen intézkedés, hogy legyen minden csomag fejrészében egy ugrásszámláló, amely minden ugráskor csökken eggyel, és ha eléri a nullát, a csomagot eldobjuk. Ideális esetben az ugrásszámláló kezdeti értékének a forrástól a célig vezető út hosszát kell beállítani. Ha a küldő nem tudja, milyen hosszú az út, beállíthatja a legrosszabb esetre, nevezetesen a hálózat teljes átmérőjére.

Az ugrásszámlálót alkalmazó elárasztás exponenciálisan megtöbbszörözheti a csomagokat az ugrásszám növekedésével, mert az útválasztók megduplázzák azokat a csomagokat, amelyeket már láttak. Az elárasztás gátak közé szorításának másik módja az, ha nyilvántartjuk, mely csomagokkal végeztük már el az elárasztást, így elkerülve, hogy másodszor is kiküldjük azokat. E cél elérésének egyik módja az, hogy a forrásútválasztó minden csomagba elhelyez egy sorszámot, amelyet a hosztjaitól kap. Ezek után minden útválasztónak szüksége van forrásútválasztónként egy listára, amely megmondja, mely sorszámokat látott már ebből a forrásból. Ha a beérkező csomag rajta van a listán, az útválasztó nem küldi tovább szomszédjainak.

A lista korlát nélküli növekedésének meggátolása érdekében minden listát ki kell egészíteni egy számlálóval, *k*-val, amelynek az a jelentése, hogy *k*-ig minden sorszám előfordult már. Amikor a csomag megérkezik, a sorszám és a *k* értékének összehasonlításával egyszerűen ellenőrizhető, hogy az másodpéldány-e. Ha igen, el kell dobni. Ráadásul az egész *k* alatti listára nincs is szükség, mivel *k* hatékonyan magába foglalja azt.

Az elárasztás sok csomag elküldésénél nem praktikus, de van néhány fontos alkalmazási helye. Először is, ez biztosítja, hogy egy csomag a hálózat összes csomópontjához eljusson. Ez azonban pazarlás lehet, ha egyetlen címzettnek van szüksége a csomagra, de adatszórásnál hatékony módszer. Vezeték nélküli hálózatokban egy állomás által továbbított összes üzenetet az adott állomással azonos vételkörzetben található összes állomás veheti, ami lényegében elárasztásnak tekinthető, és néhány algoritmus ki is használja ezt a tulajdonságot.

Másodszor az elárasztás rendkívül robusztus. Még nagyszámú útválasztó kiesése esetén (például háborús zónában lévő katonai hálózatban) is talál útvonalat – már amennyiben legalább egy létezik – a csomag célba juttatásához. Az elárasztáshoz nem szükséges túl sok előzetes beállítás. Ez azt jelenti, hogy használható más hatékonyabb, de több beállítást igénylő útválasztó algoritmus alapjaként. Az útválasztónak csak a szomszédjait kell ismernie. Az elárasztás mértékként is használható más útválasztó algoritmusok összehasonlításához. Az elárasztás mindig a legrövidebb utat választja, mert az összes lehetséges utat egyszerre választja. Ebből következően nincs olyan algoritmus, amely rövidebb késleltetést tudna produkálni (ha az elárasztási folyamatból adódó többletterhet elhanyagoljuk).

5.2.4. Távolságvektor-alapú útválasztás

A számítógép-hálózatok általában dinamikus algoritmust használnak, ami lényegesen összetettebb, mint az elárasztás, de sokkal hatékonyabb is, mivel megtalálja a legrövidebb utat az aktuális topológiában. Két dinamikus algoritmus is igen népszerű: a távolságvektor-alapú és a kapcsolatállapot-alapú útválasztás. Ebben a szakaszban az előbbit, a következőben pedig az utóbbit fogjuk tanulmányozni.

A **távolságvektor-alapú útválasztás** (**distance vector routing**) alapja, hogy minden útválasztónak egy táblázatot (vagyis egy vektort) kell karbantartania, amelyben minden célhoz szerepel a legrövidebb ismert távolság, és annak a vonalnak az azonosítója, amelyiken a célhoz lehet eljutni. A táblázatokat a szomszédokkal való információcsere útján frissítik. Végül minden útválasztó tudni fogja a legjobb adatkapcsolatot minden célcím megtalálásához.

A távolságvektor-alapú útválasztó algoritmust néha máshogy is nevezik, mint például elosztott **Bellman–Ford útválasztó algoritmus** – az algoritmust kifejlesztő kutatók után [Bellman,1957; Ford és Fulkerson,1962]. Ez volt az ARPANET eredeti útválasztó algoritmusa és ezt használták az interneten is RIP néven.

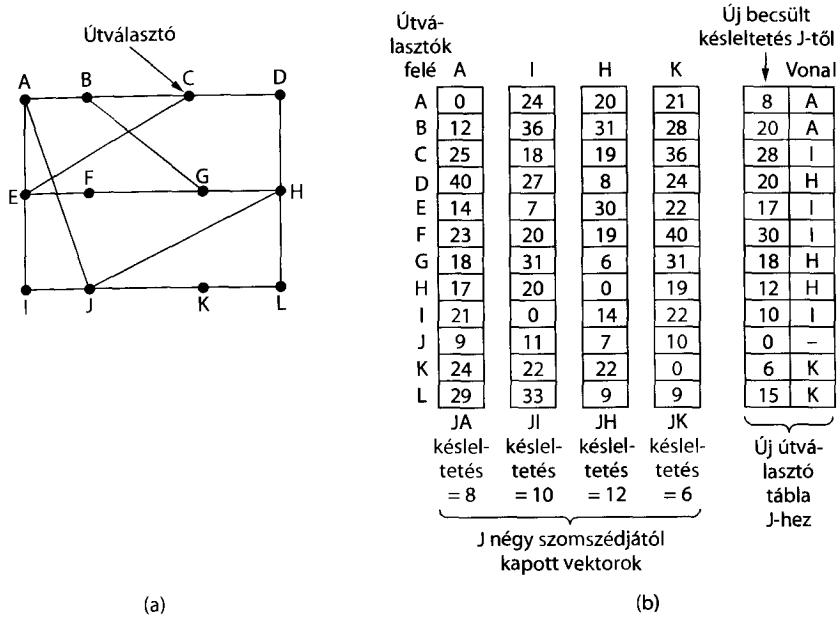
A távolságvektor-alapú útválasztó algoritmusban minden útválasztó karbantart egy táblázatot, amelyet a hálózatban levő összes útválasztó szerint indexelnek, és amely mindegyikhez tartalmaz egy bejegyzést. Ez a bejegyzés két részből áll: az adott célhoz előnyben részesített kimeneti vonalból és a becsült távolságból. A távolság mérhető az ugrások számával vagy egyéb mértékkel (például késleltetéssel), amelyet a legrövidebb utak kiszámításánál megbeszéltünk.

Feltételezzük, hogy az útválasztó tudja a szomszédjaitól való „távolságát". Ha a mérték az ugrások száma, akkor a távolság csak egy ugrás. Ha a mérték a sorhossz, akkor az útválasztó egyszerűen megvizsgál minden sort. Ha a mérték a késleltetés, az útválasztó ezt közvetlenül mérheti különleges *ECHO* csomagokkal, amelyeket a vevő csak időbélyeggel lát el és visszaküld, amilyen gyorsan csak tudja.

Példának okáért tételezzük fel, hogy a mérték a késleltetés, és az útválasztó ismeri az összes szomszédja felé a késleltetést. T ms-onként minden útválasztó minden szomszédjának listát küld az összes cél felé becsült késleltetéseiről. És minden szomszédjától kap egy hasonló listát. Képzeljük el, hogy épp most érkezett egy táblázat az X szomszédtól, amely tartalmazza X_i-t, vagyis X becslését arról, hogy mennyi időbe telik az i útválasztóig eljutni. Ha az útválasztó tudja, hogy a késleltetés X felé m ms, akkor azt is tudja, hogy az i útválasztót X-en keresztül $X_i + m$ ms idő alatt tudja elérni. Ezt a számolást minden szomszédra elvégezve, egy útválasztó megtudhatja, melyik becslés tűnik a legjobbnak, és a következőkben ezt a becslést és a hozzá tartozó vonalat használja az új útválasztó táblázatban. Vegyük észre, hogy a régi útválasztó táblázatot nem használják a számításhoz.

Ezt a frissítési folyamatot illusztrálja az 5.9. ábra. Az (a) rész egy hálózatot ábrázol. A (b) rész első négy oszlopa a J útválasztó szomszédjaitól kapott késleltetési vektorokat mutatja. A állítása szerint B felé 12, C felé 25, D felé 40 ms-os késleltetése van. Tegyük fel, hogy J a szomszédjai, A, I, H és K felé a késleltetéseket sorrendben 8, 10, 12 és 6 ms-ra méri vagy becsüli.

Gondoljuk el, hogyan számítja ki J a G felé vezető új útvonalat. Tudja, hogy el tud jutni A-ba 8 ms alatt, és A állítása szerint képes 18 ms alatt eljutni G-be, tehát J tudja,



5.9. ábra. *(a) Egy hálózat. (b) J-hez érkező késleltetési vektorok* A, I, H, K *felől, és J új útválasztó táblázata*

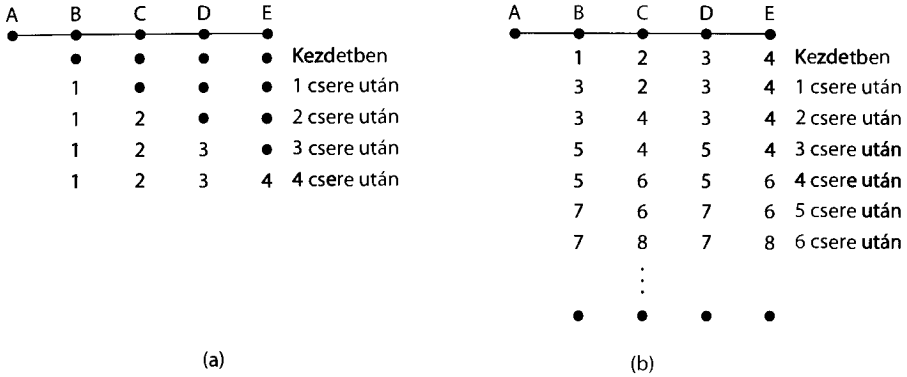
hogy 26 ms-os késleltetéssel kell számolnia, ha a G felé tartó csomagokat A-nak továbbítja. Hasonlóan, a G felé I-n, H-n és K-n keresztül tapasztalható késleltetés sorrendben 41-nek (31 + 10), 18-nak (6 + 12), és 37-nek (31 + 6) adódik. Ezen értékek közül a 18 a legjobb, így bejegyzi az útválasztó táblázatába, hogy G-ig a késleltetés 18 ms, és a használandó útvonal H-n keresztül vezet. Ugyanezt a számolást kell elvégezni az összes többi célra is. Az új útválasztó táblázat az ábra utolsó oszlopában látható.

A végtelenig számolás problémája

A hálózatban a lehetséges útvonalak közül a legjobb út beállítását **konvergenciának** nevezik. A távolságvektor-alapú útválasztás nagyon hasznos, mivel ez egy olyan egyszerű módszer, amellyel az útválasztók együttesen tudják kiszámítani a legrövidebb utat, a gyakorlatban azonban egy komoly hátránya is van: jóllehet konvergál a helyes megoldáshoz, de ezt esetleg nagyon lassan teszi. Pontosabban, gyorsan reagál a jó hírre, de nagyon kényelmesen a rossz hírre.

Hogy lássuk, milyen gyorsan terjed a jó hír, vegyük az 5.10. ábra öt csomópontból álló (lineáris) hálózatát, ahol a késleltetés mértéke az ugrások száma. Tegyük fel, hogy A kezdetben nem működik, és ezt az összes többi útválasztó tudja. Más szavakkal, mind végtelent írtak be az A felé érvényes késleltetéshez.

Amikor A megjavul, a többi útválasztó ezt a vektorcserékből tudja meg. Az egyszerűség kedvéért feltételezzük, hogy létezik valahol egy hatalmas gong, amelynek periodikus



5.10. ábra. *A végtelenig számolás problémája*

ütéseire az útválasztók egyszerre cserélik ki vektoraikat. Az első csere idején *B* megtudja, hogy bal oldali szomszédjának nulla késleltetése van *A*-ig. *B* most bejegyzi az útválasztó táblázatába, hogy *A* egy ugrásnyira van balra. Az összes többi útválasztó még mindig azt gondolja, hogy *A* nem működik. Az ekkor érvényes útválasztó táblázatok *A*-ra vonatkozó bejegyzéseit láthatjuk az 5.10.(a) ábra második sorában. A következő cserekor *C* megtudja, hogy *B*-nek van egy 1 hosszú útja *A* felé, tehát frissíti az útválasztó táblázatát, hogy az egy 2 hosszú utat jelezzen, de *D* és *E* csak később tudja meg a jó hírt. Világos, hogy a jó hír egy ugrás/csere sebességgel terjed. Egy olyan hálózatban, ahol a leghosszabb út *N* ugrás hosszú, *N* csere múlva mindenki tudni fog az újjáéledt adatkapcsolatokról és útválasztókról.

Most vegyük az 5.10.(b) ábrán látható szituációt, ahol kezdetben minden adatkapcsolat és útválasztó működik. A *B*, *C*, *D* és *E* útválasztók távolsága *A*-tól sorrendben 1, 2, 3 és 4. Hirtelen *A* elromlik, vagy más esetben, az *A* és *B* közti vonal megszakad (ami *B* felől nézve gyakorlatilag ugyanaz).

Az első vektorcserénél *B* nem hall semmit *A* felől. Szerencsére *C* azt mondja: „Ne aggódj. Van egy *A*-hoz vezető 2 hosszú utam." *B* nem tudhatja, hogy *C* útja magán *B*-n vezet keresztül. Amennyit *B* tud, aszerint *C*-nek akár tíz kimenő adatkapcsolata is lehet független *A*-hoz vezető utakkal, amelyek mind 2 hosszúak. Ennek eredményeként *B* most azt gondolja, hogy elérheti *A*-t *C*-n keresztül, egy 3 hosszú úttal. *D* és *E* nem frissítik az *A*-ra vonatkozó bejegyzéseiket az első cserekor.

A második cserekor *C* észreveszi, hogy mindkét szomszédja azt állítja, hogy az *A*-hoz vezető út 3 hosszú. Kiválaszt közülük egyet véletlenszerűen, és az *A*-hoz tartozó új távolságot 4-re állítja, mint ahogy az 5.10.(b) ábra harmadik sorában látszik. A sorozatos cserék idézik elő az 5.10.(b) ábra maradék részén mutatott történést.

Ebből az ábrából nyilvánvaló, miért terjed lassan a rossz hír: soha nincs egyik útválasztónak sem több mint eggyel magasabb értéke, mint a szomszédjainak a minimuma. Fokozatosan mindegyik útválasztó felküzdi magát a végtelenig, de az ehhez szükséges cserék száma a végtelen ábrázolásához használt számtól függ. Ennél az oknál fogva okos gondolat a végtelent a leghosszabb útvonalnál 1-gyel hosszabbra állítani.

Nem teljesen meglepő, hogy ez a probléma a **végtelenig számolás (count-to-infinity)** problémájaként ismert. Történt már néhány kísérlet a probléma megoldására, például annak megakadályozásával, hogy az útválasztók hirdessék a legjobb útvonalaikat azon szomszédjaik felé, akiktől hallották azokat, a megosztott láthatár tiltott visszaúttal (split horizon with poisoned reverse) szabályainak megfelelően, ahogy azt az RFC 1058 bemutatja. A jól csengő nevek ellenére ezek közül a heurisztikus módszerek közül egyik sem működik jól a gyakorlatban. A probléma lényege az, hogy ha *X* elmondja *Y*-nak, hogy van egy valahová vezető útja, *Y*-nak sehogy sem áll módjában megtudnia, hogy vajon ő maga rajta van-e ezen az úton.

5.2.5. Kapcsolatállapot-alapú útválasztás

Távolságvektor-alapú útválasztást használt az ARPANET 1979-ig, amikor is ezt felváltotta a kapcsolatállapot-alapú útválasztás. Az elsődleges probléma, amely ehhez a váltáshoz vezetett, hogy az algoritmus egyensúlyba kerülése túl sokáig tartott a hálózati topológia változása után (a végtelenig számolás problémája miatt). Következésképpen ezt egy teljesen új algoritmus váltotta fel, amelyet **kapcsolatállapot-alapú útválasztásnak (link state routing)** neveznek. Manapság a kapcsolatállapot-alapú útválasztás különböző változatait (IS-IS és OSPF) széles körben használják nagy hálózatokban és az interneten.

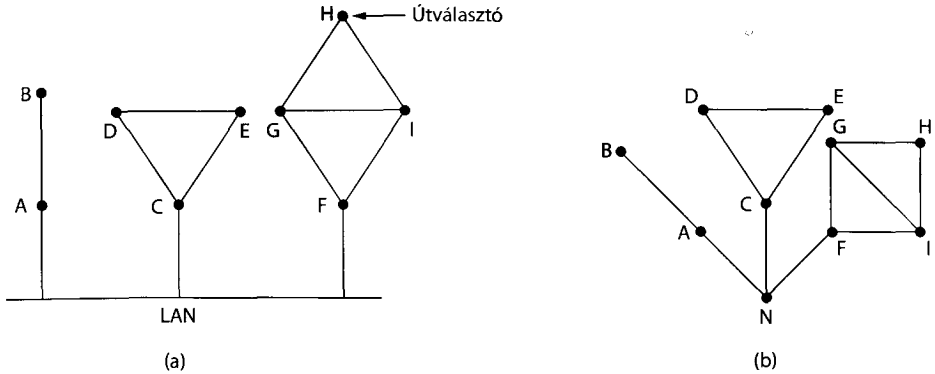
A kapcsolatállapot-alapú útválasztás mögötti ötlet egyszerű, és öt részből tehető összse. Minden útválasztónak a következőket kell tennie:

1. Felkutatni a szomszédjait és megtudni a hálózati címeiket.
2. Beállítani a távolság vagy a költség értékét a minden szomszédjáig mért késleltetés alaján.
3. Összeállítani egy csomagot, amely a most megtudottakat tartalmazza.
4. Elküldeni ezt a csomagot az összes többi útválasztónak.
5. Kiszámítani az összes többi útválasztóhoz vezető legrövidebb utat.

Lényegében a teljes topológia eljut az összes útválasztóhoz. Ezután a Dijkstra-algoritmust futtatják az útválasztók, hogy megtalálják a legrövidebb utat az összes többi útválasztóhoz. A következőkben mind az öt lépést részletesebben is megvizsgáljuk.

A szomszédok megismerése

Amikor egy útválasztó beindul, az első feladata, hogy megtudja, kik a szomszédai. Ezt a célt úgy éri el, hogy egy speciális HELLO csomagot küld ki minden hozzá kapcsolódó kétpontos vonalon. A másik végen levő útválasztótól elvárja, hogy küldjön vissza egy választ, amelyben közli azonosítóját. Ezeknek a neveknek globálisan egyedieknek kell



5.11. ábra. *(a) Kilenc útválasztó és egy üzenetszórásos LAN. (b) Az (a) gráfmodellje*

lenniük, mert amikor később egy távoli útválasztó azt hallja, hogy három útválasztó az *F*-hez csatlakozik, el kell döntenie, hogy mind a három ugyanaz az *F* vagy sem.

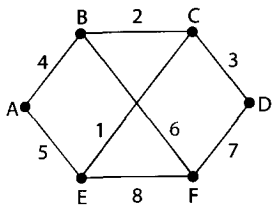
Amikor két vagy több útválasztót üzenetszórásos adatkapcsolat köt össze (például kapcsoló, gyűrű vagy klasszikus Ethernet), a helyzet kicsivel bonyolultabb. Az 5.11.(a) ábra egy üzenetszórásos LAN-t mutat, amelyhez három útválasztó: *A*, *C* és *F* kapcsolódik közvetlenül. Mindegyik útválasztóhoz egy vagy több további útválasztó kapcsolódik, amint azt az ábra mutatja.

Az üzenetszórásos LAN minden csatlakoztatott útválasztópár között kapcsolatot létesít. A LAN több kétpontos típusú adatkapcsolatként történő modellezése azonban növeli a topológia méretét és üzenetek elvesztéséhez vezet. Jobb módszer, ha a LAN-t magát egy csomópontnak tekintjük, amint azt az 5.11.(b) ábra mutatja. Itt egy új, mesterséges csomópontot vezettünk be, az *N*-t, amihez *A*, *C* és *F* kapcsolódik. Kiválasztunk a LAN-on egy **kijelölt útválasztót (designated router)**, amely az *N* szerepét fogja játszani az útválasztó protokollban. A tényt, hogy lehetséges *A*-tól *C*-ig eljutni a LAN-on, itt az *ANC* út mutatja.

Az adatkapcsolat költségének beállítása

A kapcsolatállapot-alapú útválasztó algoritmus megköveteli, hogy minden adatkapcsolat rendelkezzen a legrövidebb út megtalálására vonatkozó távolság vagy költség mérőszámmal. A szomszédok elérésének költsége beállítható automatikusan, vagy azt beállíthatja a hálózat operátora. Általános választási lehetőség az adatkapcsolat költségének beállítására, hogy a költség fordítottan arányos az adatkapcsolat sávszélességével. Például az 1 Gb/s-os Ethernet költsége 1, a 100 Mb/s-os Etherneté pedig 10. Ez a nagyobb kapacitású utakat jobb választásnak tekinti.

Ha a hálózat földrajzilag kiterjedt, akkor az adatkapcsolatok késleltetését is figyelembe lehet venni a költség kiszámításánál, így a rövidebb adatkapcsolaton lévő útvonalak jobb választásnak bizonyulnak. A késleltetés meghatározásának legközvetlenebb módja egy speciális ECHO csomag kiküldése a vonalon, amelyet a másik oldalnak azonnal vissza kell küldenie. A körülfordulási idő megmérésével és kettővel osztásával a küldő útválasztó elfogadható becslést kaphat a késleltetés mértékéről.



Kapcsolatállapot-csomagok

A	
Sorsz.	
Kor	
B	4
E	5

B	
Sorsz.	
Kor	
A	4
C	2
F	6

C	
Sorsz.	
Kor	
B	2
D	3
E	1

D	
Sorsz.	
Kor	
C	3
F	7

E	
Sorsz.	
Kor	
A	5
C	1
F	8

F	
Sorsz.	
Kor	
B	6
D	7
E	8

(b)

5.12. ábra. *(a) Hálózat. (b) A hálózat kapcsolatállapot-csomagjai*

A kapcsolatállapot-csomagok összeállítása

Az információcseréhez szükséges adatok összegyűjtése után a következő lépés, hogy minden útválasztó összeállít egy csomagot, amely az összes adatot tartalmazza. A csomag a feladó azonosítójával kezdődik, amit egy sorszám és egy korérték (ezt később tárgyaljuk), végül pedig a szomszédok listája követ. Minden szomszédhoz megadják a feléje tapasztalható költséget is. Az 5.12.(a) ábrán egy példahálózatot mutat a vonali költségek feltüntetésével. Az 5.12.(b) ábrán mind a hat útválasztóhoz láthatók a megfelelő kapcsolatállapot-csomagok.

A kapcsolatállapot-csomagok összeállítása egyszerű. A nehézséget annak eldöntése jelenti, hogy mikor állítsuk össze azokat. Az egyik lehetőség, hogy periodikusan, vagyis szabályos időközönként. Egy másik lehetőség, hogy amikor valami jelentős esemény történt, mint például egy vonal vagy szomszéd meghibásodott, megjavult vagy megváltoztatta a tulajdonságait.

A kapcsolatállapot-csomagok szétosztása

Az algoritmus legtrükkösebb része a kapcsolatállapot-csomagok szétosztása. Minden útválasztónak az összes kapcsolatállapot-csomagot gyorsan és megbízhatóan meg kell kapnia. Ha a különböző útválasztók esetleg a topológia más-más változatait használják, akkor az általuk kiszámított utak inkonzisztensek lehetnek, előfordulhatnak például hurkok, elérhetetlen gépek és egyéb problémák.

Először leírjuk az alapvető szétosztó algoritmust, majd később pár finomítást is adunk. Az alapötlet az, hogy az elárasztást használjuk a kapcsolatállapot-csomagok szétosztására. Hogy az elárasztást kézben tartsák, minden csomag tartalmaz egy sorszámot, amely minden egyes csomagküldésnél eggyel növekedik. Az útválasztók számon tartanak minden (forrásútválasztó, sorszám) párt, amit csak látnak. Amikor egy új kapcsolatállapot-csomag érkezik, összevetik a már látott csomagok listájával. Ha új, eddig nem látott csomag érkezett, akkor továbbítják minden vonalon, kivéve azon, amelyiken érkezett. Ha másodpéldány, eldobják. Ha egy csomag olyan sorszámmal érkezik, amely kisebb, mint a legnagyobb már látott sorszám, akkor az útválasztó elavult csomagnak tekinti, és nem továbbítja, hiszen az útválasztó rendelkezik már ennél frissebb információval is.

Ennek az algoritmusnak van néhány problémája, de ezek kezelhetők. Először is, ha a sorszámok átfordulnak, zűrzavar fog eluralkodni. Itt az a megoldás, hogy 32 bites sorszámokat kell használni. Másodpercenként egy kapcsolatállapot-csomag elküldését feltételezve, az átfordulás 137 év múlva következne be, így ezt a lehetőséget figyelmen kívül hagyhatjuk.

Másodszor, ha egy útválasztó valamikor összeomlik, elfelejti, melyik sorszámnál tartott. Ha újra 0-ról kezdi, a következő általa küldött csomagot másodpéldánynak fogják tekinteni, és nem továbbítják.

Harmadszor, ha egy sorszám megsérül, és 65 540 érkezik meg 4 helyett (egy 1 bites hiba), az 5-től 65 540-ig tartó csomagokat elavultként visszautasítja, mivel a jelenlegi sorszámot 65 540-nek hiszi.

Mindezekre a problémákra az a megoldás, hogy minden csomagba a sorszáma után a csomag életkorát is bele kell tenni, és ezt másodpercenként eggyel csökkenteni kell. Amikor az életkor eléri a nullát, az attól az útválasztótól származó információt eldobják. Rendszerint, mondjuk 10 másodpercenként érkezik egy új csomag, tehát az útválasztó információja csak akkor válik használhatatlanná, ha egy útválasztó meghibásodott (vagy ha hat egymás után következő csomag elveszett, de ez valószínűtlen). Az életkor mező értékét minden útválasztó csökkenti is a kezdeti elárasztási folyamat során, így biztosítva azt, hogy egyetlen csomag sem veszhet el, és hogy nem élhet közelebbről meg nem határozott ideig (egy nulla életkorú csomag eldobásra kerül).

A robusztusabbá tételhez néhány finomításra van szükség. Amikor egy kapcsolatállapot-csomag beérkezik az útválasztóhoz elárasztásra, nem kerül bele rögtön az adásra várakozó csomagok sorába. Ehelyett egy ideiglenes tárolóterületre kerül, hogy ott egy keveset várakozzon. Ha egy másik kapcsolatállapot-csomag is beérkezik ugyanattól a forrástól, mielőtt az előző továbbítódna, az elküldés előtt az útválasztó összehasonlítja a sorszámukat. Ha egyenlők, a másodpéldányt eldobja kerül. Ha különböznek, a régebbi példány kerül eldobásra. Az útválasztók közti vonalakon bekövetkező hibák kivédésére minden kapcsolatállapot-csomagot nyugtáznak.

Az 5.13. ábrán látható a *B* útválasztó által az 5.12.(a) ábra hálózatához használt adatstruktúra. Minden sor megfelel egy nemrég érkezett, de még nem teljesen feldolgozott kapcsolatállapot-csomagnak. A táblázat rögzíti, honnan indult a csomag, a sorszámát, a korát és az adatokat. Ezenkívül *B* mindhárom adatkapcsolatához (az *A*, *C*, illetve *F* felé

			Küldési jelzőbitek			Nyugtázási jelzőbitek			
Forrás	Sorszám	Kor	A	C	F	A	C	F	Adatok
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

5.13. ábra. *Az 5.12.(a) ábra* B *útválasztójának csomagpuffere*

vezetőkhöz) vannak küldési és nyugtázási jelzőbitek. A küldési jelzőbitek azt jelentik, hogy a csomagot a jelzett adatkapcsolaton tovább kell küldeni. A nyugtázási jelzőbitek azt jelentik, hogy ott a csomagot nyugtázni kell.

Az 5.13. ábrán az *A* kapcsolatállapot-csomagja közvetlenül érkezett, így el kell küldeni *C*-nek és *F*-nek, és nyugtázni *A*-nak, ahogy a jelzőbitek mutatják. Hasonlóan az *F*-től érkezett csomagot továbbítani kell *A*-nak és *C*-nek, és nyugtázni *F*-nek.

A harmadik, *E*-től érkezett csomaggal azonban más a helyzet. Ez kétszer is megjött, egyszer *EAB*-n és egyszer *EFB*-n keresztül. Következésképpen csak *C* felé kell elküldeni, de *A* és *F* felé is nyugtázni kell, ahogy a jelzőbitek mutatják.

Ha egy másodpéldány érkezik, amíg az eredeti még mindig a pufferben van, a jelzőbiteket át kell állítani. Például ha egy másolat érkezik *F* felől *C* állapotáról, mielőtt a táblázat negyedik bejegyzését továbbították volna, a hat jelzőbitet 100011-re kell változtatni, jelezve, hogy a csomagot nyugtázni kell *F* felé, de elküldeni nem.

Új útvonalak kiszámítása

Amint az útválasztó összegyűjtötte a kapcsolatállapot-csomagok teljes készletét, megszerkesztheti a hálózat teljes gráfját, mivel minden adatkapcsolat látható. Valójában minden adatkapcsolat kétszer szerepel, egyszer-egyszer mindkét irányban. A különböző irányokhoz eltérő költség tartozhat. Elképzelhető, hogy a legrövidebb útvonal kiszámítás az *A*-*B* irányhoz más útvonalat talál, mint a *B*-*A* irányhoz.

Most helyileg lefuttatja a Dijkstra-algoritmust, hogy megszerkessze a legrövidebb utat minden lehetséges célhoz. Ennek az algoritmusnak az eredményei mutatják meg az útválasztónak, hogy az egyes célok eléréséhez melyik adatkapcsolatot használja. Ezek az adatok bekerülnek az útválasztó táblába, és ezután visszatérhet a normális működés.

A távolságvektor-alapú útválasztáshoz viszonyítva a kapcsolatállapot-alapú útválasztás több memóriát és számítást igényel. Egy olyan hálózatban, amely n útválasztóból áll, és mindegyik útválasztónak k szomszédja van, a bejövő adat tárolásához szükséges memória mérete kn-nel arányos, ami legalább akkora, mint az összes célt tartalmazó útválasztó tábla. A számítási idő még a leghatékonyabb adatszerkezetek esetén is gyorsabban nő mint kn, és ez nagy hálózatoknál gondot jelenthet. Mindezek ellenére a kapcsolatállapot-alapú útválasztás sok gyakorlati helyzetben jól működik, mivel a lassú egyensúlyba hozás (konvergencia) itt nem jelent problémát.

A kapcsolatállapot-alapú útválasztást széles körben használják a jelenlegi hálózatokban, ezért helyénvaló pár szót szólni az ezeket használó példa-protokollokról. Számos szolgáltató használja az **IS-IS** (**Intermediate System to Intermediate System Intradomain Routing Protocol – közbenső rendszertől közbenső rendszerig történő körzeten belüli útválasztó protokoll**) kapcsolatállapot-alapú protokollt [Oran, 1990], amelyet a DECnet számára terveztek, és amelyet később átvett az ISO is az OSI-protokollhoz. Azóta módosították, hogy más protokollokat is kezeljen, különösen az IP-t. Az **OSPF** (**Open Shortest Path First – nyílt hozzáférésű, a legrövidebb utat előrevevő protokoll**) a másik fontos kapcsolatállapot-alapú protokoll. Ezt az IETF fejlesztette ki évekkel az IS-IS után, az IS-IS számos újítását átörökítve. Ezek közül az újítások közül említhetnénk például a kapcsolatállapot-frissítések elárasztásának önstabilizáló eljárá-

sát, a kijelölt útválasztó koncepcióját egy LAN-on, valamint az utak kettéosztásának és többfajta mérték használatának számítási és támogatási módszerét. Ennek következtében csak nagyon kicsi a különbség az OSPF és az IS-IS között. A legfontosabb eltérés az, hogy az IS-IS egyszerre több hálózati rétegbeli protokollról tud információit szállítani (például IP, IPX és AppleTalk), és ez előnyös nagy, többprotokollos környezetekben, míg az OSPF ezzel a képességgel nem rendelkezik. Az OSPF-et az 5.6.6. szakaszban ismertetjük.

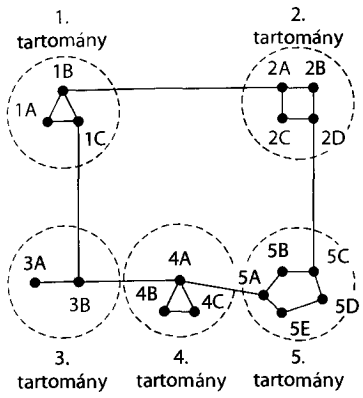
Az útválasztó algoritmusokkal kapcsolatos általános véleménnyel is szolgálunk. A kapcsolatállapot-alapú, a távolságvektor-alapú és egyéb algoritmusok az összes útválasztóra építenek az útvonalak kiszámításánál. A hardver- és szoftverproblémák még kisszámú útválasztó esetén is nagy pusztítást okozhatnak a hálózatban. Ha például egy útválasztó egy olyan adatkapcsolatot tételez fel, amely már nem létezik, vagy elfelejtkezik egy létező adatkapcsolatról, akkor a hálózati gráf helytelen lesz. Ha egy útválasztó nem tud csomagokat továbbítani, vagy elrontja a csomagokat küldés közben, akkor az útvonal nem az elvárt módon fog működni. Végül hibás működéshez vezet az is, ha elfogy a memória vagy helytelen az útválasztási számítás. Ahogy a hálózat növekszik, és több tíz- vagy akár több százezer csomópontot tartalmaz, az egyes útválasztók meghibásodásának valószínűsége már nem elhanyagolható. A fő szempont az, hogy megpróbáljuk minimalizálni a kárt, amikor az elkerülhetetlenül bekövetkezik. Ezekkel a problémákkal és lehetséges megoldásaikkal részletesen Perlman [1988] foglalkozik.

5.2.6. Hierarchikus útválasztás

Ahogy a hálózat mérete nő, az útválasztók útválasztó táblázatai is arányosan nőnek. Az egyre növekvő táblázatok nemcsak az útválasztók memóriáját fogyasztják, hanem több CPU-időre van szükség a táblázatok átvizsgálásához, és nagyobb sávszélesség szükséges ahhoz, hogy elküldjék az ezekről szóló állapotjelentéseket. Egyszer csak a hálózat akkorára nő, hogy már nem lehetséges minden egyes útválasztóban minden más útválasztó számára egy bejegyzést fenntartani, ezért az útválasztást hierarchikusan kell elvégezni, ahogy azt a telefonhálózatban is teszik.

Amikor hierarchikus útválasztást használnak, az útválasztókat úgynevezett **tartományokra (regions)** osztják. Minden útválasztó tudja, hogyan irányítsa a saját tartományán belüli célok felé a csomagokat, de nem tud semmit a többi tartomány belső szerkezetéről. Amikor különböző hálózatokat kapcsolunk össze, természetes, hogy mindegyiket különálló tartománynak tekintjük, és így az egyik hálózaton belüli útválasztóknak nem kell tudniuk semmit a többi hálózat topológiájáról.

Hatalmas hálózatok esetén lehet, hogy nem elég a kétszintű hierarchia. Szükség lehet arra, hogy a tartományokat kerületekbe, a kerületeket zónákba, a zónákat csoportokba szervezzük és így tovább, amíg csak ki nem fogyunk az egyesítésekre használt nevekből. A többszintű hierarchiára példaként nézzük meg, hogyan irányíthatnak egy csomagot a kaliforniai Berkeleyből a kenyai Malindibe. A Berkeleyben lévő útválasztó ismerné a Kalifornián belüli részletes topológiát, de minden, az államból kifelé tartó forgalmat a Los Angeles-i útválasztóhoz küldene. A Los Angeles-i útválasztó képes lenne a többi belföldi útválasztóhoz irányítani a forgalmat, de a külföldi forgalmat New Yorkba küldené. A New York-i útválasztó úgy lenne programozva, hogy minden forgalmat a célország külföldi forgalomért felelős útválasztójához küldjön, mondjuk Nairobiba. Végül a csomag lefelé haladna a kenyai fán belül, amíg el nem érné Malindit.



(a)

1A teljes táblázata

Cél	Vonal	Ugrás
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

1A hierarchikus táblázata

Cél	Vonal	Ugrás
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

(c)

5.14. ábra. *Hierarchikus útválasztás*

Az 5.14. ábra egy mennyiségi példát hoz az útválasztásra egy kétszintű hierarchiában, öt tartománnyal. Az *1A* teljes útválasztó táblázata 17 bejegyzést tartalmaz, ahogy az az 5.14.(b) ábrán látható. Amikor az útválasztást hierarchikusan végezzük, mint az 5.14.(c) ábrán, akkor, mint előzőleg, vannak a helyi útválasztókra vonatkozó bejegyzések, de a többi tartományt egy-egy csomópontba sűrítettük össze. Így a 2. tartomány felé tartó összes forgalom az *1B–2A* vonalon keresztül halad, de a távoli forgalom többi része az *1C–3B* vonalat használja. A hierarchikus útválasztás a táblázat méretét 17 bejegyzésről 7-re csökkentette. Ahogy a tartományok számának tartományonkénti útválasztók számához viszonyított aránya nő, úgy növekszik a megtakarítás a táblázathelyekben.

Sajnos ez a helymegtakarítás nincs ingyen. Ezért büntetést kell fizetni, mégpedig a megnövekedett úthossz formájában. Például az *1A* és *5C* közti legjobb út a 2-es tartományon át vezet, de a hierarchikus útválasztásnál minden, az 5-ös tartomány felé tartó forgalom a 3-as tartományon keresztül zajlik, mert az 5-ös tartományon belül a legtöbb célhoz ez a jobb.

Amikor egy egyedülálló hálózat nagyon nagyra nő, felmerül egy érdekes kérdés: hány szintje legyen a hierarchiának? Vegyünk például egy 720 útválasztóból álló hálózatot. Ha nincs hierarchia, minden útválasztónak 720 táblázatbejegyzésre van szüksége. Ha a hálózatot felosztjuk 24 tartományra, amelyek közül mindegyikben 30 útválasztó van, minden útválasztónak 30 helyi és 23 távoli bejegyzésre van szüksége, ez összesen 53 bejegyzés. Ha háromszintű hierarchiát választunk nyolc kerülettel, amelyek közül

mindegyik 9, tíz útválasztós tartományból áll, minden útválasztónak 10 bejegyzésre van szüksége a helyi útválasztókhoz, 8 bejegyzésre a saját kerületen belüli tartományokhoz, és 7 bejegyzésre a távoli kerületekhez, ez összesen 25 bejegyzés. Kamoun és Kleinrock [1979] felfedezték, hogy egy N útválasztóból álló hálózathoz a szintek optimális száma $\ln N$, amely $e \ln N$ bejegyzést igényel útválasztónként. Azt is megmutatták, hogy a hierarchikus útválasztás által okozott tényleges átlagos úthossznövekedés elegendően kicsi, úgyhogy rendszerint elfogadható.

5.2.7. Adatszóró útválasztás

Néhány alkalmazásban a hosztoknak üzeneteket kell küldeniük néhány másik hosztnak vagy az összes többi hosztnak. Például egy olyan szolgáltatás, amely időjárás-jelentést, tőzsdei híreket vagy élő rádióműsort oszt szét, a legjobban úgy működhet, hogy adatszórással minden géphez adatokat küld, és az érdeklődők elolvashatják azokat. Egy csomag mindenhová történő egyidejű elküldését **adatszórásnak** (**broadcasting**) nevezzük. Különböző módszereket javasoltak ennek megvalósítására.

Az egyik adatszóró eljárás, amely nem igényel semmi különleges képességet a hálózattól, úgy működik, hogy a forrás egyszerűen külön csomagot küld minden egyes rendeltetési helyre. Ez a megoldás nemcsak sávszélesség-pazarló és lassú, hanem azt is megköveteli a forrástól, hogy rendelkezzen a rendeltetési helyek teljes listájával. Ez a legszélesebb körben alkalmazható, de a gyakorlatban a legkevésbé kívánatos eljárás.

Ennek továbbfejlesztése a **többcélú útválasztás** (**multidestination routing**). Itt minden csomag tartalmaz vagy egy listát a rendeltetési helyekről, vagy egy bittérképet, amely a kívánt rendeltetési helyeket jelzi. Amikor egy csomag megérkezik egy útválasztóhoz, az útválasztó megnézi a rendeltetési helyek listáját, hogy eldöntse, mely kimeneti vonalakra lesz szüksége. (Akkor van kimeneti vonalra szüksége, ha az az út a legjobb út legalább egy rendeltetési hely felé.) Az útválasztó előállítja a csomag egy új másolatát minden használandó kimeneti vonal számára, és csak azokat a célcímeket teszi bele, amelyeknek azt a vonalat kell használniuk. Ezzel a rendeltetési helyek halmazát felosztja a kimenő vonalak közt. Elegendő számú ugrás megtétele után minden csomag csak egy címet fog tartalmazni, és normális csomagként kezelhető. A többcélú útválasztás olyan, mint a külön-külön címzett csomagok esete, kivéve azt az esetet, amikor több csomagnak is ugyanazt az útvonalat kell követnie. Ilyenkor az egyik fizeti a teljes viteldíjat, a többi ingyen utazik. A hálózati sávszélesség így hatékonyabban kihasználható. Ebben a sémában a forrásnak továbbra is ismernie kell az összes célt, és az útválasztónak ugyanakkora erőfeszítésbe kerül meghatároznia azt, hogy hova kell küldeni egy többcélú csomagot, mintha több különböző csomagot küldene.

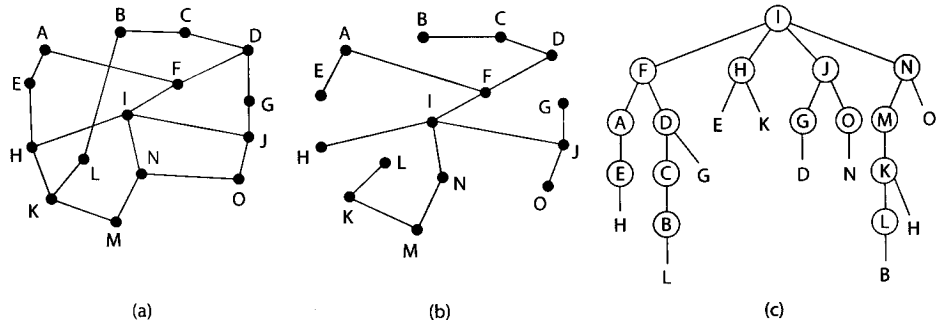
Ennél jobb adatszóró technikát is láttunk már: az elárasztást. Ha a megvalósításnál forrásonkénti sorszámozást alkalmazunk, akkor az elárasztás hatékonyan használja a vonalakat az útválasztók döntési szabályával, amely viszonylag egyszerű. Jóllehet az elárasztás nem megfelelő a szokásos kétpontos kommunikációhoz, de adatszórás esetén érdemes megfontolni az alkalmazását. Az is kiderül azonban, hogy ennél jobb megoldást is találhatunk, ha már egyszer kiszámítottuk a közönséges csomagok legrövidebb útvonalát.

A **visszairányú továbbítás** (**reverse path forwarding**) alapjául szolgáló ötlet elegáns és figyelemre méltóan egyszerű, ha már egyszer megmutatták [Dalal és Metcalfe, 1978]. Amikor egy adatszórásos csomag megérkezik egy útválasztóhoz, az útválasztó ellenőrzi, hogy azon az adatkapcsolaton kapta-e meg, amelyen rendszerint ő szokott az adatszórás forrásához csomagot küldeni. Ha igen, akkor nagy esély van rá, hogy az adatszórásos csomag a legjobb utat követte a szomszédos útválasztótól, és ezért ez az első másolat, amely megérkezett a tekintett útválasztóhoz. Ha ez az eset, az útválasztó másolatot továbbít minden adatkapcsolatra, kivéve arra, amelyiken érkezett. Viszont, ha az adatszórásos csomag más adatkapcsolaton érkezett, mint amit a forrás eléréséhez előnyben részesített, a csomagot eldobja, mint valószínű másodpéldányt.

A visszairányú továbbítás algoritmusára az 5.15. ábrán láthatunk példát. Az (a) rész a hálózatot, a (b) rész a hálózat I útválasztójának nyelőfáját, a (c) rész pedig a visszairányú algoritmus működését szemlélteti. Az első ugrás során I csomagokat küld F-nek, H-nak, J-nek és N-nek, ahogy azt a fa második sora mutatja. Ezen csomagok közül mindegyik az I felé vezető előnyben részesített úton érkezett (feltételezve, hogy az előnyben részesített utak egybeesnek a nyelőfával); ezt jelzi a betűk körüli kör. A második ugrás során nyolc csomag keletkezik: minden olyan útválasztó, amely az első ugrás során megkapta a csomagot, létrehoz kettőt. Amint látjuk, mind a nyolc még meg nem látogatott útválasztóhoz érkezik, és öt közülük az előnyben részesített vonalon. Abból a hat csomagból, amelyik a harmadik ugrás során keletkezik, csak három érkezik az előnyben részesített úton (C-hez, E-hez és K-hoz); a többi másodpéldány. Öt ugrás és 24 csomag után az adatszórás befejeződik, ezzel szemben a nyelőfa pontos követésekor négy ugrásra és 14 csomagra lett volna szükség.

A visszairányú továbbítás legfontosabb előnye, hogy hatékony és könnyen megvalósítható. Az adatszórásos csomagot minden adatkapcsolaton csak egyszer küldi ki csakúgy, mint az elárasztásnál, de csak azt követeli meg, hogy az útválasztó tudja: az egyes célok hogyan érhetők el, azt nem, hogy megjegyezze a sorszámokat (vagy egyéb módszert használjon az elárasztás megakadályozására), vagy hogy felsorolja a csomagban az összes célt.

Az utolsó adatszórásos algoritmus javítja a visszairányú továbbítás működését. Explicit módon használja a nyelőfát – vagy egyéb más megszokott feszítőfát – az adatszórást kezdeményező útválasztóhoz. A **feszítőfa** részhalmaza annak a hálózatnak, amely tartalmazza az összes útválasztót, de nem tartalmaz hurkot. A nyelőfák maguk is feszí-



5.15. ábra. *Visszairányú továbbítás. (a) Hálózat. (b) Nyelőfa. (c) A visszairányú továbbítás által felépített fa*

tőfák. Ha minden útválasztó tudja, hogy melyik vonal tartozik a feszítőfához, akkor le tudja másolni a bejövő adatszórásos csomagot a feszítőfa vonalaira, annak a kivételével, amelyen a csomag érkezett. Ez a sávszélesség tökéletes kihasználását biztosítja azáltal, hogy csak a feladat elvégzéséhez elengedhetetlenül szükséges, minimális mennyiségű csomagot generálja. Ha az 5.15. ábra (b) részében lévő nyelőfa kerül felhasználásra feszítőfaként, akkor az adatszórásos csomag a minimális 14 csomaggal kerül továbbküldésre. Az egyetlen probléma, hogy minden útválasztónak ismernie kell néhány feszítőfát a módszer alkalmazásához. Bizonyos esetekben ezek az adatok rendelkezésre állnak (kapcsolatállapot-alapú útválasztás esetén az összes útválasztó ismeri a teljes topológiát, így ki tudnak számítani egy feszítőfát), de néha nem.

5.2.8. Többesküldéses útválasztás

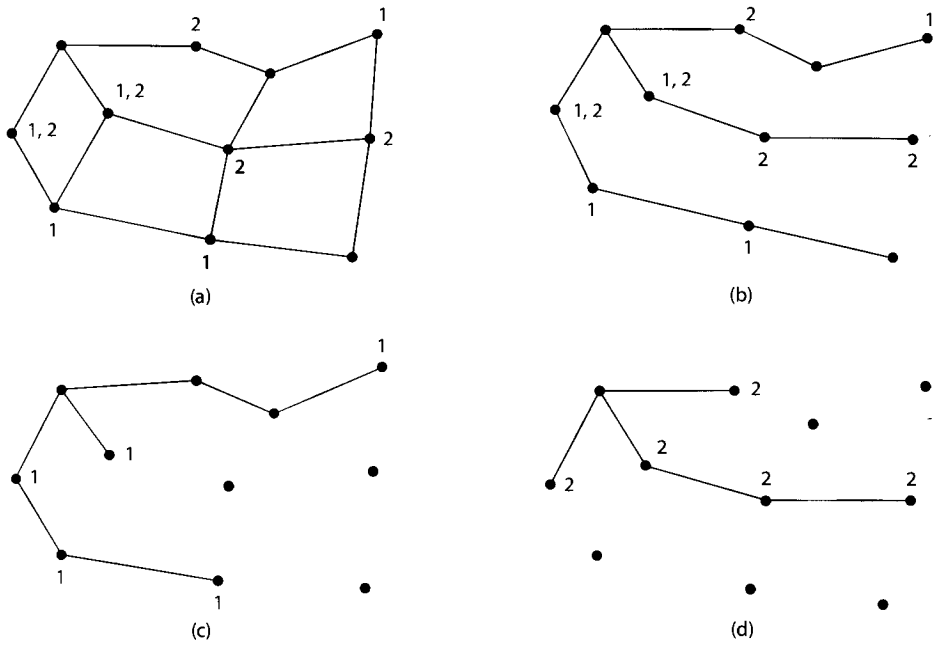
Néhány alkalmazás – például egy többszereplős játék vagy egy sportesemény élő videoközvetítése, amelyet több helyszínen néznek –, több vevőnek küld csomagokat. Hacsak a csoport nem nagyon kicsi, nagyon drága különálló csomagot küldeni minden címzettnek. Másrészről a csomag adatszórásos küldése nagyon pazarló, ha a csoport mondjuk, egy millió csomópontból álló hálózat 1000 számítógépét foglalja magában, mivel a címzettek nagy része nem kíváncsi az üzenetre (vagy ami még rosszabb, lehet, hogy érdekli őket, de nem volna szabad látniuk azt). Ezért szükség van arra, hogy az üzeneteket egy jól meghatározott csoportnak tudjuk küldeni, amelynek tagszáma relatíve nagy, de a teljes hálózathoz képest elenyésző.

Az ilyen csoportnak történő üzenetküldést **többesküldésnek (multicasting)**, a hozzá tartozó útválasztó algoritmust pedig **többesküldéses útválasztásnak (multicast routing)** nevezzük. Az összes többesküldéses séma megköveteli csoportok létrehozását és megsemmisítését, valamint a csoport tagjainak azonosítását. Hogy hogyan valósulnak meg ezek a feladatok, azzal az útválasztó algoritmus nem foglalkozik. Feltételezzük, hogy minden csoportot egy többesküldéses cím azonosít, valamint azt, hogy az útválasztók tudják, hogy melyik csoporthoz tartoznak. A csoporttagsággal újra foglalkozni fogunk az internet hálózati rétegének bemutatásakor, az 5.6. fejezetben.

A többesküldéses útválasztó séma a már tanulmányozott adatszórásos útválasztásra épül, amelyben a csomagok a feszítőfák mentén kerülnek kiküldésre, hogy a csomagokat csak a csoport tagjai kapják meg, a sávszélesség hatékony kihasználása mellett. A legjobb feszítőfa kiválasztása azonban attól függ, hogy vajon a csoport sűrű, a hálózat nagy részén szétszóródó vevőkkel, vagy ritka, a hálózat nagy részén olyan vevőkkel, amelyek nem tartoznak a csoporthoz. Ebben a fejezetben mindkét esetet tárgyaljuk.

Ha a csoport sűrű, akkor az adatszórás jó kiindulás, mivel hatékonyan eljuttatja a csomagot a hálózat minden részébe. De néhány olyan útválasztóhoz is elkerül a csomag, amely nem tagja a csoportnak, és ez pazarlás. Erre a megoldás Deering és Cheriton [1990] felfedezése szerint az, hogy az adatszórásos feszítőfát csonkolni kell azoknak az adatkapcsolatoknak az eltávolításával, amelyek nem a csoport tagjaihoz vezetnek. Ennek eredménye egy hatékony többesküldéses feszítőfa.

Az 5.16.(a) ábrán például egy hálózat látható két csoporttal, 1-gyel és 2-vel. Néhány útválasztó olyan hosztokhoz csatlakozik, amelyek legalább az egyik csoporthoz tartoznak, ahogy az az ábrán látható. A legbaloldalibb útválasztó feszítőfája látható az 5.16.(b) ábrán. Ez a fa használható adatszóráshoz, de többesküldéshez túlzás, ahogy a következő részben látható csonkolt verziók mutatják. Az 5.16.(c) ábrán az összes olyan adatkapcsolat eltávolításra került, amely nem az 1-es csoport tagját képező hoszthoz vezet. A csomagok csak ennek a feszítőfának a mentén kerülnek továbbításra, amely hatékonyabb, mint az adatszórási fa, mivel 10 adatkapcsolat helyett csak 7-et tartalmaz. Az 5.16.(d) ábra a 2-es csoport többesküldéses feszítőfáját mutatja csonkolás után. Ez szintén hatékony, mert csak öt adatkapcsolatot tartalmaz. Ez azt is mutatja, hogy a különböző többesküldéses csoportokhoz különböző feszítőfák tartoznak.



5.16. ábra. *(a) Egy hálózat. (b) A legbaloldalibb útválasztó feszítőfája. (c) Többesküldéses fa az 1-es csoporthoz. (d) Többesküldéses fa a 2-es csoporthoz*

A feszítőfa csonkolásának különféle módjai vannak. A legegyszerűbbet akkor használhatjuk, amikor kapcsolatállapot-alapú útválasztást használunk, és minden útválasztó ismeri a teljes topológiát, beleértve azt is, hogy mely hosztok mely csoportokhoz tartoznak. Minden útválasztó felépítheti a saját csonkolt feszítőfáját az adott csoport minden küldőjéhez a nyelőfa szokásos módon történő létrehozásával, majd eltávolítva azokat az adatkapcsolatokat, amelyek nem a csoporttagokat és a nyelő csomópontot kötik össze. Az **MOSPF (Multicast OSPF – többesküldéses OSPF)** példa az olyan kapcsolatállapot-alapú protokollra, amely ezt az elérést használja [Moy, 1994].

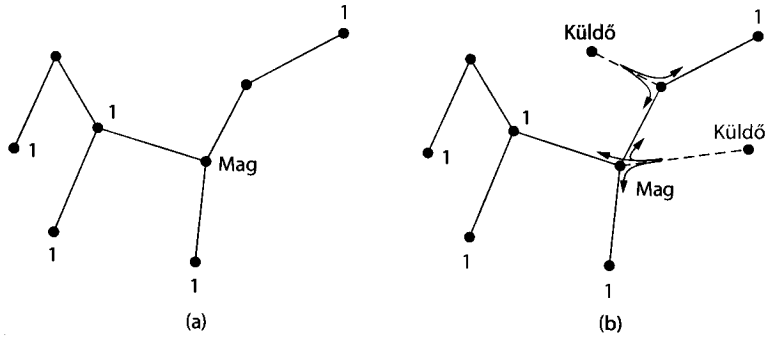
A távolságvektor-alapú útválasztásnál más csonkolási stratégiát követhetünk. Az alapalgoritmus a visszairányú továbbítás. Viszont, amikor egy olyan útválasztó kap többesküldéses üzenetet, amelynek nincs abban a csoportban érdekelt hosztja és nincs más útválasztókhoz adatkapcsolata, PRUNE üzenettel válaszol, utasítva azt a szomszédját,

amelyik az üzenetet küldte, hogy ennek a csoportnak szóló többesküldéseket ne küldjön többet. Amikor egy útválasztó, amelynek nincs csoporttag a hosztjai között, minden vonalán ilyen üzeneteket kapott, ő is PRUNE üzenettel válaszolhat. Ily módon a feszítőfa rekurzívan csonkolásra kerül. A **DVMRP (Distance Vector Multicast Routing Protocol – távolságvektor-alapú többesküldéses útválasztó protokoll)** például egy ilyen módon működő többesküldéses útválasztó protokoll [Waitzman és mások, 1988].

A csonkolás hatékony feszítőfákat eredményez, amelyek csak a csoporttagok eléréséhez ténylegesen szükséges adatkapcsolatokat használják. Egy lehetséges hátrány, hogy az útválasztókra sok munka hárul, különösen nagy hálózatok esetén. Tegyük fel, hogy a hálózatban n csoport van, valamennyi átlagosan m taggal. Minden útválasztón, minden csoporthoz m csonkolt feszítőfát kell tárolni, ez összesen mn fa. Az 5.16.(c) ábra például a legbaloldalibb útválasztó 1-es csoporthoz tartozó feszítőfáját mutatja. A legjobboldalibb útválasztó 1-es csoporthoz tartozó feszítőfája (nem látható) nagyon különbözik ettől, mivel a csomagok közvetlenül a csoporttagokhoz kerülnek elküldésre, nem a gráf bal oldali részén keresztül. Ez azt jelenti, hogy az útválasztóknak az 1-es csoporthoz címzett csomagokat más irányba kell küldeniük attól függően, hogy melyik csomópont küld a csoportnak. Amikor sok nagy csoport van számos küldővel, tekintélyes méretű tár kell az összes fa tárolásához.

Egy alternatív kialakítás **magalapú fákat (core-based trees)** használ a csoport feszítőfájának kiszámításához [Ballardie és mások, 1993]. Az összes útválasztó megegyezik egy gyökérben (ezt **magnak** vagy **találkozási pontnak** hívják), és kialakításra kerül a fa oly módon, hogy minden tag csomagot küld a gyökér felé. A fa ezen csomagok által bejárt útvonalak uniója. Az 5.17(a) ábra az 1-es csoport magalapú fáját mutatja be. Ehhez a csoporthoz való küldéshez a küldő csomagot küld a magnak. Amikor a csomag eléri a magot, továbbításra kerül lefelé a fában. Ezt az 5.17(b) ábra mutatja a hálózat legjobboldalibb küldőjére. A teljesítőképesség optimalizálásaként a csoporthoz küldött csomagoknak nem kell eljutniuk a magig többesküldésük előtt. Amint a csomag eléri a fát, továbbítható felfelé a gyökér felé, valamint lefelé az összes többi ágon. Ez vonatkozik az 5.17(b) ábra tetején lévő küldőre.

Osztott fa nem optimális az összes forráshoz. Az 5.17(b) ábrán például a jobb oldalon lévő küldőtől érkező csomag a jobb felső csoporttagot a három ugrásra lévő magon keresztül éri el, nem közvetlenül. A hatékonyság mértéke attól függ, hogy hol található a



5.17. ábra. *(a) Magalapú fa az 1-es csoporthoz. (b) Küldés az 1-es csoporthoz*

mag és a küldő. Általában érdemes a magot a küldők közepére tenni. Ha csak egyetlen küldő van – mint például egy videofolyam esetén, amely egy csoporthoz kerül továbbításra –, akkor optimális megoldásként a küldőt kell magnak választani.

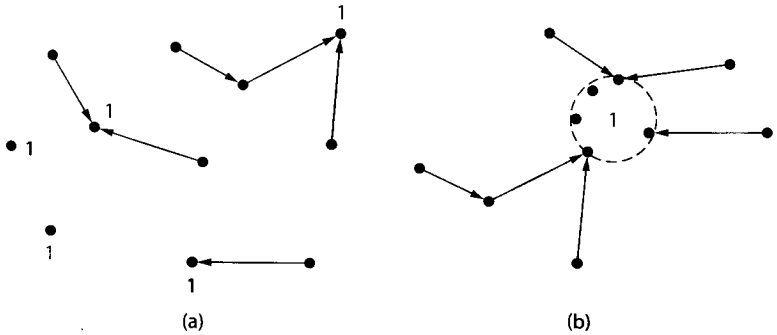
Vegyük figyelembe azt is, hogy az osztott fák jelentős megtakarítást jelentenek a tárolási költség, a küldött üzenetek mennyisége és a számítási igény szempontjából. Minden útválasztónak csoportonként csak egyetlen fát kell tárolnia m fa helyett. Azok az útválasztók, amelyek nem részei a fának, semmit nem tesznek a csoport támogatása érdekében. Emiatt az osztott fa módszer, a magalapú fához hasonlóan, többesküldéshez kerül felhasználásra ritka csoportok esetén az interneten, az olyan népszerű protokollok részeként, mint amilyen például a **PIM (Protocol Independent Multicast – protokollfüggetlen többesküldés)** [Fenner és mások, 2006].

5.2.9. Bárkinek küldéses (anycast) útválasztás

Eddig olyan továbbítási modelleket mutattunk be, amelyekben egyetlen forrás küld adatokat egyetlen címzettnek (ezt nevezik egyesküldésnek), az összes címzettnek (adatszórás) vagy a címzettek egy adott csoportjának (többesküldés). Bizonyos esetekben azonban hasznos lehet egy másik továbbítási modell, a **bárkinek küldés (anycast)** modell alkalmazása. Bárkinek küldés esetén a csomag a csoport legközelebbi tagjának kerül kézbesítésre [Partridge és mások, 1993]. Azokat a sémákat, amelyekkel az ilyen útvonalak megtalálhatók, bárkinek küldéses útválasztásnak hívjuk.

Miért lehet érdemes bárkinek küldést használni? Bizonyos esetekben a csomópontok olyan szolgáltatást nyújtanak, mint például a pontos idő vagy a tartalomelosztás, ahol csak a megfelelő információ biztosítása számít, az nem érdekes, hogy melyik csomópont kapja az információt. Tetszőleges csomópont kaphatja. A bárkinek küldés például használható az interneten a DNS részeként, ahogy az a 7. fejezetben látható.

Szerencsére a bárkinek küldéshez nem kell új útválasztási sémát kitalálni, mivel a normál távolságvektor-alapú és kapcsolatállapot-alapú útválasztás elő tudja állítani a bárkinek küldés útvonalait. Tételezzük fel, hogy az 1-es csoport tagjai felé kívánunk bárkinek küldést alkalmazni. Az összes csoporttag „1"-es címet kap, nem kapnak kü-



5.18. ábra. *(a) Bárkinek küldéses útvonalak az 1-es csoporthoz. (b) Az útválasztó protokoll által látott topológia*

lönböző címeket. A távolságvektor-alapú útválasztás a vektorokat a szokásos módon kiosztja, és a csomópontok kiválasztják a legrövidebb útvonalat az 1-es célhoz. Ennek eredményeképpen a csomópontok az 1-es cél legközelebbi példányához küldik a csomagot. Az útvonalakat az 5.18(a) ábra mutatja. Ez az eljárás azért működik, mert az útválasztó protokoll nem tudja, hogy az 1-es cél több példánnyal rendelkezik. Az 1-es csomópont összes példányát egyetlen csomópontnak tekinti, ahogy azt az 5.18(b)ábrán látható topológia mutatja.

Ez az eljárás kapcsolatállapot-alapú útválasztás esetén is működik, azzal kiegészítve, hogy az útválasztó protokollnak nem kell megtalálni azt a látszólag rövid útvonalat, amely átmegy az 1-es csomóponton. Ez a hipertéren átmenő ugrásokat okozna, mivel az 1-es csomópont példányai valóban olyan csomópontok, amelyek a hálózat különböző részein találhatók. A kapcsolatállapot-alapú protokollok azonban már különbséget tesznek az útválasztók és a hosztok között. Eddig ezt a tényt nem említettük, mivel a leíráshoz nem volt rá szükség.

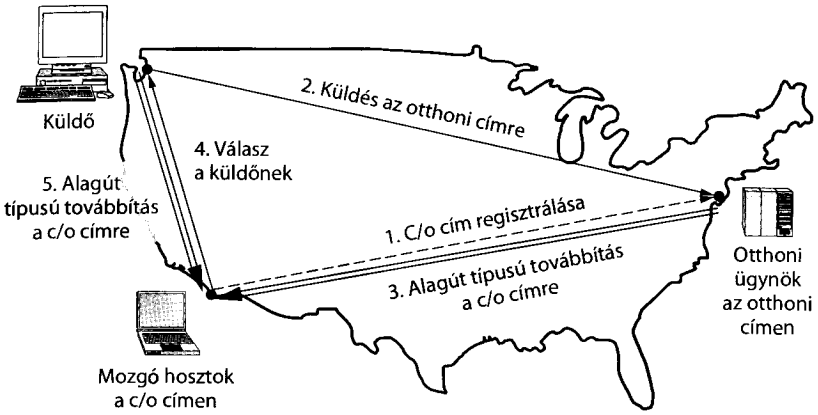
5.2.10. Útválasztás mozgó hosztokhoz

Manapság több millió ember használ számítógépet utazás közben, a mozgó autókban lévő vezeték nélkül eszközöket használó, teljesen mobil helyzetektől kezdve olyan nomád helyzetekig, ahol a hordozható számítógépet különböző helyeken használják. A **mozgó hoszt** (**mobile host**) kifejezést mindkét kategóriára alkalmazzuk, a mozdulatlan (stationary) hosztoktól való megkülönböztetés érdekében. Egyre nagyobb igény mutatkozik arra, hogy a felhasználók, bárhol is vannak a világon, ugyanolyan könnyen tudják elérni a hálózatot, mintha otthon lennének. Ezek a mozgó hosztok újabb bonyodalmat jelentenek: ahhoz, hogy egy csomagot egy mozgó hoszthoz lehessen irányítani, a hálózatnak először meg kell találnia azt a hosztot.

Az általunk elképzelt világmodellben minden hoszt rendelkezik **állandó lakhellyel** (**home location**), amely sosem változik. Minden hosztnak van egy állandó lakcíme is, amelynek segítségével meghatározható a lakhely, hasonlóan, ahogy az 1-212-5551212 telefonszámból kiolvasható az Egyesült Államok (1-es országkód) és Manhattan (212). A mozgó hosztokat tartalmazó rendszerek útválasztási célja, hogy lehetővé tegyük a mozgó felhasználók számára csomagok küldését a lakcímükre, és hogy a csomagok hatékonyan érjék el őket, bárhol is legyenek. Természetesen a trükk az, hogy meg kell őket találni.

Ennek a modellnek a bemutatása következik. Egy eltérő modell lehetne, amelyik újra meg újra kiszámítja az útvonalat, amint a mozgó hoszt helyet változtat, és a topológia változik. Ezután egyszerűen használhatnánk e szakasz korábbi részében leírt útválasztási sémát. Az egyre növekvő számú mozgó hoszt miatt azonban ez a modell azt okozná, hogy a teljes hálózat vég nélkül új útvonalakat számítana ki. Az otthoni cím használata nagymértékben csökkenti ezt a terhet.

Másik alternatíva a mobilitás biztosítása a hálózati réteg felett, ami manapság jellemzően történik laptopoknál. Amikor új internetes helyre viszik, a laptopok új hálózati címet igényelnek. Nincs összefüggés a régi és új hálózati cím között. A hálózat nem tudja, hogy ez a két cím ugyanahhoz a laptophoz tartozik. Ebben a modellben a laptop



5.19. ábra. *Csomagtovábbítás mobil hosztok felé*

használható webböngészésre, de más hosztok nem tudnak felé csomagot küldeni (például bejövő hívás esetén) magasabb rétegbeli helymeghatározási szolgáltatás kialakítása nélkül (például újra be kell jelentkezni a Skype-ba). Továbbá az összeköttetések nem tarthatók fenn, mialatt a hoszt mozgásban van. Ehelyett új összeköttetést kell elindítani. A hálózati rétegbeli mobilitással ezek a problémák megoldhatók.

Az interneten és a mobilhálózatokban a mobil útválasztásnál alkalmazott alapgondolat az, hogy a mozgó hosztok megmondják az otthoni hosztnak, hogy hol vannak. Ezt a hosztot, amely a mozgó hoszt nevében jár el, **otthoni ügynöknek** (**home agent**) hívjuk. Ha ez az ügynök tudja, hogy jelenleg hol található a mozgó hoszt, akkor csomagokat továbbíthat felé, így a csomagok kézbesítésre kerülnek.

Az 5.19. ábra a mobil útválasztást mutatja be működés közben. Az északnyugati Seattle városban lévő feladó csomagot kíván küldeni egy hosztnak, amely általános esetben az USA átellenes oldalán, New Yorkban található. Minket az az eset érdekel, amikor a mozgó hoszt nem otthon van, hanem ideiglenesen San Diegóban.

A San Diegóban lévő mozgó hosztnak helyi hálózati címet kell kérnie ahhoz, hogy használni tudja a hálózatot. Ez a szokásos módon zajlik. A fejezet későbbi részében mutatjuk be, hogy ez hogyan működik az interneten. A helyi címet **c/o címnek** (**care-of address – továbbítási cím**) hívjuk. Ha a mozgó hoszt megkapta ezt a címet, akkor meg tudja mondani ezt a címet az otthoni ügynöknek (1. lépés). Ez az üzenet látható az 5.19. ábrán szaggatott vonallal, ami azt jelzi, hogy ez egy vezérlőüzenet, nem adatüzenet.

Következő lépésként a feladó adatcsomagot küld a mozgó hoszt felé az állandó cím felhasználásával (2. lépés). Ezt a csomagot a hálózat a hoszt lakhelyére küldi, mivel ehhez tartozik az otthoni cím. New Yorkban az otthoni ügynök elfogadja ezt a csomagot, mivel a mozgó hoszt távol van az otthonától. Ezután beágyazza vagy **becsomagolja** a csomagot egy új fejrésszel, majd elküldi ezt a csomagot a továbbítási címre (3. lépés). Ezt a mechanizmust **alagút típusú átvitelnek** (**tunneling**) hívjuk. Ez nagyon fontos az interneten, így ezzel részletesebben is foglalkozunk.

Amikor a becsomagolt csomag megérkezik a c/o továbbítási címre, akkor a mozgó hoszt kibontja azt, és megkapja a feladó csomagját. A mozgó hoszt ezután a válaszcso-

magot közvetlenül a feladónak küldi (4. lépés). A teljes útvonal meghatározását **háromszögező útválasztásnak** (**triangle routing**) hívják, mivel hosszadalmas lehet, ha az új hely távol van az otthoni helytől. A 4. lépés részeként a feladó megismerheti az aktuális c/o továbbítási címet. A további csomagok közvetlenül a mozgó hoszthoz küldhetők a c/o címre történő, alagút típusú továbbítással (5. lépés), az otthoni cím kihagyásával. Ha a kapcsolat valamilyen okból megszakad a mozgó hoszt mozgása közben, akkor az otthoni cím továbbra is használható a mozgó hoszt eléréséhez.

Fontos megjegyezni, hogy a leírásból kihagytuk a biztonságot. Általában amikor egy hoszt vagy útválasztó „Mostantól Stefani leveleit küldje nekem" formátumú üzenetet kap, akkor számos kérdés merülhet fel azzal kapcsolatban, hogy ki a másik fél, és hogy jó ötlet-e neki küldeni. Az üzenetek tartalmaznak biztonsági információt, úgy hogy az érvényességük titkosító protokollokkal ellenőrizhető, amelyet a 8. fejezetben tanulmányozunk.

Számos különböző mobil útválasztás létezik. A fenti séma IPv6-mobilitáson kerül modellezésre. Ezt használják az interneten [Johnson és mások, 2004], valamint az IP-alapú mobilhálózatok – mint például az UMTS – részeként. A küldőt az egyszerűség kedvéért mozdulatlan csomópontnak mutattuk, de a kialakítás lehetővé teszi, hogy mindkét csomópont mozgó hoszt legyen. Ennek egy változata az, amikor a hoszt egy mobilhálózat része, például amikor egy számítógép egy repülőn van. Az alapséma kiterjesztései támogatják a mobilhálózatokat anélkül, hogy a hosztoknak bármit tenniük kellene [Devarapalli és mások, 2005].

Néhány séma **idegen ügynököt** (**foreign agent**) vagy távoli ügynököt használ az otthoni ügynökhöz hasonlóan, de idegen helyen, vagy a VLR-hez (Visitor Location Register – látogató-elhelyezkedési regiszter) hasonló mobilhálózatokban. Az újabb sémákban azonban az idegen ügynökre nincs szükség. A mozgó hosztok saját idegen ügynökükként is működnek. A mozgó hoszt ideiglenes helyének ismerete mindkét esetben kis számú hosztra korlátozódik (például a mozgó ügynök, otthoni ügynök és a küldők), így nagy hálózatokban nem minden útválasztónak kell újból kiszámítania az útvonalakat. A mobil útválasztással kapcsolatos további információt Perkins [1998, 2002], valamint Snoeren és Balakrishnan [2000] munkája tartalmaz.

A hálózattervezők által tipikusan használt világmodellt az 5.18. ábra mutatja. Van egy WAN-unk, amely útválasztókból és hosztokból áll. A WAN-hoz LAN-ok, MAN-ok és olyan vezeték nélküli cellák kapcsolódnak, amelyeket a 2. fejezetben tanulmányoztunk.

5.2.11. Útválasztás ad hoc hálózatokban

Azt már láttuk, hogyan történik az útválasztás, ha a hosztok mozognak, de az útválasztók helyhez kötöttek. Ennél is szélsőségesebb eset az, ha az útválasztók maguk is mozognak. Néhány a lehetőségek közül: válságstáb egy földrengés sújtotta területen, katonai járművek egy harctéren, hajóflotta a tengeren, hordozható számítógéppel rendelkező felhasználók csoportja egy olyan helyen, ahol nincs 802.11.

Az ilyen esetekben minden csomópont egy útválasztóból és egy hosztból áll, általában ugyanazon a számítógépen belül. Az olyan csomópontokból álló hálózatot, amelyben a csomópontok véletlenül éppen egymás közelében tartózkodnak, **ad hoc hálózatnak**

vagy **MANET**-nek (**Mobile Ad hoc NETworks**) hívjuk. Tekintsük át röviden most ezeket! További információért lásd Perkins [2001] munkáját.

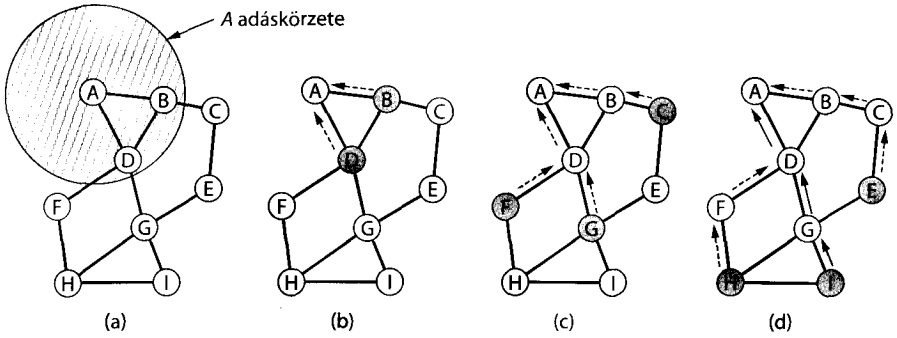
Az ad hoc hálózatokat az különbözteti meg a vezetékes hálózatoktól, hogy ezeknél a rögzített topológiákra, rögzített és ismert szomszédokra, valamint az IP-címek és fizikai elhelyezkedés közötti rögzített adatkapcsolatra vonatkozó jól bevált szabályainkat egyszeriben kidobhatjuk az ablakon. Az útválasztók itt jöhetnek-mehetnek, és egyik pillanatról a másikra új helyeken bukkanhatnak fel. Ha az útválasztónak egy vezetékes hálózatban van érvényes útvonala valamely címzett irányába, az az útvonal a végtelenségig érvényes marad (eltekintve egy esetleges rendszerhibától). Ad hoc hálózatban a topológia állandóan változásban lehet, így az egyes utak kívánatossága, sőt érvényessége is spontán, minden előzetes figyelmeztetés nélkül megváltozhat. Mondani sem kell, hogy e körülmények miatt az ad hoc hálózatok útválasztása teljesen eltér a fix, telepített hálózatokétól.

Számos javaslat született már az ad hoc hálózatok útválasztó algoritmusaira. Bár az ad hoc hálózatok használata kevésbé gyakori, mint a mobilhálózatoké, tisztázatlan, hogy ezek közül a protokollok közül melyik a leghasznosabb. Az egyik legnépszerűbb útválasztó algoritmus az **AODV**-(**Ad hoc On-demand Distance Vector – ad hoc igény szerinti távolságvektor**) algoritmus [Perkins és Royer, 1999]. Ez a távolságvektor-alapú algoritmus távoli rokona, amit hozzáigazítottak a mobilkörnyezet sajátosságaihoz: a csomópontok sávszélessége jellemzően korlátozott és szűkös az akkumulátor-kapacitás. Tekintsük át, hogy ez hogyan térképezi fel és tartja karban az útvonalakat.

Útvonal-felfedezés

Ad hoc igény szerinti távolságvektor (AODV) esetén a címzett felé vezető útvonalak igény szerint kerülnek feltérképezésre, azaz csak akkor, ha valaki csomagot kíván küldeni a címzett felé. Így több munka takarítható meg, mint amennyit az útvonal használata előtti topológia módosítása vonna maga után. Az ad hoc hálózat topológiája bármelyik pillanatban leírható a csatlakoztatott csomópontok gráfjával. Két csomópont akkor van összekötve (azaz él köti össze őket a gráfban), ha rádiós úton közvetlenül kommunikálni tudnak egymással. Alapszintű, de a célnak megfelelő modell, hogy minden csomópont kommunikálni tud az összes többi csomóponttal, amely a hatótávolságon (lefedettségi körön) belül van. A valós hálózatok ennél összetettebbek. Megtalálhatók benne épületek, hegyek és egyéb, a kommunikációt blokkoló akadályok, és előfordulhat, hogy *A* tud kommunikálni *B*-vel, de *B* nem tud *A*-val, mivel a két készülék közül az egyik nagyobb teljesítményű adóval rendelkezhet, mint a másik. Az egyszerűség kedvéért azonban tegyük fel, hogy minden kapcsolat szimmetrikus.

Az algoritmus ismertetéséhez tekintsük meg az 5.20. ábrán látható, újonnan kialakított ad hoc hálózatot, ahol az *A* csomópont egy folyamata csomagot szeretne küldeni az *I* csomópontba. Az AODV-algoritmus minden csomópontban karbantart egy távolságvektor-alapú táblázatot minden csomóponthoz, címzettekre vonatkozó kulccsal, például arról, hogy melyik csomópontnak kell küldeni a csomagokat az adott címzett eléréséhez. Először az *A* végignézi a táblázatát, de nem talál benne *I*-re vonatkozó bejegyzést. Ekkor magának kell felfedeznie az *I*-be vezető útvonalat. Az utakat tehát csak akkor térképezik



5.20. ábra. *(a) A adáskörzete. (b) Miután* B *és* D *megkapták* A *üzenetét. (c) Miután* C, F *és* G *is megkapta* A *üzenetét. (d) Miután* E, H *és* I *is megkapta* A *üzenetét. A sötétített csomópontok az új vevők. A szaggatott nyilak a lehetséges visszairányú útvonalakat, a folytonos nyilak a feltárt útvonalakat jelzik*

fel, amikor szükség van rájuk – ezen tulajdonsága miatt mondjuk az algoritmust „igény szerinti"-nek (on demand).

Annak érdekében, hogy *A* megtalálja *I*-t, egy speciális ROUTE REQUEST (ÚTVONALKÉRELEM) csomagot állít össze és ezt elárasztással (lásd 5.2.3. szakasz) minden hatósugarában lévő állomásnak elkezdi sugározni. A csomag *A*-ból eléri *B*-t és *D*-t, ahogy azt az 5.20.(a) ábra mutatja. Minden csomópont továbbsugározza a kérést, és eléri az *F*-et, *G*-t és *C*-t (5.20.(c) ábra), illetve a *H*-t, *E*-t és *I*-t (5.20.(d) ábra). A forráson beállított sorszámot használják arra, hogy megakadályozza a csomagok kettőződését az elárasztás során. Például a *D* eldobja a *B* felől jövő csomagot az 5.20.(c) ábrán, mivel az már továbbította a kérést.

Végül a kérés eléri az *I* csomópontot, amely ROUTE REPLY (ÚTVONALVÁLASZ) csomagot állít össze. Ezt a csomagot közvetlenül a címzetthez küldik a kérés által bejárt útvonalon visszafelé. Ahhoz, hogy ez működjön, minden köztes csomópontnak meg kell jegyeznie a csomópontot, amelyiktől a kérést kapta. Az 5.20.(b)–(d) ábrán lévő nyilak mutatják a tárolt fordított útvonal-információt. A válasz továbbítása során minden köztes csomópont növeli az ugrásszámlálót. Ez megmondja a csomópontoknak, hogy milyen messze vannak a címzettől. A válaszok megmondják az összes köztes csomópontnak, hogy melyik szomszédot használják a címzett eléréséhez: ez az a csomópont, amelyiktől a választ kapták. A köztes *G* és *D* csomópont a válasz feldolgozása közben beteszi a legjobb útvonalat az útválasztó táblázatba. Amikor a válasz eléri az *A* csomópontot, új *ADGI* útvonal áll elő.

Nagyméretű hálózatban az algoritmus sok csomagot generál, még közeli célok esetén is. A többletterhelés csökkentése érdekében az adatszórás hatóköre korlátozott az IP-csomag *Élettartam* mezője alapján. Ezt a mezőt a küldő inicializálja, ennek az értéke minden ugrásnál csökken. Ha eléri a nullát, a csomagot eldobják, ahelyett hogy továbbadnák. Az útfelfedezési folyamat ezután a következőképp változik. A címzett keresésekor a feladó a ROUTE REQUEST csomag *Élettartamát* 1-re állítja. Ha belátható időn belül nem érkezik válasz, egy újabb kérelem kerül elküldésre, 2-es *Élettartam* értékkel. Az ezt követő kísérletekben pedig 3, 4, 5 stb. kerül az *Élettartam* mezőbe. A feladó így először lokálisan, később pedig egyre nagyobb sugarú körökben kísérelheti meg a keresést.

Útvonal-karbantartás

Mivel a csomópontok mozoghatnak, illetve ki is kapcsolhatják őket, a topológia váratlanul megváltozhat. Az 5.20. ábrán lévő példánkban, ha *G*-t kikapcsolják, *A* nem fogja észrevenni, hogy az *I*-hez eddig használt útvonala (*ADGI*) már nem érvényes. Az algoritmusnak ezzel is meg kell birkóznia. Ennek érdekében minden csomópont periodikusan szétküld egy *Hello* üzenetet. Erre minden szomszédnak válaszolnia kell. Ha valahonnan nem jött üzenet, a feladó tudja, hogy az adott szomszédja elhagyta az adáskörzetét és már nincs vele kapcsolatban. Hasonlóképpen, abból, hogy megpróbál elküldeni egy csomagot egy szomszédnak, és az nem válaszol, megtudhatja, hogy a szomszédja többé nem elérhető.

Ezzel az információval megszabadulhatunk azoktól az utaktól, amelyek már nem működnek. Minden csomópont (ezek közül az egyik legyen *N*) minden lehetséges címzetthez nyilvántartja azokat az aktív szomszédjait, amelyek az elmúlt ΔT másodperc során csomagot továbbítottak neki az adott címzetthez. Ha bármely szomszédja elérhetetlenné válik, *N* megvizsgálja az útválasztó táblázatában, hogy mely címzettekhez vezet út az imént távozott szomszédon keresztül. Minden ilyen útvonal esetében közölni kell az aktív szomszédokkal, hogy az *N*-en át vezető útvonaluk immár érvénytelen és törölni kell az útválasztó táblázatból. Példánkban a *D* törli a *G*-hez és I-hez tartozó bejegyzéseket az útválasztó táblázatból, és értesíti *A*-t, amely törli az *I*-hez tartozó bejegyzést. Általános esetben az aktív szomszédok értesítik a saját aktív szomszédjaikat, és így tovább, míg rekurzív módon az összes, az imént távozott csomóponttól függő útvonal ki nem kerül az összes útválasztó táblázatból.

Ezen a ponton az érvénytelen útvonalak törlésre kerültek a hálózatról, és a leírt felfedezési módszerrel a feladók megkereshetik az új, érvényes útvonalakat. Idézzük fel, hogy a távolságvektor-alapú protokolloknál a topológia változása után fellép a végtelenségig számolás problémája és lassú a konvergencia is, amelyben összekeverednek a régi, érvénytelen útvonalak az új, érvényes útvonalakkal.

A gyors konvergencia biztosítása érdekében az útvonalak tartalmaznak egy sorszámot, amelyet a címzett szabályoz. A címzett sorszám egy logikai órához hasonlatos. A címzett ezt növeli minden egyes friss ROUTE REPLY üzenet küldésekor. A feladók úgy kérnek friss útvonalat, hogy az utolsó használt útvonal sorszámát berakják a címzett ROUTE REQUEST üzenetbe, amely az éppen törölt útvonal sorszáma lesz, vagy kezdeti értékként 0. A kérés addig kerül továbbításra, amíg magasabb sorszámú útvonalat nem talál. A köztes csomópontok eltárolják a magasabb sorszámú útvonalakat, vagy a legkevesebb ugrást az aktuális sorszámhoz.

Az igény szerinti protokollok szellemében a köztes csomópontok csak a használatban lévő útvonalakat tartalmazzák. Az adattovábbítások során megszerzett útvonal-információ rövid késleltetés után lejár. Azáltal, hogy csak a használt útvonalak kerülnek felfedezésre és tárolásra, sávszélesség takarítható meg és növelhető az akkumulátor-élettartam a normál távolságvektor-alapú protokollhoz képest, amely rendszeres időközönként továbbítja a frissítéseket.

Eddig csak egyetlen útvonalat tételeztünk fel az *A* és *I* között. További erőforrások megtakarítása érdekében az útvonal felfedezése és karbantartása megoszlik, ha az útvonalak átfedik egymást. Ha például a *B* is csomagokat kíván küldeni az *I*-nek, akkor út-

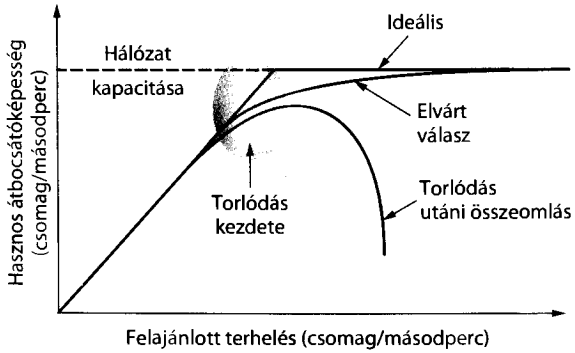
vonal-felfedezést hajt végre. Ebben az esetben azonban a kérés először eléri a *D*-t, amely már rendelkezik útvonallal az *I* felé. A *D* csomópont ezután elő tud állítani egy választ, hogy megmondja *B*-nek az útvonalat anélkül, hogy további munkára lenne szükség.

Számos más ad hoc útválasztási séma létezik. Egy másik, jól ismert igény szerinti séma a DSR (Dynamic Source Routing – dinamikus forrás útválasztás) [Johnson és mások, 2001]. Más típusú, földrajzi alapú stratégia a GPRS (Greedy Perimeter Stateless Routing – önző határfelületi állapot nélküli útválasztás) [Karp és Kung, 2000]. Ha minden csomópont tudja a földrajzi helyét, akkor a címzett felé történő továbbítás folytatható útvonal-kiszámítás nélkül azáltal, hogy a megfelelő irányban kerül továbbításra és vissza, a zsákutcák elkerülése érdekében. Az, hogy melyik protokoll nyer, az ad hoc hálózatok típusától függ, ami a gyakorlatban nagyon hasznos.

5.3. Torlódáskezelési algoritmusok

Amikor túl sok csomag van jelen a hálózatban (vagy egy részében), a teljesítőképesség visszaesik, a csomagküldés pedig késleltetést szenved. Ezt a helyzetet **torlódásnak** (**congestion**) nevezzük. A hálózati és a szállítási réteg megosztja a torlódáskezelés felelősségét. Amikor torlódás lép fel a hálózatban, a hálózati réteg ezt közvetlenül érzékeli, és neki kell meghatározni végül, hogy mi történjen a többletcsomagokkal. A leghatékonyabb módszer azonban a torlódás szabályozására, ha csökkentjük a szállítási réteg által a hálózatra rótt terhelést. Ez a hálózati és a szállítási réteg együttműködését igényli. Ebben a fejezetben a torlódás hálózati aspektusait nézzük meg. A 6. fejezetben a torlódás szállítási aspektusainak bemutatásával tesszük teljessé a témakört.

Az 5.21. ábra festi le a torlódás tüneteit. Amikor a hosztok által a hálózatba beadott csomagok száma a hálózat átviteli kapacitásán belül marad, akkor minden csomag kézbesítésre kerül (kivéve egy párat, amely az átvitel során hibákat szenvedett), és a kézbesített csomagok száma arányos az elküldött csomagok számával. Ahogy azonban az átvitelre szánt (felajánlott) terhelés eléri a hálózat átviteli kapacitását, a forgalmi löket alkalmanként megtölti az útválasztók puffereit és néhány csomag elveszik. Ezek az elve-



5.21. ábra. *Túl nagy forgalom esetén a teljesítőképesség meredeken visszaesik*

szett csomagok felemésztik a kapacitás egy részét, így a kézbesített csomagok száma az ideális szint alá csökken. A hálózaton torlódás alakul ki.

Ha a hálózat nincs jól megtervezve, akkor a torlódás **összeomlást** okozhat, amelynek hatására a teljesítőképesség gyorsan csökken, amint a terhelés túlnő a kapacitáson. Ez azért történhet meg, mert a csomagok késleltetése a hálózatban olyan nagy lehet, hogy azok már nem hasznosak, amikor elhagyják a hálózatot. A korai interneten például az az idő, ameddig egy csomag várakozott a sorban előtte várakozó többi csomagra, mielőtt átküldésre került volna egy lassú 56 kb/s-os adatkapcsolaton, elérhette a maximális hálózati tartózkodási időt, ezért azt el kellett dobni. Másféle hibamód jelentkezik akkor, amikor a feladók újraküldenek nagy késleltetésű csomagokat, mert azt hiszik, hogy azok elvesztek. Ebben az esetben ugyanannak a csomagnak több példányát továbbítja a hálózat, ez ismét a kapacitás pazarlásával jár. Ezen tényezők rögzítéséhez az 5.21. ábra *y* tengelye a **hasznos átbocsátóképességet** (**goodput**) ábrázolja. Ez az az arány, amellyel a hálózat a hasznos csomagokat kézbesíti.

Olyan hálózatot szeretnénk kialakítani, amely ahol csak lehet, elkerüli a torlódást, és nem történik a torlódás után összeomlás, amennyiben mégis torlódás alakulna ki. Sajnos a torlódás nem kerülhető el teljesen. Ha hirtelen nagy mennyiségű csomag érkezik három vagy négy bejövő vonalon, és ezek közül mindegyiknek ugyanarra a kimenő vonalra van szüksége, akkor felépül egy várakozási sor. Ha a memória kevés ahhoz, hogy mindet befogadja, akkor csomagok fognak elveszni. Több memória hozzáadása egy adott pontig segíthet, de Nagle [1987] kimutatta, hogy ha az útválasztóknak végtelen kapacitású memóriája van, a torlódás nem javul, hanem rosszabbá válik, mivel mire a csomagok a sor elejére kerülnek, az időzítésük (akár többször is) lejár, és másodpéldányok kerülnek továbbításra. Ez ront a helyzeten, nem javít – torlódás utáni összeomláshoz vezet.

A kis sávszélességű adatkapcsolatok és útválasztók, amelyek a vonali kapacitásnál lassabban dolgozzák fel a csomagokat, szintén okozhatnak torlódást. Ebben az esetben a helyzet javítható azáltal, hogy a forgalom egy része átirányításra kerül a szűk keresztmetszetről a hálózat többi részére. Végül azonban a hálózat összes régiója torlódik. Ebben a helyzetben csak a terhelés csökkentése vagy gyorsabb hálózat kiépítése jelenthet megoldást.

Érdemes rámutatni a különbségre a torlódáskezelés és a forgalomszabályozás közt, mivel a kapcsolatuk nem magától értetődő. A torlódáskezelés azzal foglalkozik, hogy a hálózat képes legyen elszállítani a kért forgalmat. Ez egy globális kérdés, amely magába foglalja minden hoszt, minden útválasztó viselkedését. A forgalomszabályozás ezzel szemben adott küldő és adott fogadó közötti forgalomra vonatkozik. Feladata megakadályozni, hogy egy gyors adó folyamatosan gyorsabban adjon, mint ahogy a vevő azt fogadni tudja.

Hogy szemléletesebbé tegyük a két fogalom közti különbséget, vegyünk egy 1000 gigabit/s kapacitású üvegszálas optikai hálózatot, amelyen egy szuperszámítógép próbál egy fájlt átvinni egy személyi számítógépre 1 Gb/s-mal. Bár nincsen torlódás (maga a hálózat nincs bajban), forgalomszabályozásra van szükség, hogy a szuperszámítógépet gyakori megállásra kényszerítsük, lélegzethez juttatva ezzel a személyi számítógépet.

Másik végletként vegyünk egy tárol-és-továbbít típusú hálózatot 1 Mb/s-os vonalakkal és 1000 nagy teljesítőképességű számítógéppel, amelyek fele fájlokat próbál átvinni

100 kb/s-mal a másik feléhez. Itt nem az a probléma, hogy a gyors adók elnyomják a lassú vevőket, hanem egyszerűen az, hogy az összes kívánt forgalom meghaladja azt, amit a hálózat kezelni képes.

A torlódáskezelést és a forgalomszabályozást gyakran összekeverik. Ennek oka, hogy mindkét probléma kezelésének legjobb módja az, hogy a hoszt működését le kell lassítani. Vagyis egy hoszt kaphat „lassíts" üzenetet azért is, mert a vevő nem tudja lekezelni a terhelést, vagy azért is, mert a hálózat nem bír vele. Erre a gondolatra később még visszatérünk a 6. fejezetben.

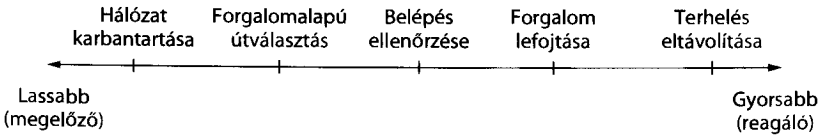
A torlódáskezelés tanulmányozását azoknak a megoldásoknak a tanulmányozásával kezdjük, amelyeket különféle időskálákon használnak. Ezután a torlódás elkerülésével kapcsolatos legfontosabb megoldásokat tekintjük át, majd azok a megoldások következnek, amelyek a torlódás bekövetkezésekor azonnal teljesülnek.

5.3.1. A torlódáskezelés alapelvei

A torlódás kialakulása azt jelenti, hogy a terhelés (ideiglenesen) nagyobb, mint amit az erőforrások (a hálózat egy részében) kezelni tudnak. Két megoldási lehetőség merül fel: az erőforrások növelése vagy a terhelés csökkentése. Ahogy ezt az 5.22. ábra mutatja, ezek a megoldások különböző időskálán alkalmazhatók: a torlódás kialakulása előtt annak megakadályozása érdekében, vagy reagálhatnak a már kialakult torlódásra.

A legalapvetőbb mód a torlódás elkerülésére, ha olyan hálózatot építünk, amely megfelel a rajta átmenő forgalomnak. Ha van kis sávszélességű adatkapcsolat az útvonalon, amely mentén a legtöbb forgalom halad, akkor a torlódás nagy valószínűséggel bekövetkezik. Néha további erőforrások dinamikusan használhatók, amikor súlyos torlódás következik be, például tartalék útválasztókat is üzembe lehet helyezni, melyek egyébként csak vésztartalékul szolgálnak (hogy a rendszert hibatűrővé tegyék), vagy sávszélesség vásárolható a szabad piacon. Nagyon gyakran először a nagy kihasználtságú adatkapcsolatok és útválasztók kerülnek frissítésre. Ezt **karbantartásnak** hívják és havi ütemezés szerint történik, amelyet hosszú távú forgalmi trendek vezérelnek.

A meglévő hálózati kapacitás legjobb kihasználása érdekében az útvonalak a forgalmi minták alapján személyre szabhatók. Ezek a forgalmi minták a nap folyamán változnak, ahogy a különböző időzónában lévő felhasználók felkelnek, illetve elmennek aludni. Például az útvonalak módosíthatók úgy, hogy a nagy kihasználtságú útvonalakról elvigyék a forgalmat a legrövidebb útvonal súlyának módosításával. Néhány helyi rádióállomás helikoptert alkalmaz a városi forgalom figyelésére, abban a reményben, hogy a rádióhallgatók elkerülik az autóikkal a torlódások helyét. Ezt **forgalomalapú útválasztásnak** (**traffic-aware routing**) hívják. A forgalom több útvonal közötti felosztása is hasznos lehet.



5.22. ábra. *A torlódáskezelés megoldásai időskálán bemutatva*

Néha azonban a kapacitás egyáltalán nem növelhető. Ekkor a torlódás leküzdésére az egyetlen mód az, hogy csökkentjük a terhelést. Virtuáliskáramkör-alapú hálózat esetén az új összeköttetések felépítése visszautasítható, ha azok a hálózat torlódását okoznák. Ezt **belépés-ellenőrzésnek** (**admission control**) hívjuk.

Egy finomabb eljárás szerint, ha torlódás várható, akkor a hálózat visszajelzést küldhet azoknak a forrásoknak, amelyek forgalma felelős a problémáért. A hálózat megkérheti ezeket a forrásokat a forgalmuk csökkentésére, vagy saját maga lassíthatja le a forgalmat.

Ennek a megoldásnak két nehézsége van: hogyan azonosítjuk a torlódás kezdetét, illetve hogyan tájékoztassuk a forrást arról, hogy lassítania kell a forgalmat. Az első probléma megoldása érdekében az útválasztók meg tudják figyelni az átlagos terhelést, a sorbanállási késleltetést, valamint a csomagvesztést. Mindkét esetben a növekvő szám torlódásnövekedést jelent.

A második probléma megoldása érdekében az útválasztóknak a forrásokkal egy visszacsatoló hurokban kell lenniük. Ahhoz, hogy a séma megfelelően működjön, az időléptéket gondosan be kell állítani. Ha az útválasztó ÁLLJ-t kiált, valahányszor két csomag érkezik egymás után, és INDULJ-t, amikor 20 μs-ig tétlen, a rendszer vadul ingadozni fog és soha nem konvergál. Másfelől, ha a biztonság kedvéért 30 percet vár, mielőtt bármit is mondana, a torlódáskezelő algoritmus túl lassan fog reagálni ahhoz, hogy bármilyen valódi haszna lenne. A megfelelő időzítés szerinti visszacsatolás biztosítása nem egyszerű. Gondot jelent az is, hogy az útválasztók további üzeneteket küldenek, amikor a hálózat már torlódott.

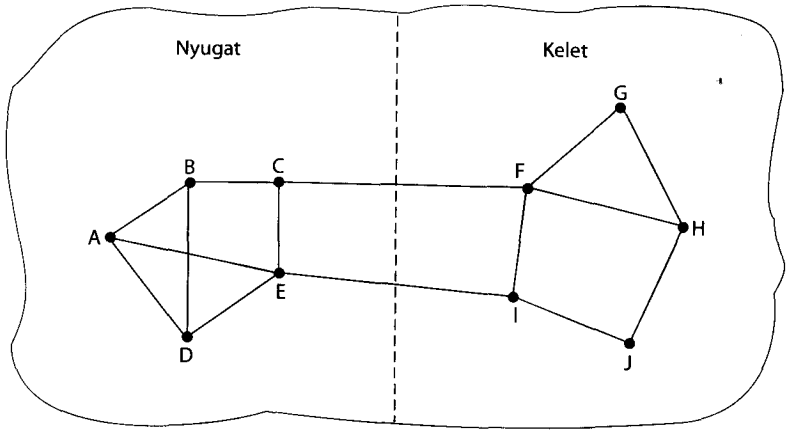
Végül, ha már minden csődöt mondott, a hálózatot kényszeríteni kell a nem kézbesíthető csomagok eldobására. Ennek általános neve **terheléslefojtás** (**load shedding**). Érdemes olyan csomagokat eldobni, amelyekkel megakadályozható a torlódás utáni összeomlás.

5.3.2. Forgalomalapú útválasztás

Először a forgalomalapú útválasztást vizsgáljuk meg. Az 5.2. alfejezetben ismertetett útválasztási séma rögzített adatkapcsolati súlyokat használt. Ezek a sémák igazodtak a topológiai változásokhoz, de a terhelésváltozáshoz nem. A cél az, hogy a rendszer a terhelést is figyelembe vegye az útvonalak kiszámításakor, hogy el lehessen terelni a forgalmat azokról a helyekről (hotspot), amelyeken először érzékelhető a torlódás.

Ez úgy érhető el a legközvetlenebbül, hogy az adatkapcsolat súlyát úgy állítjuk be, hogy az függvénye legyen a (rögzített) adatkapcsolati sávszélességnek és a terjedési késleltetésnek, plusz a (változó) mért terhelésnek vagy az átlagos sorbanállási késleltetésnek. Azok a legkisebb súlyú útvonalak lesznek ekkor a preferált útvonalak, amelyek legkevésbé terheltek, az összes többi útvonal egyforma.

A korai internet ezen modell alapján használta a forgalomalapú útválasztást [Khanna és Zinky, 1989]. Ennek azonban megvan a veszélye. Vegyük az 5.23. ábra hálózatát, amely két részre oszlik, keletire és nyugatira, amelyeket két adatkapcsolat köt össze, *CF* és *EI*. Tegyük fel, hogy a kelet–nyugat közti forgalom legnagyobb része a *CF* vonalat használja, és ennek következtében az adatkapcsolat erősen terhelt, és nagy a késleltetése. Ha bevesszük a sorbanállási késleltetést a legrövidebb út súlyának kiszámításába, akkor *EI* vonzóbbá válik. Miután feltelepítettük az új útválasztó táblázatokat, a kelet–nyugat



5.23. ábra. *Egy hálózat, amelyben a keleti és nyugati részt két adatkapcsolat köti össze*

forgalom legnagyobb része *EI*-n keresztül fog haladni, ezt az adatkapcsolatot terhelve. Következésképpen a következő frissítéskor *CF* fog a legrövidebb útnak tűnni. Ennek eredményeként az útválasztó táblázatok vadul ingadozhatnak, ami szeszélyes útválasztáshoz és számos más lehetséges problémához vezethet.

Ha a terhelést figyelmen kívül hagyjuk, és csak a sávszélességet és a terjedési késleltetést nézzük, ez a probléma nem lép fel. Ha figyelembe vesszük a terhelést, de a súlyokat szűk tartományban módosítjuk, az csak lelassítja az útválasztás ingadozását. Két módszer járulhat hozzá a sikeres megoldáshoz. Az első módszer a többszörös útvonalalapú útválasztás, amelyben több útvonal lehetséges a forrástól a cél felé. A példánkban ez azt jelenti, hogy a forgalom mindkét keleti–nyugati adatkapcsolatra szétoszthatő. A második módszer az, hogy az útválasztási séma a forgalmat elég lassan továbbítsa az útvonalakon ahhoz, hogy az stabil állapotba tudjon jutni, mint ahogy Gallagher [1977] sémájában is.

Ezen nehézségeket figyelembe véve az interneten az útválasztó protokollok általában nem a terheléstől függően állítják be az útvonalakat. A beállítás általában az útválasztó protokollon kívül történik, a bemenet lassú módosításával. Ezt **forgalomtervezésnek** (**traffic engineering**) hívják.

5.3.3. Belépés-ellenőrzés

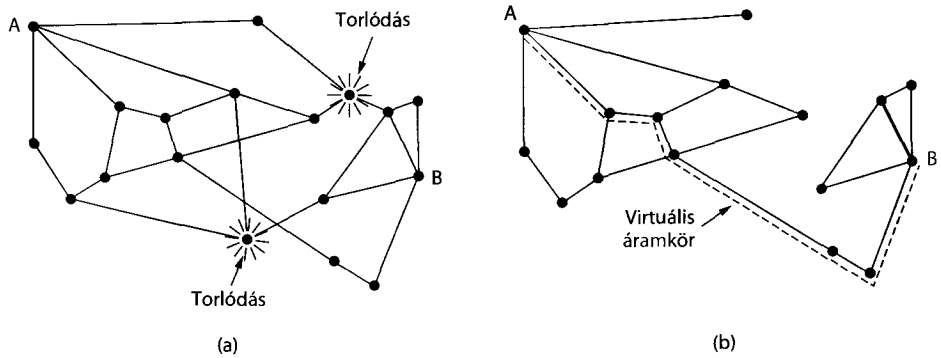
A virtuálisáramkör-alapú hálózatokban gyakran használt technika a torlódás adott mederben tartásához a **belépés-ellenőrzés** (**admission control**). Az ötlet egyszerű: csak akkor állítsunk be új virtuális áramkört, ha a hálózat el tudja szállítani a megnövekedő forgalmat anélkül, hogy torlódás alakulna ki. Így a virtuális áramkör beállítására tett kísérlet sikertelen lehet. A telefonrendszerben ehhez hasonló, amikor egy kapcsolót túlterhelnek, szintén belépés-ellenőrzést hajt végre azáltal, hogy nem ad tárcsahangot.

Ennél a megoldásnál azt trükkös kitalálni, hogy egy új virtuális áramkör mikor vezet torlódáshoz. A feladat telefonhálózatban egyszerű a hívások rögzített sávszélessége miatt (64 kb/s-os tömörítetlen hang esetén). A számítógép-hálózatokban azonban a virtuális áramkörök minden formában és méretben előfordulnak. Ezért az áramkör forgalmát valahogy jellemezni kell, ha belépés-ellenőrzést kívánunk alkalmazni.

A forgalom általában a sebességével és a formájával írható le. A probléma nehézségét az jelenti, hogyan írható le a forgalom egyszerűen, mégis értelmesen, mivel a forgalom jellemzően löketes – az átlagos sebesség csak a dolgok egy része. Például a webböngészés közbeni változó forgalmat bonyolultabb kezelni, mint a sokáig azonos sávszélességet használó mozifolyamot, mivel a webes forgalom löketei nagyobb eséllyel okoznak torlódást a hálózat útválasztóin. Ennek a hatásnak a leírására általánosan használt a **lyukas vödör** (**leaky bucket**) vagy a **vezérjeles vödör** (**token bucket**) algoritmus. A lyukas vödör két paraméterrel rendelkezik, amely meghatározza a forgalom átlagos sebességét és a pillanatnyi löketet. Mivel a lyukas vödör módszerét széles körben használják a szolgáltatásminőséghez, ezt részletesen tárgyaljuk az 5.4. szakaszban.

A hálózat a forgalomleírás birtokában eldöntheti, hogy engedélyezi-e az új virtuális áramkört. Az egyik lehetőség, hogy a hálózat elegendő kapacitást tart fenn az összes virtuális áramkör útvonalai mentén, hogy ne alakuljon ki torlódás. Ebben az esetben a forgalomleírás egy szolgáltatási szerződés arra vonatkozóan, hogy a hálózat mit garantál a felhasználók számára. Megakadályoztuk a torlódást, de a szolgáltatásminőség kapcsolódó témakörbe túl korán mentünk bele. Ehhez vissza fogunk térni a következő szakaszban.

A hálózat még garanciák vállalása nélkül is használhat forgalomleírásokat a belépés-ellenőrzéshez. A feladat ezután annak megbecslése, hogy hány áramkör fér bele a hálózat továbbítási kapacitásába torlódás kialakulása nélkül. Tételezzük fel, hogy a virtuális áramkörök, melyek csúcsforgalma 10 Mb/s, ugyanazon a 100 Mb/s-os fizikai vonalon mennek át. Hány áramkör engedhető be? Nyilvánvalóan 10 áramkör engedélyezhető a torlódás megkockáztatása nélkül, de ez a normál esetben elég pazarló, mivel ritkán fordul elő, hogy mind a 10 egyidejűleg teljes sebességgel adjon. A valós hálózatokban a múltbeli viselkedés méréseiből kapott átviteli statisztikák felhasználhatók az engedélyezhető áramkörök számának becslésére, hogy jobb teljesítőképességet érjünk el elfogadható kockázat mellett.



5.24. ábra. *(a) Torlódott hálózat. (b) A hálózat nem torlódott része. Az* A *és* B *közötti virtuális áramkör is látható*

A belépés-ellenőrzés kombinálható forgalomalapú útválasztással a forgalom problémás területei körüli útvonalak figyelembe vételével, az összeköttetés-felépítés részeként. Például tekintsük meg az 5.24.(a) ábrán látható hálózatot, amelyben két útválasztón torlódás van, a jelzett módon.

Tegyük fel, hogy egy, az *A* útválasztóhoz kapcsolódó hoszt egy, a *B* útválasztóhoz kapcsolódó hoszttal akar összeköttetést létrehozni. Rendesen ez az összeköttetés áthaladna az egyik torlódott útválasztón. Hogy ezt a helyzetet elkerüljük, újrarajzolhatjuk a hálózatot, ahogy az az 5.24.(b) ábrán látszik, kihagyva a torlódott útválasztókat és a hozzájuk kapcsolódó vonalakat. A szaggatott vonal egy lehetséges útvonalat mutat egy, a torlódott útválasztókat elkerülő virtuális áramkör számára. Shaikh és mások [1999] munkája egy ilyen terhelésre érzékeny útválasztás kialakítására ad javaslatot.

5.3.4. Forgalomlefojtás

Az interneten és számos más számítógép-hálózaton a küldők úgy állítják be az átviteliüket, hogy akkora forgalom menjen át, amekkorát a hálózat könnyedén továbbítani tud. Ebben a beállításban a hálózat célja, hogy közvetlenül a torlódás kialakulása előtt működjön. Amikor torlódás közeleg, értesíteni kell a küldőket, hogy fogják vissza az átviteleiket és lassítsanak le. Ez a visszacsatolás nem kivételes, hanem szokványos helyzet. A **torlódáselkerülés** (**congestion avoidance**) kifejezést olyan munkaponttal való szembeállításként használják, amelyben a hálózat (túlságosan) torlódottá válik.

Tekintsünk át néhány olyan forgalomlefojtási módszert, amely mind virtuális áramkör, mind datagramalapú hálózatokban használható. Minden módszernek két problémát kell megoldani. Először is az útválasztónak meg kell tudnia határozni, hogy torlódás közeledik, ideális esetben annak bekövetkezése előtt. Ehhez minden útválasztó folyamatosan figyeli a használt erőforrásokat. Három lehetőség van: a kimeneti vonalak kihasználtsága, a sorba állított csomagok pufferelése az útválasztón belül, valamint a nem megfelelő pufferelés miatt elveszett csomagok száma. Ezek közül a lehetőségek közül a második a leghasznosabb. A kihasználtság átlaga nem veszi figyelembe közvetlenül a forgalom löketes jellegét – 50%-os kihasználtság egyenletes forgalom esetén kicsi lehet, nagyon változó forgalom esetén viszont túl nagy. A csomagvesztések száma túl későn jön, a torlódás ekkorra már kialakult.

Az útválasztókon belüli sorbanállási késleltetés közvetlenül a csomagok által tapasztalt torlódást mutatja. Ennek a működési idő nagy részében kicsinek kell lennie, de meg fog ugrani, amikor a forgalom löketes, amely megnöveli a sorhosszt. Ahhoz, hogy a sorbanállási késleltetést (d) jól tudjuk becsülni, a pillanatnyi sorhosszból (s) periodikusan mintát vehetünk és a d ennek megfelelően frissíthető:

$$d_{\text{új}} = \alpha d_{\text{régi}} + (1 - \alpha)s$$

ahol az α konstans azt határozza meg, milyen gyorsan felejti el az útválasztó a közelmúlt történéseit. Ezt EWMA-nak (**Exponentially Weighted Moving Avarage – exponenciálisan súlyozott mozgóátlag**) hívjuk. Ez kisimítja az ingadozásokat és megfelel egy aluláteresztő szűrőnek. Ha d a küszöbérték fölé megy, az útválasztó észreveszi a torlódás kezdetét.

A második probléma, hogy az útválasztóknak rendszeres időközönként visszajelzést kell küldeni a torlódást okozó küldők felé. A hálózat észleli a torlódást, de a torlódás csökkentése érdekében a hálózatot használó küldőknek is tenniük kell valamit. Visszajelzés küldéséhez az útválasztónak azonosítania kell a megfelelő küldőket. Ezután óvatosan figyelmeztetniük kell a küldőket anélkül, hogy túl sok további csomagot küldenének az amúgy is torlódott hálózatba. A különböző sémák eltérő visszajelzési módszert használnak, ahogy azt a következőkben látni fogjuk.

Lefojtócsomagok

A forrást közvetlenül is értesíteni lehet a torlódásról. Ebben a megoldásban az útválasztó kiválaszt egy torlódott csomagot és **lefojtócsomagot** (**choke packet**) küld vissza a forráshosztnak, megadva benne a csomagban talált célt. Az eredeti csomagot megjelölik (egy bitet beállítanak a fejrészben), így nem fog több lefojtócsomagot generálni a további útja során, és a megszokott módon továbbítódik. A torlódás során az egyre növekvő terhelés elkerülése érdekében a hálózaton az útválasztó csak kis sebességgel küldheti a lefojtócsomagokat.

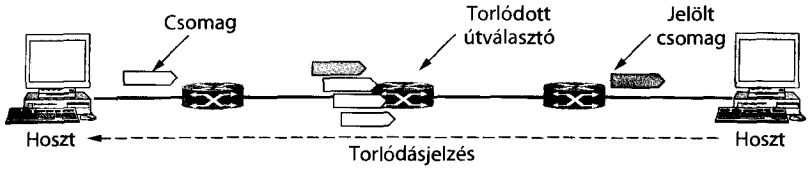
Amikor a forráshoszt megkapja a lefojtócsomagot, csökkentenie kell az adott célnak küldött forgalmat, például 50 százalékkal. Ha a datagramalapú hálózat torlódás esetén véletlenszerűen csipegeti fel a csomagokat, akkor a lefojtócsomagokat nagy valószínűséggel a gyors forrásoknak küldik, mivel nekik lesz a legtöbb csomagjuk a sorban. A protokollban implicit módon jelenlevő visszacsatolás tehát segíthet megelőzni a torlódást anélkül, hogy lefojtana bármilyen küldőt, hacsak az nem okoz bajt. Ugyanilyen okok miatt nagyon valószínű, hogy több lefojtócsomag kerül elküldésre egy adott hoszthoz. A hosztnak figyelmen kívül kell hagynia ezeket a további lefojtócsomagokat addig a rögzített időtartamig, amíg a forgalomcsökkentés hatása nem jelentkezik. Ez után az időtartam után további lefojtócsomagok érkezése azt jelzi, hogy a hálózaton még mindig torlódás van.

A korai interneten használt lefojtócsomagokra példa a SOURCEQUENCH (forráslefojtás) üzenet [Postel, 1981]. Ez azonban soha nem lett sikeres, részben azért, mert az előállításának körülményei és a hatása nincsenek egyértelműen meghatározva. A modern internet alternatív értesítést használ, amelyet később írunk le.

Explicit torlódásjelzés

Ahelyett, hogy az útválasztó további torlódásra figyelmeztető csomagokat állítana elő, bármely általa továbbított csomagot címkével láthatja el (egy bit beállításával a csomag fejrészében) annak jelzése érdekében, hogy torlódást észlelt. Amikor a hálózat kézbesíti a csomagot, a cél észlelheti, hogy torlódást lépett fel, és tájékoztathatja erről a feladót, amikor egy válaszcsomagot küld. A feladó ezután a korábbihoz hasonlóan lefojtja a forgalmát.

Ezt a kialakítást ECN-nek (**Explicit Congestion Notification – explicit torlódásjelzés**) hívják és az interneten használják [Ramakrishnan és mások, 2001]. Ez a korai torlódásjelzési protokoll elsősorban Ramakrishnan és Jain [1988] DECNET architek-



5.25. ábra. *Explicit torlódásértesítés*

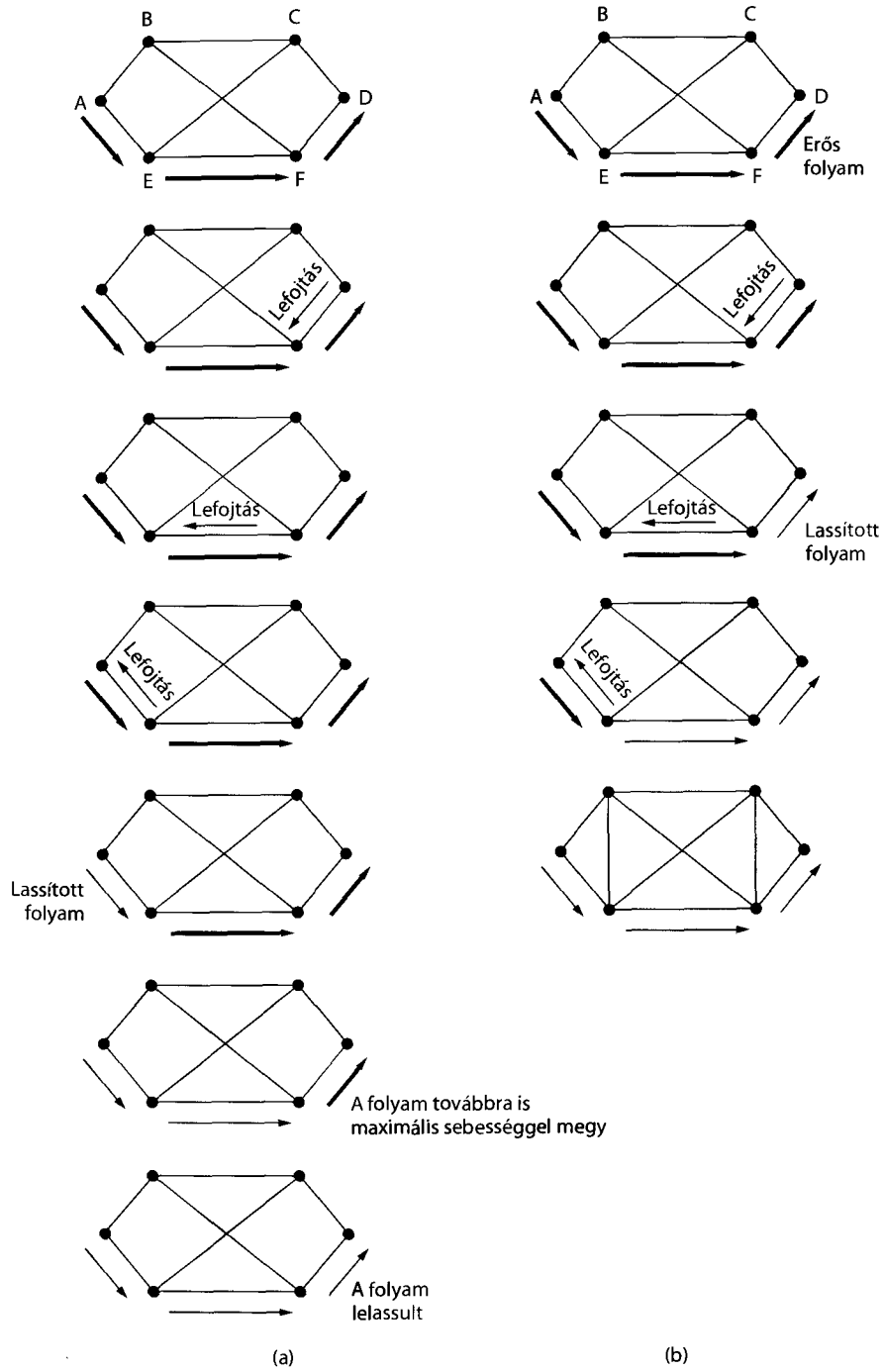
túrában használt bináris visszacsatolási sémájának továbbfinomításával alakult ki. Az IP-csomag fejrészében lévő két bit rögzíti, hogy a csomag észlelt-e torlódást. A csomagokban küldéskor nincs jelzés, ahogy azt az 5.25. ábra mutatja. Ha bármelyik útválasztón, amelyen a csomag keresztülhalad, torlódás alakul ki, az útválasztó megjelöli a csomagot, hogy torlódást észlelt a továbbítás során. A címzett ezután visszaküldi a jelölést a forrásnak explicit torlódásjelzésként a következő válaszcsomagban. Ezt szaggatott vonal mutatja az ábrán annak jelzése érdekében, hogy mindez az IP-szint fölött történik (például a TCP-ben). A forrásnak le kell fojtania a forgalmat, hasonlóan, mint a lefojtócsomagok esetén.

Lépésről lépésre történő visszaszorítás

Nagy sebesség és nagy távolság esetén számos új csomag kerülhet kiküldésre a torlódás jelzése után, a torlódásjelzés hatályba lépésének késleltetése miatt. Vegyünk például egy San Franciscó-i hosztot (az 5.26. ábrán az *A* útválasztó), amely 155 Mb/s-os OC-3 sebességgel küld adatot egy New York-i hosztnak (az 5.26. ábrán a *D* útválasztó). Ha a New York-i hoszt kezd kifogyni a pufferekből, kb. 40 ms-ig tart, hogy egy lefojtócsomag visszajusson San Franciscóba, hogy utasítsa a hosztot a lassításra. Egy ECN-jelzés még tovább tart, mivel az a címzetten keresztül kerül kézbesítésre. A lefojtócsomag terjedését az 5.26.(a) ábra második, harmadik és negyedik lépésében követhetjük nyomon. Ez alatt a 40 ms alatt további 6,2 megabit kerül továbbításra. Még ha a San Franciscó-i hoszt azonnal le is áll, a csőben levő 6,2 megabit továbbra is be fog folyni és foglalkozni kell vele. Csak az 5.26. ábra hetedik rajzán fog a New York-i útválasztó egy lassabb folyamot észlelni.

Alternatív megoldást szemléltet az 5.26.(b) ábra sorozata, amelyben a lefojtócsomag minden csomópontra hat, amelyen keresztülhalad. Itt, amint a lefojtócsomag eléri *F*-et, *F*-nek csökkentenie kell a *D* felé tartó adatfolyamot. Ha így teszünk, akkor *F*-nek több puffert kell szentelnie az összeköttetésnek, mivel a forrás továbbra is teljes gőzzel ad, de *D*-nek azonnali enyhülést hoz, mint egy fejfájás-csillapító egy tv-reklámban. A következő lépésben a lefojtócsomag eléri és utasítja *E*-t, hogy csökkentse az *F* felé tartó adatfolyamot. Ez jobban igénybe veszi *E* puffereit, de *F*-nek azonnali enyhülést ad. Végül a lefojtócsomag eléri *A*-t, és az adatfolyam valóban lelassul.

Ennek a **lépésről lépésre visszaszorítás** (**hop-by-hop backpresure**) sémának a hatása gyors megkönnyebbülést biztosít a torlódás helyénél, azon az áron, hogy a folyam felsőbb részén több puffert használ. Így a torlódást csomagvesztés nélkül csírájában elfojthatjuk. Ezt az ötletet Mishra és mások [1996] részletesebben tárgyalják.



5.26. ábra. *(a) Lefojtócsomag, amely csak a forrásra hat. (b) Lefojtócsomag, amely az összes ugrásra hatással van, amelyen keresztülmegy*

5.3.5. Terhelés eltávolítása

Amikor az előző módszerek közül egyik sem oldja fel a torlódást, az útválasztók bevethetik a nehéztüzérséget: a terhelés eltávolítását. A **terhelés eltávolítása (load shedding)** annak a „szalonképes" megfogalmazása, hogy ha az útválasztókat olyan csomagok árasztják el, amelyekkel nem tudnak megbirkózni, akkor egyszerűen kidobják azokat. A kifejezés az elektromos áram előállításának világából származik, ahol azt a gyakorlatot jelenti, hogy a közművek bizonyos területeken szándékosan áramszünetet idéznek elő azért, hogy az egész hálózat megmeneküljön az összeomlástól olyan napokon, amikor az áram iránti igény nagyban meghaladja a termelt mennyiséget.

Kulcsfontosságú kérdés, hogy a csomagoktól fuldokló útválasztó mely csomagokat dobja el. A preferált választási mód a hálózat által használt alkalmazástól függ. Fájlátvitelnél egy régebbi csomag többet ér, mint egy új, mert a 6. csomag eldobása és a 7-től 10-ig terjedő csomagok megtartása esetén például a forrásnak több adatot kell pufferbe tennie, amelyet még nem tud használni. Ezzel ellentétben a valós idejű médiában egy új csomag fontosabb, mint egy régi, mivel a csomag használhatatlan lesz, ha késleltetve érkezik és lekési azt az időpontot, amikor le kellene játszani a felhasználó számára.

Az előbbi koncepciót (a régi jobb, mint az új) gyakran **bor (wine) politikának**, az utóbbit (az új jobb, mint a régi) gyakran **tej (milk) politikának** nevezik, mivel az emberek többsége inkább friss tejet és régebbi bort iszik, mint fordítva.

Az ennél eggyel magasabb intelligenciaszintre lépés együttműködést igényel az adótól. Az útválasztási információt hordozó csomagok szolgáltatnak erre példát. Ezek a csomagok fontosabbak, mint a normál adatcsomagok, mivel ezek alakítják ki az útvonalakat. Ha ezek elvesznek, akkor a hálózat elveszítheti a kapcsolatot. Másik példa a mozgókép-tömörítő algoritmusok, mint például az MPEG, amelyek periodikusan visznek át egy egész keretet, és a következő kereteket az utolsó teljes képtől való különbség formájában küldik. Ebben az esetben egy olyan csomag eldobása, amely a különbség része, előnyösebb, mint egy olyan eldobása, amely egy teljes keret része, mivel a jövőbeli csomagok a teljes kerettől függenek.

Intelligens csomageldobó politika megvalósításához az alkalmazásoknak a csomagjaikat prioritás-osztályoknak megfelelően kell megjelölniük a fontosságuk jelzésére. Ha így tesznek, akkor csomagok eldobása esetén az útválasztók először a legalacsonyabb osztályú csomagokat dobják el, majd a következő legalacsonyabb osztályúakat és így tovább.

Persze, hacsak nincs erőteljes ösztönzés arra nézve, hogy a csomagokat máshogy kelljen megjelölni, mint csak NAGYON FONTOS – SOHA NE DOBD EL, senki nem fogja a csomagokat osztályba sorolni.

Gyakran számlázást és díjfizetést alkalmaznak a felelőtlen jelzések megakadályozására. A hálózat például hagyhatja, hogy a feladók gyorsabban küldjenek, mint az általuk megvásárolt szolgáltatás, ha a többletcsomagokat alacsony prioritásúként jelölik meg. Egy ilyen stratégia tulajdonképpen nem rossz ötlet, mivel jobban kihasználja a tétlen erőforrásokat, s megengedi a hosztoknak, hogy mindaddig használják ezeket az erőforrásokat, amíg más nem érdeklődik iránt uk, anélkül azonban, hogy a hosztok nehezebb időkben is jogot formálhatnának ezekre az erőforrásokra.

Véletlen korai detektálás

Jól tudjuk, hogy hatékonyabb, ha azonnal, az észlelés pillanatától elkezdünk foglalkozni a torlódással, mint ha hagynánk, hogy elduguljon minden, és csak azután próbálnánk meg foglalkozni vele. Ez a megfigyelés vezetett ahhoz az ötlethez, hogy már azelőtt dobáljuk el a csomagokat, mielőtt teljesen kifogynánk a pufferterületből.

Az alapötlet motivációja az, hogy a legtöbb internetes hoszt még nem kap az útválasztótól ECN formájú torlódásjelzést. Ehelyett az egyetlen megbízható torlódásjelzés, amelyet a hosztok a hálózattól kapnak, a csomagvesztés. Ezután nehéz olyan útválasztót kialakítani, amely nem dob el csomagokat túlterhelés esetén. A szállítási protokollok, mint például a TCP, a forrás lassításával reagálnak a torlódás miatti csomagvesztésre. Logikájuk mögött az az érvelés húzódik meg, hogy a TCP-t vezetékes hálózatokra tervezték, márpedig azok nagyon megbízhatók, így a csomagvesztés nagy valószínűséggel a pufferek túlcsordulásának, nem pedig az átviteli hibáknak a következménye. A vezeték nélküli adatkapcsolatoknak az átviteli hibákat az adatkapcsolati rétegben kell kijavítaniuk (így ezek nem láthatók a hálózati rétegben) annak érdekében, hogy megfelelően működjenek a TCP-vel.

Ezt a tényt kihasználhatjuk arra, hogy segítsük csökkenteni a torlódásokat. Az alapötlet az, hogy ha az útválasztók már azelőtt elkezdik eldobálni a csomagokat, hogy a helyzet végképp reménytelenné válna, akkor marad idő arra, hogy valamilyen lépéseket tegyünk, mielőtt túl késő lenne. Egy erre szolgáló, népszerű algoritmus a **RED (Random Early Detection – véletlen korai detektálás)** [Floyd és Jacobson, 1993]. Az útválasztók figyelik a sorhosszaik pillanatnyi átlagát, és ennek segítségével állapítják meg, hogy mikor kell elkezdeniük a csomagokat eldobálni. Ha az átlagos sorhossz valamelyik vonalon túllép egy küszöbszintet, akkor azt a vonalat torlódottnak tekintik, és a csomagok egy része véletlenszerűen eldobásra kerül. A csomagok véletlenszerű eldobása megnöveli annak a valószínűségét, hogy a leggyorsabb feladók érzékeljenek csomageldobást. Ez a legjobb lehetőség, mivel az útválasztó nem tudja megmondani, hogy melyik forrás okozza a legtöbb bajt a datagramalapú hálózatban. Az érintett feladó észleli a csomagvesztést, mivel nincs nyugta, és a szállítási protokoll lassul. Ezáltal az elveszett csomag ugyanazt az üzenetet adja át, mint a lefojtócsomag, de implicit módon, mivel az útválasztónak nem kell explicit jelzést küldeni.

A RED-útválasztók javítják a teljesítőképességet azon útválasztókhoz viszonyítva, amelyek csak akkor dobnak el csomagokat, ha a pufferük tele van, azonban szükség lehet a hangolásukra a megfelelő működéshez. Az ideális csomageldobási szám attól függ, hogy hány feladót kell értesíteni a torlódásról. A preferált lehetőség azonban az ECN, amennyiben rendelkezésre áll. Pontosan ugyanúgy működik, mint a RED, de explicit módon küld torlódásjelzést, nem csomagvesztés formájában. A RED akkor kerül felhasználásra, ha a hosztok nem tudnak explicit jelzést fogadni.

5.4. A szolgáltatás minősége

Az előző szakaszokban látott megoldások célja a torlódások csökkentése és a hálózati teljesítőképesség növelése. Vannak azonban olyan alkalmazások (és ügyfelek), amelyek komolyabb garanciát igényelnek a hálózat teljesítőképességével szemben, mint a „legtöbb, ami az adott körülmények között elérhető". Különösen a multimédiás alkalmazások működéséhez szükséges a minimális áteresztőképesség és a maximális késleltetés meghatározása. Ebben a fejezetben folytatjuk a hálózati teljesítőképesség vizsgálatát, de jobban koncentrálunk az alkalmazás igényeit kielégítő szolgáltatásminőségre. Ezen a téren az internet hosszú távú frissítésen megy keresztül.

Egyszerű megoldásnak tűnik a jó szolgáltatásminőség biztosításához olyan hálózat kialakítása, amelynek kapacitása megfelel a rajta átmenő forgalomnak. Ezt **túlméretezésnek (overprovisioning)** hívjuk. Az így kialakított hálózat az alkalmazási forgalmat jelentős veszteség nélkül bonyolítja le, egy megfelelő útválasztási séma feltételezésével, amely a csomagokat kis késleltetéssel kézbesíti. A teljesítőképesség nem lehet ennél jobb. Bizonyos mértékig a telefonhálózat is túlméretezett, mivel ritkán fordul elő, hogy felvesszük a telefont, és nem kapunk rögtön tárcsahangot. Egyszerűen rendelkezésre áll annyi kapacitás, hogy az igényeket mindig ki lehet elégíteni.

Ennek a megoldásnak a gyenge oldala, hogy drága. Lényegében pénzkidobással oldja meg a problémát. A szolgáltatásminőségi mechanizmusok lehetővé teszik, hogy a kisebb kapacitású hálózatok az alkalmazás követelményeinek kisebb költség mellett feleljenek meg. A túlméretezés a várható forgalmon alapul. Ez azonban mit sem ér, ha a forgalmi minták nagyon megváltoznak. A szolgáltatásminőséget garantáló módszerekkel a hálózat még forgalmi csúcsok esetén is meg tud felelni a teljesítőképességgel kapcsolatos garanciáknak, néhány kérés elutasításának az árán.

A szolgáltatásminőség biztosításához négy kérdésre kell választ adni:

1. Milyen alkalmazásokra van szüksége a hálózatnak?
2. Hogyan szabályozható a hálózatba belépő forgalom?
3. Hogyan tarthatók fenn erőforrások az útválasztón a teljesítőképesség garantálásához?
4. Biztonságosan tud-e fogadni a hálózat további forgalmat?

Önmagában egyetlen technika sem oldja meg hatékony módon ezeket a problémákat. Számos eljárást fejlesztettek már ki hálózati (és szállítási) rétegbeli használatra. A szolgáltatás minőségére a gyakorlatban alkalmazott megoldások több módszert ötvöznek. A következőkben a szolgáltatásminőség kétféle változatát ismertetjük: az integrált szolgáltatásokat és a differenciált szolgáltatásokat.

5.4.1. Alkalmazási követelmények

Egy forrásból egy adott címzett csomópont felé tartó csomagok áramát **folyamnak (flow)** nevezzük. Összeköttetés-alapú hálózatban az ugyanazon folyamhoz tartozó csomagok ugyanazt az útvonalat járják be; összeköttetés nélküli hálózatokban haladhatnak különböző utakon is. Az egyes folyamok igényeit alapvetően négy paraméterrel írhatjuk le: megbízhatóság, késleltetés, dzsitter és sávszélesség. Ezek együttesen határozzák meg a folyam által igényelt **szolgáltatásminőséget (Quality of Service, QoS)**.

Az 5.27. ábra néhány általános alkalmazást és azok hálózati követelményeit mutatja be. Vegyük figyelembe, hogy a hálózati követelmények kevésbé szigorúak, mint az alkalmazási követelmények azokban az esetekben, amikor az alkalmazás javítani tud a hálózat által biztosított szolgáltatáson. A hálózatnak nem kell veszteségmentesnek lennie a megbízható fájlátvitelhez, és nem kell azonos késleltetéssel kézbesíteni a csomagokat hang és videó lejátszása esetén. Bizonyos mennyiségű csomagvesztés újraküldéssel kijavítható, és valamilyen mértékű dzsitter kiegyenlíthető a csomagok fogadónál történő pufferelésével. Az alkalmazások azonban semmit sem tudnak tenni olyan helyzetekben, ahol túl kicsi a hálózati sávszélesség vagy túl nagy a késleltetés.

Az alkalmazások sávszélesség-igénye eltérő. Az e-mail, a hangátvitel minden formája, valamint a **távoli bejelentkezés (remote login)** nem igényel nagy sávszélességet, viszont a fájlmegosztás és a videofolyam minden formájához nagy sávszélesség szükséges.

A késleltetési követelmények még érdekesebbek. A fájlátviteli alkalmazások – beleértve az e-mailt és videót is – a késleltetésre nem érzékenyek. Ha az összes csomag késleltetése egyformán néhány másodperc, az nem okoz problémát. Az interaktív alkalmazások, például a weben való szörfözés vagy a távoli bejelentkezés azonban már sokkal érzékenyebbek a késleltetésre. A valós idejű alkalmazások pedig, például a telefon- és a videokonferencia, már kimondottan szigorú követelményeket támasztanak a késleltetéssel szemben. Ha egy telefonhívás során minden szót túl hosszú ideig késleltetünk, a felhasználók elfogadhatatlannak fogják tartani a kapcsolatot. Ezzel szemben a hang- vagy videoállományok egy kiszolgálóról történő lejátszása nem igényel alacsony késleltetést.

Alkalmazás	Sávszélesség	Késleltetés	Dzsitter	Veszteség
E-mail	Kicsi	Kicsi	Kicsi	Közepes
Fájlmegosztás	Nagy	Kicsi	Kicsi	Közepes
Webes hozzáférés	Közepes	Közepes	Kicsi	Közepes
Távoli bejelentkezés	Kicsi	Közepes	Közepes	Közepes
Hálózati zenehallgatás	Kicsi	Kicsi	Nagy	Kicsi
Hálózati videózás	Nagy	Kicsi	Nagy	Kicsi
Telefon	Kicsi	Nagy	Nagy	Kicsi
Videokonferencia	Nagy	Nagy	Nagy	Kicsi

5.27. ábra. *Alkalmazások szolgáltatásminőségi követelményeinek szigorúsága*

A késleltetésnek vagy a csomagok érkezési idejének a változását (azaz a szórást) **dzsitternek** (**jitter**) hívják. Az 5.27. ábrán lévő első három alkalmazás nem érzékeny arra, ha a csomagok szabálytalan időközönként érkeznek be. A távoli bejelentkezés egy kicsit már kényesebb, mivel a karakterek apró löketekben fognak megjelenni a képernyőn, ha a kapcsolat nagyobb dzsitterrel terhelt. A videó, de különösen a hang, rendkívül érzékeny a dzsitterre. Az nem okoz gondot, ha egy felhasználó egy filmet néz a hálózaton keresztül, és a keretek pontosan 2 másodperc késleltetést szenvednek. De ha az átvitel ideje véletlenszerűen ingadozik 1 és 2 másodperc között, máris tragikus eredményt kapunk. A hangátvitelnél még a néhány milliszekundumos dzsitter is tisztán hallható.

Az első négy alkalmazás magasabb követelményeket támaszt a csomagvesztéssel szemben, mint a hang- és videoátvitel, mivel az összes bitet helyesen át kell vinni. Ez a cél rendszerint úgy érhető el, hogy a szállítási réteg újraküldi az elveszett csomagokat a hálózaton. Ez azonban pazarlás. Jobb lenne, ha a hálózat visszautasítaná azokat a csomagokat, amelyek először nagy valószínűséggel elvesznek. A hang- és videoalkalmazások el tudják viselni néhány csomag elvesztését újraküldés nélkül, mivel a felhasználók nem észlelik a rövid szüneteket vagy az alkalmanként kihagyott kereteket.

A hálózatok különböző QoS-kategóriákat támogathatnak a különböző alkalmazások igényeinek kielégítése érdekében. Az egyik sokszor emlegetett példa az ATM-hálózatból származik, amelyek egykor a hálózatról alkotott nagy látomás részét képezték, de azóta elavult technikává vált. Ezek a hálózatok a következő QoS-kategóriákat támogatják:

1. Állandó adatsebesség (például telefon).
2. Valós idejű, változó adatsebesség (például tömörített videokonferencia).
3. Nem valós idejű, változó adatsebesség (például filmet nézni az interneten keresztül).
4. Rendelkezésre álló adatsebesség (például fájlátvitel).

Ezek a kategóriák más hálózatokban, más célokra is hasznosak lehetnek. Az állandó adatsebességgel egy olyan vezetéket szimulálhatunk, amely változatlan sávszélességet és változatlan késleltetést biztosít. Változó adatsebesség például akkor fordulhat elő, ha egy mozgóképet tömörítünk, és némely keret jobban tömöríthető, mint a többi. Így egy nagyon részletgazdag keret elküldése sok bitet igényelhet, egy fehér fal képe viszont rendkívül jól tömöríthető. A hálózaton történő filmnézés valójában nem valós idejű, mivel néhány másodperces videó egyszerűen tömöríthető a fogadónál a lejátszás indítása előtt, így a hálózat dzsitterének hatására csupán a tárolt-de-nem-lejátszott videó mennyisége változik. A rendelkezésre álló adatsebesség az olyan alkalmazások számára megfelelő, amelyek nem érzékenyek a késleltetésre vagy a dzsitterre, mint például az e-mail.

5.4.2. Forgalomformálás

Ahhoz, hogy a hálózat QoS-garanciákat biztosíthasson, tudnia kell, hogy milyen forgalomhoz szükséges a garancia. Telefonhálózat esetén ez a jellemzés egyszerű. Például

egy hanghívás (tömörítetlen formátumban) 64 kb/s-os sebességet igényel, és egy 8 bites mintát tartalmaz 125 mikroszekundumonként. Az adathálózatokban azonban a forgalom **löketekben** érkezik. Jellemzően változó sebességgel érkezik, ahogy a forgalom sebessége változik (például tömörített videokonferencia), ahogy a felhasználók használják az alkalmazásokat (például új weboldal böngészése), illetve ahogy a számítógép kapcsolgat a feladatok között. Az ingadozó forgalom kezelése bonyolultabb, mint az állandó sebességű fogalomé, mivel a pufferek telítődhetnek és csomagok veszhetnek el.

A **forgalomformálás** (**traffic shaping**) a hálózatra belépő adatfolyam átlagos sebességének és ingadozásának (löketességének) szabályozására szolgáló technika. A cél az, hogy az alkalmazások az igényeiknek megfelelő, különböző típusú – löketeket is tartalmazó – forgalmat vihessenek át, továbbá, hogy rendelkezzenek egy egyszerű és használható eszközzel a hálózat lehetséges forgalmi mintáinak leírására. Amikor egy folyam felépül, a felhasználó és a hálózat (azaz az ügyfél és a szolgáltató) megegyeznek egy adott forgalmi mintázatban az adott folyamhoz. A valóságban az ügyfél a következőt mondja a szolgáltatónak: „Ilyen az átviteli mintám. Képes vagy ezt kezelni?"

Ezt **szolgáltatásszintű megállapodásnak** (**Service Level Agreement, SLA**) is hívják, különösen akkor, ha összesített folyamra és hosszú időtartamra vonatkozik, mint például egy ügyfél teljes forgalma. Mindaddig, amíg az ügyfél betartja az alku rá vonatkozó részét, és csak az elfogadott szerződésnek megfelelő módon küld csomagokat, a szolgáltató vállalja, hogy időben le is szállítja azokat.

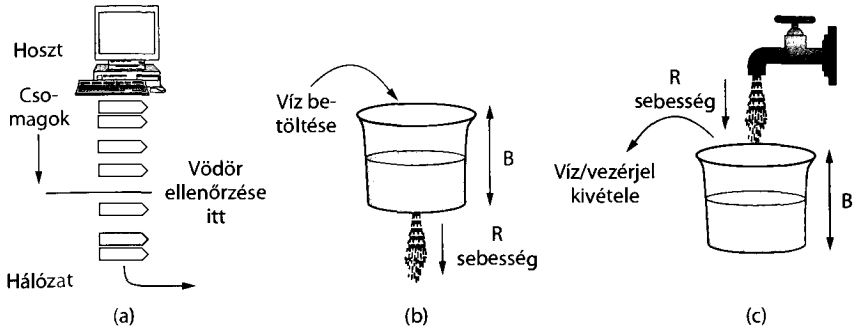
A forgalomformálás csökkenti a torlódást, ezáltal segíti a hálózatot abban, hogy teljesítse az ígéretét. Ahhoz azonban, hogy ez működjön, felmerül az a kérdés is, hogy vajon a szolgáltató hogyan tudja megállapítani, hogy az ügyfél tartja-e magát a megállapodáshoz, és mit lehet tenni, ha kiderül, hogy nem? Az egyeztetett mintát meghaladó csomagokat a hálózat eldobhatja vagy alacsony prioritásúként jelölheti meg. A forgalom folyamának figyelését **forgalmi rendfenntartásnak** (**traffic policing**) nevezzük.

A forgalomformálás és a rendfenntartás nem túl fontos a P2P-hálózatoknál és más olyan adatok átvitele esetén, amelyek a teljes rendelkezésre álló sávszélességet vagy annak egy részét felhasználják. Nagy a jelentősége azonban az olyan valós idejű adatok – mint például a hang- és videoadatok – átvitele esetén, ahol szigorúak a szolgáltatásminőségi követelmények.

Lyukas vödör és vezérjeles vödör

Korábban már láthattunk egy módszert az alkalmazás által küldött adatok mennyiségének korlátozására: ez az ún. csúszóablak, amely egy paramétert használ az adott idő alatt átvitt adatok mennyiségének korlátozására, és amely indirekt módon korlátozza a sebességet. Most általánosabb módszert ismertetünk a forgalom jellemzésére: a lyukas vödör (leaky bucket) és a vezérjeles vödör (token bucket) algoritmust. A két módszer kissé különbözik, de azonos eredményre vezetnek.

Képzeljünk el egy vödröt, az alján egy kis lyukkal, ahogy az 5.28.(b) ábrán látható. Mindegy, hogy a víz milyen sebességgel érkezik a vödörbe, a kimenő folyam konstans R sebességű, amikor van víz a vödörben, és nulla, amikor a vödör üres. Ha a vödör a B kapacitásig megtelt, az ezután érkező víz kicsordul a vödörből és elvész.



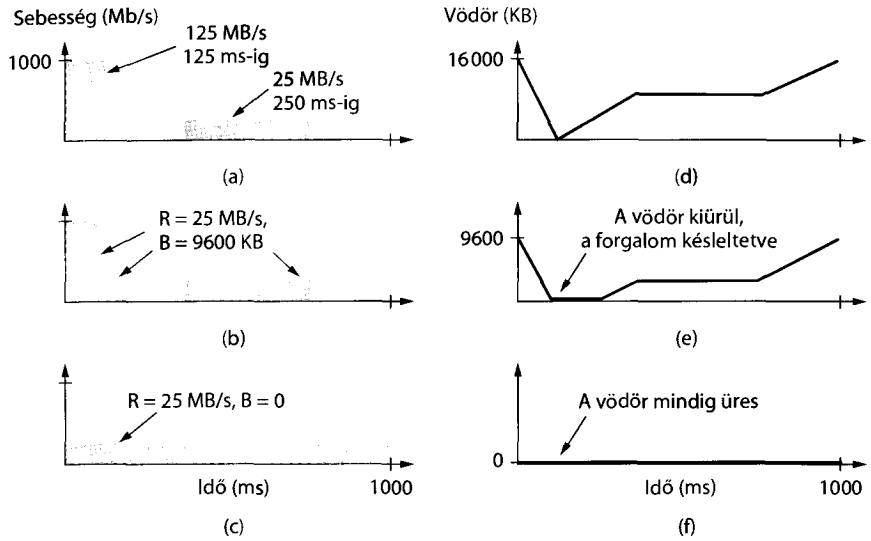
5.28. ábra. *(a) Forgalomformáló csomagok (b) Lyukas vödör. (c) Vezérjeles vödör*

Ugyanez az ötlet alkalmazható a hálózatra belépő csomagokra is, amely szerint minden hoszt egy lyukas vödröt tartalmazó interfészen keresztül csatlakozik a hálózatra (ahogy azt az 5.28.(a) ábra mutatja). Egy csomag hálózatra küldéséhez lehetővé kell tenni több víz betöltését a vödörbe. Ha egy csomag akkor érkezik, amikor a vödör tele van, a csomagot vagy sorba kell állítani, amíg nem folyik ki elég víz, vagy el kell dobni. Az előbbi olyan hoszton lehetséges, amely a hálózati forgalmat az operációs rendszer részeként formálja. Az utóbbi a szolgáltató hálózati interfészének hardverén lehetséges, amely a hálózatra belépő forgalom rendfenntartását intézi. Ezt először Turner [1986] javasolta, és **lyukas vödör (leaky bucket) algoritmusnak** nevezte.

Alapjában véve a lyukas vödörrel azonos elképzelés, bár attól eltérő megoldás, ha a hálózati interfészt olyan vödörként képzeljük el, amely meg van töltve az 5.28.(c) ábrán látható módon. A csap *R* sebességgel engedi a vizet, a vödör kapacitása pedig *B*, mint ahogy korábban. Egy csomag küldéséhez vizet vagy vezérjeleket – ahogy a vödör tartalmát rendszerint hívják – kell tudnunk kivenni a vödörből (ahelyett, hogy vizet tennénk a vödörbe). Egy rögzített számú (*B*) vezérjelnél több nem halmozható fel a vödörben, és ha a vödör üres, akkor másik csomag küldése előtt várni kell további vezérjel érkezésére. Ezt az algoritmust **vezérjeles vödör (token bucket) algoritmusnak** hívják.

A lyukas vödör és vezérjeles vödör korlátozza a folyam hosszú távú sebességét, de engedélyezi, hogy a rövid távú löketek a maximális szabályozott hosszig változatlanul menjenek át, bármilyen mesterséges késleltetés nélkül. A nagy löketeket a lyukas vödör, mint forgalomformáló, elsimítja, hogy csökkentse a hálózati torlódást. Képzeljünk el például egy számítógépet, amely maximum 1000 Mb/s-os sebességgel tud adatokat előállítani (125 millió bájt/sec) és a hálózat első vonala szintén ezzel a sebességgel működik. A hoszt által generált forgalom mintája az 5.29.(a) ábrán látható. Ez a minta löketeket tartalmaz. Az átlagos sebesség több mint egy másodpercig 200 Mb/s, még akkor is, ha a hoszt 16 000 KB-os löketet küld a maximális 1000 Mb/s-os sebességgel (a másodperc 1/8 részéig).

Tételezzük fel, hogy az útválasztók a maximális sebességű adatokat csak rövid ideig tudják fogadni, amíg a pufferek meg nem telnek. A pufferméret 9600 KB. Ez kisebb, mint a forgalomlöket. Hosszú távon az útválasztók akkor működnek a legjobban, ha a sebesség nem haladja meg a 200 Mb/s-os sebességet (mivel ez az ügyfél számára biztosított sávszélesség). Az ilyen minta szerint érkező forgalom következménye, hogy a



5.29. ábra. *(a) Hoszt forgalma. Vezérjeles vödör által formált kimenet 200 Mb/s-os sebességgel és (b) 9600 KB és (c) 0 KB kapacitással. Vezérjeles vödör szintje a formáláshoz 200 Mb/s sebességgel és (d) 16 000 KB (e) 9600 KB és (f) 0 KB kapacitással*

csomagok egy része eldobásra kerül a hálózaton, mivel az összes csomag nem fér be az útválasztókon lévő pufferekbe.

A csomagvesztés elkerülése érdekében a forgalmat a hoszton vezérjeles vödörrel alakíthatjuk. Ha az *R* sebesség 200 Mb/s, a *B* kapacitás pedig 9600 KB, akkor a forgalom a hálózat által kezelhető tartományba esik. A vezérjeles vödör kimenetét az 5.29.(b) ábra mutatja. A hoszt rövid ideig adhat 1000 Mb/s-os sebességgel, amíg a vödör meg nem telik. Utána vissza kell fognia a sebességet 200 Mb/s-ra, amíg a löket elküldésre nem kerül. A löket hatása időben elhúzódik, mivel túl nagy ahhoz, hogy a hálózat egyszerre lekezelje. A vezérjeles vödör szintje az 5.29.(e) ábrán látható. A vödör induláskor tele van, és a kezdeti löket üríti ki. Amikor üres, akkor új csomagok csak olyan sebességgel küldhetők, amilyen sebességgel a puffer töltődik. Nem lehet több löket, amíg a vödör ismét tele nem lesz. A vödör töltődik, ha nincs forgalom, és töltöttsége egyenletes marad, ha a forgalom sebessége megegyezik a töltési sebességgel.

Simább forgalmat is előállíthatunk. Az 5.29.(c) ábra a vezérjeles vödör kimenetét mutatja R = 200 Mb/s sebesség és 0 kapacitás mellett. Ez egy szélsőséges eset, amelyben a forgalom teljesen kiegyenlített. A löketek nem megengedettek, és a forgalom egyenletes sebességgel lép be a hálózatra. A megfelelő vödörszint (lásd 5.29.(f) ábra) mindig üres. A forgalom sorba állításra kerül a hoszton a hálózatra kerülés érdekében, és mindig van küldésre váró csomag, ha az engedélyezett.

Végül az 5.29.(d) egy vezérjeles vödör szintjét mutatja R = 200 Mb/s sebesség és B = 16 000 KB kapacitás mellett. Ez a legkisebb vezérjeles vödör, amelyen keresztül a forgalom változatlanul halad. Ez használható a hálózaton lévő útválasztón a hoszt által küldött forgalom rendjének fenntartásához. Ha a hoszt által küldött forgalom megfelel

a hálózattal egyeztetett vezérjeles vödörnek, akkor a forgalom a hálózat szélén lévő útválasztón futó vezérjeles vödörnek is megfelel. Ha a hoszt olyan sebességgel küldi a forgalmat, amely megfelel annak a vezérjeles vödörnek, amelyről a hálózattal megegyezett, akkor a forgalom meg fog felelni a hálózat szélén lévő útválasztón futó vezérjeles vödörnek is. Ha a hoszt gyorsabban vagy több lökettel forgalmaz, akkor a vezérjeles vödörből kifogy a víz. Amennyiben ez megtörténik, akkor a forgalom rendfenntartója tudja, hogy a forgalom nem a leírt módon halad. Ekkor a rendfenntartó a hálózat kialakításától függően vagy eldobja a többletcsomagokat, vagy csökkenti e csomagok prioritási szintjét. Példánkban a vödör a kezdeti löket végén csak egy pillanatra ürül ki, majd visszaáll a következő lökethez elegendő kapacitása.

A lyukas és a vezérjeles vödröt egyszerű megvalósítani. Most a vezérjeles vödör működését írjuk le. Annak ellenére, hogy a vödörbe befolyó és a vödörből kifolyó vízfolyamot folyamatosnak mondtuk, a gyakorlatban diszkrét mennyiségekkel kell működniük. A vezérjeles vödröt egy számlálóval valósítják meg, ami a vödör telítettségi szintjét mutatja. A számláló értéke ΔT másodpercenként $R/\Delta T$ egységgel növekszik. Ez az egység a fenti példában 200 kbit 1 milliszekundumonként. A számláló értéke eggyel csökken minden alkalommal, amikor adatok kerülnek elküldésre a hálózat irányába. Addig lehet adatokat küldeni, amíg a számláló el nem éri a nullát.

Ha a csomagok azonos méretűek, akkor a vödörszint mérhető csomagokban (például a 200 kbit 20 darab 1250 bájtos csomag). A csomagok azonban gyakran eltérő méretűek. Ebben az esetben a vödörszintet bájtban mérik. Ha a meglévő bájtszám túl kicsi nagy csomag küldéséhez, akkor a csomagnak várnia kell a következő órajelre (vagy még tovább, ha a töltési sebesség kicsi).

A maximális sebességű löket hosszának kiszámítása kissé trükkös. Nem egyszerűen 9600 KB osztva 125 MB/s-mal, mert miközben kiadjuk a löketet, további vezérjelek érkeznek. Ha a lökethossz S másodperc, a maximális kimeneti sebesség M bájt/s, akkor láthatjuk, hogy a kimeneti löket maximum $B+RS$ bájtot tartalmaz. Azt is tudjuk, hogy a bájtok száma egy S másodperc hosszú, maximális sebességű löketben MS. Így kapjuk a következőt:

$$B+RS=MS$$

Ezt az egyenletet megoldva kapjuk a következőt: $S=B/(M-R)$. A megadott paraméterekkel, ahol B = 9600 KB, M = 125 MB/s, és R = 25 MB/s, 94 ms körüli löketidőt kapunk.

Egy lehetséges probléma a vezérjeles vödör algoritmusával, hogy a nagy löketeket lelassítja a hosszú távú R sebességre. Gyakran kívánatos a csúcssebesség csökkentése, de a lyukas vödör hosszú távú sebességéhez való visszatérés nélkül (és a hosszú távú sebesség növelése nélkül, amely több forgalmat engedélyezne a hálózaton). A simább forgalom előállításának egy módja, hogy egy újabb vezérjeles vödröt teszünk az első után. Alapvetően az első vödör jellemzi a forgalmat: rögzíti az átlagsebességet, de engedélyez némi löketet. A második vödör csökkenti a löketek hálózatba küldésének csúcssebességét. Ha például a második vezérjeles vödör sebessége 500 Mb/s, a kapacitás pedig 0, a kezdeti löket 500 Mb/s-os csúcssebességgel lép be a hálózatba, amely alacsonyabb a korábbi 1000 Mb/s-os sebességnél.

Ezeknek a vödröknek az alkalmazása egy kissé trükkös lehet. Ha vezérjeles vödröt használunk a forgalomformáláshoz a hosztokon, akkor a csomagok sorban állnak és késleltetést szenvednek, amíg a vödör nem engedélyezi a küldésüket. Ha vezérjeles vödröt használunk a forgalom rendjének fenntartásához a hálózat útválasztóin, akkor az algoritmus szimulálásra kerül annak ellenőrzése érdekében, hogy a megengedettnél több csomag nem kerül elküldésre. Ezek az eszközök kezelhetőbbé teszik a hálózati forgalmat, elősegítik a szolgáltatásminőségi követelmények kielégítését.

5.4.3. Csomagütemezés

A szolgáltatásminőség szempontjából jó kezdet, ha szabályozni tudjuk a felkínált forgalom alakját. A teljesítőképesség garantálásához azonban elegendő erőforrást kell lefoglalni a csomagok hálózaton követett útvonala mentén. Ehhez feltételezzük, hogy a csomagok folyama mindig ugyanazt az útvonalat követi. Nehéz lenne bármit is garantálni úgy, hogy a csomagokat véletlenszerűen szórjuk szét a hálózaton. Következésképpen, a forrás és cél között valamilyen, a virtuális áramkörhöz hasonló kapcsolatot kell létrehozni, és a folyamhoz tartozó összes csomagnak ezt az útvonalat kell követnie.

Az algoritmust, amely útválasztó erőforrásokat oszt ki egy folyam csomagjai között és a versengő folyamok között, **csomagütemezési algoritmusnak** (**packet scheduling algorithm**) hívják. Háromféle erőforrást lehet lefoglalni a különféle folyamok számára:

1. sávszélességet,
2. pufferterületet,
3. processzoridőt.

Az első, a sávszélesség a legkézenfekvőbb. Ha egy folyamnak 1 Mb/s sávszélességre van szüksége, és a kimeneti vonal kapacitása 2 Mb/s, akkor azon a vonalon nyilván nem fogunk tudni három ilyen folyamot átvezetni. A sávszélesség lefoglalása tehát annyit jelent, hogy nem foglaljuk túl a kimeneti vonalat.

A második erőforrás, amiből gyakran hiányt szenvedünk, a pufferterület. Amikor egy csomag megérkezik, az útválasztón pufferelésre kerül, amíg át nem küldhető a választott kimeneti vonalon. A puffer célja a forgalom kis löketeinek elnyelése, amíg a folyamok versengenek egymással. Ha nincs szabad puffer, a csomagot el kell dobni, mivel egyszerűen nincs hová tenni. A jobb szolgáltatásminőség érdekében néhány puffer fenntartható meghatározott folyam számára, hogy az adott folyamnak ne kelljen a pufferért versengenie a többiekkel. Ily módon egy bizonyos határig mindig lesz szabad puffer, amikor a folyamnak szüksége van rá.

Végül a processzoridő is szűkös erőforrásnak számít. Az útválasztónak minden csomag feldolgozásához bizonyos processzoridőre van szüksége, így egy másodperc alatt csak korlátozott számú csomag feldolgozására van lehetőség. A modern útválasztók a legtöbb csomagot gyorsan fel tudják dolgozni, bizonyos csomagok azonban gyorsabb processzorfeldolgozást igényelnek, mint például az ICMP-csomagok, amelyekről az

5.6. szakaszban lesz szó. Ahhoz, hogy minden csomagot megfelelő időben feldolgozhassunk, biztosítani kell, hogy a processzor ne legyen túlterhelve.

A csomagütemezési algoritmusok a sávszélességet és más útválasztó erőforrásokat úgy foglalják le, hogy meghatározzák, melyik, a pufferben elküldésre váró csomagokat kell következőnek a kimeneti sorba helyezni. Az útválasztók működésének bemutatásánál már ismertettük a legegyszerűbb ütemezőt. Az útválasztók egy várakozási sorba rakják (pufferelik) a csomagokat minden kimeneti vonalhoz, amíg azok el nem küldhetők. A csomagok ugyanabban a sorrendben kerülnek elküldésre, mint ahogy érkeztek. Ezt az algoritmust **FIFO-nak** (**First-In First-Out – elsőnek be, elsőnek ki**) vagy **FCFS**-nek hívják (**First-Come First-Serve – elsőként bekerül, elsőként kiszolgálva**).

A FIFO-útválasztók általában akkor dobják el az újonnan érkezett csomagokat, ha a sor tele van. Mivel az újonnan érkezett csomag a sor végére kerül, ezt a viselkedést a **farokrész eldobásának** (**tail drop**) nevezik. Ez elgondolkodtathat bennünket, hogy milyen alternatívák léteznek. Az 5.3.5. részben leírt RED-algoritmus az újonnan érkezett csomagok közül véletlenszerűen dob el, amikor az átlagos sorhossz túl nagyra nő. A másik leírt ütemezési algoritmus egyéni lehetőségeket is ad annak eldöntéséhez, hogy mely csomagot kell eldobni, ha a pufferek tele vannak.

A FIFO-ütemezést egyszerű megvalósítani, de nem megfelelő a jó szolgáltatásminőség biztosításához, mivel ha több folyam van, akkor egy folyam könnyen befolyásolhatja más folyamok teljesítőképességét. Ha az első folyam agresszív, és nagy csomaglöketeket küld, akkor a csomagok egy várakozási sorba kerülnek. A csomagok beérkezési sorrendben történő feldolgozása azt jelenti, hogy az agresszív küldő lekötheti az útválasztó kapacitásának nagy részét, amelyen a csomagjai áthaladnak, kiéheztetve a többi folyamot, csökkentve ezáltal a szolgáltatásminőséget. És ami még rosszabb, más folyamok csomagjai, amelyek keresztülmennek ezen az útválasztón, valószínűleg késleltetést szenvednek, mivel sorban állnak az agresszív küldő csomagjai mögött.

Számos csomagütemezési algoritmus létezik, amelyek jobban elszigetelik a folyamokat, és megakadályozzák a zavarási kísérleteket [Bhatti és Crowcroft, 2000]. Az első ilyen elgondolások között volt az **egyenlő esélyű sorba állítás** (**fair queuing**) algoritmusa, amelyet Nagle [1987] írt le. Az algoritmus lényege az, hogy az útválasztók külön várakozási sorokat tartanak fenn minden kimeneti vonalon, minden folyam számára egyet. Ha egy vonal tétlen lesz, az útválasztó körforgásos (round-robin) alapon végignézi a sorokat, ahogy azt az 5.30. ábra szemlélteti, majd kivesz egy csomagot a következő sorból. Ily módon, ha *n* hoszt verseng egy adott kimeneti vonalért, akkor mindegyik egy csomagot küldhet el minden *n* csomagból. Ez abból a szempontból egyenlő esélyt biztosít, hogy az összes folyam azonos aránnyal küldhet csomagot. Több csomag küldése nem javítja ezt az arányt.

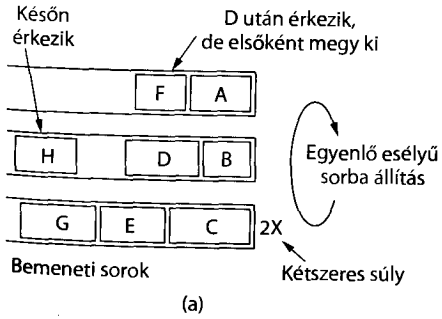
Bár kiindulásnak jó, az algoritmusnak van egy hiányossága: nagyobb sávszélességet ad a nagy csomagokat használó hosztoknak, mint a kis csomagokat használó hosztoknak. Demers és mások [1990] ennek kiküszöbölésére azt javasolták, hogy a körforgást bájtonkénti körforgásként szimulálják a csomagonkénti körforgás helyett. A trükk a körforgások számát megadó virtuális idő kiszámítása, amely alatt az egyes csomagok elküldése befejeződik. Minden kör egy bájtot vesz ki minden sorból, amelyben küldendő adatok vannak. A csomagok ezután befejezési idejük sorrendjében rendeződnek, és ebben a sorrendben kerülnek elküldésre.

5.30. ábra. *Körforgó egyenlő esélyű sorba állítás*

Ez az algoritmus és egy példa látható az 5.31. ábrán, amely három folyamban érkező csomagok befejezési idejét illusztrálja. Ha egy csomag hossza *L*, akkor a befejezéshez szükséges körök száma egyszerűen *L* lesz a kezdő időpont után. A kezdő időpont az előző csomag befejezési időpontja, vagy a csomag beérkezési ideje, ha a sor üres, amikor a csomag megérkezik.

Ha az 5.31.(b) ábrán lévő táblázatban csak a felső két sorban lévő első két csomagot nézzük, akkor a csomagok a következő sorrendben érkeznek: *A*, *B*, *D* és *F*. Az *A* csomag a 0. körben érkezik és 8 bájt hosszú, így ennek befejezési ideje 8 kör. Ehhez hasonlóan a *B* csomag befejezési ideje 11. A *D* csomag megérkezik, miközben a *B* elküldése folyamatban van. Ennek befejezési ideje 9 óraütés a *B* befejezése után, azaz összesen 20. Ehhez hasonlóan az *F* befejezési ideje 16. Új érkezők hiányában a relatív küldési sorrend *A*, *B*, *F*, *D* annak ellenére, hogy *F* a *D* után érkezett. Elképzelhető, hogy másik kis csomag érkezik a felső folyamon és a befejezési ideje *D* előtti. Csak akkor kerül *D* elé, ha a *D* csomag átvitele nem kezdődött el. Az egyenlő esélyű sorba állítás nem előzi meg azokat a csomagokat, amelyek átvitele folyamatban van. Mivel a csomagok egészben kerülnek elküldésre, az egyenlő esélyű sorba állítás az ideális bájtonkénti átviteli sémának csak egy megközelítése. De ez egy nagyon jó megközelítés, amely mindenkor az ideális séma egyetlen csomagátvitelén belül marad, vagyis az ideális sémától legfeljebb egy csomagnyira távolodik el.

Ennek az algoritmusnak az egyik hibája a gyakorlatban az, hogy minden hosztnak azonos prioritást biztosít. Sok esetben kívánatos például a videokiszolgálóknak nagyobb sávszélességet adni, mint például a hagyományos fájlkiszolgálóknak. Ez egyszerűen kivitelezhető, ha két vagy több bájtot adunk nekik óraütésenként. Ezt a módosított al-



Csomag	Érkezési idő	Hossz	Befejezési idő	Kimeneti sorrend
A	0	8	8	1
B	5	6	11	3
C	5	10	10	2
D	8	9	20	7
E	8	8	14	4
F	10	6	16	5
G	11	10	19	6
H	20	8	28	8

(b)

5.31. ábra. *(a) Súlyozott egyenlő esélyű sorba állítás. (b) Csomagok befejezési ideje*

goritmust **súlyozott egyenlő esélyű sorba állításnak** (**weighted fair queueing, WFQ**) hívják. Ha az óraütésenkénti bájtok száma a folyam súlya, W, akkor a befejezési idő kiszámításának a képlete a következő lesz:

$$F_i = \max(A_i, F_{i-1}) + L_i/W$$

ahol A_i az érkezési idő, F_i a befejezési idő, L_i pedig az i csomag hossza. Az 5.31.(a) ábra alsó sorának súlya 2, így a csomagok gyorsabban kerülnek elküldésre, ahogy azt az 5.31.(b) ábrán látható befejezési idő is mutatja.

A másik gyakorlatban felmerülő probléma a bonyolult megvalósítás. WFQ esetén a csomagokat a befejezési idejük alapján kell a rendezett sorba rakni. N folyam esetén ez a legjobb esetben is $O(\log N)$ művelet csomagonként, amit a nagy sebességű útválasztókban lévő sok folyam esetén nehéz megvalósítani. Shreedhar és Varghese [1995] ennek megoldására a **különbséggel súlyozott körbejárást** (**deficit round robin** vagy **deficit weighted round robin**) javasolja, amely hatékonyan megvalósítható, csomagonként csak $O(1)$ művelettel. A WFQ-t széles körben használják ezzel a megközelítéssel.

Más típusú ütemezési algoritmusok is léteznek. Egyszerű példa a prioritásütemezés, amelyben minden csomaghoz tartozik prioritás. A magas prioritású csomagok mindig az alacsony prioritású csomagok előtt kerülnek elküldésre, amelyek pufferbe kerülnek. Az azonos prioritású csomagok FIFO-sorrendben kerülnek elküldésre. A prioritásütemezés hátránya azonban, hogy a magas prioritású csomagok lökete kiéheztetheti az alacsony prioritású csomagokat, amelyeknek így esetlegesen határozatlan ideig kell várniuk. A WFQ gyakran jobb alternatívát kínál. Ha a magas prioritású sor nagy súlyt kap, például 3-at, akkor a magas prioritású csomagok gyakran rövid vonalon mennek át (mivel viszonylag kevés csomag lehet magas prioritású), azonban az alacsony prioritású csomagok egy része továbbra is elküldésre kerül még magas prioritású forgalom esetén is. A magas és alacsony prioritású rendszer lényegében egy kétsoros WFQ-rendszer, amelyben a magas prioritásnak a súlya végtelen.

Az ütemező utolsó példájaként a csomagok tartalmazhatnak időbélyeget, és elküldhetők az időbélyeg sorrendjében. Clark és mások [1992] olyan kialakítást ismertetnek, amelyben az időbélyeg azt jelzi, hogy a csomag hol tart az ütemezéséhez képest (előtte vagy mögötte van) az útvonalon lévő útválasztókon történő átküldés során. Azok a csomagok, amelyek más csomagok mögött kerültek sorba állításra egy útválasztón, lemaradnak az ütemezéshez képest, az elsőként kiszolgált csomagok pedig gyorsabbak lesznek, mint az ütemezés. A csomagok időbélyeg szerinti küldésének előnye, hogy a lassú csomagokat felgyorsítja, a gyorsakat pedig lelassítja. Ennek eredménye, hogy a hálózat az összes csomagot következetesebb késleltetéssel kézbesíti.

5.4.4. Belépés-ellenőrzés

Már láttuk a QoS valamennyi szükséges elemét, így eljött az ideje, hogy ezeket összerakjuk a szolgáltatásminőség tényleges biztosításához. A QoS-garanciák a belépés-ellenőrzési folyamattal valósulnak meg. A belépés-ellenőrzést először a torlódáskezeléshez alkalmaztuk, amely teljesítőképességi garancia ugyan, de elég gyenge. A most tárgyalt

garanciák erősebbek, de a modell ugyanaz. A felhasználó szolgáltatásminőségi követelménnyel együtt küldi a folyamot a hálózat felé. A hálózat ezután a kapacitás és a más folyamokkal szemben vállalt kötelezettség alapján eldönti, hogy elfogadja vagy visszautasítja a folyamot. Ha elfogadja, akkor kapacitást foglal le előre az útválasztókon a QoS garantálásához, amikor a forgalom elküldésre kerül az új folyamon.

A foglalást a csomagok által bejárt útvonal összes útválasztóján meg kell tenni. A lefoglalás nélküli útválasztókon torlódás léphet fel, és egyetlen torlódott útválasztó meg tudja hiúsítani a QoS-garanciát. A legtöbb útválasztó algoritmus minden forrás és cél között igyekszik megtalálni a legjobb utat, és minden forgalmat a legjobb úton küld. Ez néhány folyam visszautasítását okozhatja, ha nincs elegendő felesleges kapacitás a legjobb útvonal mentén. Új folyamok a QoS garanciáinak kielégítéséhez még alakítható a rendszer azáltal, hogy másik útvonalat választ a kapacitást meghaladó folyamhoz. Ezt **QoS-útválasztásnak** (**QoS routing**) nevezik. Chen és Nahrstedt [1998] áttekintést adnak ezekről a technikákról. A forgalom minden célnál több útvonalra eloszthető a felesleges kapacitás egyszerűbb megtalálása érdekében. Útválasztók esetén egyszerű megoldás az, ha egyenlő költségű útvonalakat választunk és a forgalmat egyenletesen osztjuk fel, vagy megoszthatjuk a forgalmat a kimeneti vonalak kapacitásának arányában. Léteznek azonban kifinomultabb algoritmusok is [Nelakuditi és Zhang, 2002].

Adott útvonal esetén a folyam elfogadásról vagy elutasításról szóló döntés nem pusztán a folyam által kért erőforrások (sávszélesség, pufferek, processzoridő) és az útválasztó ezen három dimenzió mértékében mért felesleges kapacitásának összehasonlításából áll. Ennél azért bonyolultabb kérdésről van szó. Kezdjük azzal, hogy néhány alkalmazás tisztában van ugyan a saját sávszélesség-igényével, a pufferekről vagy a processzoridőről azonban már kevesen tudnak. Így a legkevesebb az, hogy más módot kell találnunk a folyamok leírására és a leírásnak az útválasztó-erőforrásokra való lefordítására. Hamarosan ezzel is foglalkozunk.

Továbbá, egyes alkalmazások sokkal jobban tolerálnak egy esetlegesen lekésett határidőt, mint mások. Az alkalmazásoknak választani kell a hálózat által biztosított garanciatípusok közül, amelyek lehetnek szigorú garanciák, vagy olyan viselkedés, amely az idő nagy részében biztosítható. Ha mindezek azonosak lennének, akkor mindenki szigorú garanciákat szeretne, de a gond az, hogy ez költséges, mivel ezek korlátozzák a legrosszabb esetbeli viselkedést. A csomagok nagy részének nyújtott garancia gyakran elegendő az alkalmazások számára, és több folyam ezzel a garanciával hozzájuttatható egy rögzített kapacitáshoz.

Végül, egyes alkalmazások hajlandók lehetnek a folyamparaméterekről alkudozni, mások ellenben nem. Például egy általában 30 keret/s sebességgel működő filmlejátszó hajlandó lehet 25 keret/s sebességre visszaállni, ha nincs elég sávszélesség az előbbi sebességhez. Hasonlóképpen változtatható a keretenkénti képpontok száma, a hang sávszélessége és egyéb jellemzők.

Mivel az egyeztetésben sok fél vesz részt (a feladó, a címzett és a kettejük közötti úton lévő összes útválasztó), a tárgyalás alapját képező paramétereket tekintve pontosan le kell tudnunk írni a folyamokat. Az ilyen paraméterek halmazát **folyammeghatározásnak** (**flow specification**) hívjuk. Általában a feladó (például egy videokiszolgáló) állítja elő a folyammeghatározást, amikor is ajánlatot tesz az általa használni kívánt paraméterekre. Ahogy a meghatározás végighalad a leendő útvonalon, minden útválasztó megvizsgálja

Paraméter	Egység
Vezérjeles vödör sebessége	Bájt/s
Vezérjeles vödör mérete	Bájt
Csúcs adatsebesség	Bájt/s
Minimális csomagméret	Bájt
Maximális csomagméret	Bájt

5.32. ábra. *Példa a folyammeghatározásra*

azt, és igénye szerint módosítja az egyes paramétereket. A módosítások nem növelhetik a folyamot, csak csökkenthetik (például kisebb adatsebességet megadhatnak, nagyobbat nem). Az útvonal végére érve a paraméterek elnyerik végleges értéküket.

A folyammeghatározás lehetséges tartalmára mutat példát az 5.32. ábra. Ez az RFC 2210-et és az RFC 2211-et veszi alapul, amelyek az integrált szolgáltatásra és a QoS tervezésére vonatkoznak. Ezeket a témákat a következő szakaszban fogjuk megvizsgálni. Itt öt paraméterről van szó. Az első két paramétert, a *vezérjeles vödör sebességét*, és a *vezérjeles vödör méretét* arra használják, hogy egy vezérjeles vödör megadja azt a fenntartott legnagyobb sebességet, amellyel a küldő adhat, hosszú idő átlagában, valamint hogy megadja azt a legnagyobb löketet, amit az adó egy rövid időtartam alatt elküldhet.

A harmadik paraméter, az *adatsebesség csúcsértéke*, a maximális megengedett átviteli sebesség. Az adó sosem lépheti túl ezt az értéket, még rövid löketek esetén sem.

Az utolsó két paraméter a *minimális* és *maximális csomagméretet* határozza meg, beleértve a szállítási és a hálózati réteg fejrészeit is (például TCP és IP). A minimális méret azért fontos, mert a feldolgozás minden csomagnál fix időt vesz igénybe, függetlenül attól, hogy milyen rövid a csomag. Lehet, hogy egy útválasztó képes másodpercenként 10 000 darab 1 KB-os csomagot kezelni, de korántsem biztos, hogy megbirkózik másodpercenként 100 000 darab 50 bájtos csomaggal, hiába felel ez meg kisebb adatsebességnek. A maximális csomagméret a hálózat belső korlátjai miatt fontos, ezeket ugyanis nem lehet túllépni. Ha például az útvonal egy része egy Etherneten keresztül vezet, akkor a maximális csomagméret nem lehet több mint 1500 bájt, függetlenül attól, hogy a hálózat fennmaradó része mennyit képes kezelni.

Érdekes kérdés az, hogy az útválasztó hogyan alakíthatja át a folyammeghatározásokat konkrét erőforrás-foglalások halmazává. Első ránézésre úgy tűnhet, hogy ha egy útválasztó egyik adatkapcsolata például 1 Gb/s-os sebességű és egy átlagos csomag 1000 bites, akkor 1 millió csomagot dolgoz fel másodpercenként. Ez a megfigyelés azonban nem felel meg a valóságnak, mert a terhelés statisztikai ingadozása miatt mindig lesznek tétlen periódusok az adatkapcsolaton. Ha az adatkapcsolatoknak a kapacitás összes bitjére szüksége van a munka elvégzéséhez, akkor néhány bites kiesés miatt olyan hátrányba kerül, amit már soha nem fog tudni behozni.

Sőt még a kevéssel az elméleti kapacitás alatt maradó terhelés esetén is sorok alakulhatnak ki, és késleltetés léphet fel. Tekintsünk egy olyan helyzetet, ahol a csomagok véletlenszerű időközönként, átlagosan λ csomag/s sebességgel érkeznek be. A csomagok

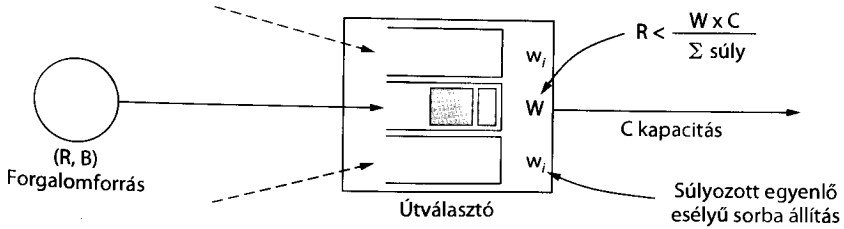
hossza véletlenszerű, és az adatkapcsolaton az átlagos kiszolgálási sebesség μ csomag/s. Ha feltesszük, hogy mind a beérkezés, mind a kiszolgálás Poisson-eloszlású (amelyet M/M/1 sorbanállási rendszernek hívunk, ahol „M" Markovot jelent, azaz Poisson), akkor a sorbanállási elmélet eszközeivel bizonyítható, hogy egy csomag által elszenvedett T késleltetés átlagos értéke

$$T = \frac{1}{\mu} \times \frac{1}{1-\lambda/\mu} = \frac{1}{\mu} \times \frac{1}{1-\rho}$$

ahol $\rho = \lambda/\mu$ a processzor kihasználtsága. Az első tényező, $1/\mu$ azt mutatja meg, hogy mennyi lenne a kiszolgálási idő versengés nélkül. A második tényező a többi folyammal való versengés okozta lassulást jelképezi. Ha például $\lambda = 950\,000$ csomag/s és $\mu = 1\,000\,000$ csomag/s, akkor $\rho = 0{,}95$ és az egyes csomagok által elszenvedett átlagos késleltetés 20 μs lesz 1 μs helyett. Ebben benn van a sorbaállás és a kiszolgálás ideje is, amint az alacsony terhelés esetén látszik ($\lambda/\mu \approx 0$). Ha a folyam útvonalán mondjuk 30 útválasztó van, akkor egyedül a sorbaállásból fakadó késleltetés is már 600 μs lesz.

Parekh és Gallagher [1993, 1994] egy módszert ad, amely a folyammeghatározások, illetve a sávszélességre és késleltetésre vonatkozó teljesítménygaranciáknak megfelelő útválasztó-erőforrások viszonyát mutatja ki. Ez az (R, B) vezérjeles vödör által formált forgalomforrásokra és az útválasztókbeli WFQ-ra épül. Minden folyam elég nagy W WFQ-súlyt kap az R vezérjeles vödör kapacitás elvezetéséhez, ahogy az az 5.33. ábrán látható. Ha például a folyam 1 Mb/s-os sebességű és az útválasztó és a kimeneti adatkapcsolat kapacitása 1 Gb/s, akkor a folyam súlyának a kimeneti vonalhoz tartozó útválasztón nagyobbnak kell lennie, mint az összes folyam összsúlyának 1/1000-e. Ez a folyam számára garantál egy minimális sávszélességet. Ha nem adható elég nagy sebesség, akkor a folyam nem engedhető meg.

A folyam által tapasztalt legnagyobb sorbanállási késleltetés a vezérjeles vödör löketméretének függvénye. Tekintsünk meg két szélsőséges esetet. Ha a forgalom egyenletes, nincs semmilyen löket, akkor a csomagok az útválasztótól a megérkezésük sebességével mennek tovább. Nincs sorbanállási késleltetés (figyelmen kívül marad a csomagba rendezési hatás). Másrészről, ha a forgalom löketekben érkezik, akkor maximálisan B méretű löket érkezhet egyszerre az útválasztóra. Ebben az esetben a D maximális sorbanállási késleltetés lesz a löket elvezetésének ideje a garantált sávszélességen, vagy B/R (újra figyelmen kívül hagyva a csomagba rendezési hatást). Ha ez a késleltetés túl nagy, akkor a folyamnak további sávszélességet kell kérni a hálózattól.



5.33. ábra. *Sávszélesség és késleltetési garanciák vezérjeles vödrökkel és WFQ-val*

Ezek a garanciák szigorúak. A vezérjeles vödrök összefogták a forrás löketszerű adását, az egyenlő esélyű sorba állítás pedig elkülöníti a különböző folyamok számára biztosított sávszélességet. Ez azt jelenti, hogy a folyam megfelel a sávszélességi és késleltetési garanciáknak attól függetlenül, hogy más versengő folyamok hogyan viselkednek az útválasztón. Ezek a folyamok még úgy sem képesek meghiúsítani a garanciát, hogy összegyűjtik és egyszerre küldik a forgalmat.

Továbbá, az eredmény érvényes egy olyan útvonalra, amely tetszőleges hálózati topológiában elhelyezkedő több útválasztón keresztül halad. Minden folyam minimális sávszélességet kap, mivel a sávszélesség minden útválasztón garantált. Annak oka, hogy az egyes folyamok késleltetése miért maximális, már bonyolultabb kérdés. A legrosszabb esetben, amikor a forgalom lökete eléri az első útválasztót, és más folyamok forgalmával versenyez, akkor ennek a késleltetése akár a legnagyobb D késleltetés is lehet. Ez a késleltetés azonban kisimítja a löketet. Viszont ez azt jelenti, hogy a löket nem okoz újabb sorbanállási késleltetést a további útválasztókon. A teljes sorbanállási késleltetés maximum D lesz.

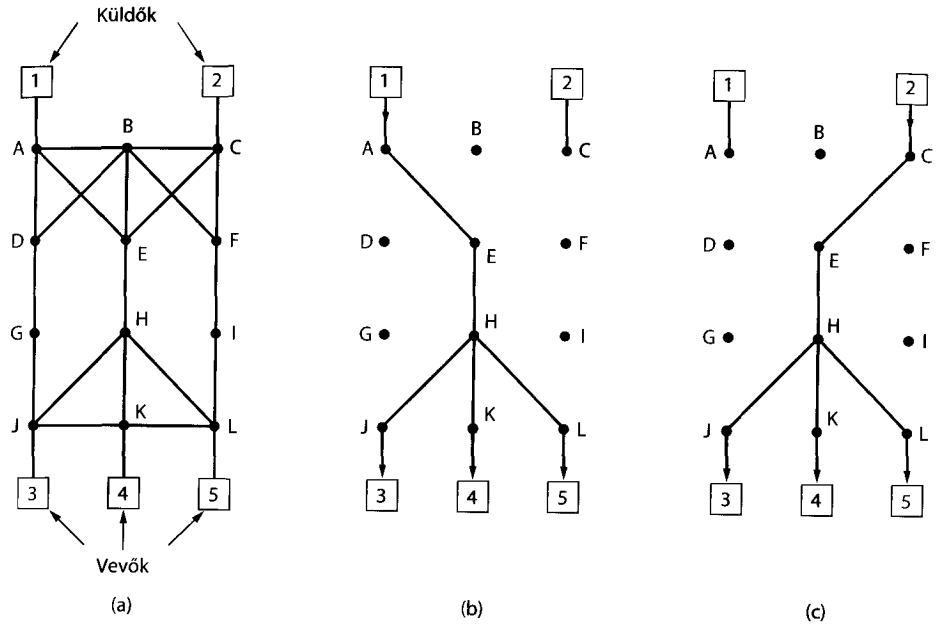
5.4.5. Integrált szolgáltatások

1995 és 1997 között az IETF sok energiát fektetett a multimédia folyamszerű átvitelét lehetővé tevő (streaming multimédia) architektúra kidolgozásába. A munka eredménye több mint két tucat RFC lett, az RFC 2205-2012-vel kezdve. E művek az **integrált szolgáltatások (integrated services, IS)** nevet viselik. Az elgondolások egyaránt irányulnak az egyesküldéses és a többesküldéses alkalmazásokra. Az előbbire példa egy szóló felhasználó, aki egy videoklipet folyamszerűen tölt le egy híroldalról; az utóbbira pedig a digitális televízióállomások egy csoportja, amelyek egy IP-csomagokból álló folyammal közvetítik műsorukat a különböző helyeken tartózkodó nézőknek. Az alábbiakban a többesküldésre fogunk koncentrálni, mivel az egyesküldés is ennek egy speciális esete.

A csoportok tagjai számos többesküldéses alkalmazásban dinamikusan megváltoztathatják a tagságukat, például bekapcsolódhatnak egy videokonferenciába, majd ha megunták, átkapcsolhatnak egy szappanoperára vagy a sportcsatornára. Ilyen feltételek mellett nem működik jól az a megoldás, hogy a feladók előre lefoglalják a sávszélességet, mert ehhez a közönségük minden be- és kilépését nyomon kéne követniük. Egy olyan rendszerben pedig, amelyet arra terveztek, hogy több millió nézőnek szolgáltasson televíziós közvetítést, ez egyáltalán nem is működne.

RSVP – erőforrás-foglalási protokoll

Az integrált szolgáltatások architektúrájának fő protokollja, amely a hálózat felhasználói számára látható, az **RSVP (Resource reSerVation Protocol – erőforrás-foglalási protokoll)**. Leírását az RFC 2205-2210 tartalmazza. Ezt a protokollt a foglalásokhoz használják; az adatok elküldését más protokollok végzik. Az RSVP lehetővé teszi, hogy több adó adjon vevők több csoportjának, továbbá hogy az egyes vevők szabadon választhassanak a csatornák között. A protokoll ezenkívül optimalizálja a sávszélesség-felhasználást, miközben kiküszöböli a torlódásokat is.



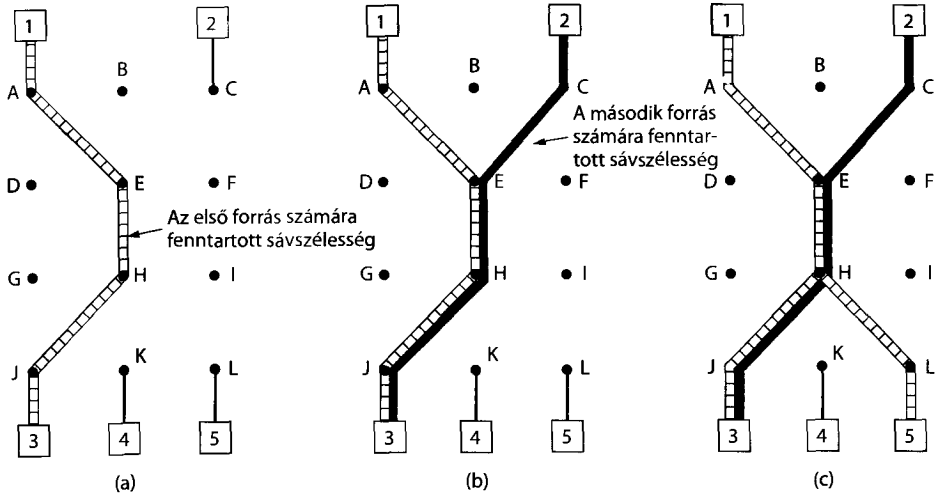
5.34. ábra. *(a) Egy hálózat. (b) A többesküldéses feszítőfa az 1. hoszt számára. (c) A többesküldéses feszítőfa a 2. hoszt számára*

A legegyszerűbb formájában a protokoll a feszítőfát használó többesküldéses útválasztást használja, ahogy azt korábban leírtuk. Minden csoporthoz hozzárendelünk egy csoportcímet. Hogy egy csoportnak küldjön, az adó a csoport címét teszi annak csomagjaiba. A szokásos többesküldéses útválasztás ezután felépít egy feszítőfát, amely a csoport minden tagját lefedi. Az útválasztó algoritmus nem az RSVP része. A normál többesküldéstől való egyetlen eltérés az, hogy a csoportnak egy kevés kiegészítő információt kell periodikusan elküldeni, hogy a fa menti útválasztókat utasítsuk bizonyos, a memóriájukban levő adatstruktúrák karbantartására.

Példaként tekintsük az 5.34.(a). ábra hálózatát. Az 1. és 2. hoszt többesküldéses adó, a 3., 4. és 5. hoszt többesküldéses vevő. Ebben a példában az adók és a vevők szétválnak, általában azonban a két halmaz átlapolódhat. A többesküldéses fa az 1. hosztra az 5.34.(b) ábrán, a 2. hosztra az 5.34.(c) ábrán látható.

Hogy jobb vételt érjen el és kiküszöbölje a torlódást, az egy csoportban levő vevők közül bármelyik küldhet egy foglalási üzenetet a fán felfelé az adóig. Az üzenetet a korábban tárgyalt visszairányú továbbítás algoritmus segítségével terjesztik. Az útválasztó minden lépésnél észreveszi a foglalást, és lefoglalja a szükséges sávszélességet. Ha nem áll rendelkezésre elegendő sávszélesség, sikertelenséget jelez vissza. Amikorra az üzenet visszajut a forrásig, már le lesz foglalva a sávszélesség a feszítőfa mentén, az adótól a foglalást kérő vevőig.

Ilyen foglalásra látható példa az 5.35.(a) ábrán. Itt a 3. hoszt egy csatornát igényelt az 1. hoszthoz. Ha ez már felépült, a csomagok torlódás nélkül folyhatnak az 1.-től a 3.-ba. Most vegyük szemügyre, mi történik, ha a 3. hoszt legközelebb a másik adóhoz, a 2.



5.35. ábra. *(a) A 3. hoszt egy csatornát igényel az 1. hoszthoz. (b) Ezután a 3. hoszt egy második csatornát igényel a 2. hoszthoz. (c) Az 5. hoszt egy csatornát igényel az 1. hoszthoz*

hoszthoz foglal csatornát, hogy a felhasználó egyszerre két tv-műsort tudjon nézni. Egy másik út is fenn lesz tartva, ahogy az 5.35.(b) ábra mutatja. Vegyük észre, hogy két külön csatornára van szükség a 3. hoszttól az *E* útválasztóig, mert két független adatfolyam kerül továbbításra.

Végül az 5.35.(c) ábrán az 5. hoszt elhatározza, hogy az 1. hoszt által adott programot akarja nézni, és szintén foglalást végez. Először, saját célra sávszélességet foglal le a *H* útválasztóig. Viszont ez az útválasztó látja, hogy már kapja a táplálást az 1. hoszttól, így ha a szükséges sávszélesség már le van foglalva, nem kell többet lefoglalnia. Vegyük észre, hogy a 3. és 5. hosztok különböző méretű sávszélességet kérhettek (például a 3. hosztnak kis képernyője van, és csak kis felbontású információt kíván), így a lefoglalt kapacitásnak olyan nagynak kell lennie, hogy kielégítse a legmohóbb vevőt is.

Amikor egy vevő foglalást végez, (opcionálisan) megnevezhet egy vagy több forrást, amelyekből venni akar. Azt is megmondhatja, hogy ezek a választások rögzítettek a foglalás időtartama alatt, vagy pedig a vevő meg akarja tartani a forrásváltoztatás lehetőségét későbbre. Az útválasztók ezt az információt a sávszélesség-tervezés optimalizálásához használják fel, nevezetesen, két vevőt csak akkor állítanak be közös út használatára, ha mindkettő beleegyezett, hogy később nem változtat forrást.

Ennek a stratégiának az oka a teljesen dinamikus esetben az, hogy a lefoglalt sávszélesség el van különítve a forrás választásától. Ha egy vevő már lefoglalt sávszélességet, átkapcsolhat egy másik forrásra, és megtarthatja a létező út azon részét, amely érvényes az új forrásra is. Ha mondjuk a 2. hoszt több videofolyamot is továbbít valós időben, például egy több csatornával rendelkező tv-adó, a 3. hoszt tetszése szerint kapcsolgathat köztük a foglalás változtatása nélkül: az útválasztók nem törődnek vele, milyen programot néz a vevő.

5.4.6. Differenciált szolgáltatások

A folyamalapú algoritmusoknak megvan az a képességük, hogy jó szolgáltatásminőséget tudnak biztosítani egy vagy több folyam számára, mivel bármilyen igényelt erőforrást le tudnak foglalni az út mentén. Megvannak azonban ennek a maguk hátrányai is. Minden egyes folyam kialakítása előkészületeket igényel, ez pedig nehezen kézben tartható, ha több ezer vagy millió folyamról van szó. Ráadásul az egyes folyamok állapotát az útválasztókban kezelik, ami sebezhetővé teszi azokat az útválasztók összeomlása esetén. Végül jelentős változtatásokat követelnek meg az útválasztók szoftverében is, a folyamok felépítése pedig bonyolult üzenetváltásokkal jár együtt az útválasztók között. Ebből fakadóan, mialatt az integrált szolgáltatásokkal kapcsolatos munkák előrehaladnak, nagyon kevés integrált szolgáltatásokkal kapcsolatos megvalósítás vagy bármi ehhez hasonló létezik.

Ezen okok miatt az IETF is egyszerűbb megoldást gondolt ki a szolgáltatásminőség megvalósítására, olyat, amely minden útválasztóban javarészt helyileg implementálható, az előkészületek és az egész útvonal bevonása nélkül. Ezt a megoldást **osztályalapú (class-based) szolgáltatásminőségnek** nevezik (szemben az eddig látott folyamalapúval). Az IETF egy architektúrát is szabványosított a számára, **differenciált szolgáltatások (differentiated services, DS)** néven, melyet az RFC 2474, RFC 2475 és számos más RFC ír le. A következőkben itt is bemutatásra kerül.

Differenciált szolgáltatásokat az egy adminisztratív körzetbe (például egy internetszolgáltató vagy telefontársaság) tartozó útválasztók egy halmaza kínálhat. Az adminisztráció definiálja a szolgáltatási osztályok egy halmazát a nekik megfelelő továbbítási szabályokkal együtt. Ha egy ügyfél előfizet egy differenciált szolgáltatásra, akkor a körzetbe belépő csomagjait megjelölik azzal az osztállyal, amelyikhez tartoznak. Ezt az információt az IPv4- és IPv6-csomagok *Differenciált szolgáltatások (Differentiated services)* mezője hordozza (erről az 5.6. szakaszban lesz szó). Az osztályokat **ugrásonkénti viselkedésekként (per hop behavior)** adják meg, mivel ezek megfelelnek annak a kezelésnek, ahogy a csomagot az egyes útválasztók venni fogják, ez nem hálózati garancia. Jobb szolgáltatások biztosíthatók az ugrásonkénti viselkedésű csomagok számára (például prémiumszolgáltatás), mint mások számára (például normál szolgáltatás). Egy adott osztályhoz tartozó forgalomtól megkövetelhető, hogy megfeleljen egy meghatározott mintának – ez lehet például egy adott sebességgel csöpögő lyukas vödör. Jó üzleti érzékkel megáldott szolgáltató külön díjat számolhat fel minden leszállított prémiumcsomag után, vagy engedélyezhet legfeljebb havi *N* darab prémiumcsomagot rögzített havi különdíj ellenében. Vegyük észre, hogy ez a séma – szemben az integrált szolgáltatásokkal – nem igényel előzetes beállítást, sem erőforrás-lefoglalást, sem időigényes végpontok közti tárgyalásokat külön-külön minden egyes folyamhoz. Ezért tehát a differenciált szolgáltatások viszonylag egyszerűen megvalósíthatók.

Osztályalapú szolgáltatásokkal más területeken is találkozhatunk. A csomagküldő szolgáltatások gyakran kínálnak éjszakai, kétnapos vagy háromnapos kézbesítést. A légitársaságok első osztályú, üzleti osztályú és turista osztályú szolgáltatásokat nyújtanak. Gyakran a távolsági vonatokon is többféle szolgáltatási osztály van, sőt még a párizsi metrónak is két osztálya van. A csomagok esetében az osztályok többek közt a késleltetés, a dzsitter és a torlódás esetén történő eldobás valószínűsége szerint különbözhetnek (ez a hosszabb Ethernet-keretekre valószínűleg nem érvényes).

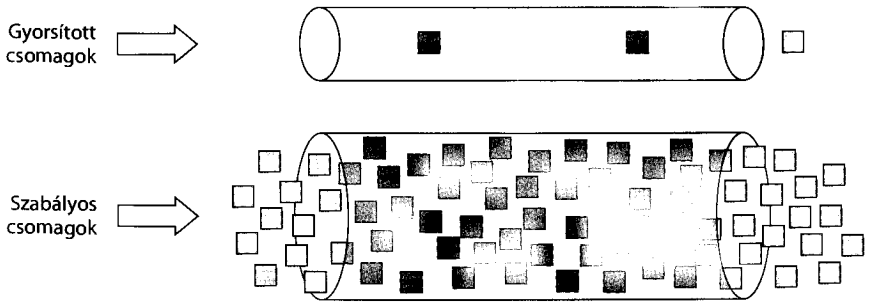
Hogy jobban meg tudjuk különböztetni a folyamalapú és az osztályalapú szolgáltatásminőséget, vegyük szemügyre az internettelefon példáját! Folyamalapú sémát használva mindegyik hívás megkapja a maga erőforrásait és garanciáit. Osztályalapú sémával az összes hívás együtt kapja meg az adott osztály számára lefoglalt erőforrásokat. Ezeket az erőforrásokat nem vehetik el a webböngésző osztályhoz vagy más osztályokhoz tartozó csomagok, de egyetlen telefonhívásnak sem lesznek fenntartva saját használatra erőforrások.

Gyorsított továbbítás

A szolgáltatási osztályok megválasztása a hálózatüzemeltetők dolga, de mivel a csomagokat gyakran különböző szolgáltatókhoz tartozó hálózatok között továbbítják, az IETF néhány hálózatfüggetlen szolgáltatási osztályt határozott meg. A legegyszerűbb ilyen osztály a **gyorsított továbbítás (expedited forwarding)**, kezdjük most ezzel. Az osztályt az RFC 3246 írja le.

A gyorsított továbbítás mögött megbúvó ötlet nagyon egyszerű. Kétfajta szolgáltatási osztály áll rendelkezésre: szabályos és gyorsított. A forgalom túlnyomó része várhatóan szabályos lesz, de a csomagok egy kis hányadát gyorsítjuk. Az ilyen gyorsított csomagoknak elvileg úgy kell keresztülhaladniuk a hálózaton, mintha más csomag nem is lenne ott jelen. Így a csomagok veszteség nélkül haladnak át, kis késleltetéssel, kis dzsitterrel – éppen úgy, ahogy a VoIP kívánja. Ezt a „kétcsöves" rendszert ábrázolja az 5.36. ábra. Vegyük észre, hogy továbbra is csak egy fizikai vonalunk van. Az ábrán látható két logikai csővezeték azt a módot ábrázolja, ahogy sávszélesség lefoglalható a különböző szolgáltatási osztályok számára, és nem egy másik fizikai vonalat.

Egy lehetséges megvalósítási stratégia a következő. A csomagokat gyorsítottként vagy szabályosként osztályozzák, és ennek megfelelően megjelölik. Ez a lépés a küldő hoszton vagy a bemeneti (első) útválasztón hajtható végre. A küldő hoszton való osztályozás előnye az, hogy több információ áll rendelkezésre arról, hogy melyik csomag melyik folyamhoz tartozik. Ezt a feladatot a hálózatkezelési szoftver vagy az operációs rendszer hajthatja végre, hogy a meglévő alkalmazásokat ne kelljen módosítani. Például VoIP-csomagok esetén általános, hogy a hosztok jelölik meg a csomagokat gyorsított szolgáltatásra. Ha a csomagok vállalati hálózaton vagy a gyorsított szolgáltatást támogató internetszolgáltató hálózatán mennek keresztül, akkor előnyben részesülnek. Akkor sincs baj, ha a hálózat nem támogatja a gyorsított szolgáltatást.



5.36. ábra. *A gyorsított csomagok forgalommentesnek látják a hálózatot*

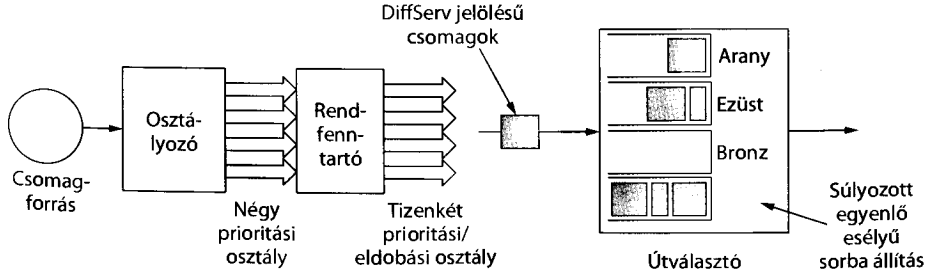
Természetesen, ha a jelölést a hoszt végzi, akkor a bemeneti útválasztó feltehetőleg elvégzi a forgalom rendfenntartását annak biztosítása érdekében, hogy az ügyfelek ne küldjenek több gyorsított forgalmat, mint amiért fizettek. A hálózatban az útválasztók két kimeneti sort tarthatnak fenn minden kimenő vonalhoz: egyet a szabályos, egyet a gyorsított csomagokhoz. Amikor egy csomag megérkezik, bekerül a megfelelő sorba. A gyorsított sor prioritást élvez a szabályossal szemben, például prioritásütemező segítségével. Ily módon a gyorsított csomagok terhelés nélküli hálózatot látnak még abban az esetben is, ha valójában a szabályos forgalom nagy terhelést ró a hálózatra.

Biztosított továbbítás

A szolgáltatási osztály az előzőnél némileg kidolgozottabb kezelését nyújtja a **biztosított továbbítás (assured forwarding)** sémája, melyet az RFC 2597 ír le. A séma négy prioritási osztályt határoz meg; itt minden osztályhoz saját erőforrások tartoznak. Ezenkívül a torlódásba került csomagok eldobására is definiálnak három különböző valószínűséget (kis, közepes és nagy). Mindent összevetve, a két tényező együtt 12 szolgáltatási osztályt határoz meg.

Az 5.37. ábra a csomagok biztosított továbbításának egy lehetséges módját mutatja. Első lépésben minden csomagot a négy prioritási osztály közül az egyikbe sorolnak. Ezt megteheti a feladó hoszt (ahogy az ábra is mutatja) vagy az első (ún. beléptető) útválasztó, és a magasabb prioritású csomagok sebességét az operátor a szolgáltatási ajánlat részeként korlátozhatja.

A következő lépés az egyes csomagok eldobási osztályának meghatározása. Ez úgy történik, hogy minden egyes prioritási osztály csomagjait egy forgalmi rendfenntartón küldik keresztül, mint amilyen például a vezérjeles vödör. A rendfenntartó az összes forgalmat átengedi, de a kis löketekbe beleillő csomagokat alacsony eldobási osztályúnak, a kis löketeket meghaladó csomagokat közepes eldobási osztályúnak, a nagy löketeket meghaladókat pedig magas eldobási osztályúként azonosítja. A prioritás és az eldobási osztály kombinációja ezután kódolásra kerül minden csomagban.



5.37. ábra. *Az adatáramlás egy lehetséges megvalósítása a biztosított továbbításnál*

Végül a csomagokat a hálózaton lévő útválasztók feldolgozzák egy csomagütemezővel, amely megkülönbözteti a különböző osztályokat. Általános választás az, hogy a súlyozott egyenlő esélyű sorba állítást használják a négy prioritási osztályhoz úgy, hogy a magasabb osztályok nagyobb súlyt kapnak. Ily módon a magasabb osztályok megkapják a sávszélesség nagy részét, de az alacsonyabb osztályok sem lesznek teljesen kiéheztetve, számukra is jut némi sávszélesség. Ha például a súlyok osztályonként megduplázódnak úgy, hogy egy magasabb szintű osztály kétszer akkora súly kap, mint az alatta lévő osztály, akkor a magasabb osztály kétszer akkora sávszélességet kap. Egy prioritási osztályon belül a magasabb eldobási osztállyal minősített csomagok preferenciabeállítások alapján dobhatók el egy olyan algoritmus futtatásával, mint amilyen például az 5.3.5. szakaszban látható **RED (Random Early Detection – véletlen korai detektálás)**. A RED elkezdi eldobni a csomagokat, ahogy a torlódás kialakul, de mielőtt az útválasztó kifogyna a pufferterületből. Ebben a fázisban van még pufferterület, amelyre az alacsony eldobási osztályú csomagokat elhelyezi, miközben a magas eldobási osztályú csomagokat eldobja.

5.5. Hálózatok összekapcsolása

Egészen eddig hallgatólagosan csupán egyetlen homogén hálózatot feltételeztünk, amelyben minden gép minden egyes rétegben ugyanazt a protokollt használja. Sajnos azonban ez a feltételezés túlzottan optimista. Sok különböző hálózat létezik, beleértve a PAN-okat, LAN-okat, MAN-okat és WAN-okat is. Foglalkoztunk az Ethernettel, a kábeltévés internettel, valamint rögzített és mobiltelefon-hálózatokkal, a 802.11-gyel, 802.16-tal, és számos egyéb hálózattal. Minden rétegben számos protokoll használata elterjedt ezekben a hálózatokban. A következő szakaszokban gondosan szemügyre vesszük azokat a kérdéseket, amelyek akkor merülnek fel, ha kettő vagy több hálózatot kapcsolunk össze **internetwork** vagy csak egyszerűen **internet** kialakítása érdekében.

Sokkal egyszerűbb lenne a hálózatokat csatlakoztatni, ha mindenki ugyanazt a hálózatkezelési technikát használná. Gyakran az a helyzet, hogy van egy domináns hálózattípus, mint például az Ethernet. Egyesek szerint a sokféle technika meg fog szűnni, amint mindenki rájön, hogy milyen csodálatos az [ide mindenki a saját kedvenc hálózatát írhatja]. Erre azonban nem számíthatunk. A történelem azt mutatja, hogy ez csak vágyálom. A különböző hálózatok különböző problémákkal küszködnek, például az Ethernet és a műholdas hálózatok valószínűleg mindig különbözni fognak. A meglévő rendszerek újrafelhasználása, mint például adathálózatok futtatása kábeltévén, telefonhálózaton és erősáramú vezetékeken olyan megszorításokkal jár, amelyek azt okozzák, hogy a hálózatok képességei különbözőek lesznek. A heterogenitás itt van és itt is marad.

Ha mindig lesznek különböző hálózatok, akkor egyszerűbb lenne, ha nem kéne összekapcsolnunk azokat. Ez azonban szintén valószínűtlen. Bob Metcalfe kifejtette, hogy az N csomóponttal megvalósított hálózat értéke egyenlő a csomópontok között létesíthető kapcsolatok számával, vagyis N^2 [Gilder, 1993]. Ez azt jelenti, hogy a nagy hálózatok értékesebbek, mint a kicsik, mivel számos összeköttetést tesznek lehetővé, így mindig lesz ösztönzés a kisebb hálózatok egyesítésére.

Az internet az elsődleges példa az összekapcsolásra. A hálózatok összekapcsolásának célja, hogy a felhasználók kommunikálni tudjanak egymással. Amikor egy internetszolgáltatónak (ISP) fizetünk a szolgáltatásért, lehet, hogy a fizetség a használt vonal sávszélességétől függ, de valójában az internetre csatlakozó más hosztokkal való csomagcsere lehetőségéért fogunk fizetni. Az internet nem lenne nagyon népszerű, ha csak az ugyanabban a városban lévő hosztoknak lehetne csomagot küldeni.

Mivel a hálózatok gyakran fontos dolgokban különböznek, a csomagok hálózatok közötti küldése nem mindig egyszerű. Foglalkoznunk kell a heterogenitás problémájával, és a skálázási problémákkal, mivel az eredményül kapott internet nagyon nagyra nő. Kezdetnek megnézzük, hogy a hálózatok miben különböznek, hogy lássuk, mivel állunk szemben. Ezután megnézzük azt a megoldást, amelyet az IP (Internet Protocol) – az internet hálózati réteg protokollja – olyan sikeresen használ, beleértve a hálózatokon keresztül alkalmazott alagúttechnikát, az összekapcsolt hálózatokban használt útválasztó technikát, valamint a csomagtördelési technikákat is.

5.5.1. Miben különböznek a hálózatok?

A hálózatok sok mindenben különbözhetnek egymástól. A különbségek egy része a fizikai és az adatkapcsolati rétegben van, mint például az eltérő modulációs eljárások vagy keretformátumok. Ezekkel most nem foglalkozunk. Ehelyett az 5.38. ábrán felsorolunk néhány olyan különbséget, melyek a hálózati rétegben fordulhatnak elő. Az teszi a hálózatok összekapcsolását nehezebbé, szemben az egyetlen hálózaton belüli működéssel, hogy ezek a különbségek el vannak rejtve, nem láthatók.

Amikor egy adott hálózaton lévő forrásgép által küldött csomagoknak legalább egy idegen hálózaton át kell haladniuk, mielőtt elérnék a célhálózatot, számos probléma

Tétel	Néhány lehetőség
Felkínált szolgáltatás	Összeköttetés nélküli vagy összeköttetés-alapú
Címzés	Különböző méretek, egyszintű vagy hierarchikus
Adatszórás	Van vagy nincs (ugyanez többesküldésre is)
Csomagméret	Minden hálózat megszab egy legnagyobb értéket
Sorrendiség	Sorrendhelyes és nem sorrendhelyes kézbesítés
Szolgáltatás minősége	Van vagy nincs; sokfajta létezik
Megbízhatóság	Különböző szintű csomagvesztés
Biztonság	Bizalmassági szabályok, titkosítás stb.
Paraméterek	Eltérő időzítések, folyam-meghatározások stb.
Számlázás	Összeköttetési idő alapján, csomagok száma alapján, bájtok száma alapján vagy sehogy

5.38. ábra. *Néhány abból a sok dologból, amelyben a hálózatok eltérhetnek egymástól*

jelentkezhet a hálózatok közti interfészeken. Először is a forrásnak meg kell tudnia címezni a célt. Mit teszünk, ha a forrás egy Ethernet-hálózaton van, a cél pedig egy WiMAX-hálózaton? Feltételezve, hogy egyáltalán meg tudunk adni WiMAX-címet egy Ethernet-hálózatról, a csomagoknak egy összeköttetés nélküli hálózatról kell átmenniük egy összeköttetés-alapú hálózatra. Ehhez szükség lehet rövid időre egy új összeköttetés kialakítására, amely késleltetést involvál, és sok többletterhelést, ha az összeköttetést nem használják fel több csomaghoz.

Több speciális különbséghez kell alkalmazkodni. Hogyan továbbítana egy többesküldéses csomagot néhány tagot tartalmazó csoporthoz egy olyan hálózaton keresztül, amelyik nem támogatja a többesküldést? A különböző hálózatok által használt különböző maximális csomagméretek szintén jelentős problémát okozhatnak. Hogyan továbbítana egy 8000 bájtos csomagot egy olyan hálózaton keresztül, amelynek maximális mérete 1500 bájt? Ha egy összeköttetés-alapú hálózat csomagjait összeköttetés nélküli hálózat viszi át, akkor a csomagok sorrendje megváltozhat. Ez olyasvalami, amire az adó valószínűleg nem számított és (kellemetlen) meglepetésként érheti a vevőt is.

Ezek a különbségek némi erőfeszítéssel elfedhetők. Például, két hálózatot összekapcsoló átjáró különálló csomagokat állíthat elő minden célhoz a többesküldés jobb hálózati támogatása helyett. Elképzelhető, hogy a nagy csomagot feldarabolja és darabokban küldi el, majd újra egyesítésre kerül. A vevők pufferelhetik a csomagokat és sorrendben kézbesíthetik azokat

A hálózatok olyanok dolgokban is különbözhetnek, amelyeket nehéz kiegyenlíteni. A legegyértelműbb példa a szolgáltatás minősége. Ha az egyik hálózatban jó a szolgáltatás minősége, a másik hálózat pedig nem nyújt garantált minőségű szolgáltatást, akkor nem lehet sávszélesség- és késleltetési garanciát vállalni a valós idejű végpontok közötti forgalomra. Valójában ezek csak akkor érhetők el, ha a nem garantált minőségű hálózat kis kihasználtsággal működik, vagy alig használják, ami a legtöbb internetszolgáltatónak nem célja. A biztonsági mechanizmusok problémásak, de legalább titkosítás alkalmazható a bizalmasság és adatintegritás biztosítása érdekében azokon a hálózatokon, amelyek még nem tartalmaznak ilyet. Végül a számlázási különbségek meglepő számlákat eredményezhetnek, ha a normál használat hirtelen megdrágul, amint azt a mobilinternetes előfizetéssel szerződött mobiltelefon-használók (keserűen) megtapasztalhatták.

5.5.2. Hogyan lehet összekapcsolni a hálózatokat?

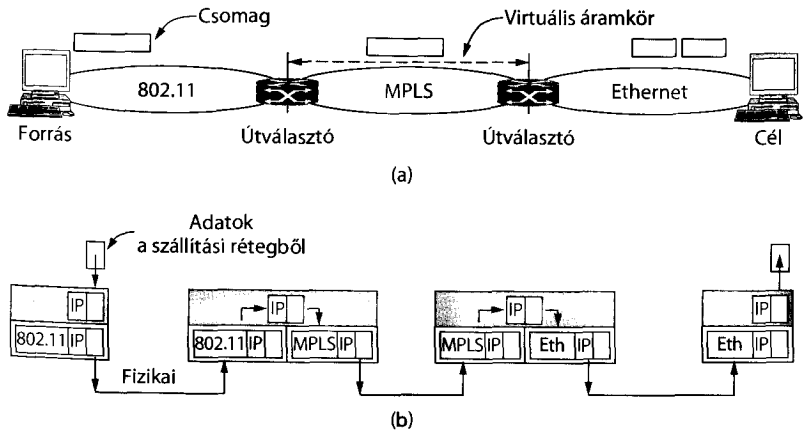
A különböző hálózatok összekapcsolására két alapvető lehetőség áll rendelkezésre: létrehozhatunk olyan eszközöket, amelyek lefordítják vagy átalakítják a csomagokat, amikor az egyik hálózatról a másikra kerülnek, vagy jó számítástechnikusként megpróbálhatjuk megoldani a problémát egy közvetett átjárást biztosító réteg hozzáadásával és egy, a különböző hálózatok fölötti közös réteg kialakításával. Az eszközök mindkét esetben a hálózatok közötti határra kerülnek.

Cerf és Kahn [1974] még a kezdet kezdetén egy általános réteg kialakítása mellett érvelt, hogy az a meglévő hálózatok közötti különbségeket elrejtse. Ez a megközelítés nagyon sikeresnek bizonyult, és az általuk ajánlott réteget valójában szétválasztották TCP- és IP-protokollra. Majdnem négy évtizeddel később az IP képezi a modern internet alapját. Ezért Cerf és Kahn megkapta a 2004-es Turing-díjat, amelyet informálisan az informatika Nobel-díjának neveznek. Az IP egy univerzális csomagformátumot biztosít, amelyet minden útválasztó felismer, és amely majdnem az összes hálózaton átküldhető. Az IP elérését kiterjesztették a számítógép-hálózatokról a telefonhálózatokra. Szenzorhálózatokon és egyéb apró eszközön is fut, amelyeket régen túl erőforrás-szegénynek gondoltak ahhoz, hogy támogassák az IP-t.

Számos különböző eszközt mutatunk be a következőkben, amelyek hálózatokat kapcsolnak össze, mint például az ismétlők, elosztók (hubok), kapcsolók, hidak, útválasztók és átjárók. Az ismétlők és az elosztók csak a biteket mozgatják a vezetékek között. Ezek többnyire analóg eszközök, és semmit nem tudnak a magasabb szintű protokollokról. A hidak és kapcsolók az adatkapcsolati rétegben működnek. Segítségükkel kialakíthatók hálózatok, csak egy kis protokollfordítást kell végezni, például a 10, 100 és 1000 Mb/s-os Ethernet-kapcsolók közötti folyamatban. Ebben a fejezetben a hálózati rétegben működő csatlakoztató eszközökre koncentrálunk, név szerint az útválasztókra. A magasabb szintű csatlakoztató eszközökkel, az átjárókkal később foglalkozunk.

Először tekintsük meg, hogy magasabb szinten hogyan köthetők össze a különböző hálózatok egy közös hálózati réteggel. Egy 802.11-es, MPLS- és Ethernet-hálózatból álló összekapcsolt hálózat látható az 5.39.(a) ábrán. Tételezzük fel, hogy a 802.11-es hálózaton lévő forrásgép csomagot kíván küldeni az Ethernet-hálózaton lévő célgépnek. Mivel ezek a technikák különböznek, és egy harmadik típusú hálózat (MPLS) választja el őket, a hálózatok határain további feldolgozás szükséges.

Mivel általában a különböző hálózatok címzési formája is különbözhet, a csomag tartalmaz egy hálózati rétegbeli címet, amely bármelyik hosztot azonosítani tudja a három hálózaton. Az első határt akkor éri el a csomag, amikor a 802.11 hálózatról átmegy az MPLS-hálózatra. A 802.11 összeköttetés nélküli szolgáltatást biztosít, az MPLS pedig összeköttetés-alapút. Ez azt jelenti, hogy egy virtuális áramkört kell kiépíteni a hálózaton. Ha a csomag végighaladt a virtuális áramkör mentén, akkor eléri az Ethernet-hálózatot. Elképzelhető, hogy ezen a határon a csomag túl nagynak bizonyul a továbbküldés-



5.39. ábra. *(a) Különböző hálózatokon átmenő csomag. (b) Hálózati és adatkapcsolati rétegbeli protokoll szerinti feldolgozás*

hez, mivel a 802.11 nagyobb kereteket tud kezelni, mint az Ethernet. A probléma megoldása érdekében a csomag feldarabolásra kerül, és minden darab külön kerül elküldésre. Amikor a darabok elérik a címzettet, összerakják azokat. Ezután a csomag befejezi útját.

Azt a protokollt, amely ezt az utazást feldolgozza, az 5.39.(b) ábra mutatja. A forrás a szállítási rétegből kap adatokat és csomagot állít elő egy közös hálózati rétegbeli fejrésszel, amely ebben a példában az IP. A hálózati fejrész tartalmazza a végső célcímet, amely meghatározza, hogy a csomagot az első útválasztón keresztül kell küldeni. Így a csomag beágyazódik a 802.11-es keretbe, amelynek célcíme az első útválasztó, majd ehhez továbbítódik. Az útválasztó a csomagot eltávolítja a keret adatmezőjéből, és a 802.11-es keretfejrészt eldobja. Az útválasztó megvizsgálja a csomagban lévő IP-címet és megkeresi azt az útválasztó táblázatban. A cím alapján eldönti, hogy elküldi a csomagot a második útválasztónak. Az útvonal ezen részéhez MPLS virtuális áramkört kell létesíteni a második útválasztóhoz, és a csomagot be kell ágyazni egy MPLS-keretbe, amely átviszi a csomagot ezen a virtuális áramkörön. A virtuális áramkör túlsó végén lévő eszköz eldobja az MPLS-fejrészt, és kiveszi a hálózati címet a következő hálózati rétegbeli ugrás meghatározásához. Minthogy ez a cím magának a címzettnek a címe, ezért nem kell a csomagot továbbküldeni. Mivel a csomag túl hosszú ahhoz, hogy átküldhető legyen az Ethernet-hálózaton, két darabra kell osztani. Mindkét rész bekerül egy-egy Ethernet-keret adatmezőjébe, és továbbítódik a cél Ethernet-címére. A címzett az Ethernet-fejrészt leválasztja minden keretről, és a csomagtartalom újból összeáll. A csomag végül eléri a címzett hálózati eszközt.

Figyeljük meg, hogy az útválasztóval és a kapcsolóval (vagy híddal) megvalósított esetek lényegesen különböznek. Az útválasztó kiveszi a csomagot a keretből és a csomagban lévő hálózati címet használja fel a küldési célcím meghatározására. Kapcsoló (vagy híd) esetén a teljes keretet továbbítja a MAC-cím alapján. A kapcsolóknak nem kell ismerniük a csomagok kapcsolásához használandó hálózati réteg protokollt, az útválasztóknak viszont igen.

Sajnálatos módon a hálózatok összekapcsolása nem olyan egyszerű, mint ahogy hangzik. A hidak bevezetésének az volt a célja, hogy különböző típusú hálózatokat, vagy legalábbis különböző típusú LAN-okat kössenek össze. Ezt úgy érték el, hogy az egyik LAN-ból származó kereteket lefordították a másik LAN-hoz tartozó keretekre. Ez azonban nem működött túl jól, ugyanazon ok miatt, ami miatt a hálózatok összekapcsolása is nehéz: a LAN-ok jellemzői közötti különbségeket nehéz elrejteni, mint például az eltérő maximális csomagméreteket, valamint a prioritási osztállyal rendelkező, illetve nem rendelkező LAN-okat. Manapság a hidakat főként arra használják, hogy azonos típusú hálózatokat kapcsoljanak össze az adatkapcsolati rétegben, és az útválasztókat pedig arra, hogy különböző hálózatokat kapcsoljanak össze a hálózati rétegben.

A hálózatok összekapcsolása nagyon sikeres nagy hálózatok kialakításánál, de csak akkor működik, ha van egy közös hálózati réteg. Az idők során számos hálózati protokoll létezett. Nem egyszerű, hogy mindenki egyetlen formátumot használjon, mivel a vállalatok abban látják a kereskedelmi előnyt, ha saját formátumuk van, amelyet maguk ellenőrizhetnek. Példák az IP mellett, amely most majdnem univerzális hálózati protokoll: az IPX, az SNA és az AppleTalk. Ezek közül a protokollok közül jelenleg egyiket sem használják széles körben, de mindig lesznek más protokollok. A legszemléletesebb példa jelenleg valószínűleg az IPv4 és IPv6. Annak ellenére, hogy ezek az IP változatai,

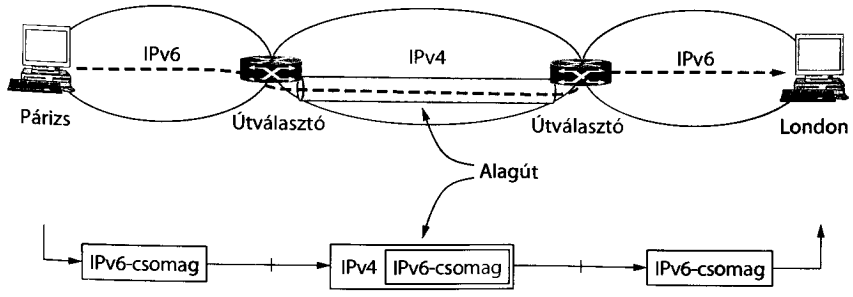
nem kompatibilisek egymással (különben semmi szükség nem lett volna az IPv6 kialakítására).

Az olyan útválasztót, amely több protokollt is képes kezelni, **többprotokollos útválasztónak** (**multiprotocol router**) nevezzük. Az ilyen útválasztónak vagy le kell fordítania a protokollokat, vagy át kell engednie a kapcsolatot egy magasabb protokollrétegnek. Egyik megoldás sem teljesen kielégítő. A magasabb rétegű, mondjuk TCP-alapú összeköttetéshez az összes hálózatnak TCP-t kell használnia (ami nem feltétlenül teljesül). Ez korlátozza a hálózat használatát azokra az alkalmazásokra, amelyek TCP-t használnak (amelyek nem tartalmaznak számos valós idejű alkalmazást).

A másik megoldási lehetőség a csomagok lefordítása a hálózatok között. Hacsak a csomagformátumok nem állnak közeli rokonságban, és használnak azonos információs mezőket, akkor az ilyen átalakítás mindig hiányos lesz, és gyakran kudarcra van ítélve. Az IPv6-cím például 128 bit hosszú. Ez nem fér el egy 32 bites IPv4-címmezőben, bármennyire is próbálja ezt az útválasztó megoldani. Az IPv4 és IPv6 ugyanazon hálózaton való futtatása bizonyítottan a legnagyobb probléma volt az IPv6 bevezetésénél. (Igazság szerint az ügyfeleket nem sikerült arról meggyőzni, hogy miért is szeretnék egyáltalán az IPv6-ot használni.) Nagyobb problémák várhatók az alapjaiban különböző protokollok közötti fordításnál, mint például az összeköttetés nélküli és összeköttetés-alapú hálózati protokollok fordításánál. Ezek miatt a problémák miatt a konverziót ritkán kísérlik meg. Még az IP is csak arra volt jó, hogy közös alapként szolgáljon. Kis igényt támaszt a futtató hálózattal szemben, de ennek eredményeképpen csak garancia nélküli szolgáltatást (best-effort) nyújt.

5.5.3. Alagút típusú átvitel

Két különböző hálózat együttműködését kezelni általános esetben túlságosan bonyolult. Ám van egy elterjedt sajátságos eset, amelynél az együttműködés még különböző hálózati protokollok esetén is megoldható. Ez az az eset, amikor a forrás- és célhosztok azonos típusú hálózatokon vannak, de van közöttük egy másfajta hálózat. Példaként gondoljunk egy nemzetközi bankra, amelynek van egy IPv6-hálózata Párizsban, egy IPv6-hálózata Londonban, és az irodákat az IPv4-es internet köti össze. Ezt a helyzetet szemlélteti az 5.40. ábra.



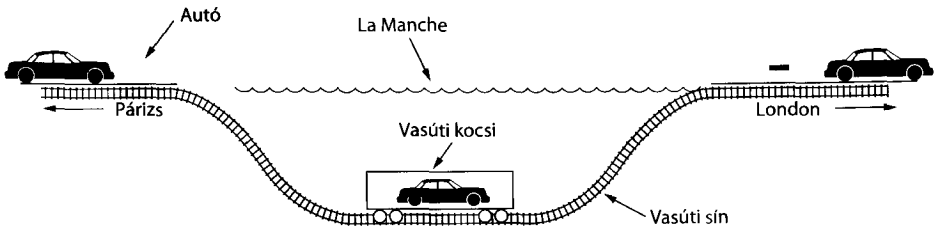
5.40. ábra. *Egy csomag alagút típusú átvitele Párizsból Londonba*

A megoldást erre a problémára egy olyan módszer jelenti, amelyet **alagút típusú átvitelnek (tunneling)** neveznek. Ahhoz, hogy a párizsi irodában lévő hoszt egy IP-csomagot küldjön a londoni irodában lévő hosztnak, a párizsi irodában lévő hoszt összeállít egy csomagot, amely tartalmazza a londoni IPv6-címet, és azt elküldi egy többprotokollos útválasztónak, amely összeköti a párizsi IPv6-hálózatot és az IPv4-internetet. Amikor az útválasztó megkapja az IPv6-csomagot, becsomagolja ezt egy IPv4-fejrésszel, amely a londoni IPv6-hálózathoz csatlakozó többprotokollos útválasztó IPv4-oldalát címezi meg. Így az útválasztó behelyez egy (IPv6) csomagot egy (IPv4) csomagba. Amikor a beágyazott csomag megérkezik, a londoni útválasztó kibontja az eredeti IPv6-csomagot és továbbküldi a címzett hosztnak.

Az IPv4-interneten keresztülhaladó útvonalat tekinthetjük úgy is, mint egy nagy alagutat, amely az egyik többprotokollos útválasztótól a másikig tart. Az IPv6-csomag egyszerűen az alagút egyik végétől a másikig utazik, kényelmesen a kis dobozában. Egyáltalán nem kell foglalkoznia az IPv4-gyel. Ugyanígy a Párizsban és Londonban levő hosztoknak sem. Csak a többprotokollos útválasztóknak kell tudniuk kezelni az IPv4- és IPv6-csomagokat egyaránt. Valójában az egyik többprotokollos útválasztótól a másikig vezető teljes útvonal olyan, mint egyetlen adatkapcsolat feletti ugrás.

Nézzünk egy hasonlatot az alagút típusú átvitel világosabbá tételére: vegyünk egy embert, aki az autójával Párizsból Londonba utazik. Franciaországon belül az autó saját erejéből mozog, de amikor eléri a La Manche csatornát, berakják egy nagy sebességű vonatba és átszállítják Angliába a Csalagúton keresztül (az autóknak nem szabad behajtani a Csalagútba). Valójában az autót mint rakományt szállítják, ahogy ez az 5.41. ábrán látható. A másik oldalon az autót kiengedik az angol utakra és ismét önjáró lesz. A csomagok alagút típusú átvitele egy idegen hálózaton keresztül ugyanígy működik.

Az alagút típusú átvitelt gyakran használják izolált hosztok és hálózatok más hálózatokkal való összekötéséhez. A hálózat lényegében az alaphálózat fölé van terítve, ezért **elfedő (overlay) hálózatnak** hívják. Egy új képességekkel ellátott hálózati protokoll bevezetése közös érdek, ahogy azt az „IPv6 IPv4-en keresztül" példánk is mutatja. Az alagút típusú átvitel hátránya, hogy azon a hálózaton, amelyen az alagútat kiépítették, egyik hoszt sem érhető el, mivel a csomagok nem tudnak kiszabadulni az alagút közepén. Az alagutaknak ez a korlátozása azonban VPN-ek (**Virtual Private Networks – virtuális magánhálózat**) esetén előnnyé válik. A VPN egyszerűen egy réteg, amelyet a biztonság növelésére használnak. A VPN-eket a 8. fejezetben vizsgáljuk meg.



5.41. ábra. *Egy autó alagút típusú átvitele Franciaországból Angliába*

5.5.4. Útválasztás összekapcsolt hálózatokban

Egy interneten keresztüli útválasztás ugyanazt az alapproblémát veti fel, mint az egyetlen hálózaton belüli útválasztás. Kiindulásként a hálózatok belsőleg különböző útválasztó algoritmusokat használhatnak. Az egyik hálózat például kapcsolatállapot-alapú útválasztást, a másik pedig távolságvektor-alapú útválasztást használ. Mivel a kapcsolatállapot-alapú algoritmusoknak ismerniük kell a topológiát, de a távolságvektor-alapú algoritmusoknak nem, ez a különbség önmagában is megbonyolítja a legrövidebb útvonal megtalálását az interneten.

A különböző operátorok által futtatott hálózatok nagyobb problémákhoz vezetnek. Először, az operátorok eltérő elképzeléssel rendelkezhetnek azzal kapcsolatban, hogy mi a jó útvonal a hálózaton. Előfordulhat, hogy az egyik operátor a legkisebb késleltetésű útvonalat akarja használni, a másik pedig a legolcsóbbat. Ennek eredményeképpen az operátorok különböző mennyiséget használnak a legrövidebb útvonal költségének beállításához (késleltetési időt szemben a pénzbeli költségekkel). A súlyozások nem lesznek összehasonlíthatók a hálózaton, így a legrövidebb útvonalakat nem jól határozzák meg.

Ami még rosszabb, elképzelhető az is, hogy az egyik operátor azt akarja, hogy a másik ne is tudja a hálózaton lévő útvonalak részleteit, talán mivel a súlyok és útvonalak érzékeny információt hordozhatnak (mint például a költség), amelyek versenyképes üzleti előnyt eredményezhetnek.

Végül az összekapcsolt hálózat lényegesen nagyobb lehet bármely benne lévő hálózatnál. Ilyenkor olyan útválasztó algoritmusra lehet szükség, amely jól skálázható hierarchia használatával, még abban az esetben is, ha egyik egyéni hálózatnak sem kell hierarchiát használnia.

Ezek a megfontolások egy kétszintű útválasztó algoritmust eredményeznek: minden hálózaton belül egy **körzeten belüli** vagy **belső útválasztó protokollt** (**intradomain** vagy **interior gateway protocol**) használnak (a „gateway" egy régebbi szóhasználat az „útválasztóra"). Ez lehet kapcsolatállapot-alapú protokoll, amelyet már leírtunk. Az összekapcsolt hálózatot alkotó hálózatokon a **körzetek közötti** vagy **külső útválasztó protokoll** (**interdomain** vagy **exterior gateway protocol**) kerül felhasználásra. A hálózatok használhatnak különböző körzeten belüli protokollokat, de ugyanazt a körzetek közötti protokollt kell használniuk. Az interneten a körzetek közötti útválasztó protokollt **BGP**-nek (**Border Gateway Protocoll – határátjáró protokoll**) hívják. Ezt a következő szakaszban tárgyaljuk.

Van még egy fontos bevezetendő kifejezés. Mivel minden hálózat a többitől függetlenül működik, gyakran hivatkoznak rájuk **autonóm rendszerekként** (**Autonomous System, AS**). Erre jó elméleti modell egy ISP-hálózat. Valójában az ISP-hálózat több AS-ből állhat, amennyiben ezt több hálózatként kezelik vagy több hálózatból lett összeszedve. De a különbség általában nem jelentős.

A két szint általában nem szigorúan hierarchikus, mivel nagyon nem optimális útvonalak alakulhatnak ki, ha egy nagy nemzetközi hálózat és egy kis regionális hálózat alkotna egyetlen hálózatot. A hálózatok útvonalaival kapcsolatban azonban viszonylag kevés információ került kiadásra az összekapcsolt hálózat útvonalainak megkeresésére. Ez segít a bonyodalmak megoldásában. Javítja a skálázhatóságot és lehetővé teszi, hogy az operátorok szabadon válasszanak útvonalakat a saját hálózataikban a saját

választott protokollal. Ehhez nem kell összehasonlítani a súlyokat a hálózatokon, vagy érzékeny adatokat kitenni a hálózaton kívülre.

Eddig nagyon kevés szót ejtettünk az összekapcsolt hálózatban levő útvonalak meghatározási módjáról. Az interneten fontos tényező az ISP-k közötti üzleti megállapodás. Minden ISP felszámíthat pénzt a többi ISP-nek a forgalom átviteléért. Másik tényező, hogy ha az összekapcsolt hálózatok útválasztásához gyakran át kell lépni nemzetközi határokat, akkor a különböző jogszabályokat is figyelembe kell venni, mint például Svédország szigorú személyiségjogi törvényeit a svéd állampolgárok személyi adatainak Svédországból történő kivitelére vonatkozóan. Ezek a nem műszaki tényezők bekerülnek az **útválasztási politikába** (**routing policy**), amely megszabja az autonóm hálózatok által használt útvonalak kiválasztásának módját. A BGP leírásakor foglalkozunk az útválasztási politikákkal is.

5.5.5. Csomag darabokra tördelése

Minden hálózat megszab valamilyen maximális csomagméretet. Ezeknek a határoknak különféle okai lehetnek. Néhány ezek közül a következő:

1. Hardver (például egy Ethernet-keret hossza).
2. Operációs rendszer (például minden puffer 512 bájtos).
3. Protokollok (például a csomaghossz mezőben a bitek száma).
4. Igazodás valamilyen nemzeti vagy nemzetközi szabványhoz.
5. Az a kívánság, hogy a hibák által okozott újraadások valamilyen szintre csökkenjenek.
6. Az a kívánság, hogy egyetlen csomag se foglalhassa le a csatornát túl sokáig.

Mindezen tényezők eredménye, hogy a hálózattervezők nem választhatnak tetszésük szerinti maximális csomagméretet. A maximális adatmezőhossz Ethernet esetén 1500 bájt, 802.11 esetén pedig 2272. Az IP nagyvonalúbb, 65 515 bájtos csomagokat engedélyez.

A hosztok általában nagy csomagok átvitelét részesítik előnyben, mivel ez csökkenti a csomag többletterhét, mint amilyen például a fejrész bájtjaira pazarolt sávszélesség. Nyilvánvaló hálózat-összekapcsolási probléma jelentkezik, amikor egy nagy csomag olyan hálózaton akar keresztülhaladni, amelynek a maximális csomagmérete túl kicsi. Ez állandó probléma, és a megoldást erre az interneten szerzett tapasztalatok alapján találták meg.

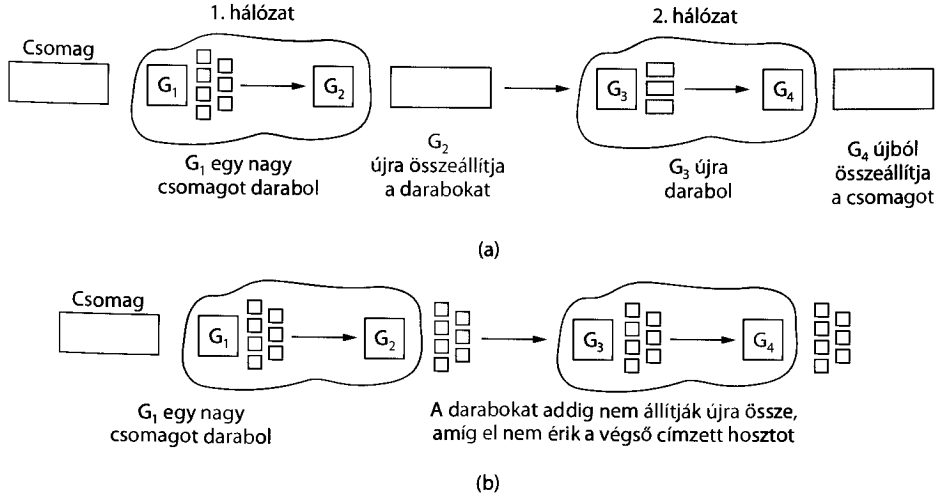
Egy lehetséges megoldás az, ha eleve elkerüljük a probléma felmerülését. Ezt azonban egyszerűbb mondani, mint megoldani. A forrás általában nem tudja, hogy a csomag milyen útvonalon megy a célig, így azt különösen nem tudja, hogy milyen kicsiknek kell lenni azoknak a csomagoknak, amelyeket ott elfogadnak. Ezt a csomagméretet az

útvonal **legnagyobb átvihető adategységének** hívjuk (**Maximum Transmission Unit, MTU**). Még ha a forrás tudja is az útvonal MTU-ját, akkor is a csomagok útválasztása független az összeköttetés nélküli hálózaton, mint például az internet. Az útválasztás azt jelenti, hogy az útvonalak hirtelen változhatnak, amely váratlanul módosíthatja az útvonal MTU-ját.

Alternatív megoldás erre a problémára, ha megengedjük az útválasztóknak, hogy a csomagokat **darabokra** (**fragments**) tördeljék, és minden darabot önálló hálózati rétegbeli csomagként küldjenek el. De mint ahogy azt minden kisgyermekes szülő tudja, egy nagy tárgy kisebb darabokká alakítása lényegesen könnyebb, mint a visszairányú folyamat. (A fizikusok még egy nevet is adtak ennek a jelenségnek: a termodinamika második főtétele.) A csomagkapcsoló hálózatoknak is gondjaik vannak a darabok újbóli összerakásával.

Két ellentétes stratégia létezik a darabok eredeti csomaggá történő visszaállítására. Az első stratégia az, hogy a „kiscsomagos" hálózat által okozott darabolást átlátszóvá kell tenni az utána következő hálózatok számára, amelyeken a csomagnak át kell haladnia a végső célcím felé. Ezt a lehetőséget az 5.42.(a) ábra mutatja. Ebben a megoldásban, amikor egy túlméretes csomag érkezik meg a G_1 útválasztóra, az útválasztó feldarabolja ezt a csomagot, és az összes darabot ugyanannak a kimenő útválasztónak, a G_2-nek címezi. Ez az útválasztó a darabokat újra összerakja egy csomaggá. Így a kiscsomagos hálózaton történő áthaladás átlátszó lesz. A következő hálózatok nem is tudnak arról, hogy darabolás történt.

Az átlátszó darabolás egyszerű, de van néhány problémája. Egyrészt, a kimeneten levő útválasztónak tudnia kell, mikor kapta meg az összes részt, tehát vagy egy számlálómezőt vagy egy „csomag vége" bitet kell megadni. Másrészt, mivel az összes csomagnak ugyanazon az útválasztón keresztül kell távoznia ahhoz, hogy újból össze lehessen azokat állítani, az útvonalak korlátozottak. Azzal, hogy nem engedjük meg, hogy egyes darabok

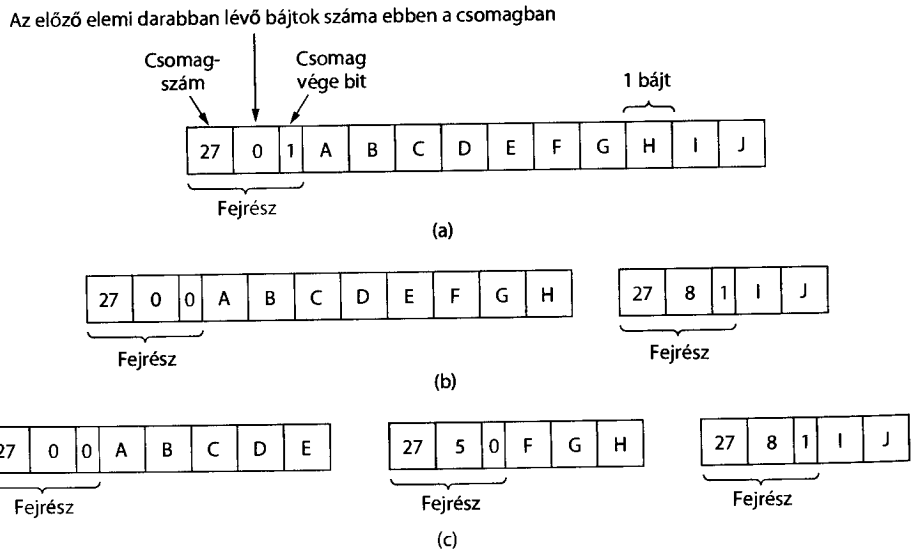


5.42. ábra. *(a) Átlátszó darabolás. (b) Nem átlátszó darabolás*

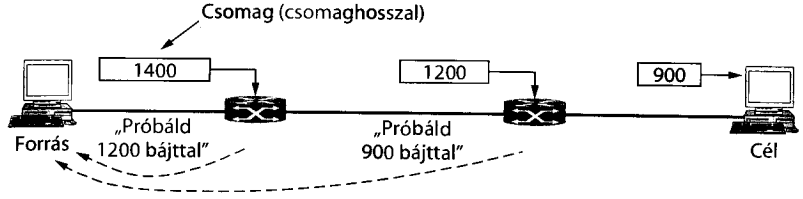
más-más útvonalat kövessenek a végső célcím felé, némi teljesítőképesség-vesztéssel kell számolni. Ennél jelentősebb az útválasztó által esetlegesen elvégzendő munka. Szükség lehet a darabok pufferelésére, amint azok megérkeznek, és el kell dönteni, hogy ezeket mikor kell eldobni, ha nem az összes darab érkezik meg. A munka egy része felesleges lehet, ha a csomag egy sor kiscsomagos hálózaton megy keresztül, és ismételten fel kell darabolni, majd újra össze kell állítani.

A másik darabolási stratégia az, hogy tartózkodjunk a közbeeső útválasztóknál az összerakástól. Ha egyszer egy csomagot feldaraboltunk, minden darabot úgy kezelünk, mintha az egy eredeti csomag lenne. Az útválasztók átviszik a darabokat, ahogy az az 5.42.(b) ábrán látszik. Az összeállítás csak a címzett hoszton történik meg.

A nem átlátszó darabolás fő előnye, hogy kevesebb munkát követel meg az útválasztóktól. Az IP így működik. A kialakítás megköveteli, hogy a darabok oly módon legyenek számozva, hogy az eredeti adatfolyam újból összeállítható legyen. Az IP által használt kialakítás minden darabhoz megad egy csomagszámot (amelyet minden csomag tartalmaz), egy abszolút bájteltolást a csomagban, és egy jelzőt, amely jelzi, hogy ez a csomag vége-e. Erre mutat példát az 5.43. ábra. Ez a kialakítás egyszerű, és vannak vonzó tulajdonságai. A darabok belehelyezhetők egy pufferbe a címzett hoszton, az összeállítás helyén, még abban az esetben is, ha a csomagok nem sorrendben érkeznek. A darabok továbbdarabolhatók, ha azok olyan hálózaton mennek keresztül, ahol még kisebb az MTU. Ezt az 5.43.(c) ábra mutatja. Elképzelhető, hogy a csomagot újbóli küldés (ha nem érkezett meg az összes darab) esetén másképpen darabolják fel. Végül, a



5.43. ábra. *Darabolás, ha az elemi darabméret 1 bájt. (a) Eredeti csomag, amely 10 adatbájtot tartalmaz. (b) A darabok egy olyan hálózaton való keresztülhaladás után, ahol a maximális csomagméret 8 adatbájt + fejrész. (c) A darabok egy 5 adatbájt + fejrész csomagméretű útválasztón való áthaladás után*



5.44. ábra. *Útvonal-MTU felderítése*

darabok tetszőleges méretűek lehetnek, akár egészen kicsik is (például csomagfejrész + 1 bájt). Az összes esetben a cél egyszerűen a csomagszámot és a darabeltolást használja az adatok megfelelő elhelyezéséhez, és a csomag-vége jelzőt, ha az egész csomag megérkezett.

Sajnos ezzel a kialakítással is van néhány gond. A többletteher több lehet, mint átlátszó darabolás esetén, mivel a darabok fejrészei olyan adatkapcsolaton kerülnek átvitelre, ahol lehet, hogy nincs is rájuk szükség. De a legfőbb probléma a darabok megléte. Kent és Mogul [1987] azzal érvelt, hogy a darabolás és a fejrész által okozott többletterhelés rontja a teljesítőképességet, mivel egy teljes csomag elveszik, ha bármelyik darab elveszik, és a darabolás nagyobb terhet ró a hosztokra, mint azt eredetileg gondoltuk.

Ez visszavezet minket az eredeti megoldáshoz, mely szerint szabaduljunk meg a hálózaton történő darabolástól. A modern internet is ezt a stratégiát követi. Ezt a folyamatot **útvonal-MTU felderítésének (path MTU discovery)** [Mogul és Deering, 1990] nevezzük. Ez a következőképp működik. Az IP-csomagokban a fejrészbitek úgy vannak beállítva, hogy jelezzék, hogy nincs engedélyezve a darabolás. Ha egy útválasztó túl nagy csomagot kap, akkor hibacsomagot állít elő, azt visszaküldi a forrásnak, és eldobja a csomagot. Ezt az 5.44. ábra mutatja. Amikor a forrás megkapja a hibacsomagot, a benne lévő információ alapján újra feldarabolja a csomagot az útválasztó által kezelhető méretű darabokra. Ha az egyik útválasztónak az útvonal mentén még kisebb az MTU-ja, akkor a folyamat megismétlődik.

Az útvonal-MTU felderítésének előnye, hogy a forrás tudja, hogy milyen hosszú csomagokat küldjön. Ha az útvonalak és az útvonal MTU-ja változik, akkor új hibacsomag aktiválódik és a forrás alkalmazkodik az új útvonalhoz. A darabolásra azonban a forrás és a cél között továbbra is szükség van, kivéve, ha a magasabb rétegek ismerik az útvonal MTU-t és a megfelelő mennyiségű adatot átadják az IP-nek. A TCP és az IP jellemzően együtt kerül megvalósításra (TCP/IP-ként), az ilyen típusú információ átadása érdekében. Ez más protokollok esetén nem feltétlenül van így, de a darabolás a hálózatról átkerült a hosztokba.

Az útvonal-MTU felderítésének hátránya, hogy adódhatnak indulási késleltetések a csomag küldésénél. Nagyobb körülfordulási késleltetésre lehet szükség az útvonal megvizsgálásához és az MTU megkereséséhez, mielőtt az adatokat a célhoz kézbesítenék. Ez felveti a kérdést, miszerint vannak-e jobb kialakítások. A válasz valószínűleg „igen". Tételezzünk fel olyan kialakítást, amelyben az útválasztó egyszerűen csonkítja az MTU-t meghaladó csomagokat. Ez biztosítja, hogy a cél a lehető leggyorsabban megismerje az MTU-t (a kézbesített adatok mennyiségéből) és megkapja az adatok egy részét.

5.6. Hálózati réteg az interneten

Eljött az ideje, hogy bemutassuk az internetet részletesen. Mielőtt belemerülnénk a részletekbe, érdemes egy pillantást vetnünk azokra az elvekre, melyek annak idején az internet tervezőit vezették, és amelyeknek mai sikerét is köszönheti. Manapság amúgy is túl gyakran tűnik úgy, hogy az emberek megfeledkeztek ezekről. Az elveket az RFC 1958 sorolja fel és tárgyalja meg. Ez ugyancsak hasznos olvasmány (sőt a protokolltervezők számára kötelezővé kellene tenni, és még vizsgázniuk is kellene belőle). Az RFC erősen támaszkodik Clark [1988], valamint Saltzer és mások [1984] ötleteire. Most pedig összefoglaljuk, hogy mi mit tartunk a legfontosabb 10 vezérelvnek (a fontosabbaktól a kevésbé fontosabbak felé haladva).

1. **A lényeg, hogy működjön.** Ne véglegesítsd a tervet vagy a szabványt, amíg több prototípus nem kommunikált sikeresen egymással. Túlságosan is gyakran fordul elő, hogy a tervezők elkészítenek egy 1000 oldalas szabványt, elfogadtatják, aztán rájönnek, hogy alapjaiban rossz és nem is működik. Ezután megírják a szabvány 1.1-es verzióját. Ezt az utat nem szabad követni.

2. **Maradj az egyszerűnél.** Ha kétségek gyötörnek, használd a legegyszerűbb megoldást. William of Occam fogalmazta meg ezt az elvet (Occam borotvája) a 14. században. Hogy mai szóhasználattal éljünk: küzdj a funkciók ellen. Ha egy funkció nem feltétlenül szükséges, hagyd ki, különösen, ha ugyanaz a hatás elérhető más funkciók kombinációjával.

3. **Válassz egyértelműen.** Ha ugyanazt a dolgot többféleképpen is meg lehet oldani, válassz ki egy megoldást. Ha két vagy több módon is megvalósíthatjuk ugyanazt, az csak bajt okoz. A szabványokban gyakran csak azért vannak különböző lehetőségek, paraméterek vagy üzemmódok, mert számos, nagy befolyású kidolgozó fél ragaszkodik ahhoz, hogy a saját elgondolása a legjobb. A tervezőknek keményen ellen kell állniuk ennek a tendenciának. Egyszerűen nemet kell mondani.

4. **Használd ki a modularitást.** Ez az elv közvetlenül a protokollvermek alkalmazásának ötletéhez vezet, melyekben minden réteg független a többitől. Ily módon, ha a körülmények egy modul vagy réteg megváltoztatását követelik meg, a többit ez nem érinti.

5. **Számíts heterogén környezetre.** Minden nagy hálózatban előfordulnak különböző hardverek, átviteli berendezések és alkalmazások. Ahhoz, hogy ezeket mind kezelni tudd, a hálózat tervezése legyen egyszerű, általános és rugalmas.

6. **Kerüld a statikus opciókat és paramétereket.** Ha a paraméterek használata végképp elkerülhetetlen (például a maximális csomagméret), akkor jobb, ha az adó és a vevő megegyeznek egy értékben, mint ha rögzített opciókat határoznánk meg.

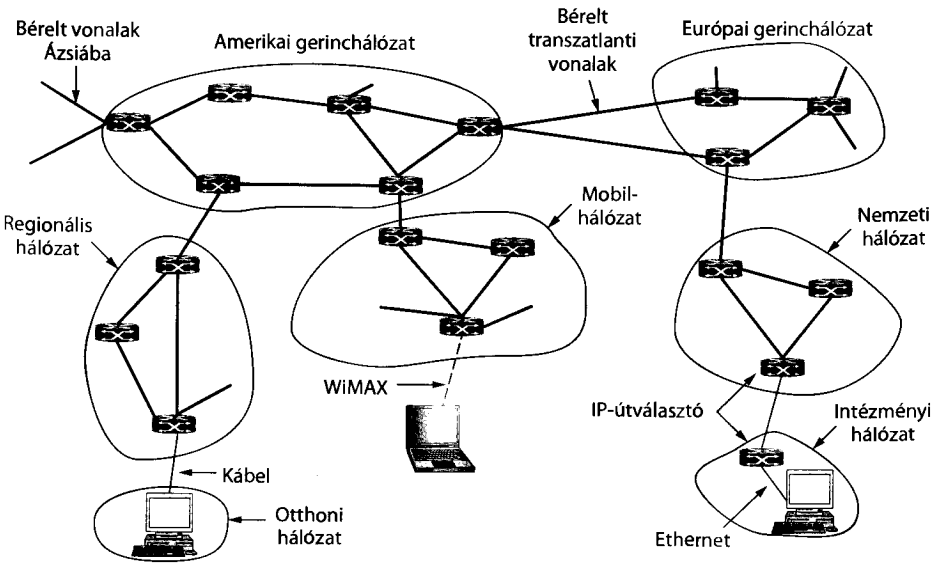
7. **Amit tervezel, az legyen jó – nem kell tökéletesnek lennie.** A tervezőknek gyakran van egy olyan tervük, ami jó ugyan, de nem képes valamilyen furcsa speciális esetet kezelni. Ahelyett, hogy erre összekuszálnák a tervet, tanácsosabb a jó terv mellett kitartaniuk, a mesterkedések terhét pedig azokra hagyni, akiknek különleges igényeik vannak.

8. **Légy szigorú a küldésnél és elnéző a vételnél.** Más szavakkal, csak olyan csomagokat küldj, amelyek szigorúan megfelelnek a szabványoknak, de számíts rá, hogy a bejövő csomagok nem lesznek teljesen szabványosak, és próbáld meg kezelni ezeket.

9. **Gondolj a skálázhatóságra.** Ha a rendszernek több millió hosztot és több milliárd felhasználót kell hatékonyan kezelnie, semmilyen központosított adatbázis nem jöhet szóba, a terhelést pedig olyan egyenletesen kell megosztani a rendelkezésekre álló erőforrások között, amennyire csak lehet.

10. **Mérlegeld a teljesítőképességet és a költségeket.** Ha egy hálózat gyatra teljesítőképességet nyújt, vagy irreális költségeket produkál, senki sem fogja majd használni.

Tegyük most félre az általános elveket, és kezdjük el az internet hálózati rétegének tanulmányozását. A hálózati réteg szintjén az internet hálózatok gyűjteményének vagy **autonóm rendszerek** (**Autonomous Systems**, **AS**) összekapcsolt együttesének tekinthető. Nincs igazi szerkezete, de létezik számos főbb gerinchálózata, amelyek nagy sávszélességű vonalakból és gyors útválasztókból állnak. Ezek közül a gerinchálózatok közül a legnagyobbat, amelyekhez mindenki más csatlakozik az internet többi részének eléréséhez, **1. rétegű hálózatoknak** (**Tier 1 networks**) nevezzük. A gerinchálózatokhoz ISP-k csatlakoznak, amelyek internet-hozzáférést biztosítanak lakások, vállalatok, adatközpontok és kolokációs létesítmények számára, amelyek tele vannak szervergépekkel és regionális (középszintű) hálózatokkal. Az adatközpontok szolgáltatják az interneten



5.45. ábra. *Az internet sok hálózat egybekapcsolt összessége*

keresztül küldött tartalom nagy részét. A regionális hálózatokhoz további ISP-k, illetve az egyetemeken, vállalatoknál és ISP-knél levő LAN-ok csatlakoznak. Az 5.45. ábrán ennek a kvázihierarchikus szerveződésnek egy vázlata látható.

Az a ragasztó, amely az internetet egybetartja, az **IP (Internet Protocol – internetprotokoll)**. Sok régebbi hálózati rétegbeli protokolltól eltérően, ezt már a kezdetektől a hálózatok összekapcsolását szem előtt tartva tervezték. A hálózati rétegről jó elképzelés lehet a következő: az a feladata, hogy best-effort (vagyis nem garantált) szállítást biztosítson csomagoknak a forrásgéptől a célgépig történő eljuttatásához, függetlenül attól, hogy ezek a gépek ugyanazon a hálózaton vannak-e vagy sem, és vannak-e köztük más hálózatok vagy nincsenek.

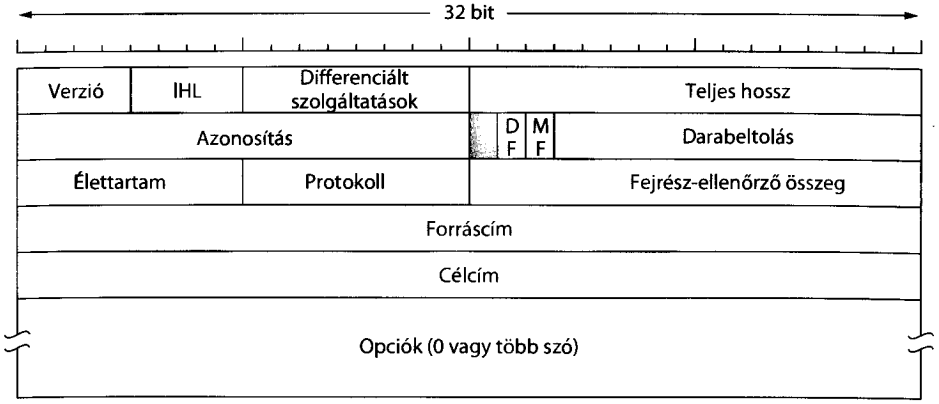
Az interneten a kommunikáció a következőképpen működik. A szállítási réteg veszi az adatfolyamokat, és feltördeli úgy, hogy azok IP-csomagként elküldhetők legyenek. Elméletben a csomagok egyenként 64 KB hosszúak lehetnek, de a gyakorlatban rendszerint nem hosszabbak 1500 bájtnál (így beleférnek egy Ethernet-keretbe). Az IP-útválasztók minden csomagot továbbítanak az interneten keresztül, az útvonal mentén az egyik útválasztóról a következőre, a végső cél eléréséig. A célnál a hálózati réteg átadja az adatokat a szállítási rétegnek, amely azután átadja azt a vételi folyamatnak.

Az 5.45. ábrán látható példában az otthoni hálózaton lévő hoszttól induló csomagnak négy hálózatot és számos IP-útválasztót kell megjárnia, hogy eljusson az vállalati hálózatra, amelyen a címzett hoszt található. Ez nem ritka a gyakorlatban, sőt számos ennél hosszabb útvonal is van. Az interneten redundáns kapcsolatok is vannak, gerinchálózatokkal és ISP-kel, amelyek sok helyen csatlakoznak egymáshoz. Ez azt jelenti, hogy két hoszt között számos lehetséges útvonal van. Az IP útválasztó protokoll feladata annak eldöntése, hogy melyek a használandó útvonalak.

5.6.1. Az IP-protokoll 4-es változata

Az internet hálózati rétegének tanulmányozásához jó kiindulási pont maguknak az IP-datagramoknak a formátuma. Egy IPv4-datagram egy fejrészből és egy törzsrészből vagy hasznos részből áll. A fejrésznek van egy 20 bájtos rögzített része és egy változó hosszúságú opcionális része. A fejrész formátuma az 5.46. ábrán látható. A bitek balról jobbra, fentről lefele kerülnek továbbításra, a *Verzió* mező legmagasabb helyi értékű bitje megy elsőnek. [Ez a „felsővég" (big endian) hálózati bájtsorrend. Az alsóvég gépeken (little endian), mint például az Intel x86 számítógépeken, adáskor és vételkor is szoftveres átalakítás szükséges.] Visszatekintve, az alsóvég jobb választás lett volna, de az IP tervezésekor senki nem tudta, hogy domináns számítástechnikai megoldás lesz belőle.

A *Verzió* mező azt tartja nyilván, hogy a datagram a protokoll melyik verziójához tartozik. Jelenleg a 4-es változat uralkodó az interneten, és ezzel kezdjük a leírást. Azáltal, hogy a verziót minden datagram elején megadják, a verziók közti átmenet évekig is eltarthat. Valójában az IP következő változata, az IPv6, több mint tíz évvel ezelőtt jelent meg, de a bevezetése még mindig csak az elején tart. Erre később térünk ki. Az IPv6 használatát akkor fogják kényszeríteni, ha Kína majd 2^{31} lakosa rendelkezni fog asztali géppel, hordozható számítógéppel vagy IP-telefonnal. Ha már a számozásnál tartunk, megemlítjük, hogy az IPv5 egy kísérleti, valós idejű folyamtovábbító protokoll volt, amit soha nem használtak széles körben.



5.46. ábra. *Az IPv4- (internetprotokoll) fejrész*

Mivel a fejrész hossza nem állandó, a fejrész egyik mezője, az *IHL* szolgál arra, hogy 32 bites szavakban megadja a fejrész hosszát. A legkisebb érték 5, ebben az esetben semmilyen opció nem szerepel. Ennek a 4 bites mezőnek a maximális értéke 15, amely a fejrészt 60 bájtra, ennélfogva az *Opciók* mezőt 40 bájtra korlátozza. Néhány opcióhoz, mint például ahhoz, amelyik a csomag által megtett utat jegyzi fel, a 40 bájt túl kicsi, ezáltal az opció értelmét veszti.

A *Differenciált szolgáltatások* mező az egyik azok közül a mezők közül, amelyek jelentése (kissé) megváltozott az évek során. Eredeti neve *Szolgáltatás típusa* volt. Akárcsak régen, ma is az a célja, hogy különbséget tegyen az eltérő szolgáltatási osztályok között. A megbízhatóságnak és a sebességnek számos kombinációja képzelhető el. A digitalizált hangnál a gyors kézbesítés fontosabb, mint a pontosság. A *Szolgáltatás típusa* mező 3 bitet biztosított a prioritás jelzéséhez, és 3 bitet annak jelzéséhez, hogy a hoszt megmondja, mi számára a legfontosabb: a késleltetés, az átbocsátóképesség vagy a megbízhatóság. Azt azonban igazából senki nem tudta, hogy mit lehet kezdeni ezekkel a bitekkel az útválasztón, ezért ezeket évekig nem használták. A differenciált szolgáltatások kialakításakor az IETF aztán végül bedobta a törölközőt, és újra felhasználta a mezőt. Jelenleg 6 bitet használnak a csomag szolgáltatási osztályának jelzésére: a fejezet korábbi részében már leírtuk a gyorsított és biztosított szolgáltatásokat. Az alsó 2 bit explicit torlódásértesítési információt hordoz, például hogy a csomag tapasztalt-e torlódást. Az explicit torlódásértesítést a fejezet korábbi részében írtuk le, a torlódáskezelés részeként.

A *Teljes hossz*ba (Total lenght) a datagram minden része beleértendő, a fejrész is és az adatrész is. A maximális hossz 65 535 bájt. Jelenleg ez a felső korlát még elegendő, de a jövőbeni gigabites hálózatoknál nagyobb datagramokra lehet majd szükség.

Az *Azonosítás* (Identification) mező a daraboláskor szükséges ahhoz, hogy a címzett hoszt eldönthesse, melyik datagramhoz tartozik az újonnan érkezett darab. Egy datagram minden darabja ugyanazt az *Azonosítás* értéket tartalmazza.

A következő egy kihasználatlan bit, amely meglepő, mivel a rendelkezésre álló hely az IP-fejrészben meglehetősen kevés. Áprilisi tréfaként Bellovin [2003] felvetette, hogy ezt a bitet használják a rosszindulatú forgalom detektálására. Ez nagyban leegyszerűsítené a biztonságot, mivel az „evil" bittel rendelkező csomagokról lehetne tudni, hogy

azokat támadók küldték, és így eldobhatók. Sajnos azonban a hálózati biztonság nem ilyen egyszerű.

Majd két 1 bites mező következik, amely a darabolással kapcsolatos. A *DF* jelentése: Ne darabold (*D*on't *F*ragment). Ez az útválasztóknak szóló parancs, hogy ne darabolják fel a csomagot. Eredetileg az volt a célja, hogy támogassa azokat a hosztokat, amelyek nem tudják újra összerakni a darabokat. Jelenleg ezt az útvonal MTU-jának felderítésére szolgáló folyamat részeként alkalmazzák. Az MTU a legnagyobb csomag, amely az útvonal mentén darabolás nélkül átküldhető. Ha a datagram *DF* bittel van megjelölve, akkor az adó tudja, hogy egy darabban fog megérkezni, ellenkező esetben az adó hibaüzenetet kap vissza.

Az *MF* jelentése: több darab (*M*ore *F*ragments). Ezt a bitet minden darabban be kell állítani, kivéve az utolsóban. Ez azért szükséges, hogy tudjuk, vajon egy datagram minden darabja megérkezett-e.

A *Darabeltolás* mező megmondja, hova tartozik a mostani darab a datagramban. Egy datagram minden darabjának – kivéve az utolsót – 8 bájt többszörösének kell lennie, mert ez az elemi darabméret. Mivel 13 bit áll rendelkezésre, ez legfeljebb 8192 darabot jelent datagramonként, amely 65 536 bájtos maximális datagramhosszt eredményez, eggyel nagyobbat, mint amit a *Teljes hossz* mező lehetővé tesz. Az *Azonosítás, MF és Darabeltolás* mező együtt valósítja meg a darabolást az 5.5.5. szakaszban leírt módon.

Az *Élettartam* mező egy számláló, amelyet a csomag élettartamának korlátozására használnak. Ez elvileg az időt mérné másodpercekben, ezáltal maximálisan 255 másodperc hosszú életet tenne lehetővé. Minden ugrásnál csökkenteni kell, és ha egy csomag hosszú ideig állt sorban egy útválasztóban, akkor elvileg többször is csökkenteni kellene. Gyakorlatilag csak az ugrásokat számolja. Amikor eléri a nullát, a csomagot el kell dobni, és egy figyelmeztető csomagot kell visszaküldeni a forráshoszthoz. Ez a tulajdonság megelőzi, hogy a datagramok a végtelenségig kóboroljanak, ami egyébként megtörténhetne, ha az útválasztó táblázatokba valamikor hiba csúszna.

Amikor a vételi oldalon a hálózati réteg összeállított egy teljes datagramot, tudnia kell, mit tegyen vele. A *Protokoll* mező mondja meg, melyik szállítási folyamatnak adja át. Lehetséges a TCP, de az UDP és pár másik protokoll is. A protokollok számozása az interneten egységes. A protokollok számozását és a többi számkiosztást az RFC 1700 tartalmazta, de ma már egy online adatbázisban találhatók a *www.iana.org* weboldalon.

Mivel a fejrész lényeges információt hordoz, mint például a címek, a védelem érdekében kiszámítja a saját ellenőrző összegét, a *Fejrész-ellenőrző összeget* (Header Checksum). Az algoritmus az, hogy egyes komplemens aritmetikával az érkezés sorrendjében összeadjuk a 16 bites félszavakat, és vesszük az eredmény egyes komplemensét. Az algoritmus alapján a *Fejrész-ellenőrző összeget* a csomag érkezésekor nullának várjuk. Az ilyen ellenőrző összeg a csomag hálózaton való átküldése során fellépő hibák észleléséhez hasznos. Ne feledjük el, hogy ezt minden ugrásnál újra kell számítani, mivel legalább egy mező (az *Élettartam* mező) mindig változik, de alkalmazhatók trükkök a számítás felgyorsítására.

A *Forráscím* és *Célcím* mutatja a forrás és a cél hálózati interfészének az IP-címét. Az internetcímeket a következő részben tárgyaljuk.

Az *Opciók* mező egy menekülési útvonal, amelyet arra terveztek, hogy a protokoll következő verzióinak lehetőségük legyen olyan információt belevenni a protokollba,

Opció	Leírás
Biztonság	Meghatározza, mennyire titkos a datagram
Szigorú forrás általi útválasztás	Megadja a teljes követendő utat
Laza forrás általi útválasztás	Felsorolja a felkeresendő útválasztókat
Útvonal feljegyzése	Felszólítás, hogy minden útválasztó fűzze hozzá az IP-címét
Időbélyeg	Felszólítás, hogy minden útválasztó fűzze hozzá az IP-címét és az időbélyegét

5.47. ábra. *Néhány IP-opció*

amelyek az eredeti tervben nem szerepeltek, hogy kísérletezhessenek új ötletek kipróbálásával, és hogy ne kelljen olyan információ számára is fejrészbiteket lefoglalni, amelyekre csak ritkán van szükség. Az opciók változó hosszúságúak. Mindegyik az opciót azonosító egybájtos kóddal kezdődik. Néhány opciónál ezt egy egybájtos hosszmező követi, majd egy vagy több adatbájt. Az *Opciók* mezőt négy bájt többszörösére töltik ki. Eredetileg öt opció létezett, ahogy az az 5.47. ábrán látszik.

A *Biztonság* opció azt mondja meg, milyen titkos az információ. Elméletben egy katonai útválasztó használhatná ezt a mezőt arra, hogy ne irányítson bizonyos, a katonaság szempontjából „rosszfiúnak" minősülő országokon keresztül. A gyakorlatban minden útválasztó figyelmen kívül hagyja, így az egyetlen gyakorlati funkciója az, hogy a kémek könnyebben megtalálhassák a jó dolgokat.

A *Szigorú forrás általi útválasztás* opció IP-címek sorozataként megadja a teljes utat a forrástól a célig. A datagramnak pontosan ezt az utat kell követnie. Ez nagyon hasznos rendszermenedzserek számára, hogy vészcsomagokat küldjenek az útválasztó táblák meghibásodása esetén, vagy időzítési méréseket végezzenek.

A *Laza forrás általi útválasztás* opció megköveteli a csomagtól, hogy a megadott útválasztókon, a megadott sorrendben áthaladjon, de útközben áthaladhat más útválasztókon is. Rendesen ez az opció csak egy pár útválasztót bocsát rendelkezésre, hogy egy bizonyos utat kényszerítsen ki. Például, hogy egy Londonból Sydneybe tartó csomagot kelet helyett nyugat felé kényszerítsünk, ez az opció például megadhat New York-i, Los Angeles-i és honolului útválasztókat. Ez az opció nagyon hasznos, ha politikai vagy gazdasági megfontolásokból keresztül kell haladni bizonyos országokon, vagy el kell kerülni azokat.

Az *Útvonal feljegyzése* opció arra utasítja az útba ejtett útválasztókat, hogy az IP-címüket fűzzék hozzá az *Opciók* mezőhöz. Ez lehetővé teszi a rendszermenedzsereknek, hogy kinyomozzák a hibákat az útválasztó algoritmusokban. („Miért keresik fel a Houstonból Dallasba menő csomagok először mind Tokiót?") Amikor az ARPANET először létrejött, egyik csomag sem érintett kilenc útválasztónál többet, így az opció 40 bájtja bőségesen elegendő volt. Ahogy fentebb említettük, most már túl kicsi.

Végül az *Időbélyeg* opció olyan, mint az *Útvonal feljegyzése* opció, kivéve, hogy minden útválasztó a 32 bites IP-címe mellé egy 32 bites időbélyeget is feljegyez. Ez az opció is főleg az útválasztó algoritmusok hibakereséséhez való.

Az IP-opciókat ma már nem részesítik előnyben. Számos útválasztó figyelmen kívül hagyja, vagy nem dolgozza fel azokat hatékonyan, félretolva őket, mint szokatlan esetet. Ezért ezeket csak részben támogatják és csak ritkán használják.

5.6.2. IP-címek

Az IPv4 meghatározó jellemzője a 32-bites cím. Az interneten minden hosztnak és útválasztónak van egy IP-címe, amely az IP-csomagok *Forráscím* és *Célcím* mezői hordozzák. Fontos megjegyezni, hogy az IP-cím valójában nem egy hoszthoz tartozik. Igazából egy hálózati interfészre utal, tehát ha egy hoszt két hálózathoz is csatlakozik, akkor két IP-címének is kell lennie. Mindazonáltal a gyakorlatban a legtöbb hoszt egy hálózathoz csatlakozik, így csak egy IP-címe van. Ezzel szemben az útválasztók több interfésszel rendelkeznek, ezért több IP-címük is van.

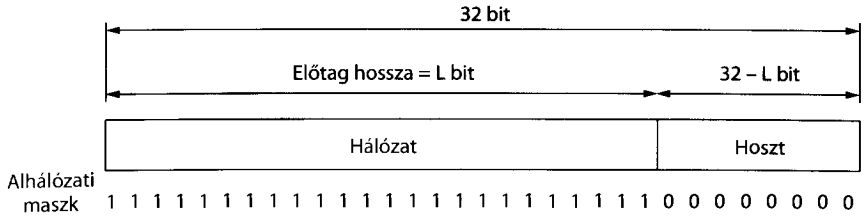
Előtagok

Az IP-címek, az Ethernet-címekkel ellentétben, hierarchikusak. Minden 32-bites cím változó hosszúságú hálózati részből (felső bitek) és hosztrészből áll (alsó bitek). A hálózati rész értéke az egy hálózaton – például Ethernet LAN-on – lévő összes hoszt esetén megegyezik. Ez azt jelenti, hogy a hálózat az IP-címtér folyamatos blokkjának felel meg. Ezt a blokkot **előtagnak** (**prefix**) hívjuk.

Az IP-címeket rendszerint **pontokkal elválasztott decimális jelölésrendszerben** (**dotted decimal notation**) írják. Ebben a formátumban minden 4 bájtot tízes számrendszerben írnak ki, 0-tól 255-ig, például a 80D00297 32 bites hexadecimális cím decimális formája a következő: 128.208.2.151. Az előtagok megadják a legkisebb IP-címet a blokkban, illetve a blokk méretét. A méretet a hálózati részben lévő bitek száma határozza meg. A bitek a hosztrészben változhatnak. Ez azt jelenti, hogy a méretnek kettő hatványának kell lennie. Megegyezés szerint az előtag IP-címe egy perjellel van kiegészítve, és ezt követi a hálózati rész bitjeinek száma. Ha az előtag a példánkban 2^8 címet tartalmaz és 24 bit marad a hálózati részre, akkor a következőképpen fest: 128.208.0.0/24.

Mivel az előtag hossza nem következtethető ki magából az IP-címből, az útválasztó protokolloknak át kell adniuk az előtagokat az útválasztóknak. Bizonyos esetekben az előtagokat egyszerűen a hosszuk írja le, mint a „/16" esetében (amelyet „per 16"-nak mondanak). Az előtag hossza megfelel az 1-esek bináris maszkjának a hálózati részben. Az ilyen írásmódot **alhálózati maszknak** (**subnet mask**) hívják. Ha ezt ÉS kapcsolatba hozzuk az IP-címmel, megkapjuk a hálózati részt. A példánkban az alhálózati maszk a 255.255.252.0. Az 5.48. ábra egy előtagot és alhálózati maszkot mutat be.

A hierarchikus címeknek számos előnye és hátránya van. Az előtagok fő előnye az, hogy az útválasztók a csomagokat továbbítani tudják a cím hálózati része alapján, amíg minden hálózat egyedi címblokkal rendelkezik. A hosztrész az útválasztók számára érdektelen, mivel az egy hálózaton lévő összes hoszt csomagját ugyanabba az irányba kell küldeni. Csak akkor kell a megfelelő hoszthoz továbbítani a csomagokat, amikor azok elérik a célhálózatot. Ezáltal kisebbek lesznek az útválasztó táblák, mint egyébként



5.48. ábra. *Egy IP-előtag és egy alhálózati maszk*

lennének. Képzeljük el, hogy a hosztok száma az interneten eléri az egymilliárdot. Így nagyon nagy táblázatot kellene minden útválasztónak fenntartania. A hierarchia alkalmazásával azonban az útválasztóknak csak körülbelül 300 000 előtagot kell fenntartaniuk a táblázatban.

A hierarchia használatával lehetővé válik az internet útválasztásának skálázása, amelynek két hátránya is van. Az első, hogy a hoszt IP-címe a hálózaton belüli helyétől függ. Az Ethernet-cím bárhol a világon használható, de minden IP-cím adott hálózathoz tartozik, és az útválasztók csak az erre a címre küldött csomagokat tudják kézbesíteni a hálózatra. Ezért szükség van olyan kialakításra – mint például a mozgó IP –, amely támogatja azokat a hosztokat, amelyek mozognak a hálózatok között, de meg akarják tartani IP-címüket.

A második hátrány, hogy a hierarchia címpazarlás, hacsak nem kezelik körültekintően. Ha a címeket (túl) nagy blokkokban rendelik a hálózatokhoz, akkor számos kihasználatlan cím kerül lefoglalásra. Ez nem számítana, ha bőségesen állnának rendelkezésre címek. Már több mint 20 évvel ezelőtt rájöttek arra, hogy az internet nagymértékű növekedése gyorsan kimeríti a szabad címteret. Az IPv6 megoldást jelent erre, de ennek széles körű bevezetéséig nagy a nyomás, hogy az IP-címeket minél hatékonyabban használják ki.

Alhálózatok

A hálózatszámokat az **ICANN** (**Internet Corporation for Assigned Names and Numbers – Internettársaság Kiosztott Nevek és Számok Kezelésére**) nevű nonprofit intézmény kezeli a konfliktusok elkerülése végett. Az ICANN különböző regionális hatóságokra bízta a címtér egy részét, hogy azok osszák ki az IP-címeket az ISP-nek és más vállalatoknak. Ez az a folyamat, amellyel egy vállalat IP-cím blokkot foglalhat le.

Ez a folyamat azonban csak a történet eleje, mivel az IP-cím hozzárendelés folyamatos, ahogy a vállalat növekszik. Azt mondtuk, hogy az előtagalapú útválasztáshoz a hálózatban minden hosztnak ugyanazzal a hálózatszámmal kell rendelkeznie. Ez a tulajdonság a hálózatok növekedésével gondokat okozhat. Tekintsünk például egy egyetemet, ahol kezdetben /16 előtag állt rendelkezésre a Számítástudományi Tanszék Ethernet-hálózatán lévő gépek számára. Egy évvel később a Villamosmérnöki Tanszék is ki akart kerülni az internetre. A Bölcsészettudományi Tanszék is követi példájukat. Milyen IP-címet kell használnia ezeknek a tanszékeknek? A beszerezhető további blokkok kívül vannak az egyetem hálózatán, ezenkívül drágák is lehetnek, és alkalmasságuk is megkér-

dőjelezhető. Továbbá a /16 előtag több mint 60 000 hoszt számára elengedő címet már lefoglalt. A nagy növekedés lehetővé tétele lehet a cél, de amíg ez be nem következik, addig nagy pazarlás lefoglalni további IP-cím blokkokat az egyetem számára. Másfajta szervezés szükséges tehát.

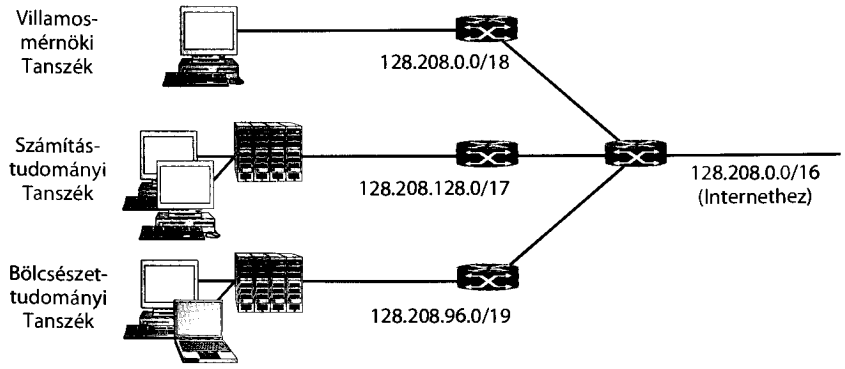
A megoldás az, hogy tegyük lehetővé a címblokk szétvágását több részre, hogy egy hálózat a belső felhasználás szempontjából több részre legyen felosztva, miközben a külvilág számára továbbra is egyetlen hálózatnak látszik. Ez az **alhálózatra osztás (subnetting)**, és a nagyobb hálózat feldarabolásából keletkezett hálózatokat (mint amilyenek például az Ethernet LAN-ok) **alhálózatoknak (subnets)** nevezik. Amint azt az 1. fejezetben említettük, ez a szóhasználat ütközik az „alhálózat" szó azon jelentésével, ahol az egy hálózatban levő útválasztók és kommunikációs vonalak összességét értjük alatta.

Az 5.49. ábra bemutatja, hogy az alhálózatok hogyan segíthetnek a példánkban felmerült probléma megoldásában. Az egyetlen /16-ot osztottuk részekre. Ennek a felosztásnak nem kell egyenletesnek lennie, de minden darabnak sorbarendezettnek kell lennie, hogy minden bit használható legyen az alacsonyabb hosztrészben. Ebben az esetben a blokk felét (/17) a Számítástudományi Tanszék (SZT) számára, egynegyedét a Villamosmérnöki Tanszék (VT) számára (/18), illetve egynyolcadát (/19) a Bölcsészettudományi Tanszék (BT) számára foglalták le. A maradék nyolcad kiosztatlan marad. Egy másik mód a címblokk felosztásának bemutatására, hogy megnézzük az eredményül kapott előtagokat, amelyeket binárisan jelölünk:

Számítástudományi Tanszék:	10000000	11010000	1|xxxxxxx	xxxxxxxx
Villamosmérnöki Tanszék:	10000000	11010000	00|xxxxxx	xxxxxxxx
Bölcsészettudományi Tanszék:	10000000	11010000	011|xxxxx	xxxxxxxx

A függőleges vonal (|) az alhálózat száma és a hosztrész közötti határt ábrázolja.

Amikor egy csomag beérkezik, honnan fogja tudni a központi útválasztó, hogy melyik alhálózatnak kell átadnia? Itt jön képbe az előtag részletezése. Lehetne, mondjuk egy 65 536 bejegyzésből álló táblázat minden útválasztóban, ami az egyetem minden hosztjához megmondja, hogy melyik útválasztót kell használni. Ez azonban meghiúsí-



5.49. ábra. *IP-előtag felosztása különálló hálózatokra alhálózatokra osztással*

taná a hierarchiából származó fő méretezési előnyöket. Ehelyett az útválasztónak csak a hálózatok alhálózati maszkjait kell ismernie az egyetemen belül.

Amikor egy IP-csomag megérkezik, az útválasztó megkeresi a célcímét a csomagban, és ellenőrzi, hogy melyik alhálózathoz tartozik. Ehhez az útválasztó a célcímet és minden egyes alhálózati maszkot ÉS kapcsolatba hoz, és ellenőrzi, hogy az eredmény a megfelelő előtag-e. Tekintsük például a 128.208.2.151 IP-címre küldött csomagot. Ahhoz, hogy kiderítsük, hogy a Számítástudományi Tanszéknek küldték-e a csomagot, ÉS kapcsolatba hozzuk a 255.255.128.0 maszkkal az első 17 bit kinyerése érdekében (ami a 128.208.0.0), és megnézzük, hogy ez megfelel-e az előtagcímnek, ami a 128.208.128.0. Ezek nem egyeznek. Ha az első 18 bitet nézzük a Villamosmérnöki Tanszéknél, akkor a 128.208.0.0 címet kapjuk az alhálózati maszkkal való ÉS művelet elvégzése után. Ez megfelel az előtag címének, így a csomag továbbításra kerül a Villamosmérnöki Tanszék hálózatához tartozó interfészre.

Az alhálózati felosztások később szükség esetén módosíthatók az egyetemen lévő összes útválasztó alhálózati maszkjának frissítésével. A hálózaton kívülről az alhálózati felosztás nem látható, így új alhálózat kiosztásához nem kell felvenni a kapcsolatot az ICANN-nel, illetve nem kell külső adatbázisokat módosítani.

CIDR – Osztály nélküli körzetek közti útválasztás

Annak ellenére, hogy IP-címblokkok kerülnek lefoglalásra, így a címek felhasználása hatékony, egy probléma továbbra is fennáll: az útválasztó tábla nagymértékű növekedése.

Egy hálózat szélén lévő intézmény (mint például az egyetem) útválasztójának útválasztó táblázatában minden alhálózathoz tartoznia kell egy bejegyzésnek, ami megmondja az útválasztónak, hogy melyik vonalat használja az adott alhálózat eléréséhez. Az intézményen kívüli célokhoz az útválasztók használhatják az egyszerű alapértelmezett szabályt, amely szerint a csomagokat a vonalon az ISP felé kell küldeni, amelyik azután az intézményt összeköti az internet többi részével. Az összes többi célcímnek valahol kívül kell lennie.

Az internet „közepén" lévő ISP-k és gerinchálózatok útválasztói számára nem biztosított ez a luxus. Ezeknek az útválasztóknak tudnia kell, hogy melyik úton érjék el az egyes hálózatokat, az egyszerű alapértelmezés nem fog működni. Ezek a központi útválasztók az internet **alapértelmezés nélküli zónájában (default-free zone)** vannak. Valójában senki nem tudja, hogy hány hálózat kapcsolódik az internethez, de ez a szám meglehetősen nagy, valószínűleg legalább egymillió. Ez nagyon nagy táblázatot eredményezhet. Ez a számítógép-szabványok szerint nem tűnik nagynak, de az útválasztóknak minden csomagtovábbításnál bele kell nézniük a táblázatba, és a nagy ISP-k útválasztói másodpercenként akár több millió csomagot is továbbíthatnak. A csomagok ilyen sebességű feldolgozásához speciális hardver és gyors memória szükséges, nem egy általános célú számítógép.

Ezenfelül az útválasztó algoritmusok azt igénylik, hogy minden útválasztó átadja a többinek az általa elért címekkel kapcsolatos információt. Minél nagyobbak a táblázatok, annál több információt kell átadni és feldolgozni. A feldolgozás legalább lineárisan

növekszik a táblázat méretével. Minél több a kommunikáció, annál nagyobb a valószínűsége annak, hogy az információ egy része elvész, legalábbis ideiglenesen, amely instabil útválasztáshoz vezethet.

Az útválasztó táblázatok problémája megoldható lett volna mélyebb hierarchia használatával. Működhetne például egy olyan IP-cím, ami ország, állam/tartomány, város, hálózat és hoszt mezőket tartalmazna. Ekkor minden útválasztónak csak azt kéne tudnia, hogyan juthat egy adott országba, a saját országán belüli államokba/tartományokba, a saját államában vagy tartományában lévő városokba, és a városán belüli hálózatokba. Sajnos ez a megoldás lényegesen több mint 32 bitet igényelne az IP-címek számára, és nem használná ki hatékonyan a címeket (Liechtensteinnek ugyanannyi bitje lenne, mint az Egyesült Államoknak).

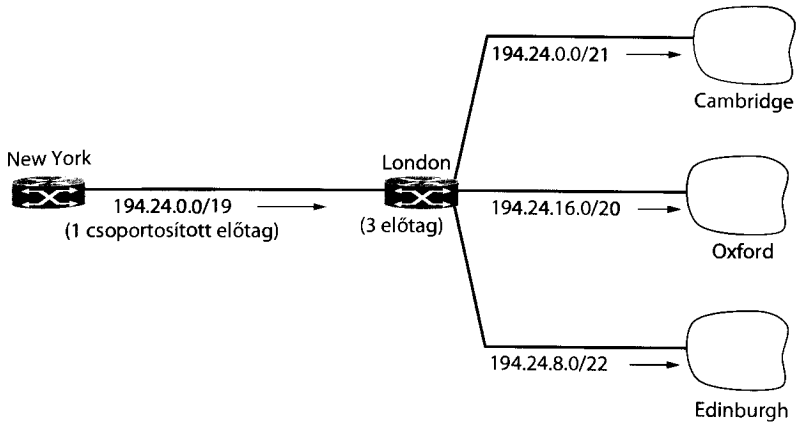
Szerencsére, azért tehetünk valamit az útválasztó táblázat méretének csökkentése érdekében. Alkalmazhatjuk ugyanazt a megoldást, mint az alhálózatokra bontás esetében: a különböző helyeken lévő útválasztók tudhatják egy adott IP-címről, hogy különböző méretű előtagokhoz tartozik. Ahelyett azonban, hogy a címblokkot alhálózatokra osztatnánk, több kis előtagot egyesítünk egyetlen nagyobb előtagba. Ezt a folyamatot **útvonal-csoportosításnak** (**route aggregation**) hívják. Az eredményül kapott nagyobb előtagot **szuperhálózatnak** (**supernet**) is nevezik, az alhálózatokkal ellentétben, amelyeket a címblokkok felosztása eredményez.

A csoportosítással az IP-címeket különböző méretű előtagok fogják tartalmazni. Ugyanazt az IP-címet, amelyet az egyik útválasztó a /22 (ez a blokk 2^{10} címet tartalmaz) részeként kezeli, elképzelhető, hogy másik útválasztó a nagyobb /20 (amely 2^{12} címet tartalmaz) részeként kezeli. Minden útválasztó saját feladata, hogy birtokában legyen a megfelelő előtag-információnak. Ez a kialakítás alhálózatokra bontást használ és **CIDR-nek** (**Classless InterDomain Routing, osztály nélküli körzetek közti útválasztás**) hívják. Ennek legújabb változatát az RFC 4632 írja le [Fuller és Li, 2006]. A név kiemeli a különbséget azon címekkel szemben, amelyek a hierarchiát osztályokkal kódolják. Ezt röviden ismertetjük.

A CIDR-t egyszerűbben megérthetjük a következő példán keresztül. Van egy 8192 IP-címből álló blokk, ahol a címek 194.24.0.0-tól kezdve állnak rendelkezésre. Tegyük fel, hogy a Cambridge-i Egyetemnek 2048 címre van szüksége, és megkapja a 194.24.0.0-tól 194.24.7.255-ig terjedő címtartományt, a 255.255.248.0 maszkkal. Ez megfelel a /21 előtagnak. Következőként az Oxfordi Egyetem kér 4096 címet. Mivel a 4096 címből álló blokknak 4096-os bájthatárra kell kerülnie, nem kaphatja meg a 194.24.8.0-tól kezdődő címeket, helyette a 194.24.16.0-tól a 194.24.31.255-ig terjedő címeket kapja,

Egyetem	Első cím	Utolsó cím	Hány hoszt?	Előtag
Cambridge	194.24.0.0	194.24.7.255	2048	194.24.0.0/21
Edinburgh	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(Nem használt)	194.24.12.0	194.24.15.255	1024	194.24.12.0/22
Oxford	194.24.16.0	194.24.31.255	4096	194.24.16.0/20

5.50. ábra. *IP-cím kiosztások egy halmaza*



5.51. ábra. *IP-előtagok csoportosítása*

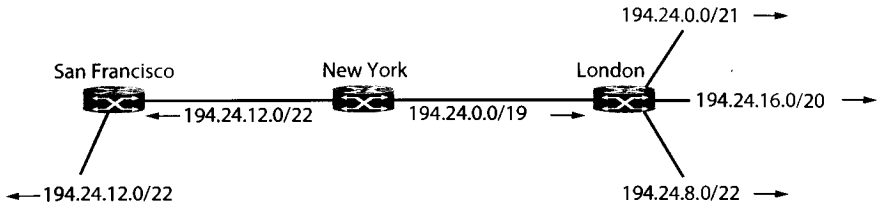
a 255.255.240.0 alhálózati maszkkal. Végül az Edinburghi Egyetem kér 1024 címet, és megkapja a 194.24.8.0-tól 194.24.11.255-ig terjedő címtartományt a 255.255.252.0 maszkkal. Ezeket a kiosztásokat foglalja össze az 5.50. ábra.

Az alapértelmezés nélküli zóna összes útválasztója megkapja a három hálózatban lévő IP-címeket. Az egyetemekhez közeli útválasztóknak szüksége lehet arra, hogy minden előtaghoz más-más kimenő vonalon küldjenek csomagokat, így minden előtaghoz kell egy bejegyzés az útválasztó táblázatban. Erre példa az 5.51. ábrán látható Londonban lévő útválasztó.

Most nézzük meg ezt a három egyetemet a New Yorkban lévő távoli útválasztó szemszögéből. A három előtagban lévő összes IP-címet New Yorkból (illetve az USA-ból általában) Londonba kell küldeni. A londoni útválasztási folyamat észleli ezt, és a három előtagot összefogja a 194.24.0.0/19 csoportos bejegyzésbe, és ezt átadja a New York-i útválasztónak. Az előtag 8K címet tartalmaz és lefedi a három egyetemet, valamint a nem kiosztott 1024 címet. Csoportosítással a három előtagból egy lesz, ezzel csökken az előtagok száma, amelyről a New Yorkban lévő útválasztót, illetve a hozzá tartozó útválasztó táblázat bejegyzéseit frissíteni kell.

Csoportosítás alkalmazásával ez automatikus folyamat; az egyes előtagok interneten elfoglalt helyétől függ, nem pedig attól, hogy az adminisztrátor hogyan rendeli hozzá a címeket a hálózatokhoz. A csoportosítást sűrűn használják az interneten, és ez az útválasztó táblázatok méretét körülbelül 200 000 előtagra csökkentheti.

További csavar, hogy az előtagok átfedhetik egymást. Az a szabály, hogy a csomagokat a legjobban meghatározott útvonal irányába kell küldeni, vagy a **leghosszabb egyező előtag** (**longest matching prefix**) irányába, amely a legkevesebb IP-címet tartalmazza. A leghosszabb egyező előtagalapú útválasztás eléggé rugalmas, ahogy az a New York-i útválasztó viselkedésén látható (lásd 5.52. ábra). Ez az útválasztó egyetlen csoportos előtagot használ három angol egyetem számára London irányába küldéséhez. A korábban ebben az előtagban elérhető címblokk azonban egy San Francisco-i hálózatnak lett kiosztva. Egyik lehetőség, hogy a New York-i útválasztó négy előtagot tart fenn. Ezek közül hármat a Londonba küldendő csomagokhoz, és egyet a San Franciscóba küldendő



5.52. ábra. *Leghosszabb egyező előtagalapú útválasztás a New York-i útválasztón*

csomagokhoz. Ehelyett a leghosszabb egyező előtagalapú útválasztás ezt a továbbítást a két előtaggal tudja kezelni, ahogy azt láthatjuk. Egy teljes (az összes előtagot tartalmazó) előtagot arra használnak, hogy a teljes címblokk forgalmát Londonba irányítsák. Egy specifikusabb előtagot arra használnak, hogy a nagyobb előtag egy részéhez tartozó forgalmat San Franciscóba irányítsák. A leghosszabb egyező előtag szabályának segítségével a San Franciscó-i hálózaton lévő IP-címek a kimenő vonalon San Francisco felé, a nagyobb előtagban lévő összes többi IP-cím pedig London felé kerül továbbküldésre.

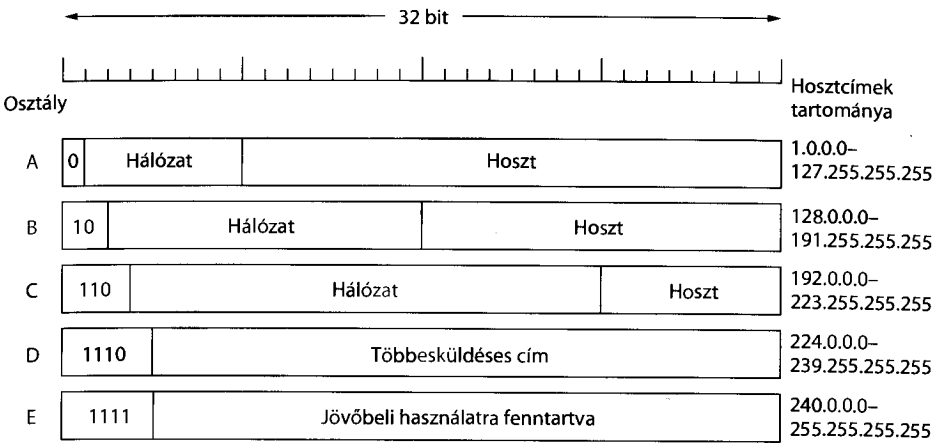
Elméletben a CIDR működése a következő. Amikor egy csomag érkezik, az útválasztó megnézi az útválasztó táblázatot annak eldöntéséhez, hogy a cél az előtagon belül található-e. Elképzelhető, hogy több különböző előtaghosszal rendelkező bejegyzés felel meg ennek. Ebben az esetben a leghosszabb előtaggal rendelkező bejegyzést használja fel. Így ha a /20 és /24 maszkkal egyaránt van egyezés, akkor a /24 bejegyzést használja a csomaghoz tartozó kimenő vonal meghatározásához. Ez a folyamat azonban fárasztó lenne, ha a táblázatot tényleg bejegyzésről bejegyzésre végignézné. Ehelyett összetett algoritmusokat alakítottak ki a címegyeztetési folyamat felgyorsítása érdekében [Ruiz-Sanchez és mások, 2001]. A kereskedelmi forgalomban lévő útválasztók egyéni VLSI-chipeket használnak, amelyekben ezt az algoritmust hardverbe ágyazzák.

Osztályalapú és speciális címzés

Annak jobb megértése érdekében, hogy miért hasznos a CIDR, röviden bemutatjuk azt a kialakítást, amely a CIDR-t megelőzte. 1993 előtt az IP-címeket öt kategóriába sorolták, amelyet az 5.53. ábra mutat. Ezt a kiosztást **osztályalapú címzésnek** (**classful addressing**) nevezték.

Az A, B és C osztályú formátumok sorrendben a következő címtartományokat teszik lehetővé: legfeljebb 128 hálózat, mindegyiken 16 millió hoszttal, 16 384 hálózat 65 536 hoszttal, illetve 2 millió hálózat (például LAN) 256 hoszttal (ezek közül néhány speciális). A többesküldés is támogatott (a D formátumú osztály), amelyben a datagram több hoszt felé kerül továbbításra. Az 1111-gyel kezdődő címeket jövőbeli használatra tartják fenn. Ezeket érdemes lenne használni az IPv4-címtér kimerülése miatt. Sajnálatos módon számos hoszt nem fogadja el ezeket a címeket érvényesként, mivel azok sokáig kívül estek a korlátokon, ezért nehéz megtanítani a régi hosztokat új trükkökre.

Ez egy hierarchikus kialakítás, de a CIDR-rel ellentétben a címblokkok mérete rögzített. Több mint kétmilliárd cím létezik ugyan, de a címtartomány osztályok szerinti szervezésének gyakorlata ebből több milliót elveszteget. Az igazi bajkeverők konkrétan a



5.53. ábra. *IP-címformátumok*

B osztályú hálózatok. A legtöbb intézménynek egy 16 millió címet tartalmazó, A osztályú hálózat túl nagy, míg a 256 címet tartalmazó C osztályú hálózat túl kicsi. Egy 65 536 címet tartalmazó B osztályú hálózat viszont éppen megfelelő. Az internet folklórjában ez a helyzet a **három medve problémája** (**three bears problem**) néven ismert [Southey, 1848].

A valóságban egy B osztályú cím a legtöbb intézmény számára túl nagy. Tanulmányok is kimutatták, hogy a B osztályú hálózatok több mint a fele 50-nél kevesebb hoszttal rendelkezik. Ilyen esetekben egy C osztályú hálózat is elegendő lett volna, de nyilván minden B osztályú címet kérő intézmény úgy gondolta, hogy egy napon kinövi a 8 bites hosztmezőt. Visszatekintve, esetleg jobb lett volna, ha a C osztályú hálózatok 8 helyett 10 bitet használtak volna a hosztszámhoz, amely így hálózatonként 1022 hosztot tenne lehetővé. Ebben az esetben a legtöbb intézmény valószínűleg megelégedett volna egy C osztályú címmel, és ezekből félmillió darab lehetne (szemben a mindössze 16 384 B osztályú hálózattal).

Nehéz az internet tervezőit hibáztatni azért, mert nem hagytak több (és kisebb) B osztályú címet. Azokban az időkben, amikor meghozták a döntést a három osztály létrehozásáról, az internet egy kutatási hálózat volt, ami a főbb amerikai kutatóegyetemeket kötötte össze (és ezeken kívül még nagyon kevés céget és katonai támaszpontot, amelyek hálózati kutatással foglalkoztak). Senki sem látta az internetben a jövő tömeges kommunikációs rendszerét, ami még a telefonhálózattal is felveszi majd a versenyt. Azokban az időkben az emberek minden bizonnyal azt mondták: „Az USA-nak körülbelül 2000 főiskolája és egyeteme van. Még ha az összes rákapcsolódik is az internetre, és más országokból is sok egyetem csatlakozik, akkor sem érjük el a 16 000-et, hiszen nincs is annyi egyetem az egész világon. Továbbá, ha a hosztszám egész számú bájtból áll, az gyorsítja a csomagok feldolgozását" (amelyet ezután teljes egészében a szoftver végzett). Elképzelhető, hogy egy napon az emberek visszanéznek, és a telefonszám-sémát kialakító szakembereket hibáztatják, a következő felkiáltással: „Micsoda idióták. Miért nem rakták bele a bolygó számát a telefonszámba?" De jelenleg ez nem tűnik szükségesnek.

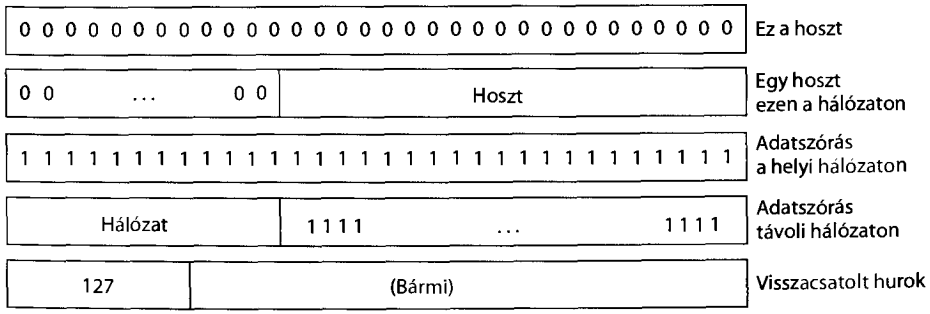
A probléma megoldása érdekében alhálózatokat vezettek be a címblokkok rugalmas hozzárendeléséhez az intézményen belül. Később kialakították a CIDR-t a globális útvá-

lasztó táblázat méretének csökkentésére. Manapság az IP-cím azon bitjei, amelyek jelzik, hogy a cím A, B vagy C osztályú hálózathoz tartozik, már nincsenek használatban, azonban az irodalomban még gyakran hivatkoznak ezekre az osztályokra.

Ahhoz, hogy lássuk, hogy az osztályok elhagyása hogyan teszi még bonyolultabbá a továbbítást, nézzük meg, hogy milyen egyszerű volt ez a régi osztályalapú rendszerben. Amikor egy csomag megérkezett egy útválasztóhoz, az IP-címének másolatát 28 bittel jobbra léptették, hogy megkapják a 4 bites osztályszámot. Ezután 16-felé szétválogatták a csomagokat A, B, C (valamint D és E) osztályra, ahol nyolc eset tartozott az A osztályhoz, négy a B-hez, illetve kettő a C-hez. Az egyes osztályok kódja ezután kimaszkolta a 8, 16 vagy 24 bites hálózatszámot, és jobbra igazította egy 32 bites szóban. A hálózatszámot azután kikeresték az A, B vagy C táblázatból, általában az A és a B hálózat szerint indexelve és a C szerint hash-elve. Amint meglett a bejegyzés, ki lehetett keresni a kimeneti vonalat és a csomag továbbítható volt. Ez sokkal egyszerűbb, mint a leghosszabb egyező előtagművelet, amely a továbbiakban nem tud egy egyszerű táblázatban keresni, mivel az IP-címnek tetszőleges hosszúságú előtagja lehet.

A D osztályú címeket továbbra is használják az interneten a többesküldéshez. Valójában pontosabb, ha azt mondjuk, hogy többesküldésre kezdik használni, mivel az interneten történő többesküldés a múltban nem terjedt el széles körben.

Számos más címnek speciális jelentése van, ahogy azt az 5.54. ábra mutatja. A legkisebb IP-címet, a 0.0.0.0-át, a hosztok az elinduláskor használják. Ez „ezt a hálózatot" vagy „ezt a hosztot" jelenti. Azok az IP-címek, amelyeknek a hálózatszáma 0, az aktuális hálózatra vonatkoznak. Ezek a címek lehetővé teszik a gépeknek, hogy a saját hálózatukra hivatkozzanak anélkül, hogy tudnák annak a számát (de a hálózati maszkot ismerniük kell, hogy tudják, hány 0-t tegyenek az elejére). A csupa 1-ből álló cím, azaz a legnagyobb, 255.255.255.255 cím a jelzett hálózat összes hosztját jelenti. Ez lehetővé teszi az adatszórást a helyi hálózaton, jellemzően egy LAN-on. Azok a címek, amelyeknek helyes a hálózatcíme, hosztmezője pedig csupa 1-est tartalmaz, lehetővé teszik adatszóró csomagok küldését az interneten bárhol elhelyezkedő távoli LAN-okra. Azonban számos hálózati adminisztrátor letiltja ezt a szolgáltatást, mivel általában veszélyt jelent a biztonságra. Végül a 127.*xx.yy.zz* formájú címek visszacsatolásos tesztelésre vannak fenntartva. Az erre a címre küldött csomagok nem kerülnek ki a vezetékre; ezeket a rendszer helyileg dolgozza fel, és bejövő csomagként kezeli. Ez lehetővé teszi csomagok küldését a hosztnak anélkül, hogy az adó ismerné annak számát. Ez a tesztelésnél hasznos.



5.54. ábra. *Speciális IP-címek*

NAT – hálózati címfordítás

Az IP-címek szűkös erőforrások. Ha egy internetszolgáltatónak mondjuk /16 címe van, akkor ezzel 65 534 darab hosztszámhoz jut. Ha ennél több ügyfele van, bajban van.

Az IP-címek hiánya olyan technikák kialakulásához vezetett, amelyek az IP-címeket takarékosan használják. Egyik megoldás, ha a hálózatra bejelentkező számítógép dinamikusan kap IP-címet, amit a kapcsolat lezárásakor vissza is vesznek. A cím ezután hozzárendelhető másik számítógéphez, amely aktívvá válik. Ily módon egyetlen /16-os címmel akár 65 534 aktív felhasználó is kiszolgálható.

Ez a stratégia jól működik néhány esetben, például betárcsázó hálózatok, valamint mobil és egyéb olyan számítógépek esetén, amelyek ideiglenesen ki vannak kapcsolva vagy nem csatlakoznak a hálózatra. Üzleti felhasználók esetén azonban nem megfelelő. Itt a PC-ktől elvárják, hogy folyamatosan be legyenek kapcsolva. Néhány alkalmazott gépéről éjszaka készítenek biztonsági mentést, más gépek pedig kiszolgálók lehetnek, amelyeknek távoli kéréseket kell kiszolgálniuk késlekedés nélkül. Ezek a vállalatok olyan összeköttetéssel rendelkeznek, amely állandó kapcsolatot biztosít az internet felé.

A dolgot tovább nehezíti, hogy egyre több otthoni felhasználó fizet elő az internetre ADSL-en vagy kábeltévé-hálózaton keresztül, mivel nincs csatlakozási költség (csak havi alapdíj). Számos felhasználónak van legalább két számítógépe otthon, gyakran minden családtagnak egy, és mindannyian egész nap el szeretnék érni az internetet. Erre a megoldás, hogy az otthoni PC-ket egy LAN-nal kötik össze, amelyre egy (vezeték nélküli) útválasztót is tesznek. Az útválasztó csatlakozik az internetszolgáltatóhoz. A szolgáltató szemszögéből a család ezután ugyanolyan, mint egy néhány számítógépet tartalmazó kisvállalat. Üdvözöljük a Kovács Rt.-nél! Az eddig bemutatott technikákkal minden számítógépnek saját IP-címmel kell rendelkeznie egész nap. Több ezer ügyféllel – különösen üzleti ügyfelekkel és kis vállalatként működő családi ügyfelekkel – kapcsolatban levő ISP esetén az IP-címek iránti igény gyorsan meghaladhatja a rendelkezésre álló blokkot.

Az IP-címek hiánya nem elvi probléma, ami talán valamikor a távoli jövőben bekövetkezik. Ez itt és most történik. A hosszú távú megoldás az, hogy az egész internet átáll az IPv6-ra, amely 128 bites címeket használ. Az átállás lassan meg is történik, de még évek telnek el, mire befejeződik. Ezért gondolták néhányan azt, hogy rövid távon gyors javításra van szükség. Ez a gyors javítás a **NAT (Network Address Translation – hálózati címfordítás)** képében jött el. A NAT-ot az RFC 3022 írja le, amelyet az alábbiakban foglaljuk össze. További információkért lásd Dutcher [2001] munkáját.

A NAT alapötlete az, hogy az internetszolgáltató minden otthon vagy cég számára egyetlen egy (vagy legalábbis kevés számú) IP-címet oszt ki. Egy vállalati hálózaton *belül* minden számítógép egyedi IP-címet kap, amit a házon belüli forgalom irányításához használnak. Amikor viszont egy csomag elhagyja a vállalati hálózatot, és kimegy az ISP felé, akkor címfordításra kerül sor: az egyedi belső IP-cím helyét egy osztott nyilvános IP-cím veszi át. Ez a fordítás három IP-címtartományt használ, amelyet privát használatra jelöltek ki. A hálózatok ezeket belsőleg tetszőlegesen használhatják. Az egyetlen kikötés az, hogy magán az interneten nem jelenhet meg olyan csomag, ami ezeket a címeket tartalmazza. A három fenntartott címtartomány a következő:

10.0.0.0	– 10.255.255.255/8	(16 777 216 hoszt)
172.16.0.0	– 172.31.255.255/12	(1 048 576 hoszt)
192.168.0.0	– 192.168.255.255/16	(65 536 hoszt)

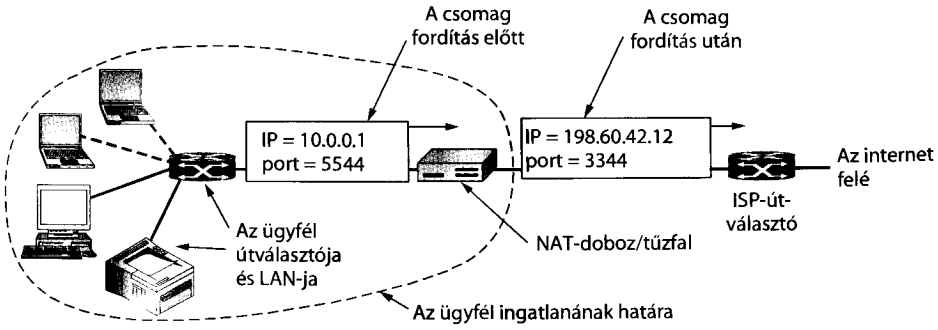
Az első tartomány 16 777 216 hoszt számára nyújt elegendő címet (leszámítva az összes 0-t és összes 1-et, mint mindig), a legtöbb vállalat ezt is választja, még akkor is, ha nincs is szükségük ennyi címre.

A NAT működését az 5.55. ábra mutatja be. A vállalat határain belül minden gépnek egyedi címe van, 10.*x.y.z* alakban. Ha azonban egy csomag elhagyja a vállalat területét, akkor áthalad egy **NAT-dobozon (NAT box)**, ami átalakítja a belső IP-forráscsomópont, vagyis az ábrán a 10.0.0.1 címet a vállalat tényleges IP-címére, ami példánkban 198.60.42.12. A NAT-dobozt gyakran a tűzfallal együtt, egy eszközben valósítják meg, mert a tűzfal a biztonság érdekében amúgy is gondosan megvizsgálja, hogy mi jön be a vállalathoz, és mi lép ki onnan. A tűzfalakat a 8. fejezetben fogjuk tanulmányozni. A NAT-dobozt ezenkívül akár az útválasztóval vagy az ADSL-modemmel is egybeépíthetik.

Eddig még mindig átsiklottunk egy apró részlet fölött: amikor a válasz visszaérkezik (például egy webszervertől), természetesen a 198.60.42.12 címre küldik. Honnan tudja ekkor a NAT-doboz, hogy melyik címre cserélje ki ezt? Ebben rejlik a NAT problémája. Ha lenne még egy maradék mező az IP-fejrészben, azt felhasználhatnánk arra, hogy nyomon kövessük, ki volt az eredeti feladó; azonban már csak egy 1 bit maradt kihasználatlanul. Elvileg készíthetnénk egy új opciót, ami az eredeti forráscímet tartalmazná, de ehhez az egész internet összes gépének szoftverét meg kellene változtatni, hogy kezeljék ezt az új opciót. Ez nem túl ígéretes alternatíva egy gyors javítás számára.

Valójában a következő történt. A NAT tervezői megfigyelték, hogy a legtöbb IP-csomag TCP- vagy UDP-tartalmat hordoz. Amikor a 6. fejezetben tanulmányozni fogjuk a TCP-t és az UDP-t, látni fogjuk, hogy mindkettejük fejrésze tartalmaz egy forrásport- és egy célportmezőt. Az alábbiakban csak a TCP-portokat tárgyaljuk, de ugyanaz érvényes az UDP-portokra is. A portok 16 bites egészek, amelyek azonosítják, hogy hol kezdődik és végződik egy TCP-kapcsolat. Ezek a portok lesznek tehát azok a mezők, amelyekre a NAT működéséhez szükségünk van.

Amikor egy folyamat egy TCP-kapcsolatot szeretne kiépíteni egy másik, távoli folyamattal, hozzácsatolja magát egy addig a saját gépén nem használt TCP-porthoz. Ezt



5.55. ábra. *A NAT-doboz elhelyezkedése és működése*

nevezzük **forrásportnak (source port)**, és ez mondja meg a TCP-szoftvernek, hogy hová küldje az adott kapcsolathoz tartozó bejövő csomagokat. A folyamat megad egy **célportot (destination port)** is, hogy megmondja, kinek kell a csomagokat adni a távoli oldalon. A 0–1023-ig terjedő portokat a jól ismert szolgáltatások számára tartották fenn. A webszerverek például a 80-as portot használják, így a távoli kliensek könnyen megtalálják őket. Minden kimenő TCP-üzenet tartalmaz egy forrás- és egy célportot is. Ez a két port együtt azonosítja a kapcsolatot használó folyamatokat mindkét végponton.

A következő hasonlat talán világosabbá teszi a portok használatát. Képzeljünk el egy vállalatot egy darab központi telefonszámmal. Amikor az emberek felhívják a központi számot, akkor egy központossal beszélhetnek, aki megkérdi, melyik melléket kérik, majd kapcsolja azt. A központi szám hasonló a vállalat IP-címéhez, a kapcsolat két végén a mellékek pedig hasonlók a portokhoz. A portok tulajdonképpen további 16 bitet adnak a címzéshez, hogy azonosítsák, melyik folyamat melyik bejövő csomagot kapja meg.

A *Forrásport* mező használatával megoldhatjuk a leképezési problémánkat. Amikor egy kilépő csomag eléri a NAT-dobozt, a 10.*x.y.z* forráscímet lecseréljük a vállalat igazi IP-címére. Ezenkívül, a TCP *Forrásport* mezőt lecseréljük egy mutatóra, amely a NAT-doboz 65 536 bejegyzésből álló fordítási táblázatába mutat. A mutatott bejegyzés tartalmazza az eredeti IP-címet és az eredeti forrásportot. Végül az IP- és a TCP-fejrészek ellenőrző összegeit is újraszámolják, és az eredményt beleírják a csomagba. Azért kell kicserélni a *Forrásport* mezőt, mert a 10.0.0.1 és a 10.0.0.2 gépekből induló összeköttetések is használhatják például véletlenül az 5000-es portot, vagyis a *Forrásport* önmagában nem elég a feladó folyamat azonosítására.

Amikor egy csomag megérkezik a NAT-dobozhoz az ISP-től, a TCP-fejrészben található *Forrásport* mezőt kiveszik, és indexként használják a NAT-doboz leképezési táblázatában. Az így talált bejegyzésből kiveszik a belső IP-címet és az eredeti TCP *Forrásport*ot, és belerakják azokat a csomagba. Ezután újraszámolják mind az IP-, mind a TCP-ellenőrző összegeket, és azokat is beleírják a csomagba. A csomagot végül átadják hagyományos továbbításra a vállalati útválasztónak a 10.*x.y.z* cím használatával.

Jóllehet ez a séma voltaképp megoldja a problémát, az IP-közösségben mégis sokan visszataszítónak tartják. Íme, az ellenérvek rövid összegzése. Először is, a NAT megsérti az IP architekturális modelljét, mely szerint minden IP-cím globálisan egyedien egyetlen gépet azonosít. Az internet teljes szoftverstruktúrája erre a tényre épül. A NAT révén azonban több ezer gép használhatja (és használja is) a 10.0.0.1 címet.

Másodszor, a NAT felbontja az internet végponttól végpontig tartó (end-to-end) összeköttetés modelljét, mely szerint bármely hoszt küldhet bármely másik hosztnak tetszőleges időpontban csomagot. Mivel a leképezést a NAT-dobozban a kimenő csomagok állítják be, a bejövő csomagok addig nem fogadhatók el, amíg a kimenők ki nem mennek. A gyakorlatban ez azt jelenti, hogy a NAT-ot használó otthoni felhasználó TCP/IP-kapcsolatot létesíthet a távoli webszerverrel, de a távoli felhasználó nem létesíthet kapcsolatot az otthoni hálózaton lévő játék szerverrel. Ennek a helyzetnek a támogatásához speciális konfiguráció vagy **NAT áthidalási technikák** szükségesek.

Harmadszor, a NAT az összeköttetés nélküli internetből egyfajta összeköttetés-alapú hálózatot csinál. A gondot itt az okozza, hogy a NAT-doboznak minden rajta keresztülmenő kapcsolatról információt (egy leképezést) kell tárolnia. A kapcsolat állapotának

ilyen jellegű számontartása az összeköttetés-alapú hálózatok sajátossága, és nem az összeköttetés nélkülieké. Ha a NAT-doboz összeomlik és a leképezési táblázat elvész, akkor a doboz összes TCP-összeköttetése megsemmisül. Ha nincs NAT, akkor az útválasztók összeomlása és újraindulása nincs hosszú távú hatással a TCP-összeköttetésekre. Ilyenkor mindössze az történik, hogy a feladó folyamat időzítője néhány másodperc alatt lejár, mire az minden nyugtázatlan csomagot újraad. A NAT használatával az internet olyan sebezhető lesz, mint egy vonalkapcsolt hálózat.

Negyedszer, a NAT megsérti a protokollrétegezés legfontosabb alapelvét: a *k.* réteg nem tehet semmilyen feltételezést arról, hogy a $k+1$. réteg mit tett az adatmezőjébe. Az alapelv az, hogy a rétegeknek függetleneknek kell lenniük. Ha a TCP-t egyszer továbbfejlesztik TCP-2-re, és megváltozik a fejrész formátuma (például 32 bitesek lesznek a portszámok), akkor a NAT megbukik. A rétegezett protokollok lényege éppen az, hogy biztosítják, hogy az egyik rétegben bekövetkezett változások nem követelik meg a többi réteg megváltoztatását. A NAT felborítja ezt a függetlenséget.

Ötödször, a folyamatok az interneten nem feltétlenül használnak TCP-t vagy UDP-t. Ha az *A* gépen egy felhasználó úgy dönt, hogy egy új szállítási protokoll segítségével fog beszélgetni a *B* gép felhasználójával (például egy multimédiás alkalmazás útján), akkor a NAT-doboz bevezetésével az alkalmazás már nem fog működni, mert a NAT-doboz nem fog megfelelő TCP *Forrásportot* találni.

A hatodik probléma, hogy néhány alkalmazás több TCP/IP-kapcsolatot vagy UDP-portot használ előre leírt módon. A szabványos **FTP (File Transfer Protocol – fájlátviteli protokoll)** például a csomag törzsébe illeszti az IP-címeket, hogy a vevő azután innen kivehesse és használja. Mivel a NAT semmit nem tud ezekről a címekről, így nem tudja kicserélni azokat, illetve másképp sem tud róluk számot adni. Ezen ismeret hiánya azt jelenti, hogy az FTP és más alkalmazások, mint például a H.323 internettelefon protokoll (amit a 7. fejezetben fogunk tanulmányozni), NAT jelenlétében csődöt mondhat, hacsak nem teszünk különleges óvintézkedéseket. A NAT-ot esetleg meg lehet javítani, hogy működjön így is, de az nem túl jó ötlet, hogy a NAT-doboz szoftverét egy új alkalmazás megjelenésekor minden egyes alkalommal újra ki kelljen javítani.

És végül, mivel a TCP *Forrásport* 16 bites, legfeljebb 65 536 gépet lehet egy IP-címhez rendelni. A tényleges érték ennél kicsit kevesebb, mert az első 4096 portot speciális célokra tartják fenn. Ha azonban nem egy, hanem több IP-cím áll rendelkezésünkre, akkor mindegyikkel lekezelhetünk legfeljebb 61 440 gépet.

A NAT több más problémája mellett ezeket is tárgyalja az RFC 2993. A problémák ellenére a NAT-ot a gyakorlatban sok helyen használják, különösen otthoni és kis vállalati hálózatokhoz, mivel ez az egyetlen kisegítő eszköz az IP-címek hiányának kiküszöbölésére. Ez a tűzfalakkal is együttműködik, és a titkosságnak is megfelel, mivel alapértelmezésben blokkolja a kéretlen bejövő csomagokat. Ezért valószínűleg az IPv6 széles körű alkalmazása esetén sem fog megszűnni.

5.6.3. Az internetprotokoll 6-os verziója

Az IP-t már évtizedek óta intenzíven használják. Eddig rendkívül jól működött, ahogy azt az internet exponenciális növekedése is mutatja. Sajnos az IP gyors tempóban lesz

saját népszerűségének áldozata: kezd kifogyni a címekből. Bár a CIDR és a NAT takarékosan használja a címeket, az ICANN várhatóan kiosztja az utolsó IPv4-címeket 2012 vége előtt.[3] Ez a fenyegető végzet sok tárgyalást és vitát szült az internet közösségén belül arról, hogy mit is lehetne ez ellen tenni.

Ebben a szakaszban megtárgyaljuk a problémát, és a megoldási javaslatokat is. Az egyetlen hosszú távú megoldás a hosszabb címekre való áttérés. Az **IPv6 (az IP 6-os változata)** helyettesítő kialakítás, amely éppen ezt teszi: 128 bites címet használ. Nem túl valószínű, hogy ezek a címek az előre látható jövőben elfogyjanak. Az IPv6 bevezetése azonban nagyon bonyolult. Ez a hálózati rétegnek egy másik protokollja, amely a sok hasonlóság ellenére nem működik igazán együtt az IPv4-gyel. A vállalatok és felhasználók egyáltalán nem biztosak abban, hogy miért kell valaha is alkalmazniuk az IPv6-ot. Ennek következtében az IPv6-ot csak az internet nagyon kis részében (körülbelül 1%-ában) használják annak ellenére, hogy 1998 óta internetszabvány. A következő néhány év érdekes időszak lesz, ahogy az utolsó néhány meglévő IPv4-cím kiosztása megtörtént. Az emberek elkezdik majd árulni az IPv4-címüket az eBayen? Fel fog virágozni a címek feketepiaci kereskedelme? Ki tudja.

Az említett címproblémákon kívül egy másik kérdés is bujkál a háttérben. Az internetet korai éveiben nagyrészt az egyetemek, a csúcstechnológiai ipar és az Egyesült Államok kormányzata (különösen a Honvédelmi Minisztérium) használta. Az internet iránti érdeklődés az 1990-es években robbanásszerűen megnőtt. Sokan kezdték el használni, mégpedig jellemzően eltérő igényű emberek. Ma egyfelől rengeteg, okostelefonnal rendelkező ember használja az otthoni bázissal való kapcsolattartásra. Másfelől a számítástechnika, a távközlés és a szórakoztatóipar közelgő konvergenciája miatt lehetséges, hogy nemsokára a világon minden telefon- és televíziókészülék egy internetcsomópont lesz, ami milliárdnyi hálózati zenehallgatásra és videózásra használt gépet eredményez. Ilyen körülmények között nyilvánvalóvá vált, hogy az IP-nek fejlődnie kell, és rugalmasabbá kell válnia.

Az IETF – látván, hogy ezek a problémák feltűnnek a horizonton – 1990-ben elkezdte a munkát az IP új verzióján, egy olyan verzión, amely soha nem fogy ki a címekből, mindenféle egyéb problémákat is megold, és ezek mellett rugalmasabb és hatékonyabb is. A fő célok a következők voltak:

1. Támogatni a több milliárd hosztot, még nem hatékony címtartomány-hozzárendelés árán is.
2. Csökkenteni az útválasztó táblázatok méretét.
3. Egyszerűsíteni a protokollt, lehetővé téve ezzel az útválasztóknak a csomagok gyorsabb feldolgozását.
4. A jelenlegi IP-nél jobb biztonságot (hitelesítés és titkosság) biztosítani.
5. Nagyobb figyelmet szentelni a szolgáltatás típusának, különösen a valós idejű adatoknál.

3 2011 februárjában az utolsó IP-cím is kiosztásra került. *(A lektor megjegyzése)*

6. Segíteni a többesküldést azáltal, hogy megadják a hatósugarakat.

7. Lehetőséget adni arra, hogy egy hoszt a címének megváltoztatása nélkül barangoljon.

8. Lehetővé tenni a protokoll fejlődését.

9. Meg kell engedni, hogy az új és a régi protokoll még évekig egymás mellett létezhessen.

Az IPv6 kialakítása lehetőséget adott az IPv4 összes jellemzőjének javítására, ami messze alatta marad a mai igényeknek. Az IETF egy mindezen igényt kielégítő protokoll kifejlesztése érdekében javaslatokra és vitára szóló felhívást tett közzé az RFC 1550-ben. Huszonegy válasz érkezett, közülük nem mind volt teljes javaslat. 1992 decemberére hét komoly javaslat feküdt az asztalon. Ezek az IP kisebb módosításaitól kezdve addig terjedtek, hogy az egészet ki kellene dobni, és egy teljesen másmilyen protokollal helyettesíteni.

Az egyik javaslat az volt, hogy a TCP-t a CLNP felett kell futtatni. A CLNP az OSI által tervezett hálózati protokoll, amely 160 bites címeivel mindörökre elegendő címtartományt biztosított volna, minthogy az óceánokban lévő vízmennyiség minden molekulájához elegendő címet (durván 2^5 számú címet) tudna adni egy kis hálózat telepítéséhez. Ezzel a választással egyesíthető lenne két fő hálózati protokoll. Sokan úgy érezték azonban, ez annak lett volna az elismerése, hogy az OSI világban tulajdonképpen valamit jól csináltak, ez az állítás pedig politikailag helytelennek minősül internetkörökben. A CLNP-t közvetlenül az IP-ről mintázták, így a kettő valójában nem különbözik annyira egymástól. Tulajdonképpen, a végül is kiválasztott protokoll sokkal jobban különbözik az IP-től, mint a CLNP. Újabb csapást az mért a CLNP-re, hogy gyatrán támogatja azokat a szolgáltatástípusokat, amelyek a multimédia hatékony átviteléhez szükségesek.

A jobb javaslatok közül hármat közölt az *IEEE Network* [Deering, 1993; Francis, 1993; Katz és Ford, 1993]. Sok vita, módosítás és pozícióharc után kiválasztották a Deering- és a Francis-féle javaslatok egy módosított kombinációját, amelyet ekkor már **SIPP**-nek (**Simple Internet Protocol Plus**) hívtak, és az **IPv6** jelölést adták neki.

Az IPv6 egészen jól megfelel az IETF céljainak. Megtartja az IP jó tulajdonságait, elveti vagy kevésbé hangsúlyossá teszi a rosszakat, és új tulajdonságokkal egészíti ki, ahol szükség van rá. Általánosságban, az IPv6 nem kompatibilis az IPv4-gyel, de az összes többi internetprotokollal igen, beleértve a TCP-, UDP-, ICMP-, IGMP-, OSPF-, BGP- és DNS-protokollokat, néhol úgy, hogy kisebb módosításokra van szükség (főleg a hosszabb címek kezelése miatt). Alább tárgyaljuk az IPv6 főbb tulajdonságait. További információ található az RFC 2460–2466-ban.

Először, ami a legfontosabb, az IPv6-nak hosszabb címei vannak, mint az IPv4-nek. 128 bit hosszúak, amely megoldja azt a problémát, amelyet az IPv6-nak meg kell oldania: egy gyakorlatilag végtelen internetcím-ellátmányt biztosít. Nemsokára több mondanivalónk is lesz a címekről.

Az IPv6 második fő fejlesztése a fejrész egyszerűsítése. Csak 7 mezőt tartalmaz (szemben az IPv4 13 mezőjével). Ez a változás lehetővé teszi az útválasztóknak, hogy gyorsabban dolgozzák fel a csomagokat, és ezáltal javítsák az átbocsátást. A fejrészt is rövidesen megtárgyaljuk.

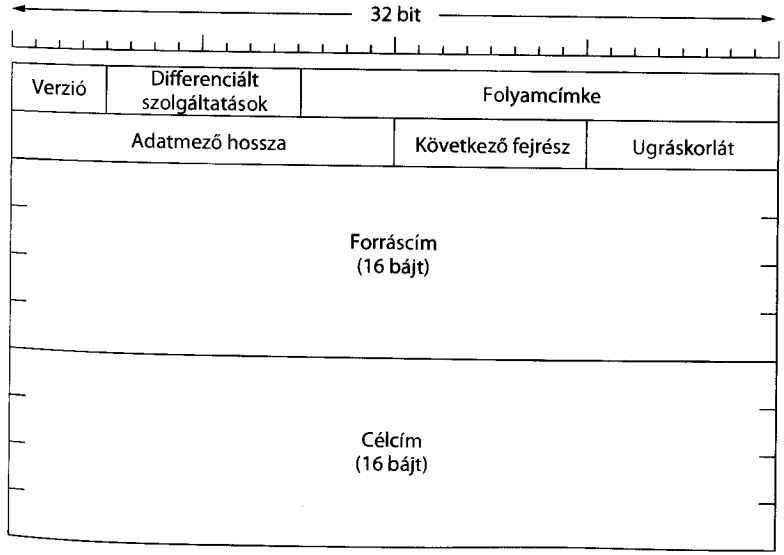
A harmadik fő előrelépés az opciók jobb támogatása. Ez a változás szükségszerűen együtt jár az új fejrésszel, mert a korábban megkövetelt mezők most opcionálisak lettek (mivel ezeket nem túl gyakran használják). Ezenkívül az opciók megjelenésének a módja is más, így az útválasztóknak egyszerű átlépni a nem nekik szánt opciókon. Ez a tulajdonság a csomagfeldolgozási időt gyorsítja fel.

A negyedik terület, ahol az IPv6 jelentős előrelépést tett, a biztonság. Az IETF-nek elege volt azokból az újságtörténetekből, ahol koraérett 12 évesek a személyi számítógépeikkel bankokba és katonai bázisokba törnek be az interneten. Erősen érezhető volt, hogy valamit tenni kell a biztonság javítása érdekében. A hitelesítés és a titkosság az új IP kulcstulajdonsága. Ezeket aztán visszamenőleg az IPv4-be is beépítették, így a biztonság területén a különbségek ma már nem olyan jelentősek.

Végül, több figyelmet szenteltek a szolgáltatásminőségnek is. Erre már a múltban is tettek több, lagymatag kísérletet, de most, ahogy a multimédia jobban teret hódít az interneten, a dolog egyre sürgetőbbé válik.

A fő IPv6-fejrész

Az IPv6-fejrész az 5.56. ábrán látható. A *Verzió* mező IPv6-nál mindig 6 (és az IPv4-nél mindig 4). Az alatt az idő alatt, amíg átállnak az IPv4-ről, ami több mint egy évtizedig is eltarthat, az útválasztók megvizsgálhatják ezt a mezőt, hogy eldöntsék, milyen fajta csomagjuk van. Viszont ez az ellenőrzés mellékhatásként elveszteget pár utasítást a kritikus úton, mivel az adatkapcsolati fejrész rendszerint jelzi a demultiplexeléshez szükséges hálózati protokollt, így néhány útválasztó esetleg átlépi ezt az ellenőrzést. Az Ethernet *Type* mezője például különféle értékekkel rendelkezik egy IPv4 vagy IPv6 adatmező



5.56. ábra. *A rögzített IPv6-fejrész (kötelező)*

jelzésére. A „Csináljuk helyesen!" és a „Legyen gyors!" táborok közt kétségkívül hosszú és parázs viták lesznek.

A *Differenciált szolgáltatások* (eredetileg *Forgalmi osztály*) mező szolgál arra, hogy különbséget tegyenek a csomagok között, amelyeknél különbözőek a követelmények a valós idejű szállítással kapcsolatban. Ezt a differenciált szolgáltatás architektúránál használják a szolgáltatásminőséghez ugyanúgy, mint ahogy az azonos nevű mezőt használták az IPv4-csomagban. Az első két bit az explicit torlódás jelzésére szolgál ugyanúgy, mint az IPv4 esetén.

A *Folyamcímke* mező lehetővé teszi, hogy a forrás és a cél megjelölje azon csomagok csoportját, amelyek azonos követelményeket támasztanak, és amelyeket a hálózatnak azonos módon kell kezelnie, ezáltal egy álösszeköttetést létesítve. Például egy bizonyos forráshoszt egy folyamatától egy meghatározott címzett hoszt egy folyamatáig tartó csomagfolyamnak szigorú késleltetési igényei lehetnek, és ezért fenntartott sávszélességre van szüksége. A folyamot előre fel lehet állítani, és egy azonosítót adni neki. Amikor egy nem nulla *Folyamcímke* mezőjű csomag tűnik fel, minden útválasztó kikeresheti a belső táblázataiból, hogy milyen különleges elbánást igényel. Tulajdonképpen a folyamok bevezetése kísérlet arra, hogy egyszerre lehessen kihasználni a datagramalapú hálózat rugalmasságát és a virtuálisáramkör-alapú hálózat garanciáit.

A szolgáltatásminőség biztosításához minden folyamot forráscím, célcím és folyamszám alapján azonosítanak. Ez azt jelenti, hogy 2^{20} folyam lehet egy időben aktív két adott IP-cím közt. Ilyen módon, ha más hosztoktól jövő, de ugyanolyan folyamszámú folyamok ugyanazon az útválasztón haladnak keresztül, az útválasztó meg tudja azokat különböztetni a forrás- és célcím segítségével. Várhatóan a folyamszámokat véletlenszerűen fogják választani ahelyett, hogy 1-től kezdve folyamatosan osztanák ki azokat, hogy az útválasztók könnyen tudják azokat hash-elni.

Az *Adatmező hossza* mező megmondja, mennyi bájt következik az 5.56. ábra 40 bájtos fejrésze után. A név megváltozott az IPv4 *Teljes hossz* mezőjéhez képest, mivel a jelentés is módosult: a 40 fejrészbájtot már nem számolják bele a hosszba, mint régebben. Ez a módosítás azt jelenti, hogy az adatmező 65 535 bájtot tartalmazhat 65 515 bájt helyett.

A *Következő fejrész* mező kiengedi a zsákbamacskát. A fejrészt azért lehetett egyszerűsíteni, mert lehetnek további (opcionális) kiegészítő fejrészek. Ez a mező mondja meg, melyik kiegészítő fejrész következik a (jelenleg) hat közül, ha egyáltalán van ilyen. Ha a fejrész az utolsó IP-fejrész, a *Következő fejrész* mező azt mondja meg, melyik szállítási protokoll kezelőjének (TCP, UDP) kell a csomagot továbbítani.

Az *Ugráskorlát* mező gátolja meg a csomagokat abban, hogy azok örökké élhessenek. Ez gyakorlatilag ugyanaz, mint az *Élettartam* mező az IPv4-ben, vagyis egy olyan mező, amelyet minden ugrásnál csökkentenek. Elméletben az IPv4-ben ez másodpercekben mért idő volt, de egy útválasztó sem használta így, ezért megváltoztatták a nevét, hogy az a tényleges működésre utaljon.

Következnek a *Forráscím* és *Célcím* mezők. Deering eredeti javaslata, az SIP, 8 bájtos címeket használt, de a felülvizsgálási folyamat során sok ember érezte úgy, hogy 8 bájtos címekkel az IPv6 néhány évtizeden belül ki fog fogyni a címekből, míg a 16 bájtos címek soha nem fognak el. Más emberek érvelése szerint a 16 bájt túlzás, megint mások pedig 20 bájtos címeket részesítettek volna előnyben, hogy az IPv6 kompatibilis legyen az OSI-datagramprotokollal. Egy másik frakció változó hosszúságú címeket akart. Sok vita és

nyomdafestéket nem tűrő szavak után úgy határoztak, hogy a legjobb kompromisszum a rögzített hosszúságú 16 bájtos címek alkalmazása.

A 16 bájtos címek leírására új jelölésrendszert is javasoltak. Nyolc, négy-négy hexadecimális számjegyből álló csoportként írjuk le a címet, a csoportok között kettősponttal, valahogy így:

8000:0000:0000:0000:0123:4567:89AB:CDEF

Mivel a sok cím sok nullát fog tartalmazni, három ésszerűsítést engedélyeztek. Először is, egy csoporton belül a bevezető nullák elhagyhatók, így a 0123 helyett 123 írható. Másodszor, egy vagy több, 16 nullából álló csoport két kettősponttal helyettesíthető. Így a fenti címből a következő lesz:

8000::123:4567:89AB:CDEF

Végül, az IPv4-címek két kettőspont és a régi, pontokkal elválasztott decimális szám formájában írhatók fel, például:

::192.31.20.46

Talán szükségtelen ennyire hangsúlyozni, de rengeteg 16 bájtos cím létezik. Egész pontosan 2^{128}, azaz körülbelül 3×10^{38} darab van belőlük. Ha az egész Föld, szárazföldön és vízen egyaránt, számítógépekkel lenne befedve, az IPv6 7×10^{23} IP-címet tenne lehetővé négyzetméterenként. A vegyészhallgatók észre fogják venni, hogy ez a szám nagyobb az Avogadro-számnál. Bár nem az volt a szándék, hogy a Föld felszínén minden molekulának saját IP-címet adjunk, azért nem is járunk annyira messze ettől.

A gyakorlatban a címtartományt nem lehet hatékonyan kihasználni, mint ahogy a telefonszámok címtartományát sem. (A manhattani körzetszám, a 212, majdnem tele van, de a wyomingi (307) majdnem üres.) Az RFC 3194-ben Durand és Huitema kiszámolták, hogy a telefonszámok kiosztását irányadónak véve még a legpesszimistább forgatókönyv szerint is bőven több mint 1000 IP-cím jut a Föld felszínének (szárazföld és víz egyaránt) minden négyzetméterére. Röviden, nem tűnik valószínűnek, hogy a belátható jövőben kifogynánk belőlük.

Tanulságos összehasonlítani az IPv4-fejrészt (5.46. ábra) az IPv6-fejrésszel (5.56. ábra), hogy lássuk, mi maradt ki az IPv6-ból. Az *IHL* mező eltűnt, mert az IPv6-fejrész rögzített hosszúságú. A *Protokoll* mezőt is kivették, mert a *Következő fejrész* mező megmondja, mi jön az utolsó IP-fejléc után (például egy UDP- vagy TCP-szegmens).

A darabolással kapcsolatos összes mezőt eltávolították, mert az IPv6 másképpen közelíti meg a darabolást. Először is minden, az IPv6-hoz igazodó hoszttól elvárjuk, hogy dinamikusan határozza meg a használt datagram méretét. Ez az MTU-felderítési protokoll segítségével lehetséges, amelynek leírását az 5.5.5. szakasz tartalmazza. Röviden: amikor egy hoszt túl nagy IPv6-csomagot küld, az az útválasztó, amely képtelen azt továbbítani, darabolás helyett egy hibaüzenetet küld vissza. Ez az üzenet arra utasítja a hosztot, hogy a jövőben minden, ehhez a címzetthez tartó csomagot tördeljen fel. Az, hogy a hoszt eleve jó méretű csomagokat küld, végül is sokkal hatékonyabb, mint ha

az útválasztók darabolják azokat menet közben. Ezenkívül a minimális csomagméret, amelyet az útválasztónak tovább kell tudnia küldeni, megnövekedett 576-ról 1280 bájtra, hogy az 1024 bájtos adatmezőt és számos fejrészt tegyen lehetővé.

Végül az *Ellenőrző összeg* mező is eltűnt, mert kiszámítása nagyban csökkenti a hatékonyságot. A ma használt hálózatok megbízhatósága és azon tény mellett, hogy az adatkapcsolati és a szállítási rétegeknek rendszerint megvan a maguk ellenőrző összege, egy újabb ellenőrző összeg hozadéka kisebb, mint amekkora romlást okoz a teljesítőképességben. Mindezen funkciók eltávolítása egy egyszerű és nagyszerű hálózati protokollt eredményezett. Az IPv6 ezzel a tervezéssel elérte célját: gyors, mégis rugalmas protokoll lett, bőséges címtartománnyal.

Kiegészítő fejrészek

A hiányzó mezők közül némelyikre azért továbbra is szükség mutatkozott, így az IPv6 bevezette az (opcionális) **kiegészítő fejrész** (**extension header**) fogalmát. Ezek a fejrészek használhatók hatékony módon kódolt külön információ biztosítására. Jelenleg a kiegészítő fejrészeknek hat típusa definiált, ezeket az 5.57. ábra sorolja fel. Mindegyik opcionális, de ha több mint egy van jelen, akkor közvetlenül a rögzített fejléc után kell szerepelniük, és célszerűen a megadott sorrendben.

Némelyik fejrésznek kötött a formátuma; mások változó számú és hosszúságú mezőt tartalmaznak. Ez utóbbiaknál minden tétel egy (*Típus*, *Hossz*, *Érték*) hármasként van kódolva. A *Típus* egy 1 bájtos mező, amely azt mondja meg, hogy melyik opcióról van szó. A *Típus* értékeket úgy választották meg, hogy az első 2 bit megmondja az útválasztónak, mit tegyen, ha nem tudja feldolgozni az opciót. A lehetőségek: átugorni az opciót; eldobni a csomagot; eldobni a csomagot és visszaküldeni egy ICMP-csomagot; valamint eldobni a csomagot és nem küldeni vissza az ICMP-csomagot a többesküldéses címre (hogy egy rossz csomag ne generáljon több millió ICMP-jelentést).

A *Hossz* is egy 1 bájtos mező. Azt mondja meg, milyen hosszú az érték (0 és 255 bájt közt). Az *Érték* 255 bájt erejéig bármilyen kívánt információ lehet.

Az *Ugrás* opciók fejrészt olyan információhoz használják, amelyeket minden, útba eső útválasztónak meg kell vizsgálnia. Eddig egyetlen ilyen opciót definiáltak a 64 KB-nál hosszabb datagramok támogatására. Ennek a fejrésznek a formátumát az 5.58. ábra mutatja. Amikor ezt az opciót használják, akkor a rögzített fejrészben az *Adatmező hossza* mező értékét 0-ra állítják be.

Kiegészítő fejrész	Leírás
Ugrás opciók	Különféle információ az útválasztók számára
Címzetti opciók	További információ a címzett számára
Útválasztás opció	Laza lista a felkeresendő útválasztókról
Darabolás opció	A datagramdarabok kezelése
Hitelesítés opció	Az adó személyazonosságának ellenőrzése
Titkosított biztonsági adatmező	Információ a titkosított tartalomról

5.57. ábra. *Az IPv6 kiegészítő fejrészei*

Következő fejrész	0	194	4
Jumbo adatmező hossza			

5.58. ábra. *Az ugrás kiegészítő fejrész óriás datagramokhoz (jumbogramokhoz)*

Mint minden kiegészítő fejrész, ez is egy olyan bájttal kezdődik, amelyik megmondja, hogy milyen lesz a következő fejrész. Ezt a bájtot egy másik követi, amelyik azt tudatja, hogy hány bájt hosszú az ugrás opció fejrésze, leszámítva az első 8, kötelező bájtot. Minden kiegészítő fejrész így kezdődik.

A következő két bájt jelzi, hogy ez az opció a datagram méretét határozza meg (194-es kód), egy 4 bájtos szám formájában. Az utolsó négy bájt adja meg a datagram méretét. A 65 536 bájtnál kisebb méretek nem megengedettek. Ha ilyen mégis előfordulna, az első útválasztó eldobja a csomagot, és visszaküld egy ICMP-hibaüzenetet. Az ilyen fejrész-kiegészítést használó datagramokat **óriás datagramoknak** vagy **jumbogramoknak** nevezik. A jumbogramok használata a szuperszámítógépes alkalmazások számára fontos, melyeknek gigabájt nagyságrendű adatokat kell hatékonyan átvinni az interneten.

A *Címzetti* opciók fejrészt olyan mezőknek szánták, melyeket csak a címzett csomópontban kell értelmezni. Az IPv6 kezdeti verziójában ide nem definiáltak mást, csak üres opciókat, hogy kitöltsék ezt a fejrészt 8 bájt többszörösére, ezt a fejrészt tehát eleinte nem fogják használni. Azért került bele mégis a protokollba, hogy az új útválasztó- és hosztszoftverek használhassák, ha egy napon valakinek egy címzetti opcióra lenne szüksége.

Az *Útválasztás* opció fejrész egy vagy több útválasztót sorol fel, melyeket a célhoz vezető úton fel kell keresni. Ez nagyon hasonlít az IPv4 laza forrás általi útválasztásához annyiban, hogy minden felsorolt címet sorban végig kell járni, de közben más, nem felsorolt útválasztókat is útba lehet ejteni. Az útválasztás opció fejrészformátumát az 5.59. ábra mutatja.

Az útválasztás opció kiegészítő fejrészének első 4 bájtja négy 1 bájtos egész számot tartalmaz. A *Következő fejrész* és a *Kiegészítő fejrész hossza* mezőket már leírtuk. Az *Útválasztás típusa* mező megadja a fejrész hátralévő részének formátumát. A 0 típus azt jelenti, hogy egy fenntartott 32 bites szó követi az első szót, majd bizonyos számú IPv6-cím következik. A jövőben más típusokat is kitalálhatnak, ha szükség lesz rá. Végül a

Következő fejrész	Kiegészítő fejrész hossza	Útválasztás típusa	Hátralévő szegmensek száma
Az adott típus által meghatározott adatok			

5.59. ábra. *A kiegészítő fejrész útválasztáshoz*

Hátralevő szegmensek száma mező tartja számon, hogy a listában szereplő címek közül még hányat nem látogattunk meg. Ezt minden útválasztó érintése után eggyel csökkentik. Amikor a számláló eléri a 0-t, akkor a csomag már csak magára hagyatkozhat, és nem kap több útmutatást arra nézve, hogy melyik utat kövesse. Ezen a ponton általában olyan közel van a céljához, hogy a legjobb útvonal már egyértelmű.

A *Darabolási* opció fejrésze az IPv4-hez hasonlóan kezeli a darabolást. A fejrészben szerepel a datagramazonosító, a darabszám és egy bit, amelyik arról értesít, hogy jönnek-e még további darabok. Az IPv4-től eltérően az IPv6-ban már csak a forráshoszt darabolhat fel egy csomagot, az útba eső útválasztók nem. Ez ugyan nagy gondolkodásmódbeli törés az eredeti IP-hez képest, de tartja magát az IPv4 gyakorlatához. Ezenfelül egyszerűbbé teszi az útválasztók munkáját, és meggyorsítja az útválasztást. Ahogy fentebb említettük, ha az útválasztó egy túl nagy csomaggal találja magát szemben, akkor eldobja azt, és visszaküld egy ICMP-csomagot a forráshosztnak. Ez az információ teszi lehetővé a forráshoszt számára, hogy ezt a fejrészt használva kisebb részekre darabolja a csomagot, és újra próbálkozzon.

A *Hitelesítés* opció fejrésze egy olyan eljárást biztosít, mellyel a csomag vevője meggyőződhet róla, hogy tényleg a feladó küldte-e a csomagot. A *Titkosított biztonsági adatmező* lehetővé teszi, hogy egy csomag tartalmát úgy titkosítsuk, hogy csak a szándékaink szerinti vevő tudja elolvasni. Ezek a fejrészek titkosító módszereket használnak küldetésük teljesítéséhez. Ezeket a módszereket a 8. fejezetben mutatjuk be.

Ellentmondások

Tudván a nyílt tervezési folyamatról és arról, hogy sok érintett ember makacsul ragaszkodott az álláspontjához, nem okozhat meglepetést, hogy számos, az IPv6 esetében hozott döntés erősen vitatott volt. Ezek közül néhányat összefoglalunk az alábbiakban. A részleteket lásd az RFC-kben.

Már megemlítettük a vitát a címek hosszáról. Az eredmény egy kompromisszum: 16 bájtos rögzített hosszúságú címek.

Szintén csata bontakozott ki az *Ugráskorlát* mező hossza körül. Az egyik tábor nagyon úgy érezte, hogy a maximális ugrásszám (a 8 bites mezőből adódó) 255-re való korlátozása öreg hiba. Végül is, a 32 ugrásból álló utak ma már elterjedtek, és 10 év múlva sokkal hosszabb utak is elterjedhetnek. Ezek az emberek azzal érveltek, hogy egy hatalmas méretű cím használata előrelátó volt, de egy kicsi ugrásszámláló használata rövidlátásra utal. Álláspontjuk szerint a legnagyobb bűn, amit egy számítástechnikus elkövethet, hogy valahol túl kevés bitet hagy meg.

A válasz az volt, hogy minden mező megnövelésére akad érv, de ez felduzzadt fejrészhez vezetne. Egyébként is, az *Ugráskorlát* mező feladata, hogy a csomagok ne kószálhassanak hosszú ideig szerteszét, és 65 535 ugrás túlságosan hosszú. Végül, ahogy az internet növekszik, egyre több és több nagy távolságú összeköttetés fog kiépülni, lehetővé téve, hogy bármely országból bármely más országba legfeljebb fél tucat ugrással eljussunk. Ha 125 ugrásnál több szükséges eljutni a forrástól a megfelelő nemzetközi átjáróikhoz, akkor valami nincs rendben a nemzeti gerinchálózatokkal. A 8 bitet támogatók nyerték meg ezt a csatát.

A másik hevesen vitatott téma a csomagméret volt. A szuperszámítógépes társadalom 64 KB-nál nagyobb csomagokat akart. Amikor egy szuperszámítógép átvitelbe kezd, akkor tényleg dolga van, és nem akarja, hogy 64 KB-onként megszakítsák. A nagy csomagok ellen az az érv merült fel, hogy ha egy 1 MB-os csomag elér egy 1,5 Mb/s-os T1 vonalat, akkor ez a csomag 5 másodpercre lefoglalja a vonalat, és jól észlelhető késleltetést okoz a szintén ezt a vonalat használó interaktív felhasználóknál. Itt kompromisszum született: a rendes csomagokat 64 KB-ra korlátozták, de az ugrás kiegészítő fejrész használható jumbogramok engedélyezésére.

A harmadik forró téma az IPv4 ellenőrző összegének eltávolítása volt. Néhány ember ezt ahhoz hasonlította, mintha egy autóból kiszednénk a fékeket. Ha így teszünk, az autó könnyebb lesz, és gyorsabban haladhat, de ha valami váratlan esemény történik, akkor gond van.

Az ellenőrző összegek elleni érv az volt, hogy bármely alkalmazásnak, amely valóban törődik az adatintegritással, amúgy is kell lennie egy ellenőrző összegének a szállítási rétegben, így túlzás az IP-be is berakni egyet (az adatkapcsolati réteg ellenőrző összegén felül).

A mozgó hosztok is vita tárgyát képezték. Ha egy hordozható számítógép félig körberepüli a világot, működhet-e tovább a célban ugyanazzal az IPv6-címmel, vagy egy hazai és idegen ügynökös sémát kelljen használnia? Néhányan a mozgó hosztok számára kifejezett támogatást akartak beépíteni az IPv6-ba. Ez az erőfeszítés meghiúsult, amikor egyik javaslatban sem tudtak megegyezni.

A legnagyobb csata valószínűleg a biztonság körül dúlt. Mindenki egyetértett abban, hogy szükség van rá. A harc azzal kapcsolatban folyt, hogy hol és hogyan legyen. Először nézzük a hol kérdését. A hálózati rétegbe helyezése mellett az az érv szólt, hogy ekkor ez szabványos szolgáltatássá válik, amelyet minden alkalmazás minden előzetes tervezés nélkül használhat. Az ellene szóló érv az, hogy a valóban titkos alkalmazások a végpontok közötti titkosításnál nem érik be kevesebbel, ahol is a forrásalkalmazás végzi a titkosítást, és a célalkalmazás fejti vissza. Bármivel, ami ennél kevésbé biztonságos, a felhasználó ki van szolgáltatva a hálózati réteg esetlegesen hibás megvalósításának, amelyek fölött nem bír ellenőrzéssel. Erre az érvre az a válasz, hogy ezek az alkalmazások tartózkodhatnak az IP titkosítási tulajdonságainak kihasználásától, és maguk végezhetik el a munkát. Erre pedig az a viszontválasz, hogy azok, akik nem bíznak meg abban, hogy a hálózat jól végezné el ezt, nem akarják megfizetni az ilyen képességű lassú és terjedelmes IP-megvalósítások árát, még ha az ki is van kapcsolva.

A biztonság elhelyezésének másik vetülete ahhoz kapcsolódik, hogy sok (de nem minden) országnak szigorú kiviteli törvényei vannak a titkosításra nézve. Néhány ország, főképpen Franciaország és Irak, belföldön is nagyban korlátozza a használatát, hogy az embereknek ne lehessenek titkaik a rendőrség előtt. Ennek eredményeként semmilyen olyan IP-megvalósítást, amely elég erős titkosító rendszert használ ahhoz, hogy valóban hasznos legyen, nem lehet kivinni az Egyesült Államokból (és sok más országból) a vásárlókhoz világszerte. A legtöbb számítógépes cég erőteljesen ellenzi azt, hogy két szoftverkészletet kelljen karbantartania, egyet belföldi használatra és egyet kivitelre.

Egy ponton nem volt vita, és ez az, hogy senki sem számít arra, hogy egy vasárnap reggel kikapcsolják az IPv4-internetet, és az hétfő reggel IPv6-internetként indul újra. Ehelyett elkülönült IPv6-„szigetek" fognak először átállni, kezdetben alagutakon ke-

resztül kommunikálva, ahogy azt az 5.5.3. fejezetben láthattuk. Ahogy az IPv6-szigetek nőnek, úgy olvadnak össze egyre nagyobb szigetekké. Végül minden sziget össze fog olvadni, és az internet teljesen át fog állni.

Legalábbis ez volt a terv. A bevezetés megmutatta az IPv6 Achilles-sarkát. Továbbra is alig használják, annak ellenére, hogy minden nagy operációs rendszer teljes mértékig támogatja. A legtöbb bevezetés új szituáció, amelyben a hálózati operátor – például egy mobiltelefonos operátor – nagyszámú IP-címet igényel. Számos stratégia került kialakításra az egyszerű átállás biztosítása érdekében. Ezek között van olyan, amely automatikusan beállítja az alagutat, amely átviszi az IPv6-csomagot az IPv4-interneten, illetve olyan, amely lehetővé teszi, hogy a hosztok automatikusan megtalálják az alagút végpontjait. A két protokollkészlettel rendelkező (dual-stack) hosztoknak egyaránt van IPv4 és IPv6 implementációja, így a csomag célcíme alapján kiválaszthatják, hogy melyik protokollt kell használni. Ezek a stratégiák fájdalommentessé teszik az IPv6 alapvető bevezetését, amely elkerülhetetlen lesz, amikor az IPv4-címek elfogynak. További információ az IPv6-tal kapcsolatban Davies [2008] munkájában található.

5.6.4. Az internet vezérlőprotokolljai

Az adattovábbításra használt IP-n kívül az internetnek számos, a hálózati rétegben használt vezérlőprotokollja van, mint például az ICMP, ARP és a DHCP. Ebben a szakaszban sorban mindegyiknek áttekintjük az IPv4-nek megfelelő változatát, mivel ezek az általánosan használt protokollok. Az ICMP és DHCP hasonló változattal rendelkezik az IPv6-hoz is. Az ARP IPv6-megfelelőjét NDP-nek (Neighbour Discovery Protocol – szomszédfelderítési protokoll) nevezik.

ICMP – internetes vezérlőüzenet protokoll

Az internet működését az útválasztók szorosan figyelemmel kísérik. Amikor valami váratlan esemény történik a csomag feldolgozása során egy útválasztóban, ezt az eseményt az **ICMP (Internet Control Message Protocol – internetes vezérlőüzenet protokoll)** segítségével jelenti. Az ICMP-t az internet tesztelésére is használják. Körülbelül egy tucat ICMP-üzenetet definiáltak. A legfontosabbakat az 5.60. ábra sorolja fel.

A CÉL ELÉRHETETLEN üzenetet akkor használják, ha az útválasztó nem tudja megtalálni a célt, vagy egy *DF* bittel rendelkező csomagot nem lehet kézbesíteni, mert egy „kiscsomagos" hálózat az útjába állt.

Az IDŐTÚLLÉPÉS üzenetet akkor küldik, ha egy csomagot azért dobnak el, mert a számlálója elérte a nullát. Ez az esemény annak a tünete, hogy a csomagok hurokba kerültek, hogy hatalmas torlódás van, vagy az időzítő értékét túl alacsonyra állították be.

Ennek a hibaüzenetnek az egyik értelmes használata a **traceroute (útkövetés)** segédprogram, amelyet Van Jacobson fejlesztett ki 1987-ben. A traceroute megtalálja a hoszt és a címzett IP-címe közötti útvonal mentén lévő útválasztókat. Ezt az információt mindenfajta privilegizált hálózati támogatás nélkül találja meg. A módszer egyszerű: csomagok sorozatának elküldése a cél felé, az Élettartam mezőbe írt először 1-es, majd

Üzenet típusa	Leírás
Cél elérhetetlen	A csomagot nem lehetett kézbesíteni
Időtúllépés	Az Élettartam mező elérte a 0-t
Paraméter probléma	Érvénytelen fejrész mező
Forráslefojtás	Lefojtócsomag
Átirányítás	Egy útválasztót tanít meg a földrajzra
Visszhang kérés és visszhang válasz	Annak ellenőrzése, hogy egy gép életben van-e
Időbélyeg kérés/válasz	Ugyanaz, mint a visszhang kérés, csak időbélyeggel
Útválasztó hirdetés/kérelmezés	Egy közeli útválasztó megtalálása

5.60. ábra. *A legfőbb ICMP-üzenettípusok*

2-es, 3-as stb. értékkel. A csomagokban lévő átlépésszámlálók elérik a nullát, ahogy végighaladnak az útvonal mentén lévő egymás utáni útválasztókon. Ezek az útválasztók egyenként IDŐTÚLLÉPÉS üzenetet küldenek vissza a hosztnak. Ezekből az üzenetekből a hoszt meg tudja határozni az útvonal mentén lévő útválasztók IP-címét, illetve statisztikát és időzítéseket tarthat fenn az útvonal egyes részeiről. Ez nem az, amire az IDŐTÚLLÉPÉS üzenetet eredetileg szánták, de talán a valaha alkalmazott leghasznosabb hálózati hibakereső eszköz.

A PARAMÉTERPROBLÉMA üzenet azt jelzi, hogy egy fejrészmezőbe érvénytelen érték került. Ez hibát jelez az adóhoszt IP-szoftverében, vagy esetleg egy, az út során érintett útválasztó szoftverében.

A FORRÁSLEFOJTÁS üzenetet régebben a túl sok csomagot küldő hosztok visszafogására használták. Amikor egy hoszt ilyen üzenetet kapott, le kellett lassítania. Manapság már ritkán használják, mert amikor torlódás következik be, ezek a csomagok csak olajat öntenek a tűzre. Az interneten a torlódáskezelés most nagyrészt a szállítási rétegben történik, és a 6. fejezetben fogjuk tanulmányozni.

Az ÁTIRÁNYÍTÁS üzenetet akkor használják, ha egy útválasztó észreveszi, hogy egy csomag rosszul irányítottnak tűnik. Az útválasztó használja, hogy a küldő hosztot értesítse a valószínű hibáról.

A VISSZHANG KÉRÉS és VISSZHANG VÁLASZ üzeneteket arra használják, hogy meggyőződjenek arról, hogy egy adott hoszt elérhető-e és pillanatnyilag életben van-e. A VISSZHANG KÉRÉS üzenetet megkapva, a címzettnek vissza kell küldenie egy VISSZHANG VÁLASZ üzenetet. Ezeket az üzeneteket a **ping** segédprogram használja, amely ellenőrzi, vajon a hoszt működik-e és elérhető-e az interneten.

Az IDŐBÉLYEG KÉRÉS és IDŐBÉLYEG VÁLASZ üzenetek hasonlók, kivéve, hogy az üzenet érkezési ideje és a válasz indulási ideje is szerepelnek a válaszban. Ezt a tulajdonságot a hálózati teljesítőképesség mérésére használják.

Az ÚTVÁLASZTÓ HIRDETÉS és ÚTVÁLASZTÓ KÉRELMEZÉS üzenetek segítségével a hosztok meg tudják keresni a közeli útválasztókat. A hosztnak meg kell tanulnia legalább egy útválasztó IP-címét ahhoz, hogy csomagokat küldhessen a helyi hálózatnak.

A fentieken kívül más üzeneteket is definiáltak. Az üzenetek online listáját a *www.iana.org/assignments/icmp-parameters* webcímen találhatjuk.

ARP – címfeloldási protokoll

Bár az interneten levő összes gépnek van egy (vagy több) IP-címe, ezeket voltaképpen nem használhatjuk csomagok küldésére, mivel az adatkapcsolati réteg hálózati csatolókártyája (NIC), mint amilyen az Ethernet-kártya is, nem érti az internetcímeket. Az Ethernet esetében minden eddig gyártott csatolókártya egy 48 bites Ethernet-címmel ellátva érkezik. Az Ethernet-kártyák gyártói egy címtartományt igényelnek az IEEE-től, hogy biztosan ne legyen két kártyának ugyanaz a címe (hogy elkerüljék az ütközéseket, ha a két kártya valaha is ugyanazon a LAN-on tűnne fel). A kártyák a 48 bites Ethernet-címekre alapozva küldenek és fogadnak kereteket. Semmit sem tudnak a 32 bites IP-címekről.

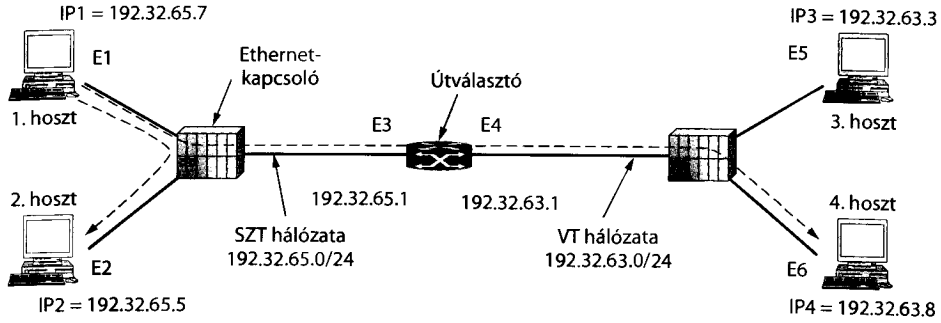
Felmerül a következő kérdés: hogyan képezik le az IP-címeket az olyan adatkapcsolati rétegbeli címekre, mint amilyen az Ethernet-cím is. Ennek a működését az 5.61. ábra példáján keresztül magyarázzuk el, ahol egy kisebb egyetem két /24-es hálózata látható. Két kapcsolt Ethernet-hálózat: egy a Számítástudományi Tanszéken (SZT), 192.32.65.0/24 előtaggal, és egy a Villamosmérnöki Tanszéken (VT), 192.32.63.0/24 előtaggal. A két LAN-t IP-útválasztó köti össze, amelynek egyedi Ethernet-címe van, *E1*-től *E6*-ig jelölve, és egyedi IP-címe az SZT vagy a VT hálózatán.

Kezdésként nézzük meg, hogyan küld az 1. hoszt egyik felhasználója egy csomagot az SZT hálózaton lévő 2. hoszt egy felhasználójának. Tegyük fel, hogy a küldő ismeri a szándéka szerinti vevő címét, valami ilyesmit: *eagle.cs.uni.edu*. Az első lépés a 2., *eagle.cs.uni.edu*-ként ismert hoszt IP-címének megkeresése. Ezt a felkutatást a DNS (Domain Name System – körzetnévkezelő rendszer) végzi, amelyet a 7. fejezetben tanulmányozunk majd. Most csak feltételezzük, hogy a DNS visszaadja a 2. hoszt IP-címét (192.32.65.5).

Az 1. hoszt felsőbb rétegbeli szoftvere most felépít egy csomagot 192.32.65.5-tel a *Célcím* mezőben, és átadja az IP-szoftvernek átvitelre. Az IP-szoftver megnézi a címet, és látja, hogy a cél az SZT (azaz a saját) hálózatán van, de valami módon meg kell találnia a cél Ethernet-címét. Egy megoldás az, hogy legyen egy konfigurációs állomány valahol a rendszerben, amely leképezi az IP-címeket Ethernet-címekre. Ez a megoldás természetesen lehetséges, de a több ezer gépet üzemeltető intézmények számára ezeknek az állományoknak a naprakészen tartása hibára hajlamos, időrabló munka.

Jobb megoldás az, hogy az 1. hoszt kiad egy adatszóró csomagot az Ethernetre, kérdezvén: „Kié a 192.32.65.5-ös IP-cím?" Az adatszórás az SZT Ethernet-hálózatának minden gépéhez megérkezik, és mindegyik ellenőrzi az IP-címét. Csak a 2. hoszt fog válaszolni a saját Ethernet-címével (*E2*-vel). Ily módon az 1. hoszt megtudja, hogy a 192.32.65.5-ös IP-cím az *E2*-es Ethernet-című hoszton van. Ennek a kérdésnek a feltevésére és a válasz megkapására szolgáló protokoll az **ARP (Address Resolution Protocol – címfeloldási protokoll)**. Az interneten majdnem minden gép futtatja, és az RFC 826 definiálja.

Az ARP előnye a konfigurációs állományokkal szemben az egyszerűség. A rendszermenedzsernek nincs sok dolga, csak minden géphez egy IP-címet kell hozzárendelnie és dönteni az alhálózati maszkok felől. Az ARP elvégzi a többit.



Keret	Forrás IP	Forrás Ethernet	Címzett IP	Címzett Ethernet
1-es hoszttól a 2-es hoszt felé az SZT hálózatán	IP1	E1	IP2	E2
1-es hoszttól a 4-es hoszt felé az SZT hálózatán	IP1	E1	IP4	E3
1-es hoszttól a 4-es hoszt felé a VT hálózatán	IP1	E4	IP4	E6

5.61. ábra. *Útválasztóval összekapcsolt két Ethernet LAN*

Ezen a ponton az 1. hoszt IP-szoftvere felépít egy Ethernet-keretet *E2*-nek címezve, belehelyezi a (192.32.65.5-nek címzett) IP-csomagot az adatmezőbe, és ráadja az Ethernetre. A csomag IP- és Ethernet-címét az 5.61. ábra mutatja. A 2. hoszt Ethernet-kártyája észleli a keretet, begyűjti, és egy megszakítást kezdeményez. Az Ethernet-meghajtó kicsomagolja az IP-csomagot az adatmezőből és átadja az IP-szoftvernek, amely látja, hogy helyes a címzés, és feldolgozza.

Változatos optimalizálások lehetségesek az ARP hatékonyabbá tételére. Először is, ha egyszer egy gép futtatta az ARP-t, eltárolja az eredményt arra az esetre, ha rövid idő múlva ugyanazzal a géppel kell kapcsolatba lépnie. A következő alkalommal megtalálja a hozzárendelést a saját gyorstárában, ezáltal szükségtelenné válik a második adatszórás. Sok esetben a 2. hosztnak vissza kell küldenie egy választ, ezáltal arra kényszerül, hogy az adó Ethernet-címének megállapításához futtassa az ARP-t. Ezt az ARP-adatszórást el lehet kerülni, ha az 1. hoszt beleteszi az IP–Ethernet hozzárendelését az ARP-csomagba. Amikor az ARP-csomag megérkezik a 2. hoszthoz, a (192.32.65.7, *E1*) pár bekerül a 2. hoszt gyorstárába majdani felhasználásra. Tulajdonképpen az Etherneten levő összes gép bejegyezheti ezt a leképezést a saját ARP gyorstárába.

Hogy lehetőséget adjunk a leképezések megváltoztatására, amikor egy hoszt új IP-cím használatára van beállítva (de megtartja a régi Ethernet-címet), akkor az ARP gyorstárában lévő bejegyzéseknek pár perc után le kell járniuk. A gyorstárban lévő adatok aktuálisan tartásának és a teljesítőképesség optimalizálásának intelligens módja, ha minden gép adatszórással közli a leképezést beállításkor. Az adatszórás általában a saját IP-címre vonatkozó ARP-kikeresés formájában történik. Erre nem szükséges válasz, de az adatszórás mellékhatása, hogy mindenki ARP-gyorstárában bejegyzés keletkezik vagy frissül a meglévő bejegyzés. Ezt **önkényes ARP**-nek (**gratuitous ARP**) hívják.

Ha a válasz (váratlanul) megérkezik, akkor a két géphez ugyanaz az IP-cím tartozik. A hibát a hálózatkezelőnek meg kell oldani ahhoz, hogy mindkét gép használni tudja a hálózatot.

Most nézzük újra az 5.61. ábrát, csak most az 1. hoszt a 4. (192.32.63.8) hosztnak akar egy csomagot küldeni a VT hálózaton. Az 1. hoszt látja, hogy a címzett IP-címe nem az SZT hálózaton van. Tudja, hogy az összes hálózaton kívüli forgalmat az útválasztóhoz kell küldeni, amelyet **alapértelmezett átjárónak (default gateway)** hívnak. Megegyezés szerint az alapértelmezett átjáró a hálózat legkisebb címe (198.31.65.1). Egy keret útválasztóhoz küldéséhez az 1. hosztnak ismernie kell az útválasztó interfész Ethernet-címét az SZT hálózaton. Ennek felderítéséhez egy ARP-adatszórócsomag kerül kiküldésre a 198.31.65.1 címre, amelyből megismeri az *E3*-at. Ezután elküldi a keretet. Ugyanezt a kikereső módszert használják, ha egy csomagot egy internetútvonalon lévő egyik útválasztótól egy másik útválasztóra küldenek.

Amikor az útválasztó Ethernet NIC-kártyája megkapja a keretet, átadja a csomagot az IP-szoftvernek. A hálózati maszkokból tudja, hogy a csomagot a VT hálózatra kell küldenie, ahol eléri a 4. hosztot. Ha az útválasztó nem tudja a 4. hoszt Ethernet-címét, akkor újra az ARP-t használja. Az 5.61. ábra táblázata megjeleníti a forrás és cél Ethernet- és IP-címet, amelyek megtalálhatók a keretekben, ahogy az SZT és VT hálózaton ezt megvizsgálták. Figyeljük meg, hogy az Ethernet-címek változnak az egyes hálózatokon lévő keretekkel, miközben az IP-címek azonosak maradnak (mivel azok az összekapcsolt hálózatok végpontjait jelzik).

Az is lehet, hogy az 1. hoszt a 4. hosztra küldjön csomagot anélkül, hogy tudná, hogy a 4. hoszt másik hálózaton található. Erre az a megoldás, hogy az útválasztó ARP-ket küld az SZT hálózaton a 4. hosztra és válaszként megadja az *E3* Ethernet-címet. A 4. hoszt erre nem tud közvetlenül válaszolni, mivel nem látja az ARP-kérést (mert az útválasztók nem továbbítják az Ethernet-szintű adatszórás-üzeneteket). Az útválasztó ezután megkapja a 192.32.63.8 címre küldött kereteket, majd továbbítja azokat a VT hálózatra. Ezt a megoldást **helyettesítő ARP-nek (proxy ARP)** nevezik. Ezt olyan speciális helyzetekben használják, ahol a hoszt meg kíván jelenni a hálózaton úgy is, hogy valójában másik hálózaton található. Általános helyzet például egy mobil számítógép, amely azt szeretné, hogy egy másik csomópont vegye fel a neki küldött csomagokat, amikor nem az otthoni hálózaton tartózkodik,

DHCP – dinamikus hosztkonfigurációs protokoll

Az ARP (ahogy más internetprotokollok is) feltételezi, hogy a hosztokat ellátták alapvető információval, mint amilyen például a saját IP-címük. Hogyan kapják meg a hosztok ezt az információt? Be lehet állítani ezt minden számítógépen kézzel is, de ez hosszadalmas és nagy a hibázás lehetősége. Van erre egy jobb módszer. Ezt **DHCP-nek (Dynamic Host Configuration Protocol – dinamikus hosztkonfigurációs protokoll)** hívják.

DHCP alkalmazása esetén minden hálózaton kell lennie egy DHCP-kiszolgálónak, amely a konfigurációért felelős. A számítógépek elindításkor beépített Ethernet- vagy egyéb, a hálózati kártyába ágyazott adatkapcsolati rétegbeli címmel rendelkeznek, de IP-címmel nem. Az ARP-hez hasonlóan a számítógép adatszórással IP-címet kér a hálózaton. Ezt DHCP FELFEDEZÉS csomag küldésével teszi. A csomagnak el kell érnie a DHCP-kiszolgálót. Ha ez a kiszolgáló nem közvetlenül csatlakozik a hálózathoz, akkor az útválasztót úgy állítják be, hogy fogadja a DHCP adatszóró üzeneteket és továbbítja azokat a DHCP-kiszolgáló felé, bárhol is található az.

Amikor a kiszolgáló megkapja a kérést, kioszt egy szabad IP-címet és elküldi azt a hosztnak a DHCP AJÁNLAT csomagban (amely újra továbbításra kerülhet az útválasztón keresztül). Ahhoz, hogy a hosztok ezt IP-cím nélkül is meg tudják tenni, a kiszolgáló a hosztot az Ethernet-címével azonosítja (amelyet a DHCP FELFEDEZÉS csomag tartalmaz).

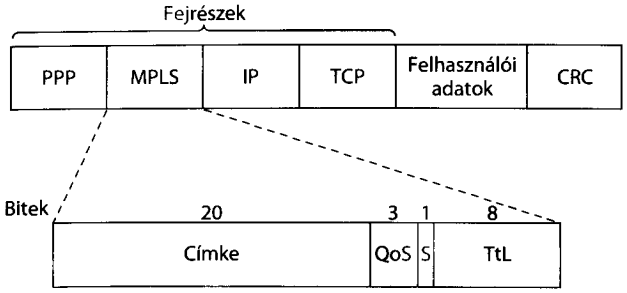
Az IP-címeknek egy külön készletből (pool) történő automatikus kiosztása felveti azt a kérdést, hogy vajon mennyi időre osszanak ki egy IP-címet. Ha egy hoszt elhagyja a hálózatot, és nem adja vissza az IP-címét a DHCP-kiszolgálónak, akkor ez a cím tartósan elveszik. Egy idő után sok cím tűnhet el így. Ennek megelőzésére kioszthatjuk az IP-címeket rögzített időtartamra is. Ezt a módszert **lízingelésnek (leasing)** nevezik. A hosztnak röviddel a lízing lejárta előtt újítást kell kérnie a DHCP-kiszolgálótól. Ha nem sikerül ilyen kérelmet küldenie vagy a kiszolgáló elutasítja a kérelmet, akkor a hoszt nem használhatja tovább a korábban kapott IP-címet.

A DHCP leírását az RFC 2131 és a 2132 tartalmazza. Ezt az interneten széles körben használják mindenféle paraméter beállítására az IP-címkiosztáson felül. A vállalati és otthoni hálózatokon egyaránt használják az ISP-k az eszközök paramétereinek internetkapcsolaton keresztüli beállításához, így az ügyfeleknek nem kell telefonon beszerezniük ezt az információt az ISP-től. A beállított adatokra általános példa a hálózati maszk, az alapértelmezett átjáró IP-címe, valamint a DNS és pontos-idő-szerver IP-címe. A DHCP jórészt teljesen felváltotta a korábbi, korlátozottabb funkcionalitású protokollokat (RARP és BOOTP).

5.6.5. Címkekapcsolás és MPLS

Eddig az internet hálózati rétegében való navigáció során a csomagokat kizárólag az IP-útválasztók által továbbított datagramokként kezeltük. Másik módszer is létezik, amely kezd széles körben elterjedni, különösen az internetszolgáltatók körében, az internetes forgalom hálózaton történő továbbítása érdekében. Ezt a technológiát **többprotokollos címkekapcsolásnak (MultiProtocol Label Switching, MPLS)** nevezik, és nagyon hasonlít a vonalkapcsoláshoz. Annak ellenére, hogy az internetes közösségben sokan erős ellenérzéseket táplálnak az összeköttetés-alapú hálózatokkal szemben, az ötlet mégis újra meg újra felbukkan. Ahogy Yogi Berra mondta, ez olyan, mint a *deja vu* újra és újra. Ugyanakkor az internet és az összeköttetés-alapú hálózatok alapvetően különböző módon építik fel az útvonalaikat, tehát ez a módszer semmiképp sem azonos a hagyományos vonalkapcsolással.

Az MPLS egy címkét rak be minden csomag elé, és a csomag a címke alapján kerül továbbításra, nem a célcím alapján. Mivel a címke egy belső táblázat indexe, a megfelelő kimenő vonal a táblázatból meghatározható. Ennek a módszernek az alkalmazásával a továbbítás nagyon gyors lehet. Főképp ez motiválta az MPLS kialakítását, amely számos különböző (szabadalmaztatott) név alatt fut, mint például a **címkekapcsolás (tag switching)**. Az IETF végül szabványosította az elgondolást. Ezt a protokollt többek kö-



5.62. ábra. *TCP-szegmens átvitele IP, MPLS és PPP segítségével*

zött az RFC 3031 írja le. A fő előny, hogy az útválasztás rugalmas, és a továbbítás megfelel a szolgáltatásminőségnek, valamint gyors.

Az első kérdés, hogy hová tegyük a címkét? Mivel az IP-csomagokat nem virtuális áramkörökhöz tervezték, az IP-fejrész nem tartalmaz mezőt a virtuálisáramkör-azonosítók számára. Ezért új MPLS-fejrészt kell tenni az IP-fejrész elé. Az 5.62. ábra két útválasztó közötti vonalon használatos PPP adatkapcsolati protokoll keretszerkezetét mutatja, beleértve a PPP-, MPLS-, IP- és TCP-fejrészeket is.

Az általános MPLS-fejrész 4 bájt hosszú és 4 mezőből áll. Ezek közül a legfontosabb a *Címke (Label)* mező, amely az indexet tartalmazza. A *QoS* mező a szolgáltatásminőségi osztályt jelzi. Az *S* mező több, egymásra halmozott címkére utal (lásd lejjebb). A *TtL* mező azt mutatja, hogy a csomag még hányszor továbbítható. Ennek értékét minden útválasztó csökkenti, és ha ez eléri a 0-t, akkor a csomagot eldobják. Ez akadályozza meg, hogy egy csomag útválasztási zavar esetén a végtelenségig keringjen.

Az MPLS az IP hálózati protokoll és a PPP adatkapcsolati protokoll közé esik. Nem tartozik igazán a 3. réteghez, mivel a címkeútvonalak kialakítása IP- vagy más hálózatiréteg-címtől függ. Nem is tartozik igazán a 2. réteghez sem, mivel több ugráson át továbbít csomagokat, nem egyetlen adatkapcsolaton. Ezért az MPLS-t 2,5 rétegbeli protokollnak is hívják. Ez szemlélteti, hogy a valós protokollok nem mindig illeszkednek tökéletesen az idealizált rétegezett protokollmodellbe.

Mivel az MPLS-fejrész sem a hálózati réteg csomagnak, sem az adatkapcsolati réteg keretnek nem része, az MPLS nagyban független mindkét rétegtől. Ez többek közt azt is jelenti, hogy olyan MPLS-kapcsolók is készíthetők, melyek IP- és nem-IP-csomagokat egyaránt képesek továbbítani, attól függően, hogy éppen mi érkezik. Innen ered a „többprotokollos" szó az elnevezésben. Az MPLS képes IP-csomagokat nem-IP-hálózatokon is átvinni.

Amikor egy MPLS-címkével kiegészített csomag megérkezik egy **LSR**-hez (**Label Switched Router – címkekapcsolt útválasztó**), a címkét táblázatindexként használják a használandó kimeneti vonal és új címke meghatározásához. Az effajta címkecserélgetést a virtuálisáramkör-alapú hálózatokban is használják, mert a címkéknek csak helyi jelentésük van, és két különböző útválasztó ugyanazon a kimeneti vonalon össze nem tartozó csomagokat is továbbíthat ugyanazon címke használatával. Ahhoz, hogy a címkék a másik oldalon megkülönböztethetők legyenek, minden ugrásnál újra le kell képezni azokat. Ezt a mechanizmust láthattuk már működés közben az 5.3. ábrán. Az MPLS ugyanezt a módszert használja.

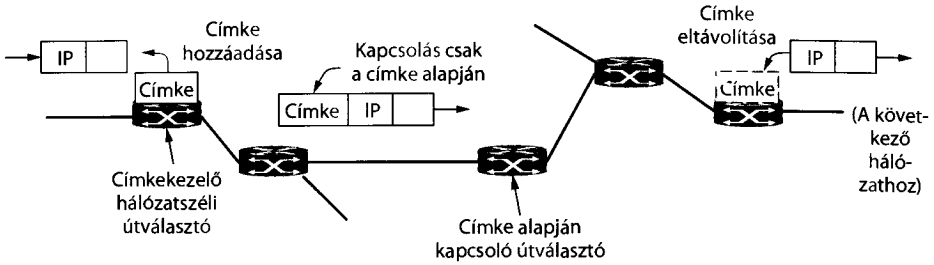
Kis kitérőként megjegyezzük, hogy egyesek különbséget tesznek a *továbbítás* és a *kapcsolás* között. A továbbítás során a célcímet keressük ki egy táblázatból, hogy megtudjuk, hová kell küldeni a csomagot. Erre példa az IP-továbbításhoz használt leghosszabb egyező előtag algoritmus. A kapcsolás ellenben a csomagban található címkét használja indexként a továbbítási táblázathoz. Ez gyorsabb és egyszerűbb. Ezek a definíciók persze még messze nem univerzálisak.

Mivel a legtöbb hoszt és útválasztó nem érti az MPLS-t, meg kell kérdeznünk, hogy mikor és hogyan kerülnek a címkék a csomaghoz. Ez akkor történik, amikor egy IP-csomag eléri az MPLS-hálózat szélét. Az **LER (Label Edge Router – hálózatszéli útválasztó)** megvizsgálja a címzett IP-címét és a többi mezőt, és ez alapján meghatározza, hogy a csomagot melyik MPLS-útvonalon kell továbbítani, és a megfelelő címkét a csomag elejére helyezi. Az MPLS-hálózat ezt a címkét használja a csomag továbbítására. Az MPLS-hálózat másik szélén a címke beteljesíti hivatását, és eltávolításra kerül, újra kinyitva az IP-csomagot a következő hálózat számára. Ezt a folyamatot az 5.63. ábra mutatja. Az MPLS és a hagyományos virtuális áramkörök között az egyik legnagyobb különbség a csoportosítás (aggregation) szintje. Az egyes folyamok minden további nélkül rendelkezhetnek saját címkekészlettel az MPLS-hálózatban. Sokkal gyakoribb azonban, hogy az útválasztók több, egy adott útválasztónál vagy LAN-nál végződő folyamot összefognak, és egy közös címkét használnak hozzájuk. Az egy címkéhez csoportosított folyamokat egyazon **továbbítási ekvivalenciaosztályhoz (Forwarding Equivalence Class, FEC)** tartozónak nevezzük. Ez az osztály nemcsak a csomagok címzettjét, hanem szolgáltatási osztályát is lefedi (a differenciált szolgáltatások értelmében), mert a továbbítás szempontjából az osztály minden csomagját egyformán kezeli.

A hagyományos virtuális áramkörös útválasztásnál nem lehet több, különböző címzetthez tartó útvonalat egyazon virtuálisáramkör-azonosítóval ellátni, mert ekkor a végső célállomásnál nem lehetne őket egymástól megkülönböztetni. MPLS használata esetén a csomagok továbbra is tartalmazzák a célcímet a címkén kívül, így a címkézett útvonal végén a címkefejrészt el lehet távolítani, és a továbbítás a megszokott módon folytatódhat a hálózati rétegbeli célcím felhasználásával.

Valójában az MPLS ennél továbbmegy. Az MPLS egyszerre több szinten is működhet azáltal, hogy több címke kerül a csomag elé. Tételezzük fel például, hogy számos csomag van, különböző címkékkel (mivel a csomagokat eltérően kívánjuk kezelni a hálózatban), amelyeknek ugyanazt az útvonalat kell követniük néhány célhoz. Ahelyett, hogy számos címkekapcsolási útvonalat kellene beállítani (minden címkéhez egyet), beállítható egyetlen útvonal. Amikor a már felcímkézett csomagok elérik az útvonal elejét, másik címke adódik hozzá. Ezt címkék vermének (stack of labels) hívjuk. A legkülső címke irányítja a csomagokat az útvonal mentén, majd az útvonal végén eltávolításra kerül, és előkerülnek az esetleges további címkék, amelyek a csomag további továbbításához használhatók. Az 5.62. ábrán bemutatott *S* bit lehetővé teszi, hogy az útválasztó eltávolítson egy címkét annak kiderítése érdekében, hogy maradtak-e még további címkék. Az *S* bitet 1-re állítják a legalsó címkében, és 0-ra minden egyéb címkében.

A végső kérdés, hogy a címketovábbítási táblázatok hogyan épülnek fel, hogy a csomagok követhessék őket. Az MPLS és a hagyományos virtuális áramkörök között ez az egyik legnagyobb különbség. Ha a felhasználó egy összeköttetést szeretne kiépíteni egy hagyományos virtuálisáramkör-alapú hálózatban, akkor egy kapcsolatfelépítő csomagot



5.63. ábra. *IP-csomag továbbítása MPLS-hálózaton keresztül*

indít útjára a hálózatban, hogy létrehozza az útvonalat és a megfelelő bejegyzéseket a továbbítási táblázatokban. Az MPLS nem vonja be a felhasználókat a beállítási fázisba. Ha a felhasználótól a datagram küldésén kívül bármi mást is elvárnánk, az túl sok meglévő internetszoftvert érintene.

Ehelyett a továbbítási információt az útválasztó és összeköttetés-beállítási protokollokat magában foglaló protokollok adják meg. Ezek a vezérlőprotokollok tisztán elkülönülnek a címketovábbítástól, amely több különböző vezérlőprotokoll használatát teszi lehetővé. Az egyik változat a következőképpen működik. Amikor egy útválasztó elindul, akkor megnézi, hogy mely útvonalak számára lesz ő a végállomás (azaz, hogy mely előtagok tartoznak az interfészeihez). Ezeknek aztán létrehoz egy vagy több FEC-t, mindegyikhez hozzárendel egy címkét, majd átadja a címkéket a szomszédjainak. Ezek aztán beírják a címkéket a továbbítási táblázataikba, és új címkéket küldenek a szomszédjaiknak, míg végül minden útválasztó megtanulja az útvonalat. A megfelelő szolgáltatásminőség érdekében erőforrások is lefoglalhatók az út kiépítése során. Más változatok különböző útvonalakat alakíthatnak ki, mint amilyenek például a forgalomtervezési útvonalak, amelyek a nem használt kapacitást veszik figyelembe, és igény szerint hoznak létre útvonalakat a különféle szolgáltatások támogatásához, mint amilyen például a szolgáltatásminőség.

Bár az MPLS mögött rejlő alapötlet eléggé kézenfekvő, a részletek már rendkívül bonyolultak, számtalan változattal és használati esettel, amelyek aktívan megvalósításra kerülnek. További információért lásd Davie és Farrel [2008], valamint Davie és Rekhter [2000] munkáit.

5.6.6. OSPF – a belső átjáró protokoll

Az imént befejeztük az internet csomagtovábbítási módjainak tárgyalását. Ideje továbblépnünk következő témánkra: az interneten belüli útválasztásra. Ahogy már korábban is említettük, az internet nagyszámú **autonóm rendszerből (AS)** tevődik össze. Minden AS-t más intézmény működtet, általában egy vállalat, egyetem vagy internetszolgáltató (ISP). A saját hálózatukon belül az intézmények saját algoritmust használhatnak a belső útválasztásra, vagy ismertebb nevén a **körzeten belüli útválasztásra (intradomain routing)**. Azonban csak néhány szabványos protokoll vált népszerűvé. Ebben a részben a körzeten belüli útválasztás problémáját fogjuk tanulmányozni, és megtekintjük

a gyakorlatban széles körben használt OSPF-protokollt. A körzeten belüli útválasztó protokollt **belső átjáró protokollnak (interior gateway protocol)** is nevezik. A következő részben a függetlenül működő hálózatok közötti útválasztás problémáját, azaz a **körzetek közötti útválasztást (interdomain routing)** fogjuk tanulmányozni. Ehhez az összes hálózatnak ugyanazt a körzetek közötti útválasztást, avagy **külső átjáró protokollt (exterior gateway protocol)** kell használnia. Az interneten használt protokoll a BGP (**Border Gateway Protocol – határátjáró protokoll**).

A korai belső átjáró protokoll a Bellman–Ford-algoritmusra épülő távolságvektor-alapú protokoll volt, amelyet az ARPANET-ből vettek át. Ennek a manapság használt fő példája a RIP (Routing Information Protocol – útválasztási információ protokoll). Ez kis rendszerekben jól működött, de ahogy a hálózatok nagyobbak lettek, egyre kevésbé volt elfogadható. Szenvedett a végtelenig számolás problémájától és általában a lassú konvergenciától is. E problémák miatt az ARPANET 1979 májusában áttért egy kapcsolatállapot-alapú protokollra, és 1988-ban az IETF elkezdett kidolgozni egy ilyen protokollt a körzeten belüli útválasztáshoz. A protokoll neve **OSPF (Open Shortest Path First – nyílt hozzáférésű, a legrövidebb utat előrevevő protokoll)**, amely 1990-ben szabvány lett. Ez az ISO-szabvány **IS-IS (Intermediate-System to Intermediate-System)** protokollra épül. A közös örökség miatt a két protokoll nagyon hasonlít egymásra. A teljes történet az RFC 2328-ban olvasható. Ezek a domináns körzeten belüli útválasztó protokollok, és a legtöbb útválasztógyártó mindkettőt támogatja. Az OSPF-et főként a vállalati hálózatok, az IS-IS protokollt pedig az ISP-hálózatok használják szélesebb körben. Mi az OSPF működését vázoljuk.

Mivel már sok tapasztalat gyűlt össze más útválasztó protokollokról, az OSFP protokollt tervező csoportnak hosszú listája volt a teljesítendő követelményekről. Először is, az algoritmust a mindenki által hozzáférhető irodalomban kellett publikálni, innen az „O" (Open) az OSPF-ben. Nem felelt volna meg egy olyan egyedi megoldás, amelyet egyetlen vállalat birtokol. Másodszor, az új protokollnak mindenféle távolságmetrikát támogatnia kellett, beleértve a fizikai távolságot, a késleltetést és így tovább. Harmadszor, dinamikus algoritmusnak kellett lennie, méghozzá olyannak, amely a topológia változásaihoz automatikusan és gyorsan alkalmazkodik.

Negyedszerre, és ez új az OSPF-ben, támogatnia kellett a szolgáltatás típusára alapozott útválasztást. Az új protokollnak képesnek kellett arra lennie, hogy a valós idejű forgalmat egy adott útvonalra, a másfajta forgalmat pedig másfelé irányítsa. Az IP-protokollnak van egy *Szolgáltatás típusa* mezője, de semmilyen létező útválasztó protokoll nem használta. Ez a mező az OSPF-ben is szerepelt, de még így sem használta senki, ezért végül eltávolították. Ez a követelmény talán megelőzte korát, ahogy az IETF munkája is megelőzte a differenciált szolgáltatásokat, amely megújította a szolgáltatási osztályokat.

Ötödszörre, és a fentiekkel összefüggésben, az új protokolltól elvárták, hogy több vonalra szétosztva egyenlítse ki a terhelést is. A legtöbb, ezt megelőző protokoll minden csomagot a legjobb útvonalon küldött el, még abban az esetben is, ha létezett két egyformán jó útvonal. A második legjobb útvonalat egyáltalán nem használták. Sok esetben a terhelés több vonalra való szétosztása jobb teljesítőképességet ad.

Hatodszor, a hierarchikus rendszerek támogatására is szükség volt. 1988-ra az internet már olyan nagyra nőtt, hogy egyik útválasztótól sem lehetett elvárni, hogy ismerje az

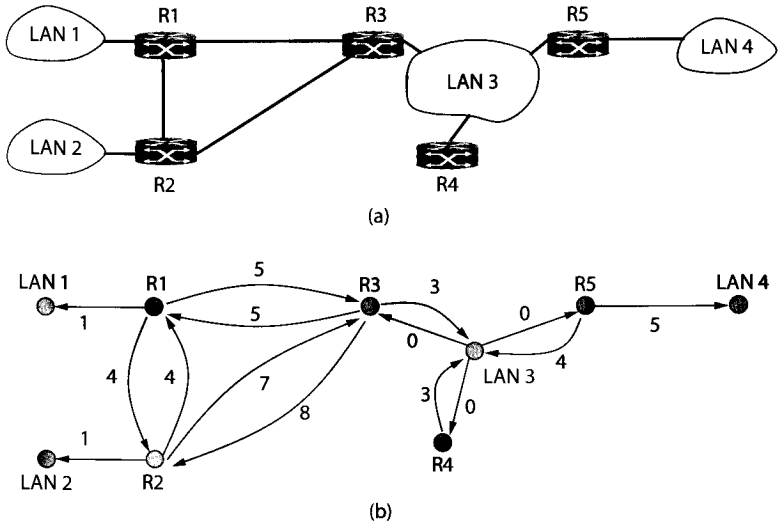
egész topológiát. Az új protokollt úgy kellett megtervezni, hogy erre egyik útválasztónak se legyen szüksége.

Hetedszerre, valami kevés biztonságra is szükség volt, hogy a mókás kedvű hallgatókat meggátolják abban, hogy az útválasztókat hamis útválasztási információ küldésével félrevezessék. Végül valamilyen rendelkezésre is szükség volt az olyan útválasztók kezeléséhez, amelyek alagút típusú átvitelen keresztül kapcsolódtak az internethez. Az előző protokollok ezt nem kezelték jól.

Az OSPF támogatja a kétpontos adatkapcsolatokat (például SONET), valamint az adatszóró hálózatokat (például a legtöbb LAN). Ezenkívül még abban az esetben is támogatja az olyan, több útválasztóval megvalósított hálózatokat, amelyek közül mindegyik közvetlenül kommunikál a többivel (**többszörös hozzáférésű hálózatok – multiaccess networks**), ha azok nem rendelkeznek adatszóró képességgel. A korábbi protokollok nem kezelték jól ezt az esetet.

Az 5.64.(a) ábra egy AS-hálózatra mutat példát. A hosztok kimaradtak, mivel általában nem játszanak szerepet az OSPF-ben, csak az útválasztók és a hálózatok (amelyek tartalmazhatnak hosztokat). Az 5.64.(a) ábrán az útválasztók nagy része kétpontos adatkapcsolattal kapcsolódik egymáshoz, illetve a hálózathoz, a hálózaton lévő hosztok elérése érdekében. Az *R3*, *R4* és *R5* útválasztót azonban egy adatszóró LAN kapcsolja össze, mint amilyen például a kapcsolt Ethernet.

Az OSPF úgy működik, hogy a valódi hálózatok, útválasztók és vonalak összességét egy irányított gráfba képezi le, ahol minden élhez tartozik egy súly (távolság, késleltetés stb.). Két útválasztó közti kétpontos adatkapcsolatot egy élpár jelképez, mindkét irányban egy-egy éllel. Ezek súlyai különbözhetnek. Az adatszóró hálózatot egy csomópont jelképezi, és minden útválasztónak megfelel egy további csomópont. A hálózatnak megfelelő csomópontból az útválasztók felé vezető élek súlya 0. Ezek azonban fontosak, mivel nélkülük nincs a hálózaton átmenő útvonal. Más, csak hosztokat tartalmazó hálózatok csak odamenő éllel rendelkeznek, visszamenővel nem. A struktúra a hosztokhoz vezető utat adja meg, a rajtuk keresztülvezetőt nem.



5.64. ábra. *(a) Egy autonóm rendszer. (b) Az (a) autonóm rendszer egy gráf-reprezentációja*

Az 5.64.(b) ábra mutatja az 5.64.(a) ábra gráf-reprezentációját. Alapjában véve az OSPF azt csinálja, hogy a valódi hálózatot egy ilyen gráfba képezi le, majd a kapcsolatállapot-alapú módszer segítségével kiszámítja a legrövidebb utat minden útválasztótól a többi csomópontig. Elképzelhető több, egyformán rövid útvonal. Ebben az esetben az OSPF megjegyzi a legrövidebb útvonalakat, és a csomagtovábbítás során a forgalmat megosztja ezek között az útvonalak között. Ez segít a terhelés kiegyenlítésében. Ezt **ECMP**-nek (**Equal Cost Multipath – több azonos költségű útvonal**) hívják.

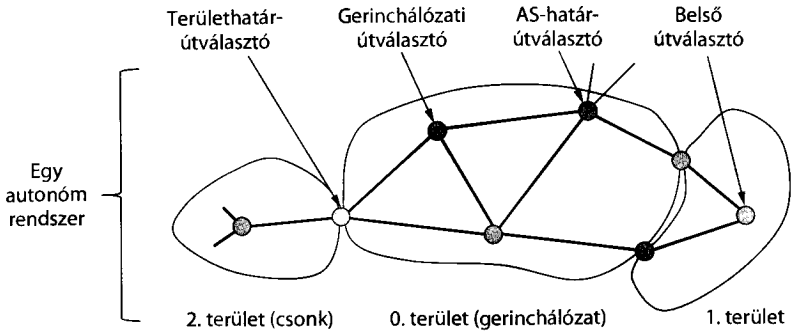
Az interneten sok AS önmagában is nagy, és nem magától értetődő a kezelésük. Az OSPF lehetővé teszi, hogy ezeket megszámozott **területekre** (**area**) osszuk fel, ahol egy terület egy hálózat vagy hálózatok összefüggő halmaza. A területek nem fedhetik át egymást, de nem kell kimerítőnek lenniük, vagyis elképzelhető, hogy néhány útválasztó egyik területhez sem tartozik. A teljes egészében egy adott területen belül elhelyezkedő útválasztót **belső útválasztónak** (**internal router**) nevezik. A terület az egyedülálló hálózat általánosítása. Egy területen kívülről a címzettek láthatók, de a topológia nem. Ez a jellemző segít az útválasztás skálázásában.

Minden AS-nek van egy **gerinchálózati** (**backbone**) területe, amit 0. területnek hívnak. A területen lévő útválasztókat **gerinchálózati útválasztóknak** (**backbone router**) hívják. Minden terület a gerinchálózathoz csatlakozik, esetleg alagutakon keresztül, így az AS bármely területéről bármelyik másik területére el lehet jutni a gerinchálózaton keresztül. Az alagutat a gráfban egy él és egy költség jelképezi. Mint az más területekre is igaz, a gerinchálózat topológiája sem látszik a gerinchálózaton kívülről.

A kettő vagy több területhez kapcsolódó útválasztót **területhatár-útválasztónak** (**area border router**) hívják. Ennek szintén a gerinchálózathoz kell tartoznia. A területhatár-útválasztó feladata az egy területen lévő címzettek összefogása és az összefogás továbbítása a többi terület felé. Ez tartalmazza a költséginformációt, de nem tartalmazza a terület topológiájának minden részletét. A költségadatok továbbításával a többi területen lévő hosztok meg tudják keresni az adott területre történő belépéshez használható legjobb területhatár-útválasztót. Azáltal, hogy topológiai adatok nem kerülnek átadásra, kisebb lesz a forgalom, és a többi terület útválasztói egyszerűbben ki tudják számítani a legrövidebb útvonalat. Ha azonban csak egyetlen területhatár-útválasztó van a területen kívül, akkor még az összefoglalást sem kell átadni. A területen kívüli címzettek útvonalai mindig ezzel kezdődnek „Menj a területhatár-útválasztóhoz". Ezt a területtípust **csonka területnek** (**stub area**) hívják.

Az utolsó útválasztótípus az **AS-határútválasztó** (**AS boundary router**). Ez a más AS-en lévő külső címzettek útvonalát küldi a területre. A külső útvonalak ezután címzettként jelennek meg, amelyek az AS-határútválasztón keresztül érhetők el, valamilyen költséggel. Egy külső útvonal egy vagy több AS-határútválasztón keresztül léphet be. Az AS-ek, a területek és a különböző típusú útválasztók közötti kapcsolatot az 5.65. ábra mutatja. Egy útválasztó több szerepet is betölthet, például egy határútválasztó lehet gerinchálózati útválasztó is.

Rendes működés közben az egy területen lévő útválasztók azonos kapcsolatállapot-alapú adatbázissal rendelkeznek, és ugyanazt a legrövidebb útvonal algoritmust futtatják. Fő feladatuk a legrövidebb útvonal kiszámítása saját maga és a teljes AS-ben lévő többi útválasztó és hálózat között. A területhatár-útválasztónak az összes hozzá csatla-



5.65. ábra. *Az AS-ek, gerinchálózatok és területek viszonya az OSPF-ben*

kozó terület adatbázisára szüksége van, és mindegyik területre függetlenül kell lefuttatnia a legrövidebb útvonal algoritmust.

Ha a forrás és cél ugyanazon a területen található, a legjobb területen belüli útvonal (amely teljes egészében a területen belül vezet) kerül kiválasztásra. Különböző területen lévő forrás és cél esetén három részre bontható az útvonal: a forrástól a gerinchálózatig, a gerinchálózaton keresztül a célterületig, majd a célterületen belül a célig. Ez az algoritmus csillag alakú elrendezést kényszerít az OSPF-re, ahol a gerinchálózat a csillag közepe és a többi terület pedig a csillag ága. Mivel az algoritmus a legkisebb költségű útvonalat választja, a hálózatok különböző részeiben lévő útválasztók különböző területhatár útválasztókat használhatnak a gerinchálózatra, illetve a célterületre való belépéshez. A csomagokat úgy irányítjuk a forrástól a célig, „ahogy vannak". Nem ágyazzuk be azokat, és nem visszük keresztül alagutakon, hacsak nem egy olyan területre mennek, amelyet a gerinchálózattal csak egy alagút köt össze. A külső célhoz vezető útvonalak szükség esetén tartalmazhatják a külső útvonalon lévő AS-határútválasztó külső költségét, vagy csak az AS belső költségét.

Amikor egy útválasztó elindul, HELLO üzeneteket küld minden kétpontos vonalán, és ezeket többesküldéssel elküldi a LAN-okon az összes többi útválasztóból álló csoportnak is. A válaszokból minden útválasztó megtudja, hogy kik a szomszédjai. Az ugyanazon a LAN-on lévő útválasztók mind szomszédok.

Az OSPF azáltal működik, hogy információt cserél a határos vagy összefüggő útválasztók között, amelyek nem ugyanazok, mint a szomszédos útválasztók. Nem hatékony ugyanis, hogy egy LAN-on minden útválasztó minden másikhoz beszéljen. Hogy ezt a szituációt elkerüljék, az egyik útválasztót megválasztják **kijelölt útválasztónak** (**designated router**). Ezt nevezik az összes többi útválasztóval **határos útválasztónak (adjacent router)** vagy **összefüggőnek**, és ez cserél velük információt. Ez az útválasztó lényegében egy olyan csomópont, amely a LAN-t képviseli. Az olyan szomszédos útválasztók, amelyek egymással nem függenek össze, egymás közt nem cserélnek információt. Mindig naprakészen tartanak egy tartalék kijelölt útválasztót is, hogy az átállást megkönnyítsék, amennyiben az elsődleges kijelölt útválasztó összeomolna, és azonnali cserére szorulna.

Rendes működés közben minden útválasztó periodikusan eláraszt KAPCSOLATÁLLAPOT FRISSÍTÉSE üzenetekkel minden vele összefüggő útválasztót. Ez az üzenet

Üzenet típusa	Leírás
Hello	A szomszédok felderítésére használatos
Kapcsolatállapot frissítése	Az adó költségeit küldi el a szomszédainak
Kapcsolatállapot nyugtázása	Nyugtázza a kapcsolatállapot frissítését
Adatbázis leírása	Bejelenti, hogy az adónak milyen frissítései vannak
Kapcsolatállapot kérése	Információt kér a partnertől

5.66. ábra. *Az OSPF öt üzenettípusa*

megadja annak állapotát, és biztosítja a topológiai adatbázisban használt költségeket. Az elárasztási üzeneteket nyugtázzák, hogy megbízhatók legyenek. Minden üzenetnek van egy sorszáma, így az útválasztók láthatják, hogy egy bejövő KAPCSOLATÁLLAPOT FRISSÍTÉSE régebbi vagy újabb annál, mint amelyik náluk van. Az útválasztók akkor is elküldik ezeket az üzeneteket, amikor egy vonal tönkremegy vagy megjavul, vagy megváltozik a költsége.

Az ADATBÁZIS LEÍRÁSA üzenetek megadják minden, a küldő által jelenleg birtokolt kapcsolatállapot bejegyzés sorszámát. A vevő a saját értékeit az adóéval összehasonlítva eldöntheti, kinek van a legújabb értéke. Ezeket az üzeneteket egy vonal megjavításakor használják.

Bármely partner kérhet kapcsolatállapot-információt a másiktól a KAPCSOLATÁLLAPOT KÉRÉSE üzeneteket használva. Ennek az algoritmusnak a tovaterjedő hatása az, hogy minden összefüggő útválasztó pár ellenőrzi, hogy kinek vannak a legújabb adatai, és ily módon az új információ elterjed a területen belül. Mindezeket az üzeneteket nyers IP-csomagokként küldik el. Az öt üzenettípus összefoglalása látható az 5.66. ábrán.

Végre összerakhatjuk a darabokat. Az elárasztást használva minden útválasztó informálja a területén belüli összes többi útválasztót a hozzájuk menő saját adatkapcsolatairól és hálózatairól, valamint ezeknek az adatkapcsolatoknak a költségéről. Ez az információ lehetővé teszi mindegyik útválasztó számára, hogy összeállítsa a területe vagy területei gráfját, és kiszámítsa a legrövidebb utat. Ezt a gerinchálózati terület is elvégzi. Ezenkívül a gerinchálózati útválasztók a területhatár-útválasztóktól is elfogadnak információt, hogy kiszámolják a legjobb útvonalat minden gerinchálózati útválasztótól minden más útválasztóig. Ezt az információt visszafelé is elterjesztik a területhatár-útválasztókhoz, amelyek kihirdetik ezt a körzeteikben. Ezt az információt használva egy útválasztó, amely egy területek közti csomagot készül küldeni, kiválaszthatja a legjobb kijáratot a gerinchálózat felé.

5.6.7. BGP – a külső átjáró protokoll

Az egyetlen AS-en belül általánosan használt protokoll az OSPF és az IS-IS. Az AS-ek között egy másik protokollt, a **BGP**-t (**Border Gateway Protocol – határátjáró protokoll**) használják. Azért van másfajta protokollra szükség, mert a körzeten belüli protokoll és a körzetek közötti protokoll célja nem ugyanaz. A körzeten belüli protokollnak

csak annyi a dolga, hogy a csomagokat a lehető leghatékonyabban mozgassa a forrás és a cél közt. Nem kell törődnie a politikával.

A körzetek közötti protokolloknak viszont sokat főhet a fejük a politika miatt [Metz, 2001]. Például egy vállalati AS-nek szüksége lehet arra a képességre, hogy csomagokat tudjon küldeni az internet tetszőleges helyére, illetve csomagokat tudjon fogadni az internet tetszőleges helyéről. Viszont nem biztos, hogy hajlandó egy idegen AS-ben keletkezett és egy másik idegen AS-be tartó csomagot szállítani, még ha a saját AS a legrövidebb úton is lenne a két idegen AS közt. („Ez az ő problémájuk, nem a mienk.") Másfelől viszont hajlandó lehet átmenő forgalmat továbbítani a szomszédainak vagy esetleg más olyan AS-eknek is, amelyek fizettek ezért a szolgáltatásért. A telefontársaságok például boldogan működnének szolgáltatóként az ügyfeleik számára, de mások számára nem. Általában a külső átjáró protokollokat, így a BGP-t is úgy tervezték, hogy lehetővé tegyenek sokfajta útválasztó politikát, amelyeket az AS-ek közötti forgalom kényszerít ki.

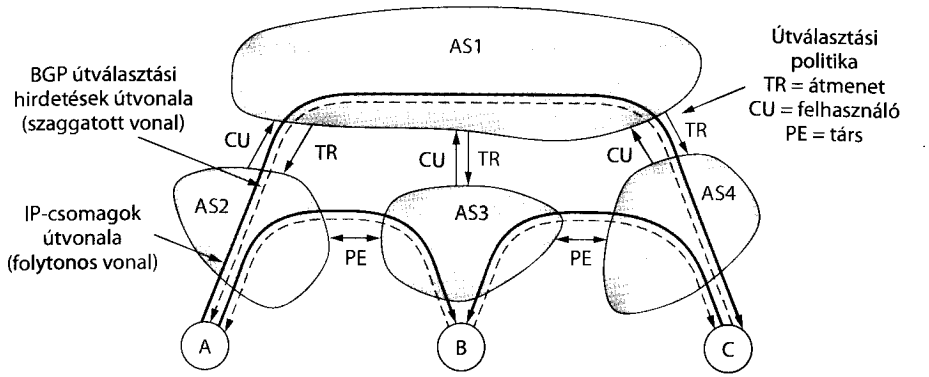
A tipikus politikák világpolitikai, biztonsági vagy gazdasági megfontolásokat foglalhatnak magukban. Egy pár példa az útválasztás lehetséges korlátozására az USA-ban:

1. Ne menjen át kereskedelmi forgalom az oktatási hálózaton.
2. Sose küldjük a Pentagontól jövő forgalmat Irakon keresztül menő útvonalon.
3. Használjuk a TeliaSonert a Verizon helyett, mert az olcsóbb.
4. Ne használjuk az AT&T-t Ausztráliában, mivel rossz a teljesítőképessége.
5. Az Apple-nél kezdődő vagy végződő forgalom ne menjen át a Google-on.

Ahogy a listából látható, az útválasztási politikák meglehetősen egyediek lehetnek. Ezek gyakran védettek, mivel érzékeny üzleti adatokat tartalmaznak. Lehet azonban olyan mintákat találni, amelyek a fenti vállalat érvelését tükrözik, és amelyeket gyakran használnak kiindulási pontként.

Az útválasztási politika megvalósítása úgy történik, hogy a rendszer eldönti, hogy melyik forgalom melyik AS-ek közötti vonalon menjen át. Egy általános politika, hogy az ügyfél ISP-je fizet a másik ISP-nek, hogy eljuttassa a csomagokat tetszőleges más célhoz az interneten keresztül, illetve hogy a más cél által küldött csomagokat fogadja. Az ügyfél ISP **átmenő szolgáltatást (transit service)** vásárol a szolgáltató ISP-től. Ez éppen olyan, mint amikor az otthoni ügyfél internet-hozzáférést vásárol egy internetszolgáltatótól. Ahhoz, hogy ez működjön, a szolgáltató ISP-nek hirdetnie kell az interneten lévő célok útvonalát az ügyfél felé az őket összekapcsoló útvonalakon. Ily módon az ügyfél tetszőleges célhely csomagküldési útvonalának birtokában lesz. Viszont az ügyfélnek csak a saját hálózatán lévő célok útvonalait kell a szolgáltató ISP felé hirdetnie. Ezáltal a szolgáltató ISP csak az e címekre irányuló forgalmat küldi el az ügyfélnek. Az ügyfél nem akarja kezelni a többi célhoz küldött forgalmat.

Az átmenő szolgáltatásra az 5.67. ábra mutat példát. Itt négy AS van összekötve. Az összeköttetés gyakran **IXP**-n (**Internet eXchange Point – internetkapcsolódási pont**) lévő adatkapcsolaton történik. Ez olyan eszköz, amelyhez számos ISP rendelkezik adat-



5.67. ábra. *Útválasztási politikák négy AS között*

kapcsolattal más ISP-hez való kapcsolódás céljából. Az *AS2*, *AS3* és *AS4* az *AS1* ügyfelei, és átmenő szolgáltatást vásárolnak az *AS1*-től. Amikor az *A* forrás a *C* felé küld, akkor a csomagok az *AS2*-től az *AS1*-re, majd végül az *AS4*-re mennek. Az útválasztási hirdetések a csomagokkal ellenkező irányba haladnak. Az *AS4* a *C*-t célként hirdeti az átmenő szolgáltatójához, az *AS1* felé, hogy a források a *C*-t elérhessék az *AS1*-en keresztül. Ezután az *AS1* hirdeti a *C*-hez vezető útvonalat a többi ügyfelének, az *AS2*-t is beleértve, hogy tudják, hogy a *C*-nek menő forgalmat az *AS1*-en keresztül küldhetik.

Az 5.67. ábrán az összes többi AS átmenő szolgáltatást vesz az *AS1*-től. Ez kapcsolódást biztosít számukra, így az internet tetszőleges hosztjával kapcsolatba léphetnek. Ezért a privilégiumért azonban fizetniük kell. Tételezzük fel, hogy az *AS2* és *AS3* nagy forgalmat bonyolít le egymással. Feltéve, hogy a két hálózat már csatlakoztatva van, más politikát is használhatnak, ha akarnak – ingyen küldhetik a forgalmat egymásnak. Ez csökkenti annak a forgalomnak a mennyiségét, amelyet az *AS1*-nek kell helyettük kézbesítenie, ezáltal remélhetőleg a számla összege is csökken. Ezt a politikát hívjuk **fizetség nélküli szolgáltatásnak (peering)**.

A fizetség nélküli szolgáltatás megvalósításához a két AS útválasztási hirdetést küld egymásnak a hálózatukon lévő címekről. Ezáltal az *AS2* az *AS3*-nak csomagokat tud küldeni *A*-ból *B*-be, és fordítva. A fizetség nélküli szolgáltatás azonban nem tranzitív. Az 5.67. ábrán az *AS3* és *AS4* szintén egymás partnere a fizetség nélküli szolgáltatásban. Ez a fizetség nélküli szolgáltatás lehetővé teszi, hogy a *C*-ből *B*-be irányuló csomagokat közvetlenül az *AS4*-nek lehessen küldeni. Mi történik, ha *C* küld csomagot az *A*-nak? Az *AS3* csak egy B-hez menő útvonalat hirdet *AS4* felé, *A* felé menő útvonalát nem hirdeti. Ennek következtében nem kerül át az *AS4*-től az *AS3*-hoz menő forgalom az *AS2*-höz, annak ellenére, hogy fizikai útvonal létezik közöttük. Ez a korlátozás pontosan az, amit az *AS3* akar. Partnerei az *AS4*-nek a forgalom cseréjében, de az *AS4*-től jövő forgalmat az internet többi részére nem kívánja átvinni, mivel az nem fizet ezért. Ehelyett az *AS4* átmenő szolgáltatást kap az *AS1*-től, és az *AS1* fogja elszállítani a csomagokat a *C*-ből az *A*-ba.

Most, az átmenő szolgáltatás és a fizetség nélküli szolgáltatás ismeretében, azt is láthatjuk, hogy az *A*, *B* és *C* rendelkezik átmenő szolgáltatási megegyezéssel. Például az *A*-nak internet-hozzáférést kell vennie az *AS2*-től. Az *A* lehet egyetlen otthoni számítógép, vagy több LAN-ból álló vállalati hálózat. Nincs azonban szüksége BGP-re, mivel

ez egy **csonka hálózat** (**stub network**), amely az internet többi részéhez egyetlen adatkapcsolattal csatlakozik. Így a hálózaton kívülre címzett csomagokat csak az *AS2* adatkapcsolatán küldheti. Másfelé nem tud küldeni. Ez az útvonal egyszerűen kialakítható egy alapértelmezett útvonal beállításával. Emiatt nem látjuk az *A*-t, *B*-t és *C*-t a körzetek közötti útválasztásban résztvevő AS-ként.

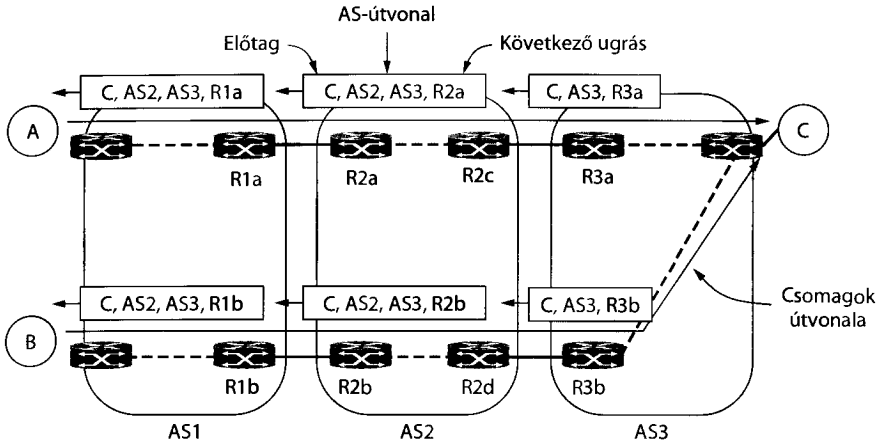
Más részről néhány vállalati hálózat több internetszolgáltatóhoz is kapcsolódik. Ez a technika javítja a megbízhatóságot, mivel ha az egyik ISP-n keresztülmenő útvonal megszakad, akkor a vállalat használhatja a másik ISP-n át vezető útvonalat. Ezt a technikát **többplatformúságnak** (**multihoming**) nevezik. Ebben az esetben a vállalati hálózat valószínűleg körzetek közötti útválasztó protokollt futtat (például a BGP-t), hogy megmondja a többi AS-nek, hogy melyik címet melyik ISP adatkapcsolatán keresztül kell elérni.

Ennek az átmenő fizetség nélküli szolgáltatási politikáknak számos változata lehetséges, de ezek már szemléltették, hogy az üzleti kapcsolatok és az útvonalhirdetések helyének szabályozása hogyan valósíthat meg különböző politikákat. Ezután részletesebben megnézzük, hogy a BGP-t futtató útválasztók hogyan hirdetik útvonalaikat egymásnak, és hogyan választják ki az útvonalat a csomagok továbbításához.

A BGP alapvetően távolságvektor-alapú protokoll, de meglehetősen eltér a legtöbb körzeten belüli távolságvektor-alapú protokolltól, mint amilyen például az RIP. Már láthattuk, hogy ezt a politikát a minimális távolság meghatározása helyett arra használják, hogy a használt útvonalakat feljegyezze. Másik nagy különbség az, hogy az egyes célokhoz menő útvonalak költségének karbantartása helyett minden BGP-útválasztó pontosan nyomon követi a használt útvonalat. Ez az **útvonalvektor protokoll** (**path vector protocol**). Az útvonal a következőket tartalmazza: a következő ugrás útválasztója (ami lehet az ISP másik oldalán, nem a szomszédos oldalon), és az AS-ek sorozata vagy **AS-útvonal**, amelyet az út követ (fordított sorrendben megadva). És végül, a BGP-útválasztók páronként TCP-összeköttetést létrehozva kommunikálnak egymással. Az ilyen működés megbízható kommunikációt biztosít és annak a hálózatnak az összes részletét is elrejti, amelyiken áthalad.

Az 5.68. ábra bemutatja a BGP-útvonalak hirdetésének módját. Három AS van, a középső átmenő forgalmat biztosít a bal és jobb oldali ISP számára. A *C* előtag felé menő útvonalhirdetés az *AS3*-mal kezdődik. Amikor ezt az ábra tetején lévő *R2c* adatkapcsolaton keresztül terjesztik, akkor az AS útvonala egyszerűen az *AS3* és a következő ugrásútválasztó, az *R3a*. Az alsó részen ugyanaz az AS-útvonal, de a következő ugrás különbözik, mivel másik adatkapcsolaton halad keresztül. A hirdetés terjesztése folytatódik és átmegy az *AS1* határán. Az *R1a* útválasztónál, az ábra felső részén, az AS-útvonal az *AS2*, *AS3*, és a következő ugrás az *R2a*.

Ha az út tartalmazza a teljes útvonalat, a fogadó útválasztó egyszerűen észlelni tudja, és fel tudja törni a hurkokat. A szabály az, hogy minden útválasztó, amely az AS-en kívülre küld, az útvonal elé fűzi a saját AS-számát. (Ezért van a lista fordított sorrendben). Amikor az útválasztó megkap egy útvonalat, megnézi, hogy a saját AS-száma szerepel-e már az AS-útvonalon. Ha igen, akkor hurkot észlelt és a hirdetést eldobja. Csak az 1990-es évek végén vették észre azonban, hogy mindezen elővigyázatosság ellenére a BGP a végtelenig számolás problémájától szenved [Labovitz és mások, 2001]. Nincsenek sokáig fennálló hurkok, de az útvonalakon lehetnek átmeneti hurkok, illetve lassú lehet a konvergencia.



5.68. ábra. *BGP-útvonalhirdetések terjesztése*

Elég kezdetleges módszer az útvonal megadása az AS-ek listájával. Egy AS lehet egy kisvállalat vagy egy nemzetközi gerinchálózat. Ez az útból nem derül ki. A BGP meg sem próbálja kideríteni, mivel a különböző AS-ek különböző körzeten belüli protokollt használhatnak, amelyek költsége nem hasonlítható össze. Még ha össze is tudná ezeket hasonlítani, elképzelhető, hogy néhány AS nem akarja elárulni a belső mértékeit. Ez az egyik különbség a körzetek közötti és a körzeten belüli protokollok között.

Eddig láthattuk, hogy az útvonalhirdetés hogyan megy keresztül két ISP közötti adatkapcsolaton. A BGP-útvonalakat azonban valahogy terjeszteni kell az ISP két oldala között, hogy elküldhetők legyenek a következő ISP-nek. Ezt a feladatot kezelheti a körzeten belüli protokoll, de mivel a BGP nagyon jól méretezhető nagy hálózatokhoz, gyakran használják a BGP egy változatát erre a funkcióra. Ennek a neve **iBGP** (**internal BGP – belső BGP**), hogy meg lehessen különböztetni a normál BGP-től, az **eBGP**-től (**external BGP – külső BGP**).

Az útvonalak ISP-n belüli terjesztésének szabálya szerint az ISP határán lévő minden útválasztó megtanulja az összes többi határútválasztó által látott útvonalat a konzisztencia érdekében. Ha az ISP egyik határútválasztója megtanulja a 128.208.0.0/16 előtagot, akkor az összes többi útválasztó is megtanulja azt. Az előtag ezután az ISP összes részéről elérhető, függetlenül attól, hogy a csomagok hogyan lépnek be az ISP-be más AS-ekből.

Ezt a terjesztést nem mutattuk be az 5.68. ábrán a zűrzavar elkerülése érdekében, de például az *R2b* útválasztó tudni fogja, hogy a *C*-t felül az *R2c* útválasztón keresztül, alul pedig az *R2d* útválasztón keresztül tudja elérni. A következő ugrás frissítésre kerül, amint az útválasztó keresztülmegy az ISP-n, így az ISP távoli oldalán lévő útválasztók tudni fogják, hogy melyik útválasztót kell használni az ISP-ből való kilépéshez a másik oldalon. Ez a legbaloldalibb útvonalakon látható, amelyben a következő ugrás ugyanazon az ISP-n belüli útválasztóra mutat, és nem a következő ISP-ben lévő útválasztóra.

Ezután leírhatjuk a hiányzó kulcsrészletet, amely azt mutatja meg, hogy a BGP-útválasztók hogyan választják ki az egyes célokhoz használandó útvonalat. Minden BGP-útválasztó megtanulhatja egy adott cél útvonalát attól az útválasztótól, amelyhez a követ-

kező ISP-nél csatlakozik, illetve az összes többi határ útválasztótól (amelyek különböző útvonalat hallottak az útválasztóktól, amelyekhez a többi ISP-nél csatlakoznak). Minden útválasztónak el kell döntenie, hogy az útvonalhalmaz melyik útvonalát érdemes használnia. A végső válasz az, hogy az ISP dolga a preferált útvonal kiválasztásához valamilyen politika kidolgozása. Ez a magyarázat azonban nagyon általános és egyáltalán nem kielégítő, így azért legalább néhány stratégiát leírunk.

Az első stratégia, hogy a fizetség nélküli szolgáltatást (peering) nyújtó hálózatokon keresztülhaladó útvonalaknak precedenciája van az átmenő szolgáltatást nyújtó (transit) hálózatokon keresztülmenő útvonalakkal szemben. Az előbbi útvonalak ingyenesek, az utóbbiakért pedig fizetni kell. Hasonló stratégia, hogy az ügyfélútvonalak kapják a legnagyobb preferenciát. Ez az egyetlen jó megoldás arra, hogy a forgalmat közvetlenül a fizető ügyfeleknek továbbítsák.

Egy másik stratégia esetén az alapértelmezett szabály az, hogy a rövidebb AS jobb. Ez azonban vitatható, mivel az AS tetszőleges méretű hálózat lehet, így elképzelhető, hogy három kis AS-en keresztülmenő útvonal rövidebb, mint az egy nagy AS-en keresztülmenő útvonal. Normális esetben azonban a rövidebb a jobb, és általánosan ez a szabály a döntő érvényű.

Az utolsó stratégia az ISP-n belüli legkisebb költségű útvonal kiválasztása. Ez az 5.68. ábrán ábrázolt stratégia. Az *A*-ból a *C*-be küldött csomagok az *AS1*-ből a felső, *R1a* útválasztón keresztül lépnek ki. A *B*-ből küldött csomagok az alsó *R1b* útválasztón keresztül lépnek ki. Ennek oka, hogy az *A* és *B* egyaránt a legkisebb költségű útvonalat, vagy az *AS1*-ből kivezető leggyorsabb útvonalat választja. Mivel ezek az ISP különböző részein találhatók, a két csomópont leggyorsabb kilépési pontja különbözik. Ugyanez történik az *AS2*-n keresztülhaladó csomagok esetén. Utolsó lépésként az *AS3*-nak a *B*-ből jövő csomagokat a saját hálózatán kell keresztülvinnie.

Ezt a stratégiát **korai kilépésnek (early exit)** vagy **forró krumpli útválasztásnak (hot-potato routing)** hívják. Ennek különös mellékhatása, hogy az útvonalakat aszimmetrikussá teszi. Vegyük például azt az útvonalat, amelyet a *C*-ből *B*-be visszaküldött csomag megtesz. A csomag gyorsan kilép az *AS3*-ból a felső útválasztón, az erőforrások pazarlásának elkerülése érdekében. Ehhez hasonlóan a felső részen marad, amikor a lehető leggyorsabban átmegy az *AS2*-n az *AS1* felé, majd a csomag hosszabb utat tesz meg az *AS1*-ben. Ez a *B*-ből *C*-be küldött csomag útvonalának tükörképe.

A fenti leírásból ki kell tűnnie, hogy minden BGP-útválasztó az ismert lehetőségek közül választja ki a saját legjobb útvonalát. Nem az a helyzet, mint amit naivan feltételeznénk, hogy a BGP a követendő útvonalat AS-szinten választja, az OSPF pedig az egyes AS-eken belül választ. A BGP és a belső átjáró protokoll mélyebben integrált. Ez azt jelenti, hogy például a BGP meg tudja találni a legjobb kilépési pontot az egyik ISP-től a következőig, és ez a pont az ISP-ken át változik, mint ahogy a forró krumpli politika esetén is. Ez azt is jelenti, hogy egy AS különböző részein lévő BGP-útválasztók különböző AS-útvonalakat választhatnak ugyanazon cél eléréséhez. Az ISP-nek körültekintően kell eljárnia a BGP-útválasztók beállításánál, hogy kompatibilis útvonalakat válasszanak ezen szabadság megtartása mellett, de ez megoldható a gyakorlatban.

Ezzel azonban még csak a BGP felszínét érintettük. További információt a BGP 4-es verziójával kapcsolatban az RFC 4271 és a kapcsolódó RFC-k tartalmaznak. Az összetettség azonban nagyban a politikából adódik, amit a BGP-protokoll specifikációja nem tartalmaz.

5.6.8. Többesküldés az interneten

A rendes IP-kommunikáció egy adó és egy vevő közt zajlik. Néhány alkalmazás esetében azonban hasznos, ha egy folyamat képes nagyszámú vevőnek egyszerre küldeni. Ilyen például az élő sportesemény közvetítése sok nézőnek, programfrissítések küldése tükrözött (replicated) kiszolgálókra, illetve digitális konferenciahívások (vagyis többrésztvevős hívások) kezelése.

Az IP támogatja az egy-több típusú kommunikációt, illetve a többesküldést, D osztályú címek használatával. Minden D osztályú cím egy hosztcsoportot azonosít. Huszonnyolc bit használható a csoportok azonosítására, így egy időben több mint 250 millió csoport lehet jelen. Amikor egy folyamat egy D osztályú címre küld egy csomagot, best-effort típusú kísérlet történik arra, hogy a megcímzett csoport minden tagjának kézbesítsék a csomagot, de erre semmi garancia nincs. Előfordulhat, hogy egyes tagok esetleg nem kapják meg a csomagot.

A 224.0.0.0/24 IP-címtartomány többesküldésre van fenntartva a helyi hálózaton. Ebben az esetben nincs szükség útválasztó protokollra. A csomagok többesküldése a LAN-on egyszerű adatszórással megoldható többesküldéses cím használatával. A LAN-on lévő összes hoszt megkapja az adatszórás üzenetet, és a csoporthoz tartozó hosztok feldolgozzák a csomagot. Az útválasztók nem továbbítják a csomagot a LAN-on kívülre. Néhány példa a helyi többesküldéses címre:

224.0.0.1	Az egy LAN-on levő összes rendszer.
224.0.0.2	Az egy LAN-on levő összes útválasztó.
224.0.0.5	Az egy LAN-on levő összes OSPF-útválasztó.
224.0.0.251	Az egy LAN-on levő összes DNS-kiszolgáló.

A többi többesküldéses címhez tartozhatnak más hálózatokon lévő tagok. Ebben az esetben útválasztó protokollra van szükség. Először is a többesküldéses útválasztónak tudnia kell, hogy mely hosztok a csoport tagjai. A folyamat megkérheti a hosztját, hogy csatlakozzon egy bizonyos csoporthoz. Arra is megkérheti a hosztját, hogy lépjen ki a csoportból. Minden hoszt nyilvántartja, hogy a folyamatai jelenleg melyik csoportokhoz tartoznak. Amikor egy hoszt utolsó folyamata is elhagyja a csoportot, akkor a továbbiakban nem lesz a csoport tagja. Körülbelül percenként egyszer minden többesküldéses útválasztó lekérdező csomagot küld a LAN-on lévő hosztokhoz (a 224.0.0.1 helyi többesküldéses cím használatával), megkérve azokat, hogy jelentsék, milyen csoportokhoz tartoznak jelenleg. A többesküldéses útválasztók lehetnek egy helyen a normál útválasztókkal, illetve lehetnek különböző helyen is. Minden hoszt az összes olyan D osztályú címre válaszol, amelyben érdekelt. Ezek a lekérdező és válaszcsomagok az **IGMP (Internet Group Management Protocol – internetes csoportkezelő protokoll)** nevű protokollt használják. Ezt a protokollt az RFC 3376 írja le.

Számos többesküldéses útválasztó protokoll többesküldési feszítőfát állít elő, amely megadja a küldő és a csoporttagok közötti útvonalat. A használt algoritmusokat az 5.2.8. szakaszban tárgyaljuk. Egy AS-en belül az elsődlegesen használt protokoll a **PIM (Protocol Independent Multicast – protokollfüggetlen többesküldés)**. A PIM-et számos formában használják. Sűrű módú PIM (Dense Mode PIM) esetén csonkolt visszairá-

nyú továbbítási fa jön létre. Ez olyan helyzetekben megfelelő, ahol a tagok a hálózatban elszórtan találhatók, mint például fájlok kézbesítése egy adatközponti hálózatban lévő kiszolgálók felé. Ritka módú PIM (Sparse Mode PIM) esetén a létrehozott feszítőfák hasonlóak a magalapú fákhoz. Ez olyan szituációkban alkalmas, amelyben például a tartalomszolgáltató tv-adást küld szét az IP-hálózat előfizetői számára. Ennek egyik változata a forrásspecifikus többesküldéses PIM, amelyet arra az esetre optimalizáltak, amelyben csak egy küldő küld a csoport felé. Végül a BGP többesküldéses kiterjesztései vagy alagutak használhatók többesküldéses útvonal létrehozásához, amikor a csoporttagok különböző AS-ekben találhatók.

5.6.9. Mobil IP

Az internet sok felhasználójának van hordozható számítógépe, és akkor is internetkapcsolatra van szükségük, amikor távol vannak az otthonuktól, illetve még az utazás közben is. Sajnos az IP címzési rendszere miatt az otthontól távoli munkavégzést könnyebb mondani, mint megteremteni annak lehetőségét, ahogy ezt hamarosan látni fogjuk. Amikor tömeges igény mutatkozott eziránt, az IETF felállított egy munkacsoportot, hogy kidolgozza a megoldást. A munkacsoport gyorsan kialakított számos olyan alapelvet, amelyet minden megoldástól elvártak. Ezek a következők voltak:

1. Minden mozgó hoszt legyen képes bárhol az otthoni IP-címét használni.
2. A rögzített hosztokban ne kelljen szoftverváltoztatásokat végrehajtani.
3. Az útválasztószoftverben és táblázatokban ne kelljen változtatásokat végrehajtani.
4. A mozgó hosztokhoz menő csomagok döntő többségének ne kellejen kitérőket tenni.
5. Ne okozzon többletmunkát az, amikor a mozgó hoszt otthon tartózkodik.

A választott megoldás az, amit az 5.2.10. szakaszban leírtunk. Hogy röviden összefoglaljuk, minden helynek, amelyik lehetővé kívánja tenni a felhasználói számára a barangolást, egy **otthoni ügynököt** (**home agent**) kell létrehoznia. Amikor egy mozgó hoszt megjelenik egy idegen helyen, új IP-címet kér (ezt **c/o címnek** vagy **továbbítási címnek** (**care-of address**) nevezzük). A mozgó hoszt a továbbítási cím használatával megmondja az otthoni ügynöknek, hogy hol van. Amikor a mozgó hoszt csomagja megérkezik az otthoni helyre, és a mozgó hoszt máshol található, akkor az otthoni ügynök megfogja a csomagot, és alagúton keresztül elküldi azt az aktuális továbbítási címen lévő mobil hoszthoz. A mozgó hoszt a csomagokat közvetlenül vissza tudja küldeni a kommunikációs partnernek, de továbbra is az otthoni címet használja forráscímként. Ez a megoldás megfelel az összes fenti követelménynek, azzal a kivétellel, hogy a mozgó hoszt felé küldött csomagok kerülő úton mennek.

Most, hogy lefedtük az internet hálózati rétegét, belemélyedhetünk további részletekbe. A mobilitás támogatása iránti igény elsősorban magából az IP-címzési sémából

fakad. Minden IP-cím tartalmaz egy hálózatszámot és egy hosztszámot. Vegyük például a 160.80.40.20/16 címen található gépet. A 160.80 a hálózatszámot, a 40.20 pedig a hosztszámot adja meg. A világ bármely helyén lévő útválasztó rendelkezik útválasztó táblázattal, amely megmondja, hogy melyik vonalon keresztül lehet eljutni a 160.80 hálózathoz. Amikor beérkezik egy csomag, amelyben a címzett IP-címe 160.80.xxx.yyy formátumú, akkor az ezen a vonalon megy ki. Ha hirtelen az adott című gép egy távoli helyre kerül, a csomagjai továbbra is az otthoni LAN-ra (vagy útválasztóra) kerülnek továbbításra.

Ezen a ponton két lehetőség áll rendelkezésre – de egyik sem túl vonzó. Az első, hogy létre tudunk hozni egy útvonalat specifikusabb előtaggal. Ha a távoli hely hirdeti a 160.80.40.20/32 felé menő útvonalat, akkor a cél felé küldött csomagok újból a megfelelő helyre kezdenek érkezni. Ez a lehetőség az útválasztókon használt leghosszabb egyező előtag algoritmustól függ. Hozzáadtunk azonban egy útvonalat az IP-előtaghoz, amely egyetlen IP-címet tartalmaz. Az összes ISP meg fogja tanulni ezt az előtagot. Ha a számítógép áthelyezésekor mindenki ily módon változtatja a globális IP-útvonalakat, akkor minden útválasztó több millió táblázatbejegyzést fog tartalmazni, amely horribilis összegbe kerül. Ez tehát nem működőképes.

A második lehetőség a mobil hoszt IP-címének módosítása. Igaz, hogy az otthoni IP-címre küldött csomagok nem kézbesíthetők addig, amíg az összes érintett ember, program és adatbázis nem értesül a változásról. De a mozgó hoszt az új helyen továbbra is használni tudja az internetet webböngészéshez és más alkalmazások futtatásához. Ez a megoldás magasabb szinten kezeli a mobilitást. Általában ez történik, amikor egy felhasználó beviszi a hordozható számítógépét egy kávézóba és az internetet a helyi vezeték nélküli hálózaton keresztül használja. Ennek hátránya, hogy néhány alkalmazás megszakítja a működését, és nem tartja fenn a kapcsolatot a mozgó hoszt helyváltoztatása közben.

A mobilitás azonban alacsonyabb rétegben, az adatkapcsolati rétegben is kezelhető. Ez történik, amikor a hordozható számítógépet 802.11 vezeték nélküli hálózaton használják. A mozgó hoszt IP-címe nem változik, és a hálózati útvonal is változatlan marad. A vezeték nélküli kapcsolat biztosítja a mobilitást. A mobilitás mértéke azonban korlátozott. Ha a számítógépet túl messzire viszik, akkor másik hálózaton keresztül kell csatlakozni az internetre, másik IP-címmel.

Az IPv4 mobil IP-megoldását az RFC 3344 rögzíti. Ez a meglévő internetes útválasztást használja, és lehetővé teszi, hogy helyváltoztatás közben is megmaradjon a hoszt kapcsolata a saját IP-címével. Ahhoz, hogy ez működjön, a mozgó hosztot fel kell tudni térképezni mozgás közben. Ez ICMP-útválasztó hirdetéssel és kérelmező üzenettel történik. A mozgó hosztok figyelik a rendszeres időközönként küldött útválasztó hirdetéseket, vagy kérelmet küldenek a legközelebbi útválasztó feltérképezése érdekében. Ha ez az útválasztó nem a mobil hoszt otthoni tartózkodáskor használt szokásos útválasztója, akkor ennek a hosztnak idegen hálózaton kell lennie. Ha ez az útválasztó változott az utolsó alkalom óta, akkor a mozgó hoszt másik idegen hálózatba lépett. Ugyanez a mechanizmus teszi lehetővé a mozgó hosztok számára a saját otthoni ügynökük megkeresését.

Ahhoz, hogy a mozgó hoszt az idegen hálózaton hozzájusson a továbbítási (c/o) IP-címhez, egyszerűen használhatja a DHCP-t, vagyis a dinamikus hosztkonfigurációs protokollt. Vagy ha kevés az IPv4-cím, akkor a mozgó hoszt tud csomagokat küldeni és

fogadni egy idegen ügynökön keresztül, amely már rendelkezik IP-címmel a hálózaton. A mozgó hoszt az idegen ügynököt ugyanazon ICMP-mechanizmussal keresi meg, mint az otthoni ügynököt. Miután egy mozgó hoszt IP-címet kap, vagy idegen ügynököt talál, használni tudja a hálózatot az otthoni ügynöknek szóló üzenet küldésére, informálva az otthoni ügynököt az aktuális helyéről.

Az otthoni ügynöknek csak akkor kell tudnia fogadni a mozgó hosztnak küldött csomagokat, amikor a mozgó hoszt nincs otthon. Az ARP megfelelő módszert biztosít erre. Ha egy csomagot Etherneten keresztül kíván küldeni az IP-hoszt felé, akkor az útválasztónak tudnia kell a hoszt Ethernet-címét. Az útválasztó által alkalmazott szokványos mechanizmus, hogy ARP-lekérdezést küld például annak kiderítéséhez, hogy mi a 160.80.40.20 Ethernet-címe. Ha a mozgó hoszt otthon van, akkor az IP-címére vonatkozó ARP-lekérdezésre a saját Ethernet-címével válaszol. Ha a mozgó hoszt nincs otthon, akkor az otthoni ügynök válaszol a lekérdezésre az Ethernet-cím megadásával. Az útválasztó ezután a csomagjait a 160.80.40.20 címen az otthoni ügynökhöz küldi. Ezt proxy ARP-nek nevezzük.

Az ARP-leképezések gyors frissítéséhez, amikor a mozgó hoszt elmegy otthonról, illetve hazatér, egy másik ARP-technika használható, amelyet **önkényes ARP**-nek (**gratuitous ARP**) neveznek. Ennek lényege az, hogy a mobil hoszt vagy az otthoni ügynök maga küld ARP-lekérdezést a mobil IP-címre vonatkozóan, amelyre a megfelelő válasz jön, úgy hogy az útválasztó ezt észleli, és a leképezését frissíti.

Ha alagúttechnikával küldünk csomagot az otthoni ügynök és a c/o továbbítási címen lévő mozgó hoszt között, akkor a csomag beágyazódik a c/o címre küldött másik IP-fejrésszel. Ha a beágyazott csomag megérkezik a c/o címre, akkor a külső IP-fejrész eltávolításra kerül és a csomag kibontható.

Ahogy számos internetprotokoll esetén, itt is a részletekben rejlik a buktató, leggyakrabban a többi bevezetett protokollokkal való kompatibilitás részleteiben. Két bonyodalom is van. Először is a NAT-dobozok megkeresik az IP-fejrészben a TCP- vagy UDP-fejrészt. A mobil IP-re alkalmazott alagút típusú átvitel eredeti formája nem használja ezeket a fejrészeket, így ez nem működik a NAT-dobozokkal. A megoldás az, hogy megváltoztatjuk a beágyazást, hogy tartalmazzon egy UDP-fejrészt.

A második bonyodalom, hogy néhány ISP megvizsgálja a csomagok forrás IP-címét, hogy ellenőrizze, vajon a forráshoszt helye egyezik-e azzal a hellyel, mint ahol annak az útválasztó protokoll szerint lenni kellene. Ezt a technikát **beléptető szűrésnek** (**ingress filtering**) nevezik. Ez egy biztonsági intézkedés annak érdekében, hogy kiszűrjék és eldobják a látszólag helytelen című csomagokat, amelyek esetleg ártalmasak lehetnek. Az idegen hálózaton lévő mozgó hoszt által egy másik, interneten lévő hosztnak küldött csomagoknak a forrás IP-címe azonban az útválasztó szerint helytelen lesz, ezért ezeket a csomagokat az útválasztó el fogja dobni.

További kérdés, amit még nem vitattunk meg, a biztonság kérdése. Amikor egy otthoni ügynök üzenetet kap, hogy továbbítsa Roberta csomagját a megadott IP-címre, akkor jobb, ha ezt nem teszi meg, hacsak Roberta meg nem győzi arról, hogy ő a kérés forrása, és nem próbálja meg valaki kéretlenül helyette. Erre a célra kriptográfiai hitelesítési protokollokat használnak. Ezeket a protokollokat a 8. fejezetben tanulmányozzuk.

Az IPv6 mobilitási protokolljai az IPv4-alapra épülnek. A fenti séma a háromszögeléses útválasztás problémájától szenved, amelyben a mozgó hosztnak küldött csomagok

kerülő utat tesznek a távoli otthoni ügynökön keresztül. Az IPv6-ban az útvonal-optimalizálás közvetlen útvonalat követ a mobil és a többi IP-cím között, miután a kezdeti csomagok a hosszú útvonalat követték. A mobil IPv6 definícióját az RFC 3775 tartalmazza.

Létezik másfajta mobilitás is, amelyet az internethez definiáltak. Néhány repülőgép beépített vezeték nélküli hálózattal rendelkezik, amelyet az utasok is használhatnak számítógépeik internetre csatlakoztatásához. A repülőnek van egy útválasztója, amely az internet többi részéhez vezeték nélküli összeköttetésen keresztül csatlakozik. (Vezetékes vonalat várt?) Így van egy repülő útválasztónk, amely azt jelenti, hogy a teljes hálózat mozog. A hálózat mobilitási kialakításai ezt a helyzetet anélkül támogatják, hogy a laptopok észlelnék, hogy a repülőgép mozog. Csupán azt látják, hogy ez egy másik hálózat. Természetesen néhány laptop mobil IP-t használ az otthoni cím megtartásához, miközben a repülőn van, így az alkalmazott mobilitás kétszintű. Az IPv6 hálózati mobilitását az RFC 3963 adja meg.

5.7. Összefoglalás

A hálózati réteg szolgáltatást nyújt a szállítási rétegnek. Alapulhat virtuális áramkörökön vagy datagramokon. Mindkét esetben a legfontosabb feladata, hogy a csomagok útválasztását végezze a forrástól a célig. A datagramalapú hálózatok az útválasztási döntést az egyes csomagok alapján hozzák meg. A virtuáliséramkör-alapú hálózatokban akkor hoznak útválasztási döntést, amikor az áramkör felépül.

A számítógép-hálózatokban sokféle útválasztó algoritmus használatos. A statikus algoritmusok közé tartozik például a legrövidebb útalapú útválasztás és az elárasztás. A dinamikus algoritmus a távolságvektor és a kapcsolatállapot-alapú útválasztás. A legtöbb meglévő hálózat ezek valamelyikét használja. Egyéb, fontos útválasztási területek: a hierarchikus útválasztás nagy hálózatokban, mozgó hosztok útválasztása, az adatszóró útválasztás, a többesküldéses útválasztás és a bárkinek szóló útválasztás.

A hálózatokban könnyen torlódás léphet fel, ami megnöveli a csomagok késleltetését és a csomagvesztést. A hálózattervezők a torlódást megfelelő tervezéssel igyekeznek elkerülni. A módszerek között van az elegendő kapacitás, a torlódás nélküli útvonalak kiválasztása, a további forgalom visszautasítása, a lassítási jelzés küldése a források számára, valamint a terhelés eltávolítása.

A torlódások puszta kezelésén túl a következő lépés az, hogy ténylegesen megpróbálunk elérni egy szavatolt szolgáltatásminőséget. Néhány alkalmazás többet foglalkozik az átbocsátóképességgel, mások pedig a késleltetéssel és a dzsitterrel. A szolgáltatásminőség különböző szintjeinek szavatolására használatos módszerek között megtalálhatjuk a következők kombinációját: forgalomformálás, erőforrás-foglalás az útválasztókon és a belépés-ellenőrzés. A jó szolgáltatásminőség biztosítása érdekében tervezett megoldások közé tartoznak az IETF által kidolgozott integrált szolgáltatások (beleértve az RSVP-t is) és a differenciált szolgáltatások.

A hálózatok sok mindenben különbözhetnek egymástól, így amikor több hálózatot kapcsolnak össze, adódhatnak nehézségek. Ha a hálózatok maximális csomagmérete el-

térő, darabolásra lehet szükség. A különböző hálózatok belül különböző útválasztó protokollokat használhatnak, de külsőleg azonos protokollt kell futtatniuk. Néha a gondokat megkerülhetjük, ha egy csomagot alagút típusú átvitellel viszünk keresztül egy idegen hálózaton, de ha a forrás- és célhálózat is eltérő, akkor ez a megoldás csődöt mond.

Az internetnek sokféle, a hálózati réteghez kapcsolódó protokollja van. Ezek között van a datagramprotokoll, az IP és a hozzá kapcsolódó vezérlőprotokollok is, mint amilyen az ICMP, az ARP és a DHCP. Az MPLS nevű összeköttetés-alapú protokoll IP-csomagokat visz át néhány hálózaton. A hálózatokban használt fő útválasztó protokollok közül a hálózaton belüli forgalomra az OSPF-et használják, a hálózatok közötti forgalomra pedig a BGP-protokollt. Az internet gyorsan kifogy az IP-címekből, ezért kifejlesztették az IP új verzióját, az IPv6-ot, amelynek használata azonban még mindig lassan terjed.

5.8. Feladatok

1. Adjon két példát olyan alkalmazásokra, amelyekhez az összeköttetés-alapú szolgáltatás a megfelelő, majd mondjon két olyan példát is, amelyeknek az összeköttetés nélküli szolgáltatás a legjobb!

2. A datagramalapú hálózatok minden csomagot különálló egységként irányítanak, függetlenül az összes többitől. A virtuálisáramkör-alapú hálózatoknak nem kell ezt megtenniük, mivel minden adatcsomag egy előre meghatározott útvonalat követ. Ez a megfigyelés jelenti-e azt, hogy a virtuálisáramkör-alapú hálózatoknak nincs szükségük arra a képességre, hogy elszigetelt csomagokat tetszőleges forrásból tetszőleges célhoz tudjanak irányítani? Indokolja válaszát!

3. Mondjon három példát olyan protokollparaméterekre, amelyek összeköttetés felépítésekor egyeztetés tárgyát képezhetik!

4. Feltételezve, hogy minden hoszt és útválasztó megfelelően működik, és mindkettő szoftvere hibáktól mentes, van-e bármilyen kicsi esély is arra, hogy egy csomagot rossz célhoz kézbesítenek?

5. Adjon egyszerű heurisztikus eljárást arra, hogyan lehet egy adott forráscsomóponttól egy adott címzett csomópontig két utat megtalálni egy olyan hálózatban, amely bármelyik kommunikációs vonal elvesztését túléli (feltéve, hogy létezik két ilyen út)! Az útválasztókat kellően megbízhatónak tekintjük, tehát nem szükséges az útválasztók összeomlásának lehetősége miatt aggódni.

6. Vegyük az 5.12.(a) ábra hálózatát. Távolságvektor-alapú útválasztást használunk, és a következő vektorok éppen most érkeztek meg a *C* útválasztóhoz: *B*-től: (5, 0, 8, 12, 6, 2), *D*-től: (16, 12, 6, 0, 9, 10) és *E*-től: (7, 6, 3, 9, 0, 4). A mért késleltetések sorrendben *B*-ig, *D*-ig és *E*-ig: 6, 3 és 5. Mi lesz *C* új útválasztó táblázata? Adja meg mind a használandó kimenő vonalat, mind a várt késleltetést!

7. Ha egy 50 útválasztóból álló hálózatban a késleltetéseket mint 8 bites számokat jegyezzük fel, és a késleltetési vektorokat másodpercenként kétszer cseréljük ki, mekkora sávszélességet emészt fel (duplex) vonalanként az elosztott útválasztó algoritmus? Tegyük fel, hogy minden útválasztónak három vonala van más útválasztók felé.

8. Az 5.13. ábrán a két ACF-bitkészlet logikai VAGY kapcsolata minden sorban 111. Ez csak egy véletlen itt, vagy igaz minden hálózatra minden körülmények között?

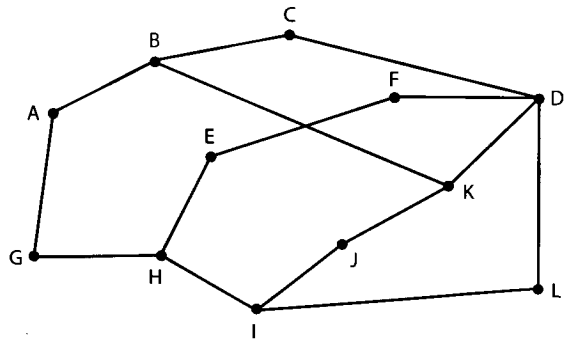
9. Milyen régió- és klaszterméreteket válasszunk egy 4800 útválasztóval történő hierarchikus útválasztási esethez, hogy a lehető legkisebb méretű útválasztó táblázatot kapjuk egy háromszintű hierarchiában? Jó kiindulás az a feltételezés, hogy közel optimális egy olyan megoldás, ahol k klaszterünk van, azok mind k régióból állnak, melyekben mindenhol k útválasztó van; ami azt jelenti, hogy k körülbelül 4800 köbgyökével egyenlő (kb. 16). Találjon próbálgatással olyan kombinációkat, ahol mindhárom paraméter 16 közelében van.

10. A szövegben azt állítottuk, hogy amikor egy mozgó hoszt nincs otthon, az otthoni LAN-jára küldött csomagokat az otthoni ügynök fogja el. Egy 802.3 LAN feletti IP-hálózaton hogyan viszi véghez ezt az elfogást az otthoni ügynök?

11. Az 5.6. ábra hálózatát tekintve mennyi csomagot generál egy adatszórás *B*-ből, ha az alkalmazott eljárás:
    (a) a visszairányú továbbítás?
    (b) a nyelőfa?

12. Tekintsük az 5.15.(a) ábra hálózatát! Képzeljük el, hogy egy új vonalat húzunk *F* és *G* közé, de az 5.15.(b) ábrán látható nyelőfa változatlan marad. Hogyan változik meg az 5.15.(c) ábra?

13. Számítson ki egy többesküldéses feszítőfát a *C* útválasztó számára az alábbi hálózatban, ha a csoporttagok az *A*, *B*, *C*, *D*, *E*, *F*, *I* és *K* útválasztókban vannak!



14. Tegyük fel, hogy az 5.20. ábra *B* csomópontja épp most indult újra, és nincs útválasztási információ a táblázataiban. Egyszer csak szüksége lesz egy a *H*-ba vezető útvonalra. Ekkor üzeneteket küld szét 1-es, 2-es, 3-as stb. *TtL* értékkel. Hány kör alatt talál egy útvonalat?

15. Egy belül virtuális áramköröket használó hálózat számára torlódáskezelési mechanizmus lehetne az, hogy egy útválasztó tartózkodik egy vett csomag nyugtázásától, amíg (1) meg nem tudja, hogy a virtuális áramkörön az utolsó átvitelét sikeresen vették és (2) nincs szabad puffere. Az egyszerűség kedvéért tételezzük fel, hogy az útválasztók egy megáll-és-vár protokollt használnak, és minden virtuális áramkörnek van egy kijelölt puffere a forgalom mindkét irányában. Ha *T* másodpercbe kerül egy csomagot átvinni (legyen az adat vagy nyugta), és *n* útválasztó van az útvonalon, milyen sebességgel kézbesítik a csomagot a címzett hosztnak? Tegyük fel, hogy az átviteli hibák ritkák és a hoszt–útválasztó összeköttetés végtelenül gyors.

16. Egy datagramalapú hálózatban az útválasztók eldobhatnak csomagokat, amikor erre szükség van. Annak a valószínűsége, hogy egy útválasztó eldob egy csomagot, *p*. Vegyük azt az esetet, amikor a forráshoszt összeköttetésben áll a forrás-útválasztóval, amely összeköttetésben áll a címzett útválasztóval, azon keresztül pedig a címzett hoszttal. Ha bármelyik útválasztó eldob egy csomagot, a forráshosztnak végül is lejár az időzítése, és újra próbálkozik. Ha mind a hoszt–útválasztó, mind az útválasztó–útválasztó vonalakat ugrásnak vesszük, mi az átlagértéke:
    (a) az egy csomag által átvitelenként megtett ugrásoknak?
    (b) a sikeresen véghezvitt átviteleknek?
    (c) az egy vett csomag által igényelt ugrásoknak?

17. Írja le az ECN-módszer és a RED-módszer közötti két fő különbséget!

18. Egy hálózat vezérjeles vödör sémát használ a forgalomformáláshoz. 5 μs-onként kerül új vezérjel a vödörbe. Minden vezérjel egy 48 bájtnyi adatot tartalmazó cella továbbítását engedélyezi. Mi a legnagyobb fenntartható adatsebesség?

19. Egy 6 Mb/s-os hálózaton elhelyezkedő számítógépet egy vezérjeles vödör szabályoz. A vezérjeles vödröt 1 Mb/s sebességgel töltik, és az kezdetben 8 megabitet tartalmaz. Milyen hosszan forgalmazhat a számítógép a teljes 6 Mb/s-os sebességen?

20. Az 5.34. ábra hálózata RSVP-t használ többbesküldéses fákkal az 1. és 2. hosztokhoz, amint az látható. Tegyük fel, hogy a 3. hoszt egy 2 MB/s sávszélességű csatornát igényel az 1. hosztból jövő folyam számára, és egy másik, 1 MB/s-os csatornát a 2. hosztból jövő folyam számára. Ezzel egy időben a 4. hoszt igényel egy 2 MB/s sávszélességű csatornát az 1. hosztból jövő folyam számára, és az 5. hoszt is igényel egy 1 MB/s sávszélességű csatornát a 2. hosztból jövő folyam számára. Összesen mekkora sávszélességeket foglalnak le ezen igények számára az *A*, *B*, *C*, *E*, *H*, *J*, *K* és *L* útválasztókban?

21. Egy útválasztó 2 millió csomagot képes feldolgozni egy másodperc alatt. A felajánlott terhelés 1,5 millió csomag/s. Ha egy útvonalon a forrástól a címzettig 10 útválasztó van, mennyi idő fordítódik a sorbaállásra és az útválasztó általi kiszolgálásra?

22. Vegyük a differenciált szolgáltatások használatát gyorsított továbbítással. Van-e garancia arra, hogy a gyorsított csomagok kisebb késleltetést szenvednek, mint a szabályos csomagok? Ha igen, miért, ha nem, miért nem?

23. Tegyük fel, hogy az *A* hoszt össze van kötve az *R1* útválasztóval, az *R1* útválasztó össze van kötve az *R2* útválasztóval, az *R2* pedig a *B* hoszttal. Tegyük fel, hogy az *A* hoszt IP-szoftverének átadunk egy olyan TCP-üzenetet, ami 900 adatbájtot és 20 bájtnyi TCP-fejrészt tartalmaz, hogy szállítsa el azt a *B* hosztnak. Adja meg az IP-fejrész *Teljes hossz*, *Azonosítás*, *DF*, *MF* és *Darabeltolás* mezőit a három összeköttetésen keresztül áthaladó összes csomagra! Feltesszük, hogy az *A-R1* összeköttetés 1024 bájtos maximális keretméretet támogat, ami tartalmazza a 14 bájtos keretfejrészt is, az *R1-R2* összeköttetés maximum 512 bájtos keretméretet támogat, benne 8 bájtos keretfejrésszel, az *R2-B* összeköttetésen pedig a maximális keretméret 512 bájt a 12 bájtos keretfejrésszel együtt.

24. Egy útválasztó olyan IP-csomagokat ont magából, melyeknek teljes hossza (az adatmező és a fejrész együtt) 1024 bájt. Feltéve, hogy a csomagok 10 másodpercig élnek, milyen maximális vonali sebességgel működhet az útválasztó anélkül, hogy fennállna az IP-datagram *Azonosítás* mezőjében lévő szám körbefordulásának veszélye?

25. Egy *Szigorú forrás általi útválasztás* opcióval ellátott IP-datagramot fel kell darabolni. Gondolja, hogy az opciót minden darabba bemásolják, vagy elegendő csak az első darabba belerakni? Indokolja válaszát!

26. Tegyük fel, hogy a B osztályú címek hálózati részéhez 16 helyett 20 bitet használtunk volna. Hány B osztályú hálózat lett volna ekkor?

27. Alakítsa át a C22F1582 hexadecimális jelölésű IP-címet pontokkal elválasztott decimális jelölésre!

28. Az interneten egy hálózatnak 255.255.240.0 az alhálózati maszkja. Legfeljebb hány hosztot képes kezelni ez a hálózat?

29. Miért az IP-címeket használják a különleges hálózatok kipróbálásához, és nem az Ethernet-címeket? Meg tudja ezt indokolni?

30. A 198.16.0.0-tól kezdve sok egymást követő IP-cím áll a rendelkezésünkre. Tegyük fel, hogy az *A*, *B*, *C* és *D* intézmények rendre 4000, 2000, 4000 és 8000 címet igényelnek. Adja meg mindegyik intézményhez az annak kiosztott első és utolsó IP-címet, és az alhálózati maszkot a *w.x.y.z/s* jelölés szerint.

31. Egy útválasztó éppen most kapta meg a következő új IP-címeket: 57.6.96.0/21, 57.6.104.0/21, 57.6.112.0/21 és 57.6.120.0/21. Lehet-e csoportosítani ezeket, ha mindegyik ugyanazt a kimeneti vonalat használja? Ha igen, mi lesz a csoportos cím? Ha nem, miért nem?

32. Az IP-címek 29.18.0.0-tól 29.18.127.255-ig tartanak, ezeket lehet csoportosítani a 29.18.0.0/17 címen. Ugyanakkor van egy eddig kiosztatlan, 1024 címet tartalmazó rés 29.18.60.0-tól 29.18.63.255-ig, amit most egyszerre kiosztanak egy olyan hosztnak, amely egy másik kimeneti vonalat használ. Szükséges lesz-e most felbontani a csoportos címet blokkjaira, hozzáadni egy új blokkot a táblázathoz, hogy aztán meglássuk, lehetséges-e valamilyen újracsoportosítás? Ha nem, mit lehet tenni helyette?

33. Egy útválasztó útválasztó-táblázatában a következő (CIDR) bejegyzések találhatók:

Cím/maszk	**Következő ugrás**
135.46.56.0/22	0. interfész
135.46.60.0/22	1. interfész
192.53.40.0/23	1. útválasztó
alapértelmezett	2. útválasztó

Mit tesz az útválasztó, ha rendre a következő IP-címeket tartalmazó csomagok érkeznek?

(a) 135.46.63.10
(b) 135.46.57.14
(c) 135.46.52.2
(d) 192.53.40.7
(e) 192.53.56.7

34. Sok cég olyan politikát folytat, hogy két (vagy több) útválasztó segítségével köti össze a hálózatát az internettel, hogy így biztosítson némi redundanciát arra az esetre, ha valamelyik útválasztó meghibásodna. Tartható-e ez a politika NAT-ok használata esetén? Indokolja válaszát!

35. Épp most magyarázta el az ARP-protokollt a barátjának. Amikor Ön végzett, ő azt mondja: „Értem. Az ARP szolgáltatást biztosít a hálózati rétegnek, tehát az adatkapcsolati réteg része." Mit mond neki?

36. Írjon le egy olyan módszert, amivel az IP-darabokat a célban össze lehetne állítani.

37. A legtöbb IP datagram-összeállító algoritmusnak van egy időzítője, hogy egy elveszett darab ne foglalhassa le örökké az összeállító puffereket. Tegyük fel, hogy egy datagramot négy részre darabolnak. Az első három darab megérkezik, de az utolsót késleltetik. Végül is az időzítő lejár, és eldobjuk a már a vevő memóriájában levő három darabot. Kicsivel később bebotorkál a negyedik darab. Mi vele a teendő?

38. Az IP-ben az ellenőrző összeg csak a fejrészt védi, az adatot nem. Mit gondol, miért választották ezt így?

39. Egy Bostonban élő személy Minneapolisba utazik, és viszi magával a hordozható számítógépét is. Meglepetésére a minneapolisi úti céljánál levő LAN egy vezeték nélküli IP LAN, így nem kell hozzá csatlakoznia. Vajon azért még mindig szükséges végigcsinálnia az egész ügyletet a hazai és idegen ügynökökkel, hogy az e-levelek és a többi forgalom helyesen érkezzen meg?

40. Az IPv6 16 bájtos címeket használ. Ha egy egymillió címből álló blokkot utalnak ki minden pikoszekundumban, meddig fognak kitartani a címek?

41. Az IPv4-fejrészben használt *Protokoll* mező nincs jelen a rögzített IPv6-fejrészben. Miért nincs?

42. Amikor bevezetik az IPv6-protokollt, meg kell-e változtatni az ARP-protokollt? Ha igen, akkor a változások elvi vagy technikai jellegűek?

43. Írjon egy programot, amely elárasztásos útválasztást szimulál. Minden csomag tartalmazzon egy számlálót, amelyet minden ugrásnál csökkentenek. Amikor a számláló eléri a nullát, a csomagot eldobják. Az idő diszkrét felosztású, és minden vonal egy csomagot kezel időintervallumonként. Készítsen három változatot a programból: minden vonal elárasztása, a bemeneti vonalon kívül minden vonal elárasztása, és csak a legjobb k (statisztikusan választott) vonal elárasztása. Hasonlítsa össze az elárasztást a determinisztikus útválasztással ($k = 1$) a késleltetés és a felhasznált sávszélesség szempontjából.

44. Írjon egy programot, amely diszkrét időosztást használva egy számítógép-hálózatot szimulál. Minden útválasztó minden várakozási sorában az első csomag egyetlen ugrást tesz meg időintervallumonként. Ha egy csomag megérkezik és nincs hely a számára, akkor eldobják, és nem adják újra. Ehelyett van egy végpontok közötti protokoll, lejáró időzítésekkel és nyugtacsomagokkal kiegészítve, amely a csomagot a forrás-útválasztótól újra elküldi. Ábrázolja a hálózat átbocsátóképességét, mint a végpontok közötti időzítési intervallum függvényét, a hibaaránnyal paraméterezve!

45. Írjon olyan függvényt, mely egy IP-útválasztó továbbítási funkcióját látja el! A függvény egy IP-címet kap paraméterként. Ezenkívül hozzáférhet még egy globális táblázathoz, ami hármasokból áll. Minden hármas három egész számot tartalmaz: egy IP-címet, egy alhálózati maszkot és a használandó kimeneti vonalat. A függvény CIDR használatával keresi ki a táblázatból az IP-címet, visszatérési értéke pedig a használandó kimeneti vonal.

46. Használja a *traceroute* (UNIX) vagy a *tracert* (Windows) programokat, hogy nyomon kövesse a saját gépétől a más kontinenseken lévő egyetemekhez vezető útvo-

nalakat! Készítsen listát a felfedezett tengerentúli összeköttetésekről! Néhány hely a próbálkozásokhoz:

www.berkeley.edu (California)
www.mit.edu (Massachusetts)
www.vu.nl (Amsterdam)
www.ucl.ac.uk (London)
www.usyd.edu.au (Sydney)
www.u-tokyo.ac.jp (Tokio)
www.uct.ac.za (Cape Town)

6. A szállítási réteg

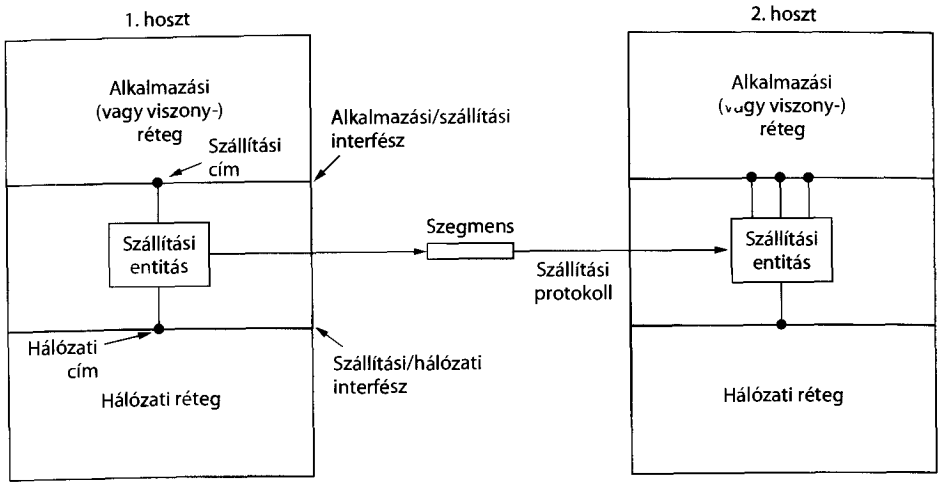
A hálózati réteggel egyetemben a szállítási réteg a protokollhierarchia legfontosabb része. A hálózati réteg két végpont között biztosít csomagtovábbítást datagramok vagy virtuális áramkörök segítségével. A szállítási réteg a hálózati rétegre épülve biztosít megbízható adatátvitelt a forrásgép egyik folyamatától a célgép valamely folyamataig, mindezt függetlenül a használt fizikai hálózattól. A szállítási réteg biztosítja az alkalmazások számára azt az elvonatkoztatást, amelyre azoknak a hálózat használatához szükségük van. A szállítási réteg nélkül a rétegezett protokollkoncepciónak nem sok értelme lenne. Ebben a fejezetben a szállítási réteget fogjuk részletesen tanulmányozni, beleértve annak szolgáltatásait, az API felépítését, a kapcsolat- és torlódáskezelést, a protokollokat (mint például a TCP és az UDP), és a hatékonyságot.

6.1. A szállítási szolgáltatás

A következő néhány alfejezet a szállítási szolgáltatást mutatja be. Áttekintjük, hogy milyen szolgáltatásokat kínál az alkalmazási réteg számára. Ahhoz, hogy a szállítási szolgáltatást még kézzelfoghatóbbá tegyük, megvizsgáljuk a szállítási szolgáltatás primitívjeinek két halmazát. Először egy egyszerű (de hipotetikus) halmazt vizsgálunk meg, hogy megmutassuk az alapvető megfontolásokat, majd pedig egy olyan interfészt mutatunk be, amelyet az internet is széles körben használ.

6.1.1. A felső rétegeknek nyújtott szolgáltatás

A szállítási réteg legfőbb célja az, hogy hatékony, megbízható és gazdaságos adattovábbítási szolgáltatást nyújtson felhasználóinak, általában az alkalmazási rétegben futó folyamatoknak. E cél érdekében a szállítási réteg felhasználja a hálózati réteg által nyújtott szolgáltatásokat. A szállítási rétegen belül azt a hardver- és/vagy szoftverelemet, amely a munkát végzi, **szállítási entitásnak** (**transport entity**) nevezzük. Ez lehet az operációs rendszer magjának (kernelének) része, egy hálózati alkalmazáshoz tartozó könyvtár része, önálló felhasználói folyamat vagy a hálózati illesztőkártya. Az interneten az első



6.1. ábra. *A hálózati, szállítási és alkalmazási réteg*

két megoldás a leggyakoribb. A hálózati, szállítási és alkalmazási réteg kapcsolatát a 6.1. ábra szemlélteti.

Ahogy a hálózati szolgáltatásoknak két típusa van, összeköttetés-alapú és összeköttetés nélküli, ugyanígy kétféle szállítási szolgáltatás létezik. Az összeköttetés-alapú szállítási szolgáltatás sok tekintetben hasonló az összeköttetés-alapú hálózati szolgáltatáshoz. Az összeköttetésnek mindkét esetben három fázisa van: létesítés, adatátvitel és lebontás. A címzés és forgalomszabályozás szintén hasonló a két rétegben. Az összeköttetés nélküli szállítási szolgáltatás is nagyon hasonló az összeköttetés nélküli hálózati szolgáltatáshoz. Vegyük észre azonban, hogy nem célszerű összeköttetés nélküli szállítási szolgáltatást nyújtani összeköttetés-alapú hálózati szolgáltatás felett, hiszen erőforráspazarló felépíteni egy összeköttetést egyetlen csomag elküldése érdekében, majd mindjárt lebontani azt.

A nyilvánvaló kérdés ezután az, hogy ha a szállítási réteg szolgáltatása ennyire hasonló a hálózati réteg szolgáltatásához, akkor miért van mégis két külön réteg. Miért nem elegendő egy réteg? A válasz egy apró, de lényeges különbségben rejlik. A szállítási réteg kódja teljes egészében a felhasználók gépein fut, szemben a hálózati réteggel, amely nagyrészt az útválasztókon, az útválasztókat pedig a szolgáltató üzemelteti (legalábbis a nagy kiterjedésű hálózatokban). Mi történik, ha a hálózati réteg nem nyújt megfelelő szolgáltatást? Esetleg gyakran veszti el a csomagokat? Mi történik, ha az útválasztók időről időre tönkremennek?

Ezekben az esetekben bizony bajok történnek. A felhasználóknak nincs igazi beleszólása a hálózati réteg működésébe és az útválasztók sem a felhasználók tulajdonában vannak, ezért nem tudják azzal megoldani a gyenge minőségű szolgáltatás problémáját, hogy jobb minőségű útválasztókat vagy jobb hibakezelést építenek be a hálózati rétegbe. Az egyetlen lehetőség az, hogy egy olyan másik réteget építsünk rá a hálózati rétegre, amely javítja a szolgáltatásminőséget. Ha egy összeköttetés nélküli hálózatban a csomagok elvesznek vagy megsérülnek, a szállítási entitás képes észlelni a hibát, és kijavíthatja

azt újraküldéssel. Ha egy összeköttetés-alapú hálózatban a szállítási entitás egy hosszú átvitel közepén arról értesül, hogy hálózati összeköttetése hirtelen megszakadt, és nem tudja meghatározni, hogy mi történt az éppen úton levő csomagokkal, akkor felépíthet egy új hálózati összeköttetést a társentitás felé. Ezen az összeköttetésen egy lekérdezéssel megkérdezheti a társentitástól, hogy mely adatok érkeztek meg és melyek nem. Ezután az átvitelt ott folytathatják, ahol félbe maradt.

A szállítási réteg létezése lényegében azt teszi lehetővé, hogy a szállítási szolgáltatás megbízhatóbb lehessen annál a hálózati szolgáltatásnál, amelyre ráépül. Mindezen felül a szállítási szolgáltatás primitívjei könyvtári függvényhívásokként is megvalósíthatók, és ezzel függetleníthetők a hálózati szolgáltatás primitívjeitől. A hálózati szolgáltatás hívásai az egyes hálózatokban jelentősen eltérhetnek (az összeköttetés nélküli Ethernet-függvényhívások például jelentősen különböznek az összeköttetés-alapú WiMAX hálózati függvényhívásoktól). Ha a hálózati szolgáltatás részletei rejtve maradnak a szállítási szolgáltatás primitívjei elől, akkor a hálózati szolgáltatás lecserélésekor mindössze arra van szükség, hogy a könyvtári függvények egyik készletét lecseréljük egy másikra, amely egy másik szolgáltatásra ráépülve látja el ugyanazt a feladatot.

A szállítási rétegnek köszönhetően az alkalmazások fejlesztői egy szabványos primitívkészletre írhatják a kódot, és az így megírt programok a hálózatok széles skáláján működnek anélkül, hogy a fejlesztőknek a különféle hálózati interfészekkel és megbízhatósági szintekkel törődniük kellene. Ha minden létező hálózat tökéletes lenne, és mindegyik egy olyan közös szolgáltatásprimitív-készletet használna, amely garantáltan soha, de soha nem változik, akkor lehet, hogy nem lenne szükség a szállítási rétegre. A gyakorlati életben azonban azt a kulcsfontosságú feladatot teljesíti, hogy elszigeteli a magasabb rétegeket a műszaki megoldásoktól és a hálózat tökéletlenségeitől.

A fenti ok miatt sokan megkülönböztetik az 1–4. rétegeket a 4. feletti réteg(ek)től. Az alsó négy réteget tekinthetjük a **szállítási szolgáltatónak** (**transport service provider**), míg a magasabb réteg(ek)et tekinthetjük a **szállítási szolgáltatás felhasználójának** (**transport service user**). A szolgáltató és a felhasználó ezen elkülönítése jelentős hatással van a rétegek kialakítására, és kulcsfontosságú helyzetbe hozza a szállítási réteget, mivel ez alkotja a fő határvonalat a szolgáltató és a megbízható adatátviteli szolgáltatás felhasználója között. Ez az a réteg, amelyet az alkalmazások látnak.

6.1.2. Szállítási szolgáltatási primitívek

A szállítási rétegnek néhány műveletet, vagyis egy szállítási szolgáltatási interfészt kell biztosítania az alkalmazási programok számára annak érdekében, hogy a felhasználók hozzáférhessenek a szolgáltatásaihoz. Minden szállítási szolgáltatás egyedi interfésszel rendelkezik. Ebben a szakaszban először egy egyszerű (hipotetikus) szállítási szolgáltatást és annak interfészét fogjuk megvizsgálni, hogy bemutassuk a legalapvetőbb jellegzetességeket. A következő szakaszban pedig egy gyakorlati példával ismerkedünk majd meg.

A szállítási szolgáltatás hasonlít a hálózati szolgáltatáshoz, azonban van néhány fontos eltérés. A fő különbség köztük az, hogy a hálózati szolgáltatás a valódi hálózatok által nyújtott szolgáltatásokat igyekszik modellezni azok gyengéivel együtt. Az igazi hálózatok csomagokat veszíthetnek, tehát a hálózati szolgáltatás általában nem megbízható.

Ezzel szemben az (összeköttetés-alapú) szállítási szolgáltatás megbízható. Természetesen a valódi hálózatok nem hibamentesek, de pontosan a szállítási réteg feladata az, hogy egy nem megbízható hálózatra épülve megbízható szolgáltatást nyújtson.

Vegyünk például két olyan folyamatot egy számítógépen, amelyek a UNIX-ban csövekkel vannak összekötve (vagy bármely más folyamatok közötti kommunikációs eszközzel). Ezek azt feltételezik, hogy a köztük levő összeköttetés tökéletes. Hallani sem akarnak nyugtázásokról, elvesztett csomagokról, torlódásról vagy más, ehhez hasonló problémáról. Egy 100 százalékosan megbízható összeköttetést akarnak használni. Az *A* folyamat beteszi az adatokat a cső egyik végén, a *B* folyamat kiveszi a másik végén. Ez a lényege a szállítási szolgáltatásnak: elrejteni a hálózati szolgáltatás hiányosságait, hogy az alkalmazási folyamatok hibamentes bitfolyamot feltételezhessenek még akkor is, ha nem azonos gépen futnak.

A szállítási réteg járulékos tulajdonsága, hogy megbízhatatlan (datagram) szolgáltatást is tud nyújtani. Erről azonban viszonylag keveset lehet mondani, így a figyelmünket ebben a fejezetben főként az összeköttetés-alapú szállítási szolgáltatásokra fogjuk irányítani. Ennek ellenére van néhány olyan alkalmazás (mint például a kliens–szerver-alkalmazások és a folyamszerű multimédia-továbbítás) amelyek összeköttetés nélküli szállítási szolgáltatást használnak, ezért erről a későbbiekben még fogunk beszélni.

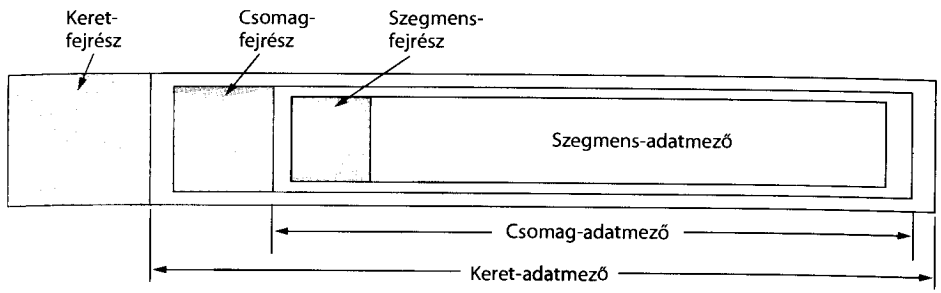
Egy másik különbség a hálózati és a szállítási szolgáltatás között a szolgáltatás felhasználóinak köre. A hálózati szolgáltatást csak a szállítási entitások használják. Kevesen írják meg a saját szállítási entitásaikat, ezért kevés program vagy felhasználó látja a csupasz hálózati szolgáltatást. Ezzel ellentétben viszont sok alkalmazás (ezzel együtt programozó) használja a szállítási primitíveket, ezért a szállítási szolgáltatásnak kényelmesnek és könnyen használhatónak kell lennie.

A 6.2. ábrán bemutatunk öt lehetséges szállítási primitívet, hogy képet kapjunk arról, milyen egy szállítási szolgáltatás. Ez a szállítási szolgáltatás csak egy puszta váz, de ízelítőt ad egy összeköttetés-alapú szállítási interfész lényeges feladataiból. Lehetővé teszi a felhasználói alkalmazások számára összeköttetések létesítését, használatát és lebontását, ami a legtöbb alkalmazásnak elegendő is.

Hogy a primitívek működésére is lássunk példát, vegyünk egy alkalmazást egy szerverrel és több távoli klienssel. Először a szerver egy LISTEN primitívet hajt végre, tipikusan egy könyvtári függvényhívással, ami rendszerhívást eredményez, hogy a szerver egy kliens jelentkezéséig blokkolódjon. Amikor egy kliens beszélni akar a szerverrel, egy CONNECT primitívet hajt végre. A szállítási entitás ezt úgy valósítja meg, hogy a

Primitív	Elküldött szegmens	Jelentése
LISTEN	(nincs)	Vár, amíg egy folyamat kapcsolódni nem próbál
CONNECT	CONNECTION REQ.	Összeköttetést próbál létrehozni
SEND	DATA	Adatot küld
RECEIVE	(nincs)	Vár, amíg adatcsomag (DATA) nem érkezik
DISCONNECT	DISCONNECTION REQ.	Ez az oldal bontani kívánja az összeköttetést

6.2. ábra. *Egy egyszerű szállítási szolgáltatás primitívjei*



6.3. ábra. *A szegmensek, csomagok és keretek beágyazása*

hívót blokkolja, és egy csomag adatmezejébe ágyazott szállítási üzenetet küld a szerver szállítási entitása részére.

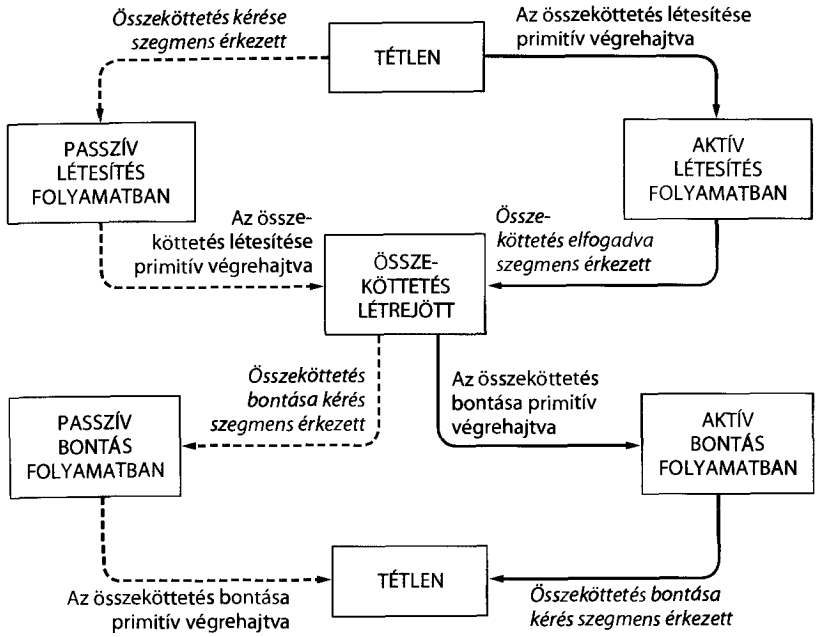
Itt rövid terminológiai kitérőt kell tennünk. Jobb híján a **szegmens** (**segment**) elnevezést fogjuk használni szállítási entitások közötti üzenetekre. A TCP, az UDP és sok más interneten használt protokoll is ezt az elnevezést használja. Pár régebbi protokoll az esetlen **TPDU** (**Transport Protocol Data Unit – szállítási protokoll adategység**) elnevezést használta, de napjainkban már nem használjuk, azonban még régebbi könyvekben és dolgozatokban előfordulhat.

Így tehát, ezek a szegmensek (melyeket a szállítási réteg küld és fogad) csomagokba (amelyeket a hálózati réteg használ) vannak beágyazva. A csomagok viszont (adatkapcsolati réteg által kezelt) keretekben (frame) foglalnak helyet. Amikor egy keret megérkezik, az adatkapcsolati réteg földolgozza a keret fejrészét, és ha a célcím megfelelő, a keret adatmezejének tartalmát továbbadja a hálózati entitásnak. A hálózati entitás szintén földolgozza a csomag fejrészét, és az adatmező tartalmát átadja a szállítási entitásnak. Ezt a beágyazott struktúrát szemlélteti a 6.3. ábra.

Visszatérve a kliens–szerver-példához, a kliens CONNECT hívása hatására a szállítási entitás CONNECTION REQUEST (összeköttetés-kérése) szegmenst küld a szerver felé. Amikor az megérkezik, a szállítási entitás meggyőződik arról, hogy a szerver LISTEN hívásban várakozik (azaz kész kéréseket kiszolgálni). Ha igen, akkor megszünteti a szerver blokkolását, és CONNECTION ACCEPTED (összekötetés kérése elfogadva) szegmenst küld vissza a kliensnek. Amint a szegmens megérkezik, a kliens blokkolása is feloldódik, így az összeköttetés létrejön.

Ekkor megkezdődhet az adatátvitel a SEND és RECEIVE (adás és vétel) primitívek segítségével. A legegyszerűbb esetben bármelyik fél végrehajthat egy (blokkoló) RECEIVE hívást, hogy várakozzon a partner által (SEND primitívvel) küldött adatra. Amikor a szegmens megérkezik, a fogadó blokkolása megszűnik, földolgozza a kapott szegmenst, és választ küld. Amíg mindkét fél nyomon tudja követni, hogy ki mikor következik, ez a rendszer jól működik.

Megjegyezzük, hogy a szállítási rétegben még egy egyszerű egyirányú adatforgalom is jóval bonyolultabb, mint a hálózati rétegben. Minden elküldött adatcsomagot nyugtáznak (előbb vagy utóbb), sőt a vezérlőszegmenseket hordozó csomagokra is érkezik közvetett vagy közvetlen nyugtázás. Ezeket a nyugtázásokat a szállítási entitások kezelik a hálózati rétegbeli protokollok segítségével, és a szállítási felhasználók számára ezek



6.4. ábra. *Egyszerű összeköttetés-kezelés állapotdiagramja. A dőlt betűvel szedett állapotátmeneteket beérkező csomagok váltják ki. A folytonos nyilak a kliens, a szaggatott nyilak a szerver állapotátmeneteit mutatják*

nem láthatók. Hasonlóan, a szállítási entitásoknak kell foglalkozniuk az időzítésekkel és az ismétlésekkel. Ezekből a mechanizmusokból semmit sem látnak a szállítási felhasználók. Számukra az összeköttetés egy megbízható, bitalapú csővezeték, azaz: amilyen biteket egy felhasználó a cső egyik végén betölt, azok a másik végén változatlanul, azonos sorrendben megjelennek. A bonyolultság elrejtésének képessége az ok, ami miatt a rétegezett protokollok nagyon hatékony eszköznek bizonyulnak.

Amikor egy összeköttetésre többé nincs szükség, azt le kell bontani, hogy ne foglaljon fölöslegesen táblahelyet a két szállítási entitáson belül. A bontásnak két változata van: *aszimmetrikus* és *szimmetrikus*. Az aszimmetrikus esetben bármelyik szállítási felhasználó kiadhat egy DISCONNECT primitívet, aminek hatására a szállítási entitás egy DISCONNECTION (összeköttetés bontása) szegmenst küld a távoli szállítási entitásnak. A szegmens megérkezésekor az összeköttetés lebomlik.

A szimmetrikus esetben mindkét irányt külön, a másiktól függetlenül zárják le. Amikor az egyik fél DISCONNECT hívást kezdeményez, az azt jelenti, hogy nincs több elküldenivaló adata, de továbbra is hajlandó partnere adatait fogadni. Ebben a modellben az összeköttetés akkor ér véget, amikor mindkét fél végrehajtotta a DISCONNECT primitívet.

Ezen egyszerű primitíveken alapuló összeköttetés-létesítés és -bontás állapotdiagramja látható a 6.4. ábrán. Minden állapotátmenetet valamilyen esemény vált ki: vagy egy, a helyi felhasználó által végrehajtott primitív, vagy egy beérkező csomag. Az egyszerűség kedvéért föltételezzük, hogy minden szegmens nyugtázása külön történik.

Feltesszük továbbá, hogy szimmetrikus összeköttetés-bontást modellezünk úgy, hogy a kliens kezdeményez. Megjegyzendő, hogy ez a modell meglehetősen elnagyolt. Később megvizsgálunk egy ennél valósághűbb modellt is, amikor a TCP működését tárgyaljuk.

6.1.3. Berkeley-csatlakozók

Vizsgáljunk meg röviden egy másik szállításiprimitív-készletet, a csatlakozóprimitíveket, ahogyan azokat a TCP-hez használják. A Berkeley-csatlakozók (Berkeley sockets) a Berkeley UNIX 4.2BSD kiadásában mutatkoztak be 1983-ban, és gyorsan nagy népszerűségre tettek szert. A primitíveket elterjedten használják az internet programozásában sok operációs rendszeren, különösen UNIX-alapú rendszereken. Windows-rendszerre is létezik egy hasonló API, amit „winsock" névvel illetnek.

A primitíveket a 6.5. ábrán felsoroltuk. Nagy vonalakban követik az első példában bemutatott modellt, de több lehetőséget és rugalmasságot nyújtanak. Itt nem térünk ki a megfelelő szegmensekre, annak tárgyalására később szorítkozunk.

Az első négy primitívet az ábrán látható sorrendben hajtja végre a szerver. A SOCKET primitív új végpontot (csatlakozót) hoz létre, és táblahelyet foglal le a szállítási entitásban. A hívás paraméterei rögzítik a használni kívánt címzési formát, a szolgáltatás típusát (például megbízható bájtfolyam) és a protokollt. A sikeres SOCKET hívás közönséges állományleíróval tér vissza, amit a további hívások használnak éppúgy, mint az OPEN rendszerhívásnál.

Az újonnan létrehozott csatlakozóknak nincs címük, a hozzárendelést a BIND primitív végzi. Amint a szerver címet rendelt a végponthoz, távoli kliensek csatlakozhatnak hozzá. Annak oka, hogy a cím megadása nem a SOCKET hívással történik az, hogy vannak olyan folyamatok, amelyek számára fontosak a címek (például évek óta ugyanazt a címet használják, és ez a cím mindenki által ismert), míg mások számára a cím megválasztása közömbös.

Ezt követi a LISTEN hívás, amely a beérkező hívások várakozási sorának foglal helyet arra az esetre, amikor a szerverhez egy időben több kliens is kapcsolódni kíván. Ellen-

Primitív	Jelentése
SOCKET	Új kommunikációs végpont létrehozása
BIND	Helyi cím hozzárendelése a csatlakozóhoz
LISTEN	Összeköttetés-elfogadási szándék bejelentése, várakozási sor hosszának megadása
ACCEPT	Bejövő összeköttetés passzív létesítése
CONNECT	Aktív próbálkozás összeköttetés létesítésére
SEND	Adatküldés az összeköttetésen keresztül
RECEIVE	Adatfogadás az összeköttetésről
CLOSE	Összeköttetés bontása

6.5. ábra. *A TCP-csatlakozó primitívek*

tétben az első példában használt LISTEN primitívvel, a csatlakozómodellben a LISTEN nem blokkoló hívás.

A szerver ACCEPT primitívet hajt végre ahhoz, hogy blokkolja magát egy bejövő összeköttetés-kérésig. Amikor egy összeköttetést kérő szegmens érkezik, a szállítási entitás új végpontot hoz létre, az eredetivel azonos tulajdonságokkal, és hozzárendel egy állományleírót. A szerver ekkor új folyamatot vagy szálat indíthat az új végponton létrejövő kapcsolat kezelésére, majd tovább várhatja a következő kérést az eredeti végponton. Az ACCEPT egy szokványos állományleírót ad vissza, amelyet ezután a megszokott módon lehet írásra és olvasásra használni ugyanúgy, mint a tényleges állományok leíróit.

Most vegyük szemügyre a kliensoldalt. Itt ugyancsak egy végpontot kell először létrehozni a SOCKET primitív segítségével, de BIND hívás nem szükséges, mivel a használt cím nem érdekli a szervert. A CONNECT primitív blokkolja a hívót, és belekezd az aktív összeköttetés-létesítési folyamatba. Amikor ezt befejezi (azaz a megfelelő szegmens megérkezett a szervertől), a kliensfolyamat blokkolása megszűnik, és az összeköttetés létrejön. Ekkor mindkét fél a SEND és RECEIVE primitív segítségével küldhet és fogadhat adatokat a duplex összeköttetésen keresztül. Amennyiben a SEND és a RECEIVE különleges lehetőségeire nincsen szükség, a UNIX szabványos READ és WRITE rendszerhívásait is lehet használni.

Az összeköttetés bontása szimmetrikus. Amikor mindkét fél végrehajtotta a CLOSE primitívet, az összeköttetés megszűnik.

A csatlakozók borzasztó népszerűnek bizonyultak, és a szállítási szolgáltatások alkalmazások felé nyújtott absztrakt interfészeinek de facto szabványává váltak. A csatlakozó API-t gyakran használják a TCP-protokollal, hogy összeköttetés-alapú szolgáltatást, ún. **megbízható bájtfolyamot** nyújtsanak, amely gyakorlatilag a már tárgyalt megbízható, bitalapú csővezeték. Mindazonáltal más protokollok is megvalósíthatják ezt a szolgáltatást, ugyanazt az API-t használva. A szállítási szolgáltatás felhasználói számára ennek észrevétlennek kell maradnia.

A csatlakozó API erőssége, hogy az alkalmazás más szállítási szolgáltatásra is használhatja. Például használhat csatlakozót összeköttetés nélküli szállítási szolgáltatásra is. Ebben az esetben a CONNECT primitív a távoli szállítási entitás címét állítja be, és a SEND és RECEIVE primitívek adatcsomagokat küldenek és fogadnak mindkét irányba. (Gyakori megoldás a függvényhívások kiterjesztett halmazának használata, például a SENDTO és RECEIVEFROM, melyek hangsúlyozzák, hogy üzenetről van szó, és nem kötik az alkalmazást egyetlen távoli szállítási entitáshoz.) Lehetőség van csatlakozók használatára olyan szállítási protokollok esetén, amelyek bájtfolyam helyett üzenetfolyamot biztosítanak, torlódáskezeléssel vagy torlódáskezelés nélkül. A **DCCP (Datagram Congestion Controlled Protocol – torlódáskezelő datagram protokoll)** az UDP torlódáskezeléssel kiegészített változata. A szállítási szolgáltatás felhasználóin múlik, hogy tudatában legyenek a kapott szolgáltatás minőségét illetően.

Elképzelhető azonban, hogy a csatlakozók nem az utolsó szót képviselik a szállítási interfészek könyvében. Az alkalmazások például gyakran használnak valamilyen szempont szerint összetartozó folyamokat (ún. folyamcsoportokat), mint amilyet a webböngészők, amelyek számos objektumot kérnek le ugyanattól a szervertől. A csatlakozókkal történő legegyszerűbb megoldás szerint minden objektumra egy saját folyamot használunk, ami azt jelenti, hogy a torlódáskezelést minden folyamra külön-külön alkalmazzuk, nem pedig a teljes csoportra együtt. Ez a megoldás nem optimális, hiszen az alkalmazásra hárítja a folyamcsoportok kezelésének a terhét. Ezért újabb protokollokat találtak ki, amelyek a folyamcsoportok kezelését hatékonyabbá és egyszerűbbé teszik az alkalmazás számára. Két példa ilyen protokollokra az RFC 4960 által definiált **SCTP (Stream Control Transmission Protocol – folyamvezérlő átviteli protokoll)** és az **SST (Structured Stream Transport – strukturált folyam szállítás)** protokoll [Ford, 2007]. Ezek a protokollok némileg változtattak a csatlakozó API-n, hogy a felhasználók a folyam csoportok előnyeit kihasználhassák, továbbá támogatják az összeköttetés-alapú és összeköttetés nélküli forgalom keverését, valamint a többszörös hálózati útvonalakat is. Az idő dönti el, hogy sikeresek lesznek, vagy nem.

6.1.4. Csatlakozóprogramozási példa: egy internetes állományszerver

A csatlakozók (socket) hívásainak használatát a 6.6. ábrán megadott kliens és szerver kódján keresztül mutatjuk be. Az ábrán egy nagyon egyszerű internetes állományszerver látható, valamint egy példa-kliens, amely ezt a szervert használja. A kódnak számos hiányossága van (ezeket meg fogjuk tárgyalni), de elméletileg a szerver kódját bármely internetre kötött UNIX-rendszeren le lehet fordítani és le is lehet futtatni. Ezután a kliens kódja is lefordítható és futtatható a világ bármely másik UNIX-os gépén. A kliens kódja a megfelelő paraméterekkel indítva bármely olyan állományt le tud tölteni a szerverről, amelyhez annak a saját gépén hozzáférése van. Az állományt a kliens a standard outputra teszi, amelyet természetesen tovább irányíthatunk egy állományba vagy egy csővezetékbe.

Elsőként vizsgáljuk meg a szerver kódját! Az eleje néhány szabványos könyvtárillesztést tartalmaz, amelyek közül az utolsó három tartalmazza a legfőbb, internettel kapcsolatos definíciókat és adatszerkezeteket. Ezután a *SERVER_PORT* definíciója következik, amelyet most önkényesen 12 345-re választottunk. Bármely olyan 1024 és 65 535 közötti szám ugyanilyen alkalmas erre a célra mindaddig, amíg egy másik folyamat nem használja azt; az 1023 alatti portszámok kitüntetett felhasználóknak vannak fenntartva.

A szerver következő két sora két szükséges állandót definiál. Az első az állományátvitel során használt adatblokkok méretét határozza meg. A második azt adja meg, hogy hány kapcsolat várakozhat egyszerre, mielőtt a szerver eldobná a további beérkező kéréseket.

A lokális változók deklarációja után kezdődik a szerver tényleges kódja. A program a szerver IP-címét tartalmazó adatszerkezetek inicializálásával indul. Ezt az adatszerkezetet hamarosan a szerver csatlakozójához kötjük majd. A *memset* meghívása az egész adatszerkezetet 0-ba állítja, a következő három hozzárendelés pedig kitölti három mezőjét. Ezek közül a legutolsó tartalmazza a szerver portját. A *htonl* és a *htons* függvények használata azért szükséges, mert az értékeket egy szabványos formára kell alakítanunk annak érdekében, hogy a kód mind a nagy-endián (például SPARC), mind a kis-endián (például Intel x86) számábrázolást használó processzorokon helyesen fusson. A pontos szemantikájuk itt most nem érdekes.

A szerver ezután létrehoz egy csatlakozót és (az $s<0$ feltétellel) ellenőrzi, hogy ez rendben lezajlott-e. A kód termékként kiadott változatában egy hangyányit bőbeszé-

```
/* Ezen az oldalon egy olyan kliensprogram található, amely a következő oldalon látható
   szerverprogramtól le tud kérni egy állományt.
 * A szerver a teljes állomány átküldésével válaszol.
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345        /* tetszőleges, de a két oldalon azonosnak kell lennie */
#define BUF_SIZE 4096            /* átvitt adatblokk mérete */

int main(int argc, char *argv)
{
   int c, s, bytes;
   char buf[BUF_SIZE];           /* puffer az érkező állománynak */
   struct hostent h;             /* info a szerverről */
   struct sockaddr_in channel;   /* ez tárolja majd az IP-címet */

   if (argc !=3) fatal("Indítás: client <szervernév> <állománynév>");
   h = gethostbyname(argv[1]);   /* lekérjük a hoszt IP-címét */
   if (!h) fatal("gethostbyname sikertelen");

   s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
   if (s <0) fatal("csatlakozó");
   memset(&channel, 0, sizeof(channel));
   channel.sin_family= AF_INET;
   memcpy(&channel.sin_addr.s.addr, h->h_addr, h->h_length);
   channel.sin_port= htons(SERVER_PORT);

   c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
   if (c <0) fatal("Az összeköttetés kiépítése sikertelen");

   /* Az összeköttetés létrejött. Elküldjük az állomány nevét, 0-val lezárva. */
   write(s, argv[2], strlen(argv[2])+1);

   /* Kiolvassuk az állományt a csatlakozóról és kiírjuk a strandard outputra. */
   while (1) {
      bytes = read(s, buf, BUF_SIZE);   /* olvasás a csatlakozóról */
      if (bytes <= 0) exit(0);          /* Vége van az állománynak? */
      write(1, buf, bytes);             /* írás a standard outputra */
   }
}
fatal(char *string)
{
   printf("%s\n", string);
   exit(1);
}
```

6.6. ábra. *A csatlakozókat használó kliens kódja. A szerver kódja a következő oldalon található*


```
#include <sys/types.h>                /* Ez a szerver kódja */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345             /* tetszőleges, de a két oldalon azonosnak kell lennie */
#define BUF_SIZE 4096                 /* átvitt adatblokkok mérete */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
   int s, b, l, fd, sa, bytes, on = 1;
   char buf[BUF_SIZE];                /* puffer a kimenő állománynak */
   struct sockaddr_in channel;        /* ez tárolja majd az IP-címet */

   /* Felépítjük a csatlakozóhoz szükséges adatszerkezetet. */
   memset(&channel, 0, sizeof(channel));          /* nullázzuk a csatornát */
   channel.sin_family = AF_INET;
   channel.sin_addr.s_addr = htonl(INADDR_ANY);
   channel.sin_port = htons(SERVER_PORT);

   /* Passzív megnyitás. Összeköttetésre várunk. */
   s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);      /* csatlakozó létrehozása */
   if (s < 0) fatal("socket sikertelen");
   setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

   b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
   if (b < 0) fatal("bind sikertelen");

   l = listen(s, QUEUE_SIZE);         /* megadjuk a várakozási sor hosszát */
   if (l < 0) fatal("listen sikertelen");

   /* A csatlakozó kész és be van kötve. Várjuk az összeköttetéseket, és feldolgozzuk azokat. */
   while (1){
      sa = accept(s, 0, 0);           /* várakozás egy összeköttetési kérésre */
      if (sa < 0) fatal("accept sikertelen");

      read(sa, buf, BUF_SIZE);        /* beolvassuk az állománynevet a csatlakozóról */

      /* Megszerezzük és átvisszük az állományt .*/
      fd = open(buf, O_RDONLY);       /* megnyitjuk a kért állományt */
      if (fd < 0) fatal("open sikertelen");

      while (1){
         bytes = read(fd, buf, BUF_SIZE);   /* olvasás az állományból */
         if (bytes <= 0) exit(0);           /* vége van az állománynak? */
         write(sa, buf, bytes);             /* bájtok kiírása a csatlakozóra */
      }
      close(fd);                      /* állomány lezárása */
      close(sa);                      /* összeköttetés lezárása */
   }
}
```

6.6. ábra. *(folytatás) A szerver kódja*

dűbb hibaüzenetet kellene írni. A *setsockopt* meghívására azért van szükség, hogy a szerver újrahasználhassa a portot, és így határozatlan ideig futhasson és szolgálhassa ki a beérkező kéréseket. Az IP-címet most hozzákötjük a csatlakozóhoz, és azt is megvizsgáljuk, hogy a *bind* hívása sikeres volt-e. Az inicializálás végső lépése a *listen* meghívása, amellyel a szerver bejelenti, hogy hajlandó a bejövő hívások elfogadására, valamint megbízza a rendszert, hogy *QUEUE_SIZE*-nyi kérést várakoztasson abban az esetben, ha akkor érkezik új kérés, amikor a szerver éppen egy másik kérés kiszolgálásán dolgozik. Ha a várakozási sor megtelik, és további kérések érkeznek, akkor azokat a rendszer csendben eldobja.

Ez az a pillanat, amikor a szerver belép a fő ciklusba, amelyet ezután már el sem hagy. Leállításának egyetlen módja az, ha kívülről lövik ki. Az *accept* meghívása addig blokkolja a szervert, amíg összeköttetési kérés nem érkezik egy klienstől. Ha az *accept* hívása sikeres, akkor egy állományleíróval tér vissza, amelyet ezután ahhoz hasonlóan lehet írásra és olvasásra használni, ahogyan egy csővezeték állományleírójával lehet a csővezetéket írni és olvasni. Az egyirányú csővezetékekkel ellentétben azonban a csatlakozók kétirányúak, így az *sa* (az elfogadott csatlakozó) egyaránt használható az összeköttetésről való olvasáshoz, illetve az arra történő íráshoz. A csővezeték állományleíróját lehet használni írásra vagy olvasásra, de nem mindkettőre.

Miután a szerver kiépítette az összeköttetést, kiolvassa az állomány nevét. Ha a név nem áll azonnal rendelkezésre, akkor addig várakozik, amíg meg nem kapja. Miután a szerver megkapta az állomány nevét, megnyitja az állományt és belép abba a hurokba, amely addig olvassa ki sorban az állomány darabjait és írja ki ezeket a csatlakozóra, amíg a teljes állományt át nem másolta. Ezután a szerver lezárja az állományt és az összeköttetést, majd várni kezd a következő kapcsolódási kérésre. Ezt a hurkot örökké ismétli.

Most vizsgáljuk meg a kliens kódját. A működésének megértéséhez elengedhetetlen megérteni azt, hogy hogyan kell indítani. Ha feltesszük, hogy a program neve *client*, akkor egy tipikus hívása a következő:

```
client flits.cs.vu.nl /usr/tom/allomanynev >f
```

Ez a hívás csak abban az esetben sikeres, ha a szerver már fut a *flits.cs.vu.nl*-en, létezik a */usr/tom/allomanynev* nevű állomány, és a szervernek olvasási joga is van rá. Ha a hívás sikeres, akkor a kliensprogram az interneten keresztül megkapja az állományt és kiírja *f*-be, majd kilép. Mivel a szerver egy-egy átvitel után tovább fut, a klienst újra és újra elindíthatjuk, ha más állományokat is meg akarunk szerezni.

A kliens kódja néhány függvénykönyvtár betöltésével és néhány deklarációval kezdődik. Az első dolga ellenőrizni azt, hogy megfelelő számú paraméterrel indították-e a programot (az *argc* = 3 a program nevén kívül még 2 paramétert jelent). Figyeljük meg, hogy az *argv*[1] a szerver nevét tartalmazza (a példában *flits.cs.vu.nl*), és ezt a *gethostbyname* használatával alakítjuk IP-címmé. Ez a függvény a DNS-szolgáltatást használja a cím kikeresésére. A DNS-t a 7. fejezetben fogjuk tanulmányozni.

Ezután a kliens egy csatlakozót (socket) hoz létre és inicializálja azt, majd a *connect* hívással megpróbál létrehozni egy TCP-összeköttetést a szerver felé. Ha a megnevezett gépen fut a szerver, továbbá figyeli a *SERVER_PORT*-ot, és vagy tétlen, vagy van szabad helye a *listen* hívás várakozási sorában, akkor az összeköttetés (előbb vagy utóbb)

kiépül. A kliens ekkor a csatlakozóra történő írással elküldi az állomány nevét az összeköttetésen keresztül. Az elküldött bájtok száma eggyel nagyobb a név tényleges hosszánál, mivel a nevet lezáró 0 bájtot is el kell küldeni a szervernek, hogy az tudja, hol van vége a névnek.

Ekkor a kliens egy olyan hurokba lép, amelyben az állományt blokkonként kiolvassa a csatlakozóról, és rámásolja a blokkokat a standard outputra. Amikor ezzel végzett, egyszerűen kilép.

A *fatal* függvény kitesz egy hibaüzenetet a kimenetre, majd kilép. A szervernek is szüksége van erre a függvényre, de a hely szűkös volta miatt nem írtuk le kétszer. A klienst és a szervert külön fordítják, és általában más számítógépeken futtatják, ezért nem oszthatják meg a *fatal* eljárás kódját.

Ez a két program (a könyvhöz kapcsolódó többi anyaggal egyetemben) megtalálható a könyv weboldalán:

```
http://www.pearsonhighered.com/tanenbaum
```

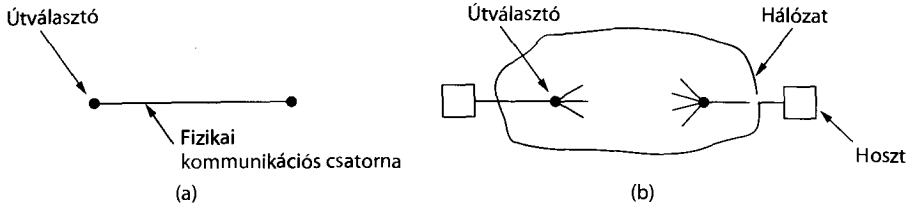
A rend kedvéért meg kell említenem, hogy ez a szerver nem a legjobb a szerverek között. A hibaellenőrzése kevéssé alapos, a hibajelentései középszerűek, valamint a teljesítménye gyenge, mert minden kérést szigorúan soros módon kezel (mivel csak egyetlen szálon fut). Tisztán látszik, hogy sohasem hallott a biztonságról, és a csupasz UNIX-rendszerhívások használata sem a legjobb eszköz a platformfüggetlenség eléréséhez. A program él továbbá néhány olyan, műszakilag lehetetlen feltételezéssel, mint például az a feltételezés, hogy az állománynév belefér a pufferbe, illetve, hogy az egy részben, osztatlanul kerül átvitelre. Mindezen hiányosságai ellenére azonban egy teljes és működőképes internetes állományszerver. A feladatokban az olvasót is buzdítjuk arra, hogy javítson ezen a két programon. A csatlakozók programozásával kapcsolatos további információért lásd Donahoo és Calvert [2008, 2009] művét.

6.2. A szállítási protokollok elemei

A szállítási szolgáltatást egy, a szállítási entitások között használt **szállítási protokoll** valósítja meg. Némely tekintetben a szállítási protokollok az adatkapcsolati protokollokra emlékeztetnek, amelyeket a 3. fejezetben tanulmányoztuk részletesen. Mindkettőnek többek között hibakezelést, sorszámozást és forgalomszabályozást kell végeznie.

Ugyanakkor jelentős eltérések is vannak a kettő között. A különbségek fő oka abban az alapvetően eltérő működési környezetben rejlik, melyben a két protokoll működik. Ezt a 6.7. ábrán láthatjuk. Az adatkapcsolati rétegben a két csomópont közvetlenül egy fizikai csatornán keresztül kommunikál, míg a szállítási rétegben a fizikai csatorna helyett egy egész hálózat szerepel. Ez a különbség fontos hatással van a protokollokra.

Egyrészt kétpontos (point-to-point) adatkapcsolatokon, mint amilyen a rézvezeték vagy az optikai szál, az útválasztónak nem kell kijelölni, hogy melyik másik útválasztóval kíván kommunikálni – minden kimenő vonal egyértelműen azonosít egy adott útválasztót. A szállítási rétegben kötelező a cél explicit címzése.



6.7. ábra. *(a) Az adatkapcsolati réteg környezete. (b) A szállítási réteg környezete*

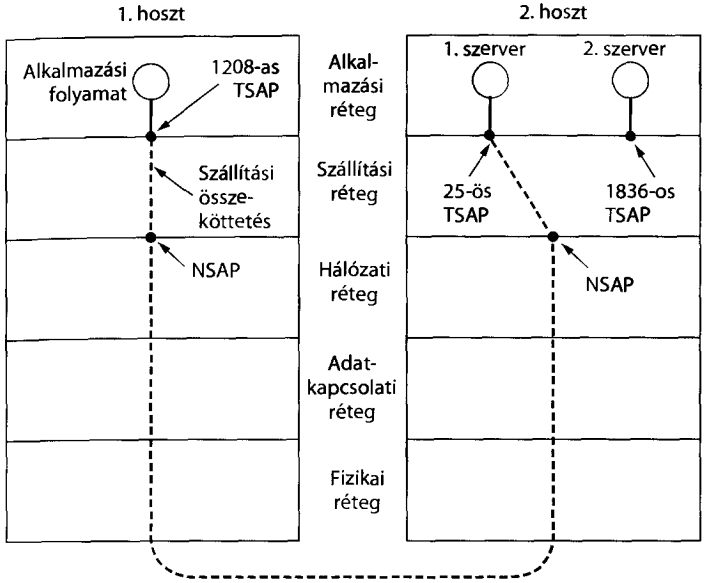
Másrészt, amikor egy folyamat a 6.7.(a) ábrán látható vezetéken összeköttetést akar létesíteni, egyszerű dolga van: a másik végpont mindig jelen van (hacsak el nem romlott, amikor is nincs jelen). Akármelyik eset következik is be, nincs sok tennivaló. Még vezeték nélküli adatkapcsolatokon sincs komolyabb különbség: elég az üzenet egyszerű elküldése is ahhoz, hogy minden címzett állomás megkapja. Ha az üzenetet nem nyugtázták valamilyen hiba miatt, akkor is újra lehet küldeni. A szállítási rétegben, mint látni fogjuk, a kezdeti összeköttetés-létesítés jóval bonyolultabb.

Egy másik nagyon bosszantó különbség az adatkapcsolati és a szállítási réteg között a hálózat potenciális adattároló képessége. Ha egy útválasztó elküld egy csomagot egy adatkapcsolaton, az vagy megérkezik, vagy elvész, de nem fog bolyongani egy darabig, elrejtőzni a világ egy távoli sarkába, majd hirtelen, egy váratlan pillanatban, a jóval később küldött csomagok után felbukkanni. Ha a hálózat a belső forgalmat datagramokkal valósítja meg, amelyek forgalomirányítása külön-külön történik, nem elhanyagolható annak a valószínűsége, hogy a csomag tesz egy festői körutazást, majd késve megérkezik, az elvárt sorrendből kilógva, vagy akár többszöröződve. A hálózat csomagkésleltető és többszöröző képességének következménye néha katasztrofális lehet, és speciális protokollok használatát teszi szükségessé, hogy helyesen tudjunk információt továbbítani.

Az utolsó különbség az adatkapcsolati és a szállítási réteg között inkább mennyiségi, mint minőségi eltérés. Pufferelés és forgalomszabályozás mindkét esetben szükséges, de a szállítási rétegben jelenlevő nagy és változó számú összeköttetés, miközben ezek egymással versengenek, hullámzó sávszélességet igényelnek, s ez eltér attól, mint amit az adatkapcsolati rétegben használtunk. Néhány, a 3. fejezetben tárgyalt protokoll rögzített számú puffert rendel minden vonalhoz, így a beérkező keretek számára mindig van szabad puffer. A szállítási rétegben kezelendő nagyszámú összeköttetés és változó sávszélesség láttán máris kevésbé vonzó ötlet mindegyiknek számos saját puffert lefoglalni. A következő alfejezetekben többek között ezekkel és más fontos problémákkal fogunk foglalkozni.

6.2.1. Címzés

Amikor egy alkalmazási folyamat (például egy felhasználói folyamat) egy távoli alkalmazási folyamattal akar összeköttetést létrehozni, meg kell jelölnie, hogy melyik folyamattal akar kapcsolatba lépni. (Az összeköttetés nélküli szállítási szolgáltatásban is megvan ugyanez a probléma: kinek kell elküldeni az egyes üzeneteket?) Az általánosan használt módszer az, hogy külön szállítási címeket definiálunk az egyes folyamatok részére, amelyeken várhatják az összeköttetési kéréseket. Az interneten ezeket a végpontokat általában **portoknak** hívják. Mi a legáltalánosabb kifejezést, a **TSAP**-t (**Transport Service Access Point – szállítási szolgáltatás hozzáférési pont**) fogjuk használni, hogy egy kitüntetett végpontra utaljunk a szállítási rétegben. Az ezekkel rokon végpontokat a hálózati rétegben (vagyis a hálózati rétegbeli címeket) ugyanígy **NSAP**-nak (**Network SAP – hálózati szolgáltatás hozzáférési pont**) hívják. Az IP-címek például NSAP-k.



6.8. ábra. *A TSAP-k, az NSAP-k és a szállítási összeköttetések*

A 6.8. ábra az NSAP, a TSAP és a szállítási összeköttetés összefüggéseit mutatja be. Az alkalmazási folyamatoknak, mind a kliens, mind a szervergépen egy helyi TSAP-re kell rákapcsolódniuk ahhoz, hogy egy távoli TSAP-vel összeköttetést tudjanak létrehozni. Ezek az összeköttetések az ábrán is látható módon mindkét hoszt NSAP-jén is keresztülhaladnak. Egyes hálózatokban minden számítógép csak egyetlen NSAP-vel rendelkezik, így szükség van egy olyan módszerre, amellyel több szállítási végpontot lehet megkülönböztetni egy NSAP-n belül. Erre a célra alkalmazzák a TSAP-ket.

A szállítási összeköttetés egy lehetséges forgatókönyve:

1. A 2. hoszton található levelezőszerver folyamat a 25-ös TSAP-hez kapcsolódik és bejövő hívásokra várakozik. Az, hogy egy folyamat hogyan tud egy TSAP-re rákapcsolódni, teljesen kívül esik a hálózat modelljén és kizárólag a helyi operációs rendszertől függ. A kapcsolódás történhet például egy a mi LISTEN-ünkhöz hasonló hívással.

2. Az 1. hoszt egyik alkalmazási folyamata szeretne egy elektronikus levelet küldeni, ezért saját magát az 1208-as TSAP-hez kapcsolja, és kiad egy CONNECT kérést. A kérés tartalmazza forrásként az 1208-as TSAP-t az 1. hoszton és célként az 25-ös TSAP-t a 2. hoszton. Ez végül egy szállítási összeköttetés kiépüléséhez vezet az alkalmazási folyamat és a szerver között.

3. Az alkalmazási folyamat ezután elküldi az elektronikus levelet.

4. A levelezőszerver válaszban közli, hogy a levelet kézbesíteni fogja.

5. A szállítási összeköttetést ezután lebontják.

Eközben persze a 2. hoszton más szerverek is várhatnak bejövő összeköttetési kéréseket. Ezek más TSAP-kre csatlakoznak, de az ezek számára érkező kérések is ugyanazon az NSAP-n keresztül érkeznek meg.

Az itt lefestett kép nagyszerű, azonban egyetlen apró kérdést elegánsan a szőnyeg alá söpörtünk: Honnan tudja az 1. hoszt alkalmazási folyamata, hogy a levelezőszerver a 25-ös TSAP-hoz kapcsolódik? Az egyik lehetőség az, hogy a levelezőszerver már évek óta a 25-ös TSAP-hoz csatlakozik, és ezt szép lassan a hálózat minden felhasználója megtanulta. Ebben a modellben a szolgáltatásoknak stabil TSAP-címük van, amelyeket közismert helyeken megtalálható állományokban sorolnak fel. Ilyen például a UNIX-rendszerek */etc/services* állománya, amely felsorolja, hogy mely szerverek mely portokhoz vannak állandó jelleggel hozzárendelve, beleértve azt az egyszerű tényt is, hogy a levelezőszerver a 25-ös TCP-porton található.

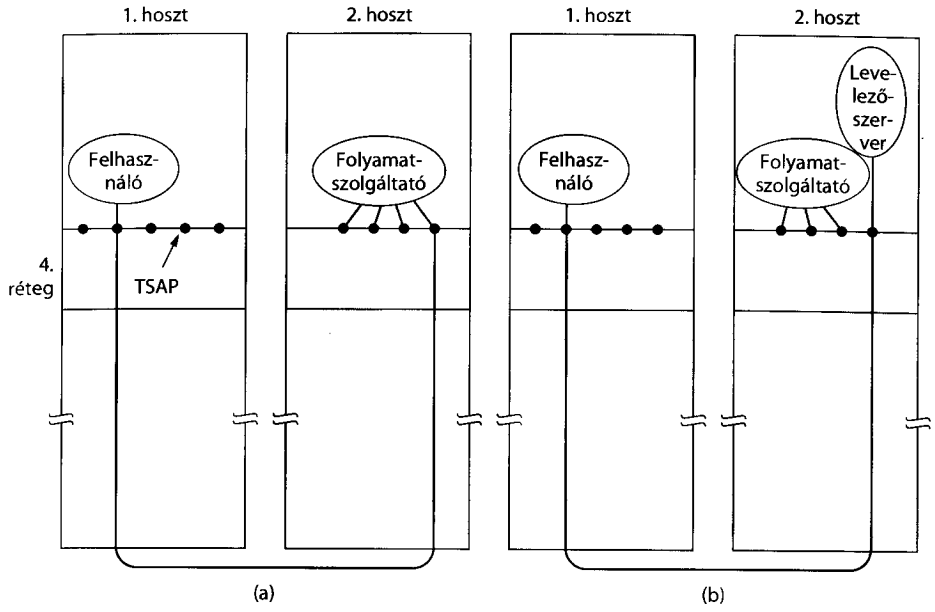
Habár a stabil TSAP-címek jól működnek kisszámú olyan szolgáltatás esetén, amelyek sohasem változnak (például webszerverek), a felhasználói folyamatok (általános értelemben) legtöbbször más olyan felhasználói folyamatokkal akarnak beszélgetni, amelyek nem rendelkeznek előre ismert TSAP-címmel, vagy csak rövid ideig létezik címük.

Az ilyen esetek kezelésére gyakran egy alternatív módszert alkalmaznak. Ebben az esetben létezik egy speciális folyamat, az ún. **portszolgáltató (portmapper)**. Ha a felhasználó egy szolgáltatás nevéhez (például „BitTorrent") keresi a TSAP címét, akkor összeköttetést létesít a portszolgáltató felé (ami egy jól ismert TSAP-címen található). A felhasználó aztán elküld egy üzenetet, ami a szolgáltatás nevét tartalmazza, és a portszolgáltató visszaküldi a szolgáltatás TSAP-címét. A felhasználó végül megszakítja a kapcsolatot a portszolgáltatóval, és kiépít egy újat a kapott TSAP-cím felé.

Ebben a modellben egy új szolgáltatás létesítésekor a szolgáltatás nevével (rendszerint egy ASCII-füzér) és TSAP-címével be kell jelentkezni a portszolgáltatónál. A portszolgáltató a kapott információt feljegyzi belső adatbázisába, hogy a későbbi lekéréseknél már tudja a választ.

A portszolgáltató feladata analóg a tudakozóéval a távbeszélőrendszerekben – nevekről számokra történő leképezést valósít meg. Éppúgy, mint a távbeszélőrendszerekben, lényeges, hogy a portszolgáltató jól ismert TSAP-címe valóban mindenki által ismert legyen. Ha nem tudjuk a tudakozó telefonszámát, nem lehet fölhívni a tudakozót, hogy megkérdezzük. Ha azt gondolnánk, hogy a tudakozó telefonszáma nyilvánvaló, próbáljuk meg valamikor fölhívni egy másik ország tudakozóját.

A szerverfolyamatok nagy részét meglehetősen ritkán használják, ezért pazarló azokhoz folyamatosan egy kitüntetett TSAP-címet rendelni, majd egész nap futtatni. A 6.9. ábrán egy alternatív megoldás leegyszerűsített formája látható, amely **kezdeti összeköttetés-protokollként (initial connection protocol)** ismert. Ahelyett, hogy az összes lehetséges szerver egy jól ismert TSAP-t figyelne, minden olyan gépnek van egy különleges **folyamatszervere (process server)**, amely szolgáltatásokat akar felkínálni a távoli felhasználóknak. A folyamatszerver a kevésbé sűrűn használt szerverek megbízottjaként (proxy) működik. UNIX-rendszereken az ilyen szerver elnevezése *inetd*. Több portot figyel egyszerre, összeköttetési kérésekre várva. A szolgáltatásokat használni kívánók azzal kezdik a tevékenységüket, hogy kiadnak egy CONNECT kérést, amelyben megjelölik, hogy melyik TSAP-címen van az általuk igényelt szolgáltatás. Ha itt nem vár rájuk szerver, akkor a folyamatszerverrel kerülnek kapcsolatba, ahogyan az a 6.9.(a) ábrán is látható.



6.9. ábra. *Az 1. hoszton futó felhasználói folyamat és a 2. gépen futó levelezőszerver közötti összeköttetés felépítése folyamatszerver segítségével*

Miután a folyamatszolgáltató megkapja a beérkező kérést, létrehozza a megfelelő szervert, aminek átadja a felhasználóval már fennálló összeköttetését. A szerver elvégzi a kért feladatot, miközben a folyamatszolgáltató visszatér, és továbbra is kérésekre várakozik. Ezt mutatja be a 6.9.(b) ábra. Ez a módszer csak akkor alkalmazható, ha a szervereket elég szükség esetén létrehozni.

6.2.2. Összeköttetés létesítése

Az összeköttetés felépítése egyszerűnek tűnik, de valójában meglepően trükkös folyamat. Első pillantásra elegendő lenne, hogy az egyik szállítási entitás CONNECTION REQUEST szegmenst küldene a másik félnek, és CONNECTION ACCEPT válaszra várna. A probléma akkor merül fel, ha a hálózatban a csomagok elvesznek, késleltetést szenvednek, megsérülnek, vagy akár megkettőződnek. Ezek a lehetőségek komoly nehézségeket okoznak.

Képzeljük el, hogy egy hálózat annyira zsúfolt, hogy a nyugtázások nemigen érkeznek vissza időben, minden csomag időtúllépéssel érkezik meg, a csomagokat kétszer-háromszor újraküldik. Tegyük föl, hogy a hálózat belül datagramokat használ, és minden csomag különböző útvonalon halad. Néhány csomag dugóba kerülhet a hálózaton, és hosszú ideig nem érkezik meg. A hálózat tehát jelentős ideig tárolja őket, majd jóval később előbukkannak, amikor már a küldő azt gondolná, hogy elvesztek.

A létező legrosszabb rémálom a következő: egy felhasználó összeköttetést létesít egy bankkal, és üzenetet küld azzal a megbízással, hogy a bank utaljon át nagy pénzösszeget egy nem igazán megbízható személy számlájára. Szerencsétlenségére a csomagok éppen úgy döntenek, hogy egy kellemes körutazást tesznek a hálózat legeldugottabb sarkába. A küldő időzítője lejár, és elküldi újra az összes csomagot. Ez esetben a csomagok a legrövidebb úton mennek, és gyors kézbesítés után a küldő bontja az összeköttetést.

Sajnos előbb vagy utóbb az előzőleg küldött csomagok abbahagyják a körutazást, és megérkeznek a célhoz, méghozzá sorrendben, megkérve a bankot, hogy létesítsen egy új összeköttetést, és utalja át a pénzt (megint). A bank semmilyen módon nem tudja megállapítani, hogy ezek másolatok. Azt feltételezi, hogy ez egy második, az előzőtől független tranzakció, és megint átutalja a pénzt.

Bár az előbbi eset elég valószínűtlen, de a lényeg, hogy a protokollokat úgy kell tervezni, hogy minden esetben helyesek legyenek. Jó hálózati teljesítőképesség eléréséhez elegendő csak a gyakori eseteket hatékonyan implementálni, de hiba nélküli működéshez a protokollnak meg kell birkóznia a valószínűtlen esetekkel is. Ha nem, akkor sikerült építenünk egy olyan hálózatot, amely figyelmeztető jelzés nélkül hibázik, ha a körülmények mostohábbra fordulnak.

Jelen alfejezetben a késleltetett kettőzések problémáját fogjuk tanulmányozni. Különös figyelmet szentelünk az összeköttetések megbízható módon történő létesítésére kifejlesztett algoritmusokra, hogy a föntiekhez hasonló rémálmok ne válhassanak valóra. A bökkenő az, hogy a késleltetett kettőződéseket új csomagoknak véljük. A csomagok kettőződését és késleltetését nem tudjuk megakadályozni. Ha azonban ez történik, akkor az ilyen kettőzött csomagokat el kell dobni, és nem szabad új csomagként feldolgozni.

A problémára többféle ellenszer létezik, de egyik sem teljesen kielégítő. Az egyik módszer azt mondja, hogy használjunk eldobható szállítási címeket. Ebben a megközelítésben minden esetben, amikor egy szállítási címre van szükség, újat generálunk. Az összeköttetés lebontása után a régi címet elvetjük, és soha nem használjuk újra. A késéssel érkező kettőződések így soha nem kerülhetnek egy szállítási folyamathoz, és így nem okozhatnak kárt. Ez a megoldás azonban még bonyolultabbá teszi az összeköttetést az előbbi folyamattal.

Egy másik lehetőség minden összeköttetésnek egy olyan egyedi összeköttetés-azonosítót adni (egy sorszámot, ami minden újabb összeköttetés létesítéskor egyesével nő), amelyet a kezdeményező fél generál és minden szegmensbe (beleértve az összeköttetést kérőt is) beletesz. Az összeköttetés lebontása után minden szállítási entitás frissíti a befejeződött összeköttetések tábláját, amelynek bejegyzései (társ szállítási entitás, összeköttetés sorszáma) párokból állnak. Minden újabb összeköttetés-kéréskor ellenőrizhető, hogy az nem egy régi összeköttetéshez tartozik-e.

Sajnos ennek a módszernek van egy alapvető hibája: minden szállítási entitásnak határozatlan ideig történeti információt kell tárolnia. Ráadásul ezt az információt mind a

forrás-, mind a célgépen tárolni kell, különben ha egy gép összeomlik, és memóriatartalmát elveszti, többé nem tudja, hogy mely összeköttetés-azonosítókat használta már.

Ehelyett más megközelítést kell használnunk a probléma leegyszerűsítésére. Ahelyett, hogy egy csomagot örök időre életben hagynánk a hálózatban, ki kell fejlesztenünk egy olyan mechanizmust, amely az öreg és még mindig bolyongó csomagokat kiirtja. Ezzel a megszorítással a probléma valamivel kezelhetőbbé válik.

A csomagok élettartama ismert maximumra korlátozható az alábbi egyik (vagy több) módszerrel:

1. Korlátozott hálózat tervezése.
2. Ugrásszámláló (hop counter) alkalmazása a csomagokban.
3. A csomagok időbélyeggel való ellátása.

Az első módszerbe minden olyan megoldás beletartozik, amely megelőzi, hogy a csomagok hurokba kerüljenek, és emellett valahogyan a torlódási késleltetést is korlátozni tudja a (pillanatnyilag ismert) leghosszabb lehetséges útvonalon. Ez a módszer meglehetősen nehézkes, hiszen az összekapcsolt hálózatok kiterjedése egyetlen várostól kezdve akár nemzetközi méreteket is ölthet. A második módszer abból áll, hogy az ugrásszámot valamilyen alkalmas értékre állítják be, és minden továbbadás alkalmával csökkentik. A hálózati protokoll minden olyan keretet egyszerűen eldob, amelynek az ugrásszámlálója nullára csökkent. A harmadik módszerhez arra van szükség, hogy minden csomagot ellássanak a keletkezésének idejével, valamint az útválasztóknak meg kell egyezniük abban, hogy eldobnak minden olyan csomagot, amely régebbi a közösen meghatározott legnagyobb lehetséges élettartamnál. Ez utóbbi módszer alkalmazásához az útválasztók óráit szinkronizálni kell, amely maga sem egyszerű feladat, és az ugrásszámláló is gyakorlatilag egy jó közelítése az élettartamnak.

Gyakorlatban nem elég azt biztosítanunk, hogy egy csomag halott, hanem ennek igaznak kell lenni minden rá vonatkozó nyugtára is, ezért bevezetjük a T időtartamot, ami a valódi maximális csomagélettartam kis egész számú többszöröse. A maximális csomagélettartam egy adott hálózatra közel állandó érték; az interneten ezt az értéket önkényesen valahová 120 másodperc köré állították be. A szorzó protokollfüggő, egyszerűen azért kell, hogy a T még hosszabb legyen. A csomag elküldését követően T időnyi várakozás után biztosak lehetünk abban, hogy a csomag már minden nyom nélkül eltűnt, és sem a csomag, sem a nyugtázások nem fognak hirtelen előtűnni a ködből, és további bonyodalmakat okozni.

Korlátozott élettartamú csomagokat felhasználva bolondbiztos eljárást lehet kifejleszteni az összeköttetés biztonságos felépítésére. Az alább ismertetett módszer Tomlinson [1975] nevéhez fűződik, melyet Sunshine és Dalal [1978] tovább finomított. Ennek különböző változatait széles körben alkalmazzák a gyakorlatban, így a TCP-ben is.

Az alapötlet az, hogy a forrás felcímkézi a szegmenseket egy sorszámmal, mely sorszámot nem használjuk újra T másodpercen belül. A T periódus és a másodpercenkénti csomagszám meghatározza a sorszámok méretét. Így egy adott sorszámmal csupán egyetlen csomag lehet kinn akármelyik időpillanatban. Másolatok még mindig létezhet-

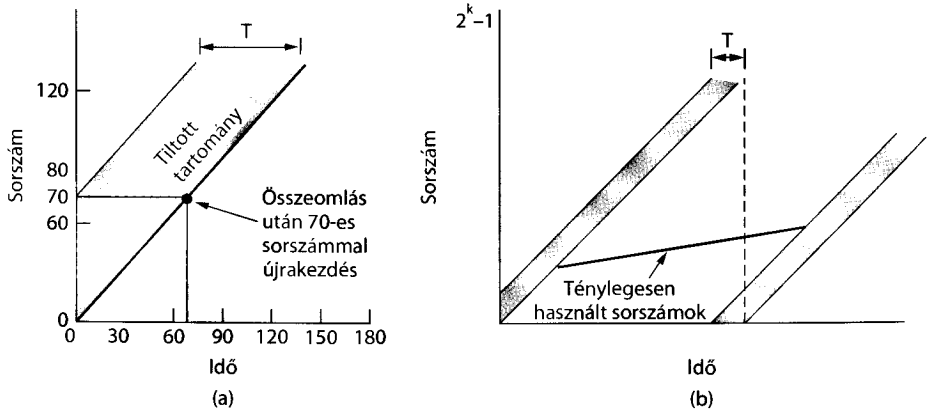
nek, azonban azokat a célállomáson el kell dobni. Így már nem áll fenn az a probléma, hogy egy késleltetett másolat ugyanolyan sorszámmal érkezik, és a cél azt elfogadja.

Annak a problémának megkerülésére, hogy egy gép egy összeomlás után nem tudja megállapítani, hogy hol is tartott, egy lehetséges megoldás, hogy a szállítási entitás a helyreállás után T másodpercig tétlenül várakozzon. Ez idő alatt minden régi szegmens kihal, és a küldő elölről kezdheti tetszőleges sorszámmal. Mindazonáltal egy nagyobb hálózatban a T időtartam olyan nagy lehet, hogy ez a megoldás nem túl vonzó.

Ehelyett Tomlinson azt javasolta, hogy minden hosztot időt mutató órával lássanak el. A különböző hosztokon levő óráknak nem szükséges szinkronban járniuk. Minden óra egy bináris számlálóval valósítható meg, ami egységes időközönként növeli értékét. Ezenkívül a számlálóban levő bitek számának egyenlőnek vagy nagyobbnak kell lennie, mint a sorszámokban levő bitek száma. Végül, ami a legfontosabb, a hoszt meghibásodásakor is tovább kell járnia az órának.

Az összeköttetés felépítésekor az óra értékének alsó k bitje alkotja a kezdeti (szintén k bites) sorszámot. Így, eltérően a 3. fejezetben leírt protokolloktól, minden összeköttetés más sorszámmal kezdi számozni a szegmenseit. A használt tartománynak olyan nagynak kell lennie, hogy mire a sorszámok körbeérnek, az azonos sorszámú, régi szegmensek már rég eltűnjenek. Az idő és a kezdeti sorszámok közti lineáris összefüggést mutatja a 6.10.(a) ábra. A **tiltott tartomány** (**forbidden region**) mutatja azokat az időpontokat, amelyekre egy adott szegmenssorszámot nem szabad használni. Ha a tiltott tartományban található sorszámmal küldünk el egy szegmenst, akkor abban az esetben, ha az késleltetést szenved, előfordulhat, hogy a vevő egy másik, valamivel később útjára bocsátott csomagnak fogja hinni. Például ha a hoszt összeomlik, majd a 70. másodpercben újraindul, akkor az óra alapján fog kezdeti sorszámot választani, nem pedig a tiltott tartományban található alacsonyabb sorszámot.

Ha valamikor a szállítási entitások már megegyeztek a kezdeti sorszámban, bármilyen csúszóablakos protokoll használható forgalomszabályozásra. Ez a csúszóablakos protokoll megtalálja és eldobja a már elfogadott csomagok másolatait. Valóságban a kezdeti sorszám időfüggését jelző vastag görbe nem igazán egyenes, hanem lépcsőzött, mivel



6.10. ábra. *(a) Szegmens nem kerülhet a tiltott tartományba. (b) Az újraszinkronizálási probléma*

az óra értéke diszkrét lépésekben növekszik. Az egyszerűség kedvéért ezt a részletet elhanyagoljuk.

Két szempontot is figyelembe kell vennünk, hogy a csomagok sorszámai ne kerülhessenek a tiltott tartományba. A protokoll kétféleképpen kerülhet bajba. Ha egy hoszt túl gyorsan küld túl sok adatot egy frissen létesített összeköttetésen, az aktuális sorszám–idő görbe meredekebben emelkedhet, mint a kezdeti sorszám–idő görbe, és így a sorszámok a tiltott tartományba kerülnek. Ennek elkerülésére a maximális adatsebesség bármelyik összeköttetésen óraütésenként egy szegmens lehet. Így az összeomlás utáni újraindításkor egy új összeköttetés megnyitása előtt a szállítási entitásnak egy óraütésig kell várni, hogy elkerülje egy sorszám ismételt használatát. Mindkét probléma rövidebb óraütem (1 mikroszekundum vagy kevesebb) használatát teszi indokolttá. Az óra azonban nem is lehet túl gyors a sorszámokhoz képest. C óraütem feltételezésével és S maximális sorszámmérettel be kell tartanunk az $S/C > T$ egyenlőtlenséget, hogy a sorszámok nehogy túl gyorsan átforduljanak.

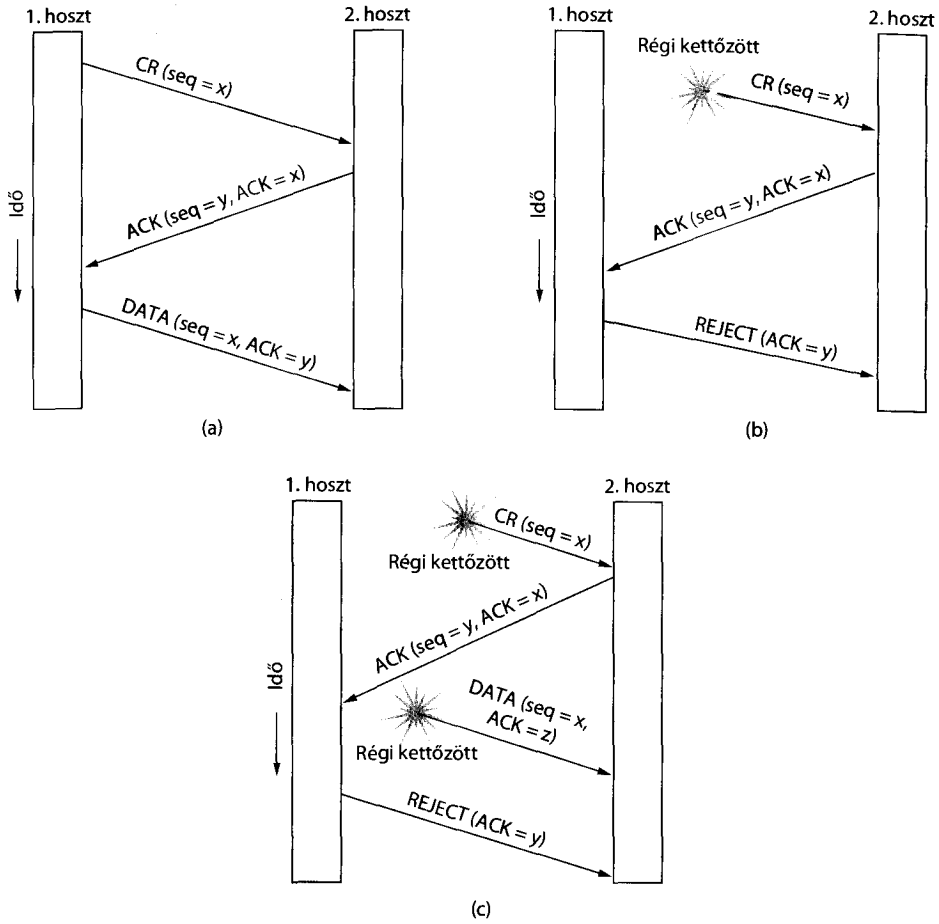
Sajnos, nemcsak úgy lehet bajba kerülni, hogy túl gyors adással alulról kerül a szállítási entitás a tiltott tartományba. A 6.10.(b) ábrán látszik, hogy tetszőleges, az óra sebességénél kisebb adási sebességnél a tényleges idő–sorszám görbe végül balról fog belépni a tiltott tartományba, amint a sorszámok átfordulnak. Minél meredekebb az említett görbe, annál később következik be ez az esemény. A probléma elkerülése érdekében egy összeköttetés sorszámainak növekedési sebességét meg kell szabnunk (vagy az összeköttetés élettartamát kell korlátoznunk).

Az óraalapú módszer megoldja az adatszegmensek késleltetett kettőzési problémáit, de egy akadály még mindig nehezíti a használatát az összeköttetés felépítése során. Mivel általában egy összeköttetésen belül nem jegyezzük meg a sorszámokat, ezért még mindig nem tudjuk megmondani, hogy egy, a kezdeti sorszámot tartalmazó CONNECTION REQUEST szegmens másolat vagy nem. A probléma az összeköttetés ideje alatt nem jelentkezik, hiszen a csúszóablakos protokoll az aktuális sorszámra emlékszik.

Ennek a problémának megoldására vezette be Tomlinson [1975] a **„háromutas kézfogás"** (**three-way handshake**) módszert. Ez az összeköttetés-létesítési protokoll magába foglal egy ellenőrzést, hogy az összeköttetés felépítési kérés érvényes vagy sem. Egy normális összeköttetés-létesítési eljárást, amikor az 1. hoszt kezdeményez, láthatunk a 6.11.(a) ábrán. Az 1. hoszt kiválaszt egy x sorszámot, és egy CONNECTION REQUEST szegmensben elküldi a 2. hosztnak. Az egy ACK szegmenssel nyugtázza x értékét, és bejelenti saját y kezdeti sorszámát. Végül az 1. hoszt jóváhagyja a 2. hoszt által választott kezdeti sorszámot az első általa küldött adatszegmensben.

Lássuk tehát, hogy működik a „háromutas kézfogás" protokollja késleltetett kettőzött vezérlőszegmensek esetén. A 6.11.(b) ábrán az első szegmens egy régebbi összeköttetés késleltetett kettőzött CONNECTION REQUEST üzenete. Ez a 2. hoszthoz anélkül érkezik meg, hogy az 1. hoszt tudna róla. A 2. hoszt válaszul egy ACK szegmenst küld, gyakorlatilag azt ellenőrizve, hogy partnere tényleg új összeköttetést akar-e létesíteni. Mivel az 1. hoszt elutasító választ küld, a 2. hoszt rájön, hogy egy késleltetett kettőzés csapta be, és felhagy az összeköttetés-létesítéssel. Ily módon egy késleltetett kettőzött szegmens nem okoz kárt.

A legrosszabb eset az, amikor mind egy megkésett CONNECTION REQUEST, mind egy ACK kering még valahol a hálózatban. Ezt az esetet mutatja a 6.11.(c) ábra. Az előző



6.11. ábra. *Három forgatókönyv a „háromutas kézfogás" protokollal történő összeköttetés-létesítésre. CR a CONNECTION REQUEST rövidítése. (a) Normális működés. (b) Régi kettőzött CR bukkan elő. (c) Kettőzött CR és kettőzött ACK esete*

példához hasonlóan a 2. hoszt itt is egy megkésett CONNECTION REQUEST-et kap, amelyre válaszol. Itt nagyon fontos azt figyelembe venni, hogy a 2. hoszt kezdetben az *y*-t javasolta a 2. hoszttól az 1. hoszt felé menő forgalom sorszámának, annak a ténynek a teljesen biztos tudatában, hogy olyan szegmens már nem létezik, amely az *y* sorszámot tartalmazza, és ezekre vonatkozó nyugták sincsenek már úton. Amikor a második késve érkező szegmens megérkezik a 2. hoszthoz, az a tény, hogy ez a *z*-t nyugtázza és nem az *y*-t, elárulja a 2. hoszt számára, hogy ez is egy régi kettőzött szegmens. A dolog legfontosabb tanulsága az, hogy a régi szegmensek semmilyen kombinációja sem okozhatja a protokoll hibázását, vagyis sosem hozhat létre olyan összeköttetéseket véletlenül, amelyeket senki nem kért.

A TCP „háromutas kézfogást" használ az összeköttetések felépítésére. Az összeköttetések egy időbélyeget is használnak a 32 bites sorszám mellett, hogy azok ne forduljanak át a maximális csomagélettartam alatt, még gigabites hálózatok esetén sem. Ez

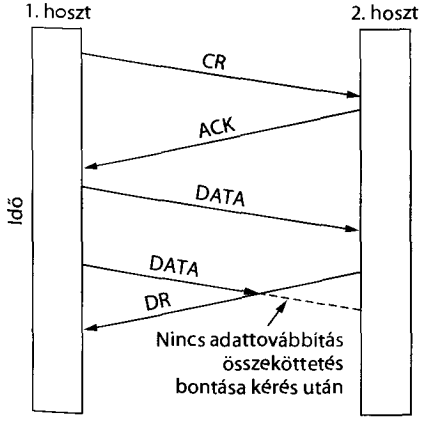
a megoldás egy javítás a TCP-ben, hogy gyorsabb hálózatokon is használni lehessen. A módszert az RFC 1323 írja le, és **PAWS (Protection Against Wrapped Sequence numbers – átfordult sorszámok elleni védelem)** néven ismert. A PAWS előtt a TCP a kezdeti sorszámok megválasztásánál a fenn leírt óraalapú megoldást használta. Kiderült azonban, hogy a módszer a TCP-ben egy biztonsági rést hagyott. Az óra egy támadó számára könnyen megjósolhatóvá tette a következő kezdeti sorszám értékét, és olyan csomagokat küldhetett, amelyek a „háromutas kézfogást" félrevezetik, és hamis összeköttetést hoznak létre. A sérülékenység befoltozására a gyakorlatban álvéletlen kezdeti sorszámokat használnak. Mindazonáltal még mindig fontos tényező, hogy egy ideig ne forduljanak elő a kezdeti sorszámok még akkor sem, ha a kiválasztásuk egy külső megfigyelő számára véletlennek tűnik. Különben nem kerülhető el a késleltetett kettőzött csomagok károkozása.

6.2.3. Összeköttetés bontása

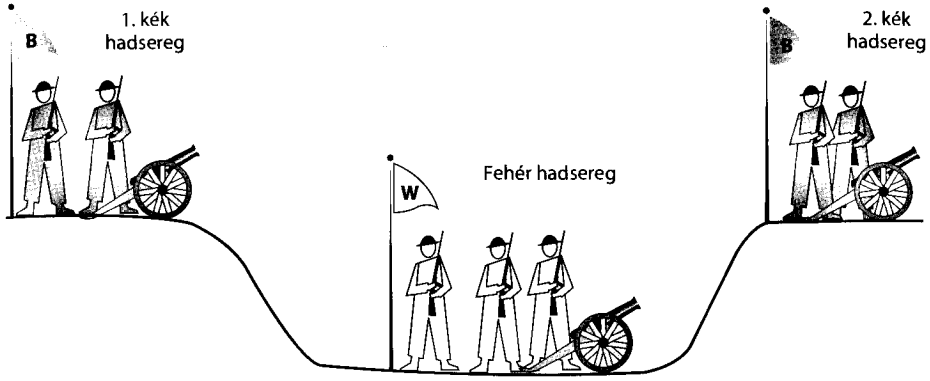
Az összeköttetést bontani jóval egyszerűbb, mint felépíteni. Azonban több buktató van itt, mint gondolnánk. Mint korábban említettük, az összeköttetés kétféleképpen bontható: *aszimmetrikus* és *szimmetrikus* módon. Aszimmetrikus bontást alkalmaznak például a távbeszélő-hálózatokban: amikor az egyik fél leteszi a kagylót, az összeköttetés megszakad. Szimmetrikus bontás esetén az összeköttetést két független egyirányú összeköttetésként kezelik, ahol mindkettőt külön kell lebontani.

Az aszimmetrikus összeköttetés-bontás váratlanul történik, és adatvesztéssel járhat. Vegyük például a 6.12. ábra forgatókönyvét. Az összeköttetés létrejötte után az 1. hoszt egy szegmenst küld a 2. hosztnak, s az rendben meg is érkezik. Az 1. hoszt újabb szegmenst küld. Sajnos a 2. hoszt kiad egy DISCONNECT-et, mielőtt a második szegmens megérkezik. Az összeköttetés lebomlik, és ezzel együtt az adat elvész.

Világos, hogy kifinomultabb bontási protokoll szükséges az adatvesztés elkerüléséhez. Egyfajta megoldás a szimmetrikus bontás használata, amikor is mindkét irányt a



6.12. ábra. *Az összeköttetés hirtelen bontása adatvesztéssel*



6.13. **ábra.** *A „két hadsereg" probléma*

másiktól függetlenül bontjuk le. Itt egy hoszt az után is fogadhat adatot, hogy már elküldött egy DISCONNECTION REQUEST szegmenst.

A szimmetrikus bontás megfelelően működik, ha mindkét félnek rögzített mennyiségű elküldendő adata van, és mindegyik pontosan tudja, hogy mikor küldte el. Más helyzetekben annak eldöntése, hogy végeztek dolgukkal, és a összeköttetés bontható, nem annyira nyilvánvaló. Elképzelhető egy olyan protokoll, melyben az 1. hoszt így szól: „Végeztem. Te is készen vagy?" „Én is elkészültem. Az összeköttetés biztonságosan bontható! Szervusz!" – válaszol a 2. hoszt.

Sajnos, ez a protokoll nem mindig működik. Egy híres, a **„két hadsereg" probléma** néven ismert feladat éppen erről szól. Képzeljük el, hogy a fehér hadsereg a völgyben táborozik, amint a 6.13. ábra mutatja. Mindkét környező dombot a kék hadsereg birtokolja. A fehér sereg bármelyik kék hadseregnél nagyobb, azonban azok együtt nagyobbak a fehér seregnél. Ha bármelyik kék hadsereg egyedül támad, biztos vereséget szenved, de együttes támadásuk során megsemmisíthetnék a fehér sereget.

A kék seregek össze akarják egyeztetni támadásukat. Azonban az összes kommunikációs lehetőségük a völgyön gyalog átküldött futárokra korlátozódik, akiket persze elfoghatnak, így az üzenet elvész (tehát megbízhatatlan kommunikációs csatornát használnak). Az a kérdés, hogy létezik-e olyan protokoll, ami győzelemre segíti a kék seregeket?

Tegyük fel, hogy az 1. kék hadsereg parancsnoka a következő üzenetet küldi: „Azt javaslom, hogy március 29-én hajnalban támadjunk. Mi a véleményetek?" Tegyük fel továbbá, hogy az üzenet megérkezik, a 2. kék hadsereg parancsnoka egyetért, és válasza szerencsésen megérkezik az 1. kék sereghez. Végrehajtják a támadást? Valószínűleg nem, mert a 2. sereg parancsnoka nem tudja, hogy üzenete átjutott-e a völgyön. Ha nem, az 1. sereg nem fog támadni, így egymaga bolond lenne fölvenni a küzdelmet.

Bővítsük hát a protokollt a „háromutas kézfogás" technikával. Az eredeti javaslat kezdeményezőjének nyugtáznia kell a kapott választ. Feltéve, hogy nem veszett el üzenet, a 2. kék hadsereg megkapja a nyugtát, de most az 1. kék sereg parancsnoka fog habozni. Végül is ő nem tudja, hogy a nyugta átjutott-e vagy sem, és ha nem, tudja, hogy nem számíthat a 2. sereg támadására. Használhatnánk „négyutas kézfogást" is, de az sem segítene.

Valójában könnyen bebizonyítható, hogy nem lehet működő protokollt létrehozni. Tegyük fel mégis, hogy van ilyen. A protokoll utolsó üzenete vagy lényeges, vagy nem. Utóbbi esetben hagyjuk el az összes többi fölösleges üzenettel együtt, amíg olyan protokollhoz nem jutunk, melynek minden üzenete nélkülözhetetlen. Mi történik, ha az utolsó üzenet nem jut át? Mivel erről megállapítottuk, hogy lényeges, így ha elvész, nem kezdődik el a támadás. Mivel az utolsó üzenet küldője sohasem lehet biztos annak célba érkezésében, nem kockáztatja meg a támadást. Sőt ami még rosszabb, ezt a másik kék sereg is tudja, így ők sem támadnak.

Hogy észrevegyük a „két hadsereg" probléma és az összeköttetések bontása között vonható párhuzamot, helyettesítsük a „támadás" szót a „bontás"-sal. Ha egyik fél sem készült föl a bontásra addig, míg meg nem győződött arról, hogy partnere is fölkészült, az összeköttetés bontására sohasem kerül sor.

A gyakorlatban elkerülhetjük ezt a bizonytalanságot azáltal, hogy nem tesszük szükségessé a felek közti megegyezést, és a problémát a szállítási felhasználókra bízzuk, hogy külön-külön döntsenek a bontásról. Ezt már sokkal könnyebb megoldani. A 6.14. ábrán négy forgatókönyvet mutatunk az összeköttetés-bontásra „háromutas kézfogás" használata esetén. Bár ez a protokoll nem sebezhetetlen, mégis kielégítően viselkedik.

A 6.14.(a) ábrán a normális működést láthatjuk, ahol az egyik partner az összeköttetés bontásának kezdeményezéseként DR (DISCONNECTION REQUEST) szegmenst küld. Amikor ez megérkezik, a vevő szintén visszaküld egy DR szegmenst, és elindít egy időzítőt arra az esetre, ha az általa küldött DR elveszne. Amikor ez a DR megérkezik, a kezdeményező visszaküld egy ACK szegmenst, és bontja az összeköttetést. Végül, mikor az ACK szegmens megérkezik, a vevő szintén bontja az összeköttetést. Az összeköttetés bontása azt jelenti, hogy a szállítási entitás az összeköttetésre vonatkozó információt törli a nyitott összeköttetések táblájából, és jelzi az összeköttetés befejezését a tulajdonosnak (a szállítási felhasználónak). Ez a művelet eltér attól, amikor a szállítási felhasználó kiad egy DISCONNECT primitív hívást.

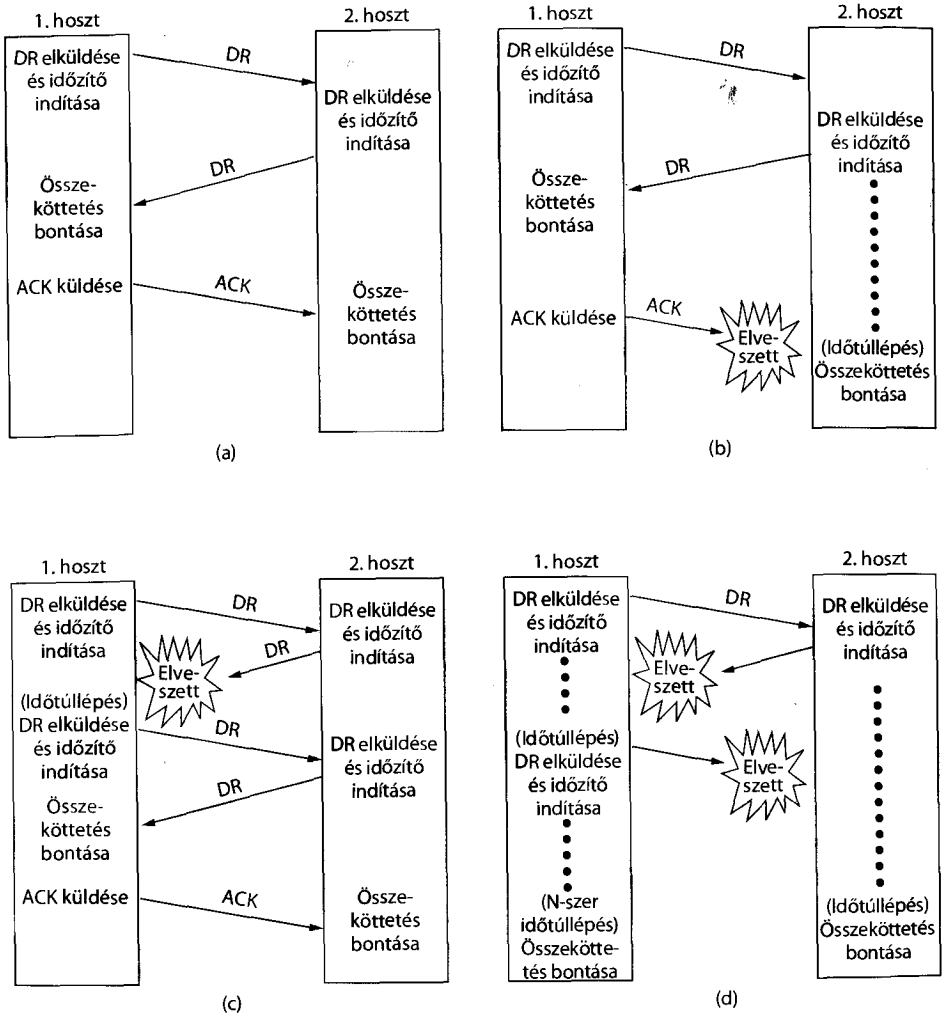
Ha az utolsó ACK szegmens elvész, ahogy az a 6.14.(b) ábrán látható, a helyzetet az időzítő menti meg. Amikor az lejár, az összeköttetés mindenképpen befejeződik.

Most tekintsük azt az esetet, amikor a második DR vész el. Az összeköttetés bontását kezdeményező fél nem kapja meg a várt választ, lejár az időzítője, és az egészet elkezdi elölről. A 6.14.(c) ábrán látható ez a működés, feltételezve, hogy a második próbálkozás során nem vesznek el a szegmensek, és mindegyik időben meg is érkezik.

Az utolsó eset, amit a 6.14.(d) ábrán láthatunk, azonos az előzővel, kivéve, hogy most feltételezésünk szerint minden további kísérlet a DR újraküldésére meghiúsul a szegmensek elvesztése következtében. *N* próbálkozás után a küldő föladja, és fölbontja az összeköttetést. Eközben a vevő időzítése szintén lejár, és ő is bontja az összeköttetést.

Bár ez a protokoll rendszerint sikeresen működik, elméletileg kudarcot vallhat, ha a kezdeti DR és a további *N* újraküldött szegmensek mind elvesznek. A küldő feladja és bontja az összeköttetést, míg a másik fél semmit sem tud a bontási kísérletekről, és még mindig teljesen aktív. Ennek a helyzetnek az eredménye egy félig nyitott összeköttetés.

Ez a probléma elkerülhető lenne, ha nem hagynánk, hogy a küldő *N* próbálkozás után föladja a kísérletezést, hanem kényszerítenénk, hogy örökké folytassa, amíg választ nem kap. Ha viszont a másik fél időzítést figyel, és az lejár, a küldő ténylegesen örökké fog



6.14. ábra. *Négy protokoll forgatókönyv az összeköttetés lebontására. (a) Normális működés a „háromutas kézfogás"-sal. (b) A végső ACK vész el. (c) A válasz vész el. (d) A válasz és a rákövetkező DR-ek vesznek el*

próbálkozni, mert többé nem is kaphat választ. Ha nem engedjük a fogadóoldalt, hogy időtúllépés esetén a várakozást feladja, a 6.14.(d) ábrán látható protokoll elakad.

A félig nyitott összeköttetések kilövésének egyik módja a következő: bevezetünk egy olyan szabályt, miszerint ha adott ideig nem érkezik szegmens, az összeköttetést automatikusan bontjuk. Ily módon, ha valamelyikük befejezi az összeköttetést, a másik észreveszi a forgalom hiányát, és szintén bontja az összeköttetést. Ez a szabály továbbá megoldja azt a helyzetet is, amikor az összeköttetés megszakad (például mert a hálózat nem képes a továbbiakban csomagokat szállítani a hosztok között) anélkül, hogy bármely fél bontást kezdeményezett volna. Természetesen ennek a szabályozásnak a bevezetése szükségessé teszi egy időzítő alkalmazását minden szállítási entitásnál, amit az minden szegmens elküldésekor nulláz és újraindít. Ha lejár, akkor egy üres (adatot nem hordozó) szegmenst küld, hogy visszatartsa a vevőt az összeköttetés bontásától. Ha azonban az automatikus lebontási szabályt alkalmazzuk, és túl sok egymás után küldött üres szegmens elvész az amúgy kihasználatlan összeköttetésen, először az egyik, majd a másik oldal is bontja az összeköttetést.

Nem ragozzuk tovább ezt a kérdést, de mostanra már az olvasó bizonyára megértette, hogy egy összeköttetés adatvesztés nélküli lebontása közel sem annyira egyszerű, mint amilyennek az első ránézésre tűnik. Amit mégis megtanultunk az, hogy a szállítási felhasználónak kell eldönteni, mikor bontsuk az összeköttetést – a problémát a szállítási entitások csupán egymás közt nem tudják megoldani. Ahhoz, hogy lássuk, milyen fontos szerepe van az alkalmazásnak, vegyük figyelembe, hogy míg a TCP általánosan szimmetrikus bontást végez (azaz mindkét fél az adatok elküldése után FIN szegmensekkel külön-külön lezárja a összeköttetés egyik felét), a kliens felé sok webszerver küld RST szegmenst, amelyek azonnal az összeköttetés bontását eredményezik. Ez leginkább az aszimmetrikus bontásra hasonlít. A módszer azért működőképes, mert a webszerver pontosan ismeri az adatcsere módját. Először is a szerver egy kérést kap a klienstől, ami tartalmazza az összes adatot, amit a kliens szeretne megkapni, majd a szerver visszaküldi a választ a kliens felé. Amikor a webszerver elküldte a választ, az adatáramlás mindkét irányban véget ért.[1] A szerver küldhet a kliensnek egy figyelmeztetést, majd azonnal bonthatja az összeköttetést. Ha a kliens megkapja a figyelmeztetést, akkor ott helyben lezárja az összeköttetést, egyéb esetben, ha a figyelmeztetés nem érkezik meg, akkor a kliens végül észreveszi, hogy a szerver nem kommunikál vele, és lezárja az összeköttetést. Az adatokat mindkét irányban sikeresen átvitték.

6.2.4. Hibakezelés és forgalomszabályozás

Az összeköttetés-létesítés és -lebontás többé-kevésbé részletes tárgyalása után vizsgáljuk meg, hogyan kezelik az összeköttetéseket használat közben. A két kulcskérdés a hibakezelés és a forgalomszabályozás. A hibakezelés biztosítja azt, hogy az adatok megfelelő biztonsággal érkezzenek meg, azaz általában hibátlanul. A forgalomszabályozás teszi lehetővé, hogy egy gyors adó nehogy adatokkal túltöltsön egy lassú vevőt.

Mindkét témakör felbukkant már korábban, amikor az adatkapcsolati réteget tanulmányoztuk. Az ott megismert módszereket használják a szállítási rétegben is. Rövid ismétlésként:

1. A keret egy hibajelző kódot tartalmaz (például CRC-t vagy ellenőrző összeget), amelyet az információ helyességének ellenőrzésére használnak.

2. A keret tartalmaz egy azonosító sorszámot, és a küldő egészen addig rendszeresen újraküldi a keretet, amíg egy pozitív nyugtát nem kap a vevőtől, jelezve, hogy az

1 Ez persze nem mindig igaz, hiszen a Keep-Alive HTTP opció esetén a webszerver nem zárja az összeköttetést, és felkészül további kérések fogadására. *(A fordító megjegyzése)*

sikeresen megkapta a keretet. Ezt **ARQ**-nak (**Automatic Repeat reQuest – automatikus ismétléskérés**) nevezik.

3. A küldő meghatároz egy számot a kint lévő keretek maximális mennyiségére bármilyen tetszőleges időpontban, aminél többet nem küld, és megáll, ha a vevő nem nyugtázza a kereteket elég gyorsan. Ha ez a maximum egyetlen keret, akkor a protokollt **megáll-és-vár** típusú protokollnak hívják. Nagyobb ablakméretek lehetővé teszik a csővezetékezést és a teljesítőképesség növelését nagy sebességű és hosszú összeköttetéseken.

4. A **csúszóablakos** protokoll egyesíti ezeket az eszközöket, és mindemellett használják a kétirányú adatátvitel megvalósítására is.

Mivel ezeket a módszereket az adatkapcsolati rétegben a kereteken már alkalmazták, érthető a kíváncsiság, miért alkalmazzák ezeket a szállítási rétegben a szegmensekre is. Valóban, van egy kevés hasonlóság az adatkapcsolati és szállítási rétegek között a gyakorlatban. Hiába azonosak azonban a módszerek, mind céljukban, mind mértékükben vannak különbségek.

Például nézzük meg, miért különböznek céljukban! Az adatkapcsolati réteg ellenőrző összege megvédi a keretet, miközben áthalad egy összeköttetésen. A szállítási réteg ellenőrző összege a szegmenst védi, miközben egy egész hálózati útvonalon áthalad. Ez utóbbi egy végponttól végpontig tartó ellenőrzés, ami nem azonos az adatkapcsolatonkénti ellenőrzéssel. Saltzer és mások [1984] mutattak példát arra, amikor a csomagok éppen az útválasztón belül sérültek meg. Az adatkapcsolati ellenőrző összeg megvédte a csomagokat az útválasztók közötti összeköttetéseken, de nem magában az útválasztóban. Így végül a csomagok hibásan érkeztek meg annak ellenére, hogy az ellenőrzés egyik összeköttetésen sem mutatott ki hibát.

Ez és egyéb példák késztették Saltzert és társait arra, hogy megfogalmazzák a **végponttól végpontig elvüket** (**end-to-end argument**). E szerint a szállítási rétegben történő ellenőrzések nélkülözhetetlenek a hibátlan működéshez, míg az adatkapcsolati rétegben történő ellenőrzések csupán a teljesítőképesség növelésében játszanak szerepet (mivel nélkülük egy hibás csomag teljesen feleslegesen végigjárná a hálózatot).

A mértékbeli különbségek megmutatásához vegyük az újraküldést egy csúszóablakos protokoll esetén. A legtöbb vezeték nélküli kapcsolaton (kivéve a műholdas összeköttetéseket) legfeljebb egy keret lehet kinn, hiszen a sávszélesség-késleltetés szorzat annyira kicsi, hogy még egy keret is alig tárolható a kapcsolaton. Ez esetben kicsi ablakméret is elegendő a jó teljesítményhez. Például a 802.11 megáll-és-vár protokollt használ, tehát minden keretnél elküldi azt, majd vár a nyugtára, esetleg újraküldi a keretet. Egészen addig nem lép a következő keretre, amíg nem érkezik meg a nyugta. Egynél nagyobb ablakméret nem okozná a teljesítmény javulását, azonban jelentősen növelné a protokoll bonyolultságát. Vezetékes és optikai adatkapcsolatok esetén, mint például a (kapcsolt) Ethernetnél vagy az ISP-k gerinchálózatain, a hibaarány elég kicsi ahhoz, hogy mellőzhessük az adatkapcsolati rétegben történő újraküldéseket, mivel a végponttól végpontig történő újraküldés majd kijavítja a maradék keretvesztéseket.

Másrészről, sok TCP-összeköttetés sávszélesség-késleltetés szorzata sokkal nagyobb, mint egyetlen szegmens. Vegyünk például egy olyan összeköttetést, amely adatokat továbbít az USA-n keresztül 1 Mbit/s sebességgel és 100 msec egyirányú terjedési idővel. A vevő még egy ilyen lassú összeköttetésen is 200 kbit mennyiségű adatot fog tárolni annyi idő alatt, amennyi egy szegmens elküldésétől a nyugta megérkezéséig tart. Ilyen esetekben tanácsos nagy ablakméretet használni. A megáll-és-vár protokoll lerontja a teljesítőképességet. A példánkban a teljesítőképességet 200 ms-ként egy szegmensre (vagy 5 szegmens/másodpercre) korlátozná függetlenül a hálózat sebességétől.

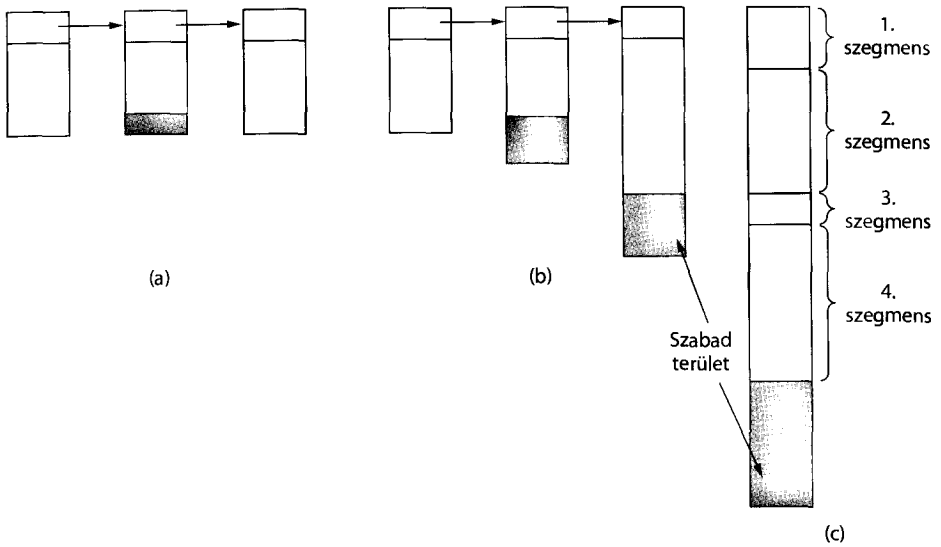
Mivel a szállítási protokollok általában nagyobb ablakméretet használnak, ezért részletesebben áttekintjük a pufferelés kérdését. Mivel egy állomásnak több élő összeköttetése is lehet, melyeket elkülönítve kell kezelni, tekintélyes méretű puffernek kell rendelkezésre állni a csúszóablakok számára. A pufferek szükségesek mind a fogadó-, mind a küldőoldalon. A küldőoldalon az elküldött, de nem nyugtázott szegmensek számára kellenek a pufferek, hiszen ezek a szegmensek még elveszhetnek, és ekkor újra kell őket küldeni.

Mivel azonban a küldő elvégzi a pufferelést, a vevő eldöntheti, hogy hozzárendelt (dedikált) speciális puffereket használ a speciális összeköttetésekhez, vagy nem. A vevő például fenntarthat egy pufferkészletet, megosztva az összeköttetések között. Amikor egy szegmens érkezik, megpróbál dinamikusan egy új puffert foglalni; ha sikerül, akkor elfogadja, egyébként eldobja a szegmenst. Mivel a küldő kész újraküldeni a hálózatban elveszett szegmenseket, nem probléma, ha a vevő eldob szegmenseket, bár ezzel erőforrást pazarol. A küldő úgyis addig próbálkozik, míg végül nyugtát nem kap.

A legjobb kompromisszum a forráspufferelés és célpufferelés között az összeköttetés forgalmának típusától függ. Kis sávszélességű, löketes forgalomra, mint amilyet például egy interaktív terminál állít elő, célszerűbb dinamikusan puffert foglalni mindkét oldalon, és bízni a küldő pufferelésében, ha véletlenül egy szegmenst el kell dobni, mint dedikált puffereket használni. Másrészt fájlátvitelhez vagy nagy sávszélességű forgalomhoz jobb dedikált puffereket használni, hogy az adatok teljes sebességgel áramolhassanak. Pontosan ezt a stratégiát alkalmazza a TCP is.

Még mindig marad azonban egy kérdés: hogyan szervezzük a pufferkészletet? Ha a legtöbb szegmens nagyjából azonos méretű, kézenfekvő a pufferterületet azonos méretű pufferek készletébe szervezni úgy, hogy egy szegmensre egy puffer jusson, amint azt a 6.15.(a) ábra mutatja. Ha azonban a szegmensek mérete széles határok közt változhat – egy weboldal lekéréséhez szükséges kéréstől a P2P-állományátvitelkor mozgatott nagy csomagokig –, a rögzített méretű pufferek problémát jelentenek. Ha a pufferméretet a leghosszabb előforduló szegmens méretében állapítjuk meg, rövid szegmens vétele esetén a hely nagy része kihasználatlan marad. Ha a pufferméret kisebb a szegmens maximális hosszánál, egy nagy szegmens több puffert és bonyolultabb kezelést igényel.

A pufferméret problémájának másik kezelési módja a változó méretű pufferek használata, mint azt a 6.15.(b) ábra mutatja. Ennek előnye a jobb memóriakihasználtság, ami viszont bonyolultabb pufferkezeléssel jár. A harmadik lehetőség összeköttetésenként egyetlen nagy körpuffer alkalmazása, mint az a 6.15.(c) ábrán látható. Ez a megoldás egyszerű és elegáns, nem függ a szegmensmérettől, de csak akkor használja ki hatékonyan a memóriát, ha az összeköttetés erősen leterhelt.



6.15. ábra. *(a) Láncolt, rögzített méretű pufferek. (b) Láncolt, változó méretű pufferek. (c) Összeköttetésenként egyetlen nagy körpuffer*

Ahogy összeköttetések létrejönnek és lebomlanak, és ahogy a forgalom jellege változik, úgy kell az adónak és a vevőnek ehhez dinamikusan hozzáigazítani a pufferfoglalást. Eszerint a szállítási protokollnak lehetővé kell tennie a küldőentitás számára, hogy a másik féltől pufferterület lefoglalását kérje. Puffereket összeköttetésenként vagy a két hoszt között élő minden összeköttetésre együtt lehet lefoglalni. Alternatív megoldásként a vevő a jövendő forgalmat nem, de saját puffereinek adatait ismerve közölheti az adóval: „Lefoglaltam részedre *X* db puffert." Ha a nyitott összeköttetések száma növekszik, akkor az eddigi foglalásokat csökkenteni kell, tehát a protokollnak erre is lehetőséget kell adnia.

A pufferek dinamikus kezelésének egy ésszerű, általános módja a pufferelésnek a nyugtázástól való különválasztása. Ez a 3. fejezetben leírt csúszóablakos protokollal ellentétes működést jelent. A dinamikus pufferkezelés valójában változó méretű csúszóablakot jelent. Először az adó a forgalom becsült igénye alapján adott mennyiségű puffert kér a vevőtől. A vevő annyi puffert ad, amennyit adhat. Az adó minden szegmens küldésekor ezt a számot csökkenti, és leáll az átvitellel, ha eléri a nullát. A vevő közben a visszafelé menő forgalomra ülteti rá a nyugtákat és a pufferek foglaltsági jellemzőit. A TCP is ezt a sémát alkalmazza, és a pufferek foglaltsági jellemzőit egy *Window size* (*Ablakméret*) nevű fejlécmezőben közli.

A 6.16. ábrán egy példát mutatunk arra, hogy hogyan működne a dinamikus ablakkezelés datagramalapú hálózat fölött négybites sorszámokkal. Példánkban az adatok szegmensekben áramlanak *A*-tól *B* felé, a nyugták és a pufferfoglaltsági jellemzők pedig ugyancsak szegmensekben áramlanak, de ellentétes irányba. Először az *A* hoszt 8 puffert szeretne, de ezekből csak négyet kaphat. *A* ezután elküld három szegmenst, amelyek közül az utolsó elvész. A 6. szegmens nyugtáz az 1-es sorszámúval bezárólag (azt is beleértve) minden elküldött szegmenst, lehetővé téve *A*-nak, hogy azok puffereit fölszabadítsa. Ezenfelül arról is értesíti *A*-t, hogy újabb három szegmenst küldhet 1-es fölötti sorszámmal (tehát a 2-es, 3-as és 4-est). *A* tudja, hogy a kettes sorszámút már továbbította, úgy gondolja, hogy elküldheti a rákövetkező kettőt, és így is tesz. Ezen a ponton blokkolódik, és további pufferek lefoglalására kell várakoznia. Viszont az időtúllépés miatti újraküldés blokkolt állapotban is lehetséges (9-es sor), mivel ilyenkor az elküldendő szegmens már pufferben van. A 10-es sorban *B* nyugtázza minden szegmens megérkezését a 4-es sorszámúval bezárólag (azt is beleértve), de nem hagyja *A*-t továbbadni. Ez a helyzet nem fordulhat elő a 3. fejezetben tárgyalt rögzített ablakméretű protokolloknál. A *B* által küldött következő szegmens újabb puffert foglal, így *A* továbbhaladhat. Ez akkor történhet meg, ha *B*-nek rendelkezésére áll puffer, például azért, mert a szállítási felhasználó több szegmens adatrészét átvette.

	A	Üzenet	B	Megjegyzés
1	→	<8 puffer kérése >	→	A 8 puffert szeretne
2	←	<nyug = 15, puff = 4 >	←	B csak 0-3 sorszámú üzeneteket engedélyezi
3	→	<sorsz = 0, adat = m0 >	→	A-nak 3 puffere maradt
4	→	<sorsz = 1, adat = m1>	→	A-nak 2 puffere maradt
5	→	<sorsz = 2, adat = m2 >	...	Az üzenet elveszett, de A azt hiszi, hogy már csak egy puffere maradt
6	←	<nyug = 1, puff = 3 >	←	B nyugtázza 0 és 1 sorszámúakat, 2-4-et engedélyezi
7	→	<sorsz = 3, adat = m3 >	→	A-nak 1 puffere maradt
8	→	<sorsz = 4, adat = m4 >	→	A-nak nincs több puffere, le kell állnia
9	→	<sorsz = 2, adat = m2 >	→	A időzítése lejár és újraküldi a 2-es üzenetet
10	←	<nyug = 4, puff = 0 >	←	Minden nyugta megérkezett, de A még mindig blokkolt
11	←	<nyug = 4, puff = 1 >	←	A most küldheti az 5-ös üzenetet
12	←	<nyug = 4, puff = 2 >	←	B valahol talált egy újabb puffert
13	→	<sorsz = 5, adat = m5 >	→	A-nak 1 puffere maradt
14	→	<sorsz = 6, adat = m6 >	→	A most ismét blokkolódik
15	←	<nyug = 6, puff = 0 >	←	A továbbra sem küldhet
16	...	<nyug = 6, puff = 4 >	←	Potenciális holtpont

6.16. ábra. *Dinamikus pufferfoglalás. A nyilak az átvitel irányát mutatják, a három pont (...) elveszett szegmenst jelöl*

Ilyen jellegű pufferfoglalási módszernél problémát jelenthet, ha a szállítási réteg datagramalapú hálózatra épül, és egy vezérlőszegmens vész el – ami persze könnyen megtörténhet. Nézzük a 16. sort! *B* újabb puffereket foglalt le *A* részére, de a szegmens, amivel erről *A*-t tájékoztatná, elveszett. Hoppá! Mivel a vezérlőszegmensek nem kapnak sorszámot, és időzítő se figyeli hiányukat, az *A* holtpontba kerül. Az ilyen helyzetek elkerülésére mindkét hoszt rendszeres időközönként vezérlőszegmenseket küld társának, amelyekben nyugta és az összeköttetésekhez tartozó pufferek állapotáról szóló információ van. Ily módon előbb-utóbb föloldódik a holtpont.

Eddig hallgatólagosan feltételeztük, hogy a küldő adási sebességének egyedül a vevő szabad puffereinek száma szab határt. Leggyakrabban azonban nem ez a helyzet, ugyanis a memóriaárak rohamos csökkenése miatt lehetséges annyi memóriát zsúfolni

a hosztokba, hogy a szabad pufferek hiánya ritkán vagy egyáltalán nem okoz problémát, még nagy kiterjedésű hálózatokban sem. Természetesen ez akkor igaz, ha a pufferek méretét elég nagyra választják, ami nem volt mindig így a TCP esetén [Zhang és társai, 2002].

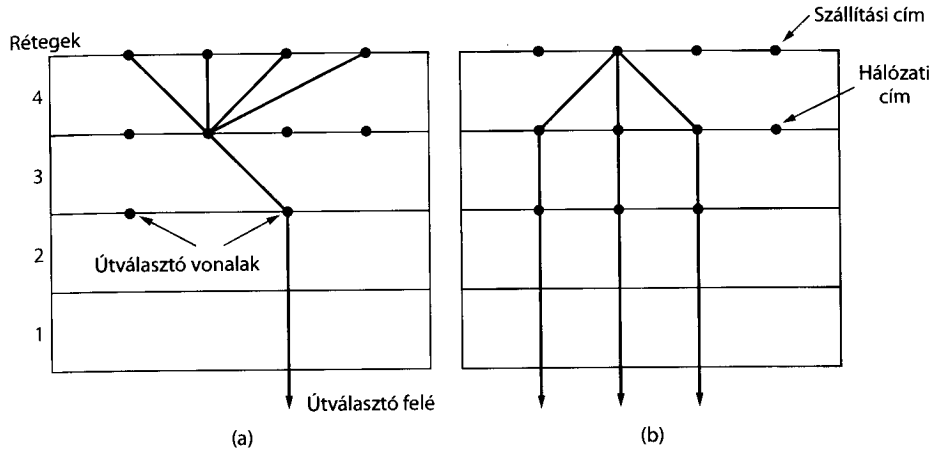
Amikor a pufferterület többé nem korlátozza a maximális forgalmat, egy újabb szűk keresztmetszet bukkan föl: a hálózat szállítókapacitása. Ha szomszédos útválasztók legfeljebb x keret/s sebességre képesek, és k független út van két hoszt között, semmilyen módon nem valósíthat meg a szóban forgó két hoszt kx szegmens/s-nál nagyobb sebességű átvitelt, akármennyi szabad puffere van is a két félnek. Ha a küldő túl sűrűn ad (tehát kx-nél több szegmenst küld másodpercenként), a hálózatban torlódás keletkezik, mert nem képes olyan gyorsan továbbítani a szegmenseket, mint amilyen gyorsan kapja azokat.

Valójában olyan, a küldő sebességét szabályozó mechanizmusra van szükség, amely a hálózat szállítókapacitására, és nem a vevő pufferfoglalási lehetőségeire épít. Belsnes [1975] egy olyan csúszóablakos forgalomszabályozási módszert javasolt, melyben a küldő dinamikusan a hálózat szállítási kapacitásához igazítja az ablak méretét. Ez azt jelenti, hogy a dinamikus ablakállítás egyszerre valósítja meg a forgalomszabályozást és a torlódáskezelést. Ha a hálózat másodpercenként c szegmens átvitelére képes, és a körülfordulási idő r (ami magában foglalja a küldési, átviteli és vevő várakozási soraiban eltöltött időt, a vevő által végzett feldolgozás idejét és a nyugta visszaérkezésének idejét), a küldő ablakmérete célszerűen cr kell legyen. Ekkora ablakmérettel normális állapotban a küldő folyamatosan tele csővel dolgozik, így a hálózati teljesítőképesség legkisebb csökkenése is a küldő blokkolását okozza. Mivel a hálózati kapacitás bármely folyamra időben változó, az ablakméretet sűrűn kell állítani, hogy a szállítási kapacitást nyomon tudja követni. Amint látni fogjuk, a TCP is hasonló megoldást alkalmaz.

6.2.5. Nyalábolás

Több beszélgetés összeköttetésekre, virtuális áramkörökre és fizikai összeköttetésekre való nyalábolása, multiplexelése a hálózati architektúra számos rétegében játszik szerepet. A szállítási rétegben a multiplexelés igénye többféle okból is felmerülhet. Ha például egy hoszton csak egyetlen hálózati cím áll rendelkezésre, akkor azon a gépen minden szállítási összeköttetésnek azt kell használnia. Szükség van valamilyen módszerre, amellyel eldönthetjük, hogy a beérkező szegmenseket melyik folyamatnak kell továbbítani. Ezt a feladatot **nyalábolásnak** vagy **multiplexelésnek** (**multiplexing**) hívják és a 6.17.(a) ábra mutatja be. Az ábrán négy különböző szállítási összeköttetés használja ugyanazt a hálózati összeköttetést (vagyis IP-címet) a távoli hoszt eléréséhez.

A nyalábolás a szállítási rétegben egy másik ok miatt is hasznos lehet. Tegyük fel például, hogy egy hálózaton belül egy hoszt több útvonalat is használhat. Ha egy felhasználónak nagyobb sávszélességre van szüksége, mint amennyit az egyik útvonal nyújtani tud, akkor egy olyan hálózati összeköttetést kell megnyitnia, mely a forgalmat körforgásos alapon elosztja több útvonal között, ahogy az a 6.17.(b) ábrán is látható. Ezt a működési módot **fordított nyalábolásnak vagy fordított multiplexelésnek** (**inverse multiplexing**) nevezik. Ha k hálózati összeköttetést nyitunk meg, akkor a rendelkezésre álló sávszélesség k-szorosára növekedhet. A fordított nyalábolás egy példája az SCTP (**Stream Control Transmission Protocol – folyamvezérlő átviteli protokoll**), amely



6.17. ábra. *(a) Nyalábolás. (b) Fordított nyalábolás*

képes több hálózati interfészt használni egyetlen összeköttetésen belül. Ezzel ellentétben a TCP csak egy hálózati végpontot használ. A fordított nyalábolás megtalálható az adatkapcsolati rétegben is, ahol több kis sebességű összeköttetést párhuzamosan használva egy nagy sebességű összeköttetést alakítanak ki.

6.2.6. Összeomlás utáni helyreállítás

Ha az útválasztók és hosztok összeomlás veszélyének vannak kitéve, vagy ha az összeköttetések hosszú életűek (például méretes szoftver- vagy médialetöltés esetén), fontos kérdéssé válik a feléledés megvalósítása. Ha a teljes szállítási entitás a hoszton belül van megvalósítva, a hálózat és az útválasztók összeomlás utáni helyreállítása nyilvánvaló. A szállítási entitások mindig számolnak elveszett szegmensekkel, és tudják, hogyan kezeljék azokat.

Ennél sokkal komolyabb probléma a hosztok összeomlás utáni újraindítása. Általános cél lehet, hogy a kliensek a szerver összeomlása és gyors föléledése után tovább tudják folytatni munkájukat. A nehézség illusztrálására tegyük föl, hogy az egyik hoszt, a kliens, egy nagy állományt küld a másik hoszt, az állományszolgáltató számára egyszerű megáll-és-vár (stop-and-wait) protokollal. A szerver szállítási rétege egyszerűen egyenként átadja a beérkező szegmenseket a szállítási felhasználónak. Az átvitel közepén a szerver összeomlik. Amikor újraéled, az összes táblázatát újrainicializálja, így nem tudja, hogy hol tartott.

Előző állapotának kiderítése érdekében a szerver minden kliens részére elküldhet egy szegmenst, melyben bejelenti, hogy tönkrement, és mindenkit kér, hogy küldjék el használt összeköttetéseik állapotát. Minden kliens az alábbi két állapot egyikében lehet: egy elküldött szegmens van függőben (*S1*), vagy nincs ilyen szegmens (*S0*). Csupán ezen információ birtokában el kell döntenie a kliensnek, hogy újraküldje-e a legutolsó szegmenst, vagy sem.

Első pillantásra nyilvánvalónak tűnhet, hogy a kliensnek, amikor megtudja, hogy a szerver összeomlott, csak akkor kell újraküldenie a kérdéses szegmenst, ha az nyugtázatlan (azaz ha *S1* állapotban van). Közelebbi vizsgálatok során azonban kiderül ennek a naiv megközelítésnek a veszélye. Vegyük például azt a helyzetet, amikor a szerver szállítási entitása elküldi a nyugtát, és miután a nyugta elindult, csak ezután adja át a szegmenst az alkalmazói folyamatnak. A szegmens kimenő folyamba írása és a nyugta elküldése két, külön-külön oszthatatlan esemény, melyeket nem lehet egyszerre végrehajtani. Ha az összeomlás a nyugta elküldése után, de még a szegmens átadása előtt történik meg, a kliens megkapja a nyugtát, és így *S0* állapotban lesz, amikor az újraindulás bejelentése megérkezik. A kliens ekkor nem fogja újraküldeni a szegmenst, mivel (helytelenül) abban a tudatban él, hogy az megérkezett. Döntése egy hiányzó szegmenst eredményez.

Ezen a ponton azt gondolhatjuk: „Ez a probléma könnyen megoldható. Csak annyit kell tenni, hogy átírjuk a szállítási entitást, és ezentúl először a folyamba ír, és csak azután küldi a nyugtát." Játsszuk el a fönti kísérletet ismét! Képzeljük el, hogy a folyamba írás már megtörtént, de az összeomlás éppen a nyugta elküldése előtt következik be. A kliens így *S1* állapotban lesz, és újraküldi a szegmenst, ami így észrevétlenül egy második szegmensként a szerveroldali alkalmazási folyamat irányába menő kimeneti adatáramba kerül.

Mindegy, hogyan programozzuk a szervert és a klienst, mindig vannak olyan helyzetek, melyekben ez a protokoll kudarcot vall. A szerver kétféleképpen valósítható meg: először nyugtáz vagy először ír. A kliens négyféleképpen programozható: mindig újraküldi az utolsó szegmenst, sohasem küldi újra, csak az *S0* állapotban küldi újra, vagy csak az *S1*-ben. Ez nyolc kombinációt jelent, de mint látni fogjuk, ezek közül mindegyikhez van egy eseményhalmaz, aminek hatására a protokoll hibázik.

A szerver oldalán három esemény lehetséges: nyugta küldése (*Ny*), kimeneti folyamba írás (*K*) és összeomlás (*Ö*). A három esemény hat különböző sorrendben történhet: *NyÖ(K)*, *NyKÖ*, *Ö(NyK)*, *Ö(KNy)*, *KNyÖ* és *KÖ(Ny)*, ahol a zárójelek azt jelölik, hogy az összeomlást már sem nyugtázás, sem írás nem követi (azaz ha a rendszer összeomlott, akkor tényleg összeomlott). A 6.18. ábrán a kliens- és szerverstratégiák nyolc kombinációját mutatjuk be az érvényes eseménysorrendekkel együtt. Vegyük észre, hogy minden stratégiára található valamilyen eseménysorrend, amire a protokoll kudarcot vall. Például, ha a kliens újraküld, az *NyKÖ* eseménysorozat észrevétlen kettőzést eredményez, miközben a másik két sorozatra helyesen működik a protokoll.

A protokoll további finomítása sem segít. Még ha a szerver és a kliens – mielőtt a szerver írni próbálna – több szegmenst váltana is egymással, hogy a kliens pontosan tudja, mi fog történni, arról semmiképpen sem szerezhet tudomást, hogy az összeomlás közvetlenül az írás előtt vagy közvetlenül utána következett-e be. A következtetés elkerülhetetlen: alapszabályunk – mely szerint az események nem történhetnek párhuzamosan – kizárja annak lehetőségét, hogy a rendszerösszeomlás és újraindulás a felső rétegek számára transzparens módon mehessen végbe.

Általánosabban szólva, ez az eredmény azt jelenti, hogy az *N*. réteg összeomlásából történő újraindulás csak az *N* + 1. réteg segítségével lehetséges, feltéve, hogy a magasabb réteg elegendő információt tárol az állapotáról, hogy helyre tudja állítani azt az összeomlás előtti állapotra. Ez egybevág azzal, amit fönt említettünk, vagyis a szállítási

	A vevőhoszt (szerver) stratégiája					
	Először nyugta, azután kiírás			Először kiírás, azután nyugta		
Az adó hoszt (kliens stratégiája	NyÖ(K)	NyKÖ	Ö(NyK)	Ö(KNy)	KNyÖ	KÖ(Ny)
Mindig újraküld	OK	KETTŐZ	OK	OK	KETTŐZ	KETTŐZ
Soha nem ad újra	VESZT	OK	VESZT	VESZT	OK	OK
S0-ban újraküld	OK	KETTŐZ	VESZT	VESZT	KETTŐZ	OK
S1-ben újraküld	VESZT	OK	OK	OK	OK	KETTŐZ

OK = A protokoll helyesen működik
KETTŐZ = A protokoll üzenetet kettőz
VESZT = A protokoll üzenetet veszít

6.18. ábra. *A kliens- és szerverstratégia különböző kombinációi*

réteg akkor képes kezelni a hálózati rétegben történt összeomlásokat, ha az összeköttetés mindkét vége állandóan nyomon követi, hogy hol tartanak.

Ez a probléma végül elvezet ahhoz a kérdéshez, hogy mit is jelent valójában az ún. végpontok közötti nyugtázás. Alapjában véve a szállítási protokoll két végpont között működik, és nem láncolt, mint az alatta levő rétegek. Most vegyük azt az esetet, amikor a felhasználó a távoli adatbázisban tranzakciókat kezdeményez. Tegyük föl, hogy a távoli szállítási entitás programja szerint először átadja a szegmenst a fölötte levő rétegnek, és ezután nyugtáz. Még ebben az esetben sem jelenti a nyugta vétele a felhasználó gépén, hogy a távoli hoszt addig működőképes maradt, amíg a tranzakció lezárása meg nem történt. Egy igazi végpontok közötti nyugtát – melynek vétele azt jelenti, hogy a tevékenység ténylegesen végbement, és hiánya annak elmaradását jelzi – valószínűleg lehetetlen elérni. A kérdésről bővebben Saltzer és munkatársai [1984] művében olvashatunk.

6.3. Torlódáskezelés

Ha sok gép szállítási entitása túl sok csomagot és túl gyorsan küld a hálózatba, a hálózatban torlódás keletkezik, és a teljesítőképessége leromlik, ahogy a csomagok egyre nagyobb késleltetést szenvednek és elvesznek. A torlódás szabályozása a hálózati és a szállítási réteg közös felelőssége, hogy elkerüljük az ilyen jellegű problémákat. A torlódás az útválasztókban történik, így a hálózati réteg fedezi fel, azonban a torlódást a szállítási réteg által a hálózatba küldött forgalom okozza. A torlódás szabályozására az egyetlen hatékony módszer a szállítási protokollok számára a hálózatba küldött forgalom intenzitásának csökkentése.

Az 5. fejezetben már esett szó torlódáskezelő algoritmusokról a hálózati rétegben. Ebben a szakaszban a probléma másik felét fogjuk tanulmányozni, a torlódáskezelést a szállítási rétegben. Miután átnéztük a torlódáskezelés céljait, áttekintjük, hogy a hosztok hogyan tudják szabályozni a csomagok hálózatba küldésének a sebességét. Az internet

erőteljesen számít a szállítási réteg torlódáskezelésére, és ezért speciális algoritmusokat építettek a TCP-be és más protokollokba.

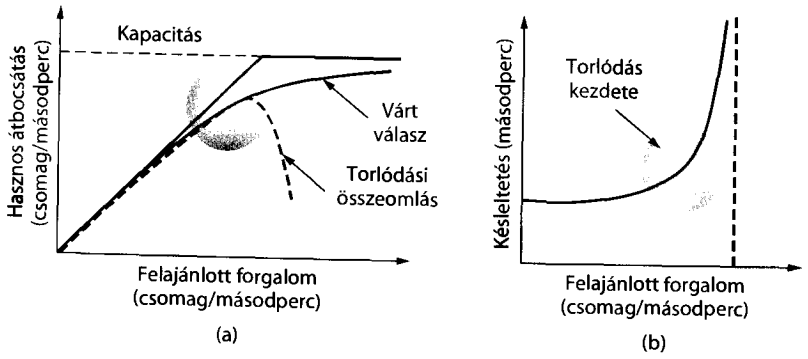
6.3.1. A szükséges sávszélesség lefoglalása

Mielőtt megnéznénk, hogyan lehet a forgalmat szabályozni, meg kell értenünk, mit is szeretnénk elérni a torlódáskezelő algoritmusok futtatásával. Meg kell tehát határoznunk azt az állapotot, amelyben egy jó torlódáskezelő algoritmus a hálózatot tartani fogja. A célunk több annál, hogy egyszerűen elkerüljük a torlódást. Ehelyett a sávszélesség megfelelő elosztása a célunk a hálózatot használó szállítási entitások között. A jó elosztás jó teljesítőképességgel jár, mert torlódásmentesen kihasználja a teljes sávszélességet, igazságosan elosztja azt a versengő szállítási entitások közt, valamint gyorsan követi a forgalmi igényeket. Ezeket a követelményeket részleteikben is áttekintjük.

Hatékonyság és teljesítőképesség

A sávszélesség hatékony elosztása a szállítási entitások között a teljes rendelkezésre álló hálózati kapacitást kihasználja. Ne gondoljuk azonban azt, hogy ha létezik egy 100 Mbit/s sebességű összeköttetésünk, akkor 5 szállítási entitás egyenként 20-20 Mbit/s sebességű darabot kap. A megfelelő teljesítőképesség érdekében általában ennél kevesebb jut rájuk. A magyarázat abban rejlik, hogy a forgalom gyakran löketes. Emlékezzünk vissza, hogy az 5.3. szakaszban bemutattuk a **hasznos átbocsátóképességet** (**goodput**) mint a felajánlott forgalom (felajánlott terhelés – offered load) függvényét. Ez és a hozzá hasonló, a késleltetést a felajánlott forgalom függvényében mutató görbék láthatók a 6.19. ábrán.

Ahogy a 6.19.(a) ábrán megfigyelhetjük, a terhelés emelkedésével kezdetben a hasznos átbocsátás hasonló ütemben nő, de ahogy a terhelés közelíti a kapacitást, a hasznos átbocsátás egyre lassabban növekszik. Mindez azért történik, mert a forgalom löketei néha csomagvesztést okoznak a hálózatban található pufferekben. Ha a szállítási protokollt rosszul tervezték, és olyan csomagokat is megismétel, amelyek még nem vesztek



6.19. ábra. *(a) Hasznos átbocsátás és (b) késleltetés a felajánlott forgalom függvényében*

el, csak késlekednek, akkor a hálózat a torlódás miatt összeomlik. Ebben az állapotban a küldők vadul küldözgetik a csomagjaikat, de egyre kevesebb hasznos adat jut át.

A felajánlott forgalomnak megfelelő késleltetést látjuk a 6.19.(b) ábrán. Kezdetben a késleltetés a hálózat terjedési késleltetésnek megfelelő állandó. Ahogy a terhelés közelíti a kapacitást, a késleltetés kezdetben lassan, majd rohamosan megnő. Ezt ismét a forgalmi löketek okozzák, amelyek nagy terhelésnél lekötik a hálózatot. A késleltetés a valóságban nem lehet végtelen, csupán a modellünkben az, ahol a pufferek végtelen méretűek. A valóságban a csomagok elvesznek, miután elérték a maximális pufferelési késleltetést.

Mind a hasznos átbocsátás, mind a késleltetés esetén a teljesítőképesség a torlódás kezdőpontjánál esik vissza. Intuitív megközelítésben azt mondhatjuk, hogy a legjobb teljesítőképességet akkor kapjuk, ha a sávszélességet a késleltetés növekedéséig foglaljuk le. Ez a pont a teljes kapacitás alatt van. Ennek megtalálásához Kleinrock [1979] bevezette a **teljesítmény** (**power**) mértékének fogalmát:

$$\text{Teljesítmény} = \frac{\text{Terhelés}}{\text{Késleltetés}}$$

A teljesítmény kezdetben növekedni fog a felajánlott forgalommal, mivel a késleltetés kicsi és közel állandó marad, majd eléri a maximumát, és a késleltetés hirtelen növekedésével erőteljes csökkenésnek indul. A legnagyobb teljesítménynél mért terhelés megmutat a szállítási entitásnak egy olyan hatékony terhelést, amelyet annak a hálózatra kell tenni.

Maximum-minimum igazságosság

Az előzőekben nem tettünk említést arról, hogy is osszuk fel a rendelkezésre álló sávszélességet a különböző küldők között. Könnyen megválaszolható kérdésnek tűnik – minden küldőnek egyenlő részeket hasítsunk a sávszélességből –, de azért néhány dolog megfontolást érdemel.

Az első talán az, hogy mi köze ennek a problémának a torlódáskezeléshez? Végül is, ha a küldő a hálózattól kap valamekkora sávszélességet, akkor a küldő akkora sávszélességet fog használni. Leggyakrabban azonban a hálózatoknak nincs a sávszélességet összeköttetésenként és folyamonként mereven felosztó rendszere. Ha a hálózat támogatja a szolgáltatásminőséget, akkor a folyamokra elképzelhető ilyen mechanizmus, de sok összeköttetés akkora sávszélességet fog használni, amekkora elérhető, vagy a hálózat egy közös felosztás alá eső csoportba sorolja azokat. Például az IETF differenciált szolgáltatásai a forgalmat két osztályra bontják, és mindkettőben az összeköttetések egymással versengenek a sávszélességért. Az IP-útválasztók gyakran az összes összeköttetést ugyanazért a sávszélességért versenyeztetik. Ebben az esetben éppen a torlódáskezelés osztja ki a sávszélességet a versenyző összeköttetéseknek.

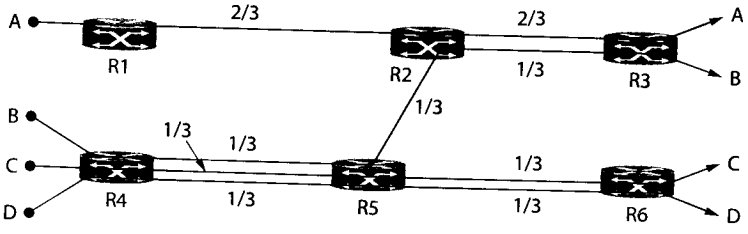
Második megfontolásunk arról szól, hogy mit jelent az igazságos felosztás a hálózati folyamok számára. Nyilvánvaló, hogy ha N folyam használ egyetlen adatkapcsolatot, akkor mindegyik folyam az adatkapcsolat sávszélességének az $1/N$-szeresét kaphatja (habár a hatékonyság azt sugallja, hogy löketes forgalom esetén ennél valamivel keve-

sebbet). De mi történik akkor, ha a folyamok különböző, de átlapolódó hálózati útvonalakat használnak? Például az egyik folyam három adatkapcsolatot használ, a többiek pedig egyet. A három adatkapcsolatot használó folyam több hálózati erőforrást használ. Bizonyos szempontból igazságosabb, ha kevesebb sávszélességet kap, mint az egyetlen adatkapcsolatot használók. Nyilvánvalóan a három adatkapcsolatot használó folyam sávszélességének csökkentésével lehetőség nyílik több, egyetlen adatkapcsolatot használó folyam kiszolgálására.

Mi az igazságosság egy olyan formáját alkalmazzuk, amely nem függ a hálózati útvonalak hosszától. Még ebben az egyszerű modellben is problémás egyenlő sávszélességszeleteket kiosztani az összeköttetéseknek, hiszen a különböző összeköttetések különböző útvonalakat használnak, és ezen útvonalak kapacitása önmagukban is különbözik. Ebben az esetben elképzelhető, hogy egy folyam szűk keresztmetszetbe kerülhet egy lefele irányú kapcsolaton, és kisebb részt kap egy felfele irányú kapcsolaton, mint más folyamok; a többi folyam sávszélességének csökkentése lelassítaná azokat, de a bajba jutott folyamon semmit nem segítene.

Az igazságosság azon formáját, amely a hálózati alkalmazásokban leggyakrabban előfordul, **maximum-minimum igazságosságnak** (**max-min fairness**) hívják. Egy kiosztás maximum-minimum igazságos, ha egy folyamnak kiosztott sávszélesség nem növelhető anélkül, hogy egy másik, kisebb vagy azonos kiosztott sávszélességű folyam sávszélességét ne csökkentenénk. Tehát egy folyam sávszélességének a növelése még rosszabbá teszi a helyzetet azon folyamok számára, amelyek kevésbé jómódúak.

Lássunk egy példát! A maximum-minimum igazságos kiosztást egy négy (*A*, *B*, *C* és *D*) folyammal rendelkező hálózatra mutatjuk be, amely a 6.20. ábrán látható. Minden, az útválasztók között húzódó adatkapcsolat azonos, 1 egységnyi kapacitású, habár egy általánosabb esetben az adatkapcsolatok kapacitása különböző. Az alsó sorban a bal oldali *R*4 és *R*5 útválasztók közötti adatkapcsolatért három folyam verseng, így tehát mindhárom folyam megkapja az adatkapcsolat 1/3-ad részét. A megmaradt *A* folyam a *B* folyammal harcol az *R*2 és *R*3 közötti adatkapcsolatért. Mivel *B* foglalása 1/3, ezért *A* megkapja a maradék 2/3 részt. Vegyük észre, hogy a többi adatkapcsolatnak kihasználatlan kapacitása van. Mindazonáltal ez a kapacitás nem adható oda egyik folyamnak sem anélkül, hogy valamely másik, kisebb folyam kapacitását ne csökkentenénk. Például ha az *R*2 és *R*3 közötti adatkapcsolat sávszélességéből többet adnánk *B*-nek, akkor kevesebb maradna *A*-nak. Ez méltányos lenne, hiszen *A* nagyobb sávszélességű, mint *B*. Azonban *B* sávszélességének növeléséhez *C* vagy *D* (vagy mindkettő) sávszélességét



6.20. ábra. *Maximum-minimum sávszélesség-kiosztás négy folyamra*

csökkenteni kellene, de ezeknek a folyamoknak így kevesebb sávszélesség jutna, mint *B*-nek. A kiosztás tehát maximum-minimum igazságos.

A teljes hálózat ismeretében a maximum-minimum igazságos kiosztás meghatározható. A legegyszerűbb módja az, hogy a folyamok sávszélességét kezdetben nullának vesszük, majd lassan növeljük. Amikor egy folyam szűk keresztmetszetbe kerül, akkor azt nem növeljük tovább. A többi folyamot továbbra is folyamatosan növeljük, hogy a rendelkezésre álló kapacitást egyenletesen kitöltsék egészen addig, amíg azok is el nem érik a saját szűk keresztmetszetüket.

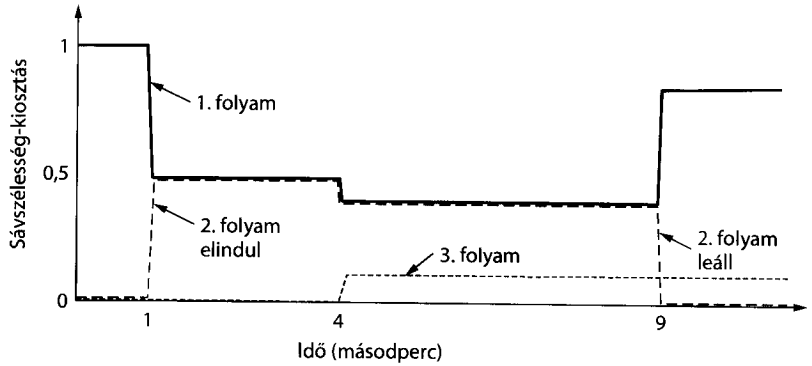
A harmadik megfontolásunk az igazságosság szintjét érinti. A hálózat az összeköttetések szintjén lehet igazságos, azaz hosztpárok közti összeköttetésekre vagy hosztonkénti összeköttetésekre. Ezt a kérdést már megvizsgáltuk az 5.4. szakaszban a **WFQ**-val (**Weighted Fair Queueing – súlyozott egyenlő esélyű sorba állítás**) kapcsolatban, és megállapítottuk, hogy mindegyik definíciónak megvan a maga problémája. Például ha az igazságosságot hosztonként definiáljuk, akkor egy leterhelt szerver semmivel sem jár jobban, mint egy mobiltelefon, míg az igazságosság összeköttetésenkénti definíciója a hosztokat több összeköttetés nyitására ösztönzi. Mivel egyértelmű válasz nincs, ezért leggyakrabban az igazságosságot összeköttetésenként értelmezik, azonban a precíz definíció nem is túl fontos. A gyakorlatban sokkal fontosabb, hogy egyetlen összeköttetés se szenvedjen annyi hiányt sávszélességben, mintha minden összeköttetés pontosan ugyanakkora mennyiségű sávszélességet kapna. Tény, hogy a TCP esetén bármikor több összeköttetés nyitható, hogy a sávszélességért keményebb harc alakuljon ki. Ezt a taktikát olyan sávszélesség-igényes alkalmazások használják, mint például a P2P-fájlmegosztásban jeleskedő BitTorrent.

Konvergencia

Az utolsó kritérium, amit egy torlódáskezelő algoritmusnak teljesíteni kell az, hogy gyorsan tartson (konvergáljon) egy igazságos és hatékony sávszélesség-kiosztás felé. Az eddigiekben tárgyalt elvárt működés egy statikus hálózatot feltételezett. Időről időre azonban új összeköttetések bukkannak fel a hálózatban, és az összeköttetések által igényelt sávszélesség is változó, például azért, mert a felhasználó egyszer a webet böngészi, máskor pedig nagy videofájlokat tölt le.

Az igények változása miatt a hálózat ideális munkapontja is változik az idővel. Egy jó torlódáskezelő algoritmusnak gyorsan meg kell találnia az ideális munkapontot, és követnie kell azt, ha változik. Ha a konvergencia lassú, akkor az algoritmus sosem lesz a folyton változó munkapont közelében. Ha az algoritmus nem stabil, a helyes munkapont elérése esetleg sikertelen vagy akár még oszcillálhat is körülötte.

Az időben változó, ám gyorsan konvergáló sávszélesség-kiosztásra a 6.21. ábrán látható példa. Kezdetben az 1. folyam birtokolja a teljes sávszélességet. Egy másodperccel később a 2. folyam is felbukkan, melynek ugyancsak sávszélességre van szüksége. A kiosztás gyorsan megváltozik, és mindkét folyam a sávszélesség felét kapja. A 4. másodpercben egy harmadik folyam kapcsolódik be a történetbe. Ez a folyam azonban a számára igazságosnak mondható sávszélesség helyett (ami a teljes harmada lenne) csupán a teljes sávszélesség 20 százalékát használja. Az 1. és 2. folyam gyorsan alkalmazkodik a



6.21. ábra. *Időben változó sávszélesség-kiosztás*

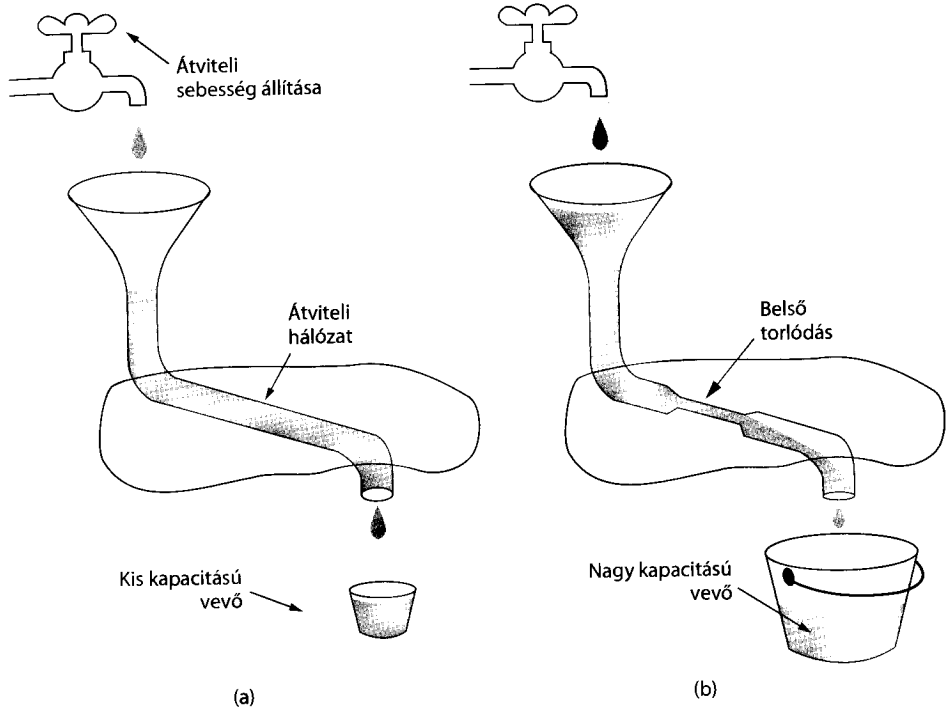
helyzethez, és mindketten a sávszélesség 40-40 százalékát kapják. A 2. folyam a 9. másodpercben eltűnik, azonban a harmadik folyam változatlan marad, így az első folyam hirtelen a sávszélesség 80 százalékát elveszi. Akármelyik időpillanatot is nézzük, a teljes lefoglalt sávszélesség megközelítőleg 100 százalék, így a hálózat teljes kihasználtsággal üzemel, és a versengő folyamok egyenlő bánásmódot kapnak (de nem kell több sávszélességet használniuk, mint amennyire szükségük van).

6.3.2. A küldési sebesség szabályozása

Itt az ideje a főfogás tálalásának: hogyan szabályozzuk a küldési sebességet, hogy az elvárt sávszélesség-kiosztást kapjuk? A küldési sebességet két tényező korlátozhatja. Az első a forgalomszabályozás, amikor is nincs elég pufferkapacitás a vevőben. A második a torlódás, amikor pedig a hálózat kapacitása elégtelen. A 6.22. ábrán ezt a problémát egy hidrosztatikai példán keresztül mutatjuk be. A 6.22.(a). ábrán egy vastag csövet láthatunk, amely egy kis kapacitású vevőhöz vezet. Ebben az esetben a forgalomszabályozás korlátozza a sebességet. Mindaddig, amíg a küldő nem enged több vizet, mint amennyit a vödör tárolni képes, a víz nem vész kárba. A 6.22.(b). ábrán a korlátozó tényező nem a vödör mérete, hanem a hálózat áteresztő-képessége. Ha túl sok vizet engedünk a csőbe, akkor az felgyülemlik és valamennyi kárba vész (ez esetben a tölcsér peremén túlfolyik).

A küldő számára ezek az esetek nagyon hasonlók lehetnek, hiszen ha a csomagokat gyorsan küldi, akkor azok elvesznek. Ezeknek az eseteknek az okai azonban különbözők lehetnek, és más-más megoldást kívánnak. Egy forgalomszabályozási megoldásról már beszéltünk, amely változó méretű ablakokat használ. Most egy torlódáskezelő algoritmust fogunk megismerni. Mivel mindkét előbb említett probléma előfordulhat, ezért a szállítási protokollnak általánosságban szüksége lesz mindkét megoldásra, és lassítania kell, ha valamelyik probléma jelentkezik.

A hálózat által biztosított visszacsatolás módja határozza meg, hogy a szállítási protokollnak miként kell a küldési sebességet szabályozni. A különböző hálózati rétegek különféle visszacsatolást biztosíthatnak. Ez lehet explicit vagy implicit, és pontos vagy pontatlan.



6.22. ábra. *(a) Gyors hálózat és kis kapacitású vevő esete. (b) Lassú hálózat és nagy kapacitású vevő esete*

Pontos, explicit visszacsatolásra egy példa, amikor az útválasztók közlik a forrásokkal, hogy mekkora sebességgel küldhetnek csomagokat. Az irodalomban található megoldások, mint például az XCP (eXplicit Congestion Protocol – explicit torlódáskezelő protokoll) is hasonló elven működik [Katabi és mások, 2002]. Egy explicit, pontatlan felépítésű az ECN (Explicit Congestion Notification – explicit torlódásjelzés) használata TCP-vel. Ebben az esetben a torlódást tapasztaló útválasztók a csomagokban található speciális bitek segítségével jelzik a forrásoknak, hogy lassítsanak, de a lassítás mértékét nem közlik.

Más megoldásokban nincs explicit jelzés. A FAST TCP a körülfordulási idő mérésével próbálja elkerülni a torlódást [Wei és mások, 2006]. Végül a legelterjedtebb, az interneten manapság használt torlódáskezelés, a TCP együtt használva drop-tail vagy RED-útválasztókkal, amely a csomagvesztésből von le következtetést, és ez alapján továbbít jelzést arról, hogy a hálózatban torlódás van. A TCP ilyen változataiból sok létezik, beleértve a CUBIC TCP-t, amelyet a Linux használ [Ha és mások, 2008]. Az előbbiek különböző kombinációi is lehetségesek, például a Windows a Compound TCP-t használja, mely mind a csomagvesztést, mind a késleltetést visszacsatolásként alkalmazza [Tan és mások, 2006]. Az itt felsorolt változatokat a 6.23. ábrán foglaltuk össze.

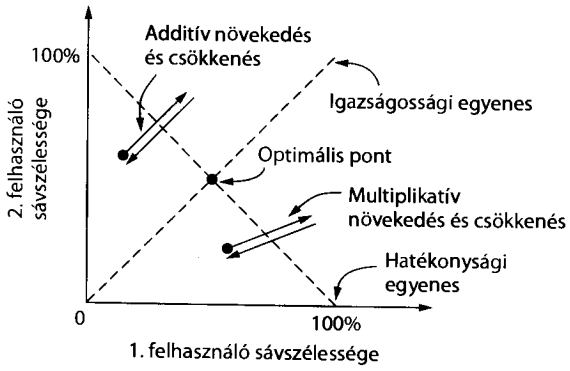
Ha a szállítási entitás határozott és pontos jelzést kap, használhatja ezt a jelzést a saját sebességének egy új munkapontra történő beállítására. Például, ha az XCP közli a kül-

Protokoll	Jelzés	Explicit?	Pontos?
XCP	Az előírt sebesség	Igen	Igen
TCP ECN-nel	Figyelmeztetés torlódásra	Igen	Nem
FAST TCP	Végpontok közötti késleltetés	Nem	Igen
Compound TCP	Csomagvesztés és végpontok közötti késleltetés	Nem	Igen
CUBIC TCP	Csomagvesztés	Nem	Nem
TCP	Csomagvesztés	Nem	Nem

6.23. ábra. *Néhány torlódásszabályozó protokoll jelzései*

dővel a használandó sebességet, akkor az egyszerűen használhatja azt a sebességet. Más esetekben azonban néha becslést kell alkalmazni. Torlódásjelzés hiányában a küldőknek növelni kell a sebességüket. Torlódási jelzés jelenléte esetén azonban a sebességet csökkenteni kell. A sebesség növelésének vagy csökkentésének a módját egy **szabályozási törvény** (**control law**) írja elő. Ezek a törvények nagy hatással vannak a teljesítőképességre.

Chiu és Jain [1989] a bináris torlódási visszacsatolás tanulmányozása közben megállapították, hogy a hatékony és igazságos munkapont eléréséhez az **AIMD** (**Additive Increase Multiplicative Decrease – additív növekedés, multiplikatív csökkenés**) szabályozási törvény a megfelelő. A törvény alátámasztására grafikus bizonyítást készítettek egy egyszerű esetre, melyben egy összeköttetésen két összeköttetés verseng a sávszélességért. A 6.24. ábra mutatja az 1. felhasználó által lefoglalt sávszélességet az x tengelyen, a 2. felhasználó által lefoglalt sávszélességet pedig az y tengelyen. Amikor a kiosztás igazságos, a két felhasználó azonos mennyiségű sávszélességet kap, amit az ábrán az igazságossági egyenes mutat. Amikor a kiosztott sávszélességek összege – tehát az összeköttetés kapacitása – 100%, akkor a kiosztás hatékony. Ezt mutatja a hatékonysági egyenes. Mindkét felhasználó torlódási jelzést kap a hálózattól abban az esetben, ha a kiosztott sávszélességeik összege átlépi ezt az egyenest. A két vonal metszéspontja mutatja



6.24. ábra. *Additív és multiplikatív sávszélesség-szabályozás*

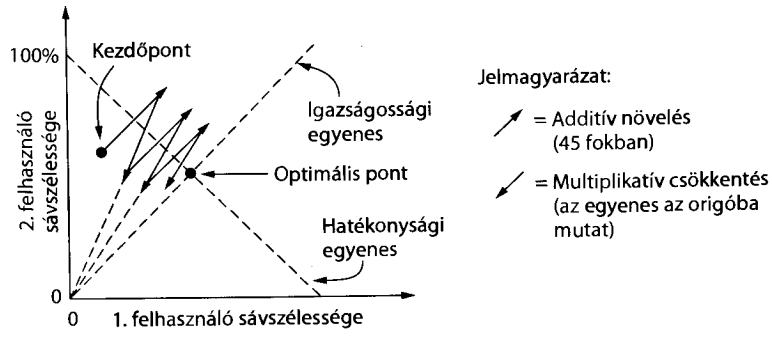
azt a kívánatos munkapontot, amelyen mindkét felhasználónak azonos sávszélesség jut, és a teljes hálózati sávszélességet kihasználják.

Tekintsük meg, hogy mi történik akkor, ha egy kezdeti kiosztásból indulva mind az 1. felhasználó, mind a 2. felhasználó additívan növelni kezdik a saját sávszélesség-foglalásukat. Például másodpercenként mindkét felhasználó 1 Mbit/s sebességgel növeli az adási sebességét. Idővel a munkapont átlépi a hatékonysági egyenest, és mindketten egy-egy torlódási jelzést kapnak a hálózattól. Ezen a ponton csökkenteni kell a sávszélesség-foglalásukat. Egy additív csökkentés azonban csak annyit érne el, hogy elkezdenének ugrálni egy additív egyenes körül. Ezt a helyzetet mutatja 6.24. ábra. A munkapont közel hatékony lenne, de nem feltétlenül lenne igazságos.

Ehhez hasonlóan, vegyük azt az esetet, amikor mindkét felhasználó multiplikatívan növeli a sávszélességét, amíg egy torlódási jelzést nem kapnak. Például mindkét felhasználó másodpercenként 10%-kal növeli a küldési sebességet. Ha most multiplikatívan csökkenteni kezdik a küldési sebességet, akkor a munkapont egy multiplikatív egyenes körül fog ugrálni. A 6.24. ábra ezt a viselkedést is mutatja. A multiplikatív egyenes meredeksége eltér az additív egyenes meredekségétől. (A multiplikatív egyenes az origóba mutat, míg az additív egyenes 45 fokos szögben áll.) Ettől eltekintve ez a megoldás sem jobb. Egyik megoldás sem segíti a felhasználókat, hogy az optimális, tehát hatékony és igazságos adási sebességek felé konvergáljanak.

Tekintsük most azt az esetet, amikor a felhasználók additívan növelik és multiplikatívan csökkentik a sávszélesség-foglalásukat, ha torlódásjelzést kapnak. Ez a módszer az AIMD-szabályozás, és a 6.25. ábra szemlélteti. Jól látható, hogy a működése során bejárt útvonal az optimális ponthoz konvergál, amely igazságos és hatékony. A módszer bármely kezdőpont esetén konvergens, így az AIMD széleskörűen használható. Az eddigiek alapján belátható, hogy minden más kombináció, tehát a multiplikatív növelés és additív csökkenés a munkapont divergenciáját okozza.

A TCP is az AIMD szabályozási törvényt alkalmazza, és az ismertetett elv mellett további stabilitási megfontolásokat is tartalmaz (ugyanis a hálózatban könnyű torlódást létrehozni, és nehéz azt megszüntetni, így a növelési szabályt óvatosra, míg a csökkentési szabályt agresszívre kell állítani). Ez nem túl igazságos, mivel a TCP-összeköttetések az ablakméretet körülfordulási időnként növelik. A különböző összeköttetések különböző körülfordulási időkkel rendelkeznek, ami ahhoz vezet, hogy a közelebbi hosztokhoz



6.25. ábra. *Additív növelés és multiplikatív csökkentés (AIMD) szabályozási törvény*

tartozó összeköttetések akkor is több sávszélességet kapnak, mint a távoli hosztokhoz kiépített összeköttetések, ha minden másban megegyeznek.

A 6.5. szakaszban részletesen áttekintjük, hogy a TCP hogyan valósítja meg az AIMD szabályozási törvényt az adási sebesség állításához és a torlódáskezelés megvalósításához. Ez a feladat sokkal nehezebb, mint amilyennek elsőre hangzik, ugyanis a sebességet intervallumokban mérjük, míg a forgalom löketes. A sebesség közvetlen állítása helyett a gyakorlatban legtöbbször a csúszóablak méretét állítják, ahogy a TCP is ezt teszi. Ha az ablakméret W és a körülfordulási idő RTT, akkor az ennek megfelelő sebesség W/RTT. Ezt a stratégiát könnyű egyesíteni a forgalomszabályozással, amely már amúgy is használ egy ablakot. A megoldás további előnye, hogy a küldő a csomagokat nyugtánként rakja a hálózatra, és így egy RTT alatt lelassít, ha nem kap értesítést arról, hogy a csomagok elhagyják a hálózatot.

Végül érdemes megemlíteni, hogy a hálózaton több különböző szállítási protokoll is lebonyolíthat forgalmat. Mi történik akkor, ha több különböző protokoll, különböző szabályozási törvényekkel próbálja elkerülni a torlódást? Az történik, hogy egyenlőtlen sávszélesség-kiosztás jön létre. Mivel az interneten nagyrészt a TCP torlódáskezelő algoritmusait használják, jelentős közösségi nyomás éri az új szállítási protokollokat, hogy a TCP-vel szemben igazságosan lépjenek fel. A korai multimédia-folyamok protokolljai (streaming media protocols) rendkívül lecsökkentették a TCP áteresztőképességét annak következtében, hogy a versengés során nem bántak igazságosan a TCP-vel. Ez vezetett ahhoz a szándékhoz, hogy létrehozzák a **TCP-barát** (**TCP-friendly**) torlódáskezelési módszert, amelyben a TCP- és nem-TCP-protokollok szabadon vegyíthetők mindenféle káros mellékhatás nélkül [Floyd és mások, 2000].

6.3.3. Torlódáskezelés vezeték nélküli hálózatokban

A szállítási protokolloknak, amelyek torlódáskezelést végeznek, mint például a TCP, függetlennek kell lenniük az alattuk húzódó hálózati és adatkapcsolati réteg megoldásaitól. Ez nagyon szép elv, azonban a gyakorlatban a vezeték nélküli hálózatokkal problémák adódnak. A gond ott van, hogy a csomagvesztést gyakran a torlódás jelzésére használják, beleértve – ahogy láttuk – a TCP-t is. A vezeték nélküli hálózatok viszont rendszeresen elveszítenek csomagokat az átviteli hibákból adódóan.

Az AIMD szabályozási törvényt alkalmazva nagy áteresztőképességhez alacsony csomagvesztési arányt kell biztosítani. Padhye és mások [1998] elemzésekkel megmutatták, hogy az áteresztőképesség a csomagvesztési arány négyzetgyökének a reciprokával arányos. A gyakorlatban ez azt jelenti, hogy a csomagvesztési arány gyors TCP-összeköttetés esetén nagyon alacsony; 1% már közepesnek mondható, és ha eléri a 10%-ot, akkor az összeköttetés gyakorlatilag működésképtelen. Vezeték nélküli hálózatok, mint például a 802.11 LAN-ok esetében viszont átlagosan legalább 10% csomagvesztéssel kell számolni. Ez a különbség azt jelenti, hogy megelőző intézkedések hiányában, a csomagvesztést torlódásnak tekintő torlódáskezelési megoldások a vezeték nélküli adatkapcsolatokon futó összeköttetéseket feleslegesen lelassítják.

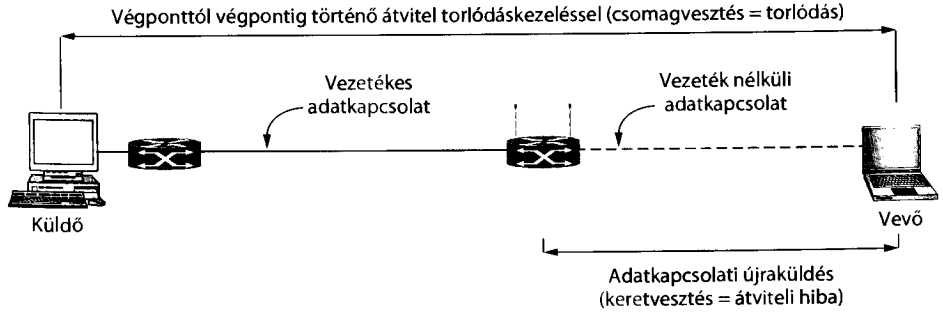
A helyes működés érdekében a torlódáskezelő algoritmusoknak csak olyan csomagvesztésekre kell odafigyelniük, amelyek az elégtelen sávszélesség miatt következtek be,

nem pedig átviteli hibák miatt. A probléma egyik megoldása lehet, ha a vezeték nélküli adatkapcsolaton történt csomagvesztéseket elfedjük (maszkoljuk) azzal, hogy újraküldjük a csomagokat az adatkapcsolaton. Például a 802.11 megáll-és-vár protokollt használ minden egyes keret továbbításához, és szükség esetén többször megismétli a keretet, mielőtt csomagvesztést jelentene a felsőbb rétegeknek. Normál esetben a csomagokat az ideiglenes átviteli hibák ellenére is kézbesíti az adatkapcsolat, és így a hibáról a felsőbb rétegek nem értesülnek.

A 6.26. ábra egy vezetékes és vezeték nélküli adatkapcsolatokból álló útvonalat mutat, melyen az álcázásos megoldást használjuk. Két megfontolást érdemes kiemelnünk. Egyrészt a küldő nem feltétlenül tudja, hogy egy vezeték nélküli adatkapcsolat is jelen van az útvonalon, ugyanis az csak azt a vezetékes összeköttetést látja, amelyhez csatlakoztatták. Az interneten az útvonalak sokfélék, és nincs általános módszer a küldő számára, hogy megállapítsa, milyen típusú adatkapcsolatok találhatók az útvonalon. Ez a tény bonyolítja a torlódáskezelést, ugyanis nem tehetjük meg azt, hogy használunk egy protokollt a vezetékes adatkapcsolatokra, egy másikat pedig a vezeték nélküli adatkapcsolatokra.

A második megfontolásunk egy rejtvény. Az ábra két, csomag- vagy keretvesztésre alapozott módszert mutat: adatkapcsolati rétegbeli keretújraküldést, valamint a szállítási rétegbeli torlódáskezelést. A rejtvény az, hogy hogyan tudnak ezek együtt működni anélkül, hogy összezavarodnának. Elvégre a csomagvesztésnek csupán az egyik mechanizmust kellene aktiválnia, ugyanis a csomagvesztés vagy átviteli hiba miatt történt, vagy torlódás miatt. Mindkettő nem lehet egyszerre. Ha mindkettő dolgozni kezd (és újraküldi a keretet, valamint csökkenti az adási sebességet), akkor megint ott vagyunk, ahol a part szakad: egy olyan szállítási protokollnál, amely túl lassú a vezeték nélküli adatkapcsolatokon. Egy pillanatra gondoljuk át a rejtvényt, és nézzük meg, vajon meg tudjuk-e oldani!

A megoldás a két mechanizmus által átfogott időskálában rejlik. Az adatkapcsolati rétegbeli újraküldés vezeték nélküli csatornán, mint például a 802.11 esetén, mikroszekundumtól milliszekundum nagyságrendbe esik. A szállítási rétegben a csomagvesztést figyelő számlálók milliszekundumtól másodperc nagyságrendben időzítenek. A különbség három nagyságrend. Ez lehetővé teszi a vezeték nélküli adatkapcsolatok számára, hogy észrevegyék a keretvesztést, és újraküldjék a keretet a hiba kijavítására jóval azelőtt, hogy a csomagvesztést a szállítási réteg észrevenné.



6.26. ábra. *Torlódáskezelés egy vezeték nélküli adatkapcsolatot tartalmazó útvonalon*

Az álcázásos stratégia lehetővé teszi, hogy a legtöbb szállítási protokoll jól fusson vezeték nélküli adatkapcsolatokon is. Mindazonáltal nem mindig ez a legjobb megoldás. Némely vezeték nélküli adatkapcsolat hosszú körülfordulási időket igényel, ilyenek például a műholdas adatkapcsolatok. Ilyen adatkapcsolatokon más megoldásokat kell használni a csomagvesztés álcázására, például FEC-et (Forward Error Correction – előre irányuló hibajavítás), vagy a szállítási protokollnak nem csomagvesztésen alapuló torlódáskezelést kell alkalmaznia.

A vezeték nélküli adatkapcsolatokon futó torlódáskezelés másik problémája az adatkapcsolat változó kapacitása, ugyanis a vezeték nélküli adatkapcsolatok kapacitása idővel változik, akár hirtelen is, ahogyan a csomópontok vándorolnak, és a jel/zaj viszony változik a csatorna jellemzőinek alakulásával.

Ennek az egyik lehetséges megoldása, hogy egyszerűen nem foglalkozunk vele. Ez a megoldás azért lehetséges, mert a torlódáskezelő algoritmusok már felkészültek arra, hogy egy új felhasználó jelenik meg a hálózatban, vagy egy már létező felhasználó megváltoztatja az adási sebességét. Még ha a vezetékes adatkapcsolatok kapacitása állandó is, a felhasználók változó viselkedése is a rendelkezésre álló sávszélesség folyamatos változásának tűnik egy adott felhasználó számára. Egy olyan útvonalon, amely 802.11 vezeték nélküli adatkapcsolatot is tartalmaz, lehetséges a TCP futtatása megfelelő teljesítmény mellett.

Ha azonban a vezeték nélküli adatkapcsolatok nagyon változó teljesítményt nyújtanak, egy vezetékes adatkapcsolatra tervezett szállítási protokoll nagyon gyenge teljesítményt fog nyújtani. Ez esetben a megoldás egy vezeték nélküli hálózatokra tervezett szállítási protokoll. Komoly kihívásokat tartalmaz egy olyan vezeték nélküli hálózat összeállítása, amelyben több, egymással interferáló és egymást keresztező vezeték nélküli adatkapcsolat helyezkedik el, az útvonalak a csomópontok mozgása miatt folyamatosan változnak, és a hálózatban nagy a csomagvesztés. Ezen a területen még kutatások folynak. Egy ilyen szállítási protokoll felépítésre mutat példát Li és mások [2009] műve.

6.4. Az internet szállítási protokolljai: az UDP

Az internet két fő protokollt használ a szállítási rétegben, az egyik összeköttetés-alapú, a másik összeköttetés nélküli. A két protokoll kiegészíti egymást. Az összeköttetés nélküli protokoll az UDP, amely majdnem semmit sem tesz azon túl, hogy csomagokat továbbít alkalmazások között, rájuk bízva, hogy erre egy olyan protokollt építsenek, amilyenre szükségük van. Az összeköttetés-alapú protokoll pedig a TCP, amely szinte mindent elintéz. Összeköttetéseket hoz létre, újraküldéssel megbízhatóvá teszi azokat, emellett forgalomszabályozást és torlódáskezelést is megvalósít, mindezt az alkalmazás helyett teszi.

A következő szakaszban megvizsgáljuk az UDP-t és a TCP-t. Az UDP-vel kezdjük, hiszen az az egyszerűbb. Megnézzük az UDP két tipikus alkalmazását is. Mivel az UDP egy szállítási protokoll, amely tipikusan az operációs rendszer része, és az UDP-t használó protokollok pedig a felhasználói térben (user space) futnak, ezért ezeket a protokollokat alkalmazásoknak is tekinthetjük. Az általuk használt módszerek azonban más alkalmazások részére is hasznosak lehetnek, ezért inkább a szállítási szolgáltatáshoz tartozónak tekintjük, és itt tárgyaljuk ezeket.

6.4.1. Az UDP bemutatása

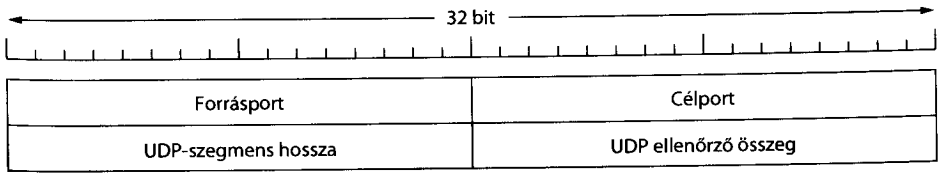
Az internet protokollkészlete egy összeköttetés nélküli protokollt is támogat, ez az **UDP** (**User Datagram Protocol – felhasználói datagram protokoll**). Az UDP olyan alkalmazásoknak kínálja a szolgáltatását, amelyek összeköttetés kiépítése nélkül akarnak IP-datagramokba beágyazott adatokat küldeni. Az UDP leírása a 768-as RFC-ben található.

Az UDP olyan **szegmenseket** (**segment**) használ az átvitelhez, amelyek egy 8 bájtos fejrészből, valamint a felhasználói adatokból állnak. A fejrész a 6.27. ábrán látható. A két port a végpontok forrás- és a célgépen belüli azonosítására szolgál. Amikor egy UDP-szegmens megérkezik, akkor az adatmezejét a szállítási entitás kézbesíti a címzett portra kapcsolódó folyamatnak. A folyamatok a BIND vagy más hasonló primitív használatával kapcsolódhatnak rá egy portra, hasonlóan a TCP 6.6. ábrán is látott esetéhez (a kötési folyamat az UDP-nél is ugyanaz). A portokra úgy gondoljunk, mint olyan postaládákra, amelyeket az alkalmazások ki tudnak bérelni csomagok fogadására. Még több mondanivalónk is lenne velük kapcsolatban, de majd a TCP tárgyalásánál foglalkozunk velük részletesebben is, ugyanis az is használ portokat. Az UDP használatának tulajdonképpen az a legnagyobb előnye a nyers IP használatával szemben, hogy a fejrészben megtalálható a feladó és a címzett portszáma is. A portszámokat tartalmazó mezők nélkül a szállítási réteg nem tudná, hogy mit kezdjen a csomaggal. A segítségükkel azonban a beágyazott szegmenseket a megfelelő alkalmazásoknak tudja kézbesíteni.

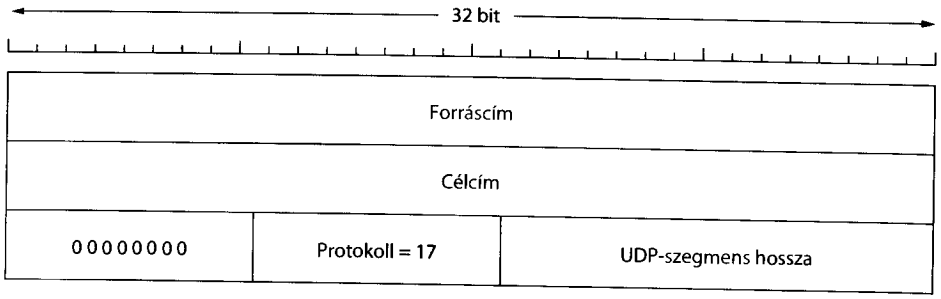
A forrás portjára elsősorban akkor van szükség, amikor választ kel küldeni a feladónak. A választ küldő folyamat úgy tudja megadni, hogy az üzenete a célgép melyik folyamatának szól, hogy a bejövő szegmens *forrásport* mezőjét átmásolja a kimenő szegmens *célport* mezőjébe.

Az *UDP-szegmens hossza* mező a 8 bájtos fejrész és az adatmező együttes hosszát tartalmazza. A legrövidebb lehetséges hossz 8 bájt, ami csak a fejrészt tartalmazza. A legnagyobb lehetséges hossz 65 515 bájt, ugyanis az IP-csomagok mérethatára nem teszi lehetővé a legnagyobb, 16 biten ábrázolható számú bájt használatát.

Az *UDP ellenőrző összeg* mező használata nem kötelező, de használata növeli a megbízhatóságot. Az ellenőrző összeget a fejrészre, az adatra és egy IP-álfejrészre számítják ki. Számítása során az *UDP ellenőrző összeg* mezőt 0-nak veszik, és az adatmezőt további 0 bájttal egészítik ki, ha az *UDP-szegmens hossza* mező páratlan értékű. Az ellenőrző összeg algoritmus a szegmenst 16 bites szavakra bontja, majd egyszerűen kiszámolja az egyes komplemens összegüket és végül a végeredmény egyes komplemensét veszi. Ennek következményeképpen, amikor a vevő ezt a számítást a teljes szegmensre végrehajtja, beleértve az *ellenőrző összeg* mezőt, az eredmény 0 lesz. Az *ellenőrző összeg* mező 0-t tartalmaz abban az esetben, ha nem számították ki, köszönhetően annak a szerencsés



6.27. ábra. *Az UDP-fejrész*



6.28. ábra. *Az UDP ellenőrző összegbe beszámított IPv4-álfejrész*

egybeesésnek, hogy a 0-nak kiszámolt összeg egyes komplemense csupa 1-esből áll. Kikapcsolni azonban nem okos dolog, kivéve, ha egyáltalán nem fontos az átvitt adatok minősége (például digitalizált beszéd).

A 6.28. ábra mutatja az álfejrészt IPv4 esetén. Az álfejrész tartalmazza a forrásgép 32 bites IPv4-címét és a célgép 32 bites IPv4-címét, az UDP-protokoll azonosítószámát (17) és egy bájtszámlálót az UDP-szegmensre (a fejrészt is beleértve). IPv6 esetén némileg különböző, de ezzel hasonló módon járunk el. Az álfejrész beszámítása az UDP ellenőrző összegbe segít megtalálni a tévesen kézbesített csomagokat, azonban megsérti a protokollhierarchiát, hiszen az IP-címek az IP-réteghez tartoznak, nem pedig az UDP-réteghez. A TCP is pontosan ezt az álfejrészt használja az ellenőrző összeg számításához.

Valószínűleg érdemes néhány olyan feladatot külön is megemlíteni, amit az UDP *nem* végez el. Az UDP nem végez forgalomszabályozást, torlódáskezelést vagy újraküldést egy rossz szegmens vétele esetén. Ez mind a felhasználói folyamatokon múlik. Amit elvégez: egy interfészt biztosít az IP-protokoll használatához, azzal a többletképességgel, hogy a portok használatával egyszerre több folyamatot képes demultiplexelni, valamint szükség esetén hibajelzést biztosítani végponttól végpontig. Ez minden, amit az UDP kínál.

Az olyan alkalmazásoknak, amelyeknek nem fontos a csomagforgalom, a hibakezelés vagy az időzítés precíz ellenőrzése, az UDP pontosan azt nyújtja, amire szükségük van. A kliens–szerver-alkalmazásoké például olyan terület, amelyen az UDP kifejezetten hasznos. A kliens gyakran küld olyan rövid kéréseket a szervernek, amelyekre rövid választ vár. Ha vagy a kérés, vagy a válasz elvész, a kliens egyszerűen megvárja, amíg az időzítő lejár, és újra próbálkozik. Így nem csak a kód egyszerűbb, de kevesebb üzenetre is van szükség (egy-egy üzenet mindkét irányban), mint egy összeköttetés felépítését igénylő protokollban (mint például a TCP-ben).

A DNS (Domain Name System – körzetnévkezelő rendszer) például egy olyan alkalmazás, amely ilyen módon használja az UDP-t. A DNS-ről részletesen a 7. fejezetben lesz szó. Röviden most csak annyit, hogy minden olyan programnak, amely valamely hoszt neve (például *www.cs.berkeley.edu*) alapján akarja kikeresni a hoszt IP-címét, egy UDP-csomagot kell elküldenie valamelyik DNS-szervernek. A szerver egy olyan UDP-csomaggal válaszol, amely a hoszt IP-címét tartalmazza. Nincs szükség előzetes összeköttetés-létesítésre, sem az összeköttetés lebontására az átvitel után. A hálózaton csak két üzenet halad át.

6.4.2. Távoli eljáráshívás

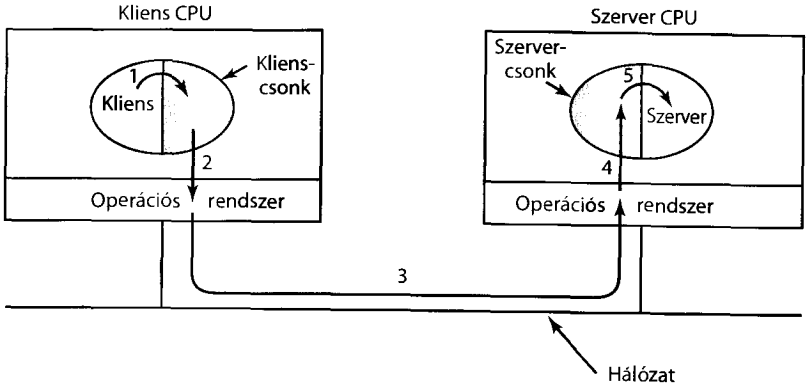
Amikor egy hoszt üzenetet küld egy másik, távoli hosztnak, és választ is kap arra, az bizonyos értelemben nagyon hasonlít a programozási nyelvek eljáráshívására. A kommunikációt mindkét esetben egy vagy több paraméterrel indítjuk, és egy eredményt kapunk vissza. Ez a megfigyelés arra késztette a mérnököket, hogy eljáráshívások formájában próbálják meg lebonyolítani a hálózatokon folyó kérés-válasz jellegű kommunikációt. Az ilyen elrendezés sokkal könnyebbé teszi a hálózati alkalmazások programozását és a használatukat is egységesebbé teszi. Elképzelhetünk például egy olyan, *get_IP_address(hoszt_név)* nevű eljárást, amely úgy működik, hogy egy UDP-szegmenst küld valamelyik DNS-szervernek és megvárja a választ, szükség esetén egy bizonyos idő után újrapróbálkozva, ha egyetlen próbálkozás nem lenne elegendő. Ezzel a hálózat működésének minden részlete elrejthető a programozók elől.

A munka kulcsfontosságú részét ezen a területen Birrell és Nelson [1984] végezte. Dióhéjban összefoglalva, Birrell és Nelson azt javasolta, hogy a programok hívhassanak távoli hosztokon futó eljárásokat is. Amikor az 1. gép egyik folyamata meghív egy eljárást a 2. hoszton, az 1. hoszt rendszere a hívó folyamatot felfüggeszti, és a 2. hoszton megkezdődik a hívás végrehajtása. A szükséges információt a hívótól a hívott felé a paraméterekben, visszafelé pedig az eljárás eredményében lehet átvinni, a programozó elől azonban minden üzenetváltás rejtve marad. Ezt a módszert **RPC**-nek (**Remote Procedure Call – távoli eljáráshívás**) nevezték el és sok hálózati alkalmazás alapjául szolgált már. A hívó folyamatot hagyományosan kliensnek, a hívott folyamatot pedig szervernek hívják. Mi is ezeket a neveket fogjuk használni.

Az RPC alapötlete az, hogy a távoli eljáráshívásoknak minél jobban hasonlítaniuk kell a helyiekhez. A legegyszerűbb formájában a kliensprogramnak a távoli eljárás meghívásához egy kis könyvtári függvényhez kell kapcsolódnia, amelyet **klienscsonknak** (**client stub**) neveztek el. Ez a függvény képviseli a szervereljárást a kliens címterében. Ehhez hasonlóan, a szerver egy **szervercsonknak** (**server stub**) nevezett eljárással áll kapcsolatban. Ezek az eljárások azt a tényt rejtik el, hogy a kliens függvényhívása nem helyi függvényhívás.

Egy távoli eljáráshívás gyakorlati lépéseit a 6.29. ábra szemlélteti. Az első lépésben a kliens meghívja a klienscsonkot. Ez a hívás egy helyi eljáráshívás, így a paraméterei a megszokott módon a verembe kerülnek. A második lépésben a klienscsonk a paramétereket belepakolja egy üzenetbe, és egy rendszerhívást hajt végre, amellyel elküldi ezt a csomagot. A paraméterek csomagba pakolását **rendezésnek** (**marshaling**) nevezik. A harmadik lépésben az operációs rendszer átküldi az üzenetet a kliensgépről a szervergépre. A negyedik lépés az, hogy a szerver operációs rendszere átadja a bejövő csomagot a szervercsonknak. Végül az ötödik lépésben a szervercsonk meghívja a szervereljárást a visszarendezett paraméterekkel. A válasz ugyanezen az útvonalon halad végig az ellenkező irányban.

A dolog legfontosabb része az, hogy a felhasználó által írt kliensfolyamat egy olyan, teljesen szokványos (vagyis helyi) eljáráshívást hajt végre a klienscsonkon, amelynek a neve is ugyanaz, mint a szervereljárásnak. Mivel a klienseljárás és a klienscsonk ugyanabban a címtérben van, a paramétereket a szokásos módon lehet átadni. Ehhez hasonlóan a szervereljárást is egy olyan folyamat hívja meg, amely a saját címterében van, olyan



6.29. ábra. *Egy távoli eljáráshívás végrehajtásának lépései. A csonkokat satírozással jelöltük*

paraméterekkel, amilyeneket vár. A szervereljárás nem érzékel semmi szokatlan dolgot. Ezen a módon tulajdonképpen a hálózati kommunikációt szokványos eljáráshívásnak álcázzuk ahelyett, hogy a csatolókon keresztül intéznénk a be- és kivitelt.

Az RPC-nek minden elméleti eleganciája ellenére van néhány rejtett hátulütője is. Ezek közül az egyik a mutató típusú paraméterek használatakor jelentkezik. Általában nem baj, ha egy eljárásnak mutatót kell átadni, mert a meghívott eljárás a hívóval teljesen azonos módon használhatja azt, mivel a két eljárás ugyanabban a virtuális címzési térben létezik. RPC-vel lehetetlen mutatókat átadni, mert a kliens és a szerver két különböző címtérben helyezkedik el.

Egyes esetekben azért bizonyos trükkök bevetésével megvalósítható a mutatók átadása. Tegyük fel, hogy az első paraméter egy olyan mutató, amely egy *k* egész számra mutat. A klienscsonk a tényleges *k* számot rendezi be a csomagba, és azt küldi el a szervernek. A szervercsonk létrehoz egy mutatót *k*-ra és ezt adja át a szervereljárásnak pontosan úgy, ahogyan az eljárás azt elvárja. Amikor a szervereljárás visszaadja a vezérlést a szervercsonknak, a csonk visszaküldi a *k* értéket a kliensnek, amely az új *k*-t bemásolja az eredeti helyére, mivel a szerver a feldolgozás során megváltoztathatta az értékét. A szokásos hivatkozásos eljáráshívást így gyakorlatilag másolással és visszaállítással helyettesítjük. Ez a trükk azonban sajnos nem mindig működik. Például nem alkalmazható, ha a mutató egy gráfra vagy más összetett adatstruktúrára mutat. Emiatt néhány korlátozást kell bevezetnünk a távolról meghívott eljárások paraméterezésében, ahogy azt látni fogjuk.

A második gond az, hogy a gyengén típusos nyelvekben (például a C) teljesen szabályos olyan eljárást írni, amely anélkül számítja ki két vektor (tömb) belső szorzatát, hogy bármelyikük komponenseinek számát megadtuk volna. Az egyes vektorok adott esetben csak a hívó és a hívott eljárás által ismert értékekkel is le lehetnek zárva. Ilyen körülmények között a klienscsonknak lényegében lehetetlen a paraméterek csomagokba rendezése; mivel sehogyan sem tudja kideríteni a méretüket.

A harmadik probléma az, hogy nem mindig lehetséges kikövetkeztetni a paraméterek típusát, még egy formális specifikációból vagy magából a kódból sem. Erre jó példa a *printf*, amely tetszőleges számú paraméterrel (de legalább eggyel) rendelkezhet, és a

paraméterei egészek, rövid és hosszú ábrázolású számok, karakterek, füzérek, különféle hosszúságú lebegőpontos számok és más típusok tetszőleges keverékei lehetnek. A *printf* eljárás távoli hívása a gyakorlatban éppen azért lenne lehetetlen, mert a C ennyire engedékeny. Mindezek ellenére egy olyan szabály nem lenne népszerű a programozók körében, amely azt mondaná ki, hogy csak akkor használhatunk RPC-t, ha nem C-ben (vagy C++-ban) írjuk a programunkat.

A negyedik probléma a globális változók használatával kapcsolatos. Rendes körülmények között a hívó és a hívott eljárás a paramétereken keresztül történő kommunikáció mellett kommunikálhat globális változók használatával is. Ha a hívott eljárást most képzeletben áttesszük egy távoli gépre, akkor a kód nem fog működni, mivel a két eljárás már nem osztozik a globális változókon.

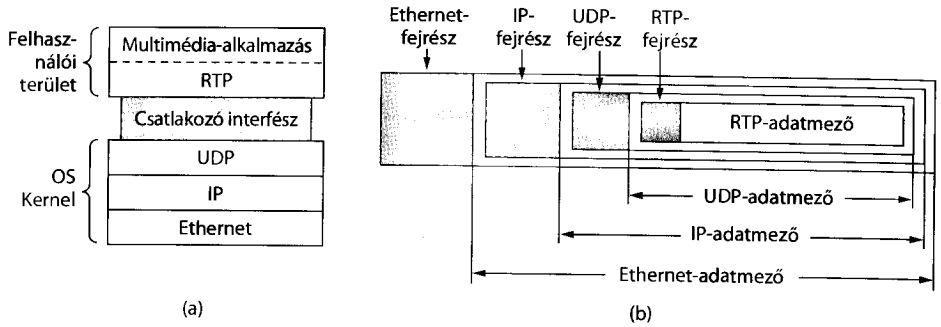
Az RPC hiányosságainak fenti felsorolásával azonban nem azt akartuk kifejezni, hogy az RPC használhatatlan. Az RPC valójában széles körben használatos, de szükség van néhány korlátozásra, hogy a gyakorlatban is jól működjön.

A szállítási protokollok körében az UDP jó választás az RPC megvalósítására. A legegyszerűbb esetben mind a kérések, mind a válaszok elküldhetők egyetlen UDP-csomagban, és a működése is gyors lesz. A megvalósításnak azonban egyéb eszközöket is fel kell használnia. Mivel a kérések és válaszok elveszhetnek, ezért a kliensnek egy időzítőt kell működtetnie a kérés újraküldésére. Vegyük észre, hogy a kéréseket felesleges külön nyugtázni, hiszen a válasz a kérés implicit nyugtájaként is szolgál. Néha a paraméterek vagy a válaszok olyan nagyok is lehetnek, hogy nem férnek el egyetlen UDP-szegmensben. Ilyen esetben valamilyen egyéb protokollt kell alkalmazni nagyobb üzenetek küldésére. Ha időben több kérés és válasz átlapolódhat (mint például a konkurens programozás esetében), akkor egy azonosítót kell csatolni az üzenetekhez, hogy a megfelelő kéréseket és válaszokat össze lehessen párosítani.

Magasabb szintű probléma, hogy a művelet nem feltétlenül idempotens (például nem ismételhető meg biztonságosan). A legegyszerűbb esetben a művelet idempotens, mint például a DNS-kérések és -válaszok. Ezeket a kéréseket a kliens újra és újra elküldheti, ha nem érkezik válasz, hiszen nem számít, hogy a szerver egyszer sem kapta meg a kérést, vagy a válasz veszett el. Amikor végül megérkezik a válasz, ugyanaz lesz (feltéve, hogy időközben nem frissítették a DNS-adatbázist). Mindazonáltal nem minden művelet idempotens, mert például olyan mellékhatásokat okoznak, mint mondjuk egy számláló növelése. Ilyen műveletek esetén az RPC erősebb szemantikát igényel, hogy amikor a programozó meghívja az eljárást, az ne fusson le többször. Ebben az esetben előfordulhat, hogy szükséges lehet UDP helyett egy TCP-összeköttetést felépíteni, és a kérést azon keresztül elküldeni.

6.4.3. Valós idejű szállítási protokollok

A kliens–szerver RPC olyan terület, ahol az UDP széles körben használatos. Egy másik ilyen a valós idejű multimédiás alkalmazásoké. Ahogyan az internetes rádió, az internetes telefon, a hálózati zeneszolgáltatás (music-on-demand), a videokonferencia, videoszolgáltatás (video-on-demand) és más multimédiás alkalmazások egyre elterjedtebbé váltak, a tervezők észrevették, hogy minden egyes alkalmazáshoz többé-kevésbé



6.30. ábra. *(a) Az RTP helyzete az egymásra épülő protokollok között. (b) Az adategységek egymásba ágyazódása*

ugyanazt a valós idejű szállítási protokollt találták fel újra és újra. Fokozatosan egyre nyilvánvalóbbá vált, hogy jó ötlet lenne kitalálni egy általános, több célra alkalmazható valós idejű szállítási protokollt.

Így született meg a napjainkban multimédia-alkalmazások széles körében használatos **RTP (Real-Time Transport Protocol – valós idejű szállítási protokoll)**, amelyet az 3550-es RFC ír le. A valós idejű átvitelt két szemszögből mutatjuk be. Az első RTP-protokoll hang- és videoadatokat csomagokban továbbít. A második pedig az, amelyik a hang és videó megfelelő időben történő lejátszásához szükséges, legtöbbször a vevő oldalán történő feldolgozás révén. Ezek a feladatok a 6.30. ábrán látható módon illenek a protokollkészletbe.

Az RTP-t a felhasználó címterébe utalták, és általában az UDP-re ráépülve, az operációs rendszerben fut. A működése a következő. A multimédiás alkalmazások több hang-, mozgókép-, szöveg- és esetleg egyéb folyamból épülnek fel. Ezeket betöltik az RTP-könyvtárba (RTP library), amely az alkalmazással együtt a felhasználói címtérben található. Ez a könyvtár multiplexeli és RTP-blokkokba kódolja a folyamokat, amelyeket ezután egy csatlakozóra (socket) továbbít. A csatlakozó másik végén (az operációs rendszerben) ezekből UDP-szegmensek lesznek, amelyeket a rendszer IP-csomagokba ágyaz be, amelyek például Ethernet-keretekbe kerülnek az átvitelhez. A vevő oldalán mindez pont fordítva történik. A multimédia-alkalmazás végül multimédia-adatot kap az RTP-könyvtártól. A lejátszásáért a multimédia-alkalmazás felel. A protokollok egymásra épülését ebben a helyzetben a 6.30.(a) ábra mutatja be. A 6.30.(b) ábrán az adategységek egymásba ágyazódása látható.

Ennek a felépítésnek az egyik következményeként kissé nehéz meghatározni, hogy az RTP melyik rétegben van. Mivel a felhasználói területen fut és az alkalmazási programmal összekapcsolták, határozottan úgy néz ki, mint egy alkalmazási protokoll. Másrészt viszont ez egy olyan általános, alkalmazásfüggetlen protokoll, amely csak szállítási lehetőséget nyújt, így nagyon hasonlít egy szállítási protokollra is. A legjobb leírás talán az, hogy az RTP egy alkalmazási rétegben megvalósított szállítási protokoll, éppen ezért ebben a fejezetben tárgyaljuk.

RTP – a valós idejű szállítási protokoll

Az RTP alapvető feladata az, hogy több valós idejű adatfolyamot multiplexeljen UDP-szegmensek egyetlen folyamába. Az UDP-folyamot egy címre (egyesküldés), vagy több címre (többesküldés) is feladhatja. Mivel az RTP csak a szabványos UDP-t használja, a blokkjait az útválasztók nem kezelik különleges módon, azt az esetet kivéve, ha a hálózaton valamelyik szokványos IP szolgáltatásminőségi tulajdonság engedélyezve van. Ez elsősorban azt jelenti, hogy nincsenek különleges garanciák a kézbesítésre, és a csomagok elveszhetnek, késhetnek, megsérülhetnek stb.

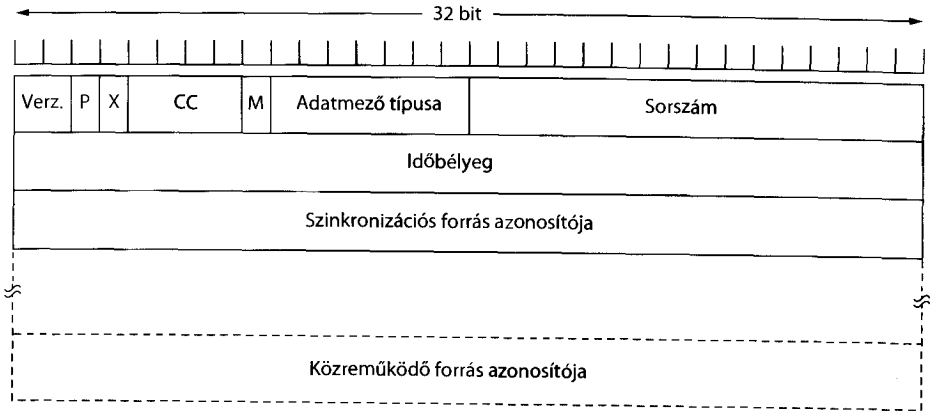
Az RTP számos lehetőséget biztosít a vevők részére a multimédia-információk kezeléséhez. Egy RTP-folyamban minden csomag kap egy sorszámot, amely eggyel nagyobb az azt megelőző csomagénál. A címzett a sorszámozás segítségével tudja eldönteni, hogy hiányoznak-e csomagok. Ha egy csomag nem érkezik meg, az alkalmazásnak kell döntenie, mi történjen. Videoadat esetén elképzelhető, hogy kihagy egy képkockát, vagy hangadat esetén interpolációval közelíti a hiányzó értéket. Az újraküldés ebben az esetben nem jó megoldás, hiszen az újra elküldött csomag valószínűleg túl későn érkezne meg ahhoz, hogy még használható legyen. Ennek következtében az RTP nem használ nyugtázást és újraküldést kérő megoldásokat.

Minden RTP-adatmezőben több minta kaphat helyet, bármely olyan kódolásban, amelyet az alkalmazás használni akar. A közreműködés megkönnyítésére az RTP számos profilt definiál (például egyetlen hangfolyam) és minden profilhoz több kódolási formátumot enged meg. Például egy egyedüli hangfolyamot kódolhatunk 8 bites PCM-mintákba 8 kHz-en, delta kódolással, prediktív kódolással, GSM-kódolással, MP3-formátumban és így tovább. Az RTP a fejrészben biztosít egy mezőt a forrásnak arra, hogy megjelölje a kódolást, de egyébként nem avatkozik bele a kódolás részleteibe.

Az időbélyegek (timestamp) kezelése egy másik olyan lehetőség, amelyre sok valós idejű alkalmazásban szükség van. Az alapötlete az, hogy a forrásgép minden csomag első mintájához egy időbélyeget rendelhet hozzá. Az időbélyegeket a folyam kezdetéhez kell viszonyítani, így csak az időbélyegek különbsége lényeges, az abszolút értéküknek nincs jelentése. Amint azt hamarosan megmutatjuk, ez a megoldás lehetővé teszi a célgép számára, hogy egy kis puffert használjon, és minden mintát megfelelő számú milliszekundummal a folyam kezdete után játsszon le, a mintát tartalmazó csomag megérkezési idejétől teljesen függetlenül.

Az időbélyegek használata nem csak a sebességingadozások hatásait egyenlíti ki, de azt is lehetővé teszi, hogy több folyamot szinkronizáljunk egymással. Például, egy digitális televízióadás állhat egy mozgóképfolyamból és két hangfolyamból. A két hangfolyamot egyrészt sztereoadások sugárzására lehet használni, másrészt olyan filmek sugárzására, amelyeknek mind az eredeti nyelvű hangsávját, mind a helyi nyelvre lefordított hangsávját egyszerre sugározzák, szabad választást kínálva a nézőnek. Minden folyam különböző fizikai eszközről érkezik, de ha ugyanazzal a számlálóval időbélyegezik, akkor szinkronban is le lehet azokat játszani még akkor is, ha a folyamokat elcsúszva és szeszélyesen adják és/vagy veszik.

Az RTP fejrészét a 6.31. ábra szemlélteti. Három 32 bites szóból és esetleg néhány kiterjesztésből áll. Az első szó a kétbites *Verzió* mezőt tartalmazza, amely már 2-nél tart. Reméljük, hogy ez a verzió már nagyon közel van a végső verzióhoz, mivel már csak



6.31. ábra. *Az RTP-fejrész*

egy érték maradt kihasználatlan (bár a 3-at lehetne úgy definiálni, hogy egy kiterjesztő szóban leírt verziószámra utaljon).

A *P* bit azt jelzi, hogy a csomagot 4 bájt vagy annak többszörösére kiegészítették (padding). Az utolsó kiegészítő bájt (padding byte) adja meg, hogy hány kiegészítő bájt került a csomagba. Az *X* bit azt jelzi, hogy a blokk egy kiterjesztő fejrésszel (eXtension header) rendelkezik. A kiterjesztő fejrész formátuma és jelentése itt nincs meghatározva. Az egyetlen definiált dolog az, hogy a kiterjesztés első bájtja adja meg a kiterjesztés hosszát. Ez egy beépített vészmegoldás az előre nem látott követelmények teljesítéséhez.

A *CC* mező azt mondja meg, hogy hány hozzájáruló forrás (lásd alább) van jelen, az értéke 0-tól 15-ig terjedhet. Az *M* bit egy alkalmazásfüggő jelölő bit (marker bit). Ez a bit jelölheti egy mozgóképkeret elejét, egy szó kezdetét egy hangcsatornán vagy bármi mást, amit az alkalmazás megért. Az *Adatmező típusa* mező adja meg a használt kódolási algoritmust (tömörítetlen 8 bites hang, MP3 stb.). Mivel ez a mező minden csomagban jelen van, a kódolást akár adás közben is meg lehet változtatni. A *Sorszám* mindössze egy csomagszámláló, amely minden elküldött RTP-csomagnál eggyel nő, és az elveszett blokkok felderítésére szolgál.

Az időbélyeget a folyam forrása állítja elő, hogy feljegyezze a csomag első mintájának keletkezési idejét. Ez az érték segítséget nyújthat az időzítési szórás, ún. dzsitter (jitter) csökkentésében a vevő oldalán, mivel lehetővé teszi a lejátszás függetlenítését a csomagok érkezési idejétől. A *Szinkronizációs forrás azonosítója* azt adja meg, hogy a csomag melyik folyamhoz tartozik. Ez a párhuzamos adatfolyamok egyetlen UDP-szegmensfolyamba való multiplexelését, majd demultiplexelését teszi lehetővé. Végül, a csomag tartalmazhat *Hozzájáruló forrás azonosítója* mezőket. Ezeket akkor használják, ha keverők is jelen vannak a forrásoldalon. Ebben az esetben a keverő a szinkronizációs forrás, és minden egybekevert folyamot ezekben a mezőkben sorolnak fel.

RTCP – valós idejű szállítást vezérlő protokoll

Az RTP-nek van egy kistestvére is, amelyet **RTCP-nek (Real-Time Transport Control Protocol – valós idejű szállítást vezérlő protokollnak)** neveztek el. A protokollt az RFC 3550 definiálja (az RTP mellett), és a visszacsatolást, a szinkronizációt és a felhasználói interfészt kezeli, de mintákat nem szállít.

Az első tulajdonságát arra használhatjuk, hogy visszacsatolást biztosítsunk a forrásgépeknek a késleltetésről, annak ingadozásáról (jitter), a sávszélességről, a torlódásokról és más hálózati tulajdonságokról. Ezt az információt a kódoló folyamat arra használhatja, hogy növelje az adatsebességet (és így javítsa a minőséget), amikor a hálózat jól működik, és visszavegye az adatsebességet, amikor baj van a hálózattal. A folyamatos visszacsatolás segítségével a kódoló algoritmusok mindig a lehető legjobb minőséget tudják biztosítani, folyamatosan alkalmazkodva a pillanatnyi körülményekhez. Például, ha az átvitel során a sávszélesség csökken vagy növekszik, akkor a kódolást szükség szerint váltogathatjuk az MP3 és a 8 bites delta kódolás között. Az *Adatmező típusa* mező a célgépet tájékoztatja az egyes csomagokhoz használt kódolási eljárásról, így szükség esetén lehetővé teszi annak azonnali megváltoztatását.

A visszacsatolással kapcsolatban azonban felmerül egy probléma is, ugyanis az RTCP a jelentéseket minden résztvevőnek szétküldi. Egy nagy csoporttal bíró többesküldéses alkalmazás esetén az RTCP által elfoglalt sávszélesség gyorsan megnőhet. Ennek elkerülésére az RTCP-küldők annyira lecsökkentik a küldési sebességüket, hogy az összes résztvevő által elfoglalt sávszélesség ne legyen több mint mondjuk a média által elfoglalt sávszélesség 5%-a. Ennek megvalósításához minden résztvevőnek tudnia kell a média által elfoglalt sávszélességről, melyet a küldőktől tudhat meg, és a résztvevők számáról, amelyet az RTCP-jelentésekre figyelve becsülhet meg.

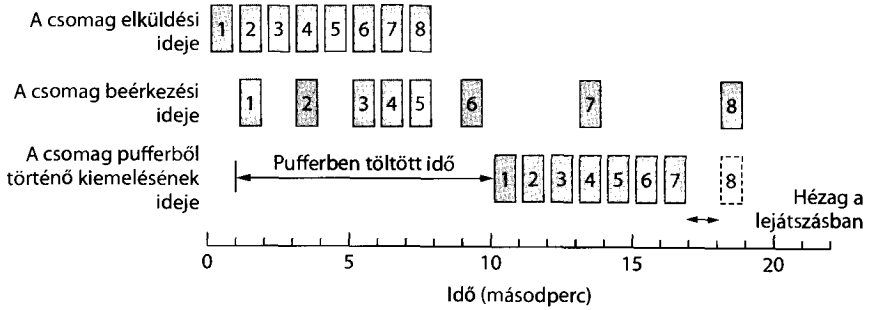
Az RTCP a folyamok közötti szinkronizációt is kezeli. A baj az, hogy a különböző folyamok más-más órákat használhatnak, amelyeknek a finomsága (granularity) és az elcsúszása (drift) is különböző lehet. Az RTCP használatával ezeket szinkronban lehet tartani.

Végül, az RTCP arra is lehetőséget nyújt, hogy a különböző forrásokat megnevezzük (például ASCII-szövegben). Ezt az információt meg lehet jeleníteni a vevő képernyőjén, jelezve a számára, hogy ki beszél éppen.

Az RTP-ről további információ Perkins [2003] könyvében található.

Lejátszás pufferelléssel és dzsitterszabályozással

Miután a médiainformáció megérkezett a vevőhöz, le kell játszani a megfelelő időben. Ez az idő általában nem az a pillanat, amikor az RTP-csomag megérkezik a vevőhöz, ugyanis a csomagok áthaladása a hálózaton más-más időt vehet igénybe. Még akkor is, ha a küldő pontosan egyenlő időközönként bocsátja a hálózatra a csomagokat, a vevőhöz különböző relatív időpontokban érkeznek meg. A késleltetésben jelentkező ilyen szórást **dzsitternek (jitter)** hívjuk. Még egy aránylag kis mértékű csomagdzsitter is kiábrándító médiaélményt tud okozni, például ugráló képkockákat és érthetetlen beszédet, ha a médiát megérkezésekor egyből lejátsszuk.



6.32. ábra. *A kimeneti folyam kisimítása a csomagok pufferelésével*

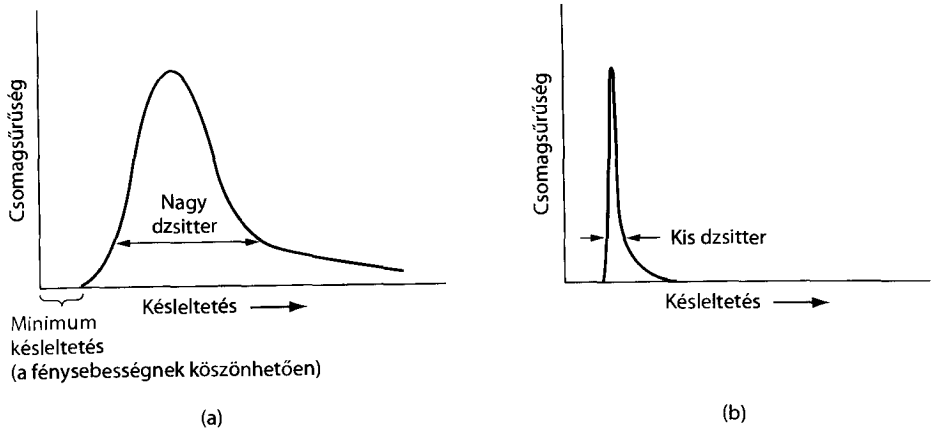
A megoldás a dzsitter csökkentésére az, hogy a csomagokat **pufferelni** kell a vevőnél, mielőtt lejátszanák azokat. A 6.32. ábrán egy olyan folyamra mutatunk példát, ahol a csomagok tekintélyes dzsitterrel érkeznek meg a vevőhöz. Az 1. csomagot a szerver $t=0$ másodperckor küldte, és $t=1$ másodpercnél érkezik a vevőhöz. A 2. csomag nagyobb késleltetést szenved, és 2 másodperc telik el a küldése és megérkezése között. Ahogy a csomagok megérkeznek, a kliens állomás puffereli azokat.

A $t=10$ másodpercnél a lejátszás elindul. Ebben a pillanatban az 1-től 6-ig terjedő csomagok már a pufferben vannak, tehát egyenlő időközönként eltávolíthatjuk és lejátszhatjuk azokat a gördülékeny lejátszás érdekében. Általános esetben nem szükséges egyenlő időközöket használni, hiszen az RTP-időbélyegek megmondják, hogy mikor kell a médiát lejátszani.

Sajnos a 8. csomag annyit késik, hogy nem érhető el akkor, amikor a lejátszására kerülne a sor. Ebben az esetben két lehetőség van. A 8. csomagot figyelmen kívül hagyjuk, és a lejátszó a következő csomaggal folytatja a lejátszást. A másik lehetőség szerint a lejátszás szünetelhet addig, amíg meg nem érkezik a 8. csomag, mindezzel zavaró megakadást okozva a zenében vagy filmben. Egy élő médiaalkalmazásban, mint például egy VoIP-hívás esetén, a csomagot tipikusan eldobják, ugyanis az élő alkalmazások nem nagyon állíthatók le. Egy médiaközvetítési alkalmazás azonban a lejátszó akár le is állhat. A probléma a lejátszás kezdeti idejének eltolásával, és így nagyobb puffer alkalmazásával enyhíthető. Video- vagy hangközvetítés esetén gyakran 10 másodperc körüli puffereket használnak, hogy biztosítsák az összes (a hálózat által nem eldobott) csomag megérkezését időben. Élő alkalmazásokhoz, mint például videokonferenciához, kis pufferek használata célszerű, hogy a lejátszási késleltetést csökkentsék.

A legfontosabb tényező a gördülékeny lejátszással kapcsolatban a **lejátszási pont** (**playback point**), azaz mennyit kell várni a vevőnek, hogy a médiát lejátszhassa. Ennek eldöntése a dzsitteren múlik. A különbség a kis és nagy dzsitterű kapcsolatok között a 6.33. ábrán látható. Az átlagos késleltetés nem különbözik nagyon a két esetben, de nagy dzsitter esetén a lejátszási pontot sokkal kijjebb kell tolni, hogy a csomagok 99%-át elkapjuk, mint ha a dzsitter kicsi volna.

A helyes lejátszási pont megválasztásához az alkalmazás megmérheti a dzsittert az RTP-időbélyegek és a beérkezési idők különbségének a figyelésével. Minden különbség



6.33. ábra. *(a) Nagy dzsitter. (b) Kis dzsitter*

egy késleltetési mintát ad (növelve egy tetszőleges, de állandó késleltetéssel). A késleltetés azonban az idővel változhat, más, versengő forgalmak és változó útvonalak következtében. Hogy az alkalmazások alkalmazkodjanak ehhez a változáshoz, a lejátszási pontjukat eltolhatják a futás során. A lejátszási pont helytelen módon történő megváltoztatása azonban észrevehető hibát okozhat a felhasználónál. Hang esetén az jelenthet megoldást, ha a lejátszási pontot a **beszédlöketek** (**talkspurt**) között, azaz a beszédszünetekben változtatjuk meg. Senki sem fogja észrevenni a különbséget egy rövid és egy hosszabb szünet között. Az RTP éppen ezért lehetővé teszi az alkalmazások számára, hogy az *M* jelölőbittel megjelöljék a beszédlöketek kezdetét.

Az élő alkalmazások nem tudják elviselni, ha a média lejátszásáig tartó abszolút késleltetés túl nagy. Ha már eddig is egy közvetlen útvonalat használunk, semmi sem tudja csökkenteni a terjedési késleltetést. A lejátszási pontot beljebb lehet húzni, ha elfogadjuk, hogy a csomagok nagyobb hányada fog késve megérkezni a lejátszáshoz. Ha ez nem elfogadható, akkor az egyetlen módja a lejátszási pont beljebb húzásának, ha a dzsittert jobb szolgáltatásminőségi módszerek használatával csökkentjük, például sürgős továbbítást használunk differenciált szolgáltatásokkal. Azaz, egy jobb minőségű hálózatot kell használnunk.

6.5. Az internet szállítási protokolljai: a TCP

Az UDP egy egyszerű protokoll és van néhány nagyon fontos alkalmazása, mint például a kliens–szerver-interakciók és a multimédia. A legtöbb internetes alkalmazás azonban megbízható, sorrendhelyes kézbesítést igényel. Az UDP ezt nem tudja nyújtani, ezért egy másik protokollra is szükség van. Ezt a protokollt TCP-nek hívják, és ez az internet legfontosabb igáslova. Lássunk neki a részletes tanulmányozásának!

6.5.1. A TCP bemutatása

A TCP-t (**Transmission Control Protocol – átvitelvezérlő protokoll**) kifejezetten arra tervezték, hogy megbízható bájtfolyamot biztosítson a végpontok között egy egyébként megbízhatatlan összekapcsolt hálózaton. Egy összekapcsolt hálózat abban különbözik egyetlen hálózattól, hogy az egyes részeinek topológiája, sávszélessége, késleltetése, csomagmérete és más paraméterei nagymértékben különbözhetnek. A TCP-t arra tervezték, hogy dinamikusan alkalmazkodjon az összekapcsolt hálózatok tulajdonságaihoz, valamint hogy nagymértékben ellenálló legyen sokféle meghibásodással szemben.

A TCP-t formálisan 1981 szeptemberében, a 793-as RFC-ben definiálták, az idő előrehaladtával azonban sok hibára és belső ellentmondásra derült fény, ezért ezeket kijavították, és a protokollt sokat tökéletesítették. Hogy képet adjunk a TCP terjedelméről, a legfontosabb RFC-k jelenleg az RFC 793, továbbá: hibajavítások és tisztázások az RFC 1122-ben; kiterjesztések a nagy teljesítőképesség érdekében az RFC 1323-ban; szelektív nyugtázás az RFC 2018-ban; torlódáskezelés az RFC 2581-ben; némely fejrészmező újraértelmezése a szolgáltatásminőség érdekében az RFC 2873-ban; továbbfejlesztett újraküldési időzítők az RFC 2988-ban; explicit torlódásjelzés az RFC 3168-ban. A teljes lista sokkal hosszabb, ami egy, az RFC-k közötti útmutató létrehozásához vezetett, természetesen egy újabb RFC formájában, ez az RFC 4614.

Minden TCP-t támogató gép rendelkezik egy TCP szállítási entitással, amely lehet egy könyvtári eljárás, egy felhasználói folyamat vagy leggyakrabban a kernel része. A TCP-folyamokat és az IP-réteg felé használható interfészeket minden esetben a TCP-entitás kezeli. A helyi folyamatoktól kapott felhasználói adatfolyamokat a TCP-entitás 64 KB-ot meg nem haladó méretű darabokra szedi szét (a gyakorlatban sokszor 1460 adatbájtos darabokra, mert így az IP- és TCP-fejrészekkel együtt is belefférnek egy Ethernet-keretbe). Az egyes darabokat önálló IP-datagramokban küldi el. Amikor egy géphez TCP-adatokat tartalmazó datagram érkezik, az a TCP-entitáshoz kerül, amely visszaállítja az eredeti bájtfolyamokat. Az egyszerűség kedvéért néha olyankor is a „TCP" szót fogjuk használni, amikor a TCP-entitásról (ami egy szoftverelem) vagy a TCP-protokollról (ami egy szabálykészlet) beszélünk, a szövegkörnyezetből azonban mindig világos lesz, hogy melyikről van szó. Például, ha azt mondjuk, hogy „a felhasználó átadja a TCP-nek az adatokat", akkor nyilván a TCP szállítási entitásról van szó.

Az IP-réteg nem ad garanciát a datagramok helyes kézbesítésére, sem útmutatást arra, hogy a datagramokat milyen sebességgel lehet küldeni. A TCP-n múlik, hogy a datagramokat olyan sebességgel küldje, hogy a teljes kapacitást kihasználja, de ne okozzon torlódást. A TCP-re marad az időzítők kezelése és a szükség szerinti újraküldés. A helyesen megérkező datagramok is érkezhetnek rossz sorrendben, ezért a TCP felelőssége az is, hogy a datagramokat megfelelő sorrendben rakja össze üzenetekké. A fentieket röviden összefoglalva azt mondhatjuk, hogy a TCP-nek kell megfelelő teljesítőképesség mellett megteremtenie azt a megbízhatóságot, amelyet a legtöbb felhasználó megkíván, és amelyet az IP nem ad meg.

6.5.2. A TCP szolgáltatási modellje

A TCP-szolgáltatás úgy valósul meg, hogy mind a küldő, mind a fogadó létrehoz egy **csatlakozónak** (**socket**) nevezett végpontot, ahogy azt a 6.1.3. pontban tárgyaltuk. Minden csatlakozónak van egy száma, azaz csatlakozócíme, ami a hoszt IP-címéből és egy hoszton belül érvényes 16 bites számból, a **port** (**kapu**) azonosítójából tevődik össze. A port a TCP-környezetben használt elnevezése a TSAP-nek. A TCP-szolgáltatás megvalósításához egy közvetlen összeköttetést kell létesíteni a küldő- és fogadógép csatlakozói (socketjei) között. A csatlakozókat kezelő hívásokat a 6.5. ábrán foglaltuk össze.

Egy csatlakozó egyidejűleg több összeköttetés kezelésére is használható, azaz két vagy több összeköttetés közös csatlakozóban is végződhet. Az összeköttetéseket a két végükön található csatlakozók azonosítói azonosítják: (*socket1, socket2*). Nem használják a virtuális áramkör sorszámát vagy más azonosítót.

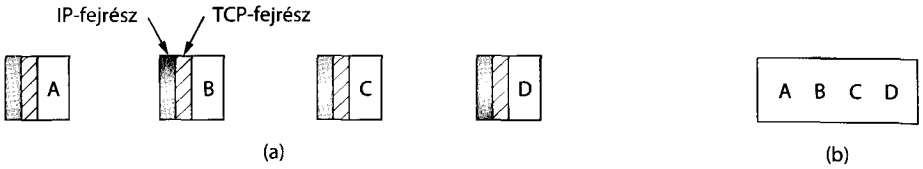
Az 1024 alatti portokat **jól ismert** vagy **közismert portoknak** (**well-known port**) hívják, és a megszokott szolgáltatások részére vannak fenntartva, amelyeket csak privilegizált felhasználók indíthatnak el (például *root* felhasználó UNIX-rendszereken). Például, bármely olyan folyamat, amely egy távoli levelezőszerverről szeretne leveleket letölteni, a levelezőszerver 143-as portján kapcsolatba léphet a szerveren futó IMAP-démonnal. A jól ismert portok listája megtalálható a *www.iana.org*-on. Eddig több mint 700 portot osztottak ki. A 6.34. ábra néhány ismertebbet sorol fel ezek közül.

Az 1024 és 49 151 közötti portok is regisztrálhatók az IANA-nál, de ezeket egyszerű felhasználók is használhatják, és az alkalmazások megválaszthatják, milyen portot használjanak. Például a BitTorrent P2P-állománymegosztó alkalmazás (nem hivatalosan) a 6881–6887 portokat használja, de más portokon is futhat.

Teljes mértékben lehetséges volna, hogy a gép indításakor az FTP-démon automatikusan rákapcsolódjon a 21-es portra, az SSH démon rákapcsolódjon a 22-es portra, és így tovább. Ezzel a megoldással azonban telezsúfolnánk a memóriát olyan démonokkal, amelyek az idő jelentős részében tétlenek. A UNIX-ban ehelyett általában egyetlen, **inetd**-nek (**Internet Daemon – internetdémon**) nevezett démont alkalmaznak. Az *inetd* egyszerre több portra kapcsolódik, és mindegyiken várja a bejövő összeköttetési

Port	Protokoll	Alkalmazás
20, 21	FTP	Állományátvitel
22	SSH	Távoli bejelentkezés, Telnet helyett
25	SMTP	E-levelezés
80	HTTP	Világháló (web)
110	POP-3	Távoli e-levél hozzáférés
143	IMAP	Távoli e-levél hozzáférés
443	HTTPS	Biztonságos világháló (HTTP SSL/TLS felett)
543	RTSP	Médialejátszó-vezérlés
631	IPP	Nyomtatómegosztás

6.34. ábra. *Néhány közismert port*



6.35. ábra. *(a) Négy 512 bájtos szegmens, amelyeket az adó külön IP-datagramokban küld el. (b) A 2048 adatbájtot a vevő alkalmazási folyamat egyetlen* READ *hívásban kapja meg.*

kéréseket. Amikor kérés érkezik, az *inetd* egy új folyamatot indít el, amelyben lefuttatja a megfelelő démont, így az lekezelheti a kérést. Ezzel a megoldással elérhető, hogy (az *inetd*-t kivéve) minden démon csak akkor fusson, amikor munkája is van. Az *inetd* a beállításait tartalmazó állományból tudja, hogy mely portokat kell figyelnie. Ez lehetővé teszi a rendszergazdának, hogy a legforgalmasabb portokra (például a 80-as portra) állandó démonokat tegyen és az összes többit az *inetd*-re bízza.

Minden TCP-összeköttetés duplex és kétpontos. A duplex azt jelenti, hogy a forgalom egyszerre haladhat minkét irányba. A kétpontos azt jelenti, hogy minden összeköttetésnek pontosan két végpontja van. A TCP nem támogatja az adatszórást és a többesküldést.

A TCP-összeköttetésen nem üzenetfolyamok, hanem bájtfolyamok áramlanak. Az üzenethatárokat a rendszer nem őrzi meg. Például, ha a küldőfolyamat négy 512 bájtos írást hajt végre a TCP-folyamon, az a vevőhöz megérkezhet négy 512 bájtos darabban, két 1024 bájtos darabban, egy 2048 bájtos darabban (lásd 6.35. ábra) vagy akár más felosztásban is. A vevőnek nem áll módjában kideríteni, hogy az adatokat mekkora darab(ok)ban küldték, akárhogy is próbálkozik.

A UNIX-os állományok is rendelkeznek ezzel a tulajdonsággal. Az állomány olvasója nem tudja megállapítani, hogy az állományt blokkonként, bájtonként vagy egyetlen darabban írták-e. A UNIX állománykezeléséhez hasonlóan a TCP-szoftver sem tud semmit a bájtok jelentéséről, és azt kitalálni sem próbálja. A TCP számára egy bájt csak egy bájt.

Amikor egy alkalmazás adatot ad át a TCP-nek, a TCP saját belátása alapján dönti el, hogy azonnal elküldi azt, vagy puffereli egy ideig, hogy így nagyobb adatmennyiséget küldhessen el egyszerre. Az alkalmazásnak azonban néha fontos, hogy az adatokat a TCP azonnal elküldje. Tegyük fel például, hogy a felhasználó egy interaktív játékban frissítések sorozatát szeretné elküldeni. Rendkívül fontos, hogy a frissítéseket a TCP azonnal továbbítsa, és ne pufferelje addig, amíg egy gyűjtemény lesz belőle. Az azonnali adatküldésre a TCP a csomagban található PUSH jelzőbit fogalmát használja. Az eredeti szándék szerint az alkalmazások a PUSH bit beállításával közölték volna a TCP-megvalósításokkal, hogy az adatot azonnal el kell küldeni. Az alkalmazások azonban nem tudják közvetlenül állítani a PUSH bitet, amikor adatot küldenek. Ehelyett különböző operációs rendszerek különböző lehetőségeket fejlesztettek ki a küldés felgyorsítására (például Windows- és Linux-rendszereken a TCP_NODELAY lehetőség).

Az internet régészeinek megemlítünk egy érdekes TCP-szolgáltatást, amely bár a protokollban megmaradt, de ritkán használják: a **sürgős adatot** (**urgent data**). Amikor egy alkalmazásnak sürgős adata van, amit azonnal fel kell dolgozni, például amikor egy interaktív felhasználó CTRL-C-t üt egy már megkezdett távoli számítás megszakítására, a küldőalkalmazás további vezérlőinformációt ad az adatfolyamhoz, és *sürgős* jelzéssel

átadja a TCP-nek. Ennek hatására a TCP abbahagyja az adatok egybegyűjtését az adott összeköttetésre, és azonnal továbbítja az eddig összegyűlt adatot.

Amikor a sürgős adat célba ér, a vevőalkalmazás végrehajtása megszakad (UNIX-terminológiában *signal*-t kap), abbahagyja amit éppen csinált, és beolvassa az adatfolyamot, hogy megtalálja a sürgős adatot. A sürgős adat vége jelezve van, így az alkalmazás tudja, hogy meddig tart. A sürgős adat kezdetén ezzel ellentétben nincs jelzés, azt az alkalmazásnak kell megtalálnia.

Ez a megoldás csak egy kezdetleges jelzési rendszert biztosít, és minden más feladatot az alkalmazásra hagy. A sürgős adat potenciálisan hasznos, de nem talált alkalmazásra, így idővel elavulttá vált. Használata nem ajánlott, mivel a megvalósítások különböznek, és az alkalmazásra hagyják a jelzési módszereik kezelését. Talán a jövőbeli szállítási protokollok jobb jelzési rendszerrel lesznek felfegyverkezve.

6.5.3. A TCP-protokoll

Ebben a szakaszban a TCP-protokollt tekintjük át nagy vonalakban. A következő szakaszban először a protokoll fejrészét mezőnként vesszük vizsgálat alá.

A TCP rendelkezik egy olyan, kulcsfontosságú tulajdonsággal, amely az egész protokoll kialakítását befolyásolja: a TCP-összeköttetéseken minden bájt rendelkezik egy 32 bites sorszámmal. Amikor az internet története elkezdődött, az útválasztókat összekötő vonalak általában 56 kb/s-os bérelt vonalak voltak, ezért egy teljes sebességgel adó hoszt esetében is több mint egy hét eltelt a sorszámozás két újrakezdése között. Később látni fogjuk, hogy a mai hálózati sebességek mellett a hosztok a sorszámokat ijesztő sebességgel fogyaszthatják. A nyugták és a csúszóablakok ettől független 32 bites sorszámokat használnak, amint az a következőkből is kiderül.

A küldő és a vevő TCP-entitások az adatokat szegmensekben viszik át egymás között. A **TCP-szegmensek** (**TCP segment**) egy rögzítetten 20 bájtos fejrészből (és egy nem kötelező, opcionális részből), valamint 0 vagy több adatbájtból állnak. A TCP-szoftver dönti el, hogy a szegmensek mekkorák legyenek. Összegyűjtheti több írási utasítás adatait egyetlen szegmensbe, de egy írás adatait is eloszthatja több szegmensbe. A szegmensek méretének két korlátja van. Egyrészt, minden szegmensnek a hozzá tartozó TCP-fejrésszel együtt bele kell férnie a 65 515 bájtos IP-adatmezőbe. Másrészt, minden hálózaton meghatározzák az úgynevezett **legnagyobb átvihető adategységet** (**Maximum Transfer Unit, MTU**). Minden szegmensnek bele kell férnie mind a küldő-, mind a vevőoldalon[2] az MTU által megszabott keretekbe, hogy egyben, további darabolás nélkül lehessen továbbítani. A gyakorlatban az MTU, vagyis a szegmensméret felső korlátja általában 1500 bájt (az Ethernet-adatmező mérete).

Továbbra is lehetséges azonban a TCP-szegmenseket hordozó IP-csomagok darabolása olyan hálózati útvonalakon, ahol valamely összeköttetés MTU-értéke kisebb. Ebben a helyzetben a teljesítőképesség romlik, és egyéb más problémák is felmerülnek [Kent és Mogul, 1987]. Ehelyett a modern TCP-megvalósítások az **útvonal MTU-meghatározását** (**path**

2 A szegmensek mérete esetén nemcsak a küldő- és a vevőoldalon, hanem a teljes útvonalon lévő legkisebb MTU-t kell figyelembe venni. *(A fordító megjegyzése)*

MTU discovery) alkalmazzák, amelynek a leírását az RFC 1191 tartalmazza, és amelyet az 5.5.5. szakaszban tárgyaltunk. Ez a módszer ICMP-hibaüzeneteket használ, hogy megtalálja az útvonalon található legkisebb MTU-val rendelkező összeköttetést. A TCP végül ehhez az értékhez igazítja a küldendő szegmensek méretét, hogy elkerülje a feldarabolást.

A TCP-entitások egy csúszóablakos protokoll-változatot használnak dinamikus ablakmérettel. A küldő minden szegmens feladásakor egy időzítőt is elindít. Amikor a szegmens megérkezik a célhoz, a vevő TCP-entitás visszaküld egy olyan szegmenst (adatokkal, ha van elküldendő adat, egyébként üresen), amely tartalmazza a maradék ablakméretét, és amelyben a következőnek várt szegmens sorszámával visszaigazolja az adást. Ha a küldőoldalon az időzítő a nyugta vétele előtt lejár, akkor a küldőentitás újra elküldi a szegmenst.

Annak ellenére, hogy ez a protokoll egyszerűnek hangzik, van néhány, esetenként apró buktatója, amelyeket alább fogunk megtárgyalni. A szegmensek érkezhetnek helytelen sorrendben, így előfordulhat, hogy a 3072–4095 sorszámú bájtok megérkeznek, de a vevő nem nyugtázhatja azokat, mivel a 2048–3071 sorszámú bájtok még nem érkeztek meg. Előfordulhat, hogy a szegmensek olyan hosszú ideig utaznak, hogy a küldő időzítője közben lejár és újra elküldi azokat. Előfordulhat az is, hogy az újraküldés során az adó nem ugyanazokat a bájttartományokat küldi el az egyes szegmensekben, ezért gondos nyilvántartásra van szükség ahhoz, hogy a TCP-entitás nyomon követhesse az adott pillanatig helyesen vett bájtokat. Ez azért kivitelezhető megoldás, mert a folyam minden bájtja egyedi eltolással (offset) rendelkezik.

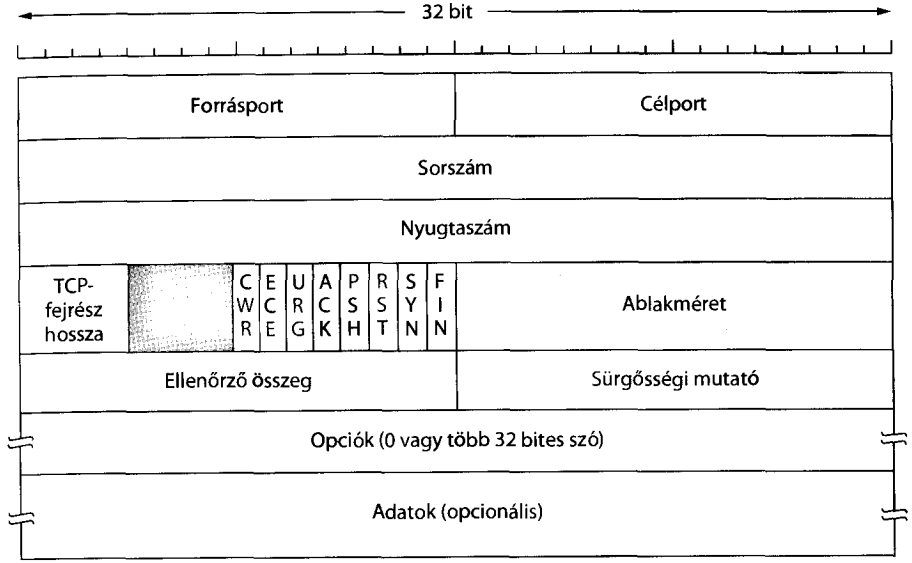
A TCP-nek felkészülten kell szembenéznie ezekkel a problémákkal, és hatékony módon kell megoldania azokat. A tervezők jelentős mennyiségű erőfeszítést fordítottak arra, hogy a TCP-folyamok teljesítménye még hálózati gondok esetén is optimális legyen. A későbbiekben ismertetni fogunk néhány olyan algoritmust, amelyet sok TCP-megvalósítás használ.

6.5.4. A TCP-szegmens fejrésze

A 6.36. ábrán láthatjuk a TCP-szegmens felépítését. Minden szegmens egy fix kiosztású 20 bájtos fejrésszel kezdődik, amit fejrészopciók követhetnek. Ezek után – ha van – legfeljebb 65 535 – 20 – 20 = 65 495 bájt adat állhat. A kivonandók közül az első 20 bájt az IP-, a második 20 bájt a TCP-fejrészt jelenti. Az adatot nem tartalmazó szegmensek érvényesek, általában nyugtázásra és vezérlőszegmensként használják azokat.

Elemezzük most mezőről mezőre a TCP-fejrészt! A *Forrásport* és *Célport* mezők azonosítják az összeköttetés helyi végpontjait. A TCP-portszám és a hoszt IP-címe együtt egy egyedi 48 bites azonosítót jelent a végpontok megkülönböztetésére. A *forrás* és a *cél végpontok* együtt azonosítják az összeköttetéseket. Ezt az összeköttetés-azonosítót **5-ösnek** (**5 tuple**) hívják, mert 5-féle információt tartalmaz: a protokollt (TCP), a forrás IP-címét, a forrás portszámát, valamint a címzett IP-címét és a címzett portszámát.

A *Sorszám (sequence number)* és *Nyugtaszám (acknowledgement number)* mezők szerepe szokásos. Jegyezzük meg, hogy az utóbbi a következő várt bájt sorszámát tartalmazza, nem az utolsó rendben beérkezett bájtét. Mivel a nyugta egyetlen számmal minden vett adatot nyugtáz, ezért **halmozott nyugtáról** (**cumulative acknowledgement**)



6.36. ábra. *A TCP-fejrész*

van szó. Elveszett adat esetén nem léptetik. Mindkét mező 32 bit széles, mivel a TCP-folyamban minden adatbájt sorszámot visel.

A *TCP-fejrész hossza (TCP header length)* mondja meg, hány 32 bites szóból áll a TCP-fejrész. Ez az információ azért szükséges, mert a fejrész mérete az *Opciók (options)* mező változó hossza miatt szintén változó. Tulajdonképpen ez a mező jelzi az adat kezdetét (32 bites szavakban mérve) a szegmensen belül, de mivel ez egyben a fejrész szavakban mért hossza is, a végeredmény ugyanaz.

Ezután egy használaton kívüli 4 bites mező következik. A TCP jól átgondolt tervezésére szolgál tanúbizonyságul, hogy ezek a bitek 30 éve változatlanul használaton kívül vannak (és az eredeti 6-ból csupán 2-t használtak fel). Kevésbé jól sikerült protokollokban már fölhasználták volna az eredeti változat hibáinak kiküszöbölésére.

Ezt nyolc egybites mező követi. A *CWR* és *ECE* mezőket torlódás jelzésére használják, amikor az RFC 3168-ban specifikált ECN-t (Explicit Congestion Notification – explicit torlódásjelzés) alkalmazzák. Ha a vevő a hálózattól torlódási jelzést kap, az *ECE* bittel *ECN-Echo* jelzést küld a TCP-küldő számára, ami azt jelenti, hogy a küldőnek le kell lassítania. A *CWR* bittel a TCP-küldő a TCP-vevő felé jelzi a *Torlódási ablak csökkentve* (*Congestion Window Reduced*) eseményt, így a vevő tudni fogja, hogy az adó lelassított, és megszüntetheti az *ECH-Echo* küldését. A TCP torlódáskezelésén belül az ECN szerepét a 6.5.10. szakaszban mutatjuk be.

Az *URG* bit értéke 1, ha *Sürgősségi mutatót* használt. A *Sürgősségi mutató* a sürgős adatnak bájtban mért eltolt helyét jelzi a jelenlegi bájtsorszámhoz viszonyítva. Ez a mechanizmus a megszakítás üzeneteket helyettesíti. Mint korábban említettük, ez a végsőkig leegyszerűsített módszer lehetőséget ad a küldőnek, hogy jelzést küldjön a vevő felé anélkül, hogy a TCP-nek külön megszakításokkal kelljen foglalkoznia, de ritkán használják.

Az *ACK* bit 1 értéke jelzi a *Nyugtaszám* mező érvényességét. A *Nyugtaszám* mező majdnem minden csomag esetén érvényes. Ha *ACK* = 0, a szegmens nem tartalmaz nyugtát, tehát a *Nyugtaszám* mező figyelmen kívül hagyható.

A *PSH* bit jelzi a késedelem nélküli adat továbbítását (PUSH). Ez egyben a vevő felé is udvarias kérést jelent: ne pufferelje a vett adatot (amit amúgy hatékonysági okokból megtehetne), hanem azonnal továbbítsa az alkalmazás felé.

Az *RST* bit egy hoszt összeomlása, vagy más okból összezavart összeköttetés helyreállítására szolgál. Ezenkívül érvénytelen szegmens elutasítására és összeköttetés létesítésének megtagadására is használják. Rendszerint ha valaki *RST* = 1 értéket viselő szegmenst kap, akkor felkészülhet valamilyen probléma megoldására.

A *SYN* bit összeköttetés létesítésére szolgál. Az összeköttetés-kérésben *SYN* = 1 és *ACK* = 0 jelzi, hogy ráültetett *Nyugtaszám* mezőt nem használnak. Az összeköttetés-kérésre adott válaszban már van nyugta, így abban *SYN* = 1 és *ACK* = 1. Lényegében a *SYN* bit jelzi a CONNECTION REQUEST és CONNECTION ACCEPTED üzeneteket, melyeket az *ACK* bit különbözteti meg egymástól.

A *FIN* bit szolgál egy összeköttetés bontására. Jelzi, hogy a küldőnek nincs több továbbítandó adata. Az összeköttetés bontása után azonban még korlátlan ideig folytathatja az adatok vételét. Mind a *SYN*, mind a *FIN* szegmensek rendelkeznek sorszámokkal, így garantált a helyes sorrendben történő feldolgozás.

A TCP forgalomszabályozása változó méretű csúszóablakkal történik. Az *Ablakméret* mező határozza meg, hogy a *Nyugtaszám* mező által mutatott bájttal kezdődően hány bájtot lehet elküldeni. Az *Ablakméret* 0 értéke is érvényes, és azt jelzi, hogy a *Nyugtaszám* bájtnál eggyel kisebb sorszámú bájtok mind rendben megérkeztek, viszont a vevőnek nagy szüksége van egy kis pihenésre és köszöni szépen, több adatot jelenleg nem kér. Később azonos *Nyugtaszám* értékkel és nem nulla *Ablakméret* mezővel ellátott szegmenssel engedélyezni lehet az adatok küldését.

A 3. fejezet protokolljaiban a vett keretek nyugtázása és új keretek küldésének engedélyezése szorosan összefonódott egymással. Ez annak a következménye, hogy az egyes protokollokban használt ablakok mérete rögzített volt. A TCP-ben a nyugtázás teljesen szétválik a további adatküldés engedélyezésétől. A vevő tulajdonképpen azt is mondhatja, hogy „*k*-ig megkaptam a bájtokat, de most egyelőre nem kérek többet". Ez a szétválasztás (amely gyakorlatilag egy változó méretű ablakot jelent) megnöveli a rendszer rugalmasságát, ezért később részletes vizsgálat alá vesszük.

A nagyobb megbízhatóság érdekében van még egy *Ellenőrző összeg* is a fejrészben. Ez ellenőrzi a fejrész, az adat és az álfejrész épségét, pontosan úgy, ahogy az UDP-nél láttuk. Az egyetlen kivétel, hogy a protokollazonosító a TCP esetén 6, és az ellenőrző összeg kötelező.

Az *Opciók* mezőt a szabályos fejrészben nem szereplő lehetőségek megvalósítására tervezték. Sok opciót definiáltak, és néhányat gyakran használnak. Az *Opciók* mező változó hosszúságú, de mindig 32 bit többszöröse, szükség esetén nullákkal kitöltve. Maximális mérete 40 bájt lehet, elfoglalva a legnagyobb TCP-fejrészt, ami lehetséges. Némely opciót összeköttetés felépítésekor küldenek, hogy egyezkedjenek, vagy információt közöljenek a képességekről a távoli féllel. Más opciókat az összeköttetés élettartama alatt minden továbbított csomag tartalmaz. Minden opció TLV (Type-Length-Value – típus-hossz-érték) kódolású.

Egy széles körben használt lehetőség lehetővé teszi a hosztoknak, hogy meghatározzák az **MSS**-t (**Maximum Segment Size – legnagyobb szegmensméret**), amit hajlandók elfogadni. Nagy szegmensek használata hatékonyabb a kicsik alkalmazásánál, hiszen a 20 bájtos fejrész több adathoz tartozik, viszont kis hosztok esetleg képtelenek nagyon nagy szegmensek kezelésére. Összeköttetés létesítésekor minden hoszt megadhatja a saját maximumát, és tudomást szerezhet a partnere maximum értékéről. Ha egy hoszt nem használja ezt a lehetőséget, az alapértelmezés 536 bájtos adat mező. Minden internetre csatlakozó hosztnak el kell fogadnia 536 + 20 = 556 bájt hosszúságú szegmenseket, viszont a két irányban nem kell azonosnak lenni a maximális hosszaknak.

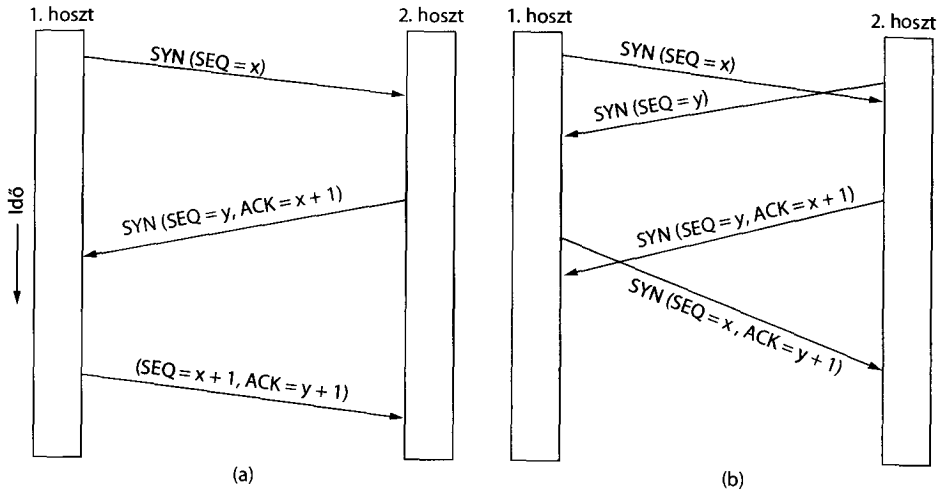
Nagy sávszélességgel vagy nagy késleltetéssel, esetleg mindkettővel rendelkező vonalak számára gyakran problémát jelent a 16 bitnek megfelelő 64 KB ablakméret. Például egy OC-12 vonalon (durván 600 Mbit/s) kevesebb mint 1 ms ideig tart egy 64 KB-os tele ablak továbbítása. Ha a teljes átviteli késleltetés 50 ms (tipikus kontinensek közötti fényvezető szálakra), a küldő a várakozási idő 98 százalékában tétlenül várakozik a nyugtára. Nagyobb ablakméret használata esetén a küldő tovább pumpálhatná az adatot. A **skálatényező** (**window scale**) opció használata lehetővé teszi a küldő és a fogadó számára, hogy összeköttetés létesítésekor megegyezzenek egy ablak-skálatényezőben. Ez a szám mindkét oldalon lehetővé teszi az *Ablakméret* mező legfeljebb 14 bites balra történő eltolását, így legfeljebb 2^{30} bájtos ablakok használatát. A legtöbb TCP-implementáció támogatja ezt a lehetőséget.

Az **időbélyeg** (**timestamp**) opció egy időbélyeget hordoz, melyet a küldő helyez a csomagba és a vevő visszaküldi. Minden csomagban megtalálható, hiszen használatáról az összeköttetés létesítése során megállapodnak, és a körülfordulási idő mintáinak számítására használják, hogy megbecsüljék, mikor veszett el egy csomag. A 32 bites sorszám logikai kiegészítésére is használják, ugyanis a gyors összeköttetéseken a sorszámok hamar átfordulnak, félreértést okozva az új és régi sorszámok között. A **PAWS** (**Protection Against Wrapped Sequence numbers** – átfordult sorszámok elleni védelem) módszer a probléma megoldása érdekében eldob minden olyan szegmenst, amely régi időbélyeggel érkezett.

Végül a **SACK** (**Selective ACKnowledgement – szelektív nyugtázás**) opció lehetővé teszi a vevőnek, hogy a küldőt tájékoztassa a vett sorszámokról. Ez az opció kiegészíti a *Nyugtaszám* mezőt, és csomagvesztés után használják abban az esetben, ha további adatok (vagy másolatok) viszont érkeztek. A további adatokat a fejrész *Nyugtaszám* mezeje nem mutatja, hiszen az csak sorrendben a következő, várt bájtokat mutatja. A SACK segítségével a küldő közvetlenül értesül arról, hogy a vevő milyen adatokat kapott, és így meghatározhatja, melyeket kell újraküldeni. A SACK-ot az RFC 2108 és az RFC 2883 definiálja, és egyre gyakrabban alkalmazzák. A SACK használatát a torlódáskezeléssel kapcsolatban a 6.5.10. szakaszban tárgyaljuk.

6.5.5. TCP-összeköttetés létesítése

Az összeköttetések létesítésére a 6.2.2. szakaszban tárgyalt „háromutas kézfogás" technikáját alkalmazzák a TCP-ben. Az összeköttetés létesítéséhez az egyik fél, nevezzük szervernek, passzívan várakozik a bejövő kérésekre a LISTEN és ACCEPT primitívek végrehajtásával. Ehhez megjelölhet egy adott forrást, vagy nem jelöl ki senkit.



6.37. ábra. *(a) TCP-összeköttetés létesítése normális esetben. (b) Egyidejű összeköttetés létesítése mindkét oldalon*

A másik oldal, melyet kliensnek hívunk, végrehajtja a CONNECT primitívet, melynek hívásakor rögzíti az IP-címet, a használni kívánt port számát, az általa megengedett maximális TCP-szegmens méretét és esetleg felhasználói adatot (például jelszót) is átad. A CONNECT primitív elküld egy TCP-szegmenst *SYN*=1 és *ACK*=0 értékkel, majd választ vár.

Amikor a szegmens megérkezik a rendeltetési helyre, az ottani TCP-entitás ellenőrzi, hogy létezik-e egy folyamat, amely a célport mezőben meghatározott porton végrehajtotta a LISTEN primitívet. Ha nem, *RST*=1 válasszal elutasítja az összeköttetés-kérést.

Ha valamilyen folyamat figyeli a megadott portot, akkor az megkapja a beérkező TCP-szegmenst, és rajta múlik, hogy elfogadja-e vagy visszautasítja az összeköttetést. Ha elfogadja, egy nyugtázó szegmenst küld vissza. A 6.37.(a) ábrán láthatjuk az elküldött TCP-szegmensek sorozatát normális esetben. Figyeljük meg, hogy a *SYN* szegmens egy bájtnak megfelelő sorszámot fogyaszt, így egyértelműen nyugtázható.

Abban az esetben, ha a két hoszt egyszerre próbál összeköttetést létesíteni ugyanazon két csatlakozó (socket) között, a 6.37.(b) ábrán látható eseménysorozat alakul ki. Ennek eredményeképpen csak egyetlen összeköttetés jön létre, nem kettő, mivel az összeköttetéseket végpontjaik azonosítják. Ha a művelet során létrejövő első összeköttetés azonosítói (x, y), és a második is ilyen azonosítóval születik meg, csak egyetlen (x, y) azonosítójú bejegyzés keletkezik a táblázatban.

Emlékezzünk vissza, hogy az összeköttetés kezdő sorszámát minden hosztnak úgy kell megválasztania, hogy az lassan körbejáró legyen, nem pedig állandó, például 0. Ezt a szabályt betartva lehet védekezni a késleltetett kettőzött csomagok ellen, ahogy azt a 6.2.2. szakaszban leírtuk. Eredetileg ezt egy óraalapú megoldással valósították meg, ahol az óra 4 mikromásodpercenként lépett előre.

A „háromutas kézfogás"-nak azonban van egy sérülékenysége, ugyanis a figyelő folyamatnak emlékeznie kell a saját a sorszámára mielőtt visszaküldené a saját SYN-szegmensét. Ez azt jelenti, hogy egy rosszindulatú küldő lekötheti egy hoszt erőforrásait

azáltal, hogy folyamatosan SYN-szegmenseket küld, és nem fejezi be az összeköttetés felépítését. Ezt a támadást **SYN-elárasztásnak (SYN flood)** nevezzük, és az 1990-es években sok webszervert megbénított.

Egy védekezés az ilyen támadások ellen a **SYN-sütik (SYN cookies)** használata. Ahelyett, hogy a hoszt megjegyezné a sorszámot, kriptográfiai módszerrel választ sorszámot, majd a soron következő szegmensben elküldi, és a sorszámot elfelejti. Ha a „háromutas kézfogás"-nál a válasz megjött, akkor a hoszt visszakapja a sorszám eggyel növelt értékét. Ezután a hoszt a helyes sorszámot újra legenerálja ugyanazzal a kriptográfiai módszerrel, feltéve, hogy a bemeneti paramétereket ismeri, amelyek például a másik hoszt IP-címe és portszáma, valamint egy helyi titkos kulcs. Ez az eljárás lehetővé teszi, hogy a hoszt leellenőrizze a nyugtázott sorszám helyességét anélkül, hogy a sorszámra külön emlékeznie kellene. Azért pár hiányosság itt is van, például a TCP-opciókat nem tudja kezelni, ezért a SYN-sütiket csak olyan hosztok esetén célszerű használni, amelyek SYN-elárasztásnak vannak kitéve. Mindazonáltal ez egy érdekes csavarás az összeköttetés létesítése területén. További információkhoz lásd az RFC 4987-et, valamint Lemon [2002] művét.

6.5.6. TCP-összeköttetés lebontása

Bár a TCP-összeköttetések duplexek, hogy megérthessük az összeköttetések bontásának módját, legjobb két szimplex összeköttetésnek tekinteni azokat. Mindkét szimplex összeköttetés a másiktól függetlenül lebontható. Egy összeköttetés bontásához bármelyik fél küldhet egy *FIN*=1 értékű TCP-szegmenst, amivel jelzi, hogy nem szándékozik több adatot küldeni. Amint a *FIN* nyugtája megérkezik, ez az irány lezárul. A másik irányban azonban ettől függetlenül korlátlanul folyhat adatátvitel. Amikor mindkét irányt lezárták, az összeköttetés befejeződött. Normális esetben egy összeköttetés bontásához négy TCP-szegmens szükséges, egy *FIN* és egy *ACK* mindkét irányban. Az első *ACK* és a második *FIN* viszont elhelyezhető ugyanabban a szegmensben is, így összesen csak háromra van szükség.

Hasonlóan a telefonhívásokhoz, ahol mindkét ember egyszerre köszön el és teszi le a kagylót, itt is küldhet mindkét fél egy időben *FIN* szegmenst. Ezeket a szokásos módon nyugtázzák, majd az összeköttetés befejeződik. Valójában nincs lényeges eltérés az egyszerre vagy egymás után történő összeköttetés-bontások között.

A „két hadsereg" probléma elkerülésére időzítőket használnak. Ha egy *FIN*-re nem érezik válasz két csomag-élettartamnyi idő alatt, a *FIN* küldője bontja az összeköttetést. A másik fél végül észreveszi, hogy már senki sem figyel rá, az általa indított időzítő is lejár. Bár ez a megoldás nem tökéletes, az tény, hogy tökéletes megoldás elméletben sem létezhet, ezért ennek elégnek kell lenni. A gyakorlatban ritkán merülnek föl problémák.

6.5.7. A TCP összeköttetés-kezelésének modellje

Összeköttetések létesítését és bontását véges automatával is modellezhetjük. A gép 11 állapotát a 6.38. ábrán láthatjuk. Minden állapotban az események bizonyos halmaza érvényes. Ha érvényes esemény történik, lejátszódik valamilyen tevékenység, más esemény hatására hibajelzés keletkezik.

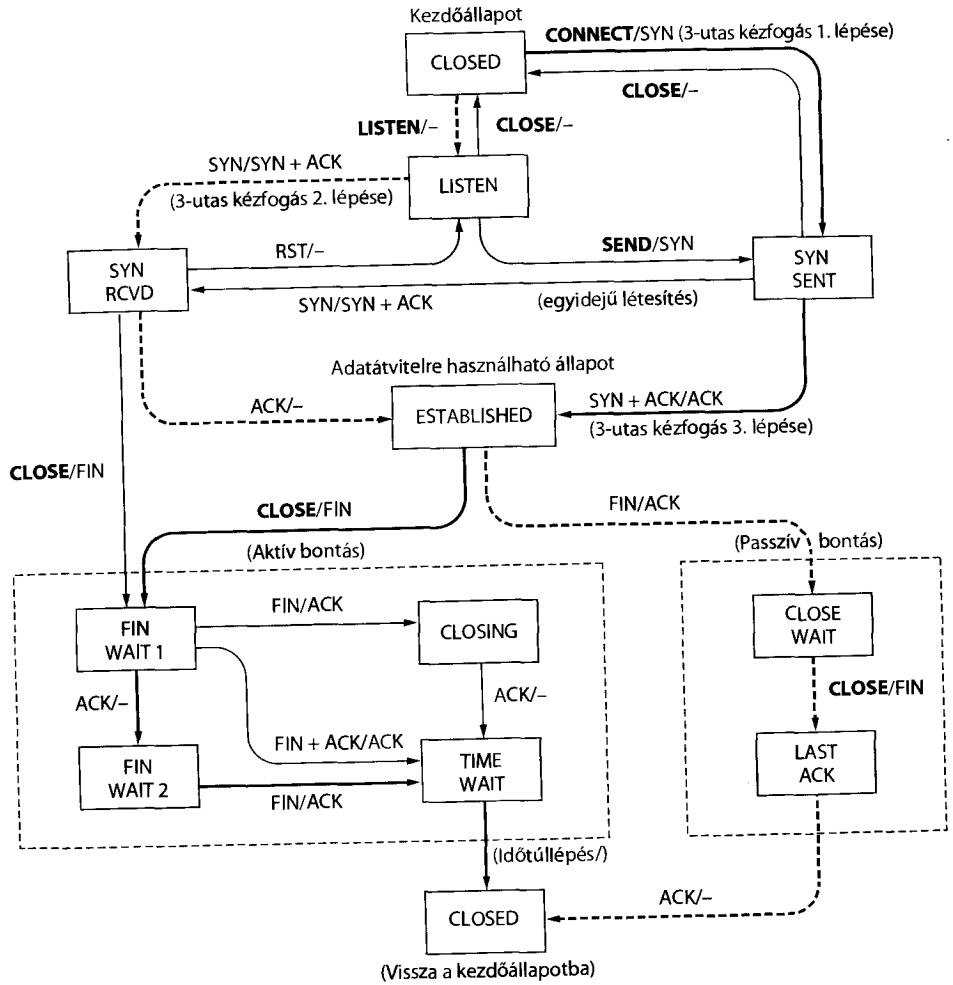
Állapot	Leírás
CLOSED	Nincs aktív vagy függő összeköttetés
LISTEN	A szerver egy hívás beérkezésére vár
SYN RCVD	Összeköttetés-kérés érkezett, ACK-ra vár
SYN SENT	Az alkalmazás összeköttetés-létesítést kezdeményezett
ESTABLISHED	Normális adatátviteli állapot
FIN WAIT 1	Az alkalmazás bejelentette, hogy végzett teendőivel
FIN WAIT 2	A másik fél beleegyezett az összeköttetés bontásába
TIME WAIT	Vár, amíg az összes csomag ki nem hal
CLOSING	Mindkét fél egyszerre próbálta bontani az összeköttetést
CLOSE WAIT	A másik fél bontást kezdeményezett
LAST ACK	Vár, amíg az összes csomag ki nem hal

6.38. ábra. *A TCP összeköttetés-kezelő véges állapotú gép állapotai*

Minden összeköttetés *CLOSED* állapotból indul. Akkor hagyja el ezt az állapotot, amikor passzív módon (LISTEN) vagy aktív módon (CONNECT) összeköttetést próbál létesíteni. Ha a másik fél az ellenkező primitívet hajtja végre, az összeköttetés felépül és *ESTABLISHED* állapotot vesz fel. Bármelyik fél kezdeményezheti az összeköttetés bontását. Amikor ez lezajlott, az állapot ismét *CLOSED* lesz.

Magát a véges állapotú gépet a 6.39. ábrán láthatjuk. Vastag vonalak mutatják egy aktív kliens összeköttetés-létesítési folyamatát egy passzív szerverrel, a folytonos vonalak a kliensre, a szaggatottak a szerverre vonatkoznak. A vékony vonalak váratlan eseményeket jeleznek. A 6.39. ábrán az összes nyílhoz tartozik egy *esemény/tevékenység* pár. Az esemény lehet egy felhasználó által végrehajtott rendszerhívás (CONNECT, LISTEN, SEND vagy CLOSE), egy szegmens érkezése (*SYN*, *FIN*, *ACK* vagy *RST*), vagy egy esetben a kétszeres maximális csomagélettartamra beállított időzítő lejárása. A tevékenység lehet vagy egy vezérlőszegmens (*SYN*, *FIN* vagy *RST*) elküldése, vagy semmi (ezt „–" jelöli). A megjegyzések zárójelben láthatók.

Az ábrát úgy a legkönnyebb megérteni, ha először végigkövetjük a kliens állapotátmeneteit (a vastag folytonos vonalakat), majd a szerver állapotátmeneteit (a vastag szaggatott vonalakat) is sorra végignézzük. Amikor az egyik alkalmazási program a kliensgépen kiad egy CONNECT kérést, a helyi TCP-entitás létrehoz egy új összeköttetés-rekordot, amelyet *SYN SENT* állapotúnak jelöl meg, és elküld egy *SYN* szegmenst. Fontos megjegyezni, hogy egyszerre több összeköttetés felépítése, illetve fenntartása lehet folyamatban, több különböző alkalmazás részére. Az állapotok emiatt külön-külön tartoznak az egyes összeköttetésekhez és a TCP-entitás az összeköttetés-rekordokban tartja nyilván azokat. Amikor a *SYN+ACK* megérkezik, akkor a TCP elküldi a „háromutas kézfogás" utolsó *ACK* szegmensét és az *ESTABLISHED* állapotba viszi át az összeköttetést. Ezután szabadon lehet adatokat küldeni és fogadni.



6.39. ábra. *A TCP-összeköttetést kezelő véges állapotú gép. A vastag folytonos vonalak a kliens szokásos állapotátmenetei. A vastag szaggatott vonalak a szerver szokásos állapotátmenetei. A vékony vonalak a rendkívüli események. Minden átmenet címkéje az átmenetet kiváltó esemény és az átmenet által okozott tevékenység, „/" jellel elválasztva*

Amikor egy alkalmazás befejezte tevékenységét, végrehajtja a CLOSE primitívet, melynek hatására a helyi TCP-entitás egy *FIN* szegmenst küld, majd az ehhez tartozó nyugtára (*ACK*) vár (szaggatott téglalap *aktív lebontás* megjegyzéssel). Amikor megérkezik az *ACK*, az új állapot *FIN WAIT 2* lesz, és az összeköttetés egyik irányban befejeződik. Amikor a másik fél is bont, egy *FIN* szegmens érkezik, amire a helyi entitás nyugtát küld. Ekkorra mindkét fél lebontotta az összeköttetést, de a TCP még a maximális csomagélettartam kétszereséig várakozik, így még akkor is garantálható, hogy az összeköttetés összes csomagja kihal, ha esetleg egy nyugta elveszett volna. Amikor az időzítő lejár, a TCP törli az összeköttetés bejegyzését.

Most vizsgáljuk meg az összeköttetés kezelését a szerver szemszögéből. A szerver LISTEN hívást ad ki és csöndben figyeli, hogy ki bukkan föl. Amikor beérkezik egy *SYN* szegmens, nyugtázza, és *SYN RCVD* állapotba lép. Amint a szerver *SYN* szegmensére is nyugta érkezik, a „háromutas kézfogás" protokoll véget ér és a szerver *ESTABLISHED* állapotba kerül. Megkezdődhet az adatátvitel.

Ha a kliens végzett tennivalóival, CLOSE hívást ad ki, melynek hatására *FIN* szegmens érkezik a szerverhez (szaggatott vonalas téglalap *passzív lebontás* felirattal). A szerver ezután megszakítást (signal) kap. Amikor a szerver is végrehajtja a CLOSE primitívet, a TCP-entitás *FIN* szegmenst küld a kliensnek. Amint a kliens nyugtája megjelenik, a szerver bontja az összeköttetést és törli a hozzá tartozó bejegyzést.

6.5.8. A TCP-csúszóablak

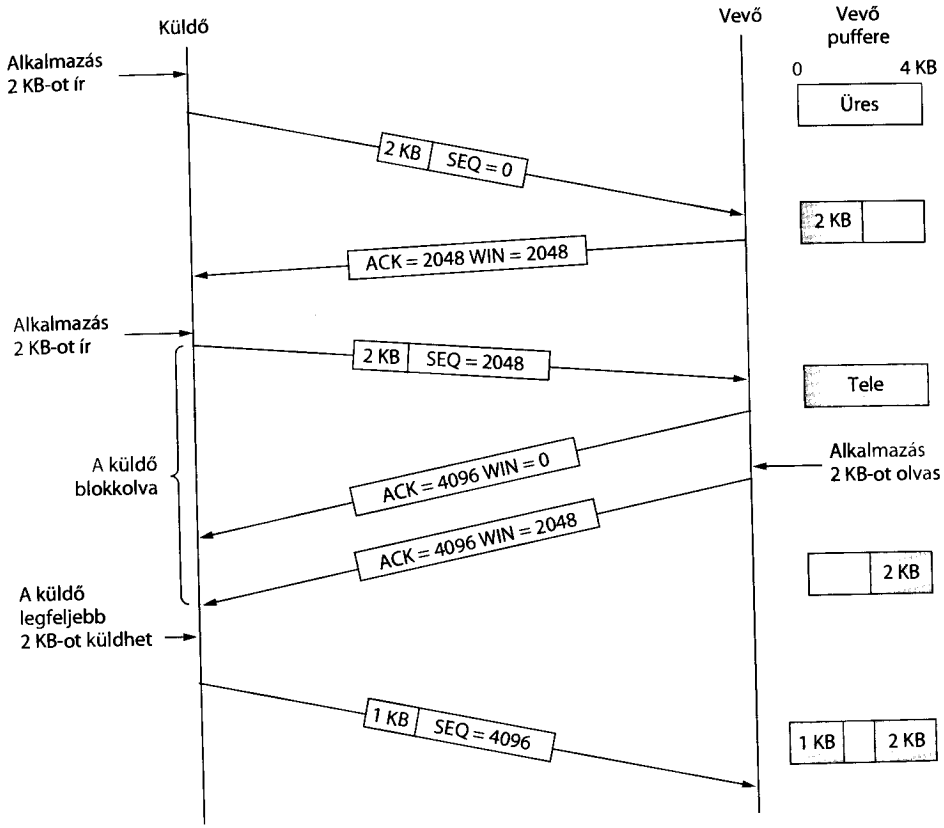
Amint azt már korábban is említettük, a TCP-ben az ablakkezelés különválasztja a helyesen vett szegmensek nyugtázását és a vevő pufferkezelését. Tegyük fel például, hogy a vevő 4096 bájtos pufferrel rendelkezik (6.40. ábra). Ha a küldő egy 2048 bájtos szegmenst küld el, ami rendben meg is érkezik, a vevő nyugtázza a szegmenst. Mivel azonban most csak 2048 bájt szabad pufferterülete van (amíg az alkalmazás ki nem olvassa a lefoglalt pufferterület egy részét vagy egészét) bejelenti, hogy a következő bájttól kezdve 2048 bájtos ablakot használ.

Most a küldő újabb 2048 bájtot továbbít, amit a vevő nyugtáz, továbbá közli, hogy az ablakméret 0 bájt. A küldőnek le kell állnia, amíg a fogadóhoszton futó alkalmazói folyamat el nem távolít valamennyi adatot a pufferből, amikor is a TCP egy nagyobb ablak használatát jelentheti be, és a küldő több adatot küldhet.

Amikor az ablakméret 0 bájt, a küldő normális esetben nem küldhet szegmenst, azonban van két kivétel. Először is a sürgős adatot továbbíthatja, hogy lehetővé tegye például a távoli gépen futó folyamat megszakítását. Másodszor, a küldő elküldhet egy egybájtos szegmenst, kényszerítve a vevőt, hogy közölje vele a következő várt bájt sorszámát és az ablakméretet. Ezt a csomagot **ablaktesztelésnek (window probe)** nevezzük. A TCP-szabvány explicit módon biztosítja ezt a lehetőséget, hogy elkerülje a holtpontot, amennyiben egy ablakméretet közlő szegmens elveszne.

A küldő nem köteles azonnal továbbítani az alkalmazástól kapott adatot. A vevő sem köteles azonnal nyugtázni a beérkezett szegmenseket. Például a 6.40. ábrát tekintve, amikor beérkezett az első 2 KB adat, a TCP, tudva, hogy 4 KB-os ablak áll rendelkezésére, teljesen korrekt módon járna el, ha az adatot újabb 2 KB beérkezéséig a pufferben tárolná, hogy 4 KB rakománnyal tudja továbbítani a szegmenst. Ezt a szabadságot a teljesítőképesség fokozásában lehet kamatoztatni.

Vegyünk egy összeköttetést egy távoli terminállal, például SSH vagy telnet használatával, ami minden billentyűleütésre reagál. Legrosszabb esetben, amikor megérkezik egy karakter a küldő TCP-entitáshoz, az létrehoz egy 21 bájtos szegmenst, amelyet átad az IP-nek, hogy 41 bájtos IP-datagramként továbbítsa. A vevőoldali TCP azonnal visszaküld egy 40 bájtos nyugtát (20 bájtnyi TCP-fejrész és újabb 20 bájtnyi IP-fejrész). Később, mikor a távoli terminál beolvasta a kapott bájtot, a TCP egy bájttal jobbra mozdítja az ablakot, és ezt közli a küldővel is. Ez a csomag szintén 40 bájtos. Végül, mikor a



6.40. ábra. *A TCP ablakkezelése*

távoli terminál feldolgozta a karaktert, egy 41 bájtos csomag segítségével megjeleníti a karaktert a helyi képernyőn. Összesen négy szegmens továbbítása, azaz a sávszélesség 162 bájtja szükséges minden egyes begépelt karakterhez. Amikor a sávszélesség az igényeltnél kisebb, nem célszerű így gazdálkodni.

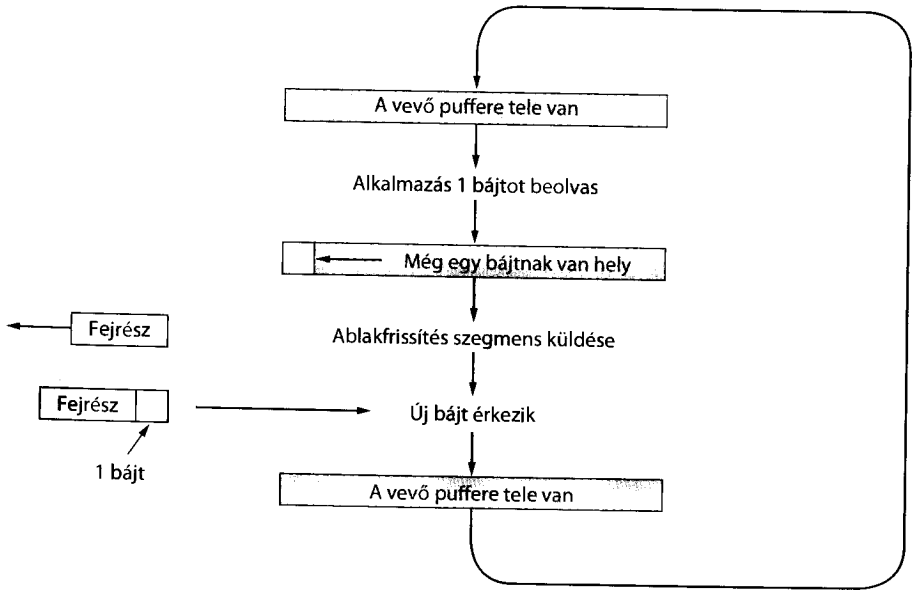
A fenti helyzet optimalizálására több TCP-implementációban alkalmazzák a **késleltetett nyugtázást (delayed acknowledgements)**. Az ötlet az, hogy késleltessük a nyugták és ablakméret-információk elküldését 500 ms-ig azt remélve, hogy egy visszaküldendő adatcsomagba ágyazva ingyen elküldhetjük azokat. Ha feltételezzük, hogy a terminál fél másodpercen belül küld visszajelzést, csak egyetlen 41 bájtos csomagot kell elküldeni a távoli felhasználónak. Az elküldött csomagok száma és a fölhasznált sávszélesség így a felére csökken.

Bár a késleltetett nyugtázás csökkenti a hálózat vevő által okozott terhelését, a küldő a sok rövid csomag (például az egyetlen adatbájtot tartalmazó 41 bájtos csomagok) küldözgetésével még mindig pazarlóan gazdálkodik. Az ennek csökkentésére kidolgozott módszer a **Nagle-féle algoritmus** [Nagle, 1984] néven ismert. Nagle ötlete egyszerű: amikor a küldőhöz kis darabokban érkezik az adat, csak az elsőt továbbítja, a többit addig puffereli, amíg az elküldött darab nyugtája meg nem érkezik. Ezután a pufferben

tárolt összes adatot egyetlen TCP-szegmensben elküldi, és újra kezdi a pufferelést, amíg a szegmens nyugtája meg nem érkezett. Így tehát csak egyetlen kis csomag lehet kinn egy időben. Ha a körülfordulási idő alatt az alkalmazás sok kisméretű adatot küld, akkor Nagle algoritmusa a sok kis darabot egy szegmensbe rakja, jelentősen csökkentve a használt sávszélességet. Az algoritmus továbbá kimondja, hogy egy új szegmenst kell küldeni, ha elég adat csordogált be, hogy megtöltsön egy maximális méretű szegmenst.

A Nagle-féle algoritmus széles körben elterjedt a TCP-implementációkban, de némely esetben szerencsésebb kikapcsolni. Például az interneten futtatott interaktív játékok esetén a játékos tipikusan rövid, frissítő csomagok tömkelegét szeretné küldeni minél hamarabb. A frissítéseket bevárva, majd löketszerűen továbbítva a játékélményt erősen lerontaná, és sok elégedetlen felhasználót eredményezne. Egy még alapvetőbb probléma is felbukkanhat, ha Nagle algoritmusát a késleltetett nyugtákkal használjuk, ugyanis ideiglenes holtpontok alakulhatnak ki: a vevő az adatokra vár, hogy a nyugtát ráültethesse, a küldő pedig a nyugtára vár, hogy több adatot küldjön. Ez az összeférhetetlenség késleltetheti a weboldalak letöltését. A fenti problémák miatt Nagle algoritmusa kikapcsolható (ezt *TCP_NODELAY* opciónak nevezik). Mogul és Minshall [2001] tárgyalja a problémát, és más megoldásokat is ad.

Egy másik probléma, ami le tudja rontani a TCP teljesítőképességét, a **buta ablak jelenség** (**silly window syndrome**) [Clark, 1982]. Ez a probléma akkor merül föl, amikor a küldő TCP-entitás nagy blokkokban kapja az adatokat, de a fogadóoldalon futó interaktív alkalmazás bájtonként olvassa be. A probléma könnyebb megértésére vegyük szemügyre a 6.41. ábrát. Kezdetben a vevő TCP-puffere tele van, és a küldő ezzel tisztában van (tehát az ablakmérete 0). Ezután az interaktív alkalmazás beolvas egy karaktert a TCP-adatfolyamról. Megörül ennek a fogadó TCP-entitás, elküldi az új ablakméretet



6.41. ábra. *A buta ablak jelenség*

a küldőnek, mondván, hogy minden rendben, egy bájtot küldhet. A küldő teljesíti ezt a kívánságot is, egy bájtot elküld. A puffer ismét tele lesz, így a vevő nyugtázza a bájt érkezését, de egyidejűleg közli, hogy az ablak mérete 0. Ez a viselkedés így mehet örökké.

Clark megoldása szerint nem szabad megengedni a vevőnek, hogy 1 bájt változásra ablakméret-frissítést küldjön. Ehelyett mindaddig várakoztatni kell, amíg elegendő hely szabaddá nem válik, és inkább azt kell a küldővel közölni. Pontosabban a fogadó nem küldhet addig ablakméret-információt, amíg az összeköttetés létesítésekor bejelentett maximális szegmensméretet nem tudja kezelni, vagy félig ki nem ürült a puffer. A két korlát közül a szabad hely méretének a kisebbet kell elérnie. Ezenkívül a küldő is segíthet azzal, hogy nem küld apró szegmenseket. Ehelyett addig el kell halasztani a továbbítást, amíg elég hely össze nem gyűlt az ablakban, hogy egy teljes szegmenst elküldhessen, vagy legalább olyan hosszút, mint a vevő ablakméretének fele.

A Nagle-féle algoritmus és Clark megoldása a buta ablak problémára kiegészítik egymást. Nagle olyan problémát próbált megoldani, melyben a küldőalkalmazás bájtonként adta át az adatokat a TCP-nek. A Clark által megoldott problémában a vevőalkalmazás kérte bájtonként az adatokat a TCP-től. Mindkét megoldás helyes és képesek együtt működni. Az a cél, hogy a küldő ne adjon apró szegmenseket, és a vevő se kérjen kicsiket.

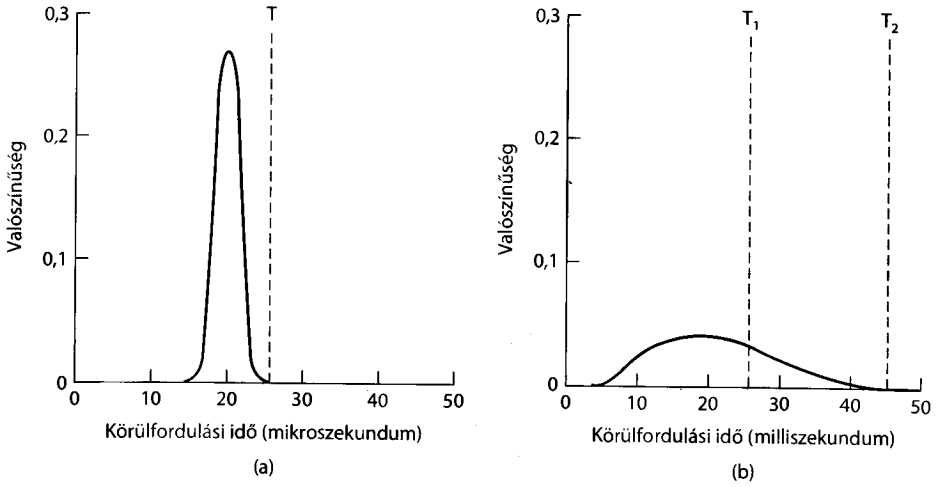
A fogadó TCP-entitás tovább növelheti a teljesítőképességet azon túl, hogy csak nagyobb egységekben frissíti az ablakot. A küldő TCP-entitáshoz hasonlóan a fogadónak is van lehetősége az adatok pufferelésére, így az alkalmazás egy READ hívását addig blokkolhatja, amíg egy nagyobb adatblokkot nem tud átadni. Ezzel csökken a TCP-hívások száma, vele együtt a többletráfordítás (overhead) is. Ez természetesen a válaszidőt is megnöveli, de az állomány átviteléhez hasonló nem interaktív alkalmazások esetében a hatékonyság ellensúlyozhatja az egyes kérések megnövekedett válaszidejét.

Egy másik feladat a vevő számára a rossz sorrendben érkező szegmensek kezelése. A vevő egészen addig fogja pufferelni az adatokat, amíg azokat egyben, helyes sorrendben át nem tudja adni az alkalmazásnak. Ha az alkalmazás pufferelés helyett eldobná a nem soron következő szegmenseket, a pazarláson kívül semmi különös nem történne, hiszen a küldő úgyis újraküldené őket.

Nyugta természetesen csak akkor küldhető, ha a nyugtázott bájtig terjedő összes adat megérkezett. Ezt a megoldást **halmozott nyugtázásnak** (**cumulative acknowledgement**) hívjuk. Ha a vevő a 0, 1, 2, 4, 5, 6 és 7 szegmenseket kapja meg, mindent nyugtázhat a 2. szegmens utolsó bájtjáig (azt is beleértve). Amikor a küldő időzítése lejár, újraküldi a 3. szegmenst. Ha a vevő megtartotta a 4–7. szegmenseket, a 3. szegmens vétele után a 7. szegmens utolsó bájtjáig mindet nyugtázhatja.

6.5.9. A TCP időzítéskezelése

A TCP (elvileg) több időzítőt használ feladata elvégzéséhez. Ezek közül legfontosabb az **RTO** (**Retransmission TimeOut – ismétlési időzítő**). Egy szegmens elküldésekor az ismétlési időzítőt is elindítja. Ha a szegmensre az időzítő lejárta előtt nyugta érkezik, az időzítő leáll. Ha viszont az időzítő még a nyugta beérkezése előtt lejár, a TCP a szegmenst újraküldi (és az időzítőt is újraindítja). Fölmerül a kérdés: milyen hosszú ideig fusson az időzítő?



6.42. ábra. *(a) A nyugtabeérkezési idők sűrűségfüggvénye az adatkapcsolati rétegben. (b) A nyugtabeérkezési idők sűrűségfüggvénye a TCP-ben*

Ez sokkal nehezebb kérdés a szállítási rétegben, mint amilyen az adatkapcsolati rétegben (például a 802.11 esetén) volt. Az adatkapcsolati rétegben a várható késleltetés mikroszekundum nagyságrendű és nagymértékben kiszámítható volt (vagyis kicsi volt a szórása), így az időzítőt úgy lehetett beállítani, hogy kevéssel a nyugta várt megérkezése után járjon le, ahogy az a 6.42.(a) ábrán látható. Az adatkapcsolati rétegben a nyugták (a torlódások hiánya miatt) ritkán késnek, ezért ha egy nyugta nem érkezik meg a várt időn belül, az általában azt jelenti, hogy vagy az adatkeret, vagy a nyugta elveszett.

A TCP ettől gyökeresen eltérő környezettel szembesül. A TCP-nyugták körülfordulási idejének sűrűségfüggvénye sokkal inkább a 6.42.(b) ábrára hasonlít, mint a 6.42.(a) ábrára. A körülfordulási idő nagyobb és a szórása is tágabb. A rendeltetési helyig terjedő körülfordulási időt nehéz megállapítani. Még ha ismert is, az időzítés időtartamáról is nehéz dönteni. Ha az időtartamot túl rövidre állítjuk be, mondjuk a 6.42.(b) ábra T_1 értékére, felesleges újraküldések történnek, az internetet haszontalan csomagok terhelik. Ha túl hosszúra állítjuk (T_2), a teljesítőképesség egy csomag elvesztekor a hosszú újraküldési késleltetés miatt csökken. Ezenkívül a nyugta érkezési idejének átlaga és szórásnégyzete is jelentősen változhat pár másodperc alatt, ha torlódás lép fel vagy szűnik meg.

A megoldás az, ha erősen dinamikus algoritmust használunk, ami a hálózat teljesítőképességének folyamatos mérése alapján állandóan újra beállítja az időintervallumot. A TCP-ben általánosan használt algoritmus Jacobson [1988] nevéhez fűződik, és a következőképpen működik. A TCP minden összeköttetés részére fenntart egy *SRTT*-nek (Smooth Round-Trip Time – csillapított körülfordulási idő) nevezett változót, ami a szóban forgó rendeltetési helyig terjedő körülfordulási idő legjobb jelenlegi becsült értéke. Egy szegmens elküldésekor egy időzítőt is elindít a TCP, hogy egyrészt megmérje, mennyi idő alatt ér vissza a nyugta, másrészt, ha túl sokáig késik, újraküldhesse a csomagot. Ha a nyugta az időzítő lejárása előtt visszaér, a TCP megméri, hogy mennyi ideig tartott *R*. Ezután újraszámolja az *SRTT* értékét a következők szerint:

$$SRTT = \alpha SRTT + (1 - \alpha)R$$

A képletben α egy csillapítási tényező, amely azt határozza meg, hogy a régi értékeket milyen gyorsan felejtsük el. Tipikusan $\alpha = 7/8$. Ez a formula **EWMA (Exponentially Weighted Moving Average – exponenciálisan súlyozott mozgóátlag)** néven ismert, és tulajdonképpen egy aluláteresztő szűrő, ami a mintákban található zajt távolítja el.

Még az *SRTT* jó értékének tudatában sem triviális egy megfelelő ismétlési késleltetés kiválasztása. Korai TCP-implementációkban a $2x$*SRTT* értéket használták, de a tapasztalat azt mutatta, hogy a konstans érték rugalmatlanul viselkedik, nem tudja követni a változásokat, ha a szórásnégyzet megnőtt. Különösen a véletlen (például Poisson) forgalomra építő sorbanállási modellek mutatták ki, hogy ha a terhelés a kapacitáshoz közeledik, akkor a késleltetés megnő, és erősen változó lesz. Ez ahhoz vezet, hogy az újraküldési időzítő lejár, és a csomag másolatát újraküldi, holott az eredeti csomag még a hálózatban tartózkodik. Ráadásul ez éppen nagy terhelés esetében történik, amikor egy további csomag hálózatba juttatása a legrosszabb dolog, ami történhet.

A problémára Jacobson azt javasolta, hogy az ismétlési késleltetés értékét a körülfordulási idő szórásától és a csillapított körülfordulási időtől kell függővé tenni. Ehhez a változtatáshoz egy újabb csillapított változó, az ***RTTVAR* (Round-Trip Time VARiation – körülfordulási idő szórása)** kiszámítását kell elvégezni:

$$RTTVAR = \beta RTTVAR + (1 - \beta)|SRTT - R|$$

Ez ugyanaz az EWMA, amit az előbb láttunk, és tipikusan $\beta = 3/4$. Az újraküldési késleltetés, az *RTO* értékét pedig a következőre kell megválasztani:

$$RTO = SRTT + 4 \times RTTVAR$$

A 4-es szorzótényező választása valamennyire önkényes, a néggyel történő szorzás megvalósítható egyetlen eltolással, és így a csomagok kevesebb mint 1%-a fog beérkezni később, mint a standard szórás négyszerese. Vegyük észre, hogy az *RTTVAR* nem egyezik meg pontosan a standard szórással (valójában inkább egy átlagos szórás), de a gyakorlatban elég közel jár hozzá. Jacobson dolgozata tele van okosabbnál okosabb trükkökkel, hogy az újraküldési késleltetést csupán egész számokat érintő összeadással, kivonással és eltolással kiszámítsuk. A mai modern számítógépek esetén ez a gazdaságosság természetesen nem szükséges, de a TCP azon törekvései közé került, hogy mindenféle eszközön futtatható legyen, napjaink szuperszámítógépeitől egészen apró eszközökig. Jelenleg még senki nem alkalmazta a TCP-t RFID-eszközökben[3], de talán egyszer fogják. Ki tudja?

Az újraküldési késleltetés számításáról, beleértve a változók kezdeti beállítását, az RFC 2988 tartalmaz részletesebb információkat. Ezenkívül az újraküldési időzítés értékét a becslésektől függetlenül 1 másodpercben minimalizálták. Ezt az óvatossági megoldást annak megelőzésére vezették be, hogy elkerüljék a mérések alapján végrehajtott hamis újraküldéseket [Allman és Paxson, 1999].

3 Apró Wi-Fi-eszközökben már használják. *(A fordító megjegyzése)*

A körülfordulási idő, *R* mintáinak gyűjtése során felmerül egy probléma: mi a teendő, ha egy szegmens időzítése lejár, és újraküldik? Amikor beérkezik a nyugta, nem tudható, hogy az első átvitelre vonatkozik, vagy az újabbra. Egy rossz tipp jelentősen megzavarhatja az ismétlési időzítés becslését. Phil Karn nehéz körülmények között fedezte fel ezt a problémát. Ő lelkes rádióamatőr, aki a TCP/IP-csomagok amatőr rádióval történő átvitelével foglalkozik, ami egy hírhedten megbízhatatlan médium. Javaslata egyszerű: ne frissítsük a becslések értékét újraküldött szegmensek esetén, hanem minden kudarc esetén duplázzuk meg az időzítés hosszát, amíg a szegmens végül át nem jut. Ezt a javítást **Karn-féle algoritmusnak** nevezik [Karn és Partridge, 1987]. A legtöbb TCP-implementációban alkalmazzák.

A TCP nem csak az ismétlési időzítőt használja, hanem a **folytatódó időzítőt** is (**persistence timer**). Ezt az alábbi holtpont elkerülésére tervezték. A vevő küld egy nyugtát 0 ablakmérettel, amivel a küldőt várakozásra kéri. Később a vevő frissíti az ablakot, de a frissítést hordozó csomag elvész. Most a küldő és a fogadó is arra vár, hogy a másik tegyen valamit. Amikor a folytatódó időzítő lejár, a küldő egy kérést küld a vevőnek, amire válaszul megkapja az ablakméretet. Ha ez még mindig 0, a folytatódó időzítőt újraindítja, és az egész folyamat megismétlődik, különben, ha nagyobb 0-nál, megkezdheti az adatátvitelt.

A harmadik időzítő, amelyet néhány implementáció használ, az **életben tartó időzítő** (**keepalive timer**). Amikor egy összeköttetés már régóta tétlen, az életben tartó időzítő lejár, és ennek hatására a TCP ellenőrzi, hogy partnere még mindig működik-e. Ha a távoli entitás nem válaszol, az összeköttetés befejeződik. Ez a szolgáltatás ellentmondásos, mert többletterhelést okoz, és egy átmeneti hálózatszakadás hatására befejezhet egy amúgy még működő összeköttetést.

Az utolsó időzítő, amit minden TCP-összeköttetésben alkalmaznak, a *TIME WAIT* állapotban az összeköttetés bontásakor használt időzítő. Ez a maximális csomagélettartam kétszereséig jár, hogy biztosítsa az összeköttetés lebontása után az összeköttetés összes korábban generált csomagjának kihalását.

6.5.10. A TCP torlódáskezelése

Utoljára hagytuk a TCP egyik legfontosabb feladatát: a torlódáskezelést. Amikor bármely hálózatban a felajánlott terhelés nagyobb, mint amennyit a hálózat kezelni képes, torlódás keletkezik. Ez alól az internet sem kivétel. A hálózati réteg az útválasztókban lévő várakozási sorok növekedését figyelve észreveszi a torlódás tényét, és – ha csak a csomagok eldobásával is – megpróbálja kezelni. A szállítási réteg feladata, hogy torlódási jelzést kapjon a hálózati rétegtől, és lecsökkentse a hálózatba küldött forgalom sebességét. Az interneten a TCP játssza a legfőbb szerepet a torlódások kezelésében, ahogy a megbízható adatátvitelben is. Éppen ezért ilyen rendkívüli protokoll.

A torlódáskezelés általános eseteit már tárgyaltuk a 6.3. szakaszban. A legfontosabb következtetésünk az volt, hogy kétállapotú torlódási jelzések esetén egy AIMD (Additive Increase Multiplicative Decrease – additív növelés, multiplikatív csökkentés) vezérlési törvényt alkalmazó szállítási protokoll egy igazságos és hatékony sávszélességkiosztáshoz közelít. A TCP torlódáskezelése is ezt a módszert valósítja meg egy ablak

segítségével, és a csomagvesztést használja fel, mint kétállapotú torlódási jelzést. Ennek elérése érdekében a TCP egy **torlódási ablakot** (**congestion window**) használ, amelynek a mérete megmutatja, hogy a küldő összesen hány bájt adatot tarthat a hálózaton egy időben. Az ennek megfelelő sebesség az ablak mérete osztva összeköttetéshez tartozó körülfordulási idővel. A TCP az ablak méretét az AIMD-szabálynak megfelelően állítja be.

Emlékezzünk vissza, hogy a torlódási ablakot a forgalomszabályozási ablak *mellett* tartjuk karban, amely a vevő által a pufferben tárolható bájtok számát mutatja. Mindkét ablakot párhuzamosan követjük, és az elküldhető bájtok száma megegyezik a kisebbik ablak méretével. Így az effektív ablak kisebb, mint amit a küldő és a vevő helyesnek vél. Tehát kettőn áll a vásár. A TCP leáll az adatok küldésével, ha bármelyik ablak, akár a torlódási, akár a forgalomszabályozási ablak ideiglenesen tele van. Ha a vevő azt mondja, „küldj 64 KB-ot", de a küldő tudja, hogy egy 32 KB-nál nagyobb löket eltömíti a hálózatot, akkor csupán 32 KB-ot fog küldeni. Másrészt, ha a vevő azt mondja, hogy „küldj 64 KB-ot", de a küldő tudja, hogy egy 128 KB-os löket is könnyedén átfér, akkor is csak a kért 64 KB-ot fogja elküldeni. Mivel a forgalomszabályozási ablakról már beszéltünk korábban, most csak a torlódási ablakot tárgyaljuk.

A modern torlódáskezelés nagyrészt Van Jacobson [1988] erőfeszítéseinek köszönhetően került a TCP-be. A történet magával ragadó. 1986-ban történt, amikor a korai internet növekvő népszerűsége elvezetett az első, **torlódási összeomlás** (**congestion collapse**) néven elterjedt, hosszan elnyúló időszakhoz, amely alatt a hasznos adatátvitel meredeken (akár több századára) lecsökkent a hálózatban található torlódás következtében. Jacobson (és sokan mások) elhatározták, hogy végére járnak a történteknek, és gyógyírt találnak rá.

Az a magas szinten elkövetett javítás, amit Jacobson megvalósított, egy AIMD torlódási ablak volt. A már létező TCP-megvalósításba Jacobson úgy olvasztotta bele a javítását, hogy az üzenetformátumokat nem kellett megváltoztatni, így a végeredmény gyorsan telepíthető lett. Habár ez a megoldás legérdekesebb része, a TCP torlódáskezelési részének az összetettségéért is nagymértékben felel. Kezdetben észrevette, hogy a csomagvesztés használható a torlódás jelzésére. Bár egy kicsit később érkezik (mivel a hálózat már torlódott), de igen megbízható. Végül is nehéz olyan útválasztót építeni, amelyik nem dobja el a csomagokat túlterhelés esetén. Ez valószínűleg a jövőben sem változik. Még ha terabájt méretű pufferekben is fogjuk tárolni a rengeteg csomagot, valószínűleg akkor is lesz ezeket feltöltő terabájt/másodperc sebességű hálózatunk.

A csomagvesztés torlódási jelzésként történő alkalmazása azonban ritkán előforduló átviteli hibákat feltételez. Vezeték nélküli hálózatokon, például a 802.11 esetén, sajnos nem ez a helyzet, éppen ezért itt az adatkapcsolati réteg saját újraküldési megoldást alkalmaz. A vezeték nélküli újraküldések ezért a hálózati réteg felé elfedik az átviteli hibákat. Más összeköttetéseken, mint vezetékes vagy optikai kapcsolatokon pedig a hibaarány tipikusan alacsony.

Minden interneten használt TCP-algoritmus azt feltételezi, hogy az elveszett csomagokat hálózati torlódások okozzák, és figyelik az időzítők lejárását, valamit a baj előjelei után kutatnak éppen úgy, ahogy a bányászok figyelik a kanárimadaraikat. A csomagvesztések pontos és időben történő felfedezéséhez egy jó újraküldési időzítőre van szükség. Arról már esett szó, hogyan veszi figyelembe a TCP az újraküldési időzítőknél a

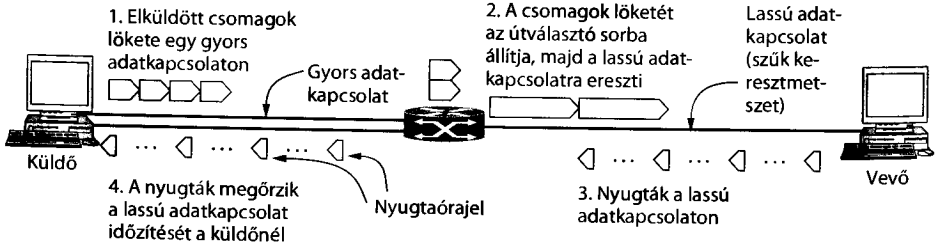
körülfordulási idő átlagának és szórásának becslését. A szórási tényező bevezetésével Jacobson egy nagyon fontos lépést tett. Egy megfelelő újraküldési időzítéssel a TCP-küldő nyomon tudja követni azokat a kinn lévő bájtokat, melyek még a hálózatban találhatók. Ehhez egyszerűen megnézi az elküldött és nyugtázott sorszámok különbségét.

Most már úgy tűnik, könnyű feladatunk van. Mindössze annyi a dolgunk, hogy nyilvántartsuk a torlódási ablakot az elküldött és nyugtázott sorszámok segítségével, és a méretét az AIMD-szabállyal állítsuk. Ahogy azonban várható, azért ez nem ilyen egyszerű. Először is, azt az utat, amelyen a csomagok a hálózatba kerülnek, még ha csak rövid időre is, meg kell feleltetni a hálózati útvonalnak, ellenkező esetben a forgalom torlódást fog okozni. Vegyünk például egy állomást 64 KB torlódási ablakkal, amely 1 Gbit/s sebességű kapcsolt Ethernetre csatlakozik. Ha az állomás a teljes ablakot egyszerre küldi el, ez a löket akár egy lassú 1 Mbit/s sebességű ADSL-vonalon is átutazhat az útvonal mentén. A löket, amelynek továbbítása csupán fél milliszekundumnyi időt vett igénybe az 1 Gbit/s sebességű vonalon, az 1 Mbit/s sebességű ADSL-vonalat fél másodpercre is eldugítja, teljesen tönkretéve olyan protokollokat, mint amilyen például a VoIP. Ez a viselkedés esetleg megfelel egy torlódást okozó protokoll számára, de nem alkalmas egy torlódáskezelő protokollhoz.

Kiderült azonban, hogy a csomagok egy kis csoportját az előnyünkre is fordíthatjuk. A 6.43. ábra bemutatja, mi történik, ha egy küldő egy gyors hálózaton (1 Gbit/s-os adatkapcsolaton) 4 csomagot küld egy vevőnek egy lassú hálózatra (1 Mbit/s), amely a hálózati útvonal szűk keresztmetszete vagy leglassúbb része. Kezdetben a négy csomag olyan gyorsan áthalad az adatkapcsolaton, amilyen gyorsan csak a küldő el tudja küldeni. Az útválasztónál a továbbküldésig a csomagok várakozási sorba kerülnek, hiszen egy csomag elküldése tovább tart a lassú adatkapcsolaton, mint egy új megérkezése a gyorsabb adatkapcsolaton. A várakozási sor mérete azonban nem nagy, hiszen csak pár csomagot küldtünk egyszerre. Vegyük észre a csomag megnövekedett hosszát a lassabb adatkapcsolaton. Ugyanaz a csomag (például 1 KB méretű) most hosszabb, hiszen több időbe telik elküldeni egy lassabb adatkapcsolaton, mint egy gyorsabb adatkapcsolaton.

A csomagok végül megérkeznek a vevőhöz, ahol az nyugtázza őket. A nyugták időpontjai a csomagok beérkezési idejét tükrözik a vevőnél, miután azok átkeltek a lassú adatkapcsolaton. A csomagok szétterülnek a gyors adatkapcsolaton haladó eredeti csomagokhoz képest. A nyugták, miközben átutaznak a hálózaton a vevőtől vissza a küldőig, megőrzik eredeti időzítéseiket.

A legfontosabb megállapításunk tehát a következő: a nyugták nagyjából olyan sebességgel érkeznek vissza a küldőhöz, mint amilyen sebességgel a csomagokat az útvonal



6.43. ábra. *A küldőtől jövő csomagok egy lökete és a visszatérő nyugta ütemezése*

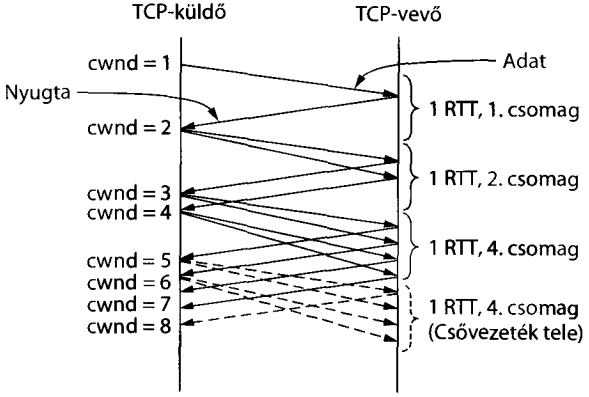
leglassabb adatkapcsolatán továbbítani lehet. Pontosan ez az a sebesség, amelyet a küldőnek használnia kell. Ha az új csomagokat ilyen sebességgel bocsátja a hálózatba, akkor a csomagokat olyan gyorsan fogja küldeni, amilyen gyorsan a leglassabb adatkapcsolat megengedi, de nem fognak a várakozási sorok feltöltődni egyik útválasztóban sem az útvonal mentén, és így nem jön létre torlódás sem. Ez az időzítés **nyugtaórajel (ack clock)** néven ismert, és alapvető része a TCP-nek. Nyugtaórajel használatával a TCP elsimítja a kimenő forgalmat, és elkerüli a felesleges sorbaállást az útválasztókban.

Másik fontos megfontolásunk az, hogy az AIMD-szabály szerint nagyon sok időt fog igénybe venni gyors hálózatokon a megfelelő munkapont elérése, ha a torlódási ablak kis méretről indul. Vegyünk egy szerény hálózati útvonalat, amely 10 Mbit/s sebességgel és 100 ms körülfordulási idővel (RTT) rendelkezik. A megfelelő torlódási ablak mérete a sávszélesség-késleltetés szorzat, ami 1 Mbit vagy 100 darab 1250 bájtos csomag. Ha a torlódási ablak mérete 1 csomagról indul, és minden körülfordulás alatt 1 csomaggal nő, akkor 100 körülfordulási időbe, tehát 10 másodpercbe fog telni, mire az összeköttetés sebessége a kívánatos közelébe ér. Ez elég hosszú idő annak kivárására, hogy elérjük a megfelelő átviteli sebességet. Csökkenthetnénk ezt az időt, ha nagyobb kezdeti ablakkal indulnánk, mondjuk 50 csomaggal. De ez az ablakméret lassú vagy rövid adatkapcsolatokra hatalmas lenne. Ha a teljes ablakot egyszerre kihasználnánk, akkor a leírtak alapján torlódást okozna.

Jacobson e helyett a fenti problémák megoldására a lineáris és az exponenciális növekedés egyfajta keveréke mellett döntött. Amikor az összeköttetés felépült, a küldő a torlódási ablak kezdeti méretét egy kicsi, maximálisan 4 szegmens méretűre választja; a részleteket az RFC 3390 tartalmazza. A 4 szegmens használata a korábbi 1 szegmens használatát váltotta fel a tapasztalatokból kiindulva. A küldő ekkor elküldi a kezdeti ablakméretet. A csomagok nyugtázása egy körülfordulási időbe kerülnek. Az újraküldési időzítő lejárta előtt nyugtázott minden szegmens után a küldő egy szegmensnek megfelelő számú bájtot hozzáad a torlódási ablakhoz, továbbá, mivel azt a szegmenst nyugtázták, eggyel kevesebb csomag lesz a hálózatban. A végeredmény tehát az, hogy minden nyugtázott szegmens után két új szegmenst lehet elküldeni. A torlódási ablak minden körülfordulási idő után megduplázódik.

Ezt az algoritmust **lassú kezdésnek** vagy **lassú kezdést biztosító algoritmusnak (slow start)** hívják, de egyáltalán nem lassú – valójában exponenciális a növekedés – csupán az előző algoritmussal szemben az, ahol a teljes forgalomszabályozási ablakot egyszerre küldték el. A lassú kezdést a 6.44. ábra mutatja. Az első körülfordulási idő alatt a küldő mindössze egy csomagot ereszt a hálózatba (és a küldő is egy csomagot kap). A következő körülfordulási idő alatt két csomagot, a harmadik körülfordulási idő alatt pedig már négy csomagot küld.

A lassú kezdés kielégítően működik olyan adatkapcsolatok esetén, ahol különböző a sebesség és a körülfordulási idő, és a küldő adási sebességének a hálózati útvonal sebességéhez való illesztéséhez nyugta órajelet használ. Vessünk egy pillantást a 6.44. ábrára, és figyeljük meg a nyugták érkezését a vevőhöz! Amikor a küldő egy nyugtát kap, akkor megnöveli a torlódási ablak méretét eggyel, és azonnal két csomagot küld a hálózatba. (Az egyik csomag az eggyel növelés miatt; a másik a nyugtázott, és így a hálózatot elhagyó csomag helyett kerül elküldésre. Bármely időpillanatban a torlódási ablak mérete adja meg az összes nyugtázatlan csomag számát.) Ez a két csomag azonban nem fel-



6.44. ábra. *Lassú kezdést biztosító algoritmus, egy szegmens méretű, kezdeti torlódási ablakkal*

tétlenül érkezik meg pontosan olyan időközzel a vevőhöz, mint ahogy elküldték azokat. Vegyünk például egy küldőt egy 100 Mbit/s Ethernet-vonalon. Minden 1250 bájtos csomag elküldése 100 μs-ot igényel. A csomagok közötti időköz tehát akár 100 μs méretű is lehet. A helyzet akkor változik meg, ha ezek a csomagok valahol az útvonal mentén áthaladnak egy 1 Mbit/s sebességű ADSL-adatkapcsolaton. Most 10 ms időbe telik ugyanannak a csomagnak a továbbítása. Ez azt jelenti, hogy a minimális időköz a csomagok között százszorosára nőtt. Ez az időköz ilyen nagy marad, hacsak nem várják be egymást egy várakozási sorban valamely gyorsabb adatkapcsolaton.

A 6.44. ábra ezt a jelenséget az adatcsomagok vevőhöz érkezésénél a nyilak végei között egy kisebb távolság feltüntetésével ábrázolja. Ugyanez a távolság szerepel a nyugták elküldésénél, és így a nyugták küldőhöz való visszaérkezésénél is. Ha a hálózati útvonal lassú, akkor a nyugták lassan érkeznek (egy körülfordulásnyi idővel később). Ha a hálózati útvonal gyors, akkor a nyugták is hamar megérkeznek (ugyancsak egy körülfordulásnyi idővel később). A küldőnek csupán a nyugtaórajelet kell követnie az új csomagok kiküldésénél. A lassú kezdés pont ezt teszi.

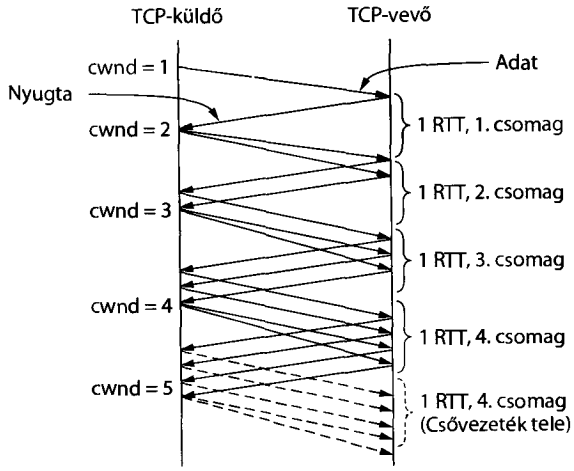
Mivel a lassú kezdés exponenciális növekedéssel jár, ezért előbb vagy utóbb túl sok csomag kerül túl gyorsan a hálózatba. Amint ez megtörténik, a hálózatban található várakozási sorok betelnek, és a teli sorok következtében csomagok fognak elveszni. Ezek után a TCP-küldő időzítője lejár, amikor egy nyugta nem érkezik meg időben. A lassú kezdés túl gyors növekedésének a 6.44. ábrán is szemtanúi lehetünk. Három körülfordulási idő után négy csomag található a hálózatban. A négy csomag megérkezéséhez a vevőhöz egy egész körülfordulásnyi időre van szükség. Tehát a négycsomagnyi torlódási ablakméret éppen megfelelő ehhez az összeköttetéshez. Ahogy azonban ezeket a csomagokat is nyugtázzák, a lassú kezdés tovább növeli a torlódási ablakot, és az a következő körülfordulási idő végére eléri a nyolccsomagnyi méretet. Összesen csupán négy csomag jut el a vevőhöz egyetlen körülfordulási idő alatt, függetlenül attól, hány csomagot küldtünk. A hálózati csővezeték tehát tele van. Ha a küldő további csomagokat bocsát a hálózatba, azok beragadnak az útválasztók várakozási soraiba, mivel nem lehet a küldő felé elég gyorsan továbbítani. A torlódás és a csomagvesztés pedig hamarosan bekövetkezik.

A küldő a lassú kezdés kordában tartására egy, a kapcsolathoz tartozó határt szab, amelyet a **lassú kezdés küszöbértékének** (**slow start threshold**) hívunk. Ennek értékét kezdetben önkényesen magasan tartjuk, a forgalomszabályozási ablak méretén, hogy az összeköttetés sebességét ne korlátozza. A TCP a lassú kezdés során a torlódási ablak méretét mindaddig növeli, amíg nem jár le az újraküldési időzítő, vagy a torlódási ablak át nem lépi ezt a küszöböt (vagy meg nem telik a vevő ablaka).

Amint a küldő – például az újraküldési időzítő lejárta miatt – felfedez egy csomagvesztést, a küszöböt a torlódási ablak méretének felére csökkenti, és a teljes folyamatot elölről kezdi. Az ötlet az, hogy az aktuális ablakméret túl nagy, mivel az előbbiekben torlódást okozott, és csak most, az időzítő lejártával vettük észre. Az ablak méretének a fele, amit már korábban is sikeresen alkalmaztunk, valószínűleg a legjobb becslése egy olyan torlódási ablaknak, amely közel van az útvonal kapacitásához, de még nem okoz csomagvesztést. A 6.44. ábrán látható példánkban a 8 csomagra növekedett torlódási ablakunk már okozhat csomagvesztést, míg a 4 csomag méretű ablak az előző körülfordulási idő alatt éppen a megfelelő nagyságú volt. A torlódási ablak méretét aztán az alacsony kezdeti értékére állítjuk, és visszatérünk a lassú kezdéshez.

Amint a lassú kezdés küszöbértékét átlépi, a TCP lassú kezdésről additív növekedésre vált. Ebben az üzemmódban a torlódási ablak méretét minden körülfordulási idő után eggyel növeli. A megvalósítás során a léptetés, ahogy a lassú kezdésnél is, általában minden nyugtázott szegmens után történik, nem pedig a körülfordulási időnként történő növeléssel. Legyen a torlódási ablak mérete *cwnd* és a legnagyobb lehetséges szegmensméret *MSS*. Egy gyakori megközelítés szerint a *cwnd* / *MSS* számú csomag minden nyugtájának megérkezése után a *cwnd* értékét (*MSS* × *MSS*) / *cwnd* értékkel növelik. A léptetésnek nem kell feltétlenül gyorsnak lenni. A megoldás lényege az, hogy a TCP-összeköttetés torlódási ablaka az idő nagy részét az optimális pont közelében töltse – ne legyen olyan kicsi, hogy lelassítsa az összeköttetést, de ne legyen olyan nagy sem, hogy torlódást okozzon.

Az additív növelést a 6.45. ábrán mutatjuk be a lassú kezdéssel megegyező esetben. A küldő torlódási ablaka minden körülfordulási idő végére annyit nő, hogy további cso-



6.45. ábra. *A kezdeti, egy szegmens méretű torlódási ablak additív növelése*

magokat tudjon a hálózatra bocsátani. A lassú kezdéshez hasonlítva azonban a lineáris növekedés sokkal lassabb. Kisméretű torlódási ablakok esetén a különbség nem nagy (ahogy jelen esetben sem), de például az ablak 100 csomagnyi méretűre növelése esetén az időkülönbség már számottevő.

A teljesítmény növelésére egy másik eszközünk is van. Az eddigi megoldás gyenge pontja, hogy az újraküldési időzítő lejárását meg kell várnunk, ez pedig hosszú idő lehet, hiszen az időzítések meglehetősen nagyok. Ha egy csomag elvész, a vevő nem tudja nyugtázni, így a nyugtázott sorszámok nem változnak, és a küldő a torlódási ablak telítettsége miatt nem tud további csomagokat küldeni a hálózatba. Ez a helyzet sokáig eltarthat, egészen az újraküldési idő lejártáig. A csomagot ebben az esetben a TCP újraküldi, és újra lassú kezdésbe fog.

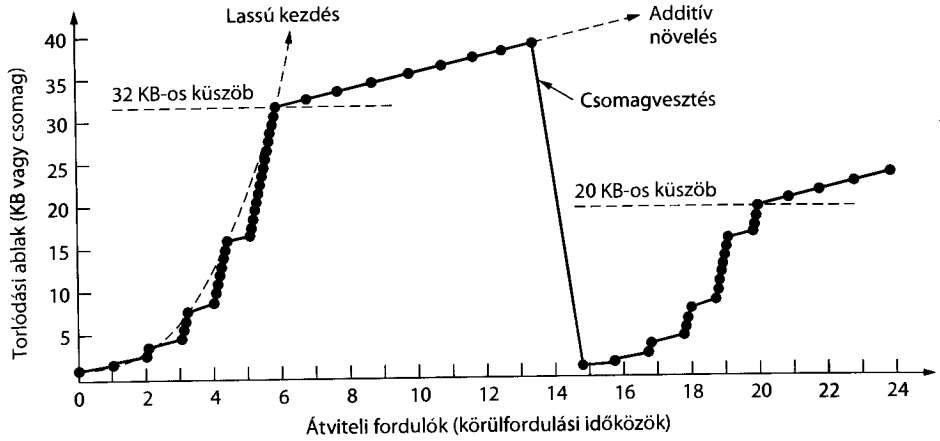
A küldő számára van egy gyorsabb módszer is a csomagvesztés megállapítására. Ahogy az elveszett csomag utáni csomagok beérkeznek a vevőhöz, a vevő nyugtákkal válaszol a küldőnek. Ezek a nyugták ugyanazt a nyugtasorszámot tartalmazzák, ezért ezeket **nyugtamásolatoknak** hívjuk. Amint a küldő egy ilyen nyugtamásolatot kap, feltételezheti, hogy a vevőhöz egy újabb csomag érkezett, de az elveszett csomag még mindig nem került elő.

Mivel a csomagok különböző útvonalakat járhatnak be a hálózatban, nem feltétlenül kell sorrendben beérkezniük. Így előfordulhat, hogy csomagvesztés nélkül kapunk nyugtamásolatokat. Az interneten azonban ez nem túl gyakori. Tulajdonképpen a csomagok sorrendje akkor sem változik meg túlságosan, ha más útvonalon haladnak a hálózatban. Így a TCP, valamelyest önkényesen, azt feltételezi, hogy három nyugtamásolat vétele csomagvesztést jelent. Az elveszett csomagra a nyugták sorszámából következtethetünk, ugyanis a sorrendben következő csomag veszett el. Ez a csomag egyből újraküldhető anélkül, hogy az időzítő lejártát meg kellene várni.

Az ilyen heurisztikát **gyors újraküldésnek** (**fast retransmission**) nevezzük. Amint ez megtörténik, a lassú kezdés küszöbértékét ugyancsak a torlódási ablak pillanatnyi értékének felére állítjuk, ahogyan azt az időzítés túllépése esetén tennénk. A lassú kezdés a torlódási ablak egy csomag méretűre állításával kezdhető újra. Ezzel az ablakmérettel a TCP egy új csomagot küld minden körülfordulási idő után, amely az újraküldött csomag, és minden, a csomagvesztés előtt elküldött adat nyugtájának a visszaérkezéséhez szükséges.

Az eddigiekben felépített torlódáskezelő algoritmust a 6.46. ábrán mutatjuk be. A TCP ilyen megvalósítását **TCP Tahoe**-nak hívjuk, ugyanis a 4.2BSD Tahoe 1988-as kiadásának része volt. A maximális szegmensméret 1 KB. Kezdetben a torlódási ablak mérete 64 KB, de egy időtúllépés következtében a küszöb értékét 32 KB-ra, a torlódási ablakot pedig 1 KB-ra állította a 0-dik menethez. A torlódási ablak mérete exponenciálisan nő, amíg el nem éri a küszöböt (32 KB). Az ablakot minden nyugta beérkezésekor állítja, nem folytonosan, így egy diszkrét lépcsőmintázat alakul ki. Amint a küszöböt átlépte, a növekedés lineárisan folytatódik, minden körülfordulási idő után egy szegmenssel nő.

A 13. körben az átvitel egy szerencsétlenség áldozata lett (sejthettük volna...) és egy csomag el is veszett a hálózatban. Három nyugtamásolat érkezése után a csomagvesztés kiderül, és ekkor a csomag újraküldésre kerül, a küszöb pedig az aktuális ablakméret felére csökken (azaz 40 KB-ról 20 KB-ra), és a lassú kezdés újraindul. Újra kezdve az egy csomagnyi méretű torlódási ablakkal, összesen egy körülfordulási időre van szükség,



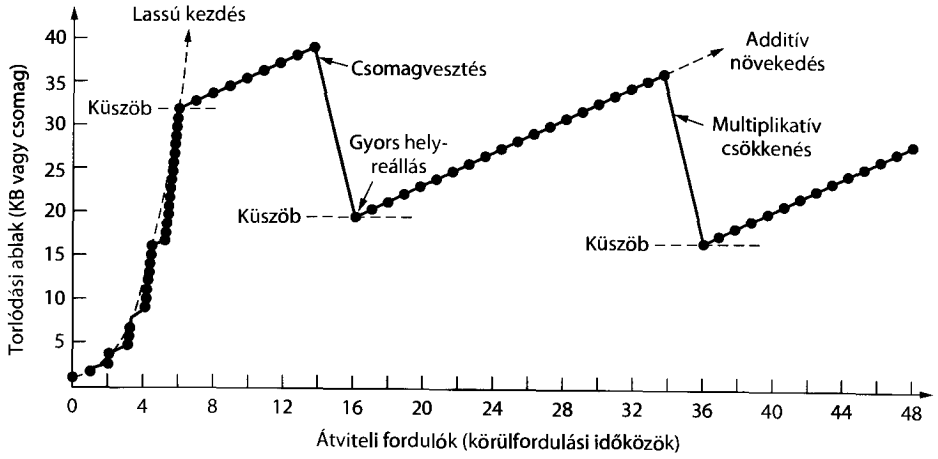
6.46. ábra. *Lassú kezdést követő additív növekedés a TCP Tahoe-ban*

hogy az előzőleg elküldött, valamint az újraküldött csomagok elhagyják a hálózatot, és nyugtázásra kerüljenek. A torlódási ablak az előzőekhez hasonlóan növekszik mindaddig, amíg el nem éri a 20 KB méretű küszöböt. Attól kezdve a növekedés megint lineárissá alakul, és a folytatásban is e szerint növekszik, amíg nyugtamásolatok vagy időtúllépések csomagvesztést nem jeleznek (vagy a vevő ablaka meg nem telik).

A TCP Tahoe (amely megfelelő újraküldési időzítőket alkalmaz) egy működő torlódáskezelő algoritmust biztosított, és megoldotta a torlódási összeomlás problémáját. Jacobson azonban rájött, hogy készíthető ennél is jobb algoritmus. Amikor gyors újraküldés történik, az összeköttetés egy túl nagy torlódási ablakot használ, de a nyugtaórajel még mindig megfelelő. Minden esetben, amikor egy nyugtamásolat érkezik, valószínűleg egy újabb csomag hagyta el a hálózatot. Ha a nyugtamásolatokat a hálózatban található csomagok megszámlálására használjuk, akkor lehetővé válik, hogy pár csomagnak megengedjük a hálózat elhagyását, és folytassuk az új csomagok küldését minden további nyugtamásolat esetén.

A **gyors helyreállítás** (**fast recovery**) egy olyan heurisztikus algoritmus, amely ezt a viselkedést valósítja meg. Ez egy olyan ideiglenes állapot, ami megpróbálja a nyugtaórajelet tovább biztosítani egy olyan torlódási ablakkal, amelynek a mérete az új küszöbbel vagy a gyors újraküldés idején érvényes torlódási ablakméret felével egyenlő. Ennek elérése érdekében a nyugtamásolatokat folyamatosan számolja (beleértve azt a hármat is, amelyek a gyors újraküldést okozták) mindaddig, amíg a hálózatban lévő csomagok száma az új küszöb értékére nem esik. Ez körülbelül fél körülfordulásnyi időbe telik. Ettől kezdve minden egyes nyugtamásolat beérkezésekor egy új csomagot küldhetünk. A gyors újraküldés után egy körülfordulásnyi idővel az elveszett csomagra megérkezik a nyugta. Ekkor megszűnik a nyugtamásolatok özöne, és a gyors helyreállás üzemmód véget ér. A torlódási ablak mérete az új lassú kezdés küszöbértékére áll, és innentől lineárisan növekszik.

A heurisztika végkövetkeztetése az tehát, hogy a TCP elkerüli a lassú kezdést, kivéve, amikor az összeköttetést először felépítjük, vagy amikor időtúllépés következik be. Ez



6.47. ábra. *A TCP Reno gyors helyreállása és fűrészfog-mintázata*

utóbbi még mindig megtörténhet abban az esetben, amikor több csomag is elvész, és a gyors újraküldés nem tudja kielégítően kijavítani a hibát. A lassú kezdések helyett egy működő összeköttetés torlódási ablakmérete **fűrészfog-mintázatot** alkot, additív növekedéssel (egy szegmenssel minden körülfordulási időnként), és multiplikatív csökkenéssel (egy körülfordulási idő alatt a felére csökken). Ez pont az AIMD-szabály, amit megkíséreltünk megvalósítani.

A fűrészfog-mintázatot a 6.47. ábra mutatja, amelyet a **TCP Reno** állított elő. A TCP Reno az 1990-ben kiadott 4.3BSD Reno része volt, és innen kapta a nevét. A TCP Reno gyakorlatilag a TCP Tahoe, gyors helyreállással megfűszerezve. A bevezető lassú kezdés után a torlódási ablak lineárisan növekszik mindaddig, amíg csomagvesztést nem észlel a nyugtamásolatoknak köszönhetően. Az elveszett csomagot újraküldi, és a gyors helyreállás segítségével az újraküldött csomag nyugtázásáig a nyugtaórajelet biztosítja. Innentől a torlódási ablak a növekedést az új lassú kezdés küszöbértékéről kezdi, nem pedig 1 szegmensről. Ez a viselkedés korlátlan ideig folytatódik, és az összeköttetés torlódási ablaka az idő nagy részét az optimális pont, tehát a sávszélesség-késleltetés szorzat közelében tölti.

A TCP Reno a torlódásiablak-szabályozó megoldásaival több mint két évtizede meghatározza a TCP torlódáskezelésének alapjait. Az azóta eltelt évek során, a módszeren csupán apró módosításokat végeztek, mint például a kezdeti ablak paramétereinek megváltoztatását vagy különböző, nem egyértelmű részek eltávolítását. Némely továbbfejlesztés arra irányult, hogy egy ablakban történt két vagy több csomag elvesztését helyreállítsák. A TCP NewReno például egy újraküldés után a nyugtasorszámokat részlegesen előrelépteti, hogy más csomagvesztéseket is megtaláljon és kijavítson [Hoe, 1996]. Az RFC 3782 részletesen ír a módszerről. A 90-es évek közepétől különböző olyan változatok jelentek meg, amelyek a leírt alapelveket követik, de valamelyest különböző vezérlési törvényeket alkalmaznak. Például a Linux a CUBIC TCP [Ha és mások, 2008], a Windows pedig a Compound TCP [Tan és mások, 2006] nevű változatot használja.

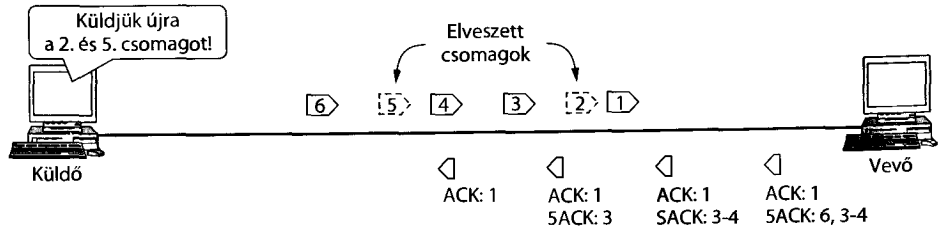
A TCP-megvalósításokra azonban két komoly változás is hatással volt. Az első szerint a TCP összetettsége főleg abból adódik, hogy a nyugtamásolatok tömkelegéből próbál

következtetni arra, hogy mely csomagok érkeztek meg épségben, és mely csomagok vesztek el. A halmozott nyugtázás ilyen információt nem tartalmaz. Az egyszerű megoldás a **SACK (Selective ACKnowledgements – szelektív nyugtázás)** használata, amely akár három bájtsorszám-intervallumot is felsorolhat, amelyek vétele sikeres volt. Ennek az információnak a segítségével a küldő sokkal közvetlenebbül el tudja dönteni, mely csomagokat kell újraküldeni, és a torlódási ablak megvalósításában is nagy segítséget nyújt, hogy a még úton levő csomagokat is nyomon lehet követni.

Amikor a küldő és a vevő felépítenek egy összeköttetést, mindketten elküldik a *SACK engedélyezve* (*SACK permitted*) TCP-opciót, amellyel jelzik, hogy a szelektív nyugtákat megértik. Amint a kapcsolaton a SACK engedélyezve van, akkor az a 6.48. ábrán látható módon működik. A vevő a TCP *Nyugtaszám* mezőjét a szokásos módon használja, tehát a legmagasabb, sorrendben beérkezett bájt halmozott nyugtázására. Ha a 3. csomagot kapja, ami nem a sorrendben következő csomag (mert a 2 csomag elveszett), a fogadott adatra egy *SACK opciót* küld, az 1 csomagra vonatkozó halmozott nyugta (-másolat) mellett. A *SACK opció* azokat a bájtsorszám-intervallumokat tartalmazza, amely adatokat sikeresen vett a vevő a halmozott nyugtában közölt sorszámon felül. Az első intervallum azt a csomagot jelenti, amely a nyugtamásolatot okozta, a többi jelenlévő intervallum pedig régebbi blokkok nyugtája. Általában legfeljebb három intervallumot használnak. Mire a hatodik csomag megérkezik, már két SACK bájtsorszám-intervallum jelzi, hogy a 6. és a 3-tól 4-ig terjedő csomagok sikeresen megérkeztek. Ez kiegészül az 1. csomagig terjedő halmozott nyugtával. Minden egyes beérkezett *SACK opció* lehetővé teszi a küldőnek, hogy meghatározza az újraküldendő csomagokat. Ebben az esetben a 2. és 5. csomag újraküldése lenne a jó ötlet.

A SACK szigorúan csak ajánlást nyújt. A tényleges csomagvesztés ténye ugyanúgy megállapítható nyugtamásolatokkal és a torlódási ablak paramétereinek állításával, mint az eddigiekben. SACK használatával azonban a TCP hamarabb helyreállíthatja az adatátvitelt olyan esetekben, amikor nagyjából egy időben több csomag is elveszett, mivel a TCP-küldő pontosan tudja, mely csomagok nem jutottak el a vevőig. A SACK manapság széles körben elterjedt. Működését az RFC 2883 írja le, a SACK-ot használó TCP-torlódáskezelést pedig az RFC 3517.

A másik komolyabb változás az ECN (Explicit Congestion Notification – explicit torlódásjelzés) használata a torlódás jelzésére a csomagvesztésen felül. Az ECN az IP-rétegbeli megoldás az állomások értesítésére, hogy a hálózatban torlódás található, ahogy arról az 5.3.4. szakaszban írtunk. Segítségével a TCP-vevő torlódási jelzést kaphat az IP-rétegtől.



6.48. ábra. *Szelektív nyugtázás*

Egy TCP-kapcsolat akkor használ ECN-t, ha mind a küldő, mind a vevő ezt jelezte az összeköttetés felépítésekor az *ECE* és *CWR* bitek beállításával. ECN használata esetén minden TCP-szegmenst megjelölnek az IP-fejlécben annak jelzésére, hogy az képes ECN-jelzések szállítására. Az ECN-t támogató útválasztók minden, ECN-jelzést szállítani képes csomagban jelzik, ha torlódás közeleg, ahelyett hogy a csomagot egyszerűen eldobnák annak bekövetkezte után.

A TCP-vevő a csomagban az ECN torlódási jelzést észlelve az *ECE* (ECN Echo) bit segítségével jelzi a küldőnek, hogy a csomagjai torlódást tapasztaltak. A küldő a vevő felé a jelzés vételét a *CWR* (Congestion Window Reduced – torlódási ablak mérete csökkentve) bit segítségével közli.

A TCP-küldő ezekre a torlódási jelzésekre pontosan úgy reagál, mint ahogy a nyugtamásolatokkal megállapított csomagvesztésekre. A helyzet azonban sokkal kellemesebb, hiszen a torlódást felfedezték, és egyetlen csomagnak sem esett bántódása. Az ECN-t az RFC 3168 írja le. Az ECN mind a hoszt, mind az útválasztó támogatását igényli, ezért még nem túl elterjedt az interneten.

Részletesebb információkat a TCP-ben alkalmazott torlódáskezelő megoldásokról, és azok viselkedéséről az RFC 5681 tartalmaz.

6.5.11. A TCP jövője

A TCP-t az internet igáslovaként már rengeteg alkalmazásban használják, és az idők folyamán sokat fejlesztették, hogy megfelelő teljesítőképességgel rendelkezzen a hálózatok széles spektrumán. Sok változatát használják, amelyek némileg különböző megoldásokkal működnek, mint az általunk tárgyalt klasszikus algoritmusok, különösen a torlódáskezelés és a támadások elleni robusztusság területén. A TCP valószínűleg az internettel együtt fog fejlődni. A következőkben két különleges esetet említünk meg.

Az egyik, hogy a TCP nem minden alkalmazásnak biztosít kielégítő szállítási szolgáltatást. Néhány alkalmazás például elvárja, hogy az általuk küldött üzenetek vagy rekordok megőrizzék az üzenethatárukat. Más alkalmazások egymáshoz kapcsolódó párbeszédeket használnak, mint például egy webböngésző, amely számos objektumot kér le ugyanattól a szervertől. Megint más alkalmazások az általuk használt hálózati útvonalak felett szeretnének nagyobb irányítási lehetőséget. A klasszikus TCP csatlakozó interfész (socket) nem elégíti ki ezeket a kívánalmakat. Az alkalmazásoknak kell tehát megbirkózni minden olyan problémával, amit a TCP nem old meg. Ez vezetett az olyan újabb protokollok kidolgozása felé, amelyek némileg különböző interfészt nyújtanak. Két példát emelhetünk ki: az RFC 4960-ban definiált SCTP- (Stream Control Transmission Protocol – folyamvezérlő átviteli protokoll) és az SST- (Structured Stream Transport – strukturált folyamszállítás) protokollt [Ford, 2007]. Sajnos azonban, amikor valaki meg akar változtatni valami olyat, ami már hosszú idők óta jól működik, mindig nagy harcot vált ki a „felhasználók több lehetőséget akarnak" és a „ha működik, ne nyúlj hozzá" tábor között.

A másik a torlódáskezelés. A fejtegetéseink, és az idők folyamán kifejlesztett megoldások következtében azt gondolhatnánk, hogy a torlódáskezelés már megoldott probléma. Sajnos nem így van. Az általunk tárgyalt, jelenlegi formájában használt TCP-

torlódáskezelés, amely ráadásul széles körben elterjedt, a csomagvesztést használja a torlódás jelzésére. Amikor Padhye és mások 1998-ban a fűrészfog mintán alapuló TCP átbocsátóképességét modellezték, megállapították, hogy a sebesség növekedésével a csomagvesztési aránynak meredeken csökkennie kell. 1 Gbit/s áteresztőképesség eléréséhez 100 ms körülfordulási idő szükséges, és 1500 bájtos csomagok esetén egy csomag veszhet el nagyjából minden 10 percben. Ez 2×10^{-8} csomagvesztési arány, ami hihetetlenül alacsony. Ebben az esetben a csomagvesztés egyszerűen túl ritka ahhoz, hogy azt torlódási jelzésként kezeljük, valamint a csomagvesztés más forrása (például 10^{-7} nagyságrendű csomagtovábbítási hibaarány) könnyedén elnyomhatja a 10^{-8} nagyságrendű csomagvesztési arányt, csökkentve az áteresztőképességet.

Ez az összefüggés régebben nem jelentett problémát, de azóta a hálózatok már sokkal gyorsabbak, ami sok mérnököt a torlódáskezelés újragondolásához vezetett. Az egyik lehetőség az, hogy egy olyan, alternatív torlódáskezelést alkalmazunk, amely nem a csomagvesztésre építi a torlódás jelzését. A 6.2. szakaszban számos példát adtunk ilyen megoldásra. A FAST TCP a körülfordulási időre épít, mint jelzésre, amely torlódás esetén megnövekszik [Wei és mások, 2006]. Több más megoldás is lehetséges, az idő pedig majd kiválasztja a legjobbat.

6.6. Teljesítőképesség

A teljesítőképességgel kapcsolatos kérdések nagyon fontosak a számítógép-hálózatok esetében. Amikor számítógépek százai vagy ezrei vannak összekapcsolva, mindennaposak az előre nem látható következményekkel járó bonyolult kölcsönhatások. Gyakran ez az összetettség gyenge teljesítőképességhez vezet, aminek senki sem tudja az okát. A következő alfejezetben sok, a hálózati teljesítőképességgel kapcsolatos kérdést megvizsgálunk, hogy láthassuk a fennálló problémákat, és hogy mit lehet tenni ellenük.

Sajnos a hálózat teljesítőképességének megértése inkább művészet, mint tudomány. E mögött kevés a gyakorlatban is valóban alkalmazható elmélet. A legtöbb, amit tehetünk, hogy a tapasztalatokból nyert ökölszabályokat és életből vett példákat mutatunk. Szándékosan halasztottuk ezt a témát a TCP szállítási rétegbeli megtárgyalása utánra, mert az alkalmazás által tapasztalt teljesítőképesség a szállítási, hálózati és adatkapcsolati rétegek kombinált teljesítőképessége. A TCP ezenkívül jó példának fog szolgálni több helyen is.

A következőkben a hálózat teljesítőképességét hat oldalról vizsgáljuk meg:

1. A teljesítőképesség problémái.
2. A hálózat teljesítőképességének mérése.
3. Hoszt tervezése gyors hálózatokhoz.
4. Gyors szegmensfeldolgozás.

5. Fejrésztömörítés

6. Protokollok „elefánthálózatokra".[4]

A fenti szempontok a hálózati teljesítőképességet mind a hoszt-, mind a hálózati oldalról megvizsgálják, továbbá elemzik azt méretben és sebességben növekvő hálózat esetén is.

6.6.1. A számítógép-hálózatok teljesítőképességének problémái

Néhány teljesítőképességgel kapcsolatos problémát, mint például a torlódást, átmeneti erőforrás-túlterhelés okoz. Ha hirtelen nagyobb forgalom alakul ki egy útválasztónál, mint amekkorát az kezelni képes, torlódás alakul ki és a teljesítőképesség romlik. A torlódást részletesen tanulmányoztuk ebben és az előző fejezetben.

A teljesítőképesség akkor is csökken, ha strukturális erőforrás-kiegyensúlyozatlanság áll fönn. Például, ha egy gigabites kommunikációs vonal csatlakozik egy kis teljesítményű PC-hez, a gyenge hoszt képtelen lesz elég gyorsan feldolgozni a beérkező csomagokat, tehát egy részük elvész. Ezeket a csomagokat később újraküldik, ami növeli a késleltetést, pazarolja a sávszélességet, és általában véve rontja a teljesítőképességet.

A túlterhelést egyidejű események is kiválthatják. Például, ha egy szegmens rossz paramétert (például rossz célportot) tartalmaz, sok esetben a figyelmes vevő egy hibajelzést küld vissza. Most képzeljük el, mi történne, ha a rossz szegmens üzenetszórással 1000 géphez jutna el: mindegyik visszaküldhet egy hibajelzést. Az ebből kialakuló üzenetszórási **vihar (broadcast storm)** romba döntheti a hálózatot. Az UDP ebben a betegségben szenvedett, amíg az ICMP-protokollt olyan értelemben meg nem változtatták, hogy a hosztok ne küldjenek hibajelzést üzenetszórásos címre érkezett UDP-szegmensekre.

Egy másik példa egyidejű események által okozott túlterhelésre, amely egy áramszünet után következhet be. Amikor helyreáll az energiaszolgáltatás, az összes gép egyszerre újraindul. Egy tipikus újraindulási menetrend megkövetelheti, hogy a hoszt először egy (DHCP) szervert keressen fel, hogy megtudja valódi azonosítóját, majd egy állományszolgáltatóról letöltse az operációs rendszerét. Ha gépek százai egyszerre tesznek így, a szerver valószínűleg összeomlik a terhelés alatt.

Még ha egyidejű események nem is okoznak túlterhelést, és elegendő erőforrás áll rendelkezésre, a rendszer rossz beállítása is oka lehet a gyenge teljesítőképességnek. Például, ha egy gépnek bőven van szabad processzorkapacitása és memóriája, viszont kevés memóriát foglaltak le pufferterület céljára, a forgalomszabályozás lelassítja a szegmensek fogadását, és korlátozni fogja a teljesítőképességet. Ahogy az internet egyre gyorsabb lett, a TCP-összeköttetések forgalomszabályozási ablakának alapértelmezett 64 KB-os mérete is hasonló problémákat okozott.

4 Az eredeti angol szakirodalomban „long fat networks", rövidítve LFN, amely angol kiejtés szerint hasonlít az „elephant" kiejtésére. Így a magyar fordítás elefánt lett. *(A fordító megjegyzése)*

Egy másik beállításokkal kapcsolatos kérdés az időzítések megfelelő értékének meghatározása. Amikor egy szegmenst elküldenek, rendszerint elindítanak egy időzítőt, ami a szegmens elvesztését figyeli. A túl rövid időtartam fölösleges újraküldésekhez vezet, terheli a vonalakat. Ha túl hosszúra állítják, egy szegmens elvesztését túlságosan nagy késleltetések követik. További példák a beállítható paraméterekre: mennyi ideig kell adatra várni, hogy ráültetett nyugtát lehessen küldeni különálló nyugta helyett, vagy mennyi az újraküldések száma, mielőtt a küldő az ismétlést föladná.

Egy másik teljesítőképességgel kapcsolatos probléma, a dzsitter, a valós idejű alkalmazásoknál – mint például hang- és videoátvitel – lép föl. A jó teljesítőképességnek nem elégséges feltétele az elegendő sávszélesség, hanem az átviteli idők rövidsége is fontos.[5] Komoly tervezői erőfeszítést igényel a késleltetések folyamatosan alacsony szinten tartása, valamint a szolgáltatásminőség biztosítása az adatkapcsolati és a hálózati rétegben.

6.6.2. A hálózati teljesítőképesség mérése

Amikor egy hálózat gyengén teljesít, a felhasználók gyakran panaszkodnak a működtetőknek, és fejlesztést követelnek. A teljesítőképesség javításához az operátoroknak először is meg kell állapítaniuk, hogy mi is történik pontosan, ehhez azonban méréseket kell végezniük. Ebben a részben a hálózatok teljesítőképességének mérését mutatjuk be. Az alábbiakban leírtak Mogul munkáján [1993] alapulnak.

A méréseket (mind fizikailag és a protokollkészlet minden szintjén) sokféleképpen és sok helyen lehet végezni. A legalapvetőbb mérés az időmérés, amikor valamilyen tevékenység kezdetén egy időzítőt indítunk, amellyel megmérjük, hogy az adott tevékenység mennyi időt vesz igénybe. Például annak ismerete, hogy mennyi időt igényel egy szegmens nyugtázása, kulcsfontosságú. Más mérések számlálók felhasználásával lehetségesek, ezek valamely esemény gyakoriságát rögzítik (például elvesztett szegmensek száma). Végül gyakran fontos tudni valaminek a mennyiségét, például egy bizonyos időintervallum alatt feldolgozott bájtok számát.

A hálózat teljesítőképességének és paramétereinek mérése számos potenciális buktatót rejt. A következőkben néhányat fölsorolunk. Minden, a hálózati teljesítőképesség mérésére tett szisztematikus próbálkozás során gondosan el kell kerülni ezeket.

Győződjünk meg, hogy elég nagy-e a minták száma!

Ne egyetlen szegmens elküldésének idejét mérjük, hanem ismételjük meg a mérést mondjuk egymilliószor és vegyük az átlagot. A betöltési periódusban (például egy 802.16 NIC, vagy egy kábelmodem egy tétlen időszak után megkapja a lefoglalt sávszélességet) az első szegmens lassan haladhat, és a sorbaállás is szórást okoz. Nagyszámú minta vizsgálata a mért átlag és szórás bizonytalanságát is csökkenti. Ezt a bizonytalanságot statisztikai képletekkel határozhatjuk meg.

5 Sőt a késleltetések szórása (a dzsitter) is meghatározó tényező. *(A fordító megjegyzése)*

Bizonyosodjunk meg arról, hogy reprezentatív mintákat használunk!

Ideális esetben az egymillió mérést különböző napszakokban és a hét napjain meg kellene ismételni, hogy láthassuk a különböző rendszerterheléseknek a mért mennyiségre gyakorolt hatását. A torlódáson végzett mérések például kevés haszonnal járnak, ha a mérés pillanatában nincs is torlódás. Néha az eredmények első ránézésre valószínűtlennek tűnhetnek, mint például súlyos torlódás délelőtt 11 órakor és délután 1 órakor, viszont nincs torlódás délben (amikor az összes felhasználó ebédelni ment).

A jelterjedés hatásai miatt a vezeték nélküli hálózatok esetén a mérés pontos helye is nagyon fontos. Még ha egy vezeték nélküli kliens mellé is helyezünk egy mérőpontot, az antennák különbözősége miatt még akkor sem biztos, hogy ugyanazt a csomagot fogják látni. Ebben az esetben a méréseket legjobb magán a kliensen végezni, hogy lássuk, mit lát. Ha ez nem sikerül, akkor alkalmazhatunk olyan módszert, amellyel kombinálni lehet a különböző, előnyös pontokon történt mérések eredményeit, hogy részletesebb képet kapjunk, mi is történik a hálózaton [Mahajan és mások, 2006].

A gyorstár működése romba döntheti a mérést

Egy mérés többszöri megismétlése esetén nagyon gyors válaszokat kaphatunk, ha a protokoll gyorstárat használ. Például, egy weboldal letöltése vagy egy DNS-név feloldása (hogy megtaláljuk az IP-címet) úgy történhet, hogy a kliens először a hálózathoz fordul válaszért, a továbbiakban pedig egy helyi gyorstárból kapja meg a választ a kérésére anélkül, hogy egyetlen csomagot is elküldene a hálózaton keresztül. Az ilyen mérések eredményei teljességgel értéktelenek (hacsak nem a gyorstár teljesítőképességét kívánjuk mérni).

A pufferelésnek hasonló hatása lehet. TCP/IP-teljesítőképességet vizsgáló tesztek arról voltak ismertek, hogy az UDP teljesítőképességére a fizikai vonal által megengedettnél jóval nagyobb értéket közöltek. Hogy történhetett ez meg? Egy UDP-hívás normál esetben akkor adja vissza a vezérlést, amikor az üzenetet elfogadta a rendszermag, és beillesztette a továbbítási sorba. Ha elegendő pufferterület van, 1000 UDP-hívás nem jár együtt az összes adat tényleges továbbításával. A legtöbb üzenet esetleg még mindig a magban várakozik, de a teljesítőképességet mérő program úgy gondolja, hogy már mindet elküldték.

Kellő odafigyeléssel meg kell győződni tehát arról, hogy teljesen megértettük-e, hogyan puffereli vagy gyorstárazza a hálózati művelet az adatokat.

Győződjünk meg arról, hogy mérés közben nem következik be váratlan esemény!

Ha a méréseket éppen akkor végezzük, amikor egy felhasználó egy videokonferenciát folytat a mérendő hálózaton, akkor a mérési eredmények egészen más képet fognak mutatni, mintha nem folyna videokonferencia. A legjobb tétlen rendszeren végezni a méréseket, és a teljes terhelést saját kezűleg generálni, azonban még ennek a megközelítésnek is akadnak buktatói. Amikor úgy gondolnánk, hogy senki sem fogja használni a hálózatot éjjel háromkor, éppen ekkor foghat hozzá egy önműködő másolatkészítő

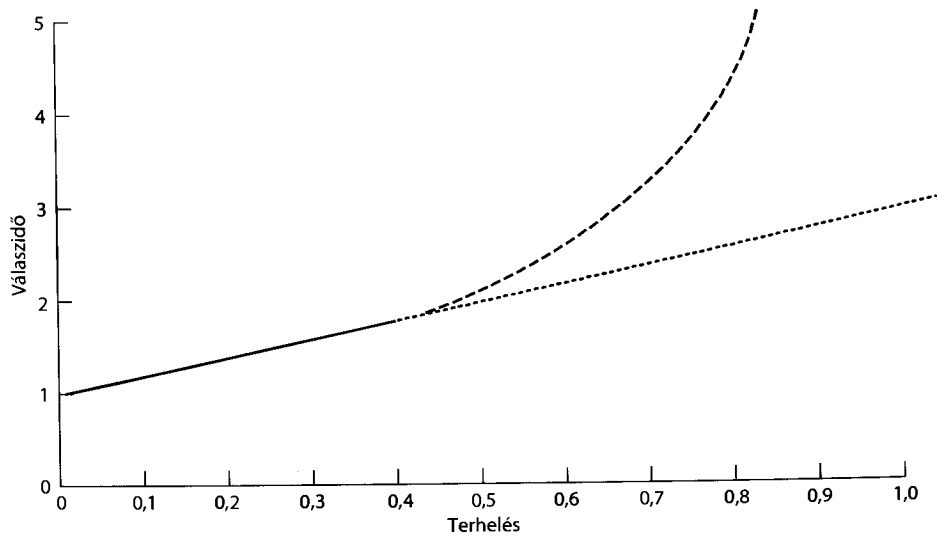
program az összes merevlemez szalagra másolásához. Továbbá erős forgalmat generálhatnak a csodálatos weboldalainkat nézegető más időzónában élő felhasználók.

A vezeték nélküli hálózatok ebben a tekintetben komoly kihívásokat tartogatnak, hiszen általában nem lehet az interferencia-forrásokat elkülöníteni. Még ha nincs is más vezeték nélküli hálózat, mely a közelben forgalmat bonyolít le, akkor is bárki dönthet úgy, hogy pattogatott kukoricát készít mikrohullámú sütőben, és ugyan nem szándékosan, de a 802.11 teljesítőképességét rontó interferenciát okoz. Ilyen okok miatt célszerű a teljes hálózati tevékenységet figyelni, hogy legalább észrevegyük, ha valami váratlan esemény történik.

Bánjunk óvatosan a durva felbontású órával!

A számítógépes órák úgy működnek, hogy szabályos időközönként egy számláló értékét eggyel növelik. Például egy ezredmásodperces időzítő ezredmásodpercenként egyet ad a számláló értékéhez. Ilyen időzítő felhasználásával nem lehetetlen 1 ms-nál rövidebb eseményt mérni, csak nagy gondosságot igényel. Egyes számítógépeknek ennél pontosabb órái vannak, de mindig léteznek rövidebb események, amelyeket meg kell mérnünk. Vegyük figyelembe továbbá, hogy az órák nem feltétlenül olyan pontosak, mint az a felbontás, amelyet az óra leolvasásakor kapunk!

Hogy például egy TCP-összeköttetés felépítéséhez szükséges időt megmérjük, kétszer kell leolvasni az órát (mondjuk ezredmásodpercekben), amikor belépünk a szállítási réteg kódjába, és amikor elhagyjuk azt. Ha az összeköttetés felépítésének tényleges ideje 300 μs, a két olvasás különbsége 0 vagy 1 lesz, ami egyaránt rossz. Ha azonban a teljes mérést egymilliószor megismételjük, és az összes mérési eredményt összeadjuk, majd egymillióval elosztjuk, az átlagos idő 1 μs-nél is pontosabb lesz.



6.49. ábra. *A válaszidő a terhelés függvényében*

Vigyázzunk az eredmények extrapolálásával!

Tegyük fel, hogy valamit szimulált hálózati terhelésnél mérünk. A terhelés 0 (tétlen) értéktől 0,4-re (a kapacitás 40 százaléka) nő. Például egy VoIP-csomag válaszideje, ha egy 802.11 hálózaton küldjük át, a 6.49. ábrán látható pontokhoz és a rájuk fektetett folytonos vonalhoz fog hasonlítani. Nehéz ilyenkor ellenállni a kísértésnek, hogy lineárisan extrapoláljuk, amit a pontozott vonal mutat. A sorbaállás tipikusan behoz azonban egy $1/(1-\rho)$ tényezőt is, ahol a ρ a terhelés, ezért az igazi eredmény inkább a szaggatott görbére emlékeztet, amely a terhelés növekedésével a lineárisnál valamivel jobban növekszik. Tehát óvatosan bánjuk olyan versengéses esetekkel, amelyek nagy terhelés esetén hangsúlyosabbá válnak.

6.6.3. Hoszt tervezése gyors hálózatokhoz

Mérésekkel és a folytonos javítgatással jelentős mértékben növelhető a teljesítőképesség, de ez nem helyettesítheti az elsődleges fontossággal bíró gondos tervezést. Egy gyengén megtervezett hálózat csak ideig-óráig javítgatható, végül az alapoktól kezdve kell újratervezni.

Ebben a részben néhány ökölszabályt mutatunk be a hálózati protokollok hosztokon történő megvalósításával kapcsolatban. Meglepő módon a tapasztalat azt mutatja, hogy gyakran ez a szűk keresztmetszet az egyébként gyors hálózatokon, két okból is. Egyrészt a NIC-eket (Network Interface Card – hálózati illesztőkártya) és az útválasztókat már (hardvertámogatással) a „vezeték sebességére" tervezték. Ez azt jelenti, hogy a csomagokat olyan gyorsan tudják feldolgozni, mint amilyen gyorsan egyáltalán azok az adatkapcsolatról beérkezhetnek. Másodsorban a tényleges teljesítőképesség az, amit az alkalmazások kapnak. Ez pedig nem az adatkapcsolat kapacitása, hanem a hálózati és a szállítási feldolgozás utáni átbocsátóképesség és a késleltetés.

A szoftver többletmunkájának csökkentése az átbocsátóképesség növekedését és a késleltetés csökkentését segíti elő. Emellett csökkenti azt az energiafelhasználást, amely a hálózaton töltött idő alatt elvész. Ez fontos érv a mobil számítógépek esetében. Ezen ötletek nagy része évek óta köztudott a hálózattervezők között. Először Mogul [1993] rögzítette őket; a mi megközelítésünk nagyjából párhuzamos az övével. Egy másik idekapcsolódó forrás Metcalfe [1993] műve.

A hoszt sebessége fontosabb, mint a hálózat sebessége

Hosszú idők tapasztalata szerint szinte minden gyors hálózatban az operációs rendszer és a protokoll többletterhelése határozza meg a vonalon töltött idő pillanatnyi hosszát. Például, elméletileg egy 1 Gbit/s Etherneten eltöltött legrövidebb RPC végrehajtási idő 1 μs, ami megfelel egy minimális (512 bájtos) kérésnek és az azt követő minimális (szintén 512 bájtos) válasznak. A gyakorlatban jelentős eredménynek számít, ha a szoftver többletterhének leszorításával az RPC idejére valamilyen ehhez közeli eredményt kapunk. A gyakorlatban azonban ez ritkán valósul meg.

Ehhez hasonlóan, az 1 Gbit/s sebességű átvitel legnagyobb problémája az, hogy a bitek elég gyorsan jussanak ki a felhasználói pufferből a hálózatra, valamint hogy a vételi folyamat olyan gyorsan dolgozza fel azokat, amilyen sebességgel beérkeznek. Ha megduplázzuk a hoszt (CPU- és memória-) sebességét, gyakran közel kétszeresére növelhetjük az átbocsátóképességet is. A hálózat kapacitásának megduplázása gyakran teljesen hatástalan, mivel a szűk keresztmetszetet általában a hosztok jelentik.

A többletterhelés csökkentéséhez csökkentsük a csomagok számát

Minden szegmens egy bizonyos mennyiségű többletadatot (például a fejrész) és hasznos adatot (például felhasználói adatot a szegmens törzsében) tartalmaz. Sávszélesség mindkettőhöz kell. Mindkét komponens feldolgozást igényel (például a fejrész feldolgozása és az ellenőrző összeg számítása). Ha egymillió bájtot küldünk el, az adatátvitel költsége a szegmens méretétől függetlenül ugyanannyi. 128 bájtos szegmens használata azonban 32-szer akkora szegmensenkénti többletráfordítást jelent, mint 4 kilobájtos szegmensek esetében. A sávszélesség és a feldolgozási többletterhek hamar lecsökkentik az átbocsátóképességet.

A csomagonkénti többletterhelés tovább növekedik az alsóbb rétegekben. Minden beérkező csomag megszakítást okoz, ha a hoszt tartani akarja a tempót. A modern csővezetékes (pipelined) processzor esetén minden megszakítás kiüríti a processzor csővezetékét, zavarja a gyorstárat, a memóriamenedzsmentben környezetváltást eredményez, kiüríti az feltételes elágazást előrejelző táblákat, és rengeteg CPU-regiszter elmentését kényszeríti ki. Egy n-szeres csökkenés az elküldött szegmensek számában n-ed részére csökkenti a megszakítás- és a csomagkezelés okozta többletterhet.

Erre azt mondhatnánk, hogy sem az emberek, sem a számítógépek nem hatékonyak több feladat egyidejű elvégzésében. Ez a megfigyelés az alapja annak, hogy lehetőleg akkora (MTU méretű) csomagokat küldjünk, hogy azt a hálózat még éppen ne tördelje szét. Az olyan mechanizmusok, mint a Nagle-féle algoritmus és Clark megoldása a buta ablak jelenségre, kísérletek a kisméretű csomagok küldésének elkerülésére.

Ritkán nyúljunk az adatokhoz

A rétegezett protokollkészlet megvalósítására a legkézenfekvőbb módszer az, ha minden réteget egy modul képvisel. Ez sajnos az adatok gyakori másolásához vezet (vagy legalábbis az adatok többszöri hozzáféréséhez), ahogy a rétegek teszik a dolgukat. Például amikor egy csomag megérkezik a NIC-hez, azt tipikusan a rendszermag egy pufferébe másolják. Onnan a hálózati réteg pufferébe kerül feldolgozásra, majd a szállítási réteg pufferébe további feldolgozásra, és végül megkapja az alkalmazási folyamat. Nem szokatlan, hogy egy csomagot háromszor vagy négyszer átmásolnak, mielőtt a benne található szegmenst a címzettnek kézbesítik.

Az összes másolás erőteljesen ronthatja a teljesítőképességet, hiszen a memóriaműveletek egy nagyságrenddel lassabbak, mint a regiszter-regiszter műveletek. Például ha a műveletek 20%-a a memóriára irányul (mert mondjuk a gyorstár nem találja az ada-

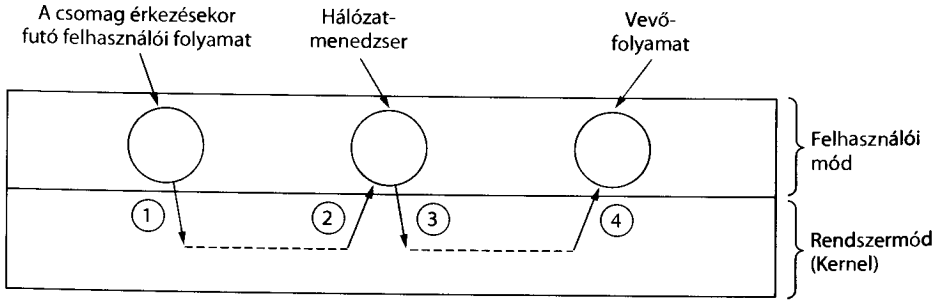
tot), ami elég valószínű a bejövő csomagok esetében, az átlagos utasítás végrehajtási idő 2,8-as szorzóval megnő. A hardvergyorsítás itt nem segít. A gond az, hogy az operációs rendszer túl sokat másol.

Egy okos operációs rendszer a másolások számát a többrétegű feldolgozással csökkenti. A TCP- és IP-rétegeket például gyakran valósítják meg egyben (mint „TCP/IP"-t), így az adatok másolása elkerülhető, ha a feldolgozást a hálózati réteg után a szállítási réteg folytatja. Másik gyakori trükk, hogy a rétegen belül több műveletet végeznek az adatokon egy menetben. Például az ellenőrző összegeket az adatok másolása közben számítják (akkor, amikor egyébként is adatot kellene másolni), és a kiszámolt ellenőrző összeget az adatok végéhez csatolják.

Minimalizáljuk a környezetváltások számát

Az előbbiekhez kapcsolódó szabály az, hogy a környezetváltások (például rendszer módból felhasználói módba) életveszélyesek. Ugyanazok a rossz tulajdonságaik vannak, mint a megszakításoknak és a másolásoknak. Ezért valósítják meg gyakran a szállítási protokollokat is a rendszermagban. Ahogy a csomagok számát, úgy a környezetváltások számát is csökkenthetjük, ha az adatokat küldő könyvtári eljárást addig kényszerítjük adatok gyűjtésére, amíg jelentős mennyiségű össze nem gyűlt. Hasonlóképpen a vevőoldalon a kicsi beérkező szegmenseket össze kell gyűjteni, és az egyenként történő átadás helyett egyetlen mozdulattal át lehet tolni a felhasználónak, hogy minimalizálni lehessen a környezetváltások számát.

A legjobb esetben egy beérkező csomag környezetváltást okoz az aktuális felhasználóról a rendszermódra, majd a vevőfolyamatra, hogy az beolvashassa a frissen érkezett adatokat. Sajnos sok operációs rendszer esetében még további környezetváltások is történnek. Például ha a hálózatmenedzser speciális folyamatként felhasználói módban fut, egy csomag érkezése valószínűleg egy környezetváltást okoz az aktuális felhasználóról a rendszermódra, majd onnan a hálózatmenedzserre, ezt követően egy újabb környezetváltás történik vissza a rendszermódra, és végül egy utolsó vissza a vevőfolyamatra. Ezt a sorozatot láthatjuk a 6.50. ábrán. Ezek a csomag érkezésekor bekövetkező környezetváltások nagyon sok processzoridőt elpazarolnak és lerontják a hálózat teljesítőképességét.



6.50. ábra. *Egy csomag felhasználói módban futó hálózatmenedzserrel történő lekezeléséhez szükséges négy környezetváltás*

Könnyebb elkerülni a torlódást, mint abból kikerülni

A régi mondás, miszerint egy szem megelőzés felér egy véka gyógyítással, biztosan igaz a hálózati torlódásokra is. Amikor torlódás lép fel a hálózaton, csomagok vesznek el, sávszélesség megy kárba, haszontalan késleltetések lépnek föl és így tovább. Mindezek az áldozatok szükségtelenek és a torlódás helyreállítása időt és türelmet igényel. Legjobb, ha nem hagyjuk, hogy kialakuljon. A torlódás elkerülése olyan, mint a védőoltás: egy kicsit fáj, amikor beadják, de megelőz valamit, ami sokkal fájdalmasabb lenne egy későbbi időpontban.

Kerüljük el az időtúllépéseket

Az időzítők szükségesek a hálózatban, de módjával kell használni őket, és az időtúllépések számát minimalizálni kell. Amikor egy időzítő lejár, általában egy tevékenység megismétlődik. Ha tényleg szükség van az ismétlésre, ám legyen, de a fölösleges ismétlés pazarlás.

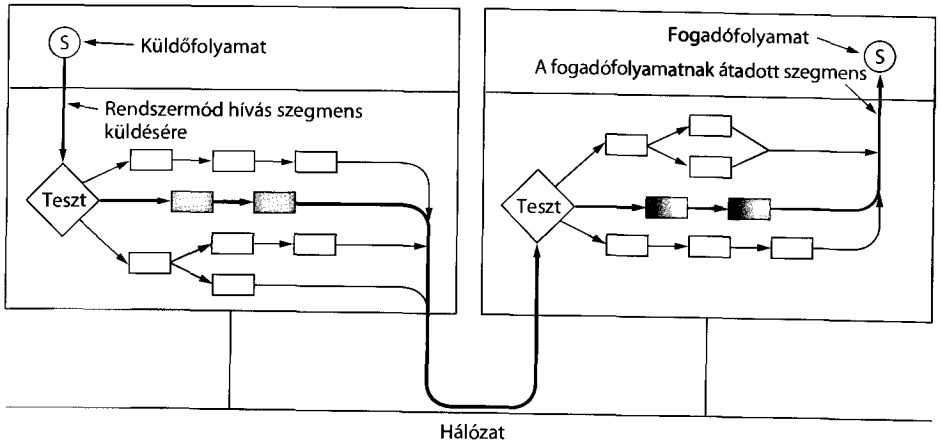
A pluszmunka elkerülésének az a módja, hogy az időzítőket gondosan, inkább egy kicsit hosszabbra állítjuk. Egy időzítő, ami hosszabb ideig működik, egy kis extra késleltetést generál az összeköttetésen abban a (valószínűtlen) esetben, ha egy szegmens elvész. Az az időzítő, ami lejár, amikor még nem kellene, értékes processzoridőt használ fel, sávszélességet pazarol, és fölösleges többletterhelést okoz akár több tucat útválasztónak is.

6.6.4. Gyors szegmensfeldolgozás

Most, hogy pár általános szabályon keresztülrágtuk magunkat, megtekintünk néhány különleges módszert a szegmensek gyors feldolgozására. További információért lásd Clark és mások [1989], valamint Chase és mások [2001] munkáját.

A szegmens feldolgozási többletterhelésének két összetevője van: a szegmensenkénti és a bájtonkénti többletteher. Mindkettő ellen harcolni kell. A gyors szegmensfeldolgozás kulcsa az, hogy válasszuk külön a normális, sikeres esetet (egyirányú adatátvitel), és kezeljük különleges módon. Sok protokoll azt hangsúlyozza, mit kell tenni akkor, ha valami gond van (például elveszett egy csomag), azonban ahhoz, hogy egy protokollt gyorssá tegyünk, a fejlesztőnek csökkentenie kell a feldolgozási időt akkor, amikor minden rendben megy. A feldolgozási idő csökkentése hiba esetén másodlagos.

Bár egy sor speciális szegmens szükséges ahhoz, hogy eljussunk az *ESTABLISHED* állapotba, ha már ott vagyunk, a szegmensek feldolgozása nyilvánvaló, amíg egyik fél nem kezdeményezi az összeköttetés bontását. Kezdjük a tanulmányozást az *ESTABLISHED* állapotban levő küldő oldalán, amikor van átküldésre váró adat. Az egyértelműség kedvéért feltételezzük, hogy a szállítási entitás a rendszermag része, bár ugyanezeket az ötleteket lehet alkalmazni akkor is, ha ez egy felhasználói folyamat, vagy ha egy könyvtári eljárás a küldőfolyamatban. A 6.51. ábrán a küldőfolyamat SEND rendszermódhívást ad ki. Az első dolog, amit a szállítási entitás csinál, hogy megvizsgálja, a normális esetről van-e szó: az állapot *ESTABLISHED*, egyik oldal se próbálja bontani az összeköttetést,

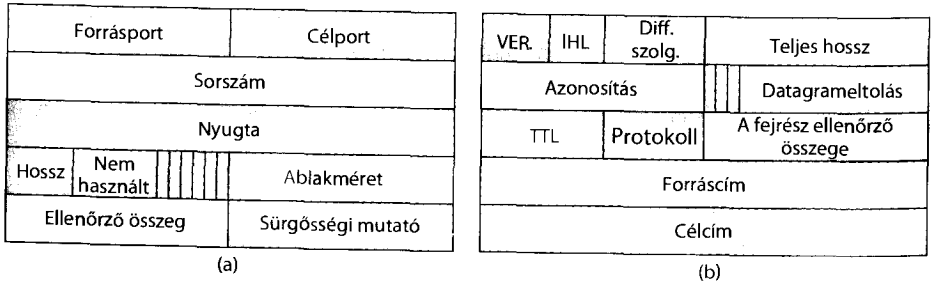


6.51. ábra. *A küldőtől a vevőig vezető gyors utat vastag vonal jelzi. Ezen az úton a feldolgozási lépések sötétítve látszanak*

egy szabályos (azaz nem sávon kívüli) teljes szegmenst küldenek, és a vevőnél elég nagy ablak áll rendelkezésre. Ha minden feltétel teljesül, nincs szükség további tesztekre, és a gyors utat lehet követni a szállítási entitáson belül. A legtöbb esetben a szegmensek többnyire ezt az útvonalat követik.

Normális esetben az egymást követő szegmensek fejrészei majdnem azonosak. Hogy ezt a tényt kiaknázhassa, a szállítási entitás egy fejrészprototípust tárol. A gyors út elején, amilyen gyorsan lehet, ezt szavanként egy átmeneti pufferbe másolja. Azokat a mezőket, amelyek szegmensről szegmensre változnak, a pufferben fölülírja. Ezek a mezők – mint például a sorszám – gyakran egyszerűen származtathatók az állapotváltozókból. Ezután átad a hálózati rétegnek két, a teljes hálózati fejrészre, illetve a felhasználói adatra mutató mutatót. Itt ugyanezt a stratégiát lehet követni (a 6.51. ábrán nem tüntettük fel). Végül a hálózati réteg átadja továbbításra a csomagot az adatkapcsolati rétegnek.

Hogy lássuk az elmélet gyakorlati működését, vegyük például a TCP/IP-t. A 6.52.(a) ábrán a TCP-fejrészt láthatjuk. Azokat a mezőket, amelyek az egyirányú folyamban egymás után következő szegmensekben azonosak, besötétítettük. A küldő szállítási entitás



6.52. ábra. *(a) TCP-fejrész. (b) IP-fejrész. Mindkét esetben a sötétített mezőket változtatás nélkül veszik a prototípusból*

tennivalója mindössze az, hogy öt szót átmásoljon a fejrész prototípusból a kimeneti pufferbe, kitöltse a *sorszám* mezőt (egy szó bemásolása a memóriából), kiszámítsa az ellenőrző összeget és növelje a memóriában a sorszám értékét. Ez után átadhatja a fejrészt és az adatot egy speciális IP-eljárásnak, hogy az továbbítson egy szabályos, teljes méretű szegmenst. Az IP ezt követően átmásolja a saját ötszavas fejrész prototípusát (6.52.(b) ábra) a pufferbe, kitölti az *Azonosítás* mezőt és kiszámítja a saját ellenőrző összegét. A csomag most átvitelre kész.

Most vizsgáljuk meg a vevőoldalon alkalmazott, a 6.51. ábrán is látható gyors feldolgozást. Az első lépés az, hogy meg kell találni a bejövő szegmenshez tartozó összeköttetés-rekordot. A TCP esetében az összeköttetés-rekordokat egy asszociatív tömbben vagy hash-táblában (hash table) tárolhatjuk, a két IP-cím és a két port valamely egyszerű függvénye szerint rendezve. Miután a TCP-entitás megtalálta az összeköttetés-rekordot, mindkét címet és mindkét portot összehasonlítja a rekordban találhatókkal, így ellenőrizve azt, hogy a megfelelő rekordot találta-e meg.

Egy, a legutóbb használt rekordra irányított mutató gyakran tovább gyorsíthatja az összeköttetés-rekord előkeresését, ha először mindig azzal próbálkozik a TCP. Clark és munkatársai [1989] kipróbálták ezt, és 90 százalékosnál magasabb találati arányt tapasztaltak.

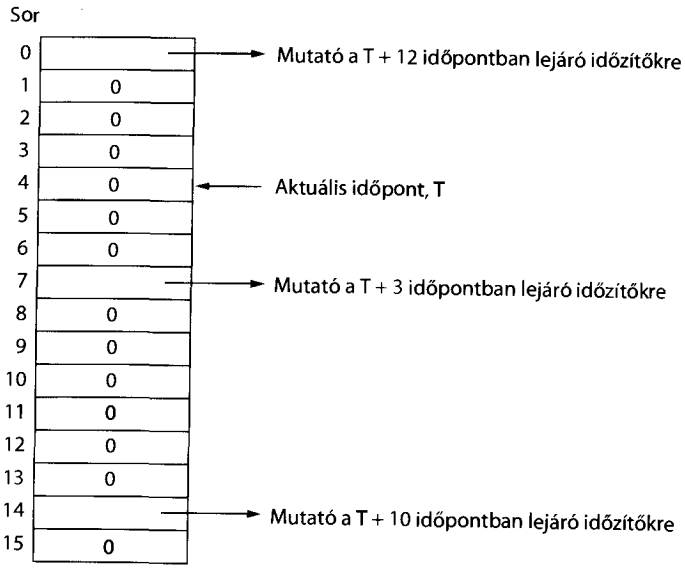
Ezután az entitás megvizsgálja, hogy normális szegmens érkezett-e: az állapot *ESTABLISHED*, egyik oldal se próbálja bontani az összeköttetést, a szegmens maximális méretű, nincs speciális jelzése, és a sorszám egyezik a várt értékkel. Ezek a vizsgálatok csak néhány utasítást igényelnek. Ha minden feltétel teljesül, a vezérlés egy speciális gyors út eljárásra kerül a TCP-n belül.

A gyors út rutin frissíti az összeköttetés-rekordot és átmásolja az adatot a felhasználóhoz. Másolás közben ellenőrző összeget is számít, így elkerülhető egy további adatfeldolgozási menet. Ha az ellenőrző összeg rendben van, frissíti az összeköttetés-rekordot, és visszaküld egy nyugtát. Azt az általános módszert, amelyik először gyors ellenőrzéssel megbizonyosodik arról, hogy a várt fejrész érkezett-e meg, majd az eset kezelését egy speciális eljárás végzi, **fejrész-előrejelzésnek (header prediction)** nevezik. Sok TCP-megvalósítás ezt alkalmazza. Ha ezt, és az itt leírt többi optimalizációt együtt használják, elérhető, hogy a TCP a helyi memóriából-memóriába másolás sebességének 90 százalékával fusson, feltéve, hogy maga a hálózat elég gyors.

Két másik terület, ahol a teljesítőképesség jelentős javulása érhető el, a pufferkezelés és az időzítőkezelés. A pufferkezelés trükkje – mint fent említettük – a fölösleges másolás elkerülése. Az időzítőkezelés fontos, mert szinte egyetlen elindított időzítő sem jár le. A szegmensek elvesztését figyelik, de a legtöbb szegmens és visszaküldött nyugta rendben megérkezik. Ezért fontos arra optimalizálni az időzítők kezelését, hogy ritkán járnak le.

Általánosan használt módszer az, hogy lejárat szerint sorba rendezve, láncolt listában tároljuk az időzítőeseményeket. A lista feje egy számlálót tartalmaz, ami azt mutatja, hogy mostantól számítva hány óraütés múlva jár le a hozzá rendelt időzítő. Az egymást követő bejegyzések azt tartalmazzák, hogy az egymás után következő időzítők az előzőtől számítva mikor járnak le. Így ha az időzítők 3, 10 és 12 ütem múlva járnak le, a három számláló értéke rendre 3, 7 és 2 lesz.

Minden óraütéskor az entitás csökkenti a lista fejének számlálóját. Amikor eléri a 0-t, a hozzá tartozó eseményt feldolgozza és a következő elem lesz a lista feje. Ennek a szám-



6.53. ábra. *Az időkerék*

lálóját nem kell módosítani. Ebben a megoldásban időzítők beszúrása és törlése drága művelet, aminek végrehajtási ideje arányos a lista hosszával.

Hatékonyabb megközelítés alkalmazható, ha a maximális időtartam értéke korlátos és előre ismert. Itt egy **időkereknek** (**timing wheel**) nevezett tömböt használunk, amit a 6.53. ábrán láthatunk. Minden sor egy óraütésnek felel meg. Az ábrán a $T = 4$ időpillanatot ábrázoltuk. Az időzítők mostantól számítva 3, 10 és 12 óraütés múlva járnak le. Ha hirtelen egy 7 ütem múlva lejáró új időzítőt kell beszúrni, csak egy új bejegyzést nyitunk a 11. sorban. Hasonlóan, ha a $T = 10$ időpontra beállított időzítőt törölni akarjuk, a 14. sorban kezdődő listát kell megkeresni és a kívánt elemét törölni. Vegyük észre, hogy a 6.53. ábrán látható tömb nem tud $T + 15$-nél későbbre állított időzítőket kezelni.

Amikor üt az óra, az aktuális idő mutatója egy sorral előremozdul (cirkulárisan). Ha a bejegyzés, amire most mutat, nullától különbözik, az ott lévő összes időzítő feldolgozásra kerül. Az alapötlet több variációját tárgyalja Varghese és Lauck [1987].

6.6.5. Fejrésztömörítés

Túl sokáig vizsgáltuk a gyors hálózatokat, más is létezik rajtuk kívül. Szemléljük meg most a vezeték nélküli és más, kis sávszélességű hálózatokat. A szoftvertöbbletteher csökkentése lehetővé teszi a hordozható eszközöknek, hogy hatékonyabban működjenek, azonban ez semmit sem ér, ha éppen a hálózat jelenti a szűk keresztmetszetet.

A sávszélesség hatékony kihasználása érdekében a fejrészeknek és a felhasználói adatoknak a lehető legkisebb helyet kell foglalniuk. Ami a felhasználói adatokat illeti, használhatunk tömörítő algoritmusokat, például JPEG-képformátumot bittérképek helyett vagy tömörítést biztosító dokumentumformátumokat, például PDF-et. Ide értjük az

alkalmazás szintjén történő gyorstárazási mechanizmusokat is, például a webes gyorstárakat, amelyek elsősorban az átvitelek számát csökkenti.

De mi legyen a protokollfejrészekkel? Az adatkapcsolati réteg szintjén, a vezeték nélküli hálózatokban használt fejrészek tipikusan kicsik, mivel a sávszélesség-takarékosság jegyében születtek. A 802.16. fejrészei például rövid összeköttetés-azonosítókat tartalmaznak hosszabb címek helyett. A felsőbb rétegekből azonban, mint az IP, a TCP és az UDP, minden adatkapcsolati rétegre csupán egyetlen változat létezik, és az általuk használt fejrészek sem rövidek. Ráadásul a szoftver túlterhelésének csökkentését szolgáló folyamszerű feldolgozás gyakran olyan fejrészeket eredményez, amelyek nem annyira teljesek, mint amilyenek máskülönben lehetnének (például az IPv6 sokkal lazábban csomagolt fejrészekkel rendelkezik, mint az IPv4).

A felsőbb rétegek fejrészei jelentősen befolyásolják a teljesítőképességet. Vegyünk például egy VoIP-adategységet, amely az IP-, UDP- és RTP-protokollok segítségével ér célt. Ezek a protokollok összesen 40 bájtos fejrészt igényelnek (az IPv4 20, az UDP 8, az RTP pedig 12 bájtot). IPv6 használatával a helyzet még rosszabb: a 40 bájtos IPv6-fejrésszel együtt összesen 60 bájt. Az átvitt adat mennyiségének nagy részét a fejrészek alkothatják, elfoglalva a sávszélesség több mint a felét.

A felsőbb rétegek fejrészei által elfoglalt sávszélesség csökkentésére használják a **fejrésztömörítést** (**header compression**). Az általános módszerek helyett kifejezetten speciális megoldásokat használnak, ugyanis a fejrészek rövidek, így nem valami jól tömöríthetők önmagukban, és a kibontás (decompression) minden korábbi adat vételét igényli. Csomagvesztés esetén ez az eset nem áll fenn.

A fejrésztömörítés erőteljesen kihasználja, hogy ismeri a protokoll által meghatározott adategység formátumát. Az első ilyen módszert Van Jacobson [1990] tervezte a TCP/IP-fejrészek tömörítésére lassú adatkapcsolatokon. A módszer képes egy tipikus TCP/IP-fejrészt 40 bájtról 3 bájtra tömöríteni. A megoldás által használt trükk a 6.52. ábra alapján kitalálható: a fejrészmezők nagy része nem változik csomagról csomagra. Nincs szükség például ugyanazt az IP TTL mezőt, vagy ugyanazokat a TCP-portszámokat minden csomagban elküldeni. Az összeköttetés küldő oldalán elhagyhatók, a vevő oldalán pedig azok kitölthetők.

Ehhez hasonlóan, a többi mező is kiszámítható módon változik. A csomagvesztést leszámítva például a TCP-sorszám a felhasználói adatokkal együtt növekszik. Ebben az esetben a vevő megjósolhatja a legvalószínűbb értékét. A tényleges számot csak akkor kell átküldeni, ha a várttól különbözik. Még ekkor is csak a kisebb méretű különbséget kell elküldeni az előző értékhez képest, amikor a nyugta értéke az ellenkező irányban vett adat vételével megnövekszik.

A fejrésztömörítés használatával lehetővé válik, hogy a felsőbb rétegek protokolljai egyszerű fejrészeket és tömörítő algoritmusokat használjanak kis sávszélességű adatkapcsolatokon. A **ROHC** (**RObust Header Compression – robusztus fejrésztömörítés**) a fejrésztömörítés egy modern változata, amelyet keretrendszerként definiáltak az RFC 5795-ben. Úgy tervezték, hogy a vezeték nélküli adatkapcsolatokon előforduló adatvesztéseket is elviselje. Minden tömörítendő protokollhalmazra létezik egy profil, mint amilyen például az IP/UDP/RTP hármasra is. A tömörített fejrészeket egy kontextusra való hivatkozással továbbítják, amely gyakorlatilag egy összeköttetés; az ugyanahhoz az összeköttetéshez tartozó csomagok fejrészmezőit könnyedén meg lehet jósolni, de egy

másik összeköttetéshez tartozó csomagok fejrészét nem. Normál esetben az ROHC az IP/UDP/RTP fejrész méretét 40 bájtról 1-3 bájtra csökkenti.

Bár a fejrésztömörítést elsősorban a sávszélességigény csökkentésére használják, hasznos lehet a késleltetés csökkentésére is. A késleltetés magába foglalja a jelterjedési késleltetést, ami egy hálózati útvonalon állandó, valamint a terjedési késleltetést, amely a sávszélességtől és a küldendő adatok mennyiségétől függ. Egy 1 Mbit/s sebességű adatkapcsolat 1 μs alatt 1 bitet küld. Abban az esetben, ha multimédiát továbbítunk vezeték nélküli hálózaton, a terjedési késleltetés fontos tényező a teljes késleltetésben, mivel a hálózat viszonylag lassú. Egyenletesen alacsony késleltetés pedig a szolgáltatásminőség szempontjából életbevágó.

A fejrésztömörítés az adatok mennyiségének csökkentésével segíthet a terjedési késleltetés csökkentésében. Ez a hatás elérhető kis csomagok küldésével is, ami azonban a lecsökkent átviteli késleltetéshez szükséges megnövekedett szoftver-többletteherrel jár. Vegyük észre, hogy a késleltetés másik forrása a vezeték nélküli hálózat eléréshez szükséges sorbanállási késleltetés. Ez ugyancsak jelentős lehet, hiszen a vezeték nélküli adatkapcsolatok leterheltek, mivel általában a szűk keresztmetszetet jelentik a hálózatban. A vezeték nélküli adatkapcsolatoknak ebben az esetben szolgáltatásminőségi eszközökkel kell biztosítania a kis késleltetést a valós idejű csomagok számára. A fejrésztömörítés önmagában nem elégséges.

6.6.6. Protokollok elefánthálózatokra

Az 1990-es évek eleje óta vannak jelen a gigabites hálózatok, amelyek nagy távolságokon bonyolítanak le forgalmat. Az eredeti angol kifejezés, a **long fat network**, a hosszú (long) késleltetésekre, és a gyors hálózatra („fat pipe" – vaskos, nagy átmérőjű csővezeték) utal.[6] Az emberek először ezeken is a régi protokollokat próbálták alkalmazni, de hamarosan számtalan megoldandó problémával kellett szembesülniük. Ebben a szakaszban az egyre növekvő sebességű és késleltetésű hálózatok protokolljainak problémáival fogunk foglalkozni.

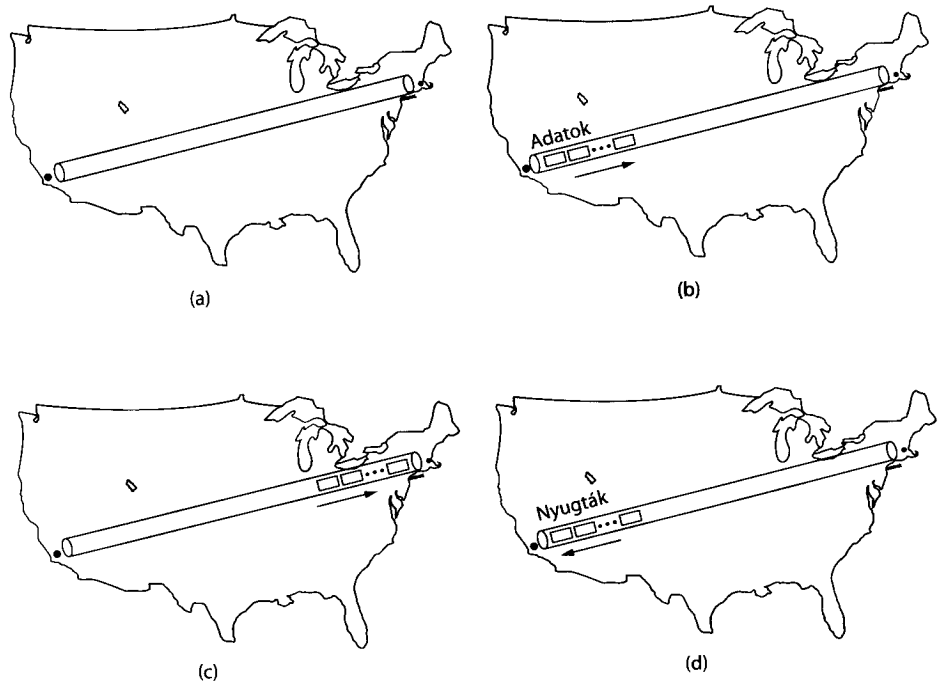
Az első probléma az, hogy sok protokoll 32 bites sorszámozást alkalmaz. Amikor az internet elkezdődött, az útválasztók közötti vonalak a legtöbb helyen 56 kb/s-os bérelt vonalak voltak, ezért egy teljes sebességgel adó hosztnak is több mint egy hét kellett ahhoz, hogy egyszer felhasználja az összes sorszámot. A TCP tervezői számára a 2^{32} még elég jól közelítette a végtelent, mivel kevés esély volt arra, hogy a régi csomagok még a hálózatban legyenek egy héttel a feladásuk után. Egy 10 Mb/s-os Ethernet esetében a körülfordulási idő 57 perc, ami sokkal rövidebb, de még kezelhető idő. Ha azonban egy 1 Gb/s-os Ethernet teljes sebességgel zúdít adatokat az internetre, akkor a körülfordulási idő körülbelül 34 másodperc, ami már jóval az interneten feltételezett 120 másodperces legnagyobb csomagélettartam alatt van. A 2^{32} hirtelen már nem közelíti jól a végtelent, mivel a küldő akkor is kimerítheti a sorszámokat, amikor a régi csomagok még a hálózaton tartózkodnak.

6 Épkézláb magyarítás híján az „elefánt" hálózat elnevezést használjuk, ugyanis rövidítését (LFT) gyakran „elephant" kiejtéssel használják. *(A fordító megjegyzése)*

Az a gond, hogy sok protokolltervező anélkül, hogy rögzítette volna, egyszerűen azt feltételezte, hogy az összes sorszám elhasználásához szükséges idő jóval nagyobb a maximális csomagélettartamnál. Következésképpen még gondolni sem kellett arra a problémára, hogy régi kettőzések még mindig létezhetnek, amikor a sorszámok körbefordulnak. Gigabites sebességnél ezek a ki nem mondott feltételezések kudarcot vallanak. Szerencsére az effektív sorszámméret növelése lehetővé vált úgy, hogy a minden csomagban jelenlévő TCP-fejrészopciók, az időbélyegek a sorszám felső bitjeiként szolgáljanak. Ezt a módszert **PAWS**-nak (**Protection Against Wrapped Sequence numbers – átfordult sorszámok elleni védelem**) hívjuk, és az RFC 1323 taglalja részletesen.

A második probléma, hogy a forgalomszabályozási ablak méretét nagymértékben növelni kell. Tekintsük például azt az esetet, amikor 64 kilobájt adatot küldünk San Diegóból Bostonba, hogy a vevő 64 kilobájtos pufferét megtöltsük. Tegyük fel, hogy a vonal sebessége 1 Gb/s és az üvegszálban a fénysebességből adódó késleltetés egy irányban 20 ms. Kezdetben $t = 0$-ban a csővezeték üres, ezt láthatjuk a 6.54.(a) ábrán. Alig 500 μs-mal később a 6.54.(b) ábrának megfelelően az összes szegmens úton van az üvegszálon. Az első szegmens most valahol Brawley magasságában járhat, még mindig mélyen Dél-Kaliforniában. A küldőnek azonban meg kell állnia, amíg ablakfrissítést nem kap.

20 ms elteltével az első szegmens eléri Bostont – mint azt a 6.54.(c) ábrán láthatjuk – és a vevő nyugtázza. Végül 40 ms-mal a kezdés után az első nyugta visszaér a küldőhöz, amely elküldheti a második löketet. Mivel az átviteli vonalat csak 1,25 ms ideig használ-



6.54. ábra. *Egy megabit továbbítása San Diegóból Bostonba. (a) t = 0-kor. (b) 500 μs múlva. (c) 20 ms elteltével. (d) 40 ms múlva*

másik összeköttetéshez tartozó csomagok fejrészét nem. Normál esetben az ROHC az IP/UDP/RTP fejrész méretét 40 bájtról 1-3 bájtra csökkenti.

Bár a fejrésztömörítést elsősorban a sávszélességigény csökkentésére használják, hasznos lehet a késleltetés csökkentésére is. A késleltetés magába foglalja a jelterjedési késleltetést, ami egy hálózati útvonalon állandó, valamint a terjedési késleltetést, amely a sávszélességtől és a küldendő adatok mennyiségétől függ. Egy 1 Mbit/s sebességű adatkapcsolat 1 μs alatt 1 bitet küld. Abban az esetben, ha multimédiát továbbítunk vezeték nélküli hálózaton, a terjedési késleltetés fontos tényező a teljes késleltetésben, mivel a hálózat viszonylag lassú. Egyenletesen alacsony késleltetés pedig a szolgáltatásminőség szempontjából életbevágó.

A fejrésztömörítés az adatok mennyiségének csökkentésével segíthet a terjedési késleltetés csökkentésében. Ez a hatás elérhető kis csomagok küldésével is, ami azonban a lecsökkent átviteli késleltetéshez szükséges megnövekedett szoftver-többletteherrel jár. Vegyük észre, hogy a késleltetés másik forrása a vezeték nélküli hálózat eléréshez szükséges sorbanállási késleltetés. Ez ugyancsak jelentős lehet, hiszen a vezeték nélküli adatkapcsolatok leterheltek, mivel általában a szűk keresztmetszetet jelentik a hálózatban. A vezeték nélküli adatkapcsolatoknak ebben az esetben szolgáltatásminőségi eszközökkel kell biztosítania a kis késleltetést a valós idejű csomagok számára. A fejrésztömörítés önmagában nem elégséges.

6.6.6. Protokollok elefánthálózatokra

Az 1990-es évek eleje óta vannak jelen a gigabites hálózatok, amelyek nagy távolságokon bonyolítanak le forgalmat. Az eredeti angol kifejezés, a **long fat network**, a hosszú (long) késleltetésekre, és a gyors hálózatra („fat pipe" – vaskos, nagy átmérőjű csővezeték) utal.[6] Az emberek először ezeken is a régi protokollokat próbálták alkalmazni, de hamarosan számtalan megoldandó problémával kellett szembesülniük. Ebben a szakaszban az egyre növekvő sebességű és késleltetésű hálózatok protokolljainak problémáival fogunk foglalkozni.

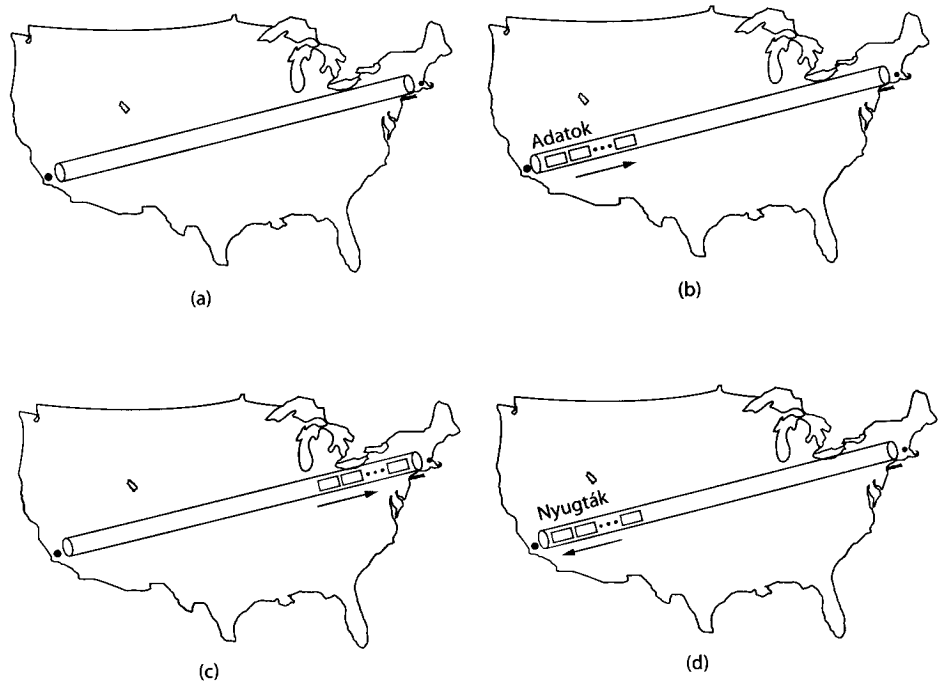
Az első probléma az, hogy sok protokoll 32 bites sorszámozást alkalmaz. Amikor az internet elkezdődött, az útválasztók közötti vonalak a legtöbb helyen 56 kb/s-os bérelt vonalak voltak, ezért egy teljes sebességgel adó hosztnak is több mint egy hét kellett ahhoz, hogy egyszer felhasználja az összes sorszámot. A TCP tervezői számára a 2^{32} még elég jól közelítette a végtelent, mivel kevés esély volt arra, hogy a régi csomagok még a hálózatban legyenek egy héttel a feladásuk után. Egy 10 Mb/s-os Ethernet esetében a körülfordulási idő 57 perc, ami sokkal rövidebb, de még kezelhető idő. Ha azonban egy 1 Gb/s-os Ethernet teljes sebességgel zúdít adatokat az internetre, akkor a körülfordulási idő körülbelül 34 másodperc, ami már jóval az interneten feltételezett 120 másodperces legnagyobb csomagélettartam alatt van. A 2^{32} hirtelen már nem közelíti jól a végtelent, mivel a küldő akkor is kimerítheti a sorszámokat, amikor a régi csomagok még a hálózaton tartózkodnak.

6 Épkézláb magyarítás híján az „elefánt" hálózat elnevezést használjuk, ugyanis rövidítését (LFT) gyakran „elephant" kiejtéssel használják. *(A fordító megjegyzése)*

Az a gond, hogy sok protokolltervező anélkül, hogy rögzítette volna, egyszerűen azt feltételezte, hogy az összes sorszám elhasználásához szükséges idő jóval nagyobb a maximális csomagélettartamnál. Következésképpen még gondolni sem kellett arra a problémára, hogy régi kettőzések még mindig létezhetnek, amikor a sorszámok körbefordulnak. Gigabites sebességnél ezek a ki nem mondott feltételezések kudarcot vallanak. Szerencsére az effektív sorszámméret növelése lehetővé vált úgy, hogy a minden csomagban jelenlévő TCP-fejrészopciók, az időbélyegek a sorszám felső bitjeiként szolgáljanak. Ezt a módszert **PAWS**-nak (**Protection Against Wrapped Sequence numbers – átfordult sorszámok elleni védelem**) hívjuk, és az RFC 1323 taglalja részletesen.

A második probléma, hogy a forgalomszabályozási ablak méretét nagymértékben növelni kell. Tekintsük például azt az esetet, amikor 64 kilobájt adatot küldünk San Diegóból Bostonba, hogy a vevő 64 kilobájtos pufferét megtöltsük. Tegyük fel, hogy a vonal sebessége 1 Gb/s és az üvegszálban a fénysebességből adódó késleltetés egy irányban 20 ms. Kezdetben $t = 0$-ban a csővezeték üres, ezt láthatjuk a 6.54.(a) ábrán. Alig 500 μs-mal később a 6.54.(b) ábrának megfelelően az összes szegmens úton van az üvegszálon. Az első szegmens most valahol Brawley magasságában járhat, még mindig mélyen Dél-Kaliforniában. A küldőnek azonban meg kell állnia, amíg ablakfrissítést nem kap.

20 ms elteltével az első szegmens eléri Bostont – mint azt a 6.54.(c) ábrán láthatjuk – és a vevő nyugtázza. Végül 40 ms-mal a kezdés után az első nyugta visszaér a küldőhöz, amely elküldheti a második löketet. Mivel az átviteli vonalat csak 1,25 ms ideig használ-



6.54. ábra. *Egy megabit továbbítása San Diegóból Bostonba. (a) t = 0-kor. (b) 500 μs múlva. (c) 20 ms elteltével. (d) 40 ms múlva*

ták a 100 ms-ból, a hatékonyság körülbelül 1,25 százalék. Ez a helyzet tipikusan akkor fordul elő, ha régi protokollokat használnak gigabites vonalak fölött.

Egy hasznos mennyiséget érdemes megjegyezni, ha számítógép-hálózatok teljesítőképességének vizsgálatával foglalkozunk. Ez a **sávszélesség-késleltetés szorzat** (**bandwidth-delay product**), ami úgy áll elő, hogy megszorozzuk a bit/másodpercben mért sávszélességet a másodpercben mért körülfordulási idővel. A szorzat a küldőtől a vevőig oda-vissza terjedő csővezeték bitben mért kapacitása.

Például a 6.54. ábrán a sávszélesség-késleltetés szorzat 40 millió bit. Másképpen szólva a küldőnek 40 millió bites löketet kellene folyamatosan teljes sebességgel adnia, amíg az első nyugta vissza nem érkezik. Ilyen sok bit szükséges a csővezeték teletöltéséhez (mindkét irányban). Ezért jelent egy félmillió bites löket csak 1,25 százalékos hatékonyságot: a csővezeték kapacitásának csak 1,25 százaléka.

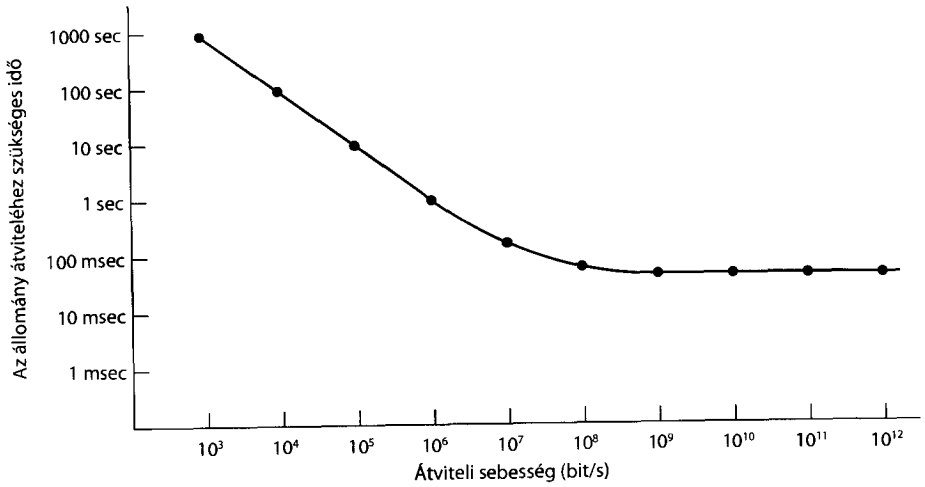
Azt a tanulságot vonhatjuk le, hogy jó teljesítőképesség eléréséhez a vevő ablakának legalább akkorának kell lennie, mint a sávszélesség-késleltetés szorzat, lehetőleg még egy kicsit nagyobbnak, hiszen a vevő esetleg nem tud azonnal válaszolni. Egy kontinenseket összekötő gigabites vonalon legalább 5 megabájt szükséges.

Az ehhez kapcsolódó harmadik probléma az, hogy az egyszerű újraküldési protokollok, mint például az n visszalépéses protokoll gyengén teljesít olyan vonalakon, ahol nagy a sávszélesség-késleltetés szorzat. Vegyünk például egy 1 Gb/s sebességgel működő transzkontinentális vonalat 40 ms körülfordulási idővel. A körülfordulási idő alatt az adó 5 megabájtot tud elküldeni. Ha az átvitelben hiba történik, ez 40 ms alatt jut az adó tudomására. Az n visszalépéses protokoll alkalmazása esetén a küldőnek nemcsak a rosszat, hanem esetleg az egész utána következő 5 megabájtnyi csomagot is újra kell küldenie. Világos, hogy ez óriási erőforrás-pazarlás, így összetettebb protokollokra, mint például a „szelektív újraküldés" protokollra van szükség.

A negyedik probléma a gigabites vonalakkal kapcsolatban, hogy ezek alapvetően mások, mint a megabites vonalak: a hosszú vonalak inkább késleltetés-, semmint sávszélesség-korlátosak. A 6.55. ábrán egy 1 megabites állomány 4000 km-es távolságra történő elküldésének időszükségletei láthatók különböző átviteli sebességek esetén. 1 Mb/s sebességig az átviteli időben az adási sebesség a meghatározó. 1 Gb/s-nél a 40 ms-es körülfordulási idő dominál, és nem a bitek üvegszálba töltéséhez szükséges 1 ms adási késleltetés. A sávszélesség további növelése aligha járhat eredménnyel.

A 6.55. ábrán a hálózati protokollok szerencsétlen hatásait láthatjuk. Ezek szerint az RPC-hez hasonló megáll-és-vár protokollok teljesítőképességének veleszületett felső korlátja van. Ezt a határt a fénysebesség szabja meg. Az optika területén semmilyen mértékű technikai előrelépés nem fogja ezt megnövelni (új fizikai törvények segítenének). Hacsak nem használjuk a gigabites vonalat valami másra, amíg a hoszt a válaszra vár, az semmivel sem jobb, mint egy megabites vonal, csupán jóval drágább.

Az ötödik probléma, hogy a kommunikációs sebességek sokkal gyorsabban növekedtek, mint a számítási sebességek. (Felhívás a számítógépekkel foglalkozó mérnököknek: gyerünk, győzzétek le a kommunikációs hálózatok mérnökeit! Számítunk rátok!) A hetvenes években az ARPANET 56 kb/s sebességgel működött, a számítógépek körülbelül 1 MIPS-esek voltak. Hasonlítsuk össze ezeket a számokat a modern 1000 MIPS-es számítógépek esetével, melyek csomagokat cserélnek egy gigabites vonalon. Az egy bájtra eső műveletek száma több mint tizedére csökkent. A pontos számok az idők folyamán



6.55. ábra. *1 megabites állomány átviteléhez és nyugtázásához szükséges idő 4000 km hosszú vonalon*

és a felhasználási céloktól függően változtak, de a tanulság az, hogy kevesebb idő van protokollfeldolgozásra, mint annak idején volt, ezért a protokolloknak egyszerűbbeknek kell lenniük.

Foglalkozzunk most a problémák helyett kezelési módjaikkal. Az alapvető elv, amit minden gigabites hálózat tervezőjének kívülről kellene tudni:

Tervezni sebesség-, és nem sávszélesség-optimumra kell.

A régi protokollokat gyakran úgy tervezték, hogy minimalizálják a vonalon tartózkodó bitek számát, gyakran oly módon, hogy kicsi mezőket használtak és bájtokba vagy szavakba zsúfolták azokat. Ez a hozzáállás vezeték nélküli hálózatok esetén még mindig érvényes, de nem az gigabites hálózatokra. A protokollok feldolgozási lépéseivel van baj, ezért a protokollokat úgy kell tervezni, hogy minimalizáljuk a feldolgozást. Tisztán látszik, hogy az IPv6 tervezői megértették ezt a követelményt.

Erős lehet a kísértés a gyors működést a hálózati illesztők hardveres megvalósításával elérni. Ez a stratégia azért nehézkes, mert hacsak nem végtelenül egyszerű a protokoll, a hardver csak egy bedugható kártyát jelent egy másik processzorral és a saját programjával. Hogy a hálózati feldolgozóegység (coprocessor) olcsóbb legyen a fő CPU-nál, gyakran lassabb csipet használnak. Ennek a tervezésnek az a következménye, hogy a fő (gyors) processzor idejének nagy részét tétlenül tölti arra várakozva, hogy a második (lassú) processzor elvégezze a kritikus munkát. Tévhit, hogy a fő processzornak várakozás közben van más tennivalója. Ezenkívül két általános célú processzor egymás közti kommunikációjában versenyhelyzetek alakulhatnak ki, ezért kifinomult protokollok szükségesek helyes szinkronizálásukhoz a versenyhelyzetek elkerülésére. Általában az a legjobb megközelítés, hogy egyszerű protokollokat készítünk, és a központi processzorral végeztetjük el a munkát.

A csomagok felépítése fontos tényező a gigabites hálózatokban. A fejrésznek a feldolgozási idő csökkentése érdekében olyan kevés bitet kell tartalmaznia, amennyit csak lehetséges. Ezeknek a mezőknek ugyanakkor elég nagynak kell lenni ahhoz, hogy feladatu-

kat elláthassák, és a feldolgozás megkönnyítéséhez szóhatáron kell kezdődniük. Ebben a környezetben „elég nagy" azt jelenti, hogy olyan problémák ne fordulhassanak elő, mint a sorszámok körbefordulása, miközben a régi csomagok még élnek, vagy a vevők nem tudnak elég nagy ablakméretet bejelenteni, mert túl kicsi az ablakmező és így tovább.

A felhasználói adatokat tartalmazó mező maximális méretének is nagynak kell lennie, hogy csökkentsük a szoftver többletterhelését, és hatékony adatátvitelt valósíthassunk meg. 1500 bájt túl kevés a nagy sebességű hálózatokon, éppen ezért a gigabites Ethernet már támogatja az akár 9 KB-os óriáskereteket, valamint az IPv6 is támogatja a 64 KB-ot meghaladó csomagokat.

Vegyük most szemügyre a visszacsatolás kérdését a nagy sebességű protokollokban. A (viszonylag) hosszú késleltetésű hurok miatt a visszacsatolást el kell kerülni: túl sokáig tart, amíg a vevő jelez a küldőnek. A visszacsatolásra egy példa: az átviteli sebesség szabályozása csúszóablakos protokollal. Hogy elkerüljék a (hosszú) késleltetéseket, melyek abból adódnak, hogy a vevő ablakfrissítéseket küld az adónak, a jövőbeli protokollok talán sebességalapú protokollokra fognak épülni. Ilyen protokollokban a küldő mindent elküldhet, amit akar, feltéve, hogy nem küld gyorsabban egy bizonyos sebességnél, mint amiben a küldő és a vevő előre megállapodott.

Egy másik példa a visszacsatolásra a Jacobson lassú kezdet algoritmusa. Ez az algoritmus több próbálkozást végez, hogy megállapítsa a hálózat terhelhetőségét. Nagy sebességű hálózatoknál a hálózat viselkedését mérő fél tucat ilyen kis próba hatalmas méretű sávszélességet pazarol. Egy hatékonyabb módszer az, ha az összeköttetés felépítésekor a küldő, a vevő és a hálózat mind lefoglalja a szükséges erőforrásokat. Az erőforrások előre történő lefoglalása azzal az előnnyel is jár, hogy így egyszerűbben csökkenthető a dzsitter. Röviden szólva nagyobb sebességek világában a tervezés feltartóztathatatlanul az összeköttetés-alapú működés felé tart, vagy legalábbis valami ahhoz nagyon közelihez.

Egy másik értékes képesség egy normális mennyiségű adat elküldésének lehetősége az összeköttetés-létesítés kérésben. Ily módon egy körülfordulási idő (RTT) megtakarítható.

6.7. Késleltetéstűrő hálózatok

A fejezet végén egy olyan új szállítási megoldást mutatunk be, amely egyszer az internet fontos részévé válhat. A TCP és a legtöbb szállítási protokoll mind azon a feltételezésen alapul, hogy a küldő és a fogadó között egy folyamatosan működőképes útvonal található, egyébként a protokoll hibázni fog, és nem képes adatokat célba juttatni. Bizonyos hálózatokban gyakran nincsen végponttól végpontig tartó útvonal. Jó példa az ilyen hálózatokra a LEO (Low-Earth Orbit – alacsony Föld körüli pálya) műholdakból álló űrbéli hálózat, amint a műholdak elhagyják, vagy belépnek a földi állomások hatókörébe. A földi állomással egy adott műhold csak bizonyos időintervallumokban tud kommunikálni, és két műhold soha nem tud – még földi állomáson keresztül sem – egymással kommunikálni, hiszen valamelyik műhold mindig a földi állomás hatókörén kívül tartózkodik. Más példának említhetnénk a tengeralattjárókból, autóbuszokból, mobiltelefonokból vagy egyéb, számítógéppel felszerelt eszközökből álló hálózatot, ahol vagy a mobilitás, vagy a szélsőséges körülmények miatt az eszközök között csak időszakosan létezik összeköttetés.

Ilyen időszakosan összekapcsolt hálózatokban az adatáramlás megoldható, ha az adatokat a csomópontokban tároljuk, majd működőképes adatkapcsolat esetén továbbítjuk őket. Az ilyen módszer neve **üzenetkapcsolás** (**message switching**). Az adatok végül eljutnak a célpontig. Az ilyen megközelítésen alapuló felépítést DTN-nek (**Delay-Tolerant Network – késleltetéstűrő hálózat** vagy **Disruption-Tolerant Network – szakadástűrő hálózat**) hívjuk.

A DTN-ekhez kapcsolódó munka 2002-ben kezdődött, amikor az IETF egy kutatócsoportot hozott létre e célra. Az ösztönzés a legváratlanabb helyről érkezett: csomagot küldeni az űrbe. Az űrbéli hálózatoknak időszakos kommunikációs lehetőségekkel és nagyon nagy késleltetéssel kell szembenézniük. Kevin Fall észrevette, hogy az ilyen bolygóközi internetekhez használt ötletek a földi hálózatokra is alkalmazhatók, amennyiben azokban az időszakos kapcsolat normálisnak tekinthető [Fall, 2003]. Ez a modell az internet egy hasznos általánosítását mutatja, amelyben tárolás és késleltetés is történhet a kommunikáció során. Az adattovábbítás, az útválasztókban történő csomagkapcsolással ellentétben, inkább a postai levél kézbesítésére vagy az elektronikus levelezésre hasonlít.

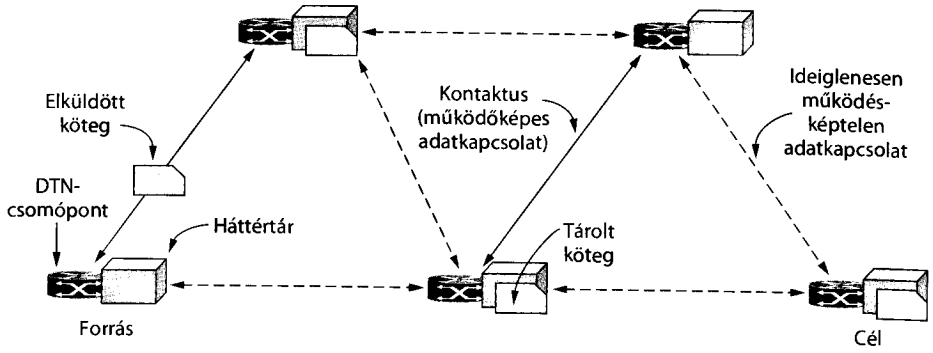
2002 óta a DTN felépítésén finomítottak, valamint a DTN-modell alkalmazásainak köre is nőtt. A legfontosabb alkalmazása a tudományos számítások, a médiaesemények, a webalapú szolgáltatások által létrehozott hatalmas, több terabájt méretű adathalmazoknak a világ különböző pontjain található adatközpontokba történő másolása. Az ilyen, késleltetéstűrő löketeket az operátorok legszívesebben csúcsidőn kívül szeretnék a hálózaton látni, hogy a már kifizetett, de nem használt sávszélességet kihasználják. Tekinthetjük úgy is, hogy a biztonsági másolatokat éjjel szeretnék elkészíteni, amikor más alkalmazás nem terheli le túlságosan a hálózatot. Világméretű szolgáltatások esetén a probléma ott jelentkezik, hogy a világ különböző helyein a csúcsidők máskor vannak. A Bostonban, illetve Perthben található adatközpontokban a csúcsidőn kívüli másolások között csak kevés átlapolódás képzelhető el, hiszen ha az egyik helyen éjszaka van, akkor a másik helyen éppen nappal.

A DTN-modell azonban megengedi a tárolást és a késleltetést az átvitel során. A modell segítségével lehetővé válik, hogy az adathalmazt Bostonból Amszterdamba csúcsidőn kívül juttassuk el, mivel az időzónák között 6 óra a különbség. Az adathalmazt Amszterdamban tárolják egészen addig, amíg nem áll rendelkezésre csúcsidőn kívüli sávszélesség Amszterdam és Perth között. Az adatokat végül Perthbe küldik, hogy az átvitel teljesüljön. Laoutaris és mások [2009] e modell tanulmányozása során megállapították, hogy tekintélyes kapacitást képes biztosítani kevés ráfordítással, és a DTN-modell alkalmazásával gyakran a kétszerese érhető el annak a kapacitásnak, amelyet a hagyományos végponttól végpontig típusú modell nyújt.

A következőkben az IETF DTN felépítését és protokolljait vizsgáljuk meg.

6.7.1. A DTN felépítése

A legfőbb feltételezés az interneten, amelyet a DTN megpróbál lazítani, az, hogy a kommunikációs viszony teljes időtartama alatt létezik egy végponttól végpontig tartó útvonal a küldő és a fogadó között. Amennyiben ilyen nem létezik, a klasszikus internetprotokollok hibáznak. A DTN a végponttól végpontig terjedő összeköttetések hiányát egy,



6.56. ábra. *Késleltetéstűrő hálózat felépítése*

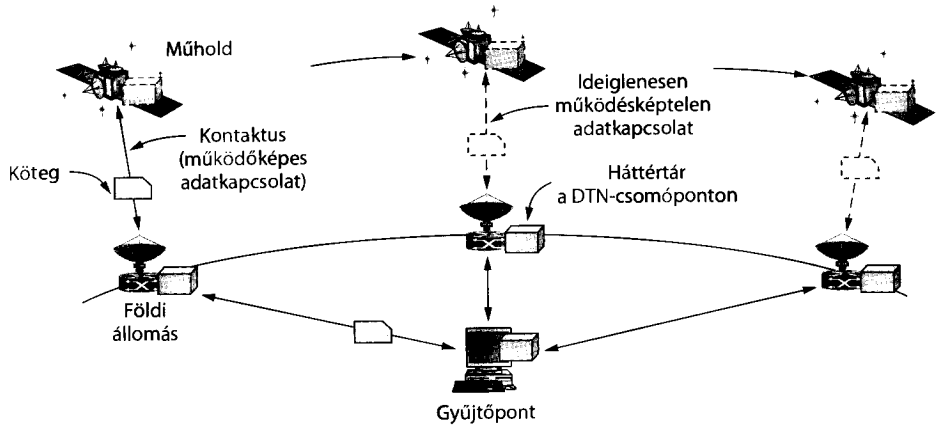
az üzenetkapcsoláson alapuló felépítéssel próbálja megkerülni, ahogy azt a 6.56. ábra mutatja. Cél továbbá az olyan adatkapcsolatok elviselése is, ahol kicsi a megbízhatóság és nagy a késleltetés. A felépítést az RFC 4838 részletezi.

A DTN terminológiájában az üzenetet **kötegnek** (**bundle**) hívjuk. A DTN-csomópontok valamilyen háttértárral vannak felszerelve, tipikusan valamilyen tartós háttértárral, mint például merevlemez vagy flash-memória. A háttértárak tárolják a kötegeket, amíg az adatkapcsolatok elérhetővé nem válnak, majd továbbítják azt. Az adatkapcsolatok időszakosan működnek. A 6.56. ábra öt ideiglenesen működésképtelen adatkapcsolatot mutat, melyek jelenleg nem működnek, valamint két működőképes adatkapcsolatot. Az éppen működő adatkapcsolatokat **kontaktusnak**[7] (**contact**) nevezzük. A 6.56. ábra mutat továbbá két DTN-csomópontot tárolt kötegekkel, amelyek kontaktusokra várnak, hogy a kötegeket továbbíthassák. A kötegek így a kontaktusokon keresztül a forrástól a végcélig továbbítódnak.

A kötegek tárolása és továbbítása a DTN-csomópontokban hasonlóan hangzik, mint a csomagok sorba állítása és továbbítása az útválasztókban, de a kettő között minőségi különbségek vannak. Az útválasztókban az interneten a sorbaállás ezredmásodpercekig, vagy legfeljebb másodpercekig tart. A DTN-csomópontokon a kötegeket órákig tárolják, amíg egy autóbusz meg nem érkezik a városba, vagy amíg egy repülő be nem fejezi repülőútját, vagy amíg egy szenzorcsomópont nem gyűjt a Napból annyi energiát, hogy képes legyen a működésre, vagy amíg egy számítógép alvó állapotából fel nem ébred és így tovább. Ezek a példák rámutatnak továbbá a másik fontos különbségre, mégpedig a csomópontok mozgására. A csomópontok ugyanis mozoghatnak (egy autóbusszal vagy repülővel), amíg adatokat tárolnak, a mozgás pedig a kézbesítés kulcskérdése is lehet. Az internet útválasztói ellenben nem mozoghatnak. A kötegek továbbításának folyamata talán a „tárol-hordoz-továbbít" hármassal jellemezhető leginkább.

Példának okáért tekintsük a 6.57. ábrán bemutatott forgatókönyvet, amely a DTN első, űrbéli alkalmazása volt [Wood és mások, 2008]. A kötegek forrása egy LEO műhold, amely a Földről készít képeket a műholdakról történő katasztrófafigyelő program (Disaster Monitoring Constellation) részeként. A képeket a gyűjtőponthoz kell továbbí-

7 Sajnos a magyar nyelv az adatkapcsolatra nem tartalmaz elég szinonimát, így a „contact" szó magyarban meghonosodott formáját használjuk. *(A fordító megjegyzése)*



6.57. ábra. *A DTN űrbéli alkalmazása*

tani. A műhold azonban a Föld körüli pályáján a három földi állomással csak időszakosan van összeköttetésben. Mindhárom állomással felváltva hoz létre kontaktust. Mind a műholdak, mind a földi állomások, mind a gyűjtőpont egy-egy DTN-csomópontként szerepelnek. Minden kontaktus létrejöttekor egy köteget (vagy annak egy részét) küld a műhold a földi állomásnak, majd a kötegeket a földi hálózaton a gyűjtőpontnak továbbítják, hogy teljessé tegyék az átvitelt.

Ebben a példában a DTN felépítésének legfőbb előnye, hogy természetes módon illeszkedik ahhoz a tényhez, hogy a műholdon a képeket tárolni kell, hiszen a képek készítésének időpontjában nincs összeköttetés. Van két további előny is. Először is elképzelhető, hogy egy kontaktus sem elég hosszú életű a képek elküldéséhez, azonban a három földi állomással létesített kontaktusok között szétoszthatóan az átvitel. Másodsorban, a műhold és a földi állomás közötti adatkapcsolat elkülönül a gerinchálózati adatkapcsolattól, így a műholdról történő letöltést nem korlátozza egy lassú földi hálózat. Az átvitel így teljes sebességgel végrehajtható a köteg földi állomáson való tárolásával, amíg az a megfelelő időpontban a gyűjtőpontnak nem továbbítható.

Egy fontos dolgot nem határoz meg ez a felépítés, mégpedig azt, hogy hogyan lehet a DTN-csomópontok között megtalálni a megfelelő útvonalat. A jó útvonalak az architektúra természetén múlnak, amely leírja, hogy mikor kell küldeni az adatokat és azt is, hogy melyik kontaktuson. Némely kontaktus élettartama meghatározható az időben. Jó példa erre az égitestek mozgása az űrbéli példánkban. Az űrkísérlethez tudták, hogy a kontaktusok mikor jönnek létre, ugyanis minden földi állomás feletti elhaladáskor a kontaktusok élettartama 5 és 14 perc között alakult, és a le-irányú adatkapcsolat kapacitása 8,134 Mbit/s volt. Ennek a tudásnak a birtokában a képeket hordozó kötegek átvitele időben tervezhető.

Más esetekben a kontaktusok megjósolhatók, de kisebb bizonyossággal. Például az autóbuszok viszonylag rendszeresen, a menetrendnek megfelelően létesítenek egymással kontaktusokat, ami kezelhető valamilyen szórással, valamint az ISP-hálózatokban rendelkezésre álló, csúcsidőn kívüli sávszélesség mértéke és ideje is megjósolható a korábbi adatok alapján. A másik véglet, amikor a kontaktusok esetlegesek és véletlenszerűek. Ilyen példa a felhasználók mobiltelefonjai közötti adatátvitel, ahol azon múlnak

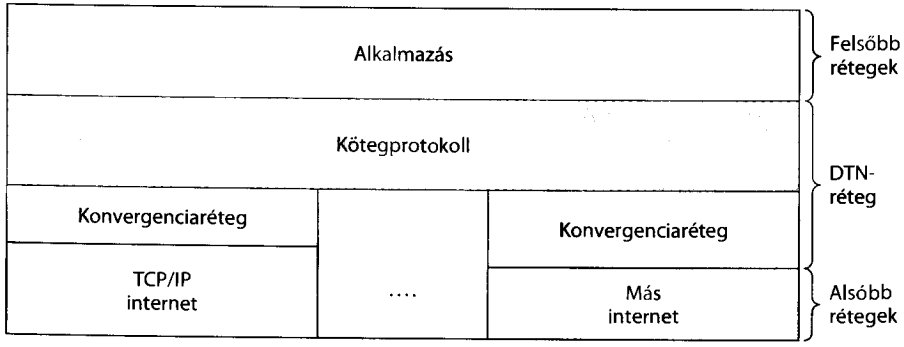
az útvonalak, hogy mely felhasználók építenek ki kontaktusokat egymás között a nap folyamán. Amikor a kontaktusok megjósolhatatlanok, akkor az egyik útválasztási stratégia az, hogy a köteg másolatait különböző útvonalakon küldik szét azt remélve, hogy legalább egy másolat megérkezik a célhoz az élettartam lejárta előtt.

6.7.2. A kötegprotokoll

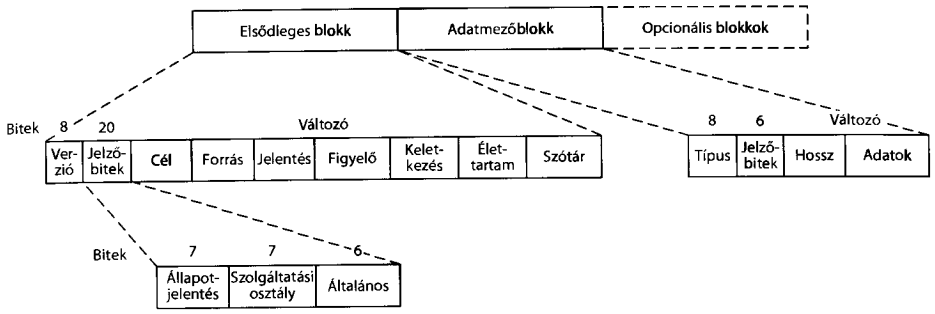
Hogy a DTN-ek működését közelebbről bemutassuk, áttekintjük az IETF-protokollokat. Mivel a DTN-ek még feltörekvőben lévő hálózatok, ezért a kísérleti DTN-ek eltérő protokollokat használnak, mivel nem kötelező IETF-protokollokat használni. Mindazonáltal az IETF-protokollok kezdésnek jók, és a kulcskérdések nagy részére rávilágítanak.

A 6.58. ábrán a DTN-protokollkészletet láthatjuk. A legfontosabb protokoll a **kötegprotokoll (bundle protocol)**, melyet az RFC 5050 tárgyal. Ez a protokoll felelős az alkalmazások felől érkező adatok fogadásáért, valamint az adatok egy vagy több kötegben történő továbbításáért „tárol-hordoz-továbbít" műveletekkel a cél-DTN-csomóponthoz. A 6.58. ábra alapján az is nyilvánvaló, hogy a köteg protokoll a TCP/IP felett fut. Más szavakkal, a DTN-csomópontok között kötegek mozgatására a kontaktusokon TCP/IP használható. A protokoll ilyen módon történő pozicionálása felveti a kérdést, hogy a kötegprotokoll szállítási vagy alkalmazási protokoll. Ahogy az RTP esetében is, itt is tartjuk azt az álláspontunkat, hogy a kötegprotokoll szállítási szolgáltatást nyújt sok különböző alkalmazás számára, annak ellenére, hogy egy szállítási protokoll felett fut. Éppen ezért a DTN-eket ebben a fejezetben tárgyaljuk.

A 6.58. ábrán láthatjuk, hogy a kötegprotokoll más protokollok felett is futhat (mint például az UDP), vagy akár másfajta internetek felett is. Például egy űrbéli hálózatban az adatkapcsolatok késleltetése nagyon nagy lehet. A körülfordulási idő a Föld és a Mars között a bolygók relatív helyzetétől függően könnyedén elérheti a 20 percet is. Képzeljük el most, hogy – különösen rövid üzenetek esetén – milyen jól működne egy ilyen adatkapcsolaton a TCP nyugtázási és újraküldési mechanizmusa. Nem valami jól. Ehelyett használhatnánk valamilyen más, hibajavító kóddal rendelkező protokollt. Vagy egy szenzorhálózatokban, amelyek erős erőforrás-korlátokkal rendelkeznek célszerűbb egy, a TCP-nél egyszerűbb protokollt használni.



6.58. ábra. *Késleltetéstűrő hálózati protokollkészlet*



6.59. ábra. *A kötegprotokoll üzeneteinek felépítése*

Mivel a kötegprotokoll kötött, de mégis több szállítási protokoll feletti futásra szánták, egy funkcionális hézag található a protokollok között. Ennek a hézagnak köszönhetik a létüket a 6.58. ábrán látható konvergenciarétegek. A konvergenciaréteg egyfajta ragasztóréteg, amely az általa összekapcsolt rétegek interfészeit egymáshoz illeszti. Definíció szerint a különböző szállítási protokollokhoz különböző konvergenciaréteg tartozik. Konvergenciarétegeket gyakran találhatunk a szabványokban a régi és új protokollok összekapcsolására.

A kötegprotokoll üzeneteinek a felépítését a 6.59. ábra mutatja. Az üzenetek különböző mezői megmutatják a kötegprotokoll által elvégzett kulcsfeladatokat.

Minden üzenet áll egy elsődleges blokkból, amely gyakorlatilag egy fejrész, egy adatmezőblokkból, amely a felhasználói adatokat tartalmazza, és opcionálisan egy vagy több blokkból, amelyek például biztonsági feladatokat látnak el. Az elsődleges blokk egy *Verzió* mezővel (értéke jelenleg 6) kezdődik, amelyet a *Jelzőbitek* követnek. A jelzőbitek egyebek mellett a szolgáltatási osztályt határozzák meg, amely a köteghez alacsonyabb vagy magasabb prioritást rendel, vagy más feldolgozási kéréseket, mint például, hogy a köteget a célnál nyugtázni kell.

Ezután következnek a címmezők, amelyek a kialakítás három érdekes részét emelik ki. A *Cél* és *Forrás* azonosítója mellett található egy *Figyelő* (*Custodian*) azonosító is. A figyelő felel azért, hogy figyelje a köteg célba érését. Az interneten a figyelő általában a forráscsomópont, hiszen ő küldi újra az adatokat, ha azok nem érték el a végleges céljukat. A DTN esetében azonban egyáltalán nem biztos, hogy a forrás éppen a hálózatra csatlakozik, és így nem is tudja megállapítani, hogy az adatok célba értek-e vagy sem. A DTN a problémát a **figyelt átvitellel (custody transfer)** oldja meg, amelynek során egy másik, a célhoz közelebb eső csomópont vállalja a felelősséget az adatok biztonságos kézbesítéséért. Például, ha egy repülőgépen tárolunk egy köteget egy későbbi helyszínen és időben történő továbbításhoz, akkor a repülőgép tölti be a figyelő szerepét.

A másik érdekesség, hogy ezek az azonosítók *nem* IP-címek. Mivel a kötegprotokollt változatos szállítási protokollok és internetek fölötti működésre szánták, ezért saját azonosítókat definiál. Ezek az azonosítók valójában inkább hasonlítanak magas szintű nevekre, például weboldalak URL-jére, mint alacsony szintű címekre, például IP-címekre. Ezek az azonosítók a DTN-ek alkalmazás szintű útválasztásáról tesznek tanúbizonyságot, amilyen például az elektronikus levél kézbesítése vagy a szoftverfrissítések szétosztása.

A harmadik érdekesség pedig az azonosítók kódolása. Az elsődleges blokk tartalmaz egy *Jelentés* azonosítót is diagnosztikai üzenetek számára. Minden azonosítót úgy kódoltak, hogy az egy változó hosszúságú *Szótár* mezőre hivatkozzon. Ha a figyelő és a jelentést kezelő csomópontok megegyeznek a forrás- vagy a címzett csomóponttal, akkor ez a megoldás egyfajta tömörítést biztosít. Valójában az üzenetformátumok nagy részét kiterjeszthetőre és hatékonyra tervezték, a változó hosszúságú mezők tömörségének köszönhetően. Ez a tömörség nagyon fontos vezeték nélküli adatkapcsolatokon és korlátozott erőforrással rendelkező csomópontokon, mint amilyenek például a szenzorhálózatokban lévő csomópontok.

A következő mező a *Keletkezés*, amely a köteg keletkezésének időpontját tartalmazza a forrás által megadott, a sorba rendezéshez szükséges sorszámmal együtt. Az *Élettartam* mező megadja azt az időpontot, amikortól a köteg által hordozott adat már nem használható. Ezek a mezők azért kaptak helyet, mert az adatokat a DTN-csomópontok hosszú ideig tárolhatják, és az elévült adatokat valahogyan el kell távolítani a hálózatból. Az internettel ellentétben ezek a mezők megkívánják, hogy a DTN-csomópontok lazán szinkronizált órákkal rendelkezzenek.

Az elsődleges blokkot a *Szótár* mező zárja. Ezután következik az *Adatmező* blokk. Ez a blokk egy adatmezőt azonosító rövid *Típus* mezővel kezdődik, majd ezt követi a *Jelzőbitek* kisebb csoportja, amely feldolgozási információt nyújt. Ezeket a *Hossz*, majd az *Adatok* mező zárja. Legvégül állhatnak még opcionális blokkok, mint például egy biztonsági paramétereket hordozó blokk.

A DTN sok vonatkozása még kutatási téma. Jó útválasztási stratégiák a kontaktusok természetétől függnek, ahogy arról már volt szó. A hálózaton belüli adattárolás újabb kérdéseket vet fel. A torlódáskezelésnek most a csomópontok háttértárait mint másfajta, kimeríthető erőforrásnak kell tekintenie. A végponttól végpontig tartó kommunikáció hiánya súlyosbítja a biztonsági problémákat is. Mielőtt egy DTN-csomópont egy köteg figyelését elvállalja, lehet, hogy tudni szeretné, hogy a küldő jogosult-e a hálózatot használni, és hogy a címzett csomópont tényleg igényt tart-e a kötegre. A fenti problémák megoldása függ a DTN fajtájától, mivel az űrbéli hálózatok különböznek a szenzorhálózatoktól.

6.8. Összefoglalás

A szállítási réteg megértése kulcsfontosságú a rétegekbe szervezett protokollok megértéséhez. Számos különböző szolgáltatást kínál, amelyek közül a legfontosabb a végpontok közötti, összeköttetés-alapú, megbízható bájtfolyam a küldőtől a vevőig. Ehhez egy olyan szolgáltatási primitív készlet segítségével lehet hozzáférni, amely az összeköttetések kiépítését, használatát és lebontását teszi lehetővé. Egy gyakran alkalmazott szállítási réteg interfész a Berkley-csatlakozók (sockets) által nyújtott interfész.

A szállítási protokolloknak képesnek kell lenniük az összeköttetések kezelésére egy megbízhatatlan hálózaton. Az összeköttetések kiépítését az olyan késleltetett és megkettőzött csomagok léte nehezíti, amelyek teljesen váratlan pillanatokban újra felbukkanhatnak. Emiatt „háromutas kézfogás"-ra van szükség az összeköttetések felépítéséhez.

Az összeköttetések bontása könnyebb, mint a kiépítésük, de a „két hadsereg" probléma miatt még így is távol áll a triviálistól.

A szállítási rétegnek még akkor is sok dolga van, ha a hálózati réteg teljesen megbízható. Kezelnie kell az összes szolgáltatási primitívet, az összeköttetéseket és az időzítőket, a sávszélességet kell felosztania torlódáskezeléssel, és a forgalomszabályozás érdekében egy változó méretű csúszóablakot kell futtatnia.

A versengő folyamok számára a torlódáskezelésnek a teljes elérhető sávszélességet igazságosan kell felosztania, és a hálózatban történt változásokat folyamatosan nyomon kell követnie. Az AIMD vezérlési törvény egy igazságos és hatékony kiosztáshoz vezet.

Az internet két fő szállítási protokollt használ, az UDP-t és a TCP-t. Az UDP egy összeköttetés nélküli protokoll, amely lényegében egy olyan héj az IP-csomagok körül, amely lehetővé teszi több folyamat nyalábolását és a nyalábok szétbontását egyetlen IP-címen. Az UDP kliens–szerver-párbeszédek lefolytatására alkalmas, például RPC használatával. Felhasználható még az RTP-hez hasonló valós idejű protokollok építőelemeként.

Az internet fő szállítási protokollja a TCP, amely megbízható, kétirányú, torlódáskezeléssel ellátott bájtfolyamot biztosít. Minden szegmens egy 20 bájtos fejrészt tartalmaz. Nagy mennyiségű munka fekszik a TCP teljesítmény-optimalizálásában, amely Nagle, Clark, Jacobson, Karn és mások algoritmusait használja fel.

A hálózat hatékonysága általában főként a protokollfeldolgozás és a szegmensfeldolgozás többletterhelésén múlik, és a sebesség növelésével a helyzet egyre romlik. A protokollokat nagy sávszélesség-késleltetés szorzattal rendelkező útvonalakon arra kell tervezni, hogy a lehető legkisebbre csökkentsék a felhasznált szegmensek számát, és hogy olyan útvonalat dolgozzanak ki, amelyen nagy a sávszélesség-késleltetés szorzat. A gigabites hálózatok egyszerű protokollokat és folyamszerű feldolgozást igényelnek.

A késleltetéstűrő hálózatok kézbesítési szolgáltatást nyújtanak olyan hálózatokon, amelyben az adatkapcsolatokon az összeköttetések esetlegesek, vagy a késleltetések nagyok. A közbeeső csomópontok az információt hordozó kötegeket tárolják és hordozzák, így azok végül célba érnek még akkor is, ha bármikor nincs működő útvonal a küldő és vevő között.

6.9. Feladatok

1. A 6.2. ábrán látható szállítási primitív példánkban a LISTEN blokkoló hívás. Feltétlenül szükséges ez? Ha nem, magyarázzuk el, hogyan lehetne egy nem blokkoló primitívet használni! Milyen előnnyel rendelkezne ez a szövegben leírtakkal szemben?

2. A szállítási szolgáltatás primitívjei az összeköttetés felépítése során a két végpont között aszimmetriát tételeznek föl. Az egyik végpont (a szerver) LISTEN hívást futtat, míg a másik végpont (a kliens) CONNECT hívást. A P2P-alkalmazásokban, amilyenek például a fájlmegosztó rendszerek (BitTorrent), azonban minden végpont egyenrangú. Nem léteznek sem szerverek, sem kliensek. A szállítási szolgáltatást hogyan használhatnánk fel ilyen P2P-alkalmazások készítésére?

3. A 6.4. ábra modelljében azt feltételeztük, hogy a hálózati réteg csomagokat veszíthet, ezért azokat egyenként nyugtázni kell. Tegyük fel, hogy a hálózati réteg 100 százalékosan megbízható, és sohasem veszít csomagokat. Milyen változtatásokat kell – ha egyáltalán szükséges – a 6.4. ábrán végezni?

4. A 6.6. ábra mindkét részén szerepel egy megjegyzés arról, hogy a *SERVER_PORT* értékének a kliensnél és a szervernél meg kell egyeznie. Miért olyan fontos ez?

5. A 6.6. ábra internetes fájlszerver példájában elképzelhető, hogy a connect( ) rendszerhívás a kliensben sikertelenül fut le, azonkívül, hogy a szerver LISTEN hívásának várakozási sora megtelik? Tételezzük fel, hogy a hálózat hibátlan.

6. Ha el kell döntenünk azt a kérdést, hogy egy olyan szerverünk legyen-e, amelyik éjjel-nappal fut, vagy pedig egy olyan szerverünk legyen, amelyiket igény szerint egy folyamatszerver indít el, akkor az egyik figyelembe vehető szempont az, hogy az általa nyújtott szolgáltatást milyen gyakran használjuk. Elképzelhető-e, hogy van más szempont is, amelyet e döntést meghozatalánál figyelembe veszünk?

7. Tegyük fel, hogy a kezdeti sorszámok előállítására az óravezérelt módszert használjuk 15 bites számlálóval mint órával. Az óra 100 ezredmásodpercenként üt egyet, a maximális csomagélettartam 60 s. Milyen gyakran történik újraszinkronizáció
(a) a legrosszabb esetben?
(b) amikor az adatok percenként 240 sorszámot használnak el?

8. Miért kell a *T* maximális csomagélettartamnak olyan nagynak lenni, hogy biztosítsa nemcsak a csomagok, hanem a nyugtáik kihalását is?

9. Képzeljük el, hogy összeköttetések létesítéséhez nem „háromutas", hanem „kétutas kézfogás" protokollt használunk. Másképpen fogalmazva a harmadik üzenet nem szükséges. Kialakulhatnak ekkor holtpontok? Mondjon egy példát, vagy bizonyítsa be, hogy nem alakulhatnak ki!

10. Képzeljünk el egy általánosított „*n* hadsereg" problémát, amelyben bármely két hadsereg megegyezése elegendő a győzelemhez. Van olyan protokoll, ami győzelemre viszi a kékeket?

11. Tekintsük a hoszt-összeomlásokból történő talpra állás problémáját (lásd 6.18. ábra). Ha az írás és nyugta küldése (vagy a fordított sorozat) közötti időintervallum viszonylag kicsivé tehető, mi a küldő, illetve a vevő legjobb stratégiája a protokoll hibázási esélyének minimalizálására?

12. Tételezzük fel, hogy egy új *E* folyamot adunk a 6.20. ábrán látható hálózathoz, amely az R1-R2-R6 útvonalon halad. Hogyan változik a maximum-minimum sávszélesség-kiosztás az öt folyamra?

13. Soroljuk fel a hitelalapú protokoll előnyeit és hátrányait a csúszóablakos protokollal szemben!

14. A torlódáskezelés igazságosságára létezik néhány másik vezérlési törvény: AIAD (Additive Increase Additiv Decrease – additív növelés, additív csökkentés), MIAD (Multiplikative Increase Additiv Decrease – multiplikatív növelés, additív csökkentés), MIMD (Multiplikative Increase Multiplikative Decrease – multiplikatív növelés, multiplikatív csökkentés). Értékeljük a fenti három módszert a konvergencia és a stabilitás szempontjából!

15. Mire van az UDP? Nem volna elegendő a felhasználói folyamatokra hagyni a nyers IP-csomagok küldését?

16. Tekintsünk egy egyszerű, UDP-re épülő alkalmazási szintű protokollt, amely azt teszi lehetővé a kliensnek, hogy egy jól ismert címen elérhető távoli szerverről állományokat kérjen le. A kliens először egy olyan kérést küld, amelyben az állomány neve szerepel. A szerver adatcsomagok olyan sorozatával válaszol, amely a kért állomány darabjait tartalmazza. A kliens és a szerver a megáll-és-vár protokoll használatával biztosítja a megbízhatóságot és a sorrendhelyes kézbesítést. Lát hiányosságot ebben a protokollban a nyilvánvalóan rossz teljesítőképességen kívül? Gondolja jól végig, hogy mi történik, ha valamelyik folyamat lefagy!

17. Egy kliens 128 bájtos kérést küld egy tőle 100 km távolságra levő szervernek egy 1 gigabites üvegszálon. Mi a vonal hatékonysága a távoli eljáráshívás alatt?

18. Tekintsük ismét az előző feladatban vázolt helyzetet! Számítsuk ki az elképzelhető legkisebb válaszidőt az adott 1 Gb/s sebességű vonalra és egy 1Mb/s sebességűre is! Milyen következtetést vonhatunk le?

19. Az üzenetek kézbesítésénél mind az UDP, mind a TCP portszámokat használ a célentitás azonosítására. Adjon két okot arra, hogy miért találtak ki ezekhez a protokollokhoz egy új absztrakt azonosítót (a portszámokat) ahelyett, hogy a folyamatazonosítókat használták volna, amelyek már akkor is léteztek, amikor ezeket a protokollokat tervezték.

20. Számos RPC-megvalósítás biztosít a kliens részére egy opciót, amelyben kiválaszthatja, hogy az RPC-t UDP avagy TCP felett szeretné használni. Milyen feltételek mellett fogja a kliens az RCP-t TCP felett futtatni, és milyen feltételek mellett fogja UDP felett futtatni?

21. Tételezzünk fel két hálózatot, *N*1-et és *N*2-t, mindkettő ugyanolyan átlagos késleltetést biztosít egy *A* forrás és egy *D* cél között. *N*1 esetén a különböző csomagok késleltetése egyenetlen eloszlású, ahol a maximális késleltetés értéke 10 másodperc, míg az *N*2 esetén a csomagok 99 százaléka 1 másodpercen belül érkezik meg, azon-

ban a legnagyobb késleltetésre nincs felső határ. Mutassa be, hogy az RTP hogyan használható ebben a két esetben élő audio/videofolyam továbbítására!

22. Mi a TCP-ben lehetséges legkisebb MTU-méret, ha beleszámítjuk a TCP és az IP többletterhét, de nem számítjuk bele az adatkapcsolati réteg keretezését?

23. A datagramok feldarabolását és összeállítását az IP végzi a TCP számára láthatatlan módon. Ez azt is jelenti, hogy a TCP-nek nem kell aggódnia az adatok rossz sorrendben érkezése miatt?

24. Az RTP-t általában CD-minőségű hang továbbítására használják, amely két 16 bites mintavételt jelent 44 100-szor másodpercenként (egy-egy minta mindkét hangcsatornáról). Hány csomagot kell másodpercenként elküldenie ebben az esetben az RTP-nek?

25. Lehetséges lenne az RTP kódját az UDP kódjához hasonlóan beleépíteni az operációs rendszer magjába? Válaszát indokolja!

26. Az 1. hoszt egyik folyamata a *p* portra kapcsolódik, a 2. hoszt egyik folyamata pedig a *q* portra. Lehetséges, hogy a *p* és *q* portok között egyszerre kettő vagy több TCP-összeköttetés legyen nyitva?

27. A 6.36. ábrán azt láthattuk, hogy a 32 bites *Nyugta* mezőn kívül egy *ACK* bit is található a negyedik szóban. Tényleges többletet jelent ez? Miért, illetve miért nem?

28. Egy TCP-szegmens maximális adatmezeje 65 495 bájt. Miért választottak ilyen furcsa számot?

29. Mutassunk két példát arra, hogy hogyan kerülhet a 6.39. ábra véges automatája a *SYN RCVD* állapotba!

30. Tekintsük a lassú kezdet protokoll hatását egy 10 ms körülfordulási idővel rendelkező torlódásmentes vonalon. A vevő ablaka 24 kilobájt, a maximális szegmensméret 2 kilobájt. Mennyi idő telik el, amíg az első tele ablak elküldhető?

31. Tegyük fel, hogy a TCP torlódási ablak 18 kilobájtra van állítva, és időtúllépés következik be. Mekkora lesz az ablak, ha a következő négy löket mind sikeresen átjut? A maximális szegmensméretet vegyük 1 kilobájtnak.

32. Ha a TCP *RTT* körülfordulási ideje jelenleg 30 ms, és a következő nyugták rendre 26, 32 és 24 ezredmásodperc elteltével érkeznek, mi az új, Jacobson algoritmusával becsült *RTT* érték? α-t vegyük 0,9-nek!

33. Egy TCP-t futtató gép 65 535 bájtos ablakokat küld egy 1 Gbit/s sebességű csatornán, melynek egyik irányban mért késleltetése 10 ms. Legfeljebb mekkora átbocsátóképesség érhető el? Mekkora a vonal hatékonysága?

34. Mi a legnagyobb vonali sebesség, amellyel egy hoszt anélkül adhat 1500 bájtos TCP-adatmezőket, hogy a sorszámokat kimerítené? Vegyük figyelembe a TCP, az IP és az Ethernet többletbitjeit is! Feltehetjük, hogy az Ethernet-kereteket a hoszt folyamatosan tudja adni.

35. Az IETF-nek komoly erőfeszítéseket kellett tennie, hogy az IPv4 korlátaira felhívja a figyelmet, mely végül az IPv6 létrehozásához vezetett. Még mindig jelentős idegenkedés tapasztalható azonban az új verzió elfogadásával kapcsolatban. A TCP korlátainak felhívására nem kellett ilyen erőfeszítéseket tenni. Magyarázzuk meg, miért lehet ez!

36. Mekkora a maximális adatsebesség összeköttetésenként abban a hálózatban, melyben a maximális szegmensméret 128 bájt, a maximális szegmensélettartam 30 másodperc és 8 bites sorszámokat használnak?

37. Tegyük fel, hogy egy szegmens vételéhez szükséges időt mérjük. Amikor megszakítás történik, leolvassuk a rendszerórát ezredmásodpercben. Amikor a szegmenst teljesen feldolgoztuk, ismét leolvassuk az óra állását. 270 000 alkalommal kapunk 0 ms eredményt és 730 000 alkalommal 1 ms-osat. Mennyi ideig tart egy szegmens vétele?

38. Egy processzor 1000 MIPS sebességgel hajtja végre az utasításokat. Egyszerre 64 bit adatot tud átmásolni, és minden szó másolása tíz utasításba kerül. Ha egy beérkező csomagot négyszer kell átmásolni, képes ez a rendszer 1 Gbit/s sebességű vonal kezelésére? Az egyszerűség kedvéért tegyük fel, hogy minden utasítás, beleértve a memóriaírásokat és -olvasásokat, a teljes 1000 MIPS-es sebességgel hajtható végre.

39. Hogy elkerüljük azt a problémát, hogy a sorszámok körbefordulásakor még élnek régi csomagok, 64 bites sorszámokat használhatunk. Elméletileg azonban egy üvegszál 75 Tbit/s sebességre képes. Mekkora legyen a maximális csomagélettartam ahhoz, hogy a jövő 75 Tbit/s sebességű hálózataiban se fordulhasson elő körbefordulási probléma még a 64 bites sorszámok használata esetén sem? Tegyük fel, hogy a TCP-hez hasonlóan minden bájtnak saját sorszáma van.

40. A 6.6.5. szakaszban azt számoltuk ki, hogy egy gigabites vonal 80 000 csomagot ad át másodpercenként a címzett hosztnak, amelynek mindössze 6250 utasítás ideje alatt fel kell dolgoznia azokat, így csak a processzoridő fele marad meg az alkalmazásoknak. Ebben a számításban 1500 bájtos csomagokat tételeztünk fel. Végezzük el újra ezt a számolást az ARPANET-en alkalmazott csomagmérettel (128 bájt)! A csomagméretek mindkét esetben tartalmazzák a többletbájtokat is!

41. Egy 4000 km hosszú 1 Gbit/s sebességű hálózaton nem a sávszélesség, hanem a késleltetés a korlátozó tényező. Tekintsünk egy MAN-t átlagosan 20 km-es forrás–cél távolsággal. Milyen adatsebességnél lesz a fény sebességéből eredő körülfordulási késleltetés az 1 kilobájtos csomag elküldési idejével egyenlő?

42. Számoljuk ki a sávszélesség és a késleltetés szorzatát a következő hálózatokra: (1) T1 (1,5 Mb/s), (2) Ethernet (10 Mb/s), (3) T3 (45 Mb/s), (4) STS-3 (155 Mb/s). Feltehetjük, hogy az oda-vissza út ideje 100 ms. Ne feledjük, hogy a TCP fejrésze fenntart 16 bitet az ablakméret megjelölésére! Számításaink alapján ennek milyen következményei vannak?

43. Mennyi a sávszélesség és a késleltetés szorzata egy geoszinkron műhold 50 Mb/s-os csatornáján? Mekkora legyen az ablak a csomagokban, ha azok mérete (a többletbájtokkal együtt) egységesen 1500 bájt?

44. A 6.6. ábra állományszervere távol áll a tökéletestől, lehetne még javítani rajta néhány helyen. Hajtsuk végre rajta az alábbi módosításokat:
(a) Adjunk egy harmadik paramétert a klienshez, amelyben egy bájttartományt lehet megjelölni.
(b) Adjunk a klienshez egy **-w** kapcsolót, amely lehetővé teszi, hogy az állományt a szerverre írjuk.

45. A hálózati protokollok számára általában szükség van az üzenetek manipulálására. Emlékezzünk vissza, hogy a protokollok az üzenetekhez fejrészeket adnak, illetve távolítanak el róluk. Némely protokoll egyetlen üzenetet több részbe tördel, majd a későbbiekben ezeket a részeket összeállítja egyetlen üzenetté. E célból tervezzünk és valósítsunk meg egy olyan üzenetkezelő könyvtárat, amely lehetőséget ad új üzenet létrehozására, fejrész hozzáadására, valamint eltávolítására, üzenetek két darabba törésére, majd a két rész egyesítésére, és egy üzenet másolatának az elmentésére. A megvalósításnak az adatok egyik pufferből másik pufferbe másolását minimalizálni kell! Nagyon fontos, hogy az üzeneteket kezelő műveletek az üzenetben helyet foglaló felhasználói adatokat ne érintsék, hanem a kezeléshez inkább mutatókat használjunk!

46. Tervezzünk és valósítsunk meg egy olyan csevegőrendszert, amely több felhasználócsoport egyidejű csevegését teszi lehetővé! Legyen egy olyan csevegő-koordinátor, amely egy jól ismert hálózati címen tartózkodik, UDP-t használ a kliensekkel való kommunikációhoz, létrehozza a csevegőszervereket az egyes csevegési viszonyokhoz és karbantartja a csevegési viszonyok jegyzékét. Minden csevegési viszonyhoz egyetlen csevegőszerver tartozik. A csevegőszerverek TCP-t használnak a kliensekkel való kommunikációhoz. A csevegő-kliensprogram segítségével a felhasználók új csevegési viszonyt indíthatnak, csatlakozhatnak egy már folyamatban levőhöz, illetve elhagyhatják a viszonyt. Tervezzük és valósítsuk meg a koordinátor, a szerver és a kliens kódját!

7. Az alkalmazási réteg

A bevezető jellegű részek után elérkeztünk az alkalmazások fejezetéhez. Az alkalmazási réteg alatt található egyéb rétegek a megbízható szállítási szolgáltatást biztosítják, de a felhasználó számára nem végeznek tényleges munkát. Ebben a fejezetben igazi hálózati alkalmazásokkal fogunk megismerkedni.

Még az alkalmazási rétegben is szükség van azonban olyan kiegészítő protokollokra, melyek az alkalmazások működését biztosítják. Így az alkalmazások tárgyalása előtt ezek közül egyet részletesebben is megnézünk: ez a protokoll a DNS, amely a nevek kezeléséért felelős az interneten. Ezután három tényleges alkalmazást is megtárgyalunk: az elektronikus levelezést, a világhálót (world wide web) és végül a multimédiát. A fejezetet a tartalomelosztás bővebb ismertetésével zárjuk, benne az egyenrangú hálózatokkal.

7.1. DNS – a körzetnévkezelő rendszer

A programok elvileg képesek hivatkozni weboldalakra, levelesládákra és más erőforrásokra azoknak a számítógépeknek a hálózati (például IP) címeinek a felhasználásával, amelyeken ezek megtalálhatók, de ezeket a címeket az emberek nemigen tudják megjegyezni. Továbbá, ha egy vállalati weboldal a *128.111.24.41* címen böngészhető, majd a vállalat áthelyezi a webszervert egy eltérő IP-című másik számítógépre, akkor mindenkinek meg kell mondani az új IP-címet is. Ezért magas szintű, olvasható neveket vezettek be, hogy különválasszák a gépek neveit a gépek címeitől. Így a vállalat webszervere *www.cs.washington.edu* néven válhat ismertté, függetlenül annak IP-címétől. A hálózat persze továbbra is csak a numerikus címeket érti meg, tehát valamilyen mechanizmusra van szükség, amely átalakítja a neveket hálózati címekké. A következő szakaszokban megnézzük, hogy hogyan megy végbe ez a leképezés az interneten.

Még az ARPANET idejében, egyszerűen volt egy fájl, a *hosts.txt*, amelyben fel voltak sorolva a számítógépek nevei és azok IP-címei. Minden éjszaka az összes hoszt kiolvasta ezt a fájlt arról a gépről, ahol azt karbantartották. Ez a megközelítés egészen jól működött néhány száz nagy időosztásos gép esetében.

Még jóval azelőtt, hogy sok millió PC-t kapcsoltak volna az internetre, mindenki rájött, hogy ez a módszer nem működhet örökké. Ha másért nem, hát azért, mert a

fájl mérete túl nagy lenne. Még fontosabb azonban az, hogy a hosztnevek állandóan ütköznének, amennyiben nem központilag kezelnénk a neveket; ez pedig a terhelés és a késleltetések miatt elképzelhetetlen lenne egy kiterjedt nemzetközi hálózatban. Ezeknek a problémáknak a megoldására találták ki a **DNS**-t (**Domain Name System – körzetnévkezelő rendszer**) 1983-ban.

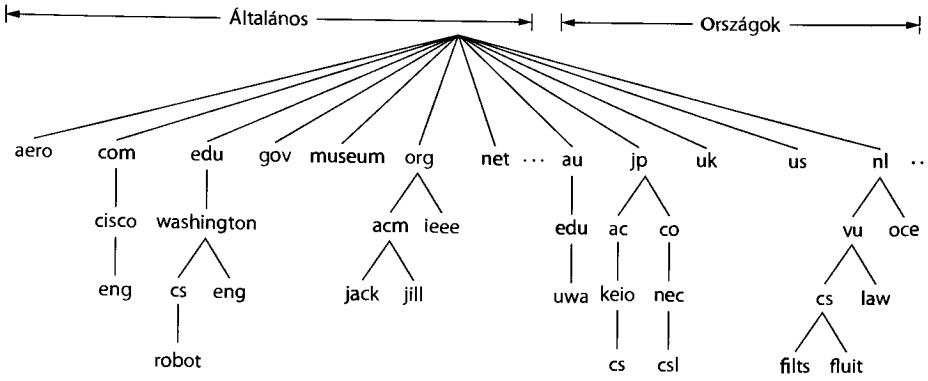
A DNS lényege egy hierarchikus körzetalapú névkiosztási séma, és az azt megvalósító osztott adatbázisrendszer kitalálásában rejlik. Elsősorban arra szolgál, hogy hosztneveket feleltessen meg IP-címeknek, de más célokra is használható. A DNS-t az RFC 1034, 1035 és 2181 definiálja, míg a részleteket számos más RFC tartalmazza.

Vázlatosan a következőképpen zajlik a DNS használata. Egy felhasználói program a név IP-címre való leképzéséhez meghívja a névvel mint paraméterrel a **címfeloldó** (**resolver**) nevű könyvtári eljárást. A címfeloldóra korábban már láttunk egy példát (lásd 6.6. ábra, *gethostbyname*). A címfeloldó lekérdezi a nevet a helyi DNS-szervertől. A szerver megkeresi és visszaküldi az IP-címet a címfeloldónak, ami visszaadja azt a hívónak. A kérés- és válaszüzenetek UDP-csomagokkal valósulnak meg. Az IP-cím birtokában a program már felépítheti a TCP-összeköttetést a célgéppel vagy küldhet neki UDP-csomagokat.

7.1.1. A DNS-névtér

Nagy mennyiségű, állandóan változó nevek halmazának kezelése nem triviális probléma. A postai rendszerben a névkezelés úgy történik, hogy a címzett eléréséhez a levélen (implicit vagy explicit módon) fel kell tüntetni az országot, az államot vagy megyét, a várost és az utcát. Ezen hierarchikus címzés mellett nincs kavarodás a Main St.-en, White Plainsben, New York államban lakó Marvin Anderson, és a Main St.-en, Austinban, Texasban lakó Marvin Anderson között. A DNS is így működik.

Az internet esetében a névhierarchia legfelső szintjét az **ICANN** (**Internet Corporation for Assigned Names and Numbers – Internettársaság Kiosztott Nevek és Számok Kezelésére**) nevű szervezet kezeli. Az ICANN-t erre a célra, és abból a megfontolásból hozták létre 1998-ban, hogy az internet világméretűvé és gazdasági tényezővé válhasson.



7.1. ábra. *Az internet DNS-névtér egy darabja*

Körzet	Tervezett alkalmazás	Bevezetés éve	Korlátozott?
com	Kereskedelmi	1985	Nem
edu	Oktatási intézmények	1985	Igen
gov	Kormányzati	1985	Igen
int	Nemzetközi szervezetek	1988	Igen
mil	Katonai	1985	Igen
net	Hálózati szolgáltatók	1985	Nem
org	Nonprofit szervezetek	1985	Nem
aero	Légi közlekedés	2001	Igen
biz	Cégek	2001	Nem
coop	Szövetkezetek	2001	Igen
info	Információforrások	2002	Nem
museum	Múzeumok	2002	Igen
name	Emberek különféle nevei	2002	Nem
pro	Szakmák és szakemberek	2002	Igen
cat	Katalán nyelv és kultúra	2005	Igen
jobs	Munkalehetőségek, állás	2005	Igen
mobi	Mobil készülékek	2005	Igen
tel	Kommunikációs szolgáltatások	2005	Igen
travel	Utazási irodák	2005	Igen
xxx	Szexipar	2010	Nem

7.2. ábra. *Általános elsődleges körzetek*

Az internet a koncepció szerint több mint 250 **elsődleges körzetre** (**top-level domain**) van osztva, ahol minden körzethez sok hoszt tartozik. Minden körzet alkörzetekre oszlik, amelyek tovább vannak osztva és így tovább. Ezek a körzetek legszemléletesebben egy fával ábrázolhatók (lásd 7.1. ábra). A fa levelei olyan körzeteket reprezentálnak, amelyek nincsenek alkörzetekre bontva (de természetesen tartoznak hozzájuk gépek). Egy levélben levő körzet tartalmazhat egyetlen hosztot is, vagy képviselhet egy egész vállalatot, és így tartalmazhat több ezer hosztot.

A legfelső szinten levő, ún. elsődleges körzetek kétfélék lehetnek: általánosak és országra vonatkozó körzetek. A 7.2. ábrán látható általános körzetek között megtalálhatók az 1980-as évek eredeti körzetei és az ICANN-nél kérvényezett körzetek is. További általános, elsődleges körzetek megjelenése várható a jövőben.

Az országra vonatkozó körzetek esetében minden országhoz tartozik egy országkörzet, az ISO 3166-nak megfelelően. 2010-ben bevezették a nemzetközi, nem csak latin betűket használó országra vonatkozó körzetneveket is. Ezek a körzetek lehetővé teszik a hosztok elnevezését arab, cirill, kínai és más nyelveken.

A másodlagos körzetnevekhez, mint amilyen például a *ceg-neve.com*, könnyű hozzájutni. Az elsődleges körzeteket az ICANN által kijelölt **adminisztrátorok (registrar)** kezelik. Egy név megszerzéséhez mindössze annyi szükséges, hogy az ember elmegy a megfelelő elsődleges körzet (ez esetben a *com*) adminisztrátorához, és meggyőződik róla, hogy a kívánt név még szabad és nem valaki más védjegye. Ha nincs semmi probléma, akkor az igénylő egy kisebb éves összeg fejében megkapja a nevet.

Ahogyan azonban az internet egyre inkább üzleti alapokra helyeződik és egyre nemzetközibbé válik, úgy lesz egyre több a vitás eset is, különösen a névadásokkal kapcsolatban. Ezekben a vitákban maga az ICANN is érintett. Például az *xxx* körzetnév létrehozása évekbe tellett, és bírósági ügyeket kellett rendezni. A felnőtt tartalmaknak önkéntesen saját körzetükbe helyezése vajon jó vagy rossz dolog? (Néhányan egyáltalán nem akarják, hogy felnőtt tartalmak elérhetők legyenek az interneten, míg mások ezeket egyetlen körzetbe kívánják helyezni, hogy a szűrők könnyen megtalálhassák és elzárhassák ezeket a gyerekek elől.) Néhány körzet önszerveződő, míg másoknál különféle megszorítások érvényesek arra vonatkozóan, hogy ki használhat egy nevet (lásd 7.2. ábra). De mely megszorítások a megfelelők? Vegyük például a *pro* körzetet. Ezt a minősített szakembereknek szánták. De ki számít szakembernek? Az orvosok és az ügyvédek nyilván szakembernek számítanak. De mi van a szabadúszó fényképészekkel, a zongoratanárokkal, varázslókkal, vízvezeték-szerelőkkel, fodrászokkal, féregirtókkal, tetoválóművészekkel, zsoldosokkal és prostituáltakkal? Vajon ezek a foglalkozások is szakmának számítanak? És ha igen, ki tanúsítja, hogy valóban azok?

A nevekben rengeteg pénz is van. Tuvalu állam 50 millió dollárért adta bérbe *tv* körzetének haszonbérletét csak azért, mert államának kódja jól megfelelt a televíziós oldalak reklámozására. Mára gyakorlatilag minden gyakori (angol) szó elkelt a *com* körzetben, a legáltalánosabb elírásaikkal együtt. Próbálja ki a háztartási cikkek, állatok, növények, testrészek stb. neveit! Még annak a bevett gyakorlatnak is külön neve van, amikor egy körzetet csak azért jegyeznek be, hogy később egy érdekelt félnek lényegesen magasabb áron lehessen továbbadni: ez az ún. **kiberkivárás (cybersquatting)**. Sok vállalat, amely lassan reagált az internet korszakának kezdetén, kézenfekvő körzetnevük bejegyeztetésekor szembesült csak azzal, hogy ezeket már korábban lefoglalták. Általában, amíg védjegyeket nem sértenek meg és nincs szó csalásról, az kapja meg a neveket, aki hamarabb igényelte. Mindazonáltal a nevekkel kapcsolatos viták megoldására szolgáló eljárásmódokat még mindig pontosítják.

Minden körzet nevét az adott névtől a (névtelen) gyökérhez felfelé vezető út adja. A komponenseket pont választja el egymástól (ezt „dot"-nak ejtik). Így a Cisco műszaki részlegének neve lehet *eng.cisco.com.*, szemben egy UNIX stílusú névvel, mint például a */com/cisco/eng*. Vegyük észre, hogy a hierarchikus felépítésből adódóan nincs ütközés az *eng.cisco.com.*-ban és az *eng.washington.edu.*-ban használt *eng* között, ami a University of Washington Angol nyelvi tanszékének neve lehetne.

A körzetnevek lehetnek mind abszolútak, mind relatívak. Az abszolút körzetnevek ponttal végződnek (például *eng.cisco.com.*), míg a relatívak nem. A relatív neveket egy adott környezetben kell értelmezni, hogy a valódi jelentésüket megállapíthassuk. Mindkét esetben egy körzetnév a fa egyik csomópontjára és az alatta levő részfára vonatkozik.

A körzetnevekben mindegy, hogy kis- vagy nagybetűket használunk-e, tehát az *edu* és az *EDU* ugyanazt jelenti. A névkomponensek maximum 63 karakter hosszúak lehetnek, és az egész útvonalnév nem haladhatja meg a 255 karaktert.

Elvileg a körzeteket a fának az általános és az országokra vonatkozó körzetei közé is beilleszthetjük. Például a *cs.washington.edu* egyenértékű megfelelője lehet a *us* országkörzet alá illeszkedő *cs.washington.wa.us* névnek. Gyakorlatilag azonban majdnem minden egyesült államokbeli szervezet az általános körzetekhez tartozik, és majdnem minden Egyesült Államokon kívüli szervezet a saját országkörzetéhez tartozik. Nem szól szabály az ellen, hogy egy szervezet két elsődleges körzetbe is be legyen jegyezve, de a multinacionális cégeken kívül (például *sony.com*, *sony.net* és *sony.nl*) kevés szervezet él ezzel a lehetőséggel.

Minden körzet maga ellenőrzi az alatta levő körzetek kiosztását. Például Japánnak külön körzetei vannak, *ac.jp*, *co.jp*, a *com* és *edu* megfeleltetésére. Hollandiában nincs ilyen szétosztás, hanem minden szervezet egyenesen az *nl*-hez tartozik. Ily módon mindhárom alábbi cím egyetemek informatika tanszékeinek címei:

1. *cs.washington.edu* (University of Washington, az Egyesült Államokban);
2. *cs.vu.nl* (Vrije Universiteit, Hollandiában);
3. *cs.keio.ac.jp* (Keio Egyetem, Japánban).

Egy új körzet létrehozásához engedély kell attól a körzettől, amelyhez tartozni fog. Például ha létrejön egy VLSI-csoport a University of Washington egyetemen belül, és *vlsi.cs.washington.edu* néven akar futni, akkor attól kell engedélyt kérnie, aki a *cs.washington.edu*-t kezeli. Hasonlóképpen, ha egy új egyetem nyílna, mondjuk a University of Northern South Dakota, akkor az *edu* körzet karbantartójától kellene engedélyt kérni, hogy rendelje hozzá a *unsd.edu* címet (amennyiben az még felhasználható). Ily módon nincsenek névkonfliktusok, és minden körzet számon tarthatja a hozzá tartozó alkörzeteket. Miután egy új körzet létrejött, már szabadon létrehozhat hozzá tartozó alkörzeteket (például *cs.unsd.edu*) anélkül, hogy a fán egy feljebb elhelyezkedőtől engedélyt kellene kérnie.

Az elnevezések nem a hálózat fizikai elrendezését, hanem a szervezetek határait követik. Például annak ellenére, hogy az Informatika és a Villamosmérnöki tanszék ugyanabban az épületben van, és ugyanazt a hálózatot használja, különböző körzetekhez tartozhatnak. Hasonlóan, még ha az Informatika tanszék a Babbage Hallban és Turing Hallban megosztva helyezkedik is el, akkor mindkét épületben a hosztok normális esetben ugyanahhoz a körzethez tartoznak.

7.1.2. Erőforrás-nyilvántartás

Minden körzethez tartozhat egy **erőforrás-bejegyzés (resource record)** halmaz, attól függetlenül, hogy a körzet csak egyetlen hosztból áll, vagy egy elsődleges körzet. Ezek a

bejegyzések alkotják a DNS-adatbázist. Egy egyedüli hoszt esetén általában ez az erőforrás-bejegyzés csak az IP-cím, de ezenkívül még sok másféle erőforrás-bejegyzés létezik. A címfeloldó a DNS-nek küldött körzetnévhez tartozó erőforrás-bejegyzéseket kapja vissza. Ezek szerint a DNS igazi feladata az, hogy megfeleltesse a körzetnevet az erőforrás-bejegyzéseknek.

Az erőforrás-bejegyzés egy adatötösből áll. Annak ellenére, hogy az erőforrás-bejegyzéseket a hatékonyság miatt binárisan tárolják, a legtöbb ismertetőben az erőforrás-bejegyzések ASCII-formában szerepelnek, bejegyzésenként egy sorban. Az általunk használt formátum a következő:

```
Körzet_név    Élettartam    Osztály    Típus    Érték
```

A *Körzet_név* jelenti azt a körzetet, amelyhez a rekord tartozik. Normális esetben minden körzethez sok bejegyzés tartozik, és az adatbázis minden másolata több, körzettel kapcsolatos információt hordoz. Ez a mező az elsődleges kulcs a kereséshez. A bejegyzések sorrendje nem érdekes az adatbázisban.

Az *Élettartam* mező jelzést ad arról, hogy a bejegyzés mennyire stabil. A nagyon stabil információhoz nagy értékek tartoznak, mint a 86 400 (1 nap másodpercekben). Azokhoz az információkhoz, amelyek erősen ingatagok, kis értékek tartoznak, mint a 60 (1 perc). Ehhez a témához visszatérünk még, ha majd a gyorstárak használatát tárgyaljuk.

Minden erőforrás-bejegyzés harmadik mezője az *Osztály*. Az internethez tartozó információnál ez mindig *IN*. A nem internetes információhoz más kódokat lehet rendelni, de a gyakorlatban ilyet ritkán lehet látni.

A *Típus* mező a bejegyzés értékének típusára vonatkozik. Sokféle DNS-bejegyzés létezik. A legfontosabb típusok a 7.3. ábrán láthatók.

Típus	Jelentés	Érték
SOA	Lista kezdete	Ehhez a zónához tartozó paraméterek
A	Egy hoszt IPv4-címe	32 bites egész
AAAA	Egy hoszt IPv6-címe	128 bites egész
MX	Levelezőszerver	A körzethez tartozó levelezőszerver neve és prioritása
NS	Névszerver	A körzethez tartozó névszerver neve
CNAME	Kanonikus név	Körzetnév
PTR	Mutató	Álnév egy IP-címhez
SPF	Küldő házirendjének keretrendszere	A levélküldési házirend szöveges kódolása
SRV	Szolgáltatás	A szolgáltatást nyújtó hoszt
TXT	Szöveg	Tetszőleges ASCII-szöveg

7.3. ábra. *A DNS erőforrás-bejegyzés legfontosabb típusai*

Az *SOA* bejegyzés megadja az elsődleges információforrás nevét a zónához tartozó névszerverről (lásd alább), az adminisztrátor e-levél címét, egy egyedi sorozatszámot, valamint különböző jelzőket és időzítőket.

A legfontosabb bejegyzéstípus az *A (cím)* bejegyzés. Egy 32 bites IP-címet tartalmaz egy hoszt valamely hálózati csatolójához. Az ennek megfelelő *AAAA* vagy „négy A" bejegyzés 128 bites IPv6-címet tartalmaz. Minden internetes hosztnak legalább egy IP-címmel kell rendelkeznie, hogy más gépek kommunikálhassanak vele. Egyes hosztok kettő vagy több hálózati csatlakozással is rendelkeznek, ebben az esetben kettő vagy több *A* vagy *AAAA* erőforrás-bejegyzéssel rendelkeznek. Ebből következően a DNS egyetlen névhez több címet is visszaadhat.

Egy szokásos bejegyzéstípus az *MX* bejegyzés. Ez tartalmazza annak a hosztnak a nevét, amely kész a körzethez tartozó levelek fogadására. Azért használják, mert nincs minden gép felkészülve e-levél fogadására. Ha valaki e-levelet szeretne küldeni például a *bill@microsoft.com* címre, akkor a küldő hosztnak találnia kell egy levelezőszervert a *microsoft.com* körzetben, amelyik hajlandó fogadni az e-levelet. Az *MX* bejegyzés erről tud információt adni.

Egy másik fontos típus az *NS* bejegyzés, amely egy névszervert határoz meg a körzet vagy alkörzet számára. Ez egy olyan hoszt, amelynek másolata van a körzet adatbázisáról, és annak a névkeresési folyamatnak a részeként használják, amelyet hamarosan ismertetni fogunk.

A *CNAME* bejegyzések segítségével álneveket lehet létrehozni. Például ha valaki, aki ismeri az internetes névkonvenciókat, egy levelet szeretne küldeni valakinek, akinek a login neve *paul* az M.I.T. Informatika tanszékén, akkor úgy gondolhatja, hogy a *paul@cs.mit.edu* cím valószínűleg megfelelő. Valójában azonban ez a cím nem jó, mert az M.I.T. Informatika tanszékének körzete *csail.mit.edu*. Az M.I.T. azonban azok részére, akik ezt nem tudják, segítségképpen létrehozhat egy *CNAME* bejegyzést, ami átirányítja az embereket és programokat a helyes útra. Ezt megteszi például a következő bejegyzés:

```
cs.mit.edu      86400      IN CNAME      csail.mit.edu
```

A *CNAME*-hez hasonlóan a *PTR* is egy másik névre mutat. A *CNAME*-el ellentétben azonban, ami tulajdonképpen csak egy makró (például olyan mechanizmus, amely egy karakterláncot valamely másikra cserél), a *PTR* egy valódi DNS-adattípus, melynek értelmezése az adott környezettől függ. A gyakorlatban majdnem mindig arra használják, hogy egy nevet megfeleltessenek egy IP-címnek, hogy lehetővé váljon az IP-címek szerinti keresés, ahol a keresett gép neve az eredmény. Ezt hívják **fordított keresésnek** (**reverse lookup**).

Az *SRV* egy újabb bejegyzéstípus, amely lehetővé teszi, hogy egy hosztot valamilyen adott szolgáltatás elvégzésére kijelöljünk a körzeten belül. Például a *cs.washington.edu* webszervere lehet a *cockatoo.cs.washington.edu*. Ez a bejegyzés általánosítja az *MX* bejegyzéseket, amelyek ugyanezt a feladatot látják el, de csak e-levél szerverekhez használhatók.

Az *SPF* is egy újabb típusú bejegyzés. Ez lehetővé teszi olyan információ kódolását, amely megadja, hogy a körzetnek mely gépei küldenek levelet az internet többi részére. Ez megkönnyíti a címzett gépek számára a levél érvényességének ellenőrzését. Ha levél

```
; A cs.vu.nl-hez tartozó irányadó adatok
cs.vu.nl.     86400   IN   SOA     star boss (9527,7200,7200,241920,86400)
cs.vu.nl.     86400   IN   MX      1 zephyr
cs.vu.nl.     86400   IN   MX      2 top
cs.vu.nl.     86400   IN   NS      star

star          86400   IN   A       130.37.56.205
zephyr        86400   IN   A       130.37.20.10
top           86400   IN   A       130.37.20.11
www           86400   IN   CNAME   star.cs.vu.nl
ftp           86400   IN   CNAME   zephyr.cs.vu.nl

flits         86400   IN   A       130.37.16.112
flits         86400   IN   A       192.31.231.165
flits         86400   IN   MX      1 flits
flits         86400   IN   MX      2 zephyr
flits         86400   IN   MX      3 top

rowboat               IN   A       130.37.56.201
                      IN   MX      1 rowboat
                      IN   MX      2 zephyr

little-sister         IN   A       130.37.62.23

laserjet              IN   A       192.31.231.216
```

7.4. ábra. *A cs.vu.nl-hez tartozó lehetséges DNS-adatbázis egy része*

érkezik egy géptől, amely magát *dodgy*-nak nevezi, de az erőforrás-bejegyzések alapján a körzetből csak az *smtp* nevű gép küld levelet, akkor ez a levél valószínűleg hamisított levélszemét.

A lista végén szereplő *TXT* bejegyzések eredetileg arra szolgáltak, hogy a körzetek tetszés szerinti módon is azonosíthassák magukat. Manapság általában gépek számára olvasható információt kódolnak, tipikusan az *SPF* információt.

Utolsóként nézzük az *Érték* mezőt. Ez a mező tartalmazhat egy számot, egy körzetnevet vagy egy ASCII-karakterláncot. A szemantika a bejegyzés típusától függ. A legfontosabb bejegyzéstípusokhoz tartozó *Érték* mezők leírása a 7.3. ábrán látható.

Arra nézve, hogy milyen típusú információ található egy körzethez a DNS-adatbázisban, a 7.4. ábra ad útmutatást. Ez az ábra a 7.1. ábrán látható *cs.vu.nl* körzet (feltételezett) adatbázisának egy részét mutatja. Az adatbázis hét különböző típusú erőforrás-bejegyzést tartalmaz.

Az első, nem megjegyzés sor a 7.4. ábrán a körzetről ad némi alapinformációt, de ezzel tovább nem foglalkozunk. A következő két bejegyzés az elsődleges és másodlagos levélfogadó helyet adja meg, a *person@cs.vu.nl* címre érkező e-levelek számára. Először a *zephyr*-rel (egy specifikus géppel) kell próbálkozni. Ha nem sikerül, akkor a *top* következik. A következő sor meghatározza a körzet névszerverét (*star*).

Az üres sort (ami az olvashatóságot segíti) követő sorok megadják a *star*, *zephyr* és *top* IP-címeit. Ezeket egy álnév követi, a *www.cs.vu.nl*, így ezt a címet anélkül lehet használni, hogy kijelölnénk egy konkrét gépet. Az álnév létrehozása lehetővé teszi a *cs.vu.nl* számára,

hogy anélkül cserélje le világháló (world wide web) szerverét, hogy érvénytelenné válna a cím, amit az emberek az eléréshez használnak. Az *ftp.cs.vu.nl*-re is ugyanez vonatkozik.

A *flits* géphez tartozó szakaszban két IP-cím és három választási lehetőség található a *flits.cs.vu.nl*-re küldött e-levelek kezelésére. Az első választási lehetőség természetesen maga a *flits*, de amennyiben éppen nem üzemel, akkor a *zephyr* és a *top* jöhet szóba, mint második és harmadik lehetőség.

A következő három sor egy tipikus munkaállomás-leírást tartalmaz, esetünkben a *rowboat.cs.vu.nl*-ét. A megadott információ tartalmazza az IP-címet, az elsődleges és másodlagos levélfogadót. Ezután egy olyan számítógép leírása jön, amely önmagában nem képes leveleket fogadni, majd ezt követően egy, az internethez csatlakozó nyomtatóra vonatkozó bejegyzés

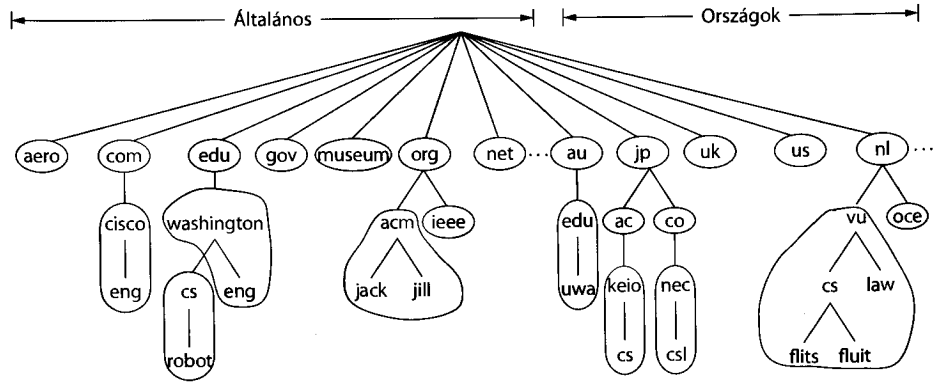
7.1.3. Névszerverek

Elvileg egyetlen szerver elegendő lenne a DNS-adatbázis tárolására, és a kérések megválaszolására. Gyakorlatilag ez a szerver annyira túl lenne terhelve, hogy használhatatlan lenne. Továbbá, ha egyszer csak felmondaná a szolgálatot, az egész internet lebénulna.

Az egyetlen szerver miatt adódó problémák elkerülése végett a DNS-névtér egymást nem átlapoló **zónákra** (**zones**) van osztva. A 7.1. ábra névterének egy lehetséges felosztása a 7.5. ábrán látható. Minden bekarikázott zóna a fa egy részét tartalmazza.

A zónán belüli zónahatárok meghatározása a szóban forgó zóna adminisztrátorától függ. A döntés meghozatalában nagy szerepet játszik, hogy hol és mennyi névszerverre van szükség. Például a 7.5. ábrán a University of Washingtonnak van egy zónája a *washington.edu*-hoz, ami kiszolgálja az *eng.washington.edu*-t, de a *cs.washington.edu*-t nem. Ez utóbbi egy különálló zóna, saját névszerverrel. Egy ilyen döntés akkor születik, ha egy tanszék, mint például az Angol nyelvi tanszék nem akar saját névszervert üzemeltetni, de például az Informatika tanszék igen.

Minden zóna egy vagy több névszerverhez is társítva van. Ezek olyan hosztok, amelyek a zóna adatbázisát tárolják. Normális esetben zónánként egy elsődleges névszerver



7.5. ábra. *A DNS-névtér egy része (karikázással jelölt) zónákra osztással*

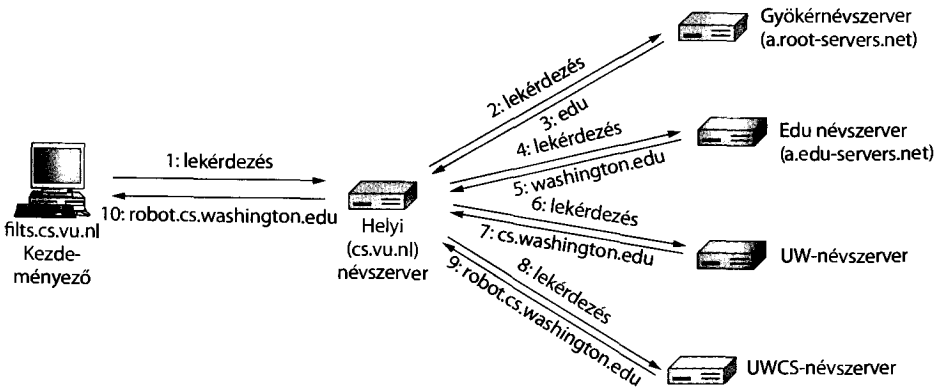
van, ami egy lemezen levő fájlból nyeri az információt, valamint egy vagy több másodlagos névszerver, amelyek az elsődleges névszervertől nyerik az információt. A megbízhatóság növelése érdekében a névszerverek egy része lehet a zónán kívül is.

Egy név megkeresésének és a hozzá tartozó cím meghatározásának folyamatát **névfeloldásnak** (**name resolution**) nevezik. Ha egy névfeloldó meg szeretne tudni valamit egy körzetnévről, akkor elküldi a lekérdezést egy helyi névszervernek. Ha a keresett körzet a névszerver hatáskörébe tartozik, mint ahogy például a *top.cs.vu.nl* a *cs.vu.edu* alá tartozik, akkor az visszaküldi a hiteles erőforrás-bejegyzéseket. A **hiteles bejegyzés** (**authoritative record**) azt jelenti, hogy a bejegyzés attól a szervtől származik, amelyik azt a bejegyzést kezeli, tehát mindig helyes. A hiteles bejegyzésekkel ellentétben a **gyorstárban levő bejegyzések** (**cached records**) esetleg idejétmúltak lehetnek.

Mi történik, ha a körzet távoli, például amikor a *flits.cs.vu.nl* akarja meghatározni a *robot.cs.washington.edu* IP-címét az UW-n (University of Washington)? Ebben az esetben, ha nincsen információ a helyi adatok közt, akkor a névszerver távoli lekérdezést kezdeményez. A lekérdezés folyamata a 7.6. ábrán látható. Az 1. lépésben a kezdeményező gép elküldi lekérdezését a helyi névszerverhez. Ez a lekérdezés tartalmazza a keresett körzet nevét, a típust (*A*) és az osztályt (*IN*).

A következő lépés a névhierarchia csúcsán álló egyik **gyökérnévszerver** (**root name server**) megkérdezése. Ezek a névszerverek minden elsődleges tartományról rendelkeznek információval. Ezt mutatja a 7.6. ábra 2. lépése. Ez az információ normális esetben a rendszer konfigurációs fájljában megtalálható, ami a DNS gyorstárába betöltődött a DNS-szerver indításakor. Ez egyszerűen csak a gyökér *NS* bejegyzéseinek listájából és a megfelelő *A* bejegyzésekből áll.

13 gyökérnévszerver létezik, melyeket nem túl ötletesen *a.root-servers.net*-tel kezdve *m.root-servers.net*-tel bezárólag hívnak. Elvileg lehetne minden gyökérszerver egy egyedülálló számítógép. Mivel azonban az egész internet működése függ a gyökérszerverektől, ezek nagy teljesítményű és erősen többszörözött számítógépek. A szerverek nagy része földrajzilag különböző helyeken található és bárkinek küldés (anycast) útválasztással érhetők el, amelynél a csomagokat a célcím legközelebbi előfordulásához



7.6. ábra. *Hogyan keres meg a címfeloldó egy távoli nevet tíz lépésben*

továbbítják (a bárkinek küldést az 5. fejezetben mutattuk be). A többszörözés megnöveli a megbízhatóságot és a teljesítőképességet.

A gyökérnévszerver valószínűleg nem ismeri az UW-n lévő gép IP-címét, és valószínűleg még az UW névszerverét sem ismeri. Ismernie kell azonban az *edu* körzet névszerverét, amelyben a *cs.washington.edu* is található, így ennek a nevét és IP-címét adja vissza a 3. lépésben.

A helyi névszerver ezután tovább folytatja a kutatást. Elküldi az egész lekérdezést az *edu* névszervernek (*a.edu-servers.net*). Ez a névszerver visszaadja az UW névszerverének nevét és IP-címét. Ez látszik a 4. és 5. lépéseken. Kicsit közelebb kerülve a célhoz, a helyi névszerver elküldi kérését az UW névszerveréhez (6. lépés). Ha a keresett körzet az Angol tanszékhez tartozott volna, megkaphatta volna a választ, mert az Angol tanszék az UW-zónába tartozik. Az Informatika tanszék azonban úgy döntött, hogy saját névszervert használ. A 7. lépésben az UW-névszerver visszaadja az UW Informatika tanszéke névszerverének nevét és IP-címét.

Végül a helyi névszerver az UW Informatika tanszék névszerveréhez fordul (8. lépés). Ez a szerver hiteles bejegyzésekkel rendelkezik a *cs.washinton.edu* körzetről, tehát tudnia kell a választ. Ezt az utolsó választ a 9. lépésben visszaküldi a helyi névszervernek, amely azt a *flits.cs.vu.nl*-nek továbbítja, megválaszolva annak kérdését (10. lépés). A névfeloldás megtörtént.

Ön is megpróbálhatja megvizsgálni a névfeloldási folyamatot olyan szabványos eszközök segítségével, mint például a *dig* program, ami a legtöbb UNIX-rendszeren telepítve van. Például az alábbi begépelése

```
dig @a.edu-servers.net robot.cs.washington.edu
```

lekérdezi a *robot.cs.washington.edu* IP-címét az *a.edu-servers.net* névszervertől, majd kiírja az eredményt. Ez egyrészt megmutatja a fenti példa 4. lépésében kapott információt, másrészt Ön megtudhatja az UW-névszerverek nevét és IP-címét.

Három technikai részletet is meg kell tárgyalnunk ennek a hosszú forgatókönyvnek a kapcsán. Az első részlet két eltérő lekérdezési mechanizmus működése, amelyek a 7.6. ábrán láthatók. Amikor a *flits.cs.vu.nl* hoszt elküldi lekérdezését a helyi névszervernek, akkor ez a névszerver végzi a névfeloldást a *flits* helyett, amíg meg nem kapja a kívánt választ, amit azután visszaküld a hosztnak. Részleges válaszokat *nem* küld vissza. Ezek hasznosak lehetnek ugyan, de a lekérdezés nem ezekre vonatkozott. Ezt az eljárást **rekurzív lekérdezésnek** (**recursive query**) nevezzük.

Másrészt a gyökérnévszerver nem (és a további névszerverek közül egyik sem) folytatja rekurzív módon a helyi névszerver kezdeményezte lekérdezést. Éppen csak visszaad egy részleges választ, majd továbbmegy a következő lekérdezésre. A helyi névszerver felelős a névfeloldás folytatásáért további lekérdezések kiadásával. Ezt az eljárást hívják **iteratív lekérdezésnek** (**iterative query**).

Egyetlen névfeloldás mindkét módszert használhatja, ahogyan azt a példa mutatta. Egy rekurzív lekérdezés mindig kívánatosabbnak tűnhet, de sok névszerver (főleg a gyökér) nem képes rá. Ehhez azok túlságosan leterheltek. Az iteratív lekérdezések a kezdeményező vállára helyezik a terhet. Annak, hogy a helyi névszerver támogatja a rekurzív lekérdezést, az az ésszerű magyarázata, hogy szolgáltatást nyújt a saját körzetéhez

tartozó hosztok részére. Ezeket a hosztokat nem kell egy teljes névszerver futtatására felkészíteni, elég, ha a helyit elérik.

A második technikai részlet a gyorstárazás. Minden válasz, beleértve a visszaadott részleges válaszokat is, a gyorstárakba kerül. Így ha egy másik *cs.vu.nl*-beli hoszt lekérdezi a *robot.cs.washington.edu* címét, a válasz már ismert lesz. Még jobb, ha egy hoszt ugyanannak a körzetnek egy másik hosztja címét kérdezi le, mondjuk a *galah.cs.washington.edu*-ét, mert a lekérdezést közvetlenül a hiteles névszervernek küldheti. Hasonlóképpen, a *washington.edu* egyéb körzeteire vonatkozó lekérdezéseket közvetlenül a *washington.edu* névszervernél lehet megkezdeni. A gyorstárakban lévő válaszok nagymértékben csökkentik a lépések számát és növelik a teljesítőképességet. Az eredetileg felvázolt forgatókönyvünk valójában a legrosszabb eset, ami akkor fordul elő, ha nincs használható információ a gyorstárakban.

A gyorstárakban lévő válaszok azonban nem hitelesek, mert a *cs.washington.edu*-nál történt változásokat nem frissítik a világ összes gyorstárában, ahol számon vannak tartva. Ezért kell, hogy a gyorstárbeli bejegyzések ne legyenek túl hosszú élettartamúak. Emiatt került be az *Élettartam* mező minden erőforrás-bejegyzésbe. Ez jelzi a távoli névszerverek számára, hogy meddig tárolják a gyorstárban a bejegyzést. Ha egy gépnek már évek óta ugyanaz az IP-címe, akkor lehet, hogy biztonságos ezt az információt 1 napig tárolni. A gyakrabban változó információ esetében biztonságosabb lehet, ha a bejegyzéseket néhány másodperc vagy egy perc elteltével kitörlik.

A harmadik technikai részlet a lekérdezésekhez, és az azokra adott válaszokhoz használt szállítási protokoll, ami az UDP. A DNS-üzeneteket, amelyek lekérdezéseket, válaszokat és a névfeloldás folytatásához használható névszervereket tartalmaznak, egyszerű formátumú UDP-csomagokban küldik. Nem fogunk belemerülni ennek a formátumnak a részleteibe. Ha rövid időn belül nem érkezik válasz, a DNS-ügyfél megismétli a lekérdezést, majd néhány ismételt próbálkozást követően a körzet más szerverénél próbálkozik. Ezt az eljárást úgy tervezték meg, hogy boldoguljon akkor is, ha a szerver leállt, és akkor is, ha a lekérdezés- vagy válaszcsomagok elvesztek. Minden lekérdezés tartalmaz egy 16 bites azonosítót, amely átmásolódik a válaszba, és így a névszerver illeszteni tudja a válaszokat a kérésekhez még abban az esetben is, ha több lekérdezés van folyamatban egyszerre.

Annak ellenére, hogy a DNS célkitűzése egyszerű, tisztában kell lenni azzal, hogy egy nagy és összetett elosztott rendszer, amely többmillió együttműködő névszervert tartalmaz. Kulcsfontosságú kapcsolatot biztosít az emberek számára olvasható körzetnevek és a gépek IP-címei között. Redundanciát és gyorstárazást tartalmaz a teljesítőképesség és megbízhatóság növelése érdekében, és rendkívül robusztusra tervezték.

Nem tárgyaltuk a biztonság kérdését, de könnyű elképzelni, hogy a név–cím megfeleltetések megváltoztatásának képessége pusztító következményekkel járhat, ha azt rosszindulatúan követik el. Ezért dolgozták ki a DNS számára a DNSsec nevű biztonsági bővítményeket. Ezeket a 8. fejezetben tárgyaljuk.

Az alkalmazások igénylik a nevek rugalmasabb módon történő használatát is, például egy megnevezett tartalomhoz tartozó olyan közeli hoszt IP-címének meghatározását, amelyik rendelkezik ezzel a tartalommal. Ez a modell megfelel egy film megkeresésének és letöltésének céljára. Ami számít, az a film, nem pedig a számítógép, amelynek másolata van róla, tehát mindössze egy olyan közelben lévő számítógép IP-címére vagyunk

kíváncsiak, ahol megvan a film egy példánya. Az ilyen leképezés végrehajtásának egy lehetséges módja a tartalommegosztó hálózatok alkalmazása. Ennek a fejezetnek egy későbbi, 7.5. szakaszában mutatjuk be, hogyan támaszkodnak ezek a DNS-re.

7.2. Elektronikus levél

Az **elektronikus levél** vagy **e-levél** (**e-mail**), ahogyan sok kedvelője nevezi, már több mint három évtizede használatban van. Olcsóbb és gyorsabb, mint a hagyományos levél, ezért az e-levél az internet kezdete óta népszerű alkalmazás. 1990 előtt jobbára csak a kutatók használták. Az 1990-es években aztán a nagyközönség is megismerte, és használata innentől kezdve exponenciális ütemben terjedt egészen addig, hogy mára a naponta elküldött e-levelek száma nagyságrendekkel meghaladja a **papíralapú levelekét** (**snail mail**). A hálózati kommunikáció egyéb formái, mint például az azonnali üzenetküldés és az IP-hálózaton keresztüli beszédátvitel használata nagymértékben megnőtt az elmúlt évtizedben, de az internetes kommunkáció igáslova az e-levél maradt. Az ipari szereplők széleskörűen alkalmazzák a vállalaton belüli kapcsolatokban, például arra, hogy a szerte a nagyvilágban, egymástól nagy távolságokra lévő alkalmazottak összetett projekteken dolgozhassanak. Sajnos, ahogyan a papíralapú levelek esetében is, az e-levelek nagy része – 10-ből kb. 9 – **kacatlevél** (**spam, junk mail**) [McAfee, 2010].

Akárcsak a kommunikáció egyéb formáinak, az e-levélnek is megvannak a saját konvenciói és stílusai. Mindenekelőtt nagyon informális, és túlságosan is csábító a használata. Azok az emberek, akik még álmukban se gondolnának arra, hogy telefonáljanak, sőt levelet írjanak egy Nagyon Fontos Személynek, egy másodpercig sem haboznak egy hanyagul megírt e-levél elküldésénél. A ranggal, életkorral és nemmel kapcsolatos fordulatok eltűnésével az e-levélváltások gyakran a tartalomra koncentrálnak, és nem a státusra. Az e-levél segítségével egy nyári munkára szerződött diák briliáns ötlete nagyobb hatású lehet, mint az ügyvezető alelnök ostobasága.

Az e-levél tele van olyan szlenggel, mint például a BTW (By The Way – apropó), ROTFL (Rolling On The Floor Laughing – gurulok a röhögéstől) és IMHO (In My Humble Opinion – szerény véleményem szerint). Sokan **mosolyikonnak** (**smiley, emoticon**) nevezett kis ASCII-szimbólumokat is használnak az e-leveleikben, például a mindenütt megtalálható „:-)"-t. Ha a jelkép nem lenne ismerős, forgassa el a könyvet az óramutató járásával megegyezően 90 fokkal. Ez a szimbólum és a többi mosolyikon segít az üzenet hangulatának közvetítésében. Ezek a kommunikáció egyéb tömör formáiban, például az azonnali üzenetküldésben is elterjedtek.

Az e-levél protokolljai is kialakultak a használatuk során. Az első e-levél rendszerek egyszerű fájlátviteli protokollokból álltak azzal a konvencióval, hogy az üzenetek (azaz a fájlok) első sora a címzettre utalt. Az idő múlásával azonban az e-levél eltávolodott a fájlátviteltől és számos képességgel ruházták fel, például az egyetlen üzenet több címzettnek való továbbításának lehetőségével. A mutimédiás képességek az 1990-es évek során váltak fontossá, hogy az üzenetekkel képeket és egyéb nem szöveges anyagokat is el lehessen küldeni. Az e-levelek olvasására szolgáló programok is sokkal összetettebbé váltak, a szöveges felhasználói felületről fokozatosan grafikusra váltottak, és lehetővé

tették a felhasználóknak, hogy elérjék leveleiket laptopjaikról, bárhol is legyenek. Végül, a kacatlevél dominanciája miatt a levelek olvasására szolgáló programoknak és a levéltovábbító protokolloknak figyelmet kell fordítaniuk a kéretlen levelek megtalálására és törlésére is.

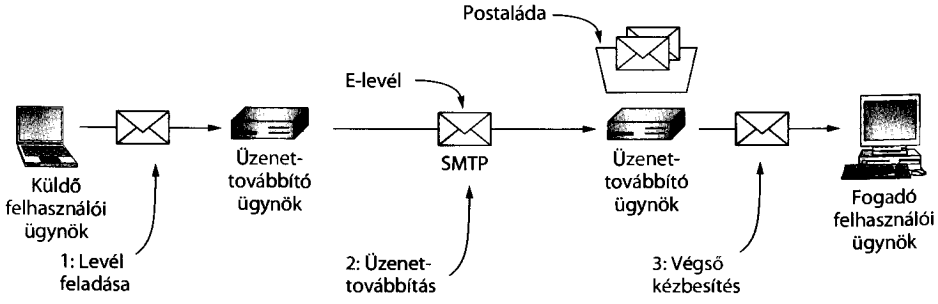
Az e-levelek ismertetésekor jobban fogunk koncentrálni az üzenetek felhasználók közötti továbbítására, mint a levélolvasó programok megjelenésére. Az architektúra átfogó ismertetését követően mégis a rendszernek a felhasználó által látható részével kezdjük az ismerkedést, mert ez a legtöbb olvasó számára ismerős.

7.2.1. Architektúra és szolgáltatások

Ebben a részben átfogó ismertetést adunk az e-levél rendszerek felépítéséről és arról, hogy mire képesek. Az e-levél rendszerek architektúrája a 7.7. ábrán látható. Két alrendszerből állnak, a **felhasználói ügynökből (user agent)**, amely lehetővé teszi a felhasználók számára az üzenetek olvasását és küldését, valamint az **üzenettovábbító ügynökből (message transfer agent)**, ami a leveleket eljuttatja a feladótól a címzettig. Az üzenettovábbító ügynökökre informálisan **levelezőszerverekként (mail server)** is fogunk hivatkozni.

A felhasználóiügynök-program grafikus, ritkábban szöveges és parancsalapú csatlakozó felületet nyújt az e-levél rendszerrel való érintkezésre. Ez magában foglalja az üzenetek írásához, megválaszolásához, a beérkező üzenetek megjelenítéséhez valamint az üzenetek iktatással, kereséssel és törléssel való rendszerezéséhez szükséges eszközöket. Az új üzeneteknek kézbesítés céljából levelezőrendszerbe történő küldését a **levél feladásának (mail submission)** nevezzük.

A felhasználói ügynök tevékenységeinek egy része automatizálható, feltételezve a felhasználó szándékát. Például a beérkező leveleket meg lehet szűrni törlés céljából, vagy hátrébb lehet sorolni a fontossági sorrendben azokat az üzeneteket, amelyek valószínűleg kacatlevelek. Néhány felhasználói ügynök fejlett szolgáltatásokat is tartalmaz, például automatikusan válaszol az e-levelekre („Most éppen a csodálatos szabadságomat töltöm, és el fog tartani egy darabig, mire visszaérek."). A felhasználói ügynök azon a számítógépen fut, amelyen a felhasználó olvassa a leveleit. Ez is csak egy program, amit időnként le lehet futtani.



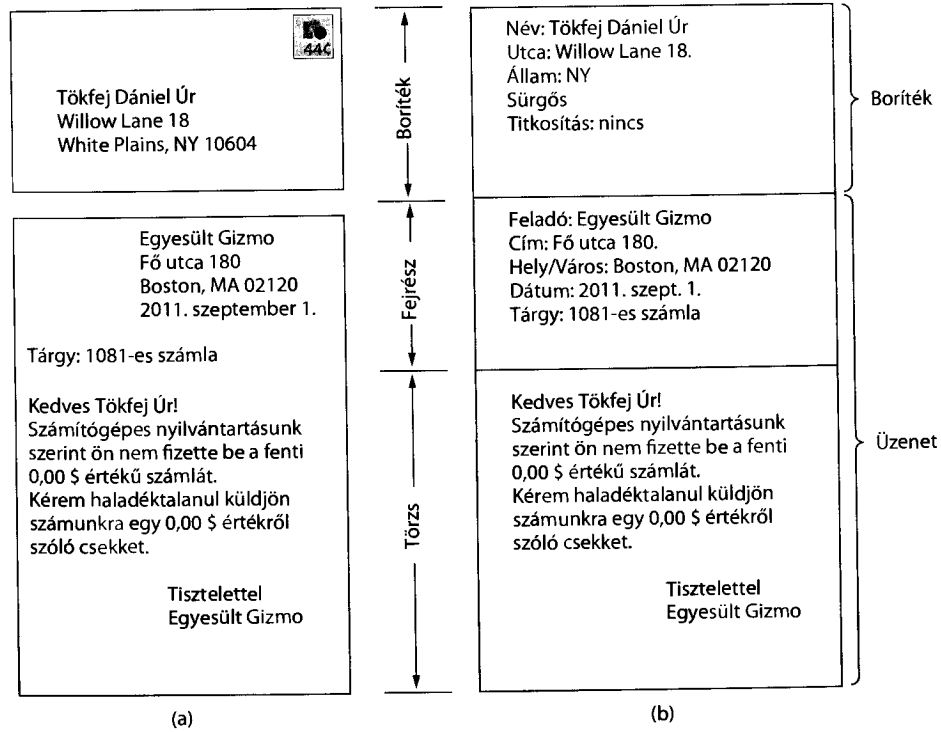
7.7. ábra. *Az e-levél rendszer architektúrája*

Az üzenettovábbító ügynökök általában rendszerfolyamatok. Ezek a levelezőszervergépeken a háttérben futnak abból a célból, hogy mindig elérhetők legyenek. Feladatuk, hogy a rendszeren keresztül automatikusan eljuttassák az e-leveleket a feladótól a címzettig az **SMTP (Simple Mail Transfer Protocol – egyszerű levéltovábbító protokoll)** segítségével. Ez az üzenettovábbítási lépés.

Az SMTP-t eredetileg az RFC 821 részletezte, aminek a módosításával jött létre a jelenleg érvényes RFC 5321. Ez összeköttetéseket használ a levelek elküldésére, és kézbesítés státusának, valamint az esetleges hibáknak a visszaküldésére. Számos alkalmazás létezik, amelyekben a továbbítás nyugtázása fontos, és akár jogi jelentősége is lehet („Tisztelt bíróság! Az e-levél rendszerem nem éppen megbízható, ezért úgy vélem, hogy az elektronikus idézés elveszett valahol.").

Az üzenettovábbító ügynökök **levelezési lista (mailing list)** funkciót is megvalósítanak; az üzenet pontos másolatait mindenkinek továbbítják, aki szerepel az e-levél címlistáján. A fejlett szolgáltatások közé tartoznak még a szokásos másolatok (carbon copy) és vakmásolatok (blind carbon copy) készítése, a sürgős e-levelek, titkos (titkosított) e-levél, alternatív címzettnek szóló e-levél készítése arra az esetre, ha az elsődleges pillanatnyilag nem elérhető, valamint annak lehetősége, hogy a titkárnők elolvassák és megválaszolják főnökük e-levelét.

A felhasználói ügynökök és az üzenettovábbító ügynökök összekapcsolása adja a postaláda és a szabványos e-levél formátum koncepcióját. A **postaládák (mailbox)** tárolják



7.8. ábra. *Borítékok és üzenetek. (a) Postai levél. (b) Elektronikus levél*

a felhasználók által kapott leveleket. Ezeket a levelezőszerverek kezelik. A felhasználói ügynökök egyszerűen csak megmutatják a felhasználóknak postaládáik tartalmát. Ennek érdekében a felhasználói ügynökök levelezőszerver-parancsokat küldenek a postaládák manipulálása, azok tartalmának megvizsgálása céljából, üzenetek törlése érdekében és így tovább. A levélnek a postaládából történő kivétele a levél kézbesítése (3. lépés a 7.7. ábrán). Ennek az architektúrának a segítségével egy felhasználó több számítógépen különféle felhasználói ügynököket használhat postaládájának elérésére.

A levéltovábbító ügynökök a levelet szabványos formában továbbítják egymásnak. Az eredeti RFC 822 formátum módosításával, valamint a multimédiás tartalmak és a nemzetközi szövegek támogatásával jött létre a jelenleg érvényes RFC 5322. Ezt a sémát MIME-nak nevezik, és később részletesen tárgyaljuk. Mindezek ellenére az emberek az internetes e-levelekre még mindig RFC 822-ként hivatkoznak.

A kulcsötlet az üzenetformátumban a **boríték (envelope)** és a tartalom különválasztása. A boríték magába foglalja az üzenetet. Tartalmazza az üzenet továbbításához szükséges információkat, mint a címet, a prioritást és biztonsági szintet, amelyek mindegyike az üzenettől teljesen elkülönül. Az üzenettovábbító ügynökök, a postához hasonlóan a borítékot használják az útvonal meghatározására.

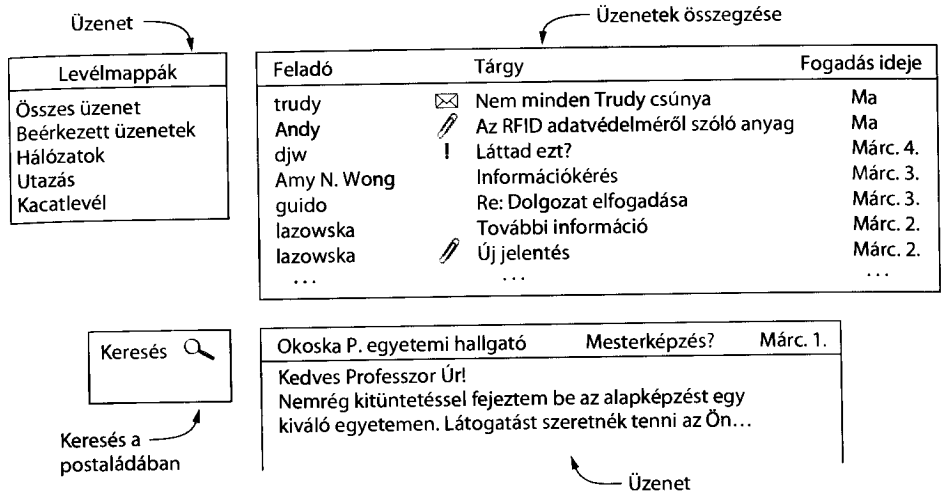
A borítékon belüli üzenet két részből áll: a **fejlécből (header)** és a **szövegrészből vagy törzsből (body)**. A fejléc vezérlési információt tartalmaz a felhasználói ügynökök részére. A szövegrész teljesen az emberi címzettnek szól. Egyik ügynök sem foglalkozik sokat vele. A borítékok és szövegrészek a 7.8. ábrán láthatók.

A továbbiakban alaposabban is megvizsgáljuk az architektúra részeit, megnézzük azokat a lépéseket, amelyek az e-levél egyik felhasználótól másikig történő elküldéséhez szükségesek. Ez az utazás a felhasználói ügynökkel kezdődik.

7.2.2. A felhasználói ügynök

A felhasználói ügynök általában egy program (melyet néha **levélolvasónak** vagy **üzenetolvasónak** neveznek), ami számtalan, az üzenetek létrehozásával, fogadásával, azok megválaszolásával és a postaládák kezelésével kapcsolatos parancsot képes értelmezni. Sok népszerű felhasználói ügynök létezik, mint például a Google gmail, Microsoft Outlook, Mozilla Thunderbird és az Apple Mail. Ezek külső megjelenése nagyon változatos. Egyes felhasználói ügynököknek menü- vagy ikonvezérelt felhasználói felülete van, melynek kezeléséhez egér, vagy a kisebb mobil eszközök esetén érintőképernyő szükséges. A régebbi felhasználói ügynökök, mint az Elm, mh és a pine szöveges felületet adnak, és 1 betűs parancsokat várnak a billentyűzetről. Funkcionálisan ezek megegyeznek, legalábbis a szöveges üzenetek esetében.

Egy felhasználói ügynök kezelőfelületének tipikus elemei láthatóak a 7.9. ábrán. Az Ön levélolvasója valószínűleg sokkal mutatósabb, de feltehetőleg egyenértékű funkciókkal bír. Amikor egy felhasználói ügynök elindul, általában összegzést ad a felhasználó postaládájában lévő üzenetekről. Az összegzés gyakran üzenetenként egyetlen sorból áll, és a sorok valamilyen szempont szerint rendezettek. Kiemeli az üzenet borítékjából vagy fejlécéből vett fontos mezők tartalmát.



7.9. ábra. *A felhasználói ügynök kezelőfelületének tipikus elemei*

A 7.9. ábrán hét összegző sor látható. A sorok a *Feladó*, *Tárgy* és *Fogadás ideje* mezőket tartalmazzák, ebben a sorrendben, amelyek megmutatják, hogy ki küldte az üzenetet, miről szól és mikor érkezett meg. Minden információ felhasználóbarát módon, az üzenetmezők alapján, de nem azok szó szerinti szöveges tartalmával jelenik meg. Tehát azok, akik elfelejtik megadni a *Tárgy* mezőt, gyakran azt tapasztalják, hogy a leveleikre adott válaszok nem szokták megkapni a legmagasabb prioritást.

Sok más mező vagy jelzés használata lehetséges. A 7.9. ábrán az üzenet tárgya mellett látható ikonok jelezhetik például az olvasatlan levelet (boríték), csatolmányt (gemkapocs) és a fontos levelet, amelyet legalábbis a küldő fontosnak tart.

Sokféle szempont szerinti rendezés is elképzelhető. A legáltalánosabb az üzenetek fogadási ideje szerinti sorba rendezés, kezdve a legújabbal, valamilyen jelzéssel ellátva arra vonatkozólag, hogy az üzenet új, vagy a felhasználó már elolvasta. Az összegzésben lévő mezőket és a rendezést a felhasználó igényének megfelelően testre szabhatja.

A felhasználói ügynököknek igény szerint meg kell tudniuk jeleníteni a beérkezett üzeneteket, hogy az emberek el tudják olvasni e-leveleiket. Gyakran az üzenet rövid előnézetét is megjelenítik, mint a 7.9. ábrán is, hogy segítsenek a felhasználóknak eldönteni, mikor olvassák tovább. Az előnézetek kis ikonokat vagy képeket használhatnak az üzenet tartalmának jellemzésére. A megjelenítéssel kapcsolatos tevékenységek közé tartozik a szöveg újraformázása, hogy kiférjen a kijelzőre, fordítás vagy a tartalom kényelmesebb formátumokra alakítása (például digitalizált beszéd átalakítása szöveggé).

Miután a felhasználó elolvasta az üzenetet, eldöntheti, hogy mit tegyen vele. Ez az **üzenet elrendezése (message disposition)**. A lehetőségek közé tartozik az üzenet törlése, megválaszolása, továbbítása egy másik felhasználónak és az üzenet megtartása későbbi hivatkozás céljából. A legtöbb felhasználói ügynök képes a beérkezett leveleket tároló postaláda és a hozzá tartozó több, mentett leveleket tartalmazó mappa kezelésére. A mappák lehetővé teszik a felhasználó számára, hogy az üzeneteket feladók, témák vagy valamilyen más kategória szerint mentse el.

A felhasználói ügynök az iktatást automatikusan is elvégezheti, még mielőtt a felhasználó elolvasná az üzeneteket. Egy egyszerű példa erre, amikor az üzenet mezőinek és tartalmának megvizsgálása után, a felhasználó korábbi visszajelzéseit felhasználva kideríti, hogy az üzenet kacatlevél-e. Sok ISP és vállalat olyan szoftvert futtat, ami a leveleket fontos vagy kacatlevél címkével jelöli meg, így a felhasználói ügynök ennek megfelelően iktathatja ezeket. Az ISP-k és a vállalatok abban az előnyös helyzetben vannak, hogy sok felhasználó leveleit láthatják és lehetnek listáik az ismert kacatlevélküldőkről. Ha több száz felhasználó szinte egyszerre kap hasonló üzenetet, akkor az valószínűleg kacatlevél. A felhasználói ügynök azzal, hogy előre szétválogatja a leveleket a „valószínűleg szabályos" és a „valószínűleg kacatlevél" kategóriák szerint, a felhasználókat jelentős mennyiségű munkától kíméli meg.

Melyek a leggyakrabban előforduló kacatlevelek? Ezeket a fertőzött számítógépekből álló, ún. **hálózati robotok (botnet)** állítják elő, és a levelek tartalma attól függ, hogy a címzett hol él. Ázsiában hamis diplomákat kínálnak, az Egyesült Államokban olcsó gyógyszereket és egyéb kétes termékeket ajánlanak. Érvénytelen nigériai bankszámlákból még mindig sok van. A különféle testrészeket megnövelő pirulák mindenütt megtalálhatók.

A felhasználók más iktatási szabályokat is meghatározhatnak. Minden egyes szabály feltételből és cselekvésből áll. Például egy szabály kimondhatja, hogy a főnöktől érkező összes üzenet egy mappába kerül azonnali olvasás céljából, egy bizonyos levelezőlista üzenetei pedig egy másikba, későbbi elolvasás végett. Számos mappa látható a 7.9. ábrán. A legfontosabb mappák a Beérkezett üzenetek (Inbox) mappa, amiben azok a levelek vannak, amelyek nem lettek máshová áthelyezve, valamint a Kacatlevél (Junk Mail) mappa a kacatlevélnek vélt üzenetek számára.

Az olyan explicit szerkezeteken kívül, mint amilyenek a mappák, a felhasználói ügynököknek manapság a postaládában való keresésre számos más képességük is van. Ez a képesség is látszik a 7.9. ábrán. A keresési lehetőségekkel a felhasználók gyorsan megtalálhatnak olyan, valaki által a múlt hónapban küldött üzenetet, amiben például arról van szó, hogy „hol lehet olcsón okostelefont venni".

Az e-levél hosszú utat tett meg az egyszerű fájlátvitel óta. A szolgáltatók ma már rendszeresen adnak postaládákat akár 1 GB tárhellyel, amiben egy felhasználó levélváltásai hosszú időre megmaradnak. A felhasználói ügynökök kifinomult levélkezelése, a keresés és az automatikus feldolgozás különféle formái teszik lehetővé, hogy az e-levelek ilyen nagy mennyiségben kezelhetők. Azok számára, akik évente több ezer üzenetet küldenek és fogadnak, ezeknek az eszközöknek az értéke felbecsülhetetlen.

Egy másik hasznos lehetőség valamilyen módon automatikusan reagálni a levelekre. Egy reakció lehet a beérkező e-levél továbbítása egy másik címre, például egy kereskedelmi személyhívó szolgáltató által üzemeltetett számítógép, amely rádiós vagy műholdas összeköttetéssel követi a felhasználót, és megjeleníti a *Tárgy:* sort a személyhívóján. Ezeknek az **automatikus válaszadóknak vagy automatikus reagálóknak (autoresponder)** a levelezőszerveren kell futniuk, mert a felhasználói ügynök nem mindig fut, és csak alkalomszerűen tölti le az e-leveleket. Emiatt a felhasználói ügynök nem tudja megvalósítani a valódi automatikus válaszadást. Az automatikus válaszadás kezelői felületét azonban általában a felhasználói ügynök biztosítja.

Az automatikus reagálásra egy másik példa a **távollétet jelző ügynök (vacation agent)**. Ez egy olyan program, amely megvizsgálja a beérkező levelet és olyan unalmas

választ küld a feladónak, mint például „Szia! Nyaralni mentem. Augusztus 24-én jövök vissza. Utána válaszolok." Az ilyen válaszok megadhatják, hogy a közbenső időben felbukkanó sürgős esetekben hogyan kell eljárni, vagy megadhatják azoknak a személyeknek a nevét, akikhez a különböző problémákkal fordulni kell stb. Legtöbb távollétet jelző démon számon tartja, hogy kinek küldött már ilyen konzervlevelet, és többször nem ismétli meg azt. Ezek az ügynökök azonban csapdákat is rejtegetnek. Például nem tanácsos konzervlevelet küldeni válaszként egy nagy levelezőlistáról érkező üzenetre.

Most vizsgáljuk meg azt a forgatókönyvet, amikor az egyik felhasználó üzenetet küld a másiknak. Nem beszéltünk még a felhasználói ügynök egy másik alapvető szolgáltatásáról, a levél összeállításáról. Ez magába foglalja az üzenetek, illetve az üzenetekre adott válaszok elkészítését, és ezeknek az üzeneteknek a levelezőrendszer többi részébe való küldését kézbesítés céljából. Bár akármilyen szövegszerkesztő használható az üzenet törzsének elkészítéséhez, a szövegszerkesztők általában be vannak építve a felhasználói ügynökbe abból a célból, hogy segítsenek a címzésnél és az üzenethez kapcsolt számos fejlécmező kitöltésénél. Például egy üzenet megválaszolásakor az e-levél rendszer kiveheti a feladó címét a beérkezett e-levélből és automatikusan beszúrhatja azt a válasz megfelelő helyére. Az általános képességek közé tartozik még az **aláírásblokk (signature block)** hozzáfűzése az üzenet aljához, a helyesírás ellenőrzése és az üzenet érvényességét mutató digitális aláírások kiszámítása.

A levelezőrendszerbe küldött üzeneteknek szabványos formátuma van, amelyet a felhasználói ügynöknek megadott információ alapján kell előállítani. Az üzenetnek a továbbítás szempontjából legfontosabb része a boríték, a boríték legfontosabb része pedig a címzett címe. Ennek a címnek olyan formátumúnak kell lennie, amit az üzenettovábbító ügynök képes feldolgozni.

A cím elvárt formája *felhasználó@dns-cím*. Mivel tanulmányoztuk a DNS-t korábban ebben a fejezetben, azt az anyagot nem fogjuk itt megismételni. Érdemes azonban megjegyezni, hogy a címzésnek más formái is léteznek. Főleg az **X.400** címek néznek ki egészen másként, mint a DNS-címek.

Az X.400 egy üzenetkezelő rendszerek számára készült ISO-szabvány, ami egykor az SMTP versenytársa volt. Az SMTP legyőzte, de az X.400-as rendszerek még mindig használatban vannak, többnyire az Egyesült Államokon kívül. Az X.400 címek *attribútum=érték* párokból tevődnek össze, amelyeket / jelek választanak el egymástól. Például:

```
/C=US/ST=MASSACHUSETTS/L=CAMBRIDGE/PA=360 MEMORIAL DR./CN=KEN SMITH/
```

Ez a cím megadja az ország nevét, az állam nevét, a helység nevét, az utca nevét és a házszámot, és egy szokványos személynevet (Ken Smith). Sok más attribútum is szóba jöhet, így annak is lehet levelet küldeni, akinek a pontos e-levél címe ugyan nem ismert, de ismert elegendő számú más attribútuma (például cég és betöltött állás).

Bár az X.400-címek jóval kényelmetlenebbek, mint a DNS-nevek, ez eldönthetetlen kérdés, mivel a felhasználói ügynökök felhasználóbarát álnevekkel rendelkeznek (ezeket néha beceneveknek is hívják), melyek lehetővé teszik, hogy a felhasználók bevigyenek vagy kiválasszanak egy személynevet, és megkapják a pontos e-levél címet. Tehát nem feltétlenül szükséges ténylegesen beírni ezeket a furcsa karakterláncokat.

Az utolsó pont, amire ki kell térnünk a levélküldéssel kapcsolatban, az a levelezési listákkal kapcsolatos. Ezek lehetővé teszik a felhasználóknak, hogy ugyanazt az üzenetet egyetlen parancs segítségével elküldjék egy listán szereplő összes embernek. Kétféleképpen is lehet működtetni egy levelezési listát. Működtetheti helyileg a felhasználói ügynök. Ebben az esetben a felhasználói ügynök külön levelet küld minden egyes címzettnek.

A másik lehetőség, hogy a listát a távolban egy üzenettovábbító ügynök üzemelteti. Az üzenetek terjesztésére ekkor az üzenettovábbító rendszerben kerül sor, amelynek az a hatása, hogy több felhasználó is beküldhet levelet a listára. Például, ha egy madárbarát csoportnak van egy *birders* nevű levelezési listája, ami a *meadowlark.arizona.edu*-n üzemel, akkor minden, a *birders@meadowlark.arizona.edu* címre küldött levél először eljut az Arizonai Egyetemre, és csak ott válik szét a lista tagjainak megfelelő üzenetekre, bárhol éljenek is a világban. A levelezési lista használói a címből nem tudják megállapítani, hogy ez egy levelezési lista. Lehet ez akár Gabriel O. Birders professzor személyes postaládája.

7.2.3. Üzenetformátumok

Térjünk most át a felhasználói felületről magukra a levélformátumokra. A felhasználói ügynök által küldött üzeneteknek szabványos formátumúnak kell lenniük, hogy az üzenettovábbító ügynökök képesek legyenek kezelni azokat. Először az alap ASCII e-leveleket vizsgáljuk meg az RFC 5322 használatával, ami az eredeti, RFC 822-ben ismertetett internet-üzenetformátumnak utolsó átdolgozása. Azután megvizsgáljuk az alapformátum multimédia-bővítményeit.

RFC 5322 – Az internet üzenetformátuma

A levelek egy egyszerű borítékból (amit az SMTP részeként az RFC 5321 tárgyal), néhány fejlécmezőből, egy üres sorból és az üzenetrészből állnak. Minden fejlécmező (logikailag) egyetlen ASCII-szövegű sorból áll, amely tartalmazza a mező nevét, egy kettőspontot és a legtöbb mezőnél egy értéket. Az RFC 822-t évtizedekkel ezelőtt tervezték, és nem különbözteti meg tisztán egymástól a boríték- és a fejlécmezőket. A szabványt kicsit átdolgozták ugyan az RFC 5322-ben, teljesen átalakítani azonban nem lehetett a kiterjedt használata miatt. Normális esetben a felhasználói ügynök összerakja a levelet, majd átadja azt az üzenettovábbító ügynöknek, amely aztán a fejléc egyes mezőiből összeállítja a tényleges borítékot, ami némileg régimódi keveréke a borítéknak és üzenetnek.

Az üzenettovábbítással kapcsolatos legfontosabb mezőket a 7.10. ábrán soroltuk fel. A *To:* mező az elsődleges címzett DNS-címét tartalmazza. Egy üzenetnek több címzettje is lehet. A *Cc:* mező az esetleges másodlagos címzettek címét adja meg. A továbbításnál nincs különbség az elsődleges és másodlagos címzettek között. Ez teljesen pszichológiai természetű megkülönböztetés, amely pusztán a levelezőknek lehet fontos, azonban a levelezőrendszernek nem. A *Cc:* (Indigós másolat – Carbon copy) elnevezés bevett szokás, azonban kissé elavult, hiszen a számítógépek nem használnak indigót. A *Bcc:* (Vak indigós másolat – Blind carbon copy) hasonló a *Cc:* mezőhöz, ez a sor azonban kitörlődik minden elsődleges és másodlagos címzettnek küldött másolatból. Ez a sajátosság

Fejlécmező	Jelentése
To:	Az elsődleges címzett(ek) e-levél címe(i)
Cc:	A másodlagos címzett(ek) e-levél címe(i)
Bcc:	A vakmásolat címzettjének/címzettjeinek e-levél címe(i)
From:	A levél szerzőjének neve
Sender:	A tényleges feladó e-levél címe
Received:	Az úton minden üzenettovábbító ügynök hozzáad egy ilyen sort a levélhez
Return-Path:	Arra használható, hogy megadjon egy visszafele vezető utat a feladóhoz

7.10. ábra. *Az RFC 5322-es üzenettovábbítással kapcsolatos fejlécmezői*

lehetővé teszi, hogy kívülállóknak is lehessen másolatot küldeni anélkül, hogy azt az elsődleges és másodlagos címzett megtudná.

A következő két mező a *From:* és *Sender:* rendre az üzenet szerzőjét, valamint elküldőjét adja meg. Ezek nem feltétlenül egyeznek meg. Például egy cégvezető megír egy levelet, de a titkárnője küldi el azt. Ebben az esetben a cégvezető szerepel a *From:* mezőben, míg a *Sender:* mezőben a titkárnő jelenik meg. A *From:* mező kötelező, a *Sender:* mező elhagyható, ha értékük megegyezik. Ezek a mezők abban az esetben szükségesek, ha a levél nem továbbítható és vissza kell küldeni a feladónak.

A levél útja mentén minden üzenettovábbító ügynök hozzáad egy *Received:* mezőt tartalmazó sort a levélhez. Ez a sor tartalmazza az ügynök azonosítóját, a dátumot, az időt és más információt, amelyek a forgalomirányító rendszerben levő hibák felderítéséhez használhatók.

A *Return-Path:* mezőt az utolsó üzenettovábbító ügynök ragasztja a levélhez, és arra szolgál, hogy megadja a feladóhoz visszavezető utat. Elvileg ez az információ a *Received:* mezőkből is összeállítható (kivéve a feladó postafiókjának nevét). Ez a mező azonban ritkán van kitöltve, és többnyire csak a feladó címét tartalmazza.

A 7.10. ábrán látható mezőkön kívül az RFC 5322 levelei tartalmazhatnak még néhány, a felhasználói ügynökök vagy az emberek által használt mezőt. A legáltalánosabbak ezek közül a 7.11. ábrán láthatók. Legtöbbjük jelentése könnyen megérthető, ezért nem ismertetjük mindegyiket részletesen.

A *Reply-To:* mezőt akkor használják, ha sem a szerző, sem a feladó nem szeretné látni a választ. Tegyük fel például, hogy egy marketingmenedzser ír egy e-levelet, amely egy új termékről számol be az üzletfeleknek. A levelet a titkárnő küldi el, de a *Reply-To:* mezőben az eladási osztály vezetőjének címe szerepel, aki képes a kérdéseket megválaszolni, és a rendeléseket felvenni. Ez a mező akkor is hasznos, ha a feladónak két e-mail címe van, és a másikra kéri a választ.

A *Message-Id:* egy automatikusan előállított szám, amit az üzenetek összekapcsolására (például amikor az *In-Reply-To:* mezőben használják) és a többszörös kézbesítés megakadályozására használnak.

Az RFC 5322 határozottan kijelenti, hogy a felhasználók kitalálhatnak saját használatra új fejlécmezőket. Az RFC 822 óta létező szabály alapján ezeknek *X-* karakterlánccal

Fejlécmező	Jelentése
Date:	Az üzenet küldésének dátuma és ideje
Reply-To:	A választ erre az e-levél címre kell küldeni
Message-Id:	Egyedi üzenetazonosításra használható szám
In-Reply-To:	Annak a levélnek a Message-Id-je, amire ez a válasz
References:	Más kapcsolódó levelek Message-Id-je
Keywords:	A felhasználó által választott kulcsszavak
Subject:	A levél tartalmának rövid összefoglalása az egysoros megjelenítéshez

7.11. ábra. *Néhány, az RFC 5322 üzenetek fejlécében előforduló mező*

kell kezdődniük. A saját használatú és hivatalos mezők közti konfliktus elkerülése végett garantált, hogy semelyik hivatalos fejlécmező nem fog a jövőben sem *X*--lel kezdődni. Néhány okostojás egyetemista az *X-Fruit-of-the-Day:* vagy *X-Disease-of-the-Week:*-hez hasonló mezőket ragaszt a levelekhez, amelyek legálisak ugyan, de nem mindig az értelemről tanúskodnak.

A fejlécmezők után következik az üzenetrész. A felhasználók azt írnak ide, amit akarnak. Sokan ASCII-rajzokat tartalmazó aláírással, kisebb-nagyobb költőktől származó idézetekkel, politikai megjegyzésekkel, felelősségelhárító nyilatkozatokkal zárják leveleiket (például: Az XYZ cég nem felelős azért, amit mondok; valójában nem is érti meg).

MIME – többcélú hálózati levelezéskiterjesztés

Az ARPANET korai szakaszában az e-levelek kizárólag ASCII-karakterekből álló angol nyelven írt szöveges üzenetekből álltak. Ehhez tökéletesen megfelelt az RFC 822: megadta a fejlécmezőket, de a tartalmat teljesen a felhasználó tetszésére bízta. Az 1990-es években a világszerte használt internet és a gazdagabb tartalmak levelezőrendszeren keresztüli elküldésének igénye miatt ez a megközelítés már nem volt megfelelő. Problémák merültek fel az ékezetes betűket tartalmazó nyelveken (például francia, német) írt üzenetek küldésénél és fogadásánál, azoknál, amelyek nem latin betűkkel (például héber és orosz) vagy ábécé nélküli nyelveken íródtak (például kínai és japán), valamint a szöveget egyáltalán nem tartalmazó üzeneteknél (például hang, kép, vagy bináris dokumentumok és programok).

A megoldás a **MIME (Multipurpose Internet Mail Extensions – többcélú hálózati levelezés kiterjesztés)** kifejlesztése volt. Széles körben alkalmazzák az interneten átküldött üzeneteknél, de a tartalom meghatározásához más alkalmazásokban is, például web böngészésnél. A MIME részleteit az RFC 2045–2047, 4288, 4289 és 2049 tartalmazza.

A MIME alapötlete az, hogy használja tovább az RFC 822 által leírt formát (ez volt az RFC 5322 előfutára a MIME-javaslat idején), de adjon struktúrát az üzenet szövegrészének, és definiáljon kódolási szabályokat a nem ASCII-üzenetek átvitele számára.

Fejlécmező	Jelentése
MIME-Version:	Azonosítja a MIME-verziót
Content-Description:	Ember által olvasható leírás az üzenet tartalmáról
Content-Id:	Egyedi azonosító
Content-Transfer-Encoding:	Megadja, hogy a szövegrész milyen módon lett csomagolva az átvitelre
Content-Type:	Megadja az üzenet típusát és formátumát

7.12. ábra. *A MIME által hozzáadott új fejlécmezők*

Azzal, hogy a továbbítandó MIME-üzenetek nem térnek el az RFC 822-től, tovább lehet használni a meglevő levéltovábbító ügynököket és protokollokat (amik az RFC 821-en alapultak akkor, és az RFC 5321-en ma). Csupán a küldésre és fogadásra való programokat kellett lecserélni, ezt pedig a felhasználók maguk is meg tudták tenni.

A MIME öt új üzenet-fejlécmezőt definiál, amelyek a 7.12. ábrán láthatók. Az első ezek közül egyszerűen megadja az üzenetet fogadó felhasználói ügynöknek, hogy ez egy MIME-üzenet, és azt, hogy a MIME melyik verzióját használja. Minden olyan üzenet, amelyben nem szerepel a *MIME-version:* fejlécmező egyszerű angol szöveges üzenetnek számít (vagy legalábbis olyannak, ami csak ASCII-karaktereket használ), és ennek megfelelően kerül feldolgozásra.

A *Content-Description:* fejlécmező egy ASCII-karakterlánc, amely megadja az üzenet típusát. Ez a fejlécmező azért kell, hogy a címzett eldönthesse, hogy érdemes-e visszakódolni az üzenetet. Ha a mező azt tartalmazza, hogy: „Fénykép Barbara versenyegeréről", és az illető, aki kapja, nem kedveli a versenyegereket, akkor valószínűleg eldobja azt, és nem kódolja vissza nagy felbontású színes képpé.

A *Content-Id:* fejlécmező azonosítja a tartalmat. Ugyanazt a formátumot használja, mint a standard *Message-Id:* fejlécmező.

A *Content-Transfer-Encoding:* azt adja meg, hogy a szövegrészt milyen módszerrel csomagolták a hálózati átvitelre. A MIME fejlesztésének idején a legnagyobb gondot az okozta, hogy a levéltovábbító (SMTP) protokollok ASCII-üzeneteket vártak, amelyben a sorok hossza nem haladta meg az 1000 karaktert. Az ASCII-karakterek 7 bitet használnak fel minden 8 bites bájtból. Az olyan bináris adatok, mint a futtatható programok és képek, a bájtoknak mind a 8 bitjét felhasználják, akárcsak a kibővített karakterkészletek. Nem volt garancia ezeknek az adatoknak a biztonságos továbbítására. Ezért szükség volt valamilyen módszerre a bináris adatoknak az átviteléhez, amelynek segítségével a bináris adatokat át lehetett alakítani úgy, hogy azok hagyományos ASCII-levélnek tűnjenek. A MIME kifejlesztése óta az SMTP bővítményei lehetővé teszik a 8 bites bináris adatok átvitelét, de a bináris adatok még ma sem mindig mennek át hibátlanül a levelezőrendszeren, ha nincsenek átkódolva.

A MIME öt sémát bocsát rendelkezésre, plusz egy lehetőséget az új sémák használatára – biztos, ami biztos. A legegyszerűbb séma a sima ASCII-szöveg. Az ASCII-karakterek 7 biten kódolhatók, és az e-levél protokoll segítségével közvetlenül szállítani lehet azokat, feltéve, hogy egy sor nem haladja meg az 1000 karaktert.

A következő legegyszerűbb séma ugyanez, csak 8 bites karaktereket használ, azaz minden érték 0-tól 255-ig megengedett. A 8 bitet használó üzeneteknek továbbra is be kell tartaniuk a maximális sorhosszra vonatkozó korlátozást.

Vannak olyan üzenetek is, amelyek valódi bináris kódolást használnak. Ezek tetszőleges bináris fájlok, amelyek nemcsak 8 bitesek, hanem az 1000 karakteres sorhossz határt sem veszik figyelembe. A végrehajtható programok ebbe a kategóriába tartoznak. Manapság a levelezőszerverek képesek egyezkedni és megállapodni az adatok bináris (vagy 8 bites) formában történő küldésében, de visszaváltanak ASCII-módba, ha a két fél közül valamelyik nem támogatja ezt a kiterjesztést.

A bináris adatok ASCII-kódolását **base64 kódolásnak (base64 encoding)** nevezik. Ebben a sémában 24 bites csoportokat 6 bites egységekre tördelnek úgy, hogy minden egység értéke szerint egy legális ASCII-karakter formájában kerül átvitelre. A kód a következő: az „A" 0-nak, a „B" az 1-nek, a „C" a 2-nek stb. felel meg, majd a nagybetűket a 26 kisbetű követi, ezután jön a tíz szám, végül a + jel és a / jel, a 62-nek és a 63-nak megfelelően. Az == és = szekvenciák arra szolgálnak, hogy jelezzék, ha az utolsó csoport csak 16 vagy 8 bitet tartalmaz. A soremelés és kocsi-vissza jelek a kódban nem számítanak, így ezeket tetszés szerint lehet használni a rövid sorok elérése érdekében. Ezzel a sémával biztonságosan, de rossz hatékonysággal lehet tetszőleges bináris szöveget küldeni. Ez a kódolás nagyon népszerű volt a bináris átvitelre képes levelezőszerverek széles körű elterjedése előtt. Még mindig gyakran használják.

Azoknál az üzeneteknél, amelyek majdnem ASCII-formátumúak, és csak néhány nem ASCII-karaktert tartalmaznak, a base64 kódolás nem kifejezetten hatékony. Ilyenkor inkább az **idézett nyomtatható karakteres kódolás (quoted-printable encoding)** használható. Ez 7 bites ASCII, de a 127 feletti karakterek értékét egy egyenlőségjelet követő, két hexadecimális szám kódolja. A vezérlőkarakterek, néhány írásjel és matematikai szimbólum, valamint a sorvégi szóközök is ugyanígy kódoltak.

Végül, ahol jogos érvek szólnak ezek ellen a kódolások ellen, ott lehetőség van egy felhasználó által definiált kódolás megadására a *Content-Trasfer-Encoding:* fejlécmező segítségével.

A 7.12. ábrán utolsóként feltüntetett fejlécmező tulajdonképpen a legérdekesebb. Ez az üzenet tartalmának természetét határozza meg, és hatása jóval túlmutat az e-levélen. Például ha a webről letöltött tartalmakat MIME-típusokkal jelölik, akkor a böngésző tudja, hogyan kell azt megjeleníteni. Ugyanez a helyzet a valós időben letölthető multimédiás tartalmak és a valós idejű átvitel, például az IP-hálózaton keresztüli beszédátvitel esetén.

Az RFC 1521 kezdetben hét típust definiált. Minden típus egy vagy több altípussal rendelkezik. A típust és az altípust perjellel választják el, valahogy így: „Content-Type: video/mpeg". Azóta több száz altípus jelent meg, más típusokkal együtt. Valahányszor egy új típusú tartalmat fejlesztenek ki, további bejegyzések jelennek meg. Az IANA kezeli a kijelölt típusok és altípusok listáját, ami megtekinthető a *www.iana.org/assignments/media-types* oldalon.

A 7.13. ábra megadja a típusokat, a gyakran használt altípusok példáival együtt. Röviden nézzük át ezeket, kezdve a *text*-tel. A *text/plain* kombináció az általános üzeneteket jelenti, amelyeket további kódolás és alakítás nélkül lehet fogadni és megjeleníteni. Ez teszi lehetővé az általános üzenetek MIME-formában való küldését mindössze néhány extra fejlécmező hozzáadásával. Amikor a világháló népszerűvé vált, bevezették a *text/*

Típus	Altípus	Leírás
text	plain, html, xml, css	Szöveg különféle formátumban
image	gif, jpeg, tiff	Állókép különféle formátumban
audio	basic, mpeg, mp4	Hang különféle formátumban
video	mpeg, mp4, quicktime	Film különféle formátumban
model	vrml	3D modell
application	octet-stream, pdf, javascript, zip	Alkalmazások adatai különféle formátumban
message	http, rfc822	Beágyazott üzenet különféle formátumban
multipart	mixed, alternative, parallel, digest	Többféle típus kombinációja

7.13. ábra. *MIME-típusok és gyakori altípusaik*

html altípust (az RFC 2854-ben), hogy weboldalakat is lehessen RFC 822-es e-levélben küldeni. Az RFC 3023 a kiterjeszthető jelölőnyelv (XML) számára létrehozta a *text/xml* altípust. Az XML-dokumentumok a világháló fejlődésével terjedtek el. A HTML-t és az XML-t a 7.3. alfejezetben fogjuk tanulmányozni.

A következő MIME-típus az *image*, ami az állóképek átvitelére alkalmazható. Számos elterjedt, tömörítéssel ellátott vagy anélküli formátum létezik manapság a képek tárolására és átvitelére. Számos ilyen formátum kezelését, köztük a GIF, JPEG és TIFF formátumokét is, csaknem az összes böngészőbe beépítették. Sok egyéb formátum és a nekik megfelelő altípusok léteznek még.

Az *audio* és *video* a hang, illetve a mozgókép átvitelére szolgál. Kérjük, vegye figyelembe, hogy a *video* csak képi információt tartalmazhat, hangot nem. Ha hangosfilmet szeretnénk átvinni, akkor lehet, hogy külön kell átvinnünk a hangot és a képet attól függően, hogy éppen milyen kódolási rendszert használunk. Az első mozgóképformátumot a szerény elnevezésű Moving Picture Experts Group (Mozgókép Szakértői Csoport, MPEG) javasolta, de azóta más formátumok is megjelentek. Az RFC 3003-ban pedig az *audio/basic* mellett bevezettek egy új hanghordozótípust, az *audio/mpeg*-et is, hogy az emberek MP3-hangfájlokat is küldhessenek az e-levelekben. A *video/mp4* és *audio/mp4* típusok az újabb, MPEG 4 formátumban tárolt mozgóképet és hangot jelzik.

A *model* típust a többi tartalomtípust követően vezették be. Ennek célja a 3D modelladatok leírása. Mostanáig azonban széles körben nem használják.

Az *application* egy gyűjtőtípus azon formátumok számára, amelyeket a többi típus egyike sem fed le, és az adatok értelmezéséhez valamilyen alkalmazásra van szükség. Példának a *pdf, javascript* és *zip* altípusokat soroltuk fel, amik a PDF-dokumentumokat, JavaScript-programokat és Zip tömörített fájlokat jelzik. A felhasználói ügynökök, ha ilyen tartalmat kapnak, akkor egy harmadik fél által készített függvénykönyvtárat vagy külső programot használnak a tartalom megjelenítésére. A megjelenítés történhet a felhasználói ügynökbe integráltan vagy azon kívül is.

A felhasználói ügynökök a MIME-típusok segítségével képessé válnak új típusú alkalmazástartalmak kezelésére, amint kifejlesztik azokat. Ez számottevő előny. Másrészről viszont, sok új formátumú tartalmat alkalmazások hajtanak végre vagy értelmeznek, ami némi veszélyt rejt magában. Egy tetszőleges végrehajtható program futtatása, amit a „barátoktól" kaptunk a levelezőrendszerben, nyilvánvalóan biztonsági kockázatot jelent. Ez a program sokféle kellemetlen kárt tud okozni a számítógépnek azon részeiben, amihez hozzáfér, különösen, ha írhatja és olvashatja a fájlokat, és használhatja a hálózatot. Kevésbé nyilvánvaló, hogy a dokumentumformátumok is ugyanilyen veszélyt jelenthetnek. Ennek az az oka, hogy az olyan formátumok, mint például a PDF, valójában álruhába bújt fejlett programozási nyelvek. Ezek ugyan értelmezett és hatókörükben korlátozott adatok, azonban az értelmezőben lévő hibák gyakran lehetővé teszik a fondorlatos dokumentumok számára, hogy kijátsszák a korlátozásokat.

Ezeken a példákon túlmenően sok más alkalmazási altípus is létezik, mivel sokkal több alkalmazás is van. Utolsó lehetőségként, ha más, megfelelőbb altípus nem ismert, az *octet-stream* jelzi a nem értelmezett bájtsorozatot. Az ilyen folyamok fogadásakor a felhasználói ügynök valószínűleg azt javasolja a felhasználónak, hogy fájlban tárolja azt. A további feldolgozás ezután a felhasználóra vár, aki feltehetőleg tudja, hogy miféle tartalom ez valójában.

Az utolsó két típus maguknak az üzeneteknek az összeállításánál és kezelésénél hasznos. A *message* típus lehetővé teszi egy üzenet teljes beágyazását egy másikba. Ez a séma például a levelek továbbítására használható. Ha egy teljes 822-es RFC formájú levél beágyazódik egy másikba, akkor az *rfc822* altípus használandó. Ehhez hasonlóan, a HTML-dokumentumok beágyazása is gyakori. A *partial* altípus lehetővé teszi egy beágyazott üzenet több darabra tördelését és külön-külön történő átvitelét (például ha a beágyazott üzenet túl hosszú). A paraméterek lehetővé teszik a címzett állomáson a darabok sorrendhelyes összeállítását.

Végül az utolsó típus a *multipart*, amely lehetővé teszi, hogy egy üzenet több részből álljon, és minden rész eleje és vége tisztán elhatárolható legyen. A *mixed* altípus lehetővé teszi, hogy minden rész különböző típusú legyen anélkül, hogy a struktúrával szemben bármilyen további követelményt támasztana. Sok levelezőprogramban egy vagy több csatolt állományt is meg lehet adni a szöveges rész mellett. Ezeket a mellékleteket a *multipart* típus használatával küldik el.

A *multipart* ellentettje az *alternative* altípus, melynek segítségével ugyanazt az üzenetet küldhetjük el egyszerre több példányban, de különböző formátumokban. Egy üzenet például elküldhető egyszerre sima ASCII-, HTML- és PDF-formátumban. Egy jól tervezett felhasználói ügynök egy ilyen üzenetet a felhasználó igényeinek megfelelően jeleníti meg. Valószínűleg a PDF volna az első választás, ha ez lehetséges. A második választás a HTML lenne. Ha ezek közül egyik sem lehetséges, akkor marad a sima ASCII-szöveg. Először a legegyszerűbb résznek kell szerepelnie a levélben, majd ezt követhetik az egyre bonyolultabbak, hogy azok a felhasználók is értelmezhessék az üzenetet, ahol nincs MIME-képes felhasználói ügynök (például még egy ilyen régi felhasználói ügynökkel is el lehet olvasni a sima ASCII-szöveget). Az *alternative* altípus használható több nyelv esetén is. Ilyen környezetben a Rosetta kő egy korai *mutipart/alternative* üzenetnek tekinthető.[1]

1 A Rosetta kő a British Museumban látható ősi egyiptomi kőtöredék, amely írásos szöveget tartalmaz három nyelven. *(A fordító megjegyzése)*

A másik két, példaként említett altípus közül az elsőt, a *parallel* altípust akkor használják, ha minden részt egyszerre kell „megjeleníteni". Például a filmek gyakran két részből, a mozgóképből és a hangból állnak. A filmek sokkal hatásosabbak, ha ezeket a részeiket egyszerre, nem pedig egymás után játsszák le. A *digest* altípust akkor használják, ha több üzenetet egyetlen összetett üzenetbe csomagolnak össze. Például az interneten néhány vitafórum összegyűjti az előfizetők leveleit, és bizonyos időszakonként szétküldi azokat a csoportnak egyetlen *multipart/digest* üzenetként.

A 7.14. ábrán a MIME-típusoknak e-levél üzenetekben történő alkalmazására, egy multimédiás üzenetre láthatunk példát. Ebben egy születésnapi köszöntés kerül átvitelre két alternatív formában: HTML és hang formájában. Ha a fogadó képes hangok lejátszására, a felhasználói ügynök le fogja játszani a hangot. Ebben a példában csak egy hivatkozás található a hangra, *message/external-body* altípusként, ezért a felhasználói ügynöknek először el kell hoznia a hálózatról a *birthday.snd* hangfájlt FTP segítségével. Ha a felhasználói ügynök nem képes hangok lejátszására, akkor csak a dalszöveg látszik síri csendben. A részeket két kötőjel, és az azt követő a *boundary* mezőben megadott (szoftveresen előállított) karakterlánc különíti el.

Vegyük észre, hogy a *Content-Type* fejlécmező háromszor fordul elő ebben a példában. Legfelül arra szolgál, hogy jelezze, az üzenet több részből áll, a részekben pedig azok típusát és altípusát adja meg. Végül a második rész szövegrészében meg kell adni a fel-

```
From: alice@cs.washington.edu
To: bob@ee.uwa.edu.au
MIME-Version: 1.0
Message-Id:<0704760941.AA00747@cs.washington.edu>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: A Föld egész számú alkalommal kerüli meg a Napot

Ez a bevezető. A felhasználói ügynök figyelmen kívül hagyja. Minden jót.

--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/html

<p>Boldog szülinapot<br>
Boldog szülinapot<br>
Boldog szülinapot kedves <b> Bob </b><br>
Boldog szülinapot</p>

--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
   access-type="anon-ftp";
   site="bicycle.cs.washington.edu";
   directory="pub";
   name="birthday.snd"

content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--
```

7.14. ábra. *Egy HTML és audio alternatívákat tartalmazó többrészes üzenet*

használói ügynöknek az elhozandó fájl nevét. Hogy érzékeltessük ezt a kis különbséget, itt kisbetűket használtunk a fejlécmezőkben, egyébként minden fejlécmező érzéketlen a kis- és nagybetűk használatára. A *Content-Transfer-Encoding* fejlécmező ugyanúgy szükséges bármilyen külső szövegtörzs esetén, amely nem 7 bites ASCII-kódolású.

7.2.4. Üzenettovábbítás

Miután bemutattuk a felhasználói ügynököket és a levélüzeneteket, készen állunk rá, hogy megnézzük, hogyan továbbítják a levéltovábbító ügynökök a leveleket a feladótól a címzetthez. A levelek továbbítását az SMTP-protokoll végzi.

Az üzenetek továbbításának legegyszerűbb módja az, hogy a forrásgép szállítási összeköttetést létesít a célgéppel, majd ezen átviszi az üzenetet. Eredetileg így működött az SMTP. Az évek során azonban az SMTP-nek két eltérő felhasználási módját különböztették meg. Az első a **levél feladása (mail submission)**, ami az e-levél architektúrát bemutató 7.7. ábrán az 1. lépés. A felhasználói ügynökök ezzel küldik be az üzeneteket a levelezőrendszerbe kézbesítés céljából. A második felhasználási mód az üzenetek továbbítása az üzenettovábbító ügynökök között (a 7.7. ábrán a 2. lépés). Ez a sorozat egyetlen ugrással eljuttatja a levelet a küldőtől a fogadó üzenettovábbító ügynökig. A végső kézbesítést különféle protokollok végzik, amelyeket a következő alfejezetben ismertetünk.

Ebben az alfejezetben bemutatjuk az SMTP-protokoll alapjait és kiterjesztési mechanizmusát. Ezután megtárgyaljuk, miben különbözik a levél feladására és az üzenettovábbításra történő használata.

SMTP (egyszerű levéltovábbító protokoll) és kiterjesztései

Az interneten belül, az e-levelek kézbesítése úgy történik, hogy a forrásgép TCP-összeköttetést teremt a célgép 25-ös portjával. Ezt a portot egy levelezőszerver figyeli, amelyik az **SMTP (Simple Mail Transfer Protocol – egyszerű levéltovábbító protokoll)** „nyelvet beszéli". Ez a szerver fogadja a beérkező összeköttetés-kéréseket, aláveti azokat néhány biztonsági ellenőrzésnek, és átveszi az üzeneteket kézbesítésre. Ha egy üzenetet nem lehet kézbesíteni, akkor a kézbesíthetetlen üzenet első részét tartalmazó hibajelentéssel a feladó visszakapja azt.

Az SMTP egy egyszerű ASCII-protokoll. Ez nem hátrány, csak egy tulajdonság. Az ASCII-szövegek használata megkönnyíti a protokollok fejlesztését, tesztelését és nyomkövetését. A protokollok tesztelhetők a parancsok kézi elküldésével, és az üzenetek bejegyzései könnyen olvashatók. A legtöbb alkalmazási rétegbeli internetprotokoll (például HTTP) ma így működik.

Körüljárunk egy levelezőszerverek közötti üzenetátvitelt, amelynek során egy üzenet kézbesítése történik. A 25-ös porttal való TCP-összeköttetés létesítése után a kliens küldőgép megvárja, amíg a szerver fogadógép meg nem szólal. A szerver azzal kezdi, hogy küld egy sornyi szöveget, amelyben azonosítja magát, és megadja, hogy fel van-e készülve levelek fogadására. Ha nem, a kliens bontja az összeköttetést, és később újra próbálkozik.

Ha a szerver felkészült a levelek fogadására, akkor a kliens megadja, hogy kitől és kinek megy az e-levél. Ha a megadott címzett létezik a célgépen, akkor a szerver szabad utat ad a kliens számára az üzenetküldéshez. Ezután a kliens elküldi a levelet, és a szerver nyugtázza azt. Semmilyen ellenőrző összegre nincs szükség, mivel a TCP-összeköttetés megbízható bájtfolyamot biztosít. Ha van még e-levél, akkor azt is elküldi. Miután mindkét irányban megtörtént az összes e-levélcsere, az összeköttetés lebomlik. A 7.15. ábrán látható egy, az SMTP által használt számkódokat is feltüntető mintapárbeszéd a 7.14. ábrán levő levél elküldéséhez. A kliens (azaz küldő) által küldött sorokat *C:*, a szerver (azaz fogadó) által küldött sorokat pedig *S:* jelöli.

```
                S: 220 ee.uwa.edu.au SMTP service ready
C: HELO cs.washington.edu
                S: 250 cs.washington.edu says hello to ee.uwa.edu.au
C: MAIL FROM: <alice@cs.washington.edu>
                S: 250 sender ok
C: RCPT TO: <bob@ee.uwa.edu.au>
                S: 250 recipient ok
C: DATA
                S: 354 Send mail; end with "." on a line by itself
C: From: alice@cs.washington.edu
C: To: bob@ee.uwa.edu.au
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@ee.uwa.edu.au>
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: A Föld egész számú alkalommal kerüli meg a Napot
C:
C: Ez a bevezető. A felhasználói ügynök figyelmen kívül hagyja. Minden jót.
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: text/html
C:
C: <p> Boldog szülinapot<br>
C: Boldog szülinapot<br>
C: Boldog szülinapot, kedves <b> Bob </b><br>
C: Boldog szülinapot</p>
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
C:      access-type="anon-ftp";
C:      site="bicycle.cs.washington.edu";
C:      directory="pub";
C:      name="birthday.snd"
C:
C: content-type: audio/basic
C: content-transfer-encoding: base64
C: --qwertyuiopasdfghjklzxcvbnm
C: .
                S: 250 message accepted
C: QUIT
                S: 221 ee.uwa.edu.au closing connection
```

7.15. ábra. *Egy üzenet küldése az alice@cs.washington.edu-ról a bob@ee.uwa.edu.au-ra*

A kliens részéről az első parancs valójában a *HELO*. A *HELLO* különféle négybetűs rövidítései közül ez számos előnnyel rendelkezik legnagyobb versenytársához képest. Azt, hogy miért kellett minden parancsnak négybetűsnek lennie, már senki sem tudja.

A 7.15. ábrán levő üzenetnek csak egy címzettje van, tehát csak egy *RCPT* parancs látható. Több ilyen parancsot is ki lehet adni, ha ugyanazon levélnek több címzettje is van. Mindegyikre egyedileg jön nyugta vagy visszautasítás. Még ha néhány címzettől visszautasítás jön is (mert nem léteznek a célgépen), a többieknek akkor is el lehet küldeni az üzenetet.

Végül, a négybetűs parancsok szintaxisa ugyan szigorúan definiált, a válaszoké azonban kevésbé az. Egyedül a számkód számít igazán. Megvalósítástól függően bármilyen karaktersorozat állhat a kód után.

Az alap SMTP jól működik, de több szempontból korlátozott a működése. Nem rendelkezik hitelesítéssel. Ez azt jelenti, hogy a példabeli *FROM* parancs olyan feladót ad meg, amilyet csak akar. Ez meglehetősen hasznos kacatlevél küldéséhez. Egy másik korlátozás, hogy az SMTP ASCII-üzeneteket továbbít, nem bináris adatokat. Ezért volt szükség a base64 MIME-kódolásra a tartalom továbbításához. Ezzel a kódolással azonban a levéltovábbítás nem hatékonyan használja ki a sávszélességet, ami a hosszú üzeneteknél problémát jelent. A harmadik korlátozás az, hogy az SMTP nyílt szövegként küldi az üzeneteket. Nincs titkosítás, ami egy kissé elrejtené a személyes tartalmakat a kívácsi szemek elől.

Ezeknek, és számos más, az üzenetkezeléshez kapcsolódó problémának a megoldása érdekében az SMTP-t felülvizsgálták és egy kiterjesztési mechanizmussal látták el. Ez a mechanizmus kötelező része az RFC 5321 szabványnak. A kiterjesztett SMTP-t ESMTP-nek (**Extended SMTP**) nevezik.

Azok a kliensek, amelyek egy kiterjesztést szeretnének használni, kezdetben *EHLO* üzenetet küldenek *HELO* helyett. Ha ezt a szerver elutasítja, akkor a szerver egy általános SMTP-szerver, és a kliensnek a szokásos módon kell eljárnia. Ha elfogadja az *EHLO*-t, a szerver az általa támogatott kiterjesztésekkel válaszol. A kliens ezen kiterjesztések közül bármelyiket használhatja. Néhány gyakori kiterjesztést mutat a 7.16. ábra. Az ábra megadja a kiterjesztési mechanizmus által használt kulcsszót az új funkciók ismertetésével együtt. A kiterjesztéseket bővebben nem részletezzük.

Jobban ráérezhetünk arra, hogy hogyan működik az SMTP és más, a fejezetben ismertetett protokoll, ha ki is próbáljuk azokat. Mint minden esetben, először most is ke-

Kulcsszó	Funkció ismertetése
AUTH	Kliens hitelesítése
BINARYMIME	A szerver bináris üzeneteket is elfogad
CHUNKING	A szerver nagyobb darabokból álló, hosszú üzeneteket is elfogad
SIZE	Az üzenet méretének ellenőrzése küldés előtt
STARTTLS	Átkapcsolás biztonságos továbbításra (a TLS-t lásd a 8. fejezetben)
UTF8SMTP	Nemzetközi címek kezelése

7.16. ábra. *Néhány SMTP-kiterjesztés*

rítenünk kell egy egy gépet internetkapcsolattal. UNIX- (vagy Linux-) rendszer esetén írjuk be a parancssorba, hogy

```
telnet mail.isp.com 25
```

és a *mail.isp.com* helyébe a saját szolgáltatónk levelezőszerverének DNS-nevét helyettesítsük! Windows XP rendszeren kattintsunk a Start, majd a Futtatás menüpontra, és írjuk be a fenti parancsot a párbeszédablakba. Vistás vagy Windows 7-es gépeken szükséges lehet először a telnet (vagy azzal egyenértékű) programot telepíteni, majd elindítani. Ez a parancs kiépít egy telnet- (azaz TCP-) összeköttetést az adott gép 25-ös portjára. A 25-ös port az SMTP-hez tartozik (a szokványos portokat lásd a 6.34. ábrán). Bizonyára valami ehhez hasonló választ kapunk:

```
Trying 192.30.200.66...
Connected to mail.isp.com
Escape character is '^]'.
220 mail.isp.com Smail #74 ready at Thu, 25 Sept 2002 13:26 +0200
```

Az első három sort a telnet írja ki, hogy elmondja, éppen mit csinál. Az utolsó sor a távoli gép SMTP-kiszolgálójától származik: ebben jelenti be, hogy hajlandó beszélni velünk és fogadni az e-leveleket. Azt is megtudhatjuk, hogy milyen parancsokat fogad el, ha beírjuk, hogy

```
HELP
```

Ettől kezdve lehetséges egy olyan parancssorozat, mint amilyet a 7.16. ábrán láthattunk, ha a szerver hajlandó leveleket fogadni.

Levélfeladás

Eredetileg a felhasználói ügynökök ugyanazon a számítógépen futottak, mint amelyen az üzenetet elküldő üzenettovábbító ügynök. Ebben az elrendezésben a felhasználói ügynöknek az üzenet elküldéséhez mindössze arra van szüksége, hogy felvegye a kapcsolatot a helyi levelezőszerverrel egy olyan párbeszéd útján, mint amilyet az előbb ismertettünk. Ez az elrendezés azonban már nem a szokásos elrendezés.

A felhasználói ügynökök gyakran laptopokon, otthoni PC-ken és mobiltelefonokon futnak. Ezek nem mindig csatlakoznak az internetre. A levéltovábbító ügynökök az internetszolgáltatók és vállalatok szerverein futnak. Ezek mindig csatlakoznak az internetre. Ez a különbség azt jelenti, hogy egy bostoni felhasználói ügynöknek lehet, hogy fel kell vennie a kapcsolatot a seattle-i szabályos levelezőszerverével, ha el akar küldeni egy levelet, mert a felhasználó éppen utazik.

Önmagában ez a távoli kommunikáció nem jelent problémát. Pontosan erre a célra tervezték a TCP/IP-protokollokat. Egy internetszolgáltató vagy vállalat azonban általában nem szeretné bármely távoli felhasználó számára lehetővé tenni, hogy levelezőszer-

verének segítségével üzenetet küldjön máshová. Az internetszolgáltató vagy vállalat a szervert nem nyilvános szolgáltatásként működteti. Ráadásul, az ilyen **nyílt levéltovábbítás (open mail relay)** vonzza a kacatlevelet küldőket. Ennek az az oka, hogy lehetőséget biztosít az eredeti feladónak arra, hogy kezei tiszták maradjanak, és így az üzenetet még nehezebb legyen kacatlevélként azonosítani.

Ezen megfontolások alapján az SMTP-t normális körülmények között csak az AUTH kiterjesztéssel együtt használják levélfeladásra. Ez a kiterjesztés megvizsgáltatja a szerverrel a kliens azonosítóit (felhasználónevét és jelszavát) annak eldöntése érdekében, hogy a szervernek kell-e levelezési szolgáltatást nyújtania.

Számos más eltérés is van az SMTP levélfeladásra való használatának módjában. Például az 587-es portot részesítik előnyben a 25-össel szemben, és az SMTP-szerver ellenőrizheti és kijavíthatja a felhasználói ügynök által küldött üzenetek formátumát. Az SMTP korlátozott levélfeladásra való használatáról szóló további információért kérjük, tekintse meg az RFC 4409-et.

Üzenetátvitel

Miután a küldő levéltovábbító ügynök megkapta a levelet a felhasználói ügynöktől, kézbesíti azt a fogadó levéltovábbító ügynöknek az SMTP segítségével. Ennek érdekében a küldő a rendeltetési címet használja. Figyeljük meg a 7.15. ábrán látható, *bob@ee.uwa.edu.au* címre küldendő üzenetet! Melyik levelezőszervernek kell továbbítani az üzenetet?

A megfelelő levelezőszerverrel való kapcsolatfelvétel érdekében meg kell kérdezni a DNS-t. Az előző fejezetben leírtuk, hogyan tárol a DNS különböző típusú bejegyzéseket, beleértve az *MX* vagy levélcserélő bejegyzést. Ebben az esetben egy DNS-lekérdezés indul az *ee.uwa.edu.au* körzet *MX* bejegyzéseiért. A válasz egy rendezett lista lesz, amely egy vagy több levelezőszerver nevét és IP-címét tartalmazza.

A küldő levéltovábbító ügynök ezután létrehoz egy TCP-összeköttetést a 25-ös porton lévő levelezőszerver IP-címére, hogy elérje a fogadó levéltovábbító ügynököt, és az SMTP-t használva továbbítja az üzenetet. Ezt követően a fogadó levéltovábbító ügynök a *bob* felhasználónak küldött levelet Bob postaládájába helyezi, hogy Bob később elolvashassa azt. Ez a helyi kézbesítési lépés nagy levelezési infrastruktúra esetén esetleg a levélnek a számítógépek közti mozgatásával is járhat.

Ennek a kézbesítési folyamatnak a során a levél a kezdeti levéltovábbító ügynöktől a végső levéltovábbító ügynökig terjedő utat egyetlen ugrással teszi meg. Nincsenek közbenső szerverek az üzenettovábbítási szakaszban. Az azonban lehetséges, hogy ez a kézbesítési folyamat többször is megtörténik. Egy példát már ismertettünk korábban, amikor egy levéltovábbító ügynök egy levelezőlistát üzemeltet. Ebben az esetben az üzenet a lista számára érkezik. Ezután a levelet kiterjesztik a lista minden egyes tagjának szóló üzenetté, azaz elküldik minden egyes tag egyéni címére.

Egy másik példa az átjátszásra a következő. Bob végzett az M.I.T.-n, és a *bob@alum.mit.edu* címen is elérhető. Ahelyett, hogy mindkét felhasználói fiók leveleit elolvasná, Bob intézkedhet arról, hogy az erre a címre küldött leveleket továbbítsák a *bob@ee.uwa.edu*-ra. Ebben az esetben a *bob@alum.mit.edu* címre küldött levél két kézbesítésen esik

át. A levelet először az *alum.mit.edu* levelezőszerverének küldik, azután továbbküldik az *ee.uwa.edu.au* levelezőszerverének. Mindkét szakasz teljes és egymástól független kézbesítés, abban a levéltovábbító ügynökök érintettek.

Egy másik szempont manapság a kacatlevél. Ma tízből kilenc elküldött üzenet kacatlevél [McAfee, 2010]. Kevesen szeretnének még több kacatlevelet, de nehéz elkerülni őket, mert közönséges levélnek álcázzák magukat. Az üzenet elfogadása előtt további ellenőrzések végezhetők a kacatlevél lehetőségeinek csökkentésére. A Bobnak szánt üzenet az *alice@cs.washington.edu*-ról lett elküldve. A fogadó levéltovábbító ügynök megkeresheti a küldő levéltovábbító ügynököt a DNS-ben. Ezzel lehetővé válik annak ellenőrzése, hogy a TCP-összeköttetés másik végének IP-címe összeillik-e a DNS-névvel. Még általánosabban, a fogadó ügynök megkeresheti a küldő tartományt a DNS-ben, hogy lássa, van-e annak levélküldési házirendje. Ezt az információt gyakran megadják a *TXT* és *SPF* bejegyzésekben. Ez azt jelezheti, hogy további ellenőrzések végezhetők. Például lehet, hogy a *cs.washington.edu*-ról küldött leveleket mindig a *june.cs.washington.edu* hosztról küldik. Ha a küldő levéltovábbító ügynök nem *june*, akkor baj van.

Ha ezeknek az ellenőrzéseknek bármelyike kudarccal végződik, akkor valószínűleg meghamisították a levelet egy hamis feladási címmel. Ebben az esetben a levelet eldobják. Az ellenőrzéseken való átjutás azonban még nem jelenti azt, hogy a levél nem kacatlevél. Az ellenőrzések csupán azt biztosítják, hogy a levél valószínűleg a hálózatnak abból a tartományából érkezett, mint amit magáról állít. Az az elképzelés, hogy a kacatlevelet küldőket rákényszerítsék a helyes feladási címek használatára, amikor levelet küldenek. Ez a kacatlevelet könnyebben felismerhetővé és letörölhetővé teszi, ha nem kívánatos.

7.2.5. Végső kézbesítés

Levélüzenetünket már majdnem sikerült kézbesíteni, hiszen már megérkezett Bob postaládájába. Mindössze annyi van hátra, hogy a megjelenítés érdekében az üzenet egy másolati példányát tobábbítani kell Bob felhasználói ügynökéhez. Ez a 7.7. ábrán látható architektúra 3. lépése. Ez a feladat magától értetődő volt az internet korai időszakában, amikor a felhasználói ügynök és a levéltovábbító ügynök ugyanazon a gépen, külön folyamatként futott. A levéltovábbító ügynök egyszerűen hozzáfűzte az új üzeneteket a postaládafájl végéhez, a felhasználói ügynök pedig csak ellenőrizte a postaládafájlt, hogy érkezett-e új levél.

Manapság a felhasználói ügynök egy PC-n, laptopon vagy mobiltelefonon fut, tehát valószínűleg nem az internetszolgáltató vagy vállalat levelezőszerverén, hanem egy másik gépen. A felhasználók távolról is el szeretnék érni leveleiket, bárhol is vannak. Hozzá szeretnének férni leveleikhez a munkahelyükről, az otthoni PC-ikről, a laptopjukról, amikor üzleti úton vannak, és az internetes kávézókból, amikor a szabadságukat töltik. Szeretnének hálózat nélkül dolgozni, majd újra a hálózatra csatlakozni, hogy fogadhassák beérkezett leveleiket és elküldhessék kimenő leveleiket. Sőt minden egyes felhasználó több felhasználói ügynököt futtathat attól függően, hogy éppen melyik számítógépet kényelmes használni. Éppenséggel még több felhasználói ügynök is futhat egyszerre.

Ebben a helyzetben a felhasználói ügynöknek az a feladata, hogy megjelenítse a postaláda tartalmát, és lehetővé tegye a postaláda távolról történő kezelését. Számos különböző protokoll használható erre a célra, de az SMTP nem tartozik ezek közé. Az

SMTP küldésalapú (push-based) protokoll. Fog egy üzenetet, majd csatlakozik a távoli szerverhez és továbbítja azt. A végső kézbesítést nem lehet ilyen módon megvalósítani, mert egyrészt a postaládát továbbra is a levéltovábbító ügynöknek kell tárolnia, másrészt a felhasználói ügynök lehet, hogy éppen nem csatlakozik az internetre, amikor az SMTP megpróbálja az üzeneteket neki továbbítani.

Parancs	Funkció ismertetése
CAPABILITY	Szerver képességeinek listázása
STARTTLS	Biztonságos továbbítás elindítása (TLS; lásd a 8. fejezetben)
LOGIN	Bejelentkezés a szerverre
AUTHENTICATE	Bejelentkezés más módon
SELECT	Mappa kiválasztása
EXAMINE	Csak olvasható mappa kiválasztása
CREATE	Mappa létrehozása
DELETE	Mappa törlése
RENAME	Mappa átnevezése
SUBSCRIBE	Mappa hozzáadása az aktív készlethez
UNSUBSCRIBE	Mappa eltávolítása az aktív készletből
LIST	Rendelkezésre álló mappák listázása
LSUB	Aktív mappák listázása
STATUS	Mappa állapotának lekérdezése
APPEND	Üzenet mappához adása
CHECK	Mappa ellenőrzési pontjának lekérdezése
FETCH	Üzenetek lekérése a mappából
SEARCH	Üzenetek keresése egy mappában
STORE	Az üzenet állapotjelzőinek megváltoztatása
COPY	Egy mappában lévő üzenet lemásolása
EXPUNGE	A törlésre megjelölt üzenetek eltávolítása
UID	Egyedi azonosítót használó parancsok kiadása
NOOP	Ne csináljon semmit
CLOSE	Megjelölt üzenetek eltávolítása és a mappa bezárása
LOGOUT	Kijelentkezés és a kapcsolat lezárása

7.17. ábra. *IMAP-parancsok (4. verzió)*

IMAP – Az internetes levél-hozzáférési protokoll

A végső kézbesítésre használt egyik legfontosabb protokoll az **IMAP** (**Internet Message Access Protocol – internetes levél-hozzáférési protokoll**). A protokoll 4. változatát az RFC 3501 határozza meg. Az IMAP használatához a levelezőszerver egy IMAP-szervert futtat, ami a 143-as portot figyeli. A felhasználói ügynök IMAP-kliensként fut. A kliens kapcsolódik a szerverhez, és elkezdi kiadni a 7.17. ábrán látható parancsokat.

Először is, ha szükséges, a kliens elindít egy biztonságos átvitelt (annak érdekében, hogy az üzeneteket és parancsokat titokban tartsa), és bejelentkezik vagy más módon igazolja hitelességét a szerveren. Miután bejelentkezett, számos parancsot használhat a mappák és parancsok listázásához, üzenetek vagy akár azok részeinek lekéréséhez, az üzenetek későbbi törlésre való megjelölésére és az üzenetek mappákba rendezéséhez. A keveredések elkerülése végett kérjük, jegyezze meg, hogy a „mappa" szót mi itt azért használjuk, hogy a fejezet további tartalma egységes legyen, melyben a felhasználónak egyetlen postaládája van, ami több mappából áll. Az IMAP részletes leírásában azonban e helyett a *postaláda* kifejezést használják. Tehát egy felhasználónak több IMAP-postaládája van, mindegyik tipikusan mappaként mutatkozik a felhasználónak.

Az IMAP számos más képességgel is rendelkezik. A leveleket például nem csak beérkezésük sorrendjében képes kezelni, hanem attribútumaik alapján is (például „Kérem az első üzenetet Alice-tól"). Kereséseket lehet indítani a szerveren, hogy megtaláljunk bizonyos kritériumoknak megfelelő üzeneteket úgy, hogy a kliensnek csak ezeket az üzeneteket kelljen letöltenie.

Az IMAP a korábbi kézbesítési protokollnak, a **POP3**-nak (**Post Office Protocol, version 3 – postahivatal protokoll, 3. verzió**) a tökéletesítése, amit az RFC 1939 határoz meg. A POP3 egyszerűbb protokoll, de kevesebb képességgel rendelkezik és a tipikus felhasználás során kevésbé biztonságos. A leveleket általában letöltik a felhasználói ügynököt futtató számítógépre ahelyett, hogy azok a levelezőszerveren maradnának. Ez könnyebbé teszi a szerver életét, de nehezebbé teszi a felhasználóét. Nem könnyű a leveleket több számítógépen olvasni, és ha a felhasználói ügynököt futtató számítógép tönkremegy, minden e-levél végleg elveszhet. Ennek ellenére a POP3 még mindig használatban van.

Egyedi protokollok is alkalmazhatók, mert a protokoll a levelezőszerver és a felhasználói ügynök között használatos, amelyeket ugyanaz a cég szállíthat. A Microsoft Exchange egy egyedi protokollal rendelkező levelezőrendszer.

Webes levelezés

Az IMAP-vel és az SMTP-vel szemben egyre népszerűbbek azok a levelezési szolgáltatások, amelyek a világhálót használják felületként a levelek küldésére és fogadására. A széles körben használt **webes levelezőrendszerek** közé tartozik a Google Gmail, a Microsoft Hotmail és Yahoo! Mail. A webes levelezőrendszer egyik példája azoknak a szoftvereknek (ebben az esetben ilyen a felhasználói ügynök), amelyek a világhálót használók számára szolgáltatást nyújtanak.

Ebben az architektúrában a szolgáltató, a szokásos módon, SMTP-vel levelezőszervereket futtat a 25-ös porton a felhasználók leveleinek fogadására. A felhasználói ügynök

azonban itt eltérő. Ahelyett, hogy egy önálló program lenne, ez egy weblapok által nyújtott felhasználói felület. Ez azt jelenti, hogy a felhasználók bármilyen nekik tetsző böngészőt használhatnak arra, hogy hozzáférjenek leveleikhez és új üzeneteket küldjenek.

Még nem tanulmányoztuk a világhálót, de egy rövid ismertető, amelyhez később visszatérhetünk, a következő. Amikor a felhasználó meglátogatja az e-levél szolgáltató weboldalát, egy űrlappal találkozik, ahol megadhatja az azonosítóját és a jelszavát. Az azonosító és a jelszó eljut a kiszolgálóhoz, amely ellenőrzi azokat. Ha a belépés sikeres, a kiszolgáló megkeresi a felhasználó postaládáját, és készít egy weboldalt, amelyen kilistázza a postaláda aktuális tartalmát. Ezt a weboldalt aztán elküldi megjelenítésre a böngészőnek.

A postaláda tartalmát mutató weboldal több elemére is rá lehet kattintani, így az üzenetek olvashatók, törölhetők és így tovább. Annak érdekében, hogy a felületen keresztül reagálni lehessen, a weboldalak gyakran JavaScript-programokat tartalmaznak. Ezek a programok a kliensen futnak helyi eseményekre (például egérkattintás) adott válaszként, ugyanakkor képesek a háttérben üzeneteket fel- és letölteni annak érdekében, hogy előkészítsék a következő üzenetet a megjelenítéshez vagy egy új üzenetet az elküldéshez. Ebben a modellben a levélküldés a normál webes protokollok segítségével, adatoknak egy adott URL-re való elküldésével történik. A webszerver gondoskodik az üzenetnek a korábban bemutatott hagyományos levélkézbesítő rendszerbe juttatásáról. A biztonság érdekében a szabványos internetprotokollok is használhatók. Ezek a protokollok maguknak a weboldalaknak a titkosításával foglalkoznak, függetlenül attól, hogy a weboldal tartalma egy levélüzenet.

7.3. A világháló

A világháló (world wide web, www), vagy népszerűbb nevén a web, egy keretrendszer, amely az internethez kapcsolódó több millió számítógépen elszórva elhelyezkedő, egymással összefüggő dokumentumok elérését teszi lehetővé. Az elmúlt 10 év alatt a svájci nagy energiájú fizikai kísérletek megtervezésének összehangolására szolgáló módszerből olyan alkalmazássá vált, amelyre az emberek mint „az internet"-re gondolnak. Hihetetlen népszerűsége onnan ered, hogy kezdők által is könnyen használható, és értékes grafikus felületének segítségével hatalmas információmennyiség elérését biztosítja majdnem minden elképzelhető témáról – az abakusztól a zulukig.

A web története 1989-ben kezdődött a svájci CERN-ben, a nukleáris kutatás európai központjában. Az volt az eredeti elképzelés, hogy a részecskefizikai kísérletek során előálló jelentések, tervek, rajzok, fényképek és más dokumentumok folyamatosan változó gyűjteményének használatával segítsék a nagy, gyakran fél tucat vagy több országban és időzónában élő tagokból álló munkacsoportok együttműködését. Az összekapcsolt dokumentumok hálózatának javaslata egy CERN-beli fizikustól, Tim Berners-Lee-től eredt. Az első (szövegalapú) prototípus 18 hónappal később kezdte meg működését. A Hypertext '91 konferencián tartott nyilvános bemutató más kutatók figyelmét is felkeltette, ennek hatására fogott bele Marc Andreessen az Illinois Egyetemen az első grafikus böngésző kifejlesztésébe, amit Mosaic-nak neveztek, és 1993-ban adtak ki.

A többi, ahogyan mondani szokás, ma már történelem. A Mosaic annyira népszerű volt, hogy egy évvel később Andreessen otthagyta az egyetemet, hogy megalapítsa a Netscape Communications Corp. társaságot, melynek célja a web szoftverének kifejlesztése volt. Az ezt követő három évben a Netscape Navigator és a Microsoft Internet Explorer „böngészőháborúba" keveredtek, mindkettő megpróbált az új piacon minél nagyobb részesedésre szert tenni, miközben fejvesztve próbáltak a másiknál több funkciót (és így még több hibát) a programjukba zsúfolni.

Az 1990-es és 2000-es évek során a webhelyeknek (web site) és weboldalnak (web page) nevezett webes tartalmak exponenciális ütemben növekedtek, amíg sok millió webhely és több milliárd weboldal létre nem jött. Néhányan ezek közül a webhelyek közül óriási népszerűségre tettek szert. Ezek a webhelyek és a mögöttük álló vállalatok alakították ki a web arculatát olyanra, amilyennek ma az emberek ismerik. Közéjük tartozik például egy könyváruház (Amazon, 1994-ben indult, piaci értéke 50 milliárd $), egy használtcikk-piac (eBay, 1995, 30 milliárd $), egy internetes keresőszolgáltatás (Google, 1998, 150 milliárd $), egy közösségi hálózat (Facebook, 2004, zártkörű társaság, melynek értéke több mint 15 milliárd $). A 2000 körüli időszaknak, amikor sok webes társaság egyik napról a másikra több százmillió dolláros értékűvé vált, és gyakorlatilag másnap csődbe ment, amikor kiderült róluk, hogy csak reklámfogások voltak, még neve is van. Ezt hívják dotcom korszaknak. A jó ötleteknek még mindig nagy sikere van a weben. Ezek közül több is egyetemi hallgatóktól származik. Például Mark Zuckerberg a Harvardon tanult, amikor elindította a Facebookot, Sergey Brin és Larry Page pedig stanfordi diákok voltak, amikor útjára bocsátották a Google-t. Talán éppen Ön fog előállni a következő nagy ötlettel.

1994-ben a CERN és az M.I.T. aláírtak egy megegyezést a **World Wide Web Konzorcium (World Wide Web Consortium, W3C)** felállításáról. A szervezet célja a világháló (web) továbbfejlesztése, a protokollok szabványosítása és a webhelyek (websites) közti együttműködés elősegítése. Berners-Lee lett az igazgató. Azóta több száz egyetem és társaság csatlakozott a konzorciumhoz. Bár több könyv szól a webről, mint égen a csillag, naprakész információt (természetesen) leginkább magán a weben lehet találni. A konzorcium honlapja a *www.w3.org* címen található. Az érdeklődő olvasók itt a konzorcium összes tevékenységét és dokumentumát lefedő oldalakra találnak hivatkozásokat.

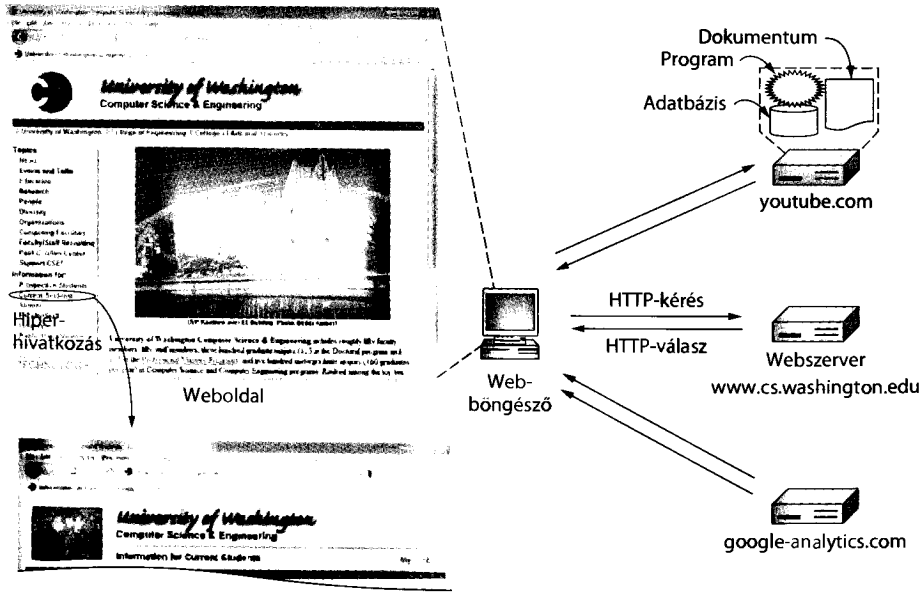
7.3.1. A web felépítésének áttekintése

A felhasználók szemszögéből a világháló (web) különféle dokumentumok vagy **weboldalak**, vagy röviden csak **oldalak (pages)** hatalmas, világméretű gyűjteményéből áll. Minden oldal tartalmazhat hivatkozásokat (links) más oldalakra, melyek a világon bárhol lehetnek. A felhasználók egy kattintással követhetik a hivatkozásokat, amelyek a hivatkozott oldalra viszik őket. Ez aztán a végtelenségig ismétlődhet. A **hipertext (hypertext)**, vagyis az egymásra mutató oldalak ötletét az M.I.T. egyik látnoki képességű villamosmérnök professzora, Vannevar Bush vetette fel [Bush, 1945]. Ez még jóval az internet feltalálása előtt történt. Valójában ez még a kereskedelmi forgalomban kapható számítógépek megjelenése előtt történt, amikor számos egyetemen készítettek olyan kiforratlan számítógép-prototípusokat, amelyek nagy géptermeket foglaltak el, ugyanakkor a teljesítményük kisebb volt, mint amilyen ma egy modern zsebszámológépnek van.

Az oldalakat általában egy **böngészőnek** (**browser**) nevezett programmal tekinthetjük meg. A Firefox, az Internet Explorer és a Chrome például népszerű böngészők. A böngésző elhozza a kívánt oldalt, értelmezi a tartalmat, majd megfelelően formázva megjeleníti az oldalt a képernyőn. Maga a tartalom szöveg, képek és formázásra vonatkozó parancsok keveréke, és mint ilyen, egy hagyományos dokumentum lehet, de másfajta tartalmakat – úgymint videókat, vagy olyan programokat, amelyeknek grafikus és interaktív felülete van – is tartalmazhat.

Egy oldal képe látható a 7.18. ábra bal felső sarkában. Ez a University of Washington Informatikai és Villamosmérnöki tanszékének oldala. Ez az oldal szöveges és grafikus elemeket tartalmaz (melyek többnyire túl kicsik ahhoz, hogy el lehessen őket olvasni). Az oldal bizonyos részei hivatkozások segítségével más oldalakhoz kapcsolódnak. A másik oldallal összekapcsolt szövegrészt, ikont, képet stb. **hiperhivatkozásnak** (**hyperlink**) nevezzük. Ha a felhasználó követni szeretné a hivatkozást, akkor az egérkurzort az oldalnak a hivatkozást tartalmazó része fölé helyezi (aminek hatására a kurzor megváltoztatja az alakját), és kattint. Egy hivatkozás követése csak egy mód arra, hogy a böngészőnek megmondjuk, hogy hozzon le egy másik oldalt. A web hajnalán a hivatkozásokat aláhúzással és színes szöveggel emelték ki, hogy feltűnőek legyenek. Manapság a weboldalak létrehozóinak több módszerük is van a hivatkozással ellátott területek megjelenésének vezérlésére, így egy hivatkozás megjelenhet ikonként, vagy megváltoztathatja megjelenési formáját, amikor az egér átsiklik fölötte. Az oldal létrehozóin múlik, hogy a hivatkozásokat láthatóan megkülönböztessék az oldal többi részétől, és ezzel használható felhasználói felületet biztosítsanak.

A tanszék hallgatói többet is megtudhatnak a főleg nekik szóló információt tartalmazó oldalra mutató hivatkozást követve. A hivatkozást a bekarikázott területre történő



7.18. ábra. *A web architektúrája*

kattintással lehet követni. A böngésző ezután lehozza az új oldalt és megjeleníti azt, ahogyan ez részlegesen látható a 7.18. ábra bal alsó sarkában. Több tucat másik oldal is összekapcsolható az előző példában szereplő első oldallal. A többi oldalon lévő tartalom is lehet ugyanaz(ok)on a gép(ek)en, mint ahol az első is van, vagy a világon bárhol lévő gépeken. A felhasználó ezt nem tudja megmondani. Az oldalt a böngésző hozza le, bármiféle felhasználói segítség nélkül. Tehát a tartalom megtekintése közben a gépek közti adatmozgatás észrevétlenül történik.

Az oldalak megjelenítése mögötti alapmodell is látszik a 7.18. ábrán. A böngésző a kliensgépen jeleníti meg a weboldalt. Minden oldal letöltése egy vagy több szerverhez intézett kérés elküldésével, majd a szerverek által az oldal tartalmának visszaküldésével történik. Az oldalak lekérésére szolgáló protokoll olyan egyszerű, szöveges alapú kérés-válasz protokoll, amely TCP fölött működik, akárcsak az SMTP. A neve **HTTP** (**HyperText Transfer Protocol – hipertext-átviteli protokoll**). A tartalom lehet egyszerűen egy lemezről leolvasott dokumentum vagy egy adatbázis-lekérdezés és programvégrehajtás eredménye. Az oldal **statikus** (**static page**), ha az egy dokumentum, amely mindig ugyanolyan megjelenítést eredményez. Ezzel ellentétben, ha azt egy program igény szerint állította elő vagy egy programot tartalmaz, akkor az egy **dinamikus oldal** (**dynamic page**).

Egy dinamikus oldal minden megjelenítés alkalmával másképpen nézhet ki. Például egy elektronikai üzlet nyitóoldala minden egyes látogató számára eltérő lehet. Ha egy könyvesbolti vásárló korábban misztikus regényeket vásárolt, a bolt főoldalát meglátogatva valószínűleg új krimiket lát majd kiemelten megjelenítve, míg egy kulináris érdeklődésű vásárlót új szakácskönyvekkel üdvözölhetnek. Hamarosan elmondjuk, hogyan követik nyomon a webhelyek, hogy ki mit kedvel. Röviden, a válaszban sütikről esik szó (még a kulináris érdeklődésű látogatók számára is).

Az ábrán a böngésző három szerverhez kapcsolódik, hogy letöltse a két oldalt; a *cs.washington.edu*-hoz, a *youtube.com*-hoz és a *google-analytics.com*-hoz. A különböző szerverektől származó tartalmakat a böngésző egyesíti a megjelenítéshez. A megjelenítés különféle, a tartalom jellegétől függő feldolgozások sorát eredményezi. A szöveg és grafika megjelenítése mellett beletartozhat ebbe videolejátszás vagy egy parancsfájl futtatása, ami az oldal részeként megjeleníti a saját felhasználói felületét. Ebben az esetben a *cs.washington.edu* szerver szolgáltatja a főoldalt, a *youtube.com* szerver a beágyazott videót, a *google-analytics.com* azonban semmi olyasmit nem nyújt, amit a felhasználó láthat, de nyomon követi a webhely látogatóit. Később bővebben is szót ejtünk a nyomkövető szerverekről.

Az ügyfél (kliens) oldala

Nézzük meg most részletesebben a 7.18. ábrán a webböngésző oldalát! A böngésző lényegében egy olyan program, mely képes megjeleníteni egy weboldalt és kezelni a megjelenített oldalon lévő elemekre történt kattintásokat. Amikor egy elemet kiválasztanak, a böngésző követi a hiperhivatkozást és letölti a kiválasztott oldalt.

A web létrehozásának kezdetén nyilvánvaló volt, hogy az egymásra mutató weboldalaknak szükségük van valamilyen módszerre az oldalak megnevezéséhez és helyének megállapításához. Három kérdést kellett megválaszolni, mielőtt a kiválasztott oldalt meg lehetett volna jeleníteni:

1. Mi az oldal neve?

2. Hol található az oldal?

3. Hogyan lehet elérni az oldalt?

Ha valamilyen módon minden oldalnak egyedi neve lenne, nem lenne semmi félreérthető az oldalak azonosításában. A probléma azonban ezzel még nem lenne megoldva. Gondoljuk át az emberek és oldalak közötti párhuzamot! Az Egyesült Államokban csaknem mindenkinek van társadalombiztosítási száma, ami egyedi azonosító, mert állítólag nincs két ember, akinek a száma ugyanolyan lenne. Mindazonáltal, ha valakinek csak egy társadalombiztosítási száma van, nem lehet kitalálni a tulajdonos címét, és valószínűleg azt sem lehet megmondani, hogy vajon angolul, spanyolul vagy kínaiul kellene levelet írni annak a személynek. A web alapvetően ugyanezekkel a problémákkal küzd.

A kiválasztott megoldás úgy azonosítja az oldalakat, hogy közben egyszerre oldja meg mindhárom problémát. Minden oldalt egy **URL (Uniform Resource Locator – egységes erőforrás-meghatározó)** segítségével jelölnek meg, ami valójában az oldal egész világon használt neve. Az URL-ek három részből állnak: a protokollból (ami **sémaként [scheme]** is ismert), annak a gépnek a DNS-nevéből, amelyen az oldal megtalálható, és az elérési útból, ami egyedileg azonosít egy bizonyos oldalt (egy beolvasandó fájlt vagy egy gépen lefuttatandó programot). Általános esetben az út egy hierarchikus név, ami a könyvtárrendszer struktúráját követi. Az út értelmezése azonban a szerverre van bízva; vagy tükrözi az aktuális könyvtárstruktúrát, vagy nem.

Például a 7.18. ábrán látható URL a következő:

http://www.cs.washington.edu/index.html

Ez az URL három részből áll: a protokollból (*http*), a hoszt DNS-nevéből (*www.cs.washington.edu*) és az útból (*index.html*).

Amikor a felhasználó rákattint egy hiperhivatkozásra, a böngésző lépéseket tesz annak érdekében, hogy elhozza a hiperhivatkozás által mutatott oldalt. Kövessük végig a példabeli hivatkozásunk kiválasztásának lépéseit!

1. A böngésző meghatározza az URL-t (megnézi, hogy mit választottak ki).

2. A böngésző megkérdezi a DNS-től a *www.cs.washington.edu szerver* IP-címét.

3. A DNS válasza: 128.208.3.88.

4. A böngésző létrehoz egy TCP-összeköttetést a 128.208.3.88 című hoszt 80-as portjával, a HTTP-protokoll jól ismert portjával.

5. Átküld egy kérést, melyben elkéri az */index.html* oldalt.

6. A *www.cs.washington.edu* kiszolgáló elküldi az oldalt egy HTTP-válasz formájában, például átküldi az */index.html* állományt.

7. Ha az oldal olyan URL-eket tartalmaz, amelyek a megjelenítéshez szükségesek, a böngésző ugyanezzel a módszerrel lekéri a többi URL-t is. Ebben az esetben az URL-ek között található több, szintén a *www.cs.washington.edu*-ról letöltött, beágyazott kép, egy beágyazott videó a *youtube.com*-ról és egy parancsfájl a *google-analytics.com*-ról.

8. A böngésző megjeleníti az */index.html* oldalt, ahogyan az a 7.18 ábrán látható.

9. A TCP-összeköttetést lebontják, ha ugyanezekhez a szerverekhez rövid időn belül nem intéznek további kéréseket.

Sok böngésző a képernyő alján egy státussorban jelzi ki, hogy éppen melyik lépést hajtja végre. Ily módon, ha a teljesítmény gyatra, a felhasználó láthatja, hogy ez azért van, mert a DNS nem válaszol, vagy a kiszolgáló nem válaszol, vagy csak egyszerűen lassú a hálózat, mivel torlódás lépett fel az oldal átvitele közben.

Az URL-eket nyílt végűre tervezték abban az értelemben, hogy egyszerű legyen többféle protokollt használó böngészőket készíteni a különféle típusú erőforrások elérése érdekében. Valójában számos más protokollhoz is meghatároztak URL-eket. A gyakoribbak egy kissé leegyszerűsített formában a 7.19. ábrán láthatók.

Nézzük át röviden a listát! A *http* protokoll a web eredeti nyelve, amin a webszerverek beszélnek. A **HTTP** a **HyperText Transfer Protocol (hipertext-átviteli protokoll)** rövidítése. Ebben a fejezetben később részletesen is meg fogjuk vizsgálni.

Az *ftp* protokollt a fájlok FTP-vel, az internetes fájlátviteli protokoll segítségével való eléréséhez használják. Az FTP időben megelőzte a webet, és még több mint három évtized múltán is használják. A web megkönnyíti a szerte a világban lévő FTP-szervereken elhelyezett nagyszámú állomány megszerzését, mivel egyszerű, kattintható felhasználói

Név	Rendeltetés	Példa
http	Hipertext (HTML)	http://www.ee.uwa.edu/~rob/
https	Hipertext biztonságos átvitele	https://www.bank.com/accounts/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Helyi állomány	file:///usr/suzanne/prog.c
mailto	E-levél küldése	mailto:JohnUser@acm.org
rtsp	Valós idejű média letöltése	rtsp://youtube.com/montypython.mpg
sip	Multimédiás hívások	sip:eve@adversary.com
about	Böngésző információ megjelenítése	about:plugins

7.19. ábra. *Néhány gyakori URL-séma*

felületet biztosít a parancssoros felület helyett. Ez az információhoz való tökéletesített hozzáférés az egyik oka a web látványos fejlődésének.

Egy helyi állományt weboldalként lehet elérni a *file* protokoll segítségével, vagy még egyszerűbben, csak megnevezve azt. Ez a megközelítés nem igényel szervert. Természetesen csak helyi fájlokkal működik, távoliakkal nem.

A *mailto* protokoll igazából nem rendelkezik weboldal-lekérési tulajdonsággal, de azért hasznos. Lehetővé teszi a felhasználóknak, hogy e-levelet küldjenek a webböngészőből. A legtöbb böngésző úgy reagál egy *mailto* hivatkozásra, hogy az üzenet megírásához elindítja a felhasználói ügynököt azzal a címmezővel, amely már ki van töltve.

Az *rtsp* és *sip* protokollok valós idejű multimédia-letöltések munkameneteinek létrehozására, valamint hang- és videohívásokhoz használatosak.

Végül az *about* protokoll egy szabály, amely információval szolgál a böngészőről. Például az *about:plugins* hivatkozás követése esetén a legtöbb böngésző olyan oldalt jelenít meg, amely felsorolja azokat a MIME-típusokat, amelyeket böngészők a kiterjesztéseik, az ún. beépülő modulok (plug-in) segítségével kezelnek.

Röviden, az URL-eket úgy tervezték meg, hogy ne csak a felhasználók weben való szörfözését könnyítsék meg, hanem futni tudjanak egyrészt a régebbi protokollok, mint az FTP és az e-levél, valamint újabb, hang- és videoadatok átviteléhez használatos protokollok is, továbbá kényelmes hozzáférést biztosítson a helyi állományokhoz és böngészőinformációhoz. Ez a megközelítés feleslegessé tette a többi szolgáltatás használatához szükséges, különleges felhasználói felülettel ellátott programokat, és csaknem minden internetes anyag elérése egyetlen programba, a webböngészőbe kerülhetett. Ha az ötlet nem egy svájci kutatólaboratóriumban dolgozó brit fizikusnak jutott volna eszébe, könnyen azt képzelhetnénk, hogy a tervet valamelyik szoftvercég hirdetési részlege álmodta meg.

Mindezen kellemes tulajdonságok ellenére a web növekvő használata felfedte az URL-sémában rejlő egyik gyengeséget. Az URL egy meghatározott hosztra mutat, de néha hasznos lenne, ha egy oldalra anélkül lehetne hivatkozni, hogy ezzel egyidejűleg megmondanánk, hogy az oldal hol található. Például a sűrűn hivatkozott oldalak esetében jó lenne, ha egymástól távol több másolata is létezne, a hálózati forgalom csökkentése érdekében. Sajnos nincs rá mód, hogy azt mondjuk: „Nekem az *xyz* oldalra van szükségem, de nem érdekel, honnan szerzed meg."

Az ilyen típusú problémának a megoldására hozták létre az URL-ek általánosított változatát, az **URI**-t (**Uniform Resource Identifier – egységes erőforrás-azonosító**). Bizonyos URI-k megadják, hogyan kell megtalálni egy erőforrást. Ezek az URL-ek. Más URI-k megadják az erőforrás nevét, de azt már nem, hogy hol található. Ezeket az URI-kat **URN**-nek (**Uniform Resource Names – egységes erőforrásnév**) nevezik. Az URI-k írásának szabályait az RFC 3986 határozza meg, míg a használatban lévő különféle sémákat az IANA követi nyomon. A 7.19. ábrán feltüntetett sémákon kívül még sok egyéb típusa létezik az URI-knak, de az itt látható sémák a meghatározói a ma használatos webnek.

MIME-típusok

Ahhoz, hogy meg tudja jeleníteni az új oldalt (vagy bármelyik oldalt), a böngészőnek értenie kell az oldal formátumát. Az oldalakat egy HTML nevű, szabványosított nyelven

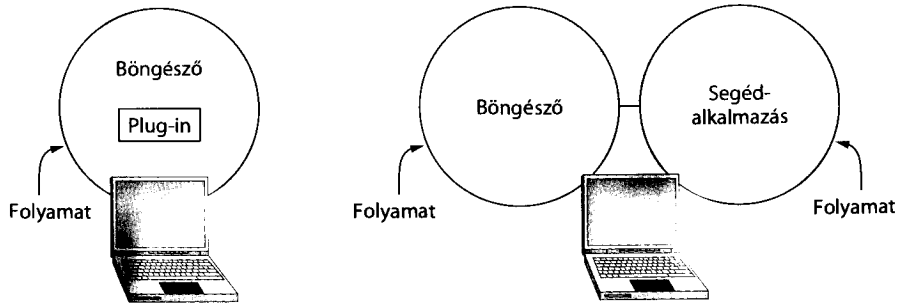
írják, hogy minden böngésző megértsen minden weboldalt. Ez (jelenleg) a web közvetítőnyelve (lingua franca), amellyel e fejezetben később még részletesen foglalkozunk.

Bár a böngésző alapjában véve csak egy HTML-értelmező, a legtöbb böngésző számos gombbal és funkcióval rendelkezik, hogy megkönnyítsék a weben való navigálást. Többnyire van egy gombjuk az előző oldalra való visszalépéshez, egy gombjuk a következő oldalra való előrelépéshez (ez csak akkor működik, ha a felhasználó előzőleg már visszalépett), és egy olyan gombjuk, amivel a felhasználó egyenesen a saját kezdőoldalára juthat. A legtöbb böngészőnek van egy gombja vagy menüpontja, amivel könyvjelzőt tehet egy adott oldalra, és egy másik, amivel meg lehet jeleníteni a könyvjelzők listáját, így pár egérkattintással bármelyiket újra meglátogathatjuk.

Ahogyan példánk mutatja, a HTML-oldalak nem csak egyszerű szöveget és hipertextet tartalmazhatnak, hanem számos más tartalmi elemet is. A még általánosabb működés érdekében nem kell minden oldalnak HTML-t tartalmaznia. Egy oldal lehet MPEG-formátumú videó, PDF-formátumú dokumentum, JPEG-formátumú fénykép, MP3-formátumú dal, vagy a több száz másik állománytípus közül bármelyik. Mivel a szabványos HTML-oldalak ezek közül bármelyikre hivatkozhatnak, a böngészőnek gondjai lesznek, amikor olyan oldallal találkozik, amit nem tud értelmezni.

A különböző állománytípusok száma rohamosan nő. A legtöbb esetben ezért, mert ahelyett hogy újabb és újabb értelmezők beépítésével egyre nagyobbá tették volna a böngészőket, egy sokkal általánosabb megoldást választottak. Amikor a kiszolgáló elküld egy oldalt, egyúttal elküld némi járulékos információt is az oldalra vonatkozóan. Ez az információ tartalmazza az oldal MIME-típusát (lásd 7.13. ábra). A böngésző a *text/html* típusú oldalakat és még néhány beépített típust közvetlenül megjelenít. Ha az adott MIME-típus nincs a beépítettek között, akkor a böngésző megnézi a MIME-típusokat tartalmazó táblázatát, hogy megtudja, hogyan kell megjelenítenie az oldalt. A táblázat minden MIME-típushoz egy megjelenítőt társít.

A böngészők bővítése kétféleképpen, beépülő modulok vagy segédalkalmazások segítségével lehetséges. A **beépülő modul** (**plug-in**) egy olyan harmadik fél által készített szoftvermodul, ami a böngésző bővítményeként van telepítve, amint azt a 7.20.(a) ábra szemlélteti. Gyakori beépülő modul például a PDF, a Flash és a Quicktime, amelyek dokumentumok megjelenítését, hangok és videók lejátszását végzik. A beépülő modulok a böngészőn belül futnak, ezért hozzáférhetnek az aktuális oldalhoz és módosíthatják annak megjelenését is.



7.20. ábra. *(a) Egy böngésző beépülő modul. (b) Egy segédalkalmazás*

A böngészőknek van egy eljáráskészletük, melyet a beépülő moduloknak implementálniuk kell, hogy meghívhatók legyenek a böngészőből. Például általában van egy olyan eljárás, amit a böngésző azért hív meg, hogy megadja a beépülő modulnak a megjelenítendő adatokat. Ez az eljáráskészlet a beépülő modul interfésze, ami böngészőnként eltérő lehet.

Mindemellett a böngésző is kínál szolgáltatásokat saját eljáráskészletén keresztül a beépülő moduloknak. A böngésző interfészében jellemzően olyan eljárások vannak, amelyek lehetővé teszik a memória lefoglalását és felszabadítását, egy üzenet megjelenítését a státussorban vagy paraméterek lekérdezését a böngészőtől.

Ahhoz, hogy a beépülő modult használni lehessen, előbb telepíteni kell. A szokásos eljárás az, hogy a felhasználó ellátogat a beépülő modul weboldalára, és letölt egy telepítőállományt. A telepítőállomány futtatása kicsomagolja a beépülő modult, majd meghívja a szükséges eljárásokat, hogy bejegyezze a beépülő modul MIME-típusát a böngészőhöz, és társítsa vele a beépülő modult. A böngészők általában előre telepített formában tartalmazzák a népszerű beépülő modulokat.

A böngészők kiterjesztésének másik módja a **segédalkalmazások (helper applications)** használata. Ezek önálló programok, melyek külön folyamatként futnak, ahogy azt a 7.20.(b) ábra szemlélteti. Mivel a segédalkalmazás különálló program, az interface karnyújtásnyi távolságra van a böngészőtől. Általában inkább csak megkapja a letöltött ideiglenes állomány nevét, megnyitja az állományt, majd megjeleníti a tartalmát. A segédalkalmazások többnyire nagy programok, melyek a böngészőtől függetlenül léteznek, mint például a Microsoft Word vagy PowerPoint.

Sok segédalkalmazás használja az *application* MIME-típust. Ennek következtében meglehetősen sok altípust definiáltak hozzá, például az *application/vnd.ms-powerpoint*-ot a PowerPoint-állományoknak. A *vnd* jelzi a gyártóspecifikus formátumokat. Ily módon egy URL mutathat közvetlenül egy PowerPoint-állományra, majd amikor a felhasználó rákattint, a PowerPoint automatikusan elindul, és lekezeli a megjelenítendő tartalmat. A segédalkalmazásoknak nem kell az *application* MIME-típus használatára szorítkozniuk. Az Adobe Photoshop például az *image/x-photoshop* típust használja.

A böngészők tehát gyakorlatilag korlátlan számú dokumentumtípus kezelésére is beállíthatók anélkül, hogy bármit is változtatni kellene rajtuk. A korszerű webszerverek gyakran több száz típus/altípus kombinációt kezelnek, és ehhez még minden alkalommal újabbak jönnek, amint egy új programot telepítenek.

Ütközések forrása lehet, ha több beépülő modul és segédalkalmazás is elérhető ugyanahhoz az altípushoz, például a *video/mpeg*-hez. Ilyenkor az történik, hogy az utoljára regisztráló szoftver felülírja a meglevő MIME-típustársítást, kisajátítva ezzel a típust. Következésképpen, egy új program telepítése után a böngésző lehet, hogy másképp fogja kezelni a meglevő típusokat.

A böngészők nem csak a távoli webszerverekről tölthetnek le állományokat, hanem megnyithatják azokat helyileg is anélkül, hogy hálózat lenne a láthatáron. A böngészőnek azonban valahogyan ki kell találnia az állomány MIME-típusát. Az operációs rendszerek szabványos megoldása a fájl kiterjesztésének MIME-típussal történő társítása. Egy tipikus konfiguráció esetén a *foo.pdf* a böngészőben fog megnyílni az *application/pdf* beépülő modul segítségével, a *bar.doc* pedig a Wordben, mint az *application/msword* segédalkalmazás.

Itt is adódhatnak ütközések, hiszen egyszerre több program is tudja – mondjuk inkább azt, hogy szeretné tudni – kezelni például az *mpg* állományokat. A tapasztaltabb felhasználóknak szánt programok a telepítés során gyakran jelölőnégyzeteket (checkbox) adnak meg minden olyan MIME-típushoz és kiterjesztéshez, amit kezelni képesek. A felhasználó így kiválaszthatja a számára szükséges típusokat, és nem fogja akaratlanul felülírni a meglevő társításokat. A nagyközönségnek szánt programok azonban azt feltételezik, hogy a felhasználónak fogalma sincs róla, hogy mi az a MIME-típus, úgyhogy magukhoz ragadnak mindent, amit csak tudnak, tekintet nélkül arra, hogy mit tettek az addig telepített programok.

A böngészők sokféle új típussal való kibővítésének lehetősége kényelmes dolog, de bajokhoz is vezethet. Amikor egy Windows-os PC-n lévő böngésző letölt egy *exe* kiterjesztésű állományt, látja, hogy ez egy futtatható program, vagyis nincs hozzá segédalkalmazás. Ekkor kézenfekvő lépés a program futtatása. Ez azonban óriási biztonsági rést jelentene. Egy rosszindulatú webhelynek nincs is más dolga, mint hogy kiállít egy oldalt, mondjuk, filmsztárok vagy élsportolók képeivel, melyek mind egy vírusra mutatnak. Egy kattintás egy képre, és máris letöltődik egy ismeretlen és esetleg ellenséges végrehajtható program, és futni kezd a felhasználó gépén. Az ilyen hívatlan vendégek elkerülésére a Firefox és más böngészők úgy vannak beállítva, hogy az ismeretlen programok automatikus futtatásával legyenek óvatosak, de nem minden felhasználó tudja, hogy melyek a biztonságos, ám kényelmetlen döntések.

A kiszolgáló (szerver) oldala

Ennyit tehát az ügyfél oldaláról, és most vessünk egy pillantást a kiszolgáló oldalára is. Amint azt már láttuk, a böngésző az URL beírása vagy egy hipertextsorra való kattintás után feldolgozza az URL-t, és a *http://* valamint a következő perjel közötti részt kikeresi a DNS-ből. Ha megvan a kiszolgáló IP-címe, a böngésző kiépít egy TCP-összeköttetést a kiszolgáló 80-as portjával. Ezután átküld egy parancsot, ami tartalmazza az URL hátralevő részét, mely az adott kiszolgálón levő oldalhoz vezető út. A kiszolgáló végül elküldi az állományt megjelenítésre a böngészőnek.

Első közelítésben egy webszerver a 6.6. ábrán látható kiszolgálóhoz hasonlóan néz ki. Az a kiszolgáló egy állomány nevét kapja meg, amit meg kell keresnie, és vissza kell küldenie a hálózaton keresztül. Bármelyik esetet is nézzük, a kiszolgáló a következő lépéseket hajtja végre a központi ciklusában:

1. Elfogad egy TCP-összeköttetést az ügyféltől (a böngészőtől).
2. Megkapja az oldalhoz vezető utat, ami a kért állomány neve.
3. Megkeresi az állományt (a háttértáron).
4. Elküldi az állomány tartalmát az ügyfélnek.
5. Bontja a TCP-összeköttetést.

A modern webszervereknek több funkciójuk van ennél, de abban az egyszerű esetben, ha a tartalmat egy fájl tárolja, alapjában véve ennyi egy webszerver feladata. A di-

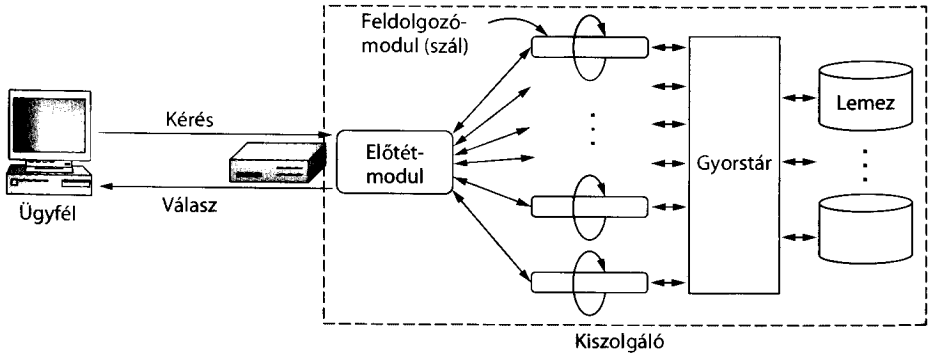
namikus tartalom esetében a harmadik lépést egy, az elérési út alapján meghatározott program végrehajtása helyettesítheti, és ez küldi vissza a tartalmakat.

A webszervereket azonban másképp valósították meg annak érdekében, hogy másodpercenként sok kérést legyenek képesek kiszolgálni. Az egyik probléma az egyszerű megvalósítással, hogy a fájlok elérése gyakran torlódást okoz. A lemezről történő olvasások nagyon lassúak a program végrehajtásához képest, és az operációs rendszer függvényeinek hívásai következtében ugyanazok a fájlok újra meg újra beolvasásra kerülhetnek a lemezről. A másik probléma, hogy egyszerre csak egy kérést dolgoznak fel. Az állomány nagy lehet, és annak továbbítása idején a többi kérés blokkolva várakozni kényszerül.

Az első nyilvánvaló javítási lehetőség (amit minden webszerver alkalmaz is), hogy egy **gyorstárat** (**cache**) tartanak a memóriában, mely az n legutoljára beolvasott állományt vagy egy bizonyos gigabájtnyi tartalmat tárol. A kiszolgáló mindig megnézi a gyorstárat, mielőtt a merevlemezhez fordulna. Ha ott megvan az állomány, akkor rögtön ki lehet olvasni a memóriából, azaz nincs szükség lemezhozzáférésre. A hatékony gyorstárazáshoz sok központi memória és valamivel több feldolgozási idő is szükséges (a tárban való keresés, illetve a tár tartalmának kezelése miatt). Az így elérhető időmegtakarítás által az erőfeszítések és kiadások azonban még így is szinte mindig megtérülnek.

Az egyidejűleg több mint egyetlen kérés kiszolgálásának problémájával való megbirkózásra az egyik stratégia a szerver **többszálúvá** (**multithreaded**) tétele. Az egyik lehetséges megoldás szerint a kiszolgáló egy előtétmodulból (front-end module) és k darab feldolgozómodulból áll, amint azt a 7.21. ábra mutatja. Az előtétmodul fogadja az összes beérkező kérést. Az így létrejött $k+1$ szál mind ugyanahhoz a folyamathoz tartozik, tehát a feldolgozómodulok mind hozzáférhetnek a folyamat címterében lévő gyorstárhoz. Amikor beérkezik egy kérés, az előtétmodul fogadja azt, és létrehoz egy rövid leíró rekordot, majd átadja a rekordot az egyik feldolgozómodulnak.

A feldolgozómodul először a gyorstárban keresi a kért állományt. Ha ott van, akkor átírja a rekordot, hogy legyen benne egy mutató az adott állományra. Ha nincs ott, akkor egy lemezműveletet indít, hogy beolvassa az állományt a gyorstárba (és esetleg kidob pár másik állományt a tárból, hogy legyen elég hely). Amikor az állomány betöltődött a lemezről, bekerül a gyorstárba, és visszaküldik az ügyfélnek is.



7.21. ábra. *Egy többszálú webszerver egy előtét- és több feldolgozómodullal*

Ennek a sémának az az előnye, hogy amíg egy vagy több feldolgozómodul blokkolódik, mert egy lemez- vagy hálózati művelet befejezésére vár (és így nem fogyaszt processzoridőt), addig más modulok aktívan dolgozhatnak a többi kérésen. k darab feldolgozómodullal k-szor nagyobb átbocsátóképesség is elérhető az egyszálú szerverrel szemben. Persze ha a lemez vagy a hálózat a korlátozó tényező, akkor több lemezre vagy gyorsabb hálózatra van szükség az egyszálú modellel szembeni bármilyen tényleges előrelépéshez.

A korszerű webszerverek nem pusztán állományneveket fogadnak és állományokat adnak vissza. A valóságban a kérések feldolgozása egészen bonyolult is lehet. Ebből kifolyólag sok kiszolgálóban minden feldolgozómodul egy lépéssorozatot hajt végre. Az előtétmodul átad minden beérkező kérést az első rendelkezésre álló feldolgozómodulnak, amely erre a következő lépések egy részhalmazát hajtja végre attól függően, hogy melyek szükségesek az adott kérés feldolgozásához. Ezekre a lépésekre a TCP-összeköttetés és bármelyik biztonságos szállítási mechanizmus (például a 8. fejezetben ismertetendő SSL/TLS) létrehozása után kerül sor.

1. A feldolgozómodul feloldja a kért weboldal nevét.
2. Ellenőrzi a weboldalra vonatkozó hozzáférési jogosultságokat.
3. Megnézi a gyorstárat.
4. Betölti a kért oldalt a háttértárról vagy lefuttat egy programot, ami létrehozza azt.
5. Megállapítja a válasz többi részét (például az elküldendő MIME-típust)
6. Elküldi a választ az ügyfélnek.
7. Elhelyez egy bejegyzést a kiszolgáló naplójában.

Az első lépésre azért van szükség, mert a bejövő kérés nem biztos, hogy szó szerint tartalmazza az állomány vagy program tényleges nevét. Beépített rövidítéseket tartalmazhat, amelyeket le kell fordítani. Egyszerű példa a *http://www.cs.vu.nl/* URL. Itt az állománynév üres, tehát az URL-t ki kell egészíteni valamilyen alapértelmezett állománynévvel, általában az *index.html*-lel. Egy másik általános szabály a *~felhasznalo/* leképezése a *felhasználó* webkönyvtárára. Ezek a szabályok együtt is alkalmazhatóak. Az egyik szerző (AST) honlapja tehát a

http://www.cs.vu.nl/~ast

címen érhető el annak ellenére, hogy az állomány neve *index.html* egy bizonyos alapértelmezett könyvtárban.

A modern böngészők a konfigurációs információt is meg tudják állapítani, mint például a böngészőszoftver típusa vagy a felhasználó alapértelmezett nyelve (például olasz vagy angol). Ez lehetővé teszi, hogy a kiszolgáló mobil készülék esetén kis képeket tartalmazó, és az előnyben részesített nyelvű weboldalt válassza, ha van ilyen. A névkiegészítés

általában nem annyira magától értetődő feladat, mint amilyennek első látásra tűnik, annak köszönhetően, hogy az utak könyvtárakra és programokra való leképezésére különféle konvenciók léteznek.

A második lépésben a feldolgozó modul megnézi, hogy az oldalra vonatkozó megkötések teljesülnek-e. Nem minden oldal érhető el a nagyközönség számára. Az, hogy az ügyfél lekérheti-e az oldalt, az ügyfél (például felhasználónevekkel és jelszavakkal megadott) személyazonosságától vagy az ügyfél DNS-en vagy IP-címterületen belüli helyétől függhet. Például egy oldalt a vállalaton belüli felhasználókra lehet korlátozni. Ennek megvalósítási módja a szerver felépítésétől függ. Például a népszerű Apache-szerver esetében az a konvenció, hogy egy *.htaccess* nevű, a hozzáférési megszorításokat tartalmazó állományt helyeznek el abban a könyvtárban, amelyik a korlátozott elérésű oldalt tartalmazza.

A harmadik és a negyedik lépés jelenti az oldal megszerzését. A feldolgozás szabályain múlik, hogy vajon megszerezhető-e a gyorstárból. Például azok az oldalak, amelyeket futó programok hoznak létre, nem mindig gyorstárazhatók, mert minden egyes futás alkalmával eltérő eredményt állíthatnak elő. Alkalmanként még az állományokat is ellenőrizni kell, hogy lássuk, megváltozott-e a tartalmuk, mert a régi tartalmak eltávolíthatók a gyorstárból. Ha az oldalnak egy program futtatására van szüksége, akkor felmerül a program paramétereinek vagy bemenetének beállítási problémája is. Ezek az adatok az útból vagy a kérés más részeiből érkeznek.

Az ötödik lépés a válasz egyéb részeinek megállapításából áll, amelyek az oldal tartalmát kísérik. Egy példa erre a MIME-típus, ami megállapítható a fájl kiterjesztéséből, az állomány vagy program kimenetének első néhány szavából, egy konfigurációs fájlból és valószínűleg más forrásokból is.

A hatodik lépés az oldal visszaküldése a hálózaton keresztül. A teljesítőképesség növelése érdekében a szerver és az ügyfél több oldal lekéréséhez is használhatja ugyanazt a TCP-összeköttetést. Ez az ismételt felhasználás azt jelenti, hogy szükség van valamilyen logikára ahhoz, hogy egy lekérdezést hozzárendeljen egy megosztott összeköttetéshez, és hogy minden egyes választ úgy küldjön vissza, hogy az korrekten kapcsolódik a kéréshez.

A hetedik lépésben más fontos nyilvántartások vezetése mellett egy bejegyzés kerül a rendszernaplóba adminisztratív célból. Az ilyen naplókból később értékes információt lehet kibányászni a felhasználók szokásaira vonatkozóan, például arról, hogy milyen sorrendben látogatják a felhasználók az oldalakat.

Sütik

A weben történő navigálás, ahogyan azt eddig leírtuk, egymástól független oldalak lekérésének sorozatából áll. Nincs semmilyen koncepció egy bejelentkezési munkamenetre (login session). A böngésző elküld egy kérést, a kiszolgáló pedig visszaküld egy állományt, aztán elfelejti, hogy valaha is látta már az adott ügyfelet.

Ez a modell tökéletesen megfelel akkor, ha nyilvánosan hozzáférhető dokumentumokat akarunk elérni, és jól működött abban az időben, amikor a webet létrehozták. Nem alkalmas azonban arra, hogy különböző felhasználóknak más-más oldalakat küldjön

vissza attól függően, hogy korábban mit kértek a szervertől. A webhellyekkel folytatott számos együttműködéshez, interakcióhoz szükséges ez a viselkedés. Az ügyfeleknek például néhány webhelyen (például újságoknál) regisztrálniuk kell magukat (esetleg fizetniük is kell), hogy használhassák az oldalakat. Ez felveti a kérdést: hogyan tudja a kiszolgáló megkülönböztetni a korábban már regisztrált felhasználóktól érkező kéréseket a többi felhasználó kérésétől? A második példát az e-kereskedelem adja. Ha egy felhasználó egy elektronikus áruházban bóklászik, és időnként belerak egy-egy árut a bevásárlókocsijába, akkor hogyan tudja a kiszolgáló nyomon követni a kocsi tartalmát? A harmadik példa a személyre szabható portálok esete, mint amilyen a Yahoo!. A felhasználók beállíthatják személyre szabott, részletes kezdőoldalukat, hogy az csak azt az információt tartalmazza, amelyekre ők kíváncsiak (például a részvényeik árfolyamáról és kedvenc sportcsapataikról). De hogyan tudja a kiszolgáló megjeleníteni a helyes oldalt, ha nem tudja, hogy ki a felhasználó?

Elsőre azt gondolhatnánk, hogy a kiszolgálók a felhasználókat az IP-címük megfigyelésével követik nyomon. Csakhogy ez az ötlet nem működik. Sok felhasználó dolgozik mások által is használt számítógépen, különösen otthon, és az IP-cím csupán a számítógépet azonosítja, nem a felhasználót. Még rosszabb, hogy sok vállalat NAT-ot használ, így minden felhasználó kimenő csomagjai ugyanazt az IP-címet hordozzák, azaz a szerver számára valamennyi NAT mögötti számítógép egyforma. Sok ISP DHCP segítségével rendeli az IP-címeket az ügyfeleihez. Az IP-címek idővel megváltoznak, így a szerver számára Ön egyszer csak a szomszédjának fog tűnni. Mindezek miatt a szerver nem használhatja az IP-címeket a felhasználók követésére.

Ezt a problémát oldja meg a gyakran bírált **sütik** (**cookies**) nevű mechanizmus. A név az ősi programozói zsargonból származik, és arra a helyzetre utal, amikor egy program meghív egy eljárást, és visszakap valamit, amit később esetleg fel kell mutatnia, hogy elvégeztessen valamit. Ebben az értelemben egy UNIX-állományleíró vagy egy Windows-objektumazonosító is tekinthető sütinek. A sütiket először a Netscape-böngészőben valósították meg 1994-ben, és ma az RFC 2109 részletezi.

Amikor egy ügyfél elkér egy weboldalt, a kiszolgáló a kért oldal mellett egy süti formájában járulékos információt is megadhat. A süti egy meglehetősen kicsi (legfeljebb 4 KB méretű), névvel ellátott karakterlánc, amit a szerver a böngészőhöz társíthat. Ez a társítás még mindig nem egyenértékű a felhasználó azonosításával, de az IP-címnél sokkal közelebb áll hozzá, és sokkal hasznosabb. A böngészők a felkínált sütiket egy bizonyos ideig, általában az ügyfél lemezének sütikönyvtárában tárolják úgy, hogy a sütik megmaradjanak a böngésző indításai során addig, amíg a felhasználó le nem tiltja a sütik használatát. A sütik nem futtatható programok, csak egyszerű karakterláncok. Elvileg egy süti is tartalmazhatna vírust, de mivel egyszerű adatként kezelik, a vírusnak hivatalosan nincs lehetősége ténylegesen futni és károkat okozni. Néhány számítógépes kalóznak persze mindig sikerül kihasználnia egy böngészőhibát, és elérnie az aktiválást.

Egy süti legfeljebb öt mezőt tartalmazhat, ahogy azt a 7.22. ábra mutatja. A *Körzet* megmondja, honnan származik. A böngészők dolga, hogy ellenőrizzék, hogy a kiszolgálók nem hazudnak-e a körzetükről. Az egyes körzeteknek ügyfelenként legfeljebb 20 sütit kell tárolniuk. Az *Útvonal* a kiszolgáló könyvtárrendszerének egy útvonalát jelenti, és meghatározza, hogy a kiszolgáló állományfájának melyik része használhatja a sütit. A mező értéke gyakran /, ami az egész fát jelenti.

Körzet	Útvonal	Tartalom	Lejárat	Biztonságos
toms-casino.com	/	CustomerID=297793521	15-10-10 17:00	Igen
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-11 14:22	Nem
aportal.com	/	Prefs=Stk:CSCO+ORCL;Spt:Jets	31-12-20 23:59	Nem
alattomos.com	/	UserID=4627239101	31-12-19 23:59	Nem

7.22. ábra. *Néhány példa a sütikre*

A *Tartalom* mező *név = érték* felépítésű. A kiszolgáló bármit írhat a *név* és az *érték* helyére is. Ebben a mezőben tárolják a süti tényleges tartalmát.

A *Lejárat* mező adja meg, hogy mikor jár le a süti érvényessége. Ha ez a mező üres, akkor a böngésző kilépéskor eldobja a sütit. Az ilyen sütiket **nemtartós sütinek** (**nonpersistent cookie**) nevezzük. Ha meg van adva egy időpont és dátum, akkor a süti **tartós** (**persistent**), és a lejárat idejéig megőrzik. Az időt a greenwichi idő szerint (GMT) adják meg. Ha a kiszolgáló el akar távolítani egy sütit egy ügyfél merevlemezéről, akkor egyszerűen elküldi még egyszer, csak múltbeli lejárati idővel.

Végül, a *Biztonságos* mezőt úgy lehet beállítani, hogy azt jelezze, hogy a böngésző a kiszolgálónak csak biztonságos átvitellel, nevezetesen SSL/TLS segítségével (amelyet a 8. fejezetben ismertetünk) küldheti vissza a sütit. Ezt a funkciót az e-kereskedelmi, banki és egyéb biztonsági alkalmazásokban használják.

Azt már láttuk, hogyan lehet megszerezni a sütiket, de hogyan használják azokat? Nos, mielőtt a böngésző elküldene egy kérést valamely webhelyre, megnézi sütikönyvtárát, hogy a kérés célját képező körzetből helyeztek-e már el ott sütiket. Ha igen, az összes ilyen sütit, de csak ezeket mellékelik a kéréshez. A kiszolgáló a visszakapott sütiket úgy értelmezi, ahogyan csak akarja.

Nézzünk meg pár lehetséges használati esetet. A 7.22. ábrán az első sütit a *toms-casino.com* helyezte el, hogy azonosíthassa a látogatót. Amikor az ügyfél a következő héten visszatér, hogy még több pénzt szórjon el, a böngésző átküldi a sütit, hogy a kiszolgáló tudja, kiről is van szó. A látogató azonosítójával felvértezve a kiszolgáló megkeresheti a látogató rekordját egy adatbázisban, hogy ezen információ felhasználásával egy megfelelő weboldalat állítson elő. Ez az oldal a látogató ismert szokásainak megfelelően tartalmazhat egy pókerleosztást, az aznapi lóversenyek listáját vagy egy félkarú rablót.

A második süti a *jills-store.com*-tól érkezett. Itt a felállás az, hogy az ügyfél az áruházban mászkálva mindenféle jó dolgot szeretne vásárolni. Amikor egy jó vételt talál és rákattint, a kiszolgáló beleteszi azt a (szerveren kezelt) bevásárlókosarába, valamint összeállít egy termékkódot is tartalmazó sütit, és visszaküldi ezt az ügyfélnek. Ahogy az ügyfél új oldalakra kattintva tovább bóklászik az áruházban, a sütit minden új oldalkérésnél visszaküldik. A kiszolgáló a közben összegyűlő további megrendeléseket mind hozzáadja a sütihez. Végül, mikor a felhasználó a TOVÁBB A PÉNZTÁRHOZ gombra kattint, a kéréssel együtt a megrendelések immár teljes listáját tartalmazó sütit is elküldik. Ezáltal a kiszolgáló pontosan tudni fogja, hogy a vevő mit kíván megvásárolni.

A harmadik sütit egy webportál használja. Amikor a felhasználó rákattint egy a portálra mutató hivatkozásra, a böngésző átküldi a sütit. A portál ebből tudni fogja, hogy egy olyan oldalt kell összeállítania, mely a Cisco és az Oracle részvényárfolyamait, valamint a New York Jets csapat eredményeit tartalmazza. Mivel egy süti akár 4 KB-os is lehet, rengeteg hely van arra, hogy részletesebb beállításokat is megadhassunk a számunkra érdekes újsághírekre, helyi időjárásra, reklámajánlatokra stb. vonatkozóan.

A sütik felhasználásának egy sokkal vitatottabb területe a felhasználók online szokásainak felderítése. Ez a webhely üzemeltetői számára lehetővé teszi annak megértését, hogy a felhasználók hogyan navigálnak webhelyeiken, a hirdetők számára pedig, hogy profilokat állítsanak össze azokból a hirdetésekből vagy webhelyekből, amelyeket egy adott felhasználó megtekintett. A probléma az, hogy a felhasználókat általában nem érdekli az, hogy tevékenységüket nyomon követik akár részletes profilokkal, akár látszólag semmivel nem kapcsolatos webhelyekkel. Ennek ellenére a **webes nyomonkövetés** (**web tracking**) nagy üzlet. A DoubleClick-et, amely hirdetéseket szolgáltat és követ nyomon, az Alexa webfelügyeleti cég a világ 100 legforgalmasabb webhelye közé sorolta. A Google Analytics-et, ami az operátorok számára nyomon követi a webhely használatát, a világháló 100 000 legforgalmasabb webhelyének több mint a fele használja.

Egy szervernek könnyű nyomon követnie a felhasználó tevékenységét sütik segítségével. Tételezzük fel, hogy egy kiszolgáló nyomon kívánja követni, hány egyedi látogatója volt, és az egyes látogatók hány oldalt tekintettek meg, mielőtt elhagyták a webhelyet. Amikor az első kérés beérkezik, még nem tartozik hozzá süti, tehát a kiszolgáló elküld egy sütit *Számláló = 1* tartalommal. Amikor a webhelyet a felhasználó ismét megtekinti, a sütit mindig visszaküldi a kiszolgálónak. A kiszolgáló a számlálót mindig megnöveli eggyel, és így küldi vissza az ügyfélnek. A számlálók nyomon követése révén a kiszolgáló láthatja, hogy hányan távoznak az első oldal megtekintése után, hányan a második után és így tovább.

A felhasználók böngészési szokásainak oldalak közötti nyomon követése csak egy kicsivel bonyolultabb. Ez a következőképpen működik. Egy reklámügynökség, mondjuk, az Alattomos Hirdetések, felkeresi a fontosabb webhelyeket, és az ügyfeleik termékeit hirdető reklámokat helyez el az oldalaikon, amiért a webhelyek tulajdonosainak bizonyos díjat fizet. Ahelyett azonban, hogy a webhelyeknek egy GIF-állományt adna elhelyezésre, csak egy URL-t ad, hogy azt rakják fel minden oldalra. Minden ilyen URL egy egyedi számot tartalmaz az elérési útban, mint például

http://www.alattomos.com/382674902342.gif

Amikor a felhasználó először látogat meg egy olyan *P* oldalt, amely egy ilyen hirdetést tartalmaz, a böngésző letölti a HTML-állományt. Ezt megvizsgálva észreveszi a *www.alattomos.com*-on levő képre mutató hivatkozást, elküld tehát egy kérést a képre vonatkozóan. Erre visszakap egy hirdetést tartalmazó GIF-állományt, egy egyedi felhasználói azonosítót tartalmazó sütivel együtt, ami a 7.22. ábra esetében 4627239101. Alattomosék feljegyzik maguknak, hogy az ilyen azonosítójú felhasználó meglátogatta a *P* oldalt. Ez könnyen megoldható, mivel az igényelt útra (*382674902342.gif*) csak a *P* oldal hivatkozik. Az adott hirdetés természetesen több ezer weboldalon is megjelenhet, de mindig különböző állománynévvel. Az Alattomos cég feltehetően egy penny töredékét kapja a termékek gyártójától minden továbbított hirdetés után.

Később, amikor a felhasználó meglátogat egy másik, Alattomosék hirdetését tartalmazó weboldalt, a böngésző először elkéri a HTML-állományt a kiszolgálótól. Azután meglátja, mondjuk a *http://www.alattomos.com/193654919923.gif*-re vonatkozó hivatkozást az oldalon, és elkéri ezt az állományt. Mivel rendelkezik már egy sütivel a *www.alattomos.com* körzetből, a böngésző a kéréshez mellékeli az Alattomos-féle sütit, ami a felhasználó azonosítóját tartalmazza. Alattomosék így már a második oldalt ismerik meg, amit a felhasználó meglátogatott.

Az Alattomos cég idővel egy teljes profilt állíthat össze a felhasználó böngészési szokásairól, még akkor is, ha az soha nem kattintott egyik hirdetésre sem. A felhasználó nevével ugyan még nem rendelkezik (bár ismeri az IP-címét, ami elég lehet ahhoz, hogy kikövetkeztesse a nevet más adatbázisokból). Ha viszont a felhasználó csak egyszer is megadja a nevét egy olyan webhelynek, mely együttműködik az Alattomossal, máris meglesz a teljes profilja névvel együtt, és bárki megvásárolhatja azt. Az ilyen adatok eladása elég jövedelmező az Alattomos számára ahhoz, hogy még több helyen még több hirdetést helyezzen el, és így még több adatot gyűjtsön.

Ráadásul, ha az Alattomos szuperalattomos akar lenni, a hirdetésnek nem is kell klasszikus reklámszalagnak (banner) lennie. Egy olyan „hirdetés", ami egyetlen, háttérrel egyező színű képpontból áll (vagyis gyakorlatilag láthatatlan), pontosan ugyanazt eredményezi, mint egy reklámszalag: hatására a böngésző letölti az 1 × 1 méretű GIF-képet, és beküld minden sütit, ami a képpont körzetéből (domainjéből) származik.

A sütik az online magánéleti jogokról szóló viták gyújtópontjába kerültek a fentihez hasonló, nyomkövető viselkedésük miatt. Az egyész ügyben az a legalattomosabb, hogy sok felhasználó egyáltalán nem tud erről az információgyűjtésről, és még akár azt is gondolhatja, hogy biztonságban van, elvégre egyetlen hirdetésre sem kattintott rá. Ebből az okból kifolyólag azokat a sütiket, amelyek a felhasználókat az oldalak között nyomon követik, sokan **kémprogramnak (spyware)** tekintik. Vessen egy pillantást a böngészője által tárolt sütikre! A legtöbb böngésző megjeleníti ezt az információt az aktuális adatvédelmi beállításokkal együtt. Lehet, hogy meg fog lepődni a fellelt neveken, e-levél címeken, jelszavakon és homályos azonosítókon. Remélhetőleg nem fog hitelkártyaszámokat találni, de a visszaélés lehetősége nyilvánvaló.

Néhány felhasználó az összes süti visszautasítására állítja be a böngészőjét, hogy magánéletük háborítatlanságának legalább a látszatát megőrizzék. Ez azonban gondokat okozhat, mert sok webhely nem működik megfelelően sütik nélkül. Alternatív megoldásként sok böngésző lehetővé teszi a felhasználók számára a **harmadik féltől származó (third-party)** sütik blokkolását. A harmadik féltől származó süti nem az éppen letöltött oldalról származik, hanem egy másik webhelyről. Például az *alattomos.com* sütije, aminek használatára a *P* oldal böngészése közben került sor, egy egész más webhelyen van. Ezeknek a sütiknek a blokkolása segít megelőzni a webhelyek közötti nyomon követést. Böngészőkiterjesztések is telepíthetők a sütik használatának finomhangolása (vagy inkább használatuk mellőzése) érdekében. Ahogy a vita folytatódik, sok vállalat adatvédelmi házirendet dolgoz ki, amely a visszaélések elkerülése érdekében korlátozza azt, hogy hogyan történjen a cégnél az információmegosztás. A házirendek persze csak azt mondják meg, hogy mit mondjanak a vállalatok az információ kezeléséről. Például: „Az Öntől gyűjtött információt csak a saját üzleti tevékenységünkhöz használjuk fel" – ami az információ értékesítése is lehet.

7.3.2. Statikus weboldalak

A web lényege a weboldalak átvitele a kiszolgálótól az ügyfélhez. A weboldalak a legegyszerűbb formájukban statikusak. Ez azt jelenti, hogy állományok tartózkodnak valamilyen kiszolgálón, amelyik minden lekérésük és megjelenítésük alkalmával ugyanúgy adja át őket. Önmagában az, hogy statikusak, még nem jelenti azt, hogy az oldalak mozdulatlanok a böngészőben. Egy videót tartalmazó oldal is lehet statikus weboldal.

Mint korábban említettük, a web közvetítő nyelve, amelyen a legtöbb oldalt írták, a HTML. A tanárok honlapjai általában statikus HTML-oldalak. A vállalati honlapok általában webtervező cégek által összeállított dinamikus oldalak. Ebben a fejezetben a későbbi anyagrészek megalapozásaként röviden áttekintjük a statikus HTML-oldalakat. Azok az olvasók, akik már ismerik a HTML-t, továbbugorhatnak a következő fejezetre, amiben a dinamikus tartalmakat és webszolgáltatásokat (web service) ismertetjük.

HTML – a hipertextjelölő nyelv

A **HTML (HyperText Markup Language – hipertextjelölő nyelv)** a világhálóval együtt mutatkozott be. Lehetővé teszi, hogy a felhasználók weboldalakat állítsanak elő, melyek tartalmazhatnak szöveget, ábrákat, videót, hivatkozásokat más weboldalakra és egyebeket. A HTML jelölőnyelv vagy dokumentumok megformázásának módját leíró nyelv. A „jelölő" (markup) kifejezés azokból az időkből származik, amikor a szerkesztők a nyomtatók számára – melyek akkoriban emberi lények voltak – ténylegesen megjelölték a dokumentumokban, hogy melyik betűtípust kell használni és így tovább. A jelölőnyelvek ezért explicit formázóparancsokat tartalmaznak. A HTML-ben például a a félkövér mód kezdetét, a pedig a végét jelzi. A jelölőnyelvek legtöbb akadémiai szerző számára jól ismert további példái a LaTeX és a TeX.

A jelölőnyelv használatának legfőbb előnye a nem jelölőnyelvekkel szemben, hogy elválasztja a tartalmat attól, ahogyan azt meg kellene jeleníteni. Ezek után már egyszerű megírni egy böngészőt: csupán csak a jelölőparancsokat kell megértenie, majd alkalmazni azokat a tartalomra. Azzal, hogy a jelölőparancsokat beágyazták minden HTML-állományba és szabványosították őket, bármely webböngésző számára lehetővé vált, hogy tetszőleges weboldalt elolvashasson és újraformázhasson. Ez rendkívül fontos, mert lehet, hogy egy oldalt 1600 × 1200-as felbontás és 24 bites színmélység mellett egy csúcskategóriás számítógépen hoztak létre, de egy 640 × 320-as ablakban kell megjeleníteni egy mobiltelefonon.

Bár dokumentumokat kétségkívül bármely egyszerű szövegszerkesztő segítségével is írhatunk (és sokan ezt is teszik), lehetőség van azonban speciális HTML-szerkesztők vagy szövegszerkesztő programok használatára is, melyek elvégzik a munka nagy részét (de cserébe kevesebb beavatkozási lehetőséget adnak a felhasználónak a végeredmény részleteire nézve).

Egy egyszerű, HTML-ben megírt weboldal és annak böngészőben történő megjelenítése látszik a 7.23. ábrán. Minden weboldal fejlécből és törzsből áll, magát az oldalt pedig <html> és </html> formázási parancsok, ún. **címkék (tags)** határolják, bár a legtöbb böngésző nem tekinti hibának ezek hiányát. Amint a 7.23.(a) ábra mutatja, a fejlécet a <head>

```
<html>
<head> <title> Egyesült Szerkentyű Rt. </title> </head>
<body> <h1> Üdvözöljük az ESZ honlapján </h1>
<img src="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <br>
Boldogok vagyunk, hogy Ön ellátogatott az <b> Egyesült Szerkentyű </b> honlapjára.
Reméljük, hogy <i> Ön </i> minden szükséges tudnivalót megtalál itt. <p> Alább
előkapcsokat biztosítunk számos kiváló termékünkhöz. Ön rendelhet elektronikusan
(WWW-n keresztül), telefonon vagy e-levélben. </p>
<hr>
<h2> Termékismertető </h2>
<ul>
  <li> <a href="http://widget.com/products/big"> Nagy szerkentyűk </a> </li>
  <li> <a href="http://widget.com/products/little"> Kis szerkentyűk </a> </li>
</ul>
<h2> Elérhetőségek </h2>
<ul>
  <li> Telefon: 1-800-WIDGETS </li>
  <li> E-levél: info@amalgamated-widget.com </li>
</ul>
</body>
</html>
```

(a)

(b)

7.23. ábra. *(a) Egy fiktív weboldal HTML-je. (b) A formázott oldal*

és </head>, a törzset pedig a <body> és </body> címkék fogják közre. A címkék szövegét **direktívának (directive)** nevezik. A legtöbb (de nem az összes) HTML-címke ilyen formátumú, vagyis a <valami> címke jelzi valaminek a kezdetét, és a </valami> a végét.

A címkék lehetnek kis- és nagybetűsek is, azaz <head> és <HEAD> ugyanazt jelentik, de kompatibilitás szempontjából a kisbetűs írásmód a legjobb. A HTML-forrás formázásának nincs jelentősége. A HTML-értelmezők figyelmen kívül hagyják a fölösleges szóköz és kocsi-vissza karaktereket, hiszen mindig az aktuális megjelenítési területnek megfelelően kell újraformázniuk a szöveget. Ebből következik, hogy az ilyen, ún. white space (puha szóköz) karakterek tetszés szerint használhatók a HTML-forrásokban, hogy azok olvashatóbbak legyenek – legtöbbjük erre bizony rá is szorul. Ugyanezért nem elég egyszerűen üres sorokkal elválasztani a bekezdéseket, mivel azokat az értelmezők nem veszik figyelembe; erre a célra külön címkét definiáltak.

Egyes címkék **attribútumokkal**, azaz (névvel ellátott) paraméterekkel is rendelkeznek. Például a 7.23. ábrán látható <img> címkét egy képnek a soron belüli megjelenítésére használják. Két paramétere van: az src és az alt. Az első attribútum megadja a képhez tartozó URL-t. A HTML-szabvány nem rendelkezik arról, hogy mely képformátumok megengedettek. A gyakorlatban minden böngésző támogatja a GIF- és JPEG-állományokat. A böngészők támogathatnak más formátumokat is, de ez a lehetőség kétélű fegyver. Ha a felhasználó olyan böngészőhöz szokott, amely támogatja, mondjuk, a TIFF-állományokat, akkor esetleg ilyeneket fog rakni saját weboldalaira, és később meg fog lepődni, amikor a többi böngésző egyszerűen figyelmen kívül hagyja gyönyörű műveit.

A második attribútum megadja azt a helyettesítő szöveget, amit akkor kell használni, amikor a képet nem lehet megjeleníteni. A HTML-szabvány minden címkéhez definiálja az esetleges paramétereit, ha van egyáltalán, és azok jelentését. Minden paraméternek saját neve van, így a paraméterek megadásának sorrendje tetszőleges.

Maguk a HTML-dokumentumok az ISO 8859 Latin-1 karakterkészletben íródnak, de escape sorozatok segítségével a csak ASCII-t támogató billentyűzeteken is használhatók speciális karakterek, mint például az è. A szabvány a speciális karakterek listáját is tartalmazza. Ezek escape sorozatainak mindegyike & jellel kezdődik és pontosvesszővel végződik, az   például egy szóközt, az è egy è, míg az é egy é karaktert generál. Mivel a <, > és & karaktereknek speciális jelentésük van, ezeket csak az <, > illetve az & sorozatok segítségével lehet kifejezni.

A fejléc legfontosabb eleme a cím, melyet a <title> és </title> címkék határolnak. Ezenkívül egyéb járulékos információk is szerepelhetnek itt, bár egy sem fordul elő a példánkban. Maga a cím nem jelenik meg az oldalon, a böngészők leginkább az oldalt tartalmazó ablakot címkézik meg vele.

A 7.23. ábrán több címsor használatos. Minden címsort a <h*n*> címkék állítottak elő, ahol *n* egy 1 és 6 közé eső számot jelent. A <h1> a legfontosabb címsor, a <h6> a legkevésbé fontos. Ezek után a böngészőre van bízva, hogy a címsorokat ténylegesen hogyan jeleníti meg. A kisebb számmal jelzett címsorok tipikusan nagyobb és vastagabb betűtípussal jelennek meg, de a böngésző akár különböző színeket is rendelhet a különböző szintű címsorokhoz. A <h1> címsorok általában nagy, félkövér betűtípussal szerepelnek, alattuk és felettük legalább egy üres sorral. A <h2> szintű címsorok ugyanakkor kisebb méretű betűtípust használnak, és kisebb helyet is hagynak a címsor alatt és fölött.

A <b> és <i> címkék a félkövér (boldface) és dőlt betűs (italics) módba történő váltáshoz használatosak. A <hr> címke kikényszerít egy sortörést és vízszintes vonalat húz a kijelzőn.

A címke új bekezdést kezd. A böngésző ezt például egy üres sor beszúrásával és némi behúzással jelenítheti meg. Érdekes módon a címkét, ami a bekezdés végének megjelölésére szolgál, a lusta HTML-programozók gyakran elhagyják.

A HTML különféle mechanizmusokat biztosít listák, sőt akár egymásba ágyazott listák létrehozására is. A sorrend nélküli listák, mint amilyenek a 7.23. ábrán is láthatók, <ul> címkével kezdődnek, a listaelemeket pedig az <li> jelzi. Létezik egy <ol> címke is, a sorszámozott listák indításához. A sorrend nélküli listákban az egyes elemek előtt gyakran pontok (•) jelennek meg. A sorszámozott listák elemeit a böngésző számozza meg.

Végül elérkeztünk a hiperhivatkozásokhoz. Erre látunk példákat a 7.23. ábrán, melyek az <a> (anchor, horgony) és címkéket használják. Az <a>-nak számos paramétere van, melyek közül a legfontosabb a *href*, ami a hivatkozott URL-t tartalmazza. Az <a> és közötti szöveg megjelenítésre kerül. Ha rákattintunk, akkor egy új oldalra jutunk el. Más elemekkel történő hivatkozás is megengedett. Például meg lehet adni egy képet az <a> és címkék között az <img> használatával. Ebben az esetben a kép jelenik meg, és az erre való kattintás aktiválja a hiperhivatkozást.

Számos más HTML-címke és attribútum létezik, amelyeket nem láttunk ebben az egyszerű mintában. Például az <a> címke kaphat egy *name* paramétert is, melynek segítségével beültet egy hiperhivatkozást magába az oldalba. Ez lehetővé teszi például, hogy egy hiperhivatkozás az oldal közepére mutasson. Ez azoknak a weboldalaknak hasznos, amelyek olyan tartalomjegyzékkel kezdődnek, amelynek elemeire rá lehet kattintani. A tartalomjegyzék egy elemére kattintva a felhasználó ugyanannak az oldalnak a megfelelő részére fog ugrani. Egy eltérő címkére példa a
. Ez a böngészőt a sortörésre és egy új sor megkezdésére kényszeríti.

Valószínűleg az a legjobb módja a címkék megértésének, ha működés közben látjuk őket. Ehhez keressen egy weboldalt és nézze meg a HTML-t a böngészőjében, hogy lássa, hogyan állították össze az oldalt. A böngészők többségében található egy FORRÁS MEGJELENÍTÉSE (VIEW SOURCE) vagy hasonló nevű menüpont. Ezt kiválasztva az aktuális oldal tényleges, formázott képe helyett annak HTML-forrását tekinthetjük meg.

Felvázoltuk azokat a címkéket, amelyek a web kezdete óta léteznek. A HTML folyamatosan fejlődik. A 7.24. ábra bemutat néhány tulajdonságot, amelyeket a HTML egymást követő verzióiban vezettek be. A HTML 1.0 a HTML-nek arra a verziójára utal, amit a világháló megjelenésekor használtak. A HTML 2.0, 3.0 és 4.0 verziói néhány éven belül, egymást gyorsan követve jelentek meg, a web robbanásszerű fejlődésével. A HTML 4.0 után közel tíz év telt el, mire a következő fő verzió, a HTML 5.0 szabványosításához vezető út világossá vált. Mivel ez egy nagyobb frissítés, amely egységesíti azokat a módszereket, amelyekkel a böngészők a gazdag tartalmakat kezelik, a HTML 5.0 kidolgozása folyamatban van, és nem várható, hogy a szabvány 2012 előtt megszületik. A szabványok dacára a fontosabb böngészők már támogatják a HTML 5.0 funkcióit.

A HTML-verziók fejlődése az olyan új képességekkel történő bővülésről szól, amelyeket az emberek igényeltek, de a szabvánnyá válás előtt nem szabványos módon (például beépülő modulokkal) kellett megoldani a problémáikat. Például a HTML 1.0 és 2.0 verziókban még nem voltak táblázatok. Ezek a HTML 3.0-ban jelentek meg. Egy

Elem	HTML 1.0	HTML 2.0	HTML 3.0	HTML 4.0	HTML 5.0
Hiperhivatkozások	+	+	+	+	+
Képek	+	+	+	+	+
Listák	+	+	+	+	+
Aktív térképek és képek		+	+	+	+
Űrlapok		+	+	+	+
Egyenletek			+	+	+
Eszköztárak			+	+	+
Táblázatok			+	+	+
Hozzáférési lehetőségek				+	+
Objektumok beágyazása				+	+
Stíluslapok				+	+
Szkriptek				+	+
Videók és hangok					+
Vektorgrafika a szövegben					+
XML megjelenítés					+
Háttérben futó szálak					+
Böngésző által tárolt adatok					+
Rajzolás a vászonra					+

7.24. ábra. *Néhány eltérés a HTML-verziók között*

HTML-táblázat egy vagy több sorból áll, a sorok pedig egy vagy több **cellából** (**table cell**), melyek sok mindent tartalmazhatnak (például szöveget, képeket, további táblázatokat). A HTML 3.0 előtt azok a szerzők, akiknek táblázatra volt szükségük, valamilyen ad-hoc módszerhez kellett folyamodniuk, például beillesztettek egy táblázatot tartalmazó képet.

A HTML 4.0 további lehetőségeket tartalmazott. Ezek között van olyan, amely a hátrányos helyzetű felhasználók számára biztosít hozzáférési lehetőséget, amely objektumbeágyazást tesz lehetővé (ez az <img> címke általánosítása úgy, hogy más objektumok is beágyazhatók legyenek az oldalakba), amely támogatja a szkripteket (megengedve a dinamikus tartalmat). További lehetőségek is vannak.

A HTML 5.0 számos képességgel rendelkezik azoknak a gazdag médiumoknak a kezeléséhez, amelyeket ma rendszeresen használnak a világhálón. Videót és hangot lehet beágyazni az oldalakba, és lejátszatni azokat a böngészővel anélkül, hogy a felhasználónak beépülő modulokat kellene telepítenie. Az ábrákat a böngészőben lehet létrehozni

vektorgrafikaként, a bittérképes képformátumok (mint a JPEG és a GIF) használata helyett. A parancsfájlok futtatását is jobban támogatja, például a háttérben futó számítási szálakkal és a háttértárolóhoz történő hozzáféréssel. Ezek a szolgáltatások segítenek azoknak a weboldalaknak a támogatásában, amelyek sokkal inkább hasonlítanak felhasználói felülettel rendelkező hagyományos alkalmazásokra, mint dokumentumokra. A web ebbe az irányba halad.

Adatbevitel és űrlapok

Van egy fontos képesség, amit eddig nem tárgyaltunk meg: az adatbevitel. A HTML 1.0 alapvetően egyirányú volt. A felhasználók lekérhettek oldalakat a tartalomszolgáltatóktól, de bonyolult volt a másik irányba információt visszaküldeni. Gyorsan nyilvánvalóvá vált, hogy szükség van a kétirányú forgalomra a világhálón elhelyezett termékekre történő rendelések felvételéhez, regisztrációs kártyák online kitöltéséhez, a megadott kifejezés megkereséséhez és még sok minden máshoz.

A felhasználó által megadott adat elküldése (a böngészőn keresztül) a kiszolgálóhoz kétfajta segítséget igényel. Először is arra van szükség, hogy a HTTP legyen képes adatokat szállítani abban az irányban. Egy későbbi szakaszban leírjuk ennek a módját; a HTTP a *POST* metódust használja. A második követelmény, hogy legyen képes olyan felhasználói csatlakozó felületi elemek megjelenítésére, amelyek összegyűjtik és becsomagolják a betáplált adatokat. Az **űrlapok** ezzel a funkcionalitással a HTML 2.0-ban jelentek meg.

Az űrlapok dobozokat vagy gombokat tartalmazhatnak, amelyek lehetővé teszik a felhasználók számára, hogy információt írhassanak be, vagy a felkínált lehetőségek közül választhassanak, majd az információt visszaküldhessék az oldal tulajdonosának. Az űrlapokat éppen úgy írják meg, mint a HTML más részeit, ahogyan az a 7.25. ábra példáján látható. Megjegyezzük, hogy az űrlapok még mindig statikus tartalmak. Ugyanúgy viselkednek, függetlenül attól, hogy ki használja őket. A dinamikus tartalom, amelyet később tárgyalunk, sokkal kifinomultabb módokat kínál a bevitt adatok összegyűjtésére egy olyan program segítségével, amelynek viselkedése a böngésző környezetétől függhet.

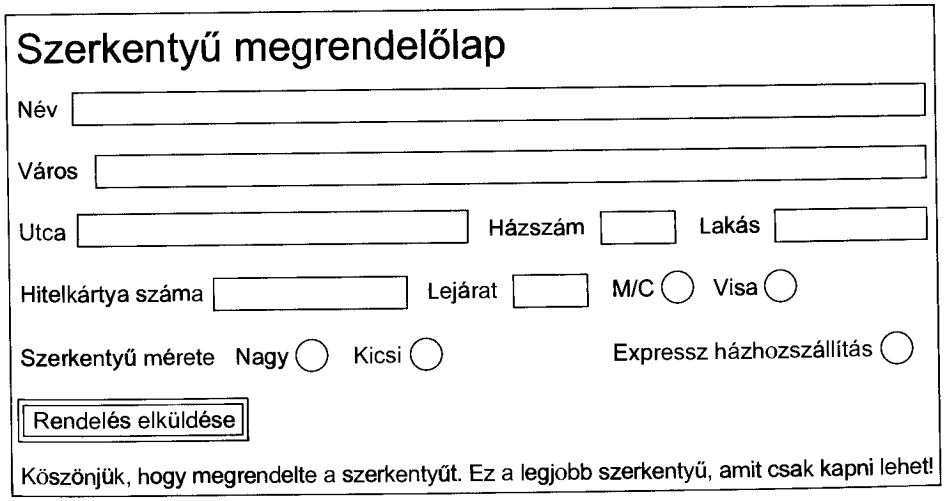
Mint minden űrlapot, ezt is a <form> és </form> címkék fogják közre. Ennek a címkének az attribútumai meghatározzák, hogy mit kell tenni a bevitt adatokkal. Ebben az esetben a POST metódus használatával el kell küldeni az adatokat az URL-lel meghatározott címre. A címkébe nem zárt szöveg egyszerűen megjelenik. Minden szokásos címke (mint például a <b>) megengedett az űrlapon belül, hogy az oldal készítője vezérelhesse az űrlapnak a képernyőn történő megjelenését.

Ezen az űrlapon háromfajta bemeneti dobozt használtunk, melyek mindegyike az <input> címkét használja. Ennek sokfajta paramétere van, amelyek meghatározzák a megjelenített doboz méretét, természetét és használatának módját. Ennek legközönségesebb formái a felhasználó szövegét befogadó üres mezők, a kipipálható dobozok, és a *submit* (elküld) gombok, amelyek az adatoknak a kiszolgálóra történő visszaküldését idézik elő.

Az első fajta bemeneti doboz a „Név" szöveg után következő *szöveg* (text) típusú doboz. A doboz 46 karakter hosszú, és a felhasználótól egy szöveget várunk bele, amelyet azután a *customer* változóban tárolunk el.


```
<html>
<head> <title> SZERKENTYŰ MEGRENDELŐLAP </title> </head>
</body>
<h1> Szerkentyű megrendelőlap </h1>
<form ACTION="http://widget.com/cgi-bin/order.cgi" method=POST>
<p> Név <input name="customer" size=46> </p>
<p> Város <input name="city" size=46> </p>
<p> Utca <input name="street" size=18> Házszám <input name="house" size=7>
Lakás <input name="apt" size=10> </p>
<p> Hitelkártya száma <input name="cardno" size=10>
Lejárat <input name="expires" size=5>
M/C <input name="cc" type=radio value="mastercard">
Visa <input name="cc" type=radio value="visacard"> </p>
<p> Szerkentyű mérete Nagy <input name="product" type=radio value="expensive"
Kicsi <input name="product" type=radio value="cheap">
Expressz házhozszállítás <input name="express" type=checkbox> </p>
<p> <input type=submit value="Rendelés elküldése" </p>
Köszönjük, hogy megrendelte a szerkentyűt. Ez a legjobb szerkentyű, amit csak kapni lehet!
</form>
</body>
</html>
```

(a)



(b)

7.25. ábra. *(a) Egy megrendelőlap HTML-je. (b) A megformázott oldal*

Az űrlap következő sora a felhasználó városának nevét kérdezi, 46 oszlop szélesen. Az ezután következő sor az utcára, a házszámra és a lakásszámra kérdez rá. Mivel nem használtunk <p> címkéket a mezők között, így a böngésző mindet egy sorban fogja megjeleníteni (ahelyett, hogy külön bekezdésekbe helyezné azokat), ha elférnek. A böngésző szemében ez a bekezdés csak hat elemet tartalmaz: három karakterláncot és három dobozt, váltakozva. A következő sor a hitelkártya számát és lejáratának dátumát kérdezi

meg. A hitelkártyaszámokat csak megfelelő biztonsági intézkedések megtétele után lenne szabad átvinni az interneten. Ezek közül néhányat a 8. fejezetben fogunk tárgyalni.

A lejárat dátuma után új funkcióval találkozunk: a rádiógombokkal (radio buttons). Ezeket akkor használják, amikor két vagy több lehetőség között kell választani. Az elméleti modell itt egy autórádió volt, melynek fél tucat gombja van az állomások kiválasztására. Ha az egyik gombra rákattintanak, az az ugyanabban a csoportban levő összes többit kikapcsolja. A vizuális megjelenítés a böngészőtől függ. A szerkentyű méretének megadására is két rádiógombot használtunk. A két csoportot a *name* paramétereikkel különböztetjük meg, nem pedig olyan statikus címkékkel, mint mondjuk a <radiobutton> és a </radiobutton>.

A *value* paramétereket a lenyomott rádiógomb jelzésére használják. Például attól függően, hogy a felhasználó melyik hitelkártya-változatot választotta, a *cc* változó értéke vagy a „visacard", vagy a „mastercard" karakterláncra fog beállni.

A két rádiógombkészlet után eljutottunk a szállítási lehetőségekhez, amelyet egy ellenőrző gomb, *checkbox* típusú négyzet (jelölőnégyzet) jelképez. Ez lehet ki- vagy bekapcsolva. A rádiógomboktól eltérően, amelyeknél a halmazból pontosan egyet kell kiválasztani, minden *checkbox* típusú doboz ki- vagy bekapcsolt állapotban lehet, az összes többitől függetlenül.

Végül elérkeztünk a *submit* (küldés) gombhoz. A *value* (érték) karakterlánc a gomb címkéje, és ez jelenik meg. Amikor a felhasználó a *Rendelés elküldése* gombra kattint, a böngésző egyetlen hosszú sorba csomagolja az összegyűlt információt, és visszaküldi a kiszolgálónak arra az URL-re, amit a <form> címke részeként adtak meg. A böngésző egy egyszerű kódolást használ. A mezők elválasztására az & szolgál, a szóközt pedig a + jelképezi. Példánk űrlapjában ez a sor a 7.26. ábrán láthatóhoz hasonló lehet.

```
customer=Kiss+Tibor&city=Budapest+III.+ker.&street=Mester+utca&
house=38/A&apt=VII.+em.+78.&cardno=1234567890&expires=06/14&cc=mastercard&
product=cheap&express=on
```

7.26. ábra. *A böngésző egy lehetséges válasza a kiszolgálónak a felhasználó által megadott információval*

Ezt a karakterláncot egyetlen sorként küldik vissza a kiszolgálónak. (Itt azért törtük három sorba, mert nem elég széles az oldal.) A karakterlánc értelmezése a kiszolgálóra van bízva; nagy valószínűséggel átadja az információt egy programnak, ami feldolgozza azt. Ennek lehetséges módjait a következő szakaszban tárgyaljuk.

Másfajta beviteli mezők is léteznek, melyeket nem mutattunk be ebben az egyszerű példában. Két ilyen típus a *password* (jelszó) és a *textarea* (szöveges terület). Egy *password* doboz ugyanolyan, mint egy *text* doboz (az alapértelmezett típus, melyet nem szükséges megnevezni), kivéve, hogy a begépelt karakterek nem jelennek meg. A *textarea* doboz megegyezik a *text* dobozzal, csak több sort is tartalmazhat.

Hosszú listákhoz, amelyekből egy elemet kell kiválasztani, létrehozták a <select> és </select> címkéket, amely címkék között a választási lehetőségek felsorolása található. Ez a felsorolás gyakran legördülő menü formájában jelenik meg. A szemantika olyan, mint a rádiógomboké, hacsak nincs megadva a *multiple* (többszörös) paraméter, amelynél a szemantika megegyezik a jelölőnégyzetek szemantikájával.

Végül, arra is van mód, hogy jelezzük az alapértelmezett vagy kezdeti értékeket, amelyeket a felhasználó megváltoztathat. Például, ha a *text* dobozt ellátjuk a *value* mezővel, annak tartalma megjelenik a felhasználó számára az űrlapon, hogy szerkeszthesse vagy törölje azt.

CSS – egymásba ágyazható stíluslapok

A HTML eredeti célja a dokumentum *szerkezetének*, nem pedig a *megjelenésének* meghatározása volt. Például a

```
<h1> Deborah fényképei </h1>
```

parancs arra utasítja a böngészőt, hogy emelje ki a címsort, de nem mond semmit a betűtípusról, a betűméretről vagy a színről. Ezek a böngészőre vannak bízva, hiszen az ismeri a képernyő tulajdonságait (például hogy hány képpontból áll). Sok webtervező azonban teljesen uralni akarta az oldalaik megjelenítését, ezért új címkéket adtak a HTML-hez, hogy befolyásolni lehessen a megjelenítést, mint például az alábbi címkével:

```
<font face="helvetica" size="24" color="red"> Deborah fényképei </font>
```

Emellett lehetővé vált a képernyőterületen való elhelyezkedés pontos megadása is. Ezzel a megközelítéssel az a baj, hogy fárasztó és nagyra duzzadt, nem hordozható HTML-t eredményez. Az oldal ugyan tökéletesen jelenhet meg abban a böngészőben, amelyben kifejlesztették, de teljes lehet a káosz egy másik böngészőben vagy ugyanannak a böngészőnek egy másik kiadásában, illetve más képernyőfelbontásnál.

Jobb választás a stíluslapok (style sheets) használata. A szövegszerkesztőkben lévő stíluslapok lehetővé teszik a szerzők számára, hogy a szöveghez logikai stílust rendeljenek fizikai stílus helyett, például „első bekezdés"-t a „dőlt betűs szöveg" helyett. Minden egyes stílus megjelenését önállóan határozzák meg. Ily módon, ha a szerző úgy dönt, hogy a 14 pontos, kék és dőlt betűs első bekezdéseket 18 pontos, félkövér, megdöbbentő rózsaszínűre változtatja, mindössze egyetlen definíciót kell megváltoztatnia az egész dokumentum átalakításához.

A CSS (**Cascading Style Sheets – egymásba ágyazható stíluslapok**) a HTML 4.0-val vezette be a stíluslapokat a világhálóra, azonban a széles körű használat és a böngészők támogatása nem kezdődött meg 2000 előtt. A CSS egy egyszerű nyelvet határoz meg a címkézett tartalom megjelenését vezérlő szabályok leírásához. Nézzünk meg egy példát! Tételezzük fel, hogy az Egyesült Szerkentyű Rt. menő oldalakat szeretne létrehozni, tengerészkék Arial betűtípusú szöveggel a piszkosfehér háttéren, különböző szintű címsorokkal, amelyek 100%-kal és 50%-kal nagyobbak, mint a szöveg. A 7.27. ábra CSS-definíciója megadja ezeket a szabályokat.

```
body {background-color:linen; color:navy; font-family:Arial;}
h1 {font-size:200%;}
h2 {font-size:150%;}
```

7.27. ábra. *Egy CSS-példa*

```
<head>
<title> Egyesült Szerkentyű Rt. </title>
<link rel="stylesheet" type="text/css" href="awistyle.css" />
</head>
```

7.28. ábra. *Egy CSS-stíluslap beágyazása*

Mint látható, a stílusok meghatározása tömör lehet. Minden sor kiválaszt egy elemet, amelyre vonatkozik, és megadja a tulajdonságok értékeit. Az elem tulajdonságai alapértelmezés szerint minden olyan HTML-elemre is vonatkoznak, amelyeket az tartalmaz. Tehát a body stílusa meghatározza a dokumentum törzsében lévő bekezdések szövegének stílusát. Léteznek kényelmes rövidítések a színekre (például red a vörös színhez). Valamennyi olyan stílusparaméternek, amit nem határoztunk meg, a böngésző ad alapértelmezett értéket. Ez a viselkedés választhatóvá teszi a stíluslap definíciókat, de nélkülük is meg fognak jelenni az oldalak valamilyen értelmes módon.

A stíluslapok beágyazhatók a HTML-állományokba (például a <style> címke használatával), de sokkal gyakrabban helyezik őket egy különálló állományba és hivatkoznak rájuk. Például az Egyesült Szerkentyű oldalának <head> címkéje úgy módosítható, hogy a 7.28. ábrán látható módon hivatkozzon az *awistyle.css* állományban lévő stíluslapra. A példa a CSS-fájlok MIME-típusát is mutatja, ami *text/css*.

Ennek a stratégiának két előnye van. Először is, lehetővé teszi, hogy a stílusoknak egy halmazát egy webhely több oldalán is felhasználjuk. Ez a szervezés egységes megjelenést kölcsönöz az oldalaknak még akkor is, ha különböző szerzők különböző időpontokban fejlesztették, és lehetővé teszi az egész webhely megjelenésének módosítását egyetlen CSS-állomány, és nem a HTML szerkesztésével. Ezt a módszert egy C program #include állományához hasonlíthatjuk: ha ott megváltoztatunk egy makrodefiníciót, akkor az megváltozik az erre hivatkozó összes állományban is. A másik előny az, hogy a letöltött HTML-állományok kicsik maradhatnak. Ennek az az oka, hogy a böngésző a CSS-állománynak egyetlen másolatát tölti le az összes olyan oldal számára, amelyek hivatkoznak rá. Nem szükséges minden egyes weboldallal együtt egy új másolatot is letöltenie.

7.3.3. Dinamikus weboldalak és webalkalmazások

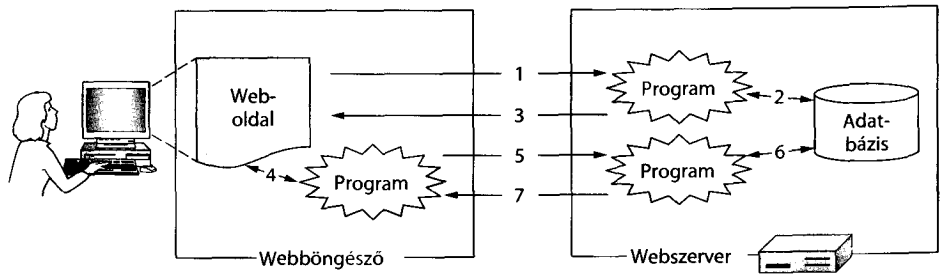
A statikus weboldalak modellje, amelyet eddig használtunk, a weboldalakat könnyen elérhető, egymással összekapcsolt multimédia dokumentumokként kezeli. Ez a modell megfelelő volt a világháló kezdeti időszakában, amikor hatalmas mennyiségű információt tettek online elérhetővé. Manapság a világháló körüli izgalmakat nagyrészt a webalkalmazások és webszolgáltatások okozzák. Ilyen példák többek között: e-kereskedelmi helyeken termékek vásárlása, könyvtári katalógusokban történő keresés, térképek tanulmányozása, e-levelek olvasása és írása, valamint közös munkavégzés dokumentumokon.

Ezek az új felhasználási területek hasonlítanak a hagyományos alkalmazói szoftverre (például levélolvasókra és szövegszerkesztőkre). A különbség az, hogy ezek az alkalmazások a böngészőben futnak, az internetes adatközpontokban lévő kiszolgálókon tárolt felhasználói adatokkal. Webes protokollokat használnak az információ interneten keresztül történő eléréséhez, és böngészőt a felhasználói felület megjelenítéséhez. Ennek a megközelítésnek az az előnye, hogy a felhasználóknak nem kell különálló alkalmazói programokat telepítenie, a felhasználói adatok különböző számítógépekről is elérhetők, és azokról a szolgáltatás üzemeltetője biztonsági másolatokat készít. Ez annyira sikeresnek bizonyult, hogy a hagyományos alkalmazói szoftverekkel vetekszik. Természetesen az a tény, hogy ezeket az alkalmazásokat a nagy szolgáltatók ingyenesen biztosítják, sokat segít. Ez a modell a **számítási felhő** (**cloud computing**) elterjedt formája, melyben a számítások elvégzésének helye az egyéni asztali számítógépekből az interneten lévő megosztott szerverfürtökbe helyeződik át.

Ha a weboldalak alkalmazásként funkcionálnak, akkor már nem lehetnek többé statikusak. Dinamikus tartalomra van szükség. Például a könyvtári katalógus oldalának tükröznie kell, melyek a jelenleg elérhető könyvek és melyeket jegyeztek elő, és így nem elérhetők. Hasonlóképpen, egy hasznos értéktőzsdei weblap lehetővé tenné a felhasználónak, hogy együttműködésbe lépjen az oldallal abból a célból, hogy megnézhesse különféle időtávokon a részvényárfolyamok alakulását, és kiszámíthassa nyereségeit és veszteségeit. Ahogyan ezek a példák sugallják, a dinamikus tartalom előállítása a kiszolgálón vagy a böngészőben (esetleg mindkét helyen) futó programokkal lehetséges.

Ebben a szakaszban egymás után mindkét esetet meg fogjuk vizsgálni. Az általános helyzetet a 7.29. ábra mutatja. Képzeljünk el például egy térképszolgáltatást, ami lehetővé teszi, hogy a felhasználó megadjon egy utcanevet, majd a szolgáltatás megjeleníti a helynek megfelelő térképet. A helyre vonatkozó kérés megadása után a webszervernek egy programot kell használnia annak a weboldalnak az előállításához, amely az utcák adatbázisa alapján megjeleníti a hely térképét és más földrajzi információt. Ez a tevékenység látható az 1-3 lépéseken. A kérés (1. lépés) egy program futását okozza a kiszolgálón. A program a megfelelő oldal létrehozása érdekében konzultál az adatbázissal (2. lépés), és azt az oldalt elküldi a böngészőnek (3. lépés).

A dinamikus tartalom azonban több ennél. A visszaküldött oldal maga is tartalmazhat programokat, amelyek a böngészőben futnak. Térképes példánkban a program lehetővé tehetné a felhasználónak, hogy útvonalakat találjon meg, és különböző részletességgel megvizsgálja a környező területeket. A program frissíthetné a weboldalt, a felhasználó utasításainak megfelelően nagyíthatná vagy kicsinyíthetné azt (4. lépés). Bizonyos interaktív tevékenységek lekezeléséhez a programnak több adatra lehet szüksége a szervertől. Ebben az esetben a program kéréssel fordul a kiszolgálóhoz (5. lépés), amely további információt vesz ki az adatbázisból (6. lépés) és azt visszaküldi a válaszban (7. lépés). A program ezután folytatja az oldal frissítését (4. lépés). A kérések és válaszok a háttér-



7.29. ábra. *Dinamikus oldalak működése*

ben történnek. A felhasználó akár még csak nem is tud ezekről, mert az oldal URL-je és címe rendszerint nem változik meg. Az ügyféloldali programok telepítésével a weboldal sokkal szimpatikusabb felhasználói felületet valósíthat meg, mint csak a kiszolgálóoldali programokkal.

Dinamikus weboldalak előállítása a kiszolgáló oldalán

Nézzük meg a kiszolgálóoldali tartalom-előállítás esetét egy kicsit részletesebben! Egy egyszerű helyzet, amelyben szerveroldali feldolgozásra van szükség, az űrlapok használata. Tegyük fel, hogy a felhasználó kitöltötte a 7.25.(b) ábra szerkentyű megrendelő űrlapját, és rákattintott a *Rendelés elküldése* gombra. Amikor a felhasználó kattint, az űrlappal meghatározott URL-en lévő kiszolgálónak egy kérés megy a felhasználó által kitöltött űrlap tartalmával együtt (ebben az esetben egy *POST* metódussal a *http://widget.com/cgi-bin/order.cgi* URL-re). Ezeket az adatokat át kell adni feldolgozásra egy programnak vagy szkriptnek. Tehát az URL azonosítja a futtatandó programot, az adatok pedig a program bemenetére kerülnek. Ebben az esetben a feldolgozás magába foglalná a rendelésnek a cég belső rendszerébe léptetését, az ügyfélnyilvántartás frissítését és a hitelkártya megterhelését. A kérésre válaszként kapott weboldal attól függ, hogy mi történt a feldolgozás során. Ez nem fix, mindig ugyanaz, mint egy statikus oldalnál. Ha a megrendelés sikeres, a visszaküldött oldal megadhatja a kiszállítás várható időpontját. Ha sikertelen, a visszaküldött oldal tájékoztathat arról, hogy a kért szerkentyűk nincsenek raktáron, vagy a hitelkártya valamilyen okból nem volt érvényes.

Az, hogy a kiszolgáló miért egy programot futtat ahelyett, hogy egy állományt keresne meg és küldene vissza, a szerver kialakításától függ. Ezt nem maguk a webes protokollok határozzák meg. Ennek az az oka, hogy a felhasználói csatlakozófelület egyedi kialakítású, és a böngészőnek nem szükséges ismernie a részleteket. Ami a böngészőt illeti, az egyszerűen csak egy kérés összeállítását és egy oldal lekérését végzi.

Ennek ellenére szabványos API-kat dolgoztak ki a webszerverek számára a programok meghívásához. Ezeknek a csatlakozófelületeknek a léte megkönnyíti a fejlesztők dolgát akkor, amikor a különféle szervereket webalkalmazásokkal egészítik ki. Röviden áttekintünk két API-t, hogy legyen némi elképzelésünk ezekről, hogy miket is tartalmaznak.

Az első API egy dinamikus oldallekérések kezeléséhez használható módszer, mely a web kezdete óta elérhető. Ezt **CGI**-nek (**Common Gateway Interface – általános átjáró interfész**) nevezik, és az RFC 3875 ismerteti. A CGI egy csatlakozófelületet, interfészt biztosít, mely lehetővé teszi, hogy a webszerverek olyan kiszolgálóoldali programokkal és szkriptekkel beszéljenek, melyek valamilyen bemenetet fogadnak el (például űrlapokból), és válaszul HTML-oldalakat állítanak elő. Ezeket a programokat valamilyen fejlesztőknek kényelmes nyelven írhatják, a fejlesztés könnyebbsége érdekében általában egy szkriptnyelven. Ez lehet Python, Ruby, Perl vagy az Ön kedvenc nyelve.

A CGI által meghívott programok szokás szerint egy *cgi-bin* nevű könyvtárban helyezkednek el, amely az URL-ben is látszik. A kiszolgáló leképezi az erre a könyvtárra vonatkozó kéréseket egy programnévre, és végrehajtja ezt a programot egy különálló folyamatként. A kéréssel együtt elküldött minden adatot bemenetként átadja a programnak. A program kimenete egy weblap, amit a kiszolgáló visszaküld a böngészőnek.

Példánkban az *order.cgi* programot a 7.26. ábrán látható módon kódolt űrlapról vett bemenettel a kiszolgáló hívja meg. A program elemzi a paramétereket és feldolgozza a rendelést. Egy hasznos megállapodás, hogy a program visszaküldi a megrendelő űrlap HTML-jét, ha bemenetként nem kapott bemeneti adatokat űrlapról. Így a program biztos lehet abban, hogy ismeri az űrlap megjelenési formáját.

A második API, amit szemügyre veszünk, az előzőtől eléggé eltérő. Itt az a megközelítés, hogy kis szkripteket ágyaznak be a HTML-oldalakba, melyeket maga a kiszolgáló futtat a weboldal előállítása érdekében. A **PHP (PHP: Hypertext Preprocessor – hipertext előfeldolgozó)** egy népszerű nyelv, mely megfelelő az ilyen szkriptek megírásához. Használatához a kiszolgálónak értenie kell a PHP-t csakúgy, mint ahogy a böngészőnek is értenie kell az CSS-t, hogy értelmezni tudja a stíluslapokat tartalmazó weboldalakat. A kiszolgálók a PHP-t tartalmazó weboldalakat általában az állomány kiterjesztése alapján azonosítják, ami *php* és nem *html* vagy *htm*.

A PHP-t egyszerűbb használni, mint a CGI-t. A 7.30.(a) ábra az űrlapok PHP-val történő kezelésére mutat egy példát. Az ábra felső részén egy szokványos HTML-oldal látható, benne egy egyszerű űrlappal. A <form> címke ezúttal az *action.php* állományt adja meg; ezt kell meghívni a paraméterek kezeléséhez, amikor a felhasználó beküldi az űrlapot. Az oldal két szövegdobozt jelenít meg: az egyik a nevet, a másik az életkort kéri.

```
<html>
<body>
<form action="action.php" method="post">
<p> Kérem, adja meg a nevét: <input type="text" name="name"> </p>
<p> Kérem, adja meg az életkorát: <input type="text" name="age"> </p>
<input type="submit">
</form>
</body>
</html>
```

(a)

```
<html>
<body>
<h1> Válasz: </h1>
Helló <?php echo $name; ?>
Jóslat: jövőre <?php echo $age + 1; ?> éves leszel.
</body>
</html>
```

(b)

```
<html>
<body>
<h1> Válasz: </h1>
Helló Barbara.
Jóslat: jövőre 33 éves leszel.
</body>
</html>
```

(c)

7.30. ábra. *(a) Egy űrlapot tartalmazó weboldal. (b) PHP-szkript az űrlap adatainak kezelésére. (c) A PHP-szkript kimenete, ha a bemenet „Barbara", illetve „32" volt*

Miután ezeket kitöltötték, és az űrlapot beküldték, a kiszolgáló elemzi a 7.26. ábrához hasonló visszaküldött karakterláncot, és a nevet a *name*, a kort pedig az *age* változóba teszi. Ezután válaszként megkezdi a 7.30.(b) ábrán látható *action.php* állomány feldolgozását. Az állomány feldolgozása során a PHP-parancsok végrehajtásra kerülnek. Ha a felhasználó azt írta a dobozokba, hogy „Barbara" és „32", akkor a visszaküldött HTML-oldal a 7.30.(c) ábrán látható lesz. Az űrlapok kezelése tehát rendkívül egyszerű lesz a PHP használatával.

Annak ellenére, hogy a PHP-t könnyű használni, valójában egy nagyon hatékony programozási nyelv, mely a web és a kiszolgálóoldali adatbázis közötti kapcsolatot hivatott megteremteni. Vannak benne változók, karakterláncok, tömbök, tartalmazza a C-ben megtalálható vezérlési szerkezetek többségét, és ezenfelül egy, a puszta *printf*-nél sokkal hatékonyabb I/O-t. A PHP nyílt forráskódú, szabadon hozzáférhető és széles körben használják. A PHP-t kifejezetten úgy tervezték, hogy jól működjön az Apache-csal, amely a világ legelterjedtebb webszervere, és szintén nyílt forráskódú. A PHP-ről többet is megtudhatunk Valade [2009] munkájából.

Már két különböző módot is láttunk dinamikus HTML-oldalak előállítására: a CGI-szkripteket és a beágyazott PHP-t. Számos más megoldás is létezik, amelyek közül választani lehet. A **JSP (JavaServer Pages – Java kiszolgáló oldalak)** hasonlít a PHP-hez, azzal a különbséggel, hogy a dinamikus rész PHP helyett a Java programozási nyelven íródik. Azoknak az oldalaknak, melyek ezt az eljárást használják, *.jsp* a kiterjesztésük. Az **ASP.NET (Active Server Pages .NET – aktív kiszolgálóoldalak)** nem más, mint a PHP vagy a JSP Microsoft-féle változata. Ezek olyan programok, amelyek a Microsoft saját .NET hálózatos alkalmazási keretrendszerét használják a dinamikus tartalom előállítására. Az ilyen módszerrel készült oldalakhoz az *.aspx* kiterjesztést használják. E három technika közti választás általában inkább politikai (nyílt forrás vagy Microsoft), mint technikai kérdés, mivel a három nyelv nagyjából egyenértékű.

Dinamikus weboldalak előállítása az ügyfél oldalán

A PHP- és CGI-szkriptek megoldják a kiszogáló oldalán az adatbevitel kezelésének és az adatbázisok elérésének problémáját. Ezek a szkriptek képesek az űrlapokból beérkező információ fogadására, információ kikeresésére egy vagy több adatbázisból, valamint az eredményeket tartalmazó HTML-oldalak előállítására. Egyikük sem képes azonban egérműveletekre reagálni vagy a felhasználókkal közvetlenül együttműködni. Ilyen célok érdekében olyan szkripteket kell beágyazni a HTML-oldalakba, amelyek végrehajtása nem a kiszolgáló, hanem az ügyfél oldalán történik. A HTML 4.0-tól kezdődően vált lehetővé az ilyen szkriptek használata, a <script> címke segítségével. Azokat a módszereket, amelyeket ezeknek az interaktív weblapoknak a létrehozására használnak, **dinamikus HTML-ként** említik.

A legnépszerűbb ügyféloldali szkriptnyelv a **JavaScript**, melyet most röviden be is mutatunk. A nevek közötti hasonlóság ellenére a JavaScriptnek szinte semmi köze sincs a Java programozási nyelvhez. A többi szkriptnyelvhez hasonlóan ez is nagyon magas szintű nyelv. Egyetlen JavaScript-sorral megoldható például az, hogy földobjunk egy dialógusablakot, megvárjuk a szöveges bevitelt, majd az eredményül kapott karakterlán-

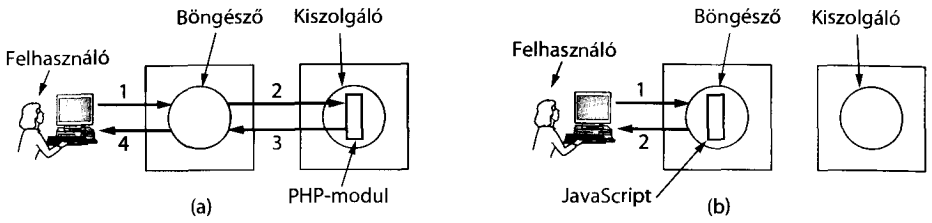

```
<html>
<head>
<script language="javascript" type="text/javascript">
function valasz(test_form) {
   var szemely = test_form.nev.value;
   var evek = eval(test_form.kor.value) + 1;
   document.open();
   document.writeln("<html> <body>");
   document.writeln("Helló " + szemely + ".<br>");
   document.writeln("Jóslat: jövőre " + evek + "éves leszel.");
   document.writeln("</body> </html>");
   document.close();
}
</script>
</head>

<body>
<form>
Kérem, adja meg a nevét: <input type="text" name="nev">
<p>
Kérem, adja meg az életkorát: <input type="text" name="kor">
<p>
<input type="button" value="submit" onclick="valasz(this.form)">
</form>
</body>
</html>
```

7.31. **ábra.** *A JavaScript használata egy űrlap feldolgozására*

cot egy változóban tároljuk. Az ilyen magas szintű képességek miatt a JavaScript ideális nyelv interaktív weboldalak létrehozására. Másrészt viszont az a tény, hogy e nyelv gyorsabban mutálódik, mint a röntgengépbe szorult muslica, rendkívül nehézzé teszi, hogy olyan JavaScript-programokat írjunk, amelyek minden platformon működnek. Egy nap talán ez a helyzet is rendeződik.

A 7.31. ábra egy JavaScript-programra mutat példát. A 7.30. ábrához hasonlóan itt is egy űrlap jelenik meg, ami megkérdezi a nevet és az életkort, majd megjósolja, hogy hány éves lesz az adott személy jövőre. Az oldal törzse majdnem ugyanaz, mint a PHP-példában. A legfontosabb különbség itt az *Elküld* gomb deklarációjában és a benne levő hozzárendelési utasításban van. Ez a hozzárendelési utasítás mondja meg a böngészőnek, hogy gombnyomásra hívja meg a *valasz* szkriptet, és adja át neki az űrlapot mint paramétert.

Ami itt teljesen új, az a *valasz* nevű JavaScript-függvény deklarációja a HTML-állomány fejlécében, ahol rendszerint csak a cím, háttérszín stb. található. Ez a függvény kiolvassa a *nev* mező értékét az űrlapból, és karakterláncként tárolja a *szemely* nevű változóban. Kiolvassa még a *kor* mező értékét is, átalakítja egésszé az *eval* függvény használatával, hozzáad egyet, és az eredményt tárolja az *evek* változóban. Azután megnyit egy dokumentumot a kimenet számára, elvégez négy kiíratást a *writeln* metódussal, majd lezárja a dokumentumot. A dokumentum egy HTML-állomány lesz, amint az a benne lévő különféle HTML-címkékből is látszik. Végül a böngésző megjeleníti a dokumentumot a képernyőn.



7.32. ábra. *(a) Kiszolgálóoldali szkript PHP-vel. (b) Ügyféloldali szkript JavaScripttel*

Fontos megértenünk, hogy a PHP és JavaScript hasonlók abból a szempontból, hogy mindkettő kódot ágyaz be a HTML-állományokba, de teljesen máshogy kerülnek feldolgozásra. A 7.30. ábrán látható PHP-példában, miután a felhasználó rákattintott az *Elküld* gombra, a böngésző összegyűjti az információt egy hosszú karakterláncba, és egy PHP-oldalra vonatkozó kérés formájában elküldi azt a kiszolgálónak. A kiszolgáló betölti a PHP-állományt, és egy új HTML-oldal létrehozása érdekében végrehajtja a benne lévő PHP-szkriptet. Ezt az oldalt visszaküldi megjelenítésre a böngészőnek. A böngésző még abban sem lehet biztos, hogy az oldalt egy program állította elő. Ez a feldolgozási folyamat látható a 7.32.(a) ábrán az 1-4 lépésekben.

A 7.31. ábrán látható JavaScript-példában az *Elküld* gomb megnyomása után a böngésző értelmezi az adott oldalon található JavaScript-függvényt. Mindez ott helyben, a böngészőn belül kerül végrehajtásra. A kiszolgálóval nincs semmilyen kapcsolat. Ez a feldolgozási folyamat látható a 7.32.(b) ábrán az 1-es és 2-es lépésekben. Ennek következtében az eredmény látszólag azonnal megjelenik, míg a PHP esetében előfordulhat néhány másodperces késedelem, amíg az eredményül kapott HTML-oldal megérkezik az ügyfélhez.

Ez az eltérés nem azt jelenti, hogy a JavaScript jobb, mint a PHP, hiszen a kettőnek egészen más a felhasználási területe. A PHP (és ebből adódóan a JSP és az ASP is) akkor használatos, amikor egy kiszolgálón lévő adatbázissal kell együttműködni. A JavaScriptet (és más ügyféloldali nyelveket is, amelyeket később említünk, mint például a VBScriptet) ezzel szemben akkor használják, amikor a kliensszámítógép előtt ülő felhasználóval kell együttműködni. Mindazonáltal lehetséges ezek kombinálása, ahogyan azt hamarosan látni fogjuk.

A JavaScript nem az egyedüli mód, ha erősen interaktív weboldalakat kívánunk előállítani. Windows-platformokon az egyik alternatíva erre a Visual Basic-alapú **VBScript**. Egy további népszerű, több platformon is támogatott módszer a **kisalkalmazások** (**applets**) használata. Ezek apró Java-programok, melyeket egy virtuális számítógép, a **JVM** (**Java Virtual Machine – Java virtuális gép**) számára gépi utasításokra fordítottak. A HTML-oldalakba (az <applet> és </applet> címkék közé) ágyazható kisalkalmazásokat a JVM-képességgel rendelkező böngészők értelmezik. Mivel a Java-kisalkalmazásokat nem közvetlenül futtatják, hanem értelmezik, a Java-értelmező meggátolhatja, hogy Rossz Dolgokat tegyenek. Legalábbis elméletben. A gyakorlatban azonban a kisalkalmazás írók végtelen sok hibát találnak a Java I/O könyvtárakban.

A Microsoft, válaszul a Sun Java kisalkalmazásaira, létrehozta az **ActiveX-vezérlőket** (**ActiveX controls**) tartalmazó weboldalakat. Ezek a vezérlők olyan programok, melyeket az x86-os processzorok gépi nyelvére fordítottak le, és teljesen hardveren futtatnak.

Ez a képességük sokkal gyorsabbá és rugalmasabbá teszi ezeket, mint az értelmezőt igénylő Java-kisalkalmazások, mert megtehetnek mindent, amit egy program megtehet. Amikor az Internet Explorer egy ActiveX-vezérlőt talál egy weboldalon, akkor letölti azt, ellenőrzi az azonosságát, és futtatja. Ismeretlen programok letöltése és futtatása azonban súlyos biztonsági kérdéseket vet fel, melyeket a 8. fejezetben válaszolunk meg.

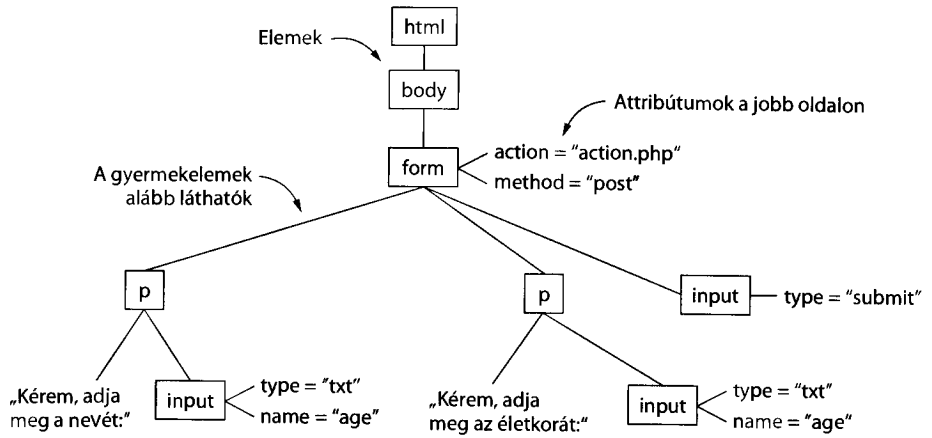
Mivel szinte minden böngésző képes mind a Java-programok, mind a JavaScript értelmezésére, az a tervező, aki egy erősen interaktív weboldalt szeretne elkészíteni, legalább két módszer közül választhat sőt, ha a különböző platformok közti hordozhatóság nem szempont, akkor ehhez még az ActiveX is hozzájön. Általános szabályként elmondhatjuk, hogy a JavaScript-programokat egyszerűbb megírni, a Java-kisalkalmazások gyorsabban hajtódnak végre, az ActiveX-vezérlők pedig a leggyorsabban futnak. Továbbá, mivel minden böngésző pontosan ugyanazt a JVM-et implementálja, de nincs két olyan böngésző, mely ugyanazt a JavaScript-verziót implementálná, ezért a Java-kisalkalmazások hordozhatóbbak, mint a JavaScript-programok. A JavaScriptről többet is megtudhatunk a témában íródott számos terjedelmes (olykor 1000 oldalnál is hosszabb) könyvből, mint például Flanagan [2010] művéből.

AJAX – Aszinkron JavaScript és XML

A rendkívül vonzó webalkalmazásoknak könnyen kezelhető felhasználói felületre és a távoli webszerverken tárolt adatok könnyű elérésére van szükségük. Azok a szkriptek, amelyeket az ügyfél oldalán (például JavaScript), és amelyeket a kiszolgáló oldalán (például PHP) létrehoznak, alapvető technikák egy adott probléma megoldásához. Ezeket a technikákat általában számos más, kulcsfontosságú technikával kombinálva, együtt szokták használni, amit **AJAX**-nak (**Asynchronous JAvascript and Xml – aszinkron JavaScript és XML**) neveznek. Számos teljes körű szolgáltatást nyújtó webalkalmazás, mint például a Google Gmail, a Maps és a Docs, AJAX segítségével készült.

Az AJAX némiképp zavarba ejtő, mert ez nem programnyelv, hanem olyan technikák halmaza, amelyek együttműködnek annak érdekében, hogy lehetővé tegyék a webalkalmazásokat, és amelyeknél minden bit ugyanolyan könnyen kezelhető és hatékony, mint a hagyományos asztali számítógépes alkalmazásoknál. Ezek a technikák a következők:

1. HTML és CSS az információ weboldalak formájában történő megjelenítéséhez.
2. DOM (Document Object Model – dokumentum objektummodell) a weboldalak részeinek megjelenítés közben történő megváltoztatásához.
3. XML (eXtensible Markup Language – kiterjeszthető jelölőnyelv) a programok és a kiszolgáló közötti adatcsere megvalósításához.
4. Egy aszinkron módszer a programok számára XML-adatok küldéséhez és fogadásához.
5. JavaScript mint programnyelv mindezen funkciók összekapcsolásához.



7.33. **ábra.** *A 7.30.(a) ábra HTML-jének DOM-fája*

Mivel ez egy egész gyűjtemény, végignézzük mindegyik elemét, hogy lássuk, mivel járul hozzá a működéshez. Korábban már megvizsgáltuk a HTML-t és a CSS-t. Ezek a szabványok arra szolgálnak, hogy megadják a tartalmat és annak megjelenítési módját. Bármelyik program, amelyik képes HTML-t és CSS-t előállítani, a webböngészőt úgy használhatja, mint egy megjelenítő motor (display engine).

A **DOM** (**Document Object Model – dokumentum objektum modell**) egy HTML-oldal egyfajta megjelenési formája, amely programok számára elérhető. Ez egy fa struktúrájú ábrázolás, ami tükrözi a HTML-elemek szerkezetét. Például a 7.30.(a) ábrán megadott HTML DOM-fája látható a 7.33. ábrán. A gyökér egy *html* elem, ami az egész HTML-blokkot ábrázolja. Ez az elem a *body* elem szülője, ami pedig a *form* elem szülője. Az űrlap két, jobb oldalra rajzolt attribútummal rendelkezik, az egyik attribútum az űrlap továbbításához szükséges metódus (egy *POST*), a másik attribútum az űrlappal kapcsolatos tevékenységet határozza meg (egy URL-t ad meg, ahová a kérést küldeni kell). Ennek az elemnek három gyermeke van, amelyek az űrlapon lévő két bekezdéscímke és egy beviteli címke tükröződései. A fa alján levelek találhatók, amelyek elemeket vagy literálokat, például karakterláncokat tartalmaznak.

A DOM-modell jelentősége abban áll, hogy a programok számára egyszerű lehetőséget kínál az oldal részeinek megváltoztatására. Nem kell újraírni az egész oldalt, csak a változást tartalmazó csomópontot kell kicserélni. Amikor ez a változás megtörténik, a böngésző ennek megfelelően frissíti a kijelzőt. Például, ha az oldal egyik részén lévő képet kicserélik a DOM-ban, a böngésző anélkül fogja frissíteni ezt a képet, hogy az oldal többi részét megváltoztatná. Már láttuk a DOM-ot működés közben, amikor a 7.31. ábra JavaScript példája sorokat adott hozzá a *document* elemhez, hogy új szövegsorokat jelenítsen meg a böngésző ablakának alján. A DOM hatékony módszer olyan oldalak készítéséhez, amelyek működés közben megváltozhatnak.

A harmadik technika, az XML (**eXtensible Markup Language – kiterjeszthető jelölőnyelv**) egy olyan nyelv, amely strukturált tartalom meghatározására szolgál. A HTML összevegyíti a tartalmat a formázással, mert az információ megjelenítésével törődik. A webalkalmazások általánosabbá válásával azonban növekszik az igény a strukturált

tartalomnak a formázástól való elválasztására. Például képzeljünk el egy programot, ami valamilyen könyv legjobb árát keresi a weben. Ennek számos weboldalt kell elemeznie a tétel címe és ára után kutatva. HTML-ben leírt weboldalak esetén a programnak nagyon nehéz kitalálnia, hogy hol van a cím és hol van az ár.

Ezért a W3C kifejlesztette az XML-t [Bray és mások, 2006], hogy lehetővé tegyék a webes tartalom strukturálását az automatizált feldolgozás érdekében. A HTML-lel ellentétben, az XML-ben nincsenek definiált címkék. Minden felhasználó meghatározhatja a saját egyéni címkéit. Az XML-dokumentumnak egy egyszerű példája látható a 7.34. ábrán. Ez egy „könyvlista" nevű szerkezetet határoz meg, ami történetesen könyvek listája. Minden könyvnek három mezője van: a cím, a szerző és a kiadás éve. Rendkívül egyszerű módon, ezeknek a struktúráknak lehetnek ismétlődő mezőik (például több szerző), opcionális mezőik (például egy CD-ROM-melléklet) és alternatív mezőik (például egy könyvesbolt URL-je, ha a könyv még üzletben kapható, vagy egy aukciós hely URL-je, ha már üzletben nem kapható).

Példánkban mindhárom mező oszthatatlan entitás, de egyébként a mezők további részekre való osztása is megengedett. Például, a még aprólékosabb keresések és formázások érdekében a *szerző* mezőt a következőképp is megadhattuk volna:

```
<szerzo>
  <keresztnev> George </keresztnev>
  <vezeteknev> Zipf </vezeteknev>
</szerzo>
```

Minden mezőt almezőkre, majd al-almezőkre stb. oszthatunk, tetszőleges mélységig.

A 7.34. ábrán látható állomány mindössze egy három könyvből álló listát ad meg. Ez kiválóan alkalmas a böngészőben és a kiszolgálón futó programok közötti informá-

```
<?xml version="1.0" ?>
<konyvlista>
<konyv>
  <cim> Az emberi viselkedés és a legkisebb erőfeszítés elve </cim>
  <szerzo> George Zipf </szerzo>
  <ev> 1949 </ev>
</konyv>
<konyv>
  <cim> A hírközlés matematikai elmélete </cim>
  <szerzo> Claude E. Shannon </szerzo>
  <szerzo> Warren Weaver </szerzo>
  <ev> 1949 </ev>
</konyv>
<konyv>
  <cim> 1984 </cim>
  <szerzo> George Orwell </szerzo>
  <ev> 1949 </ev>
</konyv>
</konyvlista>
```

7.34. **ábra.** *Egy egyszerű XML-dokumentum*

ciótovábbításra, de semmit nem árul el arról, hogy a dokumentumot mint weboldalt hogyan kell megjeleníteni. Ennek érdekében a program, amely feldolgozza az információt és 1949-et a könyvek szempontjából nagyszerű évnek ítéli, egy olyan HTML-t állíthat elő, melyben a címek dőlt betűs szövegként jelennek meg. Egy másik lehetőség az, hogy az **XSLT (eXtensible Stylesheet Language Transformations – kiterjeszthető stíluslapnyelv-átalakítások)** nyelvet használjuk annak meghatározására, hogy az XML-t miként kell HTML-lé transzformálni. Az XSLT hasonlít a CSS-re, de sokkal hatékonyabb annál. A részletektől az olvasót megkíméljük.

Egy másik előnye az adatok HTML helyett XML-ben történő kifejezésének az, hogy ezt a programok könnyebben tudják elemezni. A HTML-t eredetileg kézzel írták (gyakran még ma is), ezért sok közülük egy kicsit hanyag. A zárócímkéket (például a </p>-t) gyakran kifelejtik. Más címkéknek, mint a
, nincsen megfelelő zárócímkéje. Ismét más címkéket nem megfelelően ágyaznak egymásba, valamint a címkéket és attribútumokat felváltva hol kis-, hol nagybetűkkel írják. A legtöbb böngésző minden tőle telhetőt elkövet az eredeti szándék megfejtése érdekében. Az XML definíciója szigorúbb és tisztább. A címkék nevei és az attribútumok mindig kisbetűsek, a címkéket a megnyitással ellentétes sorrendben mindig be kell zárni (vagy egyértelműen jelezni kell, ha ez egy megfelelő záróelem nélküli üres címke), továbbá az attribútumok értékét idézőjelek között kell feltüntetni. Ez a pontosság az elemzést könnyebbé és félreérthetetlenné teszi.

A HTML-t még az XML kifejezéseivel is meghatározták. Ezt a megközelítést **XHTML**-nek **(eXtended Hypertext Markup Language – kiterjesztett hipertextjelölő nyelv)** nevezik. Ez alapvetően a HTML-nek egy „nagyon válogatós" változata. Az XHTML-oldalaknak szigorúan be kell tartaniuk az XML szabályait, máskülönben a böngésző nem fogadja el azokat. Nincs több silány weboldal és böngészők közti következetlenség! Akár az XML esetében, itt is az a szándék, hogy a programokkal (ebben az esetben webalkalmazásokkal) történő feldolgozás számára jobb oldalak készüljenek. Bár az XHTML 1998 körül született meg, csak lassan zárkózik fel. A HTML-t készítő emberek nem értik, miért van szükségük az XHTML-re, és a böngésző támogatás is késlekedett. A HTML 5.0 meghatározása éppen most van folyamatban, ezért egy weboldal akár HTML-formában, akár XHTML-formában leírható az átmenet megkönnyítése érdekében. Végső soron az XHTML-nek le kellene váltania a HTML-t, de még hosszú idő fog eltelni, míg ez az átmenet végbemegy.

Az XML népszerűnek bizonyult a programok közötti kommunikáció nyelveként is. Amikor ezt a kommunikációt a (következő szakaszban ismertetett) HTTP-protokollal valósítják meg, akkor webszolgáltatásról (web service) beszélünk. Konkrétan a **SOAP (Simple Object Access Protocol – egyszerű objektumhozzáférési protokoll)** a webszolgáltatások megvalósításának egy olyan módja, amely programok között távoli eljáráshívásokat (RPC) hajt végre nyelv- és rendszerfüggetlen módon. Az ügyfél csak összeállítja a kérést egy XML-üzenet formájában, és a HTTP-protokoll segítségével elküldi a kiszolgálónak. A kiszolgáló a választ szintén XML-üzenetként küldi vissza. Ily módon a különböző platformokon lévő alkalmazások is kommunikálhatnak egymással.

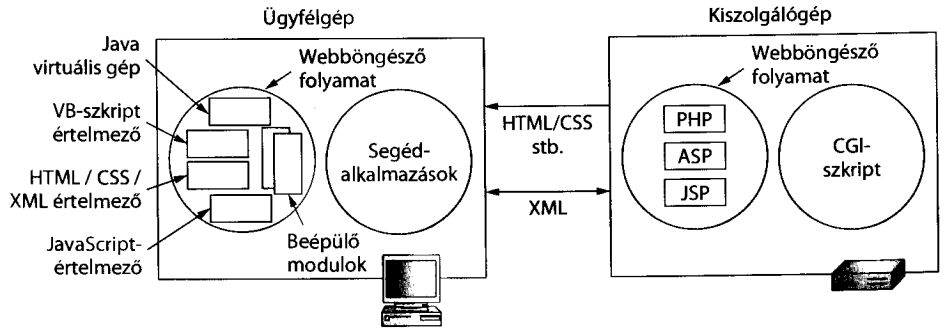
Visszatérve az AJAX-hoz, a lényeg számunkra egyszerűen csak az, hogy az XML egy hasznos formátum az adatoknak a böngészőben és a kiszolgálón futó programok közötti kicseréléséhez. Annak érdekében azonban, hogy a böngésző felhasználói felülete az adatok küldése és fogadása közben könnyen kezelhető maradhasson, a szkripteknek

képesnek kell lenniük olyan **aszinkron I/O (asynchronous I/O – aszinkron bevitel/kivitel)** végrehajtására, amely nem blokkolja a kijelzőt addig, amíg a kérésre adott válasz meg nem érkezik. Például képzeljünk el egy térképet, ami görgethető a böngészőben! Amikor a szkript értesítést kap a görgetési műveletről, a térkép oldalán lévő szkript további térképadatokat igényelhet a szervertől, ha a térkép megjelenített része közel jár az adatok széléhez. A felhasználói felületnek nem szabad lefagynia ezeknek az adatoknak a letöltése alatt. Egy ilyen felhasználói felület nem nyerne felhasználói díjat. A görgetésnek folyamatosan kell folytatódnia. Amikor az adatok megérkeznek, a szkriptet értesítik, hogy felhasználhatja az adatokat. Ha minden jól megy, az új térképadatok hamarabb megérkeznek, mint szükség lenne rájuk. A korszerű böngészők támogatják a kommunikációnak ezt a modelljét.

A kirakó utolsó darabja egy szkriptnyelv, ami összetartja az AJAX-ot azáltal, hogy hozzáférést biztosít a fent felsorolt technikákhoz. A legtöbb esetben ez a nyelv a JavaScript, de vannak alternatívák, mint például a VBScript. Korábban bemutattunk egy egyszerű JavaScript-példát. Ne tévesszen meg senkit ez az egyszerűség! A JavaScript-nek sok fura megoldása van, de ez egy teljes értékű programozási nyelv, a C vagy Java minden erősségével. Vannak változói, karakterláncai, tömbjei, objektumai, függvényei és a szokásos vezérlési szerkezetei. Ezenkívül különleges felhasználói felületekkel rendelkezik a böngészőhöz és a weblapokhoz. A JavaScript képes követni az egér mozgását a képernyőn lévő objektumok felett, ami megkönnyíti a hirtelen előbukkanó menük elkészítését és eleven weboldalakhoz vezet. A weboldalak eléréséhez tudja a DOM-ot használni, tudja a HTML-t és az XML-t manipulálni, és képes aszinkron HTTP-kommunikációt végrehajtani.

Mielőtt befejeznénk a dinamikus oldalak témakörét, röviden összegezzük az eddig érintett technikákat úgy, hogy hivatkozunk rájuk egy ábrán! Teljes weboldalakat is elő lehet állítani menet közben különféle szkriptekkel a kiszolgáló oldalán. A szkriptek megírhatók olyan kiszolgálót kiterjesztő nyelveken, mint a PHP, JSP vagy az ASP.NET, vagy futhat különálló CGI-folyamatokként, és így bármilyen nyelven megírhatók. Ezek a lehetőségek láthatók a 7.35. ábrán.

Miután a böngésző megkapta ezeket a weboldalakat, úgy bánik velük, mint a közönséges HTML-ben, CSS-ben és más MIME-típusokkal megírt oldalakkal, csak megjeleníti azokat. Böngészőben futó beépülő modulok és böngészőn kívül futó segédalkalmazások telepíthetők a böngésző által támogatott MIME-típusok kiterjesztése érdekében.



7.35. ábra. *Különböző, dinamikus oldalak előállítására szolgáló technikák*

A dinamikus tartalmat az ügyfél oldalán is elő lehet állítani. A weboldalakba ágyazott programok megírhatóak JavaScript-ben, VBScript-ben, Java-ban és más nyelveken. Ezek a programok tetszőleges számításokat végezhetnek, és frissíthetik a kijelzőt. A weboldalakban lévő programok az AJAX segítségével aszinkron módon cserélhetnek XML és más típusú adatokat a szerverrel. Ez a modell támogatja a legkülönfélébb webalkalmazásokat, amelyek éppen úgy néznek ki, mint a hagyományos alkalmazások, eltekintve attól, hogy a böngésző belsejében futnak, és hozzáférnek az interneten lévő kiszolgálók által tárolt információhoz.

7.3.4. HTTP – a hipertext-átviteli protokoll

Most, hogy már tisztában vagyunk a webes tartalmakkal és alkalmazásokkal, elérkezett az ideje, hogy megvizsgáljuk azt a protokollt, amelyet a webszerverek és az ügyfelek közötti információcseréhez használnak. Ez a **HTTP (HyperText Transfer Protocol – hipertext-átviteli protokoll)**, amit az RFC 2616 határoz meg.

A HTTP egy kérés-válasz protokoll, ami rendszerint TCP felett működik. A protokoll meghatározza, hogy az ügyfelek milyen üzeneteket küldhetnek a kiszolgálóknak, és hogy ezekre milyen válaszokat kaphatnak. A kérések és válaszok fejlécei ASCII-szövegek, akárcsak az SMTP esetében. A tartalmakat MIME-szerű formátumban adják meg, ami szintén hasonlít az SMTP-re. Részben ez az egyszerű modell a felelős a világháló korai sikeréért, mert egyszerűvé tette a fejlesztést és a telepítést.

Ebben a szakaszban szemügyre vesszük a HTTP-nek a manapság használt fontosabb tulajdonságait. Mielőtt azonban elmerülnénk a részletekben, felhívjuk a figyelmet arra, hogy az interneten történő felhasználásának módja folyamatosan fejlődik. A HTTP egy alkalmazási rétegbeli protokoll, mert TCP felett fut, és szorosan kapcsolódik a webbel. Ezért foglalkozunk vele ebben a fejezetben. Más értelemben azonban a HTTP egyre inkább hasonlít egy szállítási protokollra, ami a folyamatok számára lehetőséget ad arra, hogy tartalmakat küldjenek és fogadjnak különféle hálózatok határai között. Ezeknek a folyamatoknak nem kell webböngészőnek és webszervernek lenniük. Egy médialejátszó HTTP-t használhat a szerver megszólítására, és albuminformáció lekérésére. A víruskereső szoftver HTTP-t használhat a legújabb frissítések letöltéséhez. A fejlesztők a projektállományok lekérésére használhatják a HTTP-t. A szórakoztató elektronikai termékek, mint a digitális képkeretek, gyakran beépített HTTP-kiszolgálót használnak felhasználói felületként a külvilág felé. A két számítógép közötti kommunikáció egyre nagyobb mértékben zajlik HTTP felett. Például egy légitársaság kiszolgálója SOAP-ot (egy HTTP feletti XML-alapú távoli eljáráshívást) használhat arra, hogy kapcsolatba lépjen az autókölcsönző szerverével és az autófoglaláshoz, mindezt egy a szabadidő aktív eltöltésére javasolt programcsomag részeként. Ezek az irányzatok valószínűleg folytatódni fognak a HTTP egyre növekvő használatával.

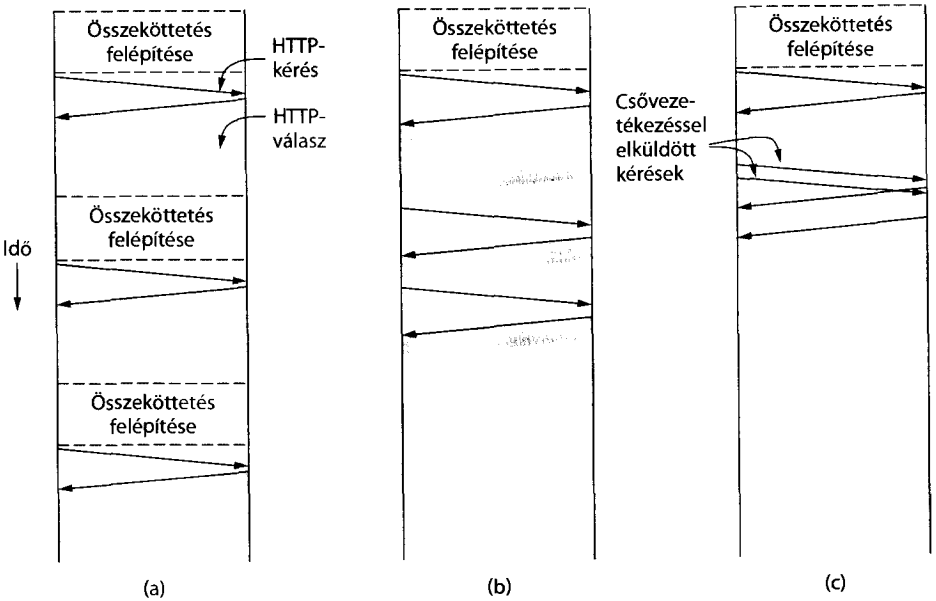
Kapcsolatok

A böngészők általában úgy veszik fel a kapcsolatot a kiszolgálókkal, hogy egy TCP-összeköttetést építenek ki a kiszolgáló gépének 80-as portjával, bár a szabvány ezt formálisan nem követeli meg. A TCP használatának előnye az, hogy sem a böngészőnek, sem a kiszolgálónak nem kell aggódnia a hosszú üzenetek kezelésének módja, a megbízhatóság és a torlódásvezérlés miatt. Az összes ilyen kérdésről a TCP-implementáció gondoskodik.

A világháló korai szakaszában, a HTTP 1.0-ban az összeköttetés kiépítése után egyetlen kérést küldtek el, amire egyetlen válasz érkezett. Ezután a TCP-összeköttetést lebontották. Abban a világban, amikor egy tipikus weboldal még kizárólag szövegből állt, ez még helyénvaló is volt. Az átlagos weboldalak mérete gyorsan növekedett, kezdett már nagyon sok beágyazott hivatkozást tartalmazni olyan tartalmakra, mint például az ikonok és egyéb vizuális nyalánkságok. A működést nagyon költségessé tette, hogy minden egyes ikon átviteléhez külön TCP-összeköttetést kellett kiépíteni.

Ez a megfigyelés vezetett a HTTP 1.1-hez, ami már támogatja a **tartós kapcsolatokat (persistent connections)**. Ezáltal lehetővé vált, hogy kiépítsünk egy TCP-összeköttetést, elküldjünk egy kérést, megkapjuk a választ, majd pedig további kéréseket küldjünk és válaszokat kapjunk. Ezt a stratégiát a **kapcsolat újrahasználásának (connection reuse)** is nevezik. Azáltal, hogy több kérés között oszlik meg a TCP-kiépítés, indítás és lebontás költsége, az egyes kérésekre jutó, a TCP által okozott relatív többletterhelés kisebb lesz. A kéréseket akár csővezeték (pipeline) módszerrel is lehet küldeni, azaz már azelőtt el lehet küldeni a 2-es kérést, hogy az 1-es kérésre megérkezett volna a válasz.

E három eset közötti teljesítménybeli különbséget mutatja a 7.36. ábra. Az (a) rész három kérést mutat, az egyik követi a másikat, mindegyik külön összeköttetést használ. Tegyük fel, hogy ez egy weboldalt ábrázol, amin két beágyazott kép található, amelyek ugyanazon a kiszolgálón helyezkednek el. A képek URL-jeinek meghatározása a főol-



7.36. ábra. *HTTP (a) több összeköttetéssel és egymást követő kérésekkel. (b) Tartós kapcsolat és egymást követő kérések. (c) Tartós kapcsolat és csővezeték módszerrel küldött kérések*

dal lekérése után történik, ezért azokat a főoldal után töltik le. Manapság egy átlagos oldalnak körülbelül 40 további objektuma van, amit le kell kérni a megjelenítéshez, de ez túlságosan megnövelné az ábránkat, ezért csak két beágyazott objektumot fogunk használni.

A 7.36.(b) ábrán az oldalt tartós kapcsolattal kérik le. Ez azt jelenti, hogy kezdetben TCP-összeköttetés létesül, aztán elküldésre kerül ugyanaz a korábbi három kéréssorban egymás után, és majd csak ezután bomlik le az összeköttetés. Figyeljük meg, hogy a lekérés sokkal gyorsabban végbemegy! A gyorsulásnak két oka van. Először is, nem veszett el idő a további összeköttetések kiépítésével, ugyanis minden egyes TCP-összeköttetés létrehozásához legalább egy körbefordulási időre van szükség. Másodszor, ugyananannak a két képnek a továbbítása gyorsabban megtörténik. Miért? Ennek a TCP torlódáskezelése az oka. Az összeköttetés kezdetén a TCP a lassú indítási eljárást használja az átvitel növelése érdekében, amíg megtanulja a hálózati utak viselkedését. Ennek a bemelegedési időszaknak az a következménye, hogy több rövid TCP-összeköttetésnek aránytalanul hosszabb ideig tart az információátvitel, mint egy hosszabb TCP-összeköttetésnek.

Végül, a 7.36.(c) ábrán egyetlen tartós kapcsolat és csővezeték módszerrel küldött kérések láthatóak. A második és harmadik lekérés különösen gyorsan követi egymást, amint a főoldalnak elegendően nagy része megérkezett ahhoz, hogy meg lehessen állapítani a képek lekérésének szükségességét. Végül ezekre a kérésekre adott válaszok következnek. Ez a módszer lerövidíti a kiszolgáló tétlenül eltöltött idejét, és így tovább fokozza a teljesítőképességet.

A tartós összeköttetések azonban nincsenek ingyen. Ezek használata kapcsán egy új kérdés merül fel: mikor kell lezárni az összeköttetést? Egy kiszolgálóval létrehozott összeköttetésnek megnyitva kell maradnia az oldal töltése közben. És azután? Jó esély van rá, hogy a felhasználó rá fog kattintani egy hivatkozásra, ami új oldalt kér a kiszolgálótól. Ha az összeköttetés megnyitva marad, a következő kérés azonnal elküldhető. Arra azonban nincs garancia, hogy az ügyfél valamikor nemsokára egy másik kérést intéz a kiszolgálóhoz. A gyakorlatban az ügyfelek és a kiszolgálók általában addig tartják megnyitva a tartós kapcsolatokat, amíg egy rövid időt (például 60 másodpercet) tétlenül nem töltenek, vagy amíg a nyitott kapcsolatok száma túl nagy nem lesz, amikor is be kell zárni néhányat.

A figyelmes olvasó észrevehette, hogy egy kombinációt eddig kihagytunk. Lehetséges TCP-összeköttetésenként egy-egy kérés elküldése is, de egyidejűleg több TCP-összeköttetés párhuzamos használatával. Ezt a **párhuzamos összeköttetési (parallel connection)** módszert széles körben alkalmazták a böngészők a tartós kapcsolatok előtt. Ugyanazokkal a hátrányokkal bír, mint az egymást követő összeköttetések – magasabb költségek – de annál sokkal jobb a teljesítőképessége. Ennek az az oka, hogy az összeköttetések egyidejű elindítása és felfutása a késleltetési idő egy részét elrejti. Példánkban mindkét beágyazott képhez tartozó összeköttetés egyidejűleg elindulhat. Ugyanazzal a kiszolgálóval azonban nem javasolt túl sok összeköttetést használni. Ennek oka az, hogy a TCP minden egyes összeköttetésnél külön torlódáskezelést végez. Ennek következtében az összeköttetések egymással versengenek, ami csomagvesztést okoz, és összességében a hálózatot sokkal aggresszívebben használják, mint egy egyedi összeköttetés. A tartós kapcsolatok jobbak, és használatukat előnyben részesítik a párhuzamos összeköttetésekkel szemben, mert kivédik az állandó költségeket és nem szenvednek a torlódási problémáktól.

Metódusok

Bár a HTTP-t a weben való használatra tervezték, szándékosan a szükségesnél általánosabbra készítették, szem előtt tartva az eljövendő objektumorientált alkalmazásokat. A protokoll ebből kifolyólag olyan műveleteket, más néven **metódusokat (methods)** is támogat, melyek nem pusztán egy weboldal elkérésére irányulnak. Ez az általánosság tette lehetővé a SOAP létrejöttét is.

Minden kérés egy- vagy többsornyi ASCII-szövegből áll, ahol az első sor első szava a kért metódus neve. A beépített metódusokat a 7.37. ábra sorolja fel. A nevek érzékenyek a kisbetű/nagybetű különbségre, így a *GET* egy legális eljárás, de a *get* nem az.

A *GET* metódus arra kéri a kiszolgálót, hogy küldje el az oldalt. (Amikor „oldalról" beszélünk, akkor ez alatt a legáltalánosabb esetben „objektumot" értünk, de a fogalmak megértéséhez elegendő, ha az oldalra úgy gondolunk, mint egy állomány tartalmára.) Az oldal megfelelő módon MIME-ben van kódolva. A webszerverekhez intézett kérések túlnyomó többsége *GET*. A *GET* szokásos alakja:

```
GET állománynév HTTP/1.1
```

ahol az *állománynév* a letöltendő oldal neve, és 1.1 a használt protokoll verziója.

A *HEAD* metódus csak az üzenet fejlécét kéri, a tényleges oldal nélkül. Ez a metódus használható arra, hogy információt gyűjtsünk indexelési célokra, vagy egyszerűen ellenőrizzük egy URL érvényességét.

A *POST* metódust használják űrlapok elküldésére. Ezt és a *GET*-et is használják a SOAP-webszolgáltatásokhoz. A *GET*-hez hasonlóan ez is egy URL-t hordoz, de ahelyett, hogy egyszerűen lehozna egy oldalt, adatokat tölt fel a kiszolgálóra (például az űrlap tartalmát vagy távoli eljáráshívás paramétereit). A szerver ezután az URL-től függően elvégez valamit az adatokkal, elméletileg hozzáfűzi az adatokat az objektumhoz. A hatás lehet például egy cikk megvásárlása, vagy egy távoli eljárás meghívása. Végül az eljárás visszatér az eredményt jelző oldallal.

Metódus	Leírása
GET	Weboldal olvasása
HEAD	Weboldal fejlécének olvasása
POST	Weboldalhoz történő hozzáfűzés
PUT	Weboldal tárolása
DELETE	Weboldal eltávolítása
TRACE	Bejövő kérés visszaküldése
CONNECT	Kapcsolódás proxy-n keresztül
OPTIONS	Egy oldal opcióinak lekérdezése

7.37. ábra. *A beépített HTTP-kérés-metódusok*

A többi metódust nem nagyon használják a web böngészésére. A *PUT* metódus a *GET* fordítottja: az oldal olvasása helyett írja azt. Ez a metódus teszi lehetővé weboldalak gyűjteményének felépítését a távoli kiszolgálókon. A kérés törzse tartalmazza az oldalt. Ez lehet MIME szerint kódolt; ebben az esetben a *PUT*-ot követő sorok tartalmazhatják a hitelesítési fejléceket annak bizonyítására, hogy a hívónak valóban van engedélye a kívánt feladat végrehajtására.

A *PUT*-hoz némileg hasonló a *POST* metódus. Ez is egy URL-t hordoz, de a meglevő adat felülírása helyett az új adatot „hozzáadja" ahhoz valamilyen általános értelemben. Az ilyen értelmű hozzáadásra lehet példa egy üzenet postázása egy hírcsoportba, vagy egy állomány hozzáadása egy hirdetőtábla-rendszerhez. A gyakorlatban sem a *PUT*-ot, sem a *POST*-ot nem nagyon használják.

A *DELETE* azt teszi, amit várunk: törli az oldalt, vagy legalább is jelzi, hogy a webszerver elfogadta az oldal törlését. Akárcsak a *PUT*-nál, itt is nagy szerepet játszik a hitelesítés és az engedélyezés.

A *TRACE* metódus a hibakeresést szolgálja. Arra utasítja a kiszolgálót, hogy küldje vissza a kérést. Ez akkor hasznos, amikor a kéréseket nem a megfelelő módon dolgozzák fel, és az ügyfél tudni akarja, hogy ténylegesen milyen kérést kapott meg a kiszolgáló.

A *CONNECT* metódus segítségével a felhasználó számára lehetővé válik egy webszerverhez közbeeső egységen, mint például egy webes gyorstáron keresztül történő csatlakozás.

Az *OPTIONS* metódus lehetővé teszi, hogy az ügyfél lekérdezzen egy oldalt, valamint megkapja az oldallal használható metódusokat és fejléceket.

Minden kérésre érkezik egy válasz, ami egy állapotsorból, és esetleg további információból (például egy weboldal része vagy egésze) áll. Az állapotsor tartalmazza a háromjegyű állapotkódot, ami megmondja, hogy a kérést teljesítették-e, és ha nem, miért nem. Az első számjegy öt nagy csoportra osztja a válaszokat, ahogy azt a 7.38. ábra mutatja. Az 1xx kódokat ritkán használják a gyakorlatban. A 2xx kódok azt jelentik, hogy a kérést sikeresen kezelték, és érkezik is vissza a tartalom (ha van). A 3xx kódok arra utalnak, hogy az ügyfélnek máshol kell keresgélnie vagy egy másik URL-lel vagy a saját gyorstárjában (ezt később tárgyaljuk). A 4xx kódok azt jelentik, hogy a kérést az ügyfél hibája miatt nem lehet teljesíteni, mert például érvénytelen a kérés, vagy nem létezik az oldal.

Kód	Jelentése	Példák
1xx	Információ	100 = a kiszolgáló jóváhagyja az ügyfél kérését
2xx	Siker	200 = sikeres kérés; 204 = nincs tartalom
3xx	Átirányítás	301 = az oldal elköltözött; 304 = a gyorstárban tárolt oldal még érvényes
4xx	Ügyfél hibája	403 = tiltott oldal; 404 = az oldal nem található
5xx	Kiszolgáló hibája	500 = belső hiba a kiszolgálóban; 503 = próbálkozzon újra később

7.38. ábra. *A válasz állapotkódjának csoportjai*

Végül, az 5xx kódok azt jelzik, hogy magának a kiszolgálónak van valamilyen belső gondja vagy egy szoftverhiba, vagy az ideiglenes túlterhelés miatt.

Üzenetfejlécek

A kérés sora (például a *GET* metódust tartalmazó sor) után további információt tartalmazó sorok következhetnek. Ezeket a **kérésfejlécének (request headers)** nevezzük. Az ilyen információ az eljáráshívások paramétereihez hasonlatos. A válaszoknak szintén lehetnek **válaszfejlécei (response headers)**. Egyes fejléceket akár mindkét irányban is lehet használni. A legfontosabb fejléceket a 7.39. ábra mutatja be. Ez a lista nem rövid, ezért ahogyan azt feltehetőleg az olvasó is gondolja, minden egyes kérés- és válaszfejlécnek gyakran több változata létezik.

A *Felhasználói-ügynök (User-Agent)* fejléc lehetővé teszi, hogy az ügyfél közölje a kiszolgálóval böngészőjének típusát (például Mozilla/5.0 és Chrome/5.0.375.125). Ez az információ azért hasznos, mert lehetővé teszi, hogy a kiszolgálók a célnak megfelelően alakítsák a böngészőknek szánt válaszaikat, mivel a különféle böngészők erősen eltérő képességekkel rendelkezhetnek, és másképp viselkedhetnek.

A négy *Elfogadható (Accept)* fejléc azt mondja meg a kiszolgálónak, hogy az ügyfél mit hajlandó elfogadni, amennyiben csak egy korlátozott választék elfogadására van módja. Az első típus azt adja meg, hogy az ügyfél milyen MIME-típusokat lát szívesen (például *text/html*). A második megadja a karakterkészletet (például *ISO-8859-5* vagy *Unicode-1-1*). A harmadik a tömörítő eljárásokkal foglalkozik (például *gzip*). A negyedik egy természetes nyelvet jelöl (például spanyol). Ha a kiszolgáló több oldal közül is választhat, akkor ezen információ felhasználásával azt adhatja vissza, amit az ügyfél keres. Ha nem képes teljesíteni a kérést, akkor egy hibakóddal tér vissza, és a kérés meghiúsul.

A *Ha-változott-azóta (If-Modified-Since)* és a *Ha-egyik-sem-egyezik (If-None-Match)* fejléceket a gyorstárakkal kapcsolatban használják. Lehetővé teszik az ügyfél számára, hogy csak akkor kérje egy oldal küldését, ha annak gyorstárban lévő másolati példánya már nem érvényes. A gyorstárazást hamarosan bemutatjuk.

A *Hoszt (Host)* fejléc a kiszolgálót nevezi meg; ezt az URL-ből veszik ki. Ez a fejléc kötelező. Azért használják, mert némely IP-címhez több DNS-név is tartozik, és a kiszolgálónak valahogy el kell döntenie, hogy melyik hosztnak adja tovább a kérést.

A *Hitelesítés (Authorization)* fejlécre a védett oldalak esetén van szükség. Ekkor az ügyfélnek esetleg bizonyítania kell, hogy jogosult megnézni az oldalt – erre szolgál ez a fejléc.

Az ügyfél a *Hivatkozó* (a hibásan írt *Referer*) fejlécet használja arra, hogy megadja azt az URL-t, amelyik a most kért URL-re hivatkozott. Leggyakrabban ez az előző oldal URL-je. Ez a fejléc különösen fontos a webböngészés nyomon követéséhez, mert megmondja a kiszolgálóknak, hogy a kliens hogyan jutott az oldalra.

Bár a sütikkel nem az RFC 2616, hanem az RFC 2109 foglalkozik, ezeknek is vannak fejléceik. A kiszolgálók a *Süti-beállítás (Set-Cookie)* sütifejléc útján küldik el sütijeiket az ügyfeleknek. Az ügyfélnek el kell mentenie a sütit, és az ezt követő kérésekben a *Süti (Cookie)* fejlécet használva vissza kell küldenie a kiszolgálónak. (Jegyezze meg, hogy

Fejléc	Típus	Tartalom
Felhasználói-ügynök	Kérés	Információ a böngészőről és annak platformjáról
Elfogadható	Kérés	Az ügyfél által kezelhető oldalak típusa
Elfogadható-karakterkészlet	Kérés	Az ügyfél által elfogadható karakterkészletek
Elfogadható-kódolás	Kérés	Az ügyfél által kezelhető oldalkódolási típusok
Elfogadható-nyelv	Kérés	Az ügyfél által kezelhető természetes nyelvek
Ha-változott-azóta	Kérés	Idő és dátum a frissesség ellenőrzéséhez
Ha-egyik-sem-egyezik	Kérés	Korábban küldött címkék a frissesség ellenőrzéséhez
Hoszt	Kérés	A kiszolgáló DNS-neve
Hitelesítés	Kérés	Az ügyfél igazolásainak listája
Hivatkozó	Kérés	Az előző URL, ahonnan a kérés érkezett
Süti	Kérés	Az előzőleg kapott süti visszaküldése a kiszolgálónak
Süti-beállítás	Válasz	Az ügyfél által tárolandó süti
Kiszolgáló	Válasz	Információ a kiszolgálóról
Tartalom-kódolás	Válasz	Hogyan van kódolva a tartalom (például *gzip*)
Tartalom-nyelv	Válasz	Az oldalon használt természetes nyelv
Tartalom-hossz	Válasz	Az oldal hossza bájtokban
Tartalom-típus	Válasz	Az oldal MIME-típusa
Tartalom-tartomány	Válasz	Azonosítja az oldal tartalmának egy részét
Utoljára-módosítva	Válasz	Az oldal utolsó változtatásának dátuma és ideje
Lejár	Válasz	Az oldal érvénytelenné válásának ideje és dátuma
Hely	Válasz	Megmondja az ügyfélnek, hogy hová küldje a kérését
Elfogadható-tartományok	Válasz	Jelzi, hogy a kiszolgáló elfogadja a bájttartományokra vonatkozó kéréseket
Dátum	Mindkettő	Az üzenet elküldésének dátuma és ideje
Tartomány	Mindkettő	Azonosítja egy oldal valamelyik részét
Gyorstár-vezérlés	Mindkettő	A gyorstár kezelésére vonatkozó irányelvek
ECímke	Mindkettő	Az oldal tartalmát jelző címke
Átállás	Mindkettő	Az a protokoll, amire a küldő váltani szeretne

7.39. ábra. *Néhány HTTP-üzenetfejléc*

létezik egy sokkal frissebb sütikre vonatkozó előírás új fejlécekkel, az RFC 2965, de ezt az ipar nagymértékben visszautasította és nem valósították meg széles körben.)

Számos más fejlécet is használnak a válaszokban. A *Kiszolgáló* (*Server*) lehetővé teszi a kiszolgáló számára, hogy felfedje a szoftverének sorszámát, ha akarja. A következő öt, „*Tartalom-*" (*Content*) kezdetű fejléc segítségével a kiszolgáló megadhatja az éppen elküldött oldal jellemzőit.

Az *Utoljára-módosítva* (*Last-modified*) fejléc megmondja, hogy mikor módosították utoljára az oldalt, a *Lejár* (*Expires*) fejléc pedig arról informál, hogy az oldal meddig marad érvényes. Mindkét fejléc fontos szerepet játszik a gyorstárak kezelésében.

A *Hely* (*Location*) fejlécet a kiszolgáló arra használja, hogy közölje az ügyféllel: próbálkozzon egy másik URL-lel. Ez akkor hasznos, ha az oldal elköltözött, vagy ha több URL is mutat ugyanarra az oldalra (esetleg különböző kiszolgálókon). Használják még azok a vállalatok is, melyeknek van egy központi weboldaluk a *com* körzetben, de onnan az ügyfeleket átirányítják a nemzeti vagy regionális oldalra az IP-címük vagy a preferált nyelvük alapján.

Ha egy oldal nagyon nagy méretű, akkor egy kisebb ügyfél esetleg nem akarja megkapni egyszerre az egészet. Egyes kiszolgálók bájttartományokra vonatkozó kéréseket is elfogadnak, így az oldalt több kisebb egységben is le lehet tölteni. Az *Elfogadható-tartományok* (*Accept-Ranges*) fejléc azt jelenti be, hogy a kiszolgáló hajlandó ilyen jellegű részleges kéréseket kezelni.

Ezzel elérkeztünk azokhoz a fejlécekhez, melyek mindkét irányban használhatók. A *Dátum* (*Date*) fejléc mindkét irányban használható, és az üzenet elküldésének dátumát és időpontját tartalmazza, míg a *Tartomány* (*Range*) fejléc megadja az oldalnak a válasz által szolgáltatott bájttartományát.

Az *ECímke* (*ETag*) fejléc egy rövid címkét ad meg, ami az oldal tartalmának neveként szolgál. Ezt a gyorstárazáshoz használják. A *Gyorstár-vezérlés* (*Cache-Control*) fejléc további explicit utasításokat ad az oldalak gyorstárazására (vagy még gyakrabban, a gyorstárazás mellőzésére) vonatkozóan.

Végül az *Átállás* (*Upgrade*) egy új kommunikációs protokollra, mint például egy jövőbeli HTTP-protokollra vagy biztonságos átvitelre történő átkapcsolásra szolgál. Ez lehetővé teszi, hogy az ügyfél bejelentse, mit képes kezelni, és hogy a kiszolgáló is megmondja, mi az, amit éppen használ.

Gyorstárazás

Az emberek gyakran visszatérnek az általuk korábban megtekintett weboldalakra, és a kapcsolódó weboldalaknak gyakran ugyanazok a beágyazott erőforrásaik. Néhány példa: azok a képek, amelyeket a webhely oldalai közti navigáláshoz használnak, valamint a közös stíluslapok és szkriptek. Nagyon pazarló lenne minden egyes megjelenítés előtt az ezekhez az oldalakhoz tartozó összes ilyen erőforrást lekérni, mert a böngészőnek már van egy másolati példánya.

Az oldalak későbbi felhasználás céljából történő tárolását **gyorstárazásnak** (**caching**) nevezzük. Ennek az az előnye, hogy amikor a gyorstárban lévő oldalt ismét felhasználják, nem kell megismételni az átvitelt. A HTTP beépített támogatással rendelkezik, hogy segítsen az ügyfeleknek azonosítani, mikor lehet biztonságosan újrahasználni az olda-

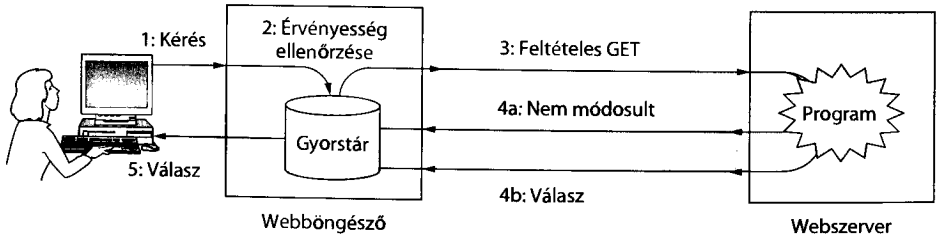
lakat. Ez a támogatás azáltal növeli meg a teljesítőképességet, hogy csökkenti a hálózati forgalmat és a késleltetést. A kompromisszum az, hogy a böngészőnek most már tárolnia kell az oldalakat, de ezt a kompromisszumot csaknem mindig érdemes megkötni, mert a helyi tárolás nem költséges. Az oldalakat általában lemezen tárolják, így újra felhasználhatók a böngésző későbbi futása alkalmával.

A nehéz ügy a HTTP gyorstárazásával kapcsolatban annak megállapítása, hogy egy oldalnak a korábban gyorstárba helyzetett másolati példánya ugyanolyan-e, mint az eredeti oldal lenne, ha azt újra lekérnék. Ezt a döntést nem lehet kizárólag az URL alapján meghozni. Például az URL megadhat egy olyan oldalt, ami megjeleníti a legfrissebb híreket. Ennek az oldalnak a tartalma sűrűn fog frissülni annak ellenére, hogy az URL ugyanaz marad. Vagy az oldal tartalma a görög és a római mitológia isteneinek listája is lehet. Ennek az oldalnak valamelyest kevésbé gyorsan kellene megváltoznia.

A HTTP két stratégiát használ ennek a problémának a megoldására. Ezek, mint a feldolgozás módjai láthatók a 7.40. ábrán, a kérés (1. lépés) és a válasz (5. lépés) között. Az első stratégia az oldal érvényesítése (2. lépés). A gyorstárhoz fordulnak, és ha az rendelkezik a kért URL-hez tartozó oldal olyan másolati példányával, amiről tudni lehet, hogy az friss (azaz még mindig érvényes), akkor nem szükséges azt újból lekérni a kiszolgálótól. Ehelyett közvetlenül visszaadható a gyorstárban tárolt oldal. Amikor a gyorstárban lévő oldalt eredetileg lekérték, visszaérkezett vele a *Lejár* fejléc, amelyben lévő aktuális dátum és időpont használható ennek a döntésnek a meghozatalához.

Nem minden oldal érkezik azonban megfelelő *Lejár* fejrésszel, ami megadja, mikor kell az oldalt újból lekérni. Végül is, jósolni nehéz – különösen a jövőre nézve. Ebben az esetben a böngésző valamilyen heurisztikát használhat. Például, ha az oldalt nem változtatták meg az elmúlt évben (ahogyan azt az *Utoljára-módosítva* fejléc állítja), akkor elég biztonságosan lehet fogadni arra, hogy nem fog megváltozni a következő órában. Garancia azonban nincsen rá, és ez egy rossz fogadás is lehet. Például lehet, hogy az értéktőzsde mára bezárt, és így az oldal órákig nem fog megváltozni, de gyorsan fog változni, amikor majd elkezdődik a következő kereskedési időszak. Ennél fogva egy oldal gyorstárban való tárolhatósága idővel erősen változhat. Emiatt a heurisztikát óvatosan kell használni, noha a gyakorlatban gyakran jól működik.

A még le nem járt oldalak megtalálása a legkedvezőbb eset a gyorstár használata során, mert ez azt jelenti, hogy a kiszolgálóval egyáltalán nem kell kapcsolatba lépni. Sajnos ez nem mindig sikerül. A kiszolgálóknak a *Lejár* fejlécet óvatosan kell használniuk, mert nem lehetnek biztosak abban, hogy mikor fogják frissíteni az oldalt. Tehát lehet, hogy a gyorstárban lévő másolati példányok még mindig frissek, de az ügyfél ezt nem tudja.



7.40. ábra. *HTTP-gyorstárazás*

Ebben az esetben a második stratégiát használják. Ez abból áll, hogy megkérdezik a kiszolgálót, vajon a gyorstárban lévő másolati példány még mindig érvényes-e? Ez a kérés a **feltételes GET** (**conditional GET**), és a 7.40. ábra 3. lépésében látható. Ha a szerver tudja, hogy a gyorstárban lévő másolat még mindig érvényes, akkor ezt megmondhatja egy rövid válaszban (4a lépés). Máskülönben el kell küldenie a teljes választ (4b lépés).

Több fejlécmezőt használnak arra, hogy lehetővé tegyék a kiszolgáló számára annak ellenőrzését, hogy a gyorstárban lévő másolat még mindig érvényes-e. Az ügyfél tudja a gyorstárban lévő oldal utolsó frissítésének idejét az *Utoljára-módosítva* fejlécből. Ezt az időt elküldheti a szervernek a *Ha-változott-azóta* fejléc segítségével, hogy az oldalt csak akkor kérje le, ha az időközben megváltozott.

Másik lehetőségként a kiszolgáló az oldallal együtt visszaküldhet egy *ECímke* fejlécet. Ez a fejléc egy címkét ad meg, ami az oldal tartalmának rövid neve. Ez hasonló, mint egy ellenőrző összeg, de jobb annál. (Ez egy kriptográfiai hash-érték lehet, melyet a 8. fejezetben ismertetünk.) Az ügyfél a gyorstárban lévő másolatokat a kiszolgálónak küldött *Ha-egyik-sem-egyezik* fejrésszel érvényesítheti, amely a gyorstárban lévő másolatok címkéit sorolja fel. Ha a címkék közül bármelyik megfelel annak a tartalomnak, amellyel a kiszolgáló válaszolna, a megfelelő gyorstárbeli másolat felhasználható. Ez a módszer akkor használható, amikor a frissesség megállapítása nem kényelmes vagy nem hasznos. Például egy kiszolgáló ugyanarra az URL-re különböző tartalmakat adhat vissza attól függően, hogy melyek az előnyben részesített nyelvek vagy MIME-típusok. Ebben az esetben önmagában a módosítás dátuma nem fog segíteni a kiszolgálónak meghatározni, hogy a gyorstárban lévő oldal friss-e.

Végül jegyezzük meg, hogy mindkét gyorstárazási stratégiát hatástalanítják a *Gyorstár-vezérlés* fejlécben szállított direktívák. Ezek a direktívák korlátozhatják (például *no-cache*) a gyorstárazást, amikor az nem alkalmazható. Ilyen például egy dinamikus oldal, ami a következő lekérés alkalmával eltérő lesz. A hitelesítést igénylő oldalakat sem tartják a gyorstárban.

A gyorstárazás sokkal összetettebb ennél, de már csak két fontos pontnak van helyünk. Először is, a gyorstárazást a böngészőn kívül, más helyeken is elvégezhetik. Általános esetben a HTTP-kérések útvonala keresztülmehet egy sor gyorstáron. A böngészőn kívüli gyorstárak használatát **helyettes gyorstárazásnak** (**proxy caching**) nevezzük. A gyorstárazásnak minden egyes újabb szintje segíthet a lánc további részeire eljutó kérések számának csökkentésében. Az olyan szervezeteknél, mint amilyenek az internetszolgáltatók (ISP-k) vagy a vállalatok, megszokott a helyettes gyorstárazás használata, hogy kiaknázzák az oldalak különböző felhasználók közti gyorstárazásából származó előnyöket. A helyettes gyorstárazást a tartalomelosztás szélesebb témakörén belül fogjuk megtárgyalni a 7.5 szakaszban, ennek a fejezetnek a végén.

Másodszor, a gyorstárazások jelentősen növelhetik a teljesítőképességet, de nem annyira, mint azt egyesek remélik. Ennek az az oka, hogy míg vannak nyilvánvalóan népszerű dokumentumok a világhálón, nagyon sok olyan népszerűtlen dokumentum is van, amit lekérnek az emberek, és ezek között sok még ráadásul nagyon hosszú is (például videók). A népszerűtlen dokumentumok „hosszú farka" elfoglalja a helyet a gyorstárban, és a gyorstárból kiszolgálható kérések száma csak lassan növekszik a gyorstár méretével. A webes gyorstárak mindig valószínűleg a kéréseknek csak kevesebb, mint felét képesek lekezelni. További információért lásd Breslau és mások [1999] munkáját.

Kísérletezés a HTTP-vel

Mivel a HTTP egy ASCII-protokoll, ezért meglehetősen egyszerűen lehet egy terminálról (nem pedig böngészőből) személyesen és közvetlenül a webszerverrel beszélni. Ehhez nem kell más, csak egy TCP-összeköttetés a kiszolgáló 80-as portjával. Javasoljuk, hogy az Olvasó személyesen is kísérletezzen a következő parancssorozattal. A legtöbb UNIX-héjprogramban és Windowson a parancsablakban működni fog (ha a telnet program engedélyezett).

```
telnet www.ietf.org 80
GET /rfc.html HTTP/1.1
Host: www.ietf.org
```

Ez a parancssorozat egy telnet (azaz TCP-) összeköttetést kezdeményez az IETF webszerverének 80-as portjával a *www.ietf.org* címen. Ezután következik a *GET* parancs, ami megnevezi az URL útját és a protokollt. Próbáljon ki Ön által választott kiszolgálókat és URL-eket! A következő sor a kötelező *Hoszt* fejléc. Az utolsó fejlécet követő üres sorra szintén szükség van. Ez mondja meg a kiszolgálónak, hogy nincs több kérésfejléc. A kiszolgáló ezután elküldi a választ. A kiszolgálótól és az URL-től függően számos különféle fejléc és oldal figyelhető meg.

7.3.5. A mobilweb

A világhálót a legkülönfélébb számítógépekről használják, és ebbe beletartoznak a mobiltelefonok is. A világháló mozgás közben vezeték nélküli hálózaton keresztül történő böngészése nagyon hasznos lehet. Ez számos technikai problémát vet fel, mert sok webes tartalmat széles sávú összeköttetéssel rendelkező asztali számítógépeken bemutatva szemet gyönyörkedtető megjelenésűre terveztek. Ebben a szakaszban bemutatjuk, hogy a világháló elérése mobil készülékekről, vagyis a **mobilweb** (**mobile Web**) hogyan alakult ki.

A munkahelyi vagy otthoni asztali számítógépekkel összehasonlítva a mobiltelefonok számos nehézséget jelentenek a webböngészés szempontjából:

1. A relatív kis kijelzők eleve kizárják a nagy oldalakat és nagy képeket.
2. A korlátozott beviteli képességek fárasztóvá teszik az URL-ek és más terjengős bemenetek megadását.
3. A hálózati sávszélesség korlátozott a vezeték nélküli kapcsolatokon, különösen a mobilhálózatokon (3G), ami gyakran drága is.
4. A kapcsolat időszakos lehet.
5. A számítási teljesítmény korlátozott az akkumulátoros üzemidő, méret, hőtermelődés és költség miatt.

Ezek a nehézségek azt jelentik, hogy az asztali tartalom egyszerűen a mobilweben használva valószínűleg kiábrándító felhasználói élményt ad.

A mobilweb korai megközelítése kialakított egy új, korlátozott képességű vezetékmentes eszközökhöz igazított protokollkészletet. Ennek a stratégiának a legismertebb példája a **WAP** (**Wireless Application Protocol – vezeték nélküli alkalmazási protokoll**). A WAP-pal kapcsolatos erőfeszítéseket 1997-ben kezdték meg a nagy mobiltelefon-forgalmazók, köztük a Nokia, az Ericsson és a Motorola. Közben azonban történt valami váratlan dolog. A következő évtized során a hálózati sávszélesség és a készülékek képességei elképesztő mértékben megnövekedtek a 3G-adatszolgáltatások és a nagyobb, színes kijelzős mobiltelefonok, a gyorsabb processzorok, valamint a 802.11 vezeték nélküli képességek használatba vételével. A mobiltelefonok hirtelen képessé váltak egyszerű webböngészők futtatására. Még mindig van egy rés az ilyen mobiltelefonok és az asztali gépek között, ami sosem fog bezárulni, de sok olyan technikai probléma, ami ösztönözte egy önálló protokollkészlet létrehozását, elhalványult.

Az egyre növekvő mértékben terjedő megközelítés az, hogy ugyanazokat a web protokollokat kell alkalmazni a mobiltelefonokon és az asztali gépeken, és a webhelyeknek mobiltelefon-barát tartalmat kell biztosítaniuk, amikor a felhasználó történetesen egy mobil eszközt használ. A webszerverek képesek a kérés fejlécét megtekintve megállapítani, hogy a weboldalaknak a mobil vagy az asztali változatát kell-e visszaadniuk. A *Felhasználói-ügynök* fejrész ebben a tekintetben különösen hasznos, mert azonosítja a böngészőszoftvert. Tehát amikor a webszerver megkap egy kérést, megnézi a fejléceket, és iPhone-ra egy kis képeket, kevesebb szöveget és egyszerűbb navigációt tartalmazó oldalt ad vissza, a laptopot használó felhasználónak pedig egy teljes értékű oldalt.

A W3C többféle módon is bátorítja ezt a megközelítést. Az egyik mód a mobilwebes tartalmak bevett gyakorlatának szabványosítása. Az első előírás egy 60 elemű, a bevett gyakorlatokat tartalmazó listát bocsát rendelkezésre [Rabin és McCathieNevile, 2008]. Ezeknek a legnagyobb része érzékelhető lépéseket tesz az oldalak méretének csökkentésére tömörítéssel, mivel a kommunikáció költségei magasabbak, mint a számításoké, valamint a gyorstárazás hatékonyságának maximalizálásával. Ez a megközelítés arra bátorítja a webhelyeket, különösen a nagy webhelyeket, hogy hozzák létre tartalmuknak mobilweb-változatait, mert ez az, amire szükség van ahhoz, hogy megragadják a mobilweb felhasználóinak figyelmét. Ezeknek a felhasználóknak további segítséget nyújt egy embléma, ami azokat az oldalakat jelöli, amelyek (jól) megtekinthetők a mobilweben.

Egy másik hasznos eszköz a HTML lecsupaszított változata, amit **XHTML Basic**-nek neveznek. Ez a nyelv az XHTML részhalmaza, ami olyan készülékeket célzott meg, mint amilyenek a mobiltelefonok, televíziók, PDA-k, árusítóautomaták, személyhívók, autók, játékgépek, sőt akár a karórák is. Ebből kifolyólag nem használhatók a stíluslapok, szkriptek vagy keretek, de a legtöbb szabványos címke igen. Ez utóbbiakat 11 modulba csoportosították. Egyesek kötelezők, mások opcionálisak, de mindegyiküket XML-ben definiálták. A modulokat, néhány példával együtt a 7.41. ábra sorolja fel.

Nem minden oldalt fognak azonban úgy megtervezni, hogy jól működjön a mobilweben. Ezért egy kiegészítő megközelítést használnak, ez a **tartalomátalakítás** (**content transformation**) vagy **átkódolás** (**transcoding**). Ebben a megközelítésben egy számítógép, ami a mobiltelefon és a kiszolgáló között helyezkedik el, fogadja a kéréseket a mobiltelefontól, lekéri a tartalmat a szervertől, és átalakítja azt mobilweb-tartalommá. Ez

Modul	Kötelező?	Funkció	Példa címkék
Szerkezet	Igen	Dokumentum szerkezete	body, head, html, title
Szöveg	Igen	Információ	br, code, dfn, em, hn, kbd, p, strong
Hiperszöveg	Igen	Hiperhivatkozások	a
Lista	Igen	Felsorolási listák	dl, dt, dd, ol, ul, li
Űrlapok	Nem	Kitölthető űrlapok	form, input, label, option, textarea
Táblázatok	Nem	Négyszögletes táblázatok	caption, table, td, th, tr
Kép	Nem	Képek	img
Objektum	Nem	Kisalkalmazások, térképek stb.	object, param
Metainformáció	Nem	Kiegészítő információ	meta
Hivatkozás	Nem	Hasonló az <a>-hoz	link
Alap	Nem	URL-kezdőpont	base

7.41. ábra. *Az XHTML Basic moduljai és címkéi*

egy egyszerű átalakítás abból a célból, hogy a nagy képek méretét csökkentse egy kisebb felbontásra történő átformázással. Sok más kicsi, de hasznos átalakítás lehetséges. Az átkódolást sikerrel használják a mobilweb kezdete óta. Lásd például Fox és mások [1996] művét. Amikor azonban mindkét megközelítést használják, némi feszültség keletkezik a kiszolgáló és az átkódoló által meghozott, mobiltartalommal kapcsolatos döntéseknél. Például egy webhely kiválaszthatja a képnek és szövegnek egy bizonyos összeállítását a mobilweb-felhasználó számára csak azért, hogy az átkódolónak meg kelljen változtatnia a kép formátumát.

Tárgyalásunk egészen idáig a tartalomról szólt, nem a protokollokról, mivel a tartalom a legnagyobb probléma a mobilweb megvalósításánál. Röviden megemlítjük azonban a protokollok kérdését is. A világháló által használt HTTP-, TCP- és IP-protokollok esetén a sávszélesség jelentős mennyiségét a protokollok által okozott többletterhelés, például fejlécek emésztik fel. Ennek a problémának a megoldására a WAP és más megoldások különleges célú protokollokat határoztak meg. Kiderült, hogy ez nagyrészt szükségtelen. Az olyan fejléctömörítő technikák, mint például a 6. fejezetben ismertetett ROHC (RObust Header Compression – robusztus fejléctömörítés) képes csökkenteni ezeknek a protokolloknak többletterhét. Ily módon lehetséges, hogy egyetlen protokollkészlet létezik (HTTP, TCP, IP), amely nagy sávszélességű és kis sávszélességű összeköttetésekkel is használható. A kis sávszélességű összeköttetésekkel történő használathoz csak arra van szükség, hogy a fejléc tömörítése be legyen kapcsolva.

7.3.6. Webes keresés

A világhálóról szóló ismertetőnk befejezéseként megtárgyaljuk a vitathatatlanul legsikeresebb webalkalmazást, a webes keresőt. 1998-ban Sergey Brin és Larry Page egyetemi hallgatók, miután végeztek a Stanfordon, megalapították a Google-t, hogy létrehozzanak egy jobb webes keresőmotort. Volt egy akkoriban radikálisnak számító ötletük, miszerint az a keresőalgoritmus, amelyik azt számolja meg, hogy az egyes oldalakra hány másik oldal mutat, jobban méri az oldalak fontosságát, mint az, amelyik azt számolja, hogy a keresett kulcsszót hányszor tartalmazta az oldal. Például sok oldal hivatkozik a Cisco főoldalára, ami ezt az oldalt a „Cisco"-ra kereső felhasználók számára fontosabbá teszi, mint egy vállalaton kívüli oldalt, amelyik történetesen sokszor használja a „Cisco" szót.

Igazuk volt. Beigazolódott, hogy lehet jobb keresőmotort készíteni, és az emberek özönlöttek a keresőjükhöz. A Google, kockázati tőkével a háta mögött, rettenetesen megnőtt. 2004-ben nyilvánosan működő részvénytársasággá vált, 23 milliárd $ piaci tőkével. Becslések szerint 2010-re a világ adatközpontjaiban több mint egymillió kiszolgálót üzemeltettek.

Bizonyos értelemben a kereső egyszerűen egy másik webalkalmazás, ha nem az egyik legjobban kifejlett webalkalmazás, mert a világháló kezdete óta fejlesztés alatt áll. A webes kereső azonban elengedhetetlen a mindennapos használat során. A becslések szerint több mint egymilliárd keresést végeznek naponta. A mindenféle információt kereső emberek kiindulási pontként használják a keresőszolgáltatást. Például, ha azt szeretné megtudni, hogy hol vásárolhat okostelefont Seattle-ben, nincs olyan kézenfekvő webhely, amelyet kiindulási pontként használhatna. De van esély rá, hogy a keresőmotor ismer egy, a keresett információt tartalmazó oldalt, és gyorsan a megoldás felé irányítja.

Egy hagyományos módon végzett kereséshez a felhasználó a webes kereső webhelyének URL-jére irányítja böngészőjét. A fontosabb keresők közé tartozik a Google, a Yahoo! és a Bing. Ezt követően a felhasználó egy űrlap segítségével elküldi a keresőkifejezéseket. Ez a tevékenység azt okozza, hogy a keresőmotor lekérdezi az adatbázisában lévő releváns oldalakat, képeket vagy bármilyen más keresett erőforrást, és az eredményt egy dinamikus oldal formájában visszaadja. A felhasználó ezután követheti a megtalált oldalakra mutató hivatkozásokat.

A webes keresés érdekes vitatéma, mert ez kihat a hálózatok tervezésére és használatára. Először is felmerül a kérdés, hogyan találja meg a webes kereső az oldalakat. A webes kereső motorjának a lekérdezés végrehajtásához rendelkeznie kell az oldalak adatbázisával. Minden egyes HTML-oldal hivatkozásokat tartalmazhat más oldalakra, és minden érdekes (vagy legalábbis kereshető) dologra hivatkoznak valahol. Ez azt jelenti, hogy elméletileg egy maréknyi oldalról elindulva meg lehet találni a világháló összes többi oldalát valamennyi oldal és hivatkozás bejárásával. Ezt a folyamatot hívják a web **bejárásának (Web crawling)**. Minden webes keresőmotor használ keresőrobotokat (Web crawler) a világháló bejárására.

A bejárással kapcsolatos egyik kérdés, hogy a keresőrobot milyen fajta oldalakat képes megtalálni. Könnyű lekérni a statikus dokumentumokat és követni a hivatkozásokat. Sok weboldal azonban programokat tartalmaz, amelyek a felhasználó tevékenységétől függően különböző oldalakat jelenítenek meg. Ilyen például egy áruház online katalógusa. Egy katalógus a termékadatbázis és különféle termékekre vonatkozó lekérdezések

alapján létrehozott dinamikus oldalakat tartalmazhat. Ez a fajta tartalom eltér a statikus oldalaktól, amelyeket könnyű bejárni. Hogyan találják meg a keresőrobotok ezeket a dinamikus oldalakat? A válasz az, hogy a nagyobb részüket sehogy. Ezt a fajta rejtett tartalmat hívják **mély webnek (deep Web)**. A mély web keresésének módja még nyitott probléma, amit a kutatók most probálnak megoldani. Lásd például Madhaven és mások [2008] munkáját. Léteznek szabályok, melyek szerint a webhelyek (*robots.txt*-ként ismert) oldalakat készítenek azzal a céllal, hogy megmondják a keresőrobotoknak, a webhely melyik részét kell vagy nem szabad meglátogatniuk.

A második szempont, hogy hogyan lehet feldolgozni valamennyi bejárt adatot. Annak érdekében, hogy az indexelő algoritmusok átfuthassák az adathalmazt, az oldalakat tárolni kell. A becslések változók, de úgy gondolják, hogy a fő keresőmotorok több tízmilliárd, a világháló látható részéből származó oldal indexével rendelkeznek. Az átlagos oldalméretet 320 KB-ra becsülik. Ezek a számok azt jelentik, hogy a világháló bejárással készített másolata 20 petabájt vagy 2×10^{16} bájt nagyságrendű tárhelyet igényel. Miközben ez valóban óriási szám, egyszesmind akkora mennyiségű adat, amit az internetes adatközpontokban kényelmesen lehet tárolni és feldolgozni [Chang és mások, 2006]. Például, ha egy lemezes tár 20 $-ba kerül terabájtonként, akkor 2×10^4 TB 400 000 $-ba kerül, ami nem éppen óriási összeg olyan méretű vállalatoknak, mint a Google, a Microsoft és a Yahoo!. És amíg a világháló terjeszkedik, a lemezek ára drámaian esik, így a teljes világháló tárolása a belátható jövőben továbbra is megvalósítható lehet a nagyvállalatok számára.

Ezeknek az adatoknak a megértése egy másik kérdés. Értékelni fogja az olvasó, ahogyan az XML képes segíteni a programoknak könnyedén kinyerni az adatok szerkezetét, miközben az ad hoc formátumok sok találgatáshoz fognak vezetni. A formátumok közötti átalakítás és a nyelvek közötti fordítás is kérdés. De az adatok szerkezetének ismerete is csak egy része a problémának. A kemény dió annak megértése, hogy ez mit jelent. Ez az, ahol sok érték napvilágot láthat, kezdve a keresésekre válaszként adott relevánsabb oldalakkal. A végső cél az, hogy képes legyen válaszolni arra a kérdésre, hogy például hol lehet olcsó, de jó minőségű kenyérpirítót venni az Ön városában.

A webes keresés harmadik aspektusa az, hogy az elnevezéseknek egy magasabb szintjét nyújtja. Nincs szükség egy hosszú URL megjegyzésére, ha éppen olyan (vagy talán még inkább) megbízható megkeresni egy weboldalt valaki neve alapján, feltéve, hogy Ön jobban meg tudja jegyezni a neveket, mint az URL-eket. Ez a stratégia egyre inkább sikeres. Ugyanúgy, ahogyan a DNS-nevek számítógépekre vezették vissza az IP-címeket, a webes keresés is számítógépekre vezeti vissza az URL-eket. Szintén a keresés mellett szól, hogy kijavítja a helyesírási és gépelési hibákat, míg ha Ön rosszul gépel be egy URL-t, rossz oldalt kap.

Végül, a webes keresés megmutat nekünk valamit, aminek nem sok köze van a hálózattervezéshez, de annál több van néhány internetes szolgáltatás fejlődéséhez: sok pénz van a hirdetésben. A hirdetés az a gazdasági motor, ami a webes keresés növekedését hajtotta. A nyomtatott hirdetéshez képest jelentős változás az a lehetőség, hogy a hirdetés fontosságának növelése érdekében az embereket a keresésük tárgyának függvényében lehet a hirdetésekkel megcélozni. A keresési lekérdezésnek megfelelő legértékesebb hirdetés megtalálására az árverési mechanizmus változatait használják [Edelman és mások, 2007]. Ez a modell természetesen új problémákat eredményezett, mint amilyen például a **rosszindulatú kattintások (click fraud)**, amelynek során a programok utánozzák a felhasználókat és hirdetésekre kattintanak rá, meg nem érdemelt kifizetések előidézése érdekében.

7.4. Hang és mozgókép folyamszerű átvitele

A webalkalmazások és a mobilweb a hálózatok felhasználásában nem az egyedüli izgalmas fejlesztés. Sokak számára a multimédia jelenti a számítógépes hálózatok csúcsát. Amikor ez szóba kerül, mind a kockafejűeknek, mind az öltönyös alakoknak mintegy varázsütésre felcsillan a szemük. Az előbbiek hatalmas műszaki kihívásokat látnak abban, hogy minden számítógépen lehetővé tegyék az IP-hálózaton keresztül történő beszédátvitelt és a hálózati videózást. Az utóbbiak ugyanilyen hatalmas profitot látnak benne.

Míg az interneten keresztül történő hang és mozgókép küldésének ötlete az 1970-es években vagy még korábban megszületett, a **valós idejű hang és mozgókép** átvitel forgalma csak nagyjából 2000 óta növekedett meg, talán túlságosan is. A valós idejű forgalom abban különbözik a webes forgalomtól, hogy azt egy bizonyos előre meghatározott sebességgel kell lejátszani, hogy használható legyen. Elvégre a legtöbb embernek nem az az elképzelése a szórakozásról, hogy egy megakadó és elinduló lassított mozgóképet néz. Ezzel ellentétben a világhálón előfordulhatnak rövid megszakítások, az oldalbetöltések bizonyos kereteken belül több-kevesebb időt vehetnek igénybe anélkül, hogy ez komoly gond lenne.

Két dolog történt, ami lehetővé tette ezt a növekedést. Először is, a számítógépek sokkal nagyobb teljesítményűvé váltak, valamint felszerelték azokat mikrofonokkal és kamerákkal. Így könnyedén képesek a hangot és a mozgóképet beolvasni, feldolgozni és kiadni. Másodszor, lehetővé vált az internet sávszélességének jelentős növelése. Az internet belsejének hosszú távú összeköttetései több gigabit/s sebességgel működnek, a széles sávú és 802.11 vezeték nélküli szolgáltatás pedig eléri az internet szélén lévő felhasználókat. Ezek a fejlesztések lehetővé tették az internetszolgáltatók (ISP-k) számára, hogy hatalmas forgalmat bonyolítsanak le a gerinchálózatukon, ami azt jelenti, hogy az átlagos felhasználó így százszor-ezerszer gyorsabban fér hozzá az internethez, mint egy 56 kb/s sebességű telefonos modemmel.

A sávszélesség jelentős növekedése idézte elő a hang- és videoátviteli forgalom növekedését, de az okok eltérőek. A telefonhívások aránylag kis sávszélességet igényelnek (elvileg 64 kb/s-ot, tömörítéskor kevesebbet), de a telefonszolgáltatás hagyományosan drága. A vállalatok lehetőséget láttak arra, hogy a telefonszámláik csökkentése érdekében, a meglévő sávszélességet felhasználva a beszédforgalmat interneten keresztül továbbítsák. Az első ilyen cégek, mint például a Skype, megtalálták a módját annak, ahogyan lehetővé tehetik ügyfeleik számára az ingyenes telefonálást internet-hozzáférésük felhasználásával. A hirtelen meggazdagodott telefontársaságok meglátták azt, hogyan lehet a hagyományos hanghívásokat IP-hálózati berendezések felhasználásával olcsón továbbítani. Ennek eredménye az internethálózaton továbbított hangadatok robbanásszerű növekedése lett, amit **IP-hálózaton keresztül történő beszédátvitelnek (voice over IP, VoIP)** vagy **internetes telefonálásnak (Internet telephony)** neveznek.

A hangátvitellel ellentétben a mozgókép (videó) átvitele nagy sávszélességet igényel. Az elfogadható minőségű internetes videókat 1 Mb/s körüli adatsebességgel tömörítik, és egy tipikus DVD 2 GB adatot tartalmaz.[2] A széles sávú internet-hozzáférés előtt a

2 Jelenleg (2011-ben) a legnagyobb kapacitású DVD-18 már 17 GB adatot tartalmaz. Tipikus inkább a DVD-5, amely 4,7 GB adatot tartalmaz. *(A fordító megjegyzése)*

filmek átküldése a hálózaton megengedhetetlen volt. Többé nem az. A széles sávú elérések elterjedésével először vált lehetővé a felhasználók számára, hogy a körülményekhez képest elfogadható minőségű, folyamszerűen letöltött (streaming) videót nézzenek az otthonukban. Az emberek szeretik ezt. Becslések szerint az internet felhasználóinak körülbelül a negyede mindennap meglátogatja a YouTube-ot, a népszerű videomegosztó helyet. A mozifilmek kölcsönzésével kapcsolatos üzlet eltolódott az online letöltések irányába. Az internetre feltöltött döbbenetes mennyiségű videó megváltoztatta az internet forgalmának teljes összetételét. Az internetes forgalom többségét ma már a videó adja, és úgy becsülik, hogy néhány éven belül az internet forgalmának 90%-a videoforgalom lesz [Cisco, 2010].

Tekintettel arra, hogy a hang- és videoátvitelhez elegendő a sávszélesség, a folyamszerű átvitelt és konferenciaszolgáltatásokat megvalósító alkalmazások tervezése szempontjából a hálózati késleltetés a kulcsfontosságú kérdés. A hang és a mozgókép valós idejű megjelenítést igényel, ami azt jelenti, hogy előre meghatározott sebességgel kell lejátszani ezeket ahhoz, hogy használhatók legyenek. A hosszú késleltetés azt jelenti, hogy a hívások, amelyeknek interaktívnak kellene lenniük, többé nem azok. Ez a probléma eléggé nyilvánvaló, ha Ön valaha is beszélt már műholdas telefonon, ahol ez az akár fél másodperces késleltetés meglehetősen zavaró. Zene és filmek hálózaton keresztüli lejátszásánál az abszolút késleltetés nem számít, mert annak csak akkor van hatása, amikor a média lejátszása megkezdődik. De a késleltetés változása (szórása), amit **dzsitternek** (**jitter**) neveznek, gondot okoz. Ezt el kell fednie a lejátszónak, különben a hang érthetetlen, a videó pedig szaggatott lesz.

Ebben a szakaszban meg fogunk tárgyalni néhány, a késleltetés problémáját kezelő stratégiát, valamint a hang- és video-munkamenetek létrehozására szolgáló protokollt. A digitális hang és mozgókép témakörébe történő bevezetés után ismertetőnk három eset tárgyalására tagolódik, melyek közül mindegyikben eltérő konstrukciót használnak. Az első és legkönnyebb eset a tárolt média folyamszerű letöltése, mint amilyen egy videó megtekintése a YouTube-on. A nehézség szempontjából a következő eset az élő médiaközvetítés. Ennek két példája az internetes rádió és az IPTV, amelyekkel a rádió- és televízióállomások sok felhasználó számára élőben sugározzák műsorukat az interneten. Az utolsó és legbonyolultabb eset a telefonbeszélgetések lebonyolítása, amit Skype-pal lehetne megtenni, vagy általában az interaktív audio- és videokonferencia.

Mellesleg, a **multimédia** (**multimedia**) kifejezést az internettel kapcsolatban gyakran használják a hangra és a mozgóképre. Szó szerint értelmezve, a multimédia két vagy több médiumot jelent. E definíció szerint ez a könyv egy multimédia-bemutató, mert szöveget és grafikát (ábrákat) tartalmaz. Valószínűleg azonban Ön nem erre gondolt, ezért a „multimédia" kifejezés alatt két vagy több **folytonos médiát** (**continuous media**) értünk, vagyis olyan médiumokat, amelyeket valamely jól meghatározott időintervallum alatt kell lejátszani. A két médium rendszerint az audio és a video, vagyis mozgókép hanggal. Sokan a puszta hangátvitelre, például az internettelefóniára vagy az internetes rádiózásra is multimédiaként hivatkoznak, pedig itt nyilván nem erről van szó. Valójában minden ilyen esetben helyesebb lenne a **folyamszerű letöltésű média** (**streaming media**) kifejezés használata. Mindazonáltal mi is követjük a nyájat, és a valós idejű hangátvitelt is multimédiának fogjuk tekinteni.

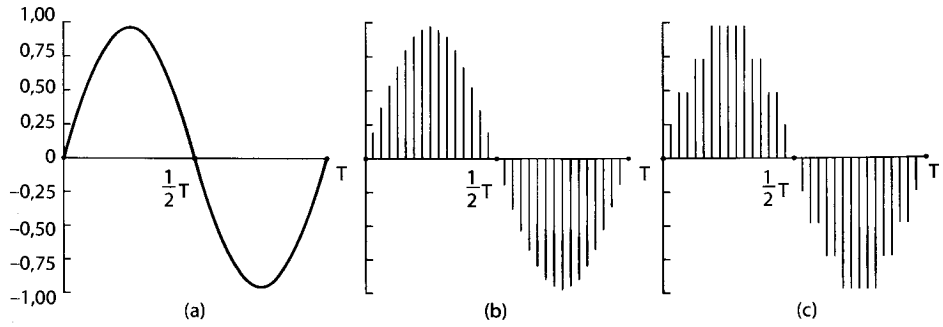
7.4.1. Digitális hang

A hanghullám egy egydimenziós akusztikai (nyomás) hullám. Amikor egy akusztikai hullám belép a fülbe, a dobhártya elkezd rezegni, erre a belsőfül kicsiny csontjai is elkezdenek vele együtt rezegni, és ingerimpulzusokat küldenek az agyba. Ezeket az impulzusokat a hallgató hangként érzékeli. Hasonló módon, amikor egy akusztikai hullám elér egy mikrofont, a mikrofon egy villamos jelet állít elő, amely a hang amplitúdóját jellemzi az idő függvényében.

Az emberi fül által hallható hangok frekvenciatartománya 20 Hertztől 20 000 Hertzig terjed, bár néhány állat, főképpen a kutyák, képesek magasabb frekvenciákat is meghallani. A fül logaritmikusan érzékeli a hangerőt, így két A és B amplitúdójú hang arányát szokásosan **dB**-ben (**decibel**) fejezik ki, amely a $10 \log_{10}(A/B)$ összefüggéssel határozható meg. Ha egy 1 kHz-es szinuszhullám hallhatóságának alsó határát (amely kb. 20 μPa hangnyomásnak felel meg) 0 dB-nek definiáljuk, akkor egy szokásos beszélgetés 50 dB körül van, a fájdalomküszöb pedig 120 dB. A dinamikatartomány több mint egymilliószoros.

A fül meglepően érzékeny az akár csak pár ezredmásodpercig tartó hangváltozásokra is. A szem, ezzel ellentétben, nem veszi észre a fényerősség olyan változásait, amelyek csak pár ezredmásodpercig tartanak. Ennek a megfigyelésnek az eredménye az, hogy egy multimédia lejátszásnál bekövetkező pár ezredmásodperces dzsitter az észlelt hangminőséget sokkal jobban befolyásolja, mint az észlelt képminőséget.

A digitális hang egy hanghullám digitális ábrázolása, ami felhasználható a hang újbóli előállításához. A hanghullámokat egy **ADC** (**Analog Digital Converter – analóg-digitális átalakító**) segítségével lehet digitális formára alakítani. Az ADC villamos feszültséget kap a bemenetén, és a kimenetén bináris számokat állít elő. A 7.42.(a) ábrán látható egy szinuszhullám példája. Hogy digitálisan kezelhessük ezt a jelet, minden ΔT időközönként mintát veszünk belőle, ahogy a 7.42.(b) ábrán az oszlopok mutatják. Ha egy hanghullám nem tiszta szinuszhullám, de szinuszhullámok lineáris szuperpozíciója, amelyek közt a legmagasabb frekvenciájú f, akkor a Nyquist-tétel értelmében (lásd 2. fejezet) elegendő $2f$ frekvenciával mintákat venni. Gyakrabban mintavételezni nincs értelme, mivel azok a magasabb frekvenciák, amelyeket egy ilyen mintavételezés észlelni tudna, nincsenek jelen.



7.42. ábra. *(a) Egy szinuszhullám. (b) A szinuszhullám mintavételezettje. (c) A minták 3 bitre történt kvantáltja*

A fordított folyamat digitális értékeket használ fel, és analóg villamos feszültséget állít elő. Ezt a feladatot a **DAC (Digital-to-Analog Converter – digitális-analóg átalakító)** végzi el. Az analóg villamos feszültséget azután a hangszóró hanghullámokká alakítja, így az emberek hallhatják a hangokat.

A digitális minták soha nem teljesen pontosak. A 7.42.(c) ábra mintái csak nyolc értéket tesznek lehetővé –1,00 és +1,00 közt, 0,25-ös lépésekben. Egy 8 bites minta 256 különböző értéket tenne lehetővé. Egy 16 bites minta 65 536 különböző értéket tenne lehetővé. A mintánkénti véges számú bit által behozott hibát **kvantálási zajnak (quantization noise)** nevezik. Ha ez túl nagy, a fül ezt érzékeli.

A mintavételezett hangok két jól ismert példája a telefon és a hang-CD-k. A pulzuskód-moduláció, amelyet a telefonrendszerben használnak, 8 bites mintákkal dolgozik, és másodpercenként 8000 mintát vesz. A skála az érzékelhető torzítás csökkentése érdekében nem lineáris, és a másodpercenkénti mindössze 8000 minta miatt a 4 kHz fölötti hangok elvesznek. Észak-Amerikában és Japánban a **μ-law**, Európában és nemzetközileg az **A-law** kódolást használják. Mindkét kódolás 64 000 b/s adatsebességet eredményez.

A hang-CD-k digitálisak, másodpercenként 44 100 mintával, amely elég ahhoz, hogy 22 050 Hz-ig visszaadja a hangokat, amely az emberek számára jó, de a zenerajongó kutyák számára rossz. Mindegyik minta 16 bites, és az amplitúdótartományon belül lineáris. Vegyük észre, hogy a 16 bites minták csak 65 536 különböző értéket tesznek lehetővé, bár a fül dinamikatartománya 1 millió fölötti. Tehát annak ellenére, hogy a CD-minőségű hang sokkal jobb a telefonminőségűnél, mintánként csak 16 bit használata észrevehető kvantálási zajt okoz (bár nincs lefedve a teljes dinamikatartomány, a CD-knek nem szabad fájdalmat okozniuk). Néhány fanatikus audiofil még mindig előnyben részesíti a 33-as fordulatú bakelitlemezeket a CD-kkel szemben, mert ezeknél nem jelentkezik a Nyquist frekvencialevágás 22 kHz-nél, és nincs kvantálási zaj sem. (Azonban sercegnek, ha nem kezelik őket rendkívül óvatosan.) Másodpercenként 44 100 16 bites minta esetén a tömörítetlen CD-minőségű hangnak 705,6 kb/s (mono) vagy 1,411 Mb/s (sztereo) sávszélességre van szüksége.

Hangtömörítés

Annak ellenére, hogy a hangátvitelhez szükséges sebesség sokkal kisebb, mint a mozgóképátvitelhez szükséges sebesség, a hangokat gyakran tömörítik a sávszélességigény és az átviteli idő csökkentése érdekében. Minden tömörítő rendszernek két algoritmusra van szüksége: az egyik a forrásnál tömöríti össze az adatokat, a másik pedig a célban kibontja azokat. Az irodalomban ezekre mint **kódoló (encoding)** és **dekódoló (decoding)** algoritmusokra hivatkoznak. Mi is ezt a terminológiát fogjuk használni.

A tömörítő algoritmusok bizonyos aszimmetriákat mutatnak, amelyeket fontos megérteni. Annak ellenére, hogy először a hangokkal foglalkozunk, ezek az aszimmetriák a videókra is érvényesek. Sok alkalmazás esetén egy multimédiás dokumentumot csak egyszer kódolják (a multimédiakiszolgálón történő tároláskor), de több ezerszer dekódolják (amikor az ügyfelek lejátsszák). Ez az aszimmetria azt jelenti, hogy a kódoló algoritmus lehet lassú, és igényelhet drága hardvert, feltéve, hogy a dekódoló algoritmus gyors és nem igényel drága hardvert. Egy népszerű hang (vagy videó) kiszolgáló

üzemeltetője lehet, hogy nagyon szívesen vásárol egy több számítógépből álló fürtöt (clustert) teljes könyvtárának kódolásához, de nem valószínű, hogy nagy sikert aratna, ha ugyanezt követelné meg az ügyfelektől a zene hallgatásához vagy a film nézéséhez. Sok gyakorlati tömörítő rendszer sokat megtesz azért, hogy a dekódolást gyorssá és egyszerűvé tegye, még annak árán is, hogy a kódolás lassú és komplikált lesz.

Másrészről viszont az élő hang és videó esetében, mint amilyen például az IP-hálózaton keresztül történő beszédhívás, a lassú kódolás elfogadhatatlan. A kódolásnak röptében, valós időben kell megtörténnie. Következésképpen a valós idejű multimédia más algoritmusokat vagy paramétereket használ, mint a filmek lemezen való tárolása, gyakran jelentősen kisebb tömörítéssel.

Egy másik aszimmetria, hogy a kódolási/dekódolási folyamatnak nem kell visszafordíthatónak lennie. Egy adatállomány tömörítése, átvitele és visszaállítása után a felhasználó arra számít, hogy az eredetivel az utolsó bitig megegyező eredményt kap vissza. A multimédiában ez a követelmény nem létezik. Általában elfogadható, ha a hang (vagy videó) jel a kódolás és az azt követő dekódolás után csak kissé tér el az eredetitől mindaddig, amíg ugyanolyannak hangzik (látszik). Amikor a dekódolt kimenet nem teljesen azonos az eredeti bemenettel, a rendszert **veszteségesnek (lossy)** nevezzük. Ha a bemenet és a kimenet megegyeznek, a rendszer **veszteségmentes (lossless)**. A veszteséges rendszerek fontosak, mivel egy kismértékű információvesztést elfogadva általában hatalmas nyereség elérhető el a tömörítési arányban.

Régen, a telefonhálózatokban, a nagy távolságon biztosított sávszélesség nagyon drága volt, ezért jelentős munka hárult a **vokóderekre (vocoder**, ami a „vocal coder", azaz a hangtömörítő rövidítése), amelyek kimondottan emberi beszédhangokat tömörítenek. Az emberi beszéd általában 600 és 6000 Hz közé esik, és olyan mechanikai folyamat állítja elő, ami a beszélő hangképző szervétől, nyelvétől és állkapcsától függ. Néhány vokóder a hangképző rendszer modelljét használja, hogy a beszédet mindössze pár paraméterre sűrítse össze (a különféle üregek méretére és alakjára), az adatátviteli sebességet pedig mindössze 2,4 kb/s-ra csökkentse. Az ilyen vokóderek működése azonban túlmutat könyvünk keretein.

Mi az interneten keresztül küldött hangokra fogunk koncentrálni, ami általában közel CD-minőségű. Az ilyen hangok adatátviteli sebességét is kívánatos volna csökkenteni. A sztereó hang 1,411 Mb/s sebessége sok széles sávú adatkapcsolatot foglalna le, kevesebb helyet hagyva a videóknak és más webes forgalomnak. Ennek adatátviteli sebessége tömörítéssel egy nagyságrenddel csökkenthető, észrevehetelen vagy alig észrevehető minőségromlás mellett.

A tömörítéshez és kibontáshoz jelfeldolgozás szükséges. Szerencsére a digitalizált hang és filmek a számítógépes szoftverrel könnyen feldolgozhatók. Valójában több tucat olyan program létezik személyi számítógépekre, amelyek lehetővé teszik, hogy a felhasználók különböző forrásokból származó médiát rögzítsenek, megjelenítsenek, szerkesszenek, keverjenek és eltároljanak. Ez vezetett oda, hogy nagy mennyiségű zene és film érhető el az interneten – nem mindegyik legálisan –, ez az oka annak, hogy a művészek és a szerzőijog-tulajdonosok számos bírósági pert kezdeményeztek.

Számos hangtömörítő algoritmust fejlesztettek ki. A legnépszerűbb formátumok valószínűleg az **MP3 (MPEG audio layer 3 – MPEG-hangréteg 3)** és az **MP4 (MPEG-4)** állományokban alkalmazott **AAC (Advanced Audio Coding – fejlett hangkódolás)**.

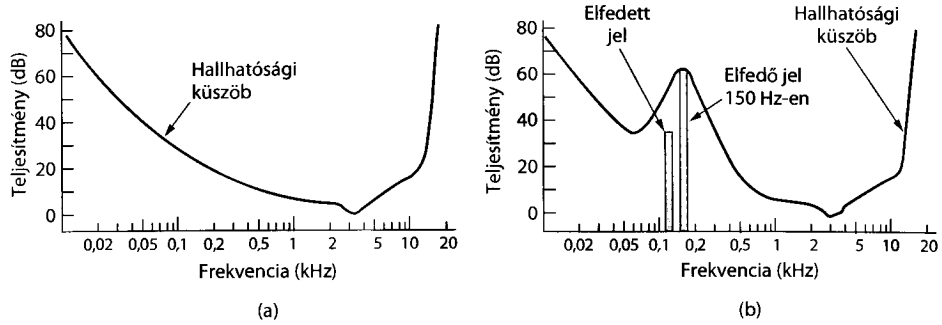
A félreértés elkerülése végett jegyezzük meg, hogy az MPEG hang- és videotömörítést is biztosít. Az MP3 az MPEG-1 szabvány hangtömörítő részére (3. rész) utal, nem pedig az MPEG harmadik változatára. Az MPEG harmadik változatát valójában nem is adták ki, csak az MPEG-1-et, MPEG-2-t és az MPEG-4-et. Az AAC az MP3 utóda, és az MPEG-4 alapértelmezett hangkódolása. Az MPEG-2 megengedi mind az MP3, mind az AAC használatát is. Világos most már? Az a szép a szabványokban, hogy oly sok közül lehet választani. És ha valakinek ezek közül valamelyik nem tetszik, csak várjon egy-két évet!

A hangtömörítést kétféleképpen lehet megoldani. A **hullámforma-kódolás** (**waveform coding**) során a jelet matematikai eszközökkel, Fourier-transzformációval a frekvenciatartományba transzformálják. A 2. fejezetben a 2.1.(a) ábrán egy időfüggvényre és a hozzá tartozó Fourier-amplitúdókra mutattunk egy példát. Ezután minden komponens amplitúdóját minimális módon kódolják. A cél az, hogy a másik oldalon elég pontosan visszaállítható legyen a hullámforma, a lehető legkevesebb bit felhasználásával.

A másik megoldás az **érzékelési kódolás** (**perceptual coding**), mely az emberi hallórendszer bizonyos hiányosságait használja ki a jelek kódolására. Ezeket a kódolt jeleket az emberi fül ugyanolyannak hallja, mint az eredeti jeleket, pedig egy oszcilloszkópon egészen más képet mutatnak. Az érzékelési kódolás a **pszichoakusztika** (**psychoacoustics**) tudományán alapszik, mely azzal foglalkozik, hogy az emberek hogyan érzékelik a hangokat. Az MP3 és az AAC is az érzékelési kódoláson alapul.

Az érzékelési kódolás kulcsa az, hogy egyes hangok **elfedhetnek** (**mask**) másokat. Képzeljük el, hogy egy élő fuvolakoncertet közvetítünk egy meleg nyári napon. Egy csapat munkás a közelben egyszer csak beindítja a légkalapácsokat, és elkezdik felbontani az utcát. Onnan kezdve a fuvolát senki nem fogja hallani – a hangját elfedik a légkalapácsok. Az átvitel szempontjából ezután elég lesz csak a légkalapácsok frekvenciasávját kódolni, mert a hallgatók úgysem fogják meghallani a fuvolát. Azt a jelenséget, amikor az egyik frekvenciasávban lévő erősebb hang képes elnyomni egy másik frekvenciasávban lévő halkabb hangot, mely az erősebb hang hiányában amúgy hallható lenne, **frekvenciaelfedésnek** (**frequency masking**) nevezik. A fuvola valójában még a légkalapácsok leállítása után sem fog hallatszani egy kis ideig, mert a fül érzékenysége csökken, amikor a légkalapácsok beindulnak, és utána eltart egy bizonyos ideig, míg ez az érzékenység újra helyreáll. Ezt a jelenséget **ideiglenes elfedésnek** (**temporal masking**) nevezik.

Ahhoz, hogy ezek a hatások mennyiségileg is szemléletesebbé váljanak, képzeljük el az alábbi első kísérletet. A kísérleti alany egy csendes szobában felvesz egy fejhallgatót, melyet egy számítógép hangkártyájához kötöttek. A számítógép egy 100 Hz-es tiszta szinuszhullámot generál, először halkan, aztán fokozatosan egyre hangosabban. A kísérleti alanynak egy gombot kell megnyomnia, amikor először meghallja a hangot. A számítógép ekkor feljegyzi az aktuális hangerőt, majd megismétli a kísérletet 200 Hz-en, 300 Hz-en és így tovább, az összes frekvencián az emberi hallástartományon belül. Ha sok ember eredményeit átlagolják, akkor a 7.43.(a) ábrán láthatóhoz hasonló logaritmikus görbét kapnak, mely megmutatja, hogy az egyes hangoknak milyen hangerővel kell megszólalniuk ahhoz, hogy hallhatók legyenek. A görbe közvetlen következménye, hogy soha sincs szükség azon frekvenciák kódolására, melyek hangereje a hallhatósági küszöb alatt marad. Például, ha a 7.43.(a) ábrán látható esetben a 100 Hz-es hang hangereje 20 dB lenne, akkor azt érzékelhető minőségvesztés nélkül is kihagyhatnánk a kimenetből, mert a 20 dB 100 Hz-en a hallhatósági küszöb alatt marad.



7.43. ábra. *(a) A hallhatósági küszöb a frekvencia függvényében. (b) Az elfedés jelensége*

Tekintsünk most egy második kísérletet. A számítógép ismét az előző kísérletet futtatja, de a teszthangokra ezúttal egy másik, állandó hangerővel rendelkező, mondjuk 150 Hz-es szinuszhullámot ültet. Azt tapasztaljuk, hogy a hallhatósági küszöb a 150 Hz körüli frekvenciák esetében megemelkedik, ahogy azt a 7.43.(b) ábra mutatja.

Új megfigyelésünknek az a következménye, hogy egyre több frekvenciát hagyhatunk ki a kódolt jelből, vagyis biteket takaríthatunk meg, ha számon tartjuk, hogy milyen jeleket takarnak el a közeli frekvenciasávok erősebb jelei. A 7.43. ábrán a 125 Hz-es jelet teljesen elhagyhatjuk a kimenetből, mégse fogja meghallani a különbséget senki. Sőt az ideiglenes elfedési tulajdonságok ismeretében egy bizonyos ideig még azután is elhagyhatjuk az elfedett frekvenciákat, hogy az erősebb jel egy adott frekvenciasávban véget ért; egészen addig, míg a hallás helyre nem áll. Az MP3 lényege az, hogy a hangot Fourier-transzformálják, hogy megkapják az egyes frekvenciák teljesítményét, majd ezek közül csak a nem elfedett frekvenciákat viszik át, a lehető legkevesebb bitben kódolva.

Ezzel a háttérrel már nekivághatunk a kódolás tárgyalásának is. A hangtömörítéshez a hullámformát az AAC esetében 8 és 96 kHz közötti frekvenciával mintavételezik, a CD hangjának utánzása érdekében gyakran 44,1 kHz-en. A mintavételezés történhet egy (mono) vagy két (sztereo) csatornán. Ezt követően a kimeneti adatsebességet határozzák meg. Az MP3 segítségével egy sztereó rock and roll CD-t akár 96 kb/s adatsebességgel is be lehet tömöríteni úgy, hogy a minőségvesztést még a nem halláskárosult rajongók is alig veszik észre. Egy zongorakoncerthez ugyanakkor már legalább 128 kb/s adatsebességű AAC szükséges. A különbség abból fakad, hogy a rock and roll jel/zaj viszonya sokkal magasabb, mint a zongorakoncerté (legalábbis mérnöki értelemben). Persze kisebb kimeneti sebességet is lehet választani, ha el tudunk fogadni némi minőségromlást.

A mintákat kis kötegekben dolgozzák fel. Minden köteget egy digitális szűrősorozaton visznek át, hogy frekvenciasávokat kapjanak. A frekvenciainformációt egy pszichoakusztikus modellel dolgozzák fel, hogy meghatározzák az elfedett frekvenciákat. Ezután a rendelkezésre álló bitkészletet felosztják a sávok között, melynek során több bitet osztanak ki azoknak a sávoknak, ahol több elfedetlen spektrális teljesítmény van, kevesebb bitet azoknak az elfedetlen sávoknak, melyeknek kisebb a spektrális teljesítménye, és egyetlen bitet sem adnak az elfedett sávoknak. A biteket végül a Huffman-kódolás

segítségével kódolják, ami rövid kódszavakat rendel a gyakran előforduló számokhoz és hosszú kódszavakat a ritkábbakhoz. Sokkal több részlet is létezik a kíváncsi olvasó számára. További információért lásd Brandenburg [1999] munkáját.

7.4.2. Digitális mozgókép

Most, hogy már mindent tudunk a fülről, ideje áttérnünk a szemre. (Nyugalom, ezután már nincs egy következő fejezet az orról.) Az emberi szemnek megvan az a tulajdonsága, hogy ha egy kép jelenik meg a retinán, akkor az pár ezredmásodpercig ott is marad, mielőtt eltűnne. Ha egy képsorozatot 50 kép/másodperc sebességgel rajzolnak ki, akkor a szem nem észleli, hogy valójában különálló képeket lát. Minden videorendszer ezt az elvet használja ki a mozgóképek előállítására.

A mozgókép legegyszerűbb digitális reprezentációja képek sorozata, ahol mindegyik kép képelemek (**pixelek, képpontok**) téglalap alakú rácsából áll. Minden képpont lehet egyetlen bit, amely vagy fehéret, vagy feketét jelképez. Egy ilyen rendszer minősége azonban szörnyű. Próbálja meg kedvenc képszerkesztője segítségével egy színes kép képelemeit feketére és fehérre (és *nem* szürkeárnyalatosra) alakítani!

A következő szint az, hogy 8 bitet használjunk képpontonként, így 256 szürkeárnyalatot tudunk leírni. Ez az eljárás jó minőségű „fekete-fehér" mozgóképet eredményez. A színes mozgóképhez sok rendszer a vörös, a zöld és a kék (RGB) elsődleges színösszetevőket használja úgy, hogy mindegyikhez 8 bitet használ. Azért van lehetőség ennek az ábrázolásnak a használatára, mert bármilyen szín előállítható a megfelelő intenzitású vörös, zöld és kék színek lineáris szuperpozíciójaként. Képpontonként 24 bit használatával a színek száma körülbelül 16 millió, ami több, mint amennyit az emberi szem képes megkülönböztetni.

A színes LCD-monitorokon és -televíziókon minden diszkrét képpont egymáshoz közel elhelyezett vörös, zöld és kék pontrészből, ún. szubpixelekből (subpixel) épül fel. A képkockákat (kereteket) a szubpixelek intenzitásának beállításával jelenítik meg, és a szem összemossa a színösszetevőket.

Gyakori képkockasebességek a (35 mm-es mozifilmtől örökölt) 24 képkocka/s, (az NTSC szabványú Egyesült Államokbeli televízióktól örökölt) 30 képkocka/s, és (a csaknem a világ összes többi részén használt PAL televíziós rendszertől örökölt) 30 képkocka/s. (Az igazán igényesek számára megjegyezzük, hogy az NTSC-rendszerű színes televíziók 29,97 képkocka/s sebességgel működnek. Az eredeti, fekete-fehér rendszer 30 képkocka/s sebességgel működött, de a színek bevezetésekor a mérnököknek további egy bit sávszélességre volt szükségük a jelzésekhez, ezért lecsökkentették a képkockasebességet 29,97 képkocka/s sebességre. A számítógépekhez készített NTSC-videók valóban 30 képkocka/s-t használnak.) A PAL-t az NTSC után találták ki, és valójában 25 képkocka/s sebességet használ. Hogy teljes legyen a történet, Franciaországban, frankofon Afrikában és Kelet-Európában egy harmadik rendszert, a SECAM-ot használják. Kelet-Európában először Kelet-Németországban vezették be, így a kelet-németek nem tudták nézni a nyugat-német (PAL) televíziót. Sok ilyen ország azonban átváltott PAL-ra. Ez a legtöbb, amire a technika és a politika képes.

A sugárzott televízióadások számára valójában nem elég jó a 25 képkocka/s az egyenletes mozgáshoz, ezért a képeket két **mezőre** (**field**) bontják, az egyik a páratlan sorszámú pásztázási sorokat tartalmazza, a másik a párosakat. A két (feleakkora felbontású) mezőt egymást követően sugározzák, ami majdnem 60 mező/s (NTSC-nél) vagy pontosan 50 mező/s (PAL-nál) sebességet biztosít. Ez a megjelenítési rendszer **sorugrásos megjelenítés** (**interlacing**) néven ismert. A számítógépen történő megjelenítésre szánt mozgóképek **progresszívek** (**progressive**), tehát nem használják a sorugrásos megjelenítést, mert a számítógép monitorokhoz kapcsolt grafikus kártyák pufferekkel rendelkeznek, ami lehetővé teszi a CPU számára, hogy másodpercenként 30 alkalommal helyezzen új képet a pufferbe, de a villogás megszüntetése érdekében a grafikus kártyának másodpercenként 50 vagy akár 100 alkalommal kell újrarajzolnia a képernyőt. Az analóg televíziókészülékeknek nincs olyan képkockapufferük, mint a számítógépeknek. Amikor egy gyors mozgásokat tartalmazó sorugrásos videót számítógépen jelenítenek meg, rövid vízszintes vonalak válnak láthatóvá az éles széleknél. Ezt a hatást **fésülésnek** (**combing**) nevezik.

Az interneten továbbított videókhoz használt képkockák méretei erősen változóak abból az egyszerű okból kifolyólag, hogy a nagyobb képkockák nagyobb sávszélességet igényelnek, ami nem áll mindig rendelkezésre. A kis felbontású videó 320 × 240 képpontból állhat, a „teljes képernyős" pedig 640 × 480 képpontból. Ezek a méretek megközelítik a régi számítógép-monitorokét és az NTSC-televízióét. A **képarány** (**aspect ratio**) vagy szélesség-magasság arány 4:3, ugyanaz, mint a szabvány televíziónál. A **HDTV** (**High Definition TeleVision – nagy felbontású tv**) videók 1280 × 720 képpont felbontással tölthetőek le. Ezeknek a „szélesvásznú" képeknek a képaránya 16:9, hogy jobban megfeleljen a mozifilm 3:2 képarányának. Összehasonlításképpen, egy szabványos DVD-videó általában 720 × 480 képpont felbontású, a Blu-ray lemezeken lévő videók pedig általában HDTV típusúak 1080 × 720 képpont[3] felbontással.

Az interneten a képpontok száma csak a történet egyik része, mert a médialejátszók ugyanazt a képet különféle méretekben képesek megjeleníteni. A videó is csak egy ablak a számítógép képernyőjén, amit fel lehet nagyítani vagy össze lehet zsugorítani. A több képpont szerepe az, hogy a kép minőségét javítani lehet annak érdekében, hogy ne tűnjön elmosódottnak, amikor felnagyítják. Sok monitor azonban még a HDTV-nél is több képpontot tartalmazó képeket (és így mozgóképeket is) képes megjeleníteni.

Mozgókép-tömörítés

A digitális videó tárgyalása után már nyilvánvaló kell, hogy legyen: a tömörítés kritikus fontosságú a videók interneten történő továbbítása szempontjából. Még egy 640 × 480 képpontos képkockákkal, képpontonként 24 bit színinformációval és 30 képkocka/s sebességgel működő videó is több mint 200 Mb/s sávszélességet igényel. Ez messze meghaladja azt a sávszélességet, amivel a legtöbb vállalati iroda csatlakozik az internethez, nem beszélve az otthoni felhasználókról, és ez még csak egyetlen videofolyam. Mivel a tömörítetlen videoátvitel – legalábbis a nagy kiterjedésű hálózatokon – teljességgel

3 Újabban a nagyobb felbontású HDTV is megjelent 1920 × 1080 képpontos felbontással. *(A lektor megjegyzése)*

kizárt, az egyedüli remény az, hogy lehetséges a nagymértékű tömörítés. Szerencsére az utolsó néhány évtized nagy mennyiségű kutatása sok tömörítő technikához és algoritmushoz vezetett, ami lehetségessé tette a mozgókép-továbbítást.

Sok formátumot használnak az interneten történő mozgóképátvitelhez, amelyek közül némelyik egyedi, némelyik szabványos. A legnépszerűbb kódoló a különféle formában megjelenő MPEG. Ez egy nyílt szabvány, melyet az .mpg és .mp4 kiterjesztésű állományokban, valamint más tároló formátumokban is megtalálhatunk. Ebben a szakaszban a videotömörítés végrehajtásának tanulmányozása céljából megtekintjük az MPEG-et. Kezdetben megvizsgáljuk az állóképek tömörítését JPEG-gel. A mozgókép pusztán képek (és hangok) sorozata. A videó tömörítésének egyik módja az összes kép kódolása egymás után. Első megközelítésben az MPEG csupán az összes képkocka JPEG-kódolása, és még néhány további képesség a képek közötti redundancia eltávolításához.

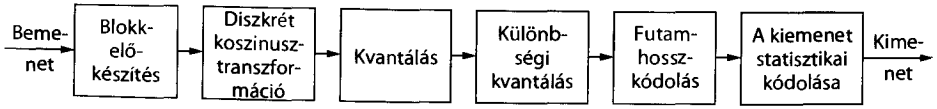
A JPEG-szabvány

A **JPEG**- (**Joint Photographic Experts Group**) szabványt, amely folytonos tónusú állóképek (vagyis fényképek) tömörítésére szolgál, az ITU, az ISO és az IEC felügyelete alatt dolgozó fényképszakértők fejlesztették ki. Széleskörűen használják (keressen .jpg kiterjesztésű állományokat), és gyakran 10:1 vagy jobb tömörítési arányt nyújt természetes képekre.

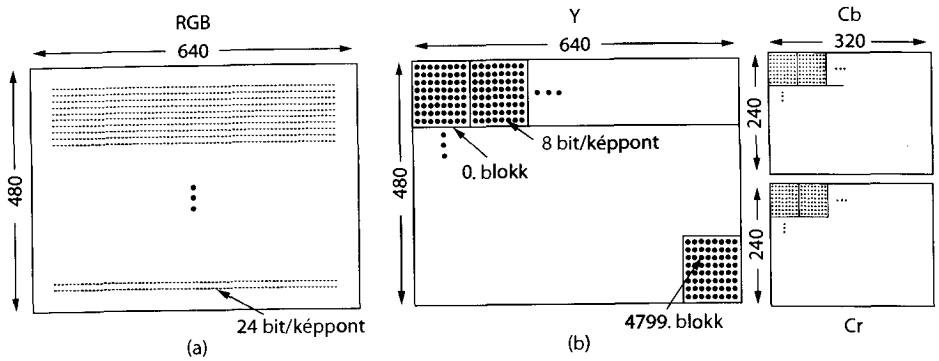
A JPEG-et az ISO 10918 Nemzetközi Szabvány definiálja. Valójában jobban hasonlít egy bevásárlólistára, mint egy algoritmusra, de a négy meghatározott működési mód közül csak a veszteséges soros mód (lossy sequential mode) bír jelentőséggel ismertetőnk szempontjából. Ezenkívül a JPEG normál használati módjára összpontosítunk, amely a 24 bites RGB-képek tömörítése, és némely apróbb részletet az egyszerűség kedvéért ki fogunk hagyni.

Az algoritmust a 7.44. ábra mutatja. Az első lépés a blokkok előkészítése. Az egyszerűség kedvéért tegyük fel, hogy a JPEG bemenetén egy 640×480-as, képpontonként 24 bites RGB kép van, amint a 7.45.(a) ábrán látszik. Az RGB nem a legjobb modell a tömörítéshez. A szem sokkal érzékenyebb a videojelek **luminanciájára** vagy fényességére, mint azok **krominanciájára** vagy színességére. Ezért először kiszámítjuk az *Y* luminanciát és a két, *Cb* és *Cr* krominanciát az *R*, *G* és *B* színösszetevőkből. A következő képleteket használjuk a 8 bites értékek meghatározására, melyek 0 és 255 közöttiek lehetnek:

$$Y = 16+0{,}26R+0{,}50G+0{,}09B$$
$$Cb=128+0{,}15R-0{,}29G-0{,}44B$$
$$Cr=128+0{,}44R-0{,}37G+0{,}07B$$



7.44. ábra. *A JPEG veszteséges soros kódolásának lépései*

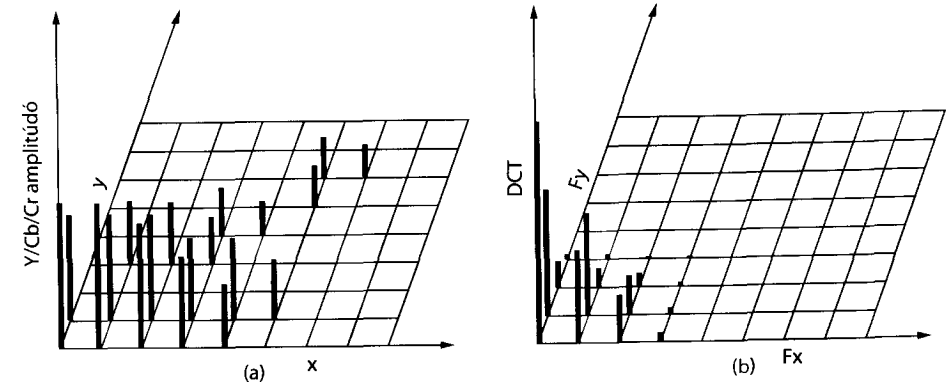


7.45. ábra. *(a) Az RGB bemeneti adat. (b) A blokkok előkészítése után*

Az *Y*, *Cb* és *Cr* számára külön mátrixokat készítünk. Következőnek az *Cb* és *Cr* mátrixokban a négy pixelből álló négyzetes blokkokat átlagoljuk, hogy azok méretét 320 × 240-re csökkentsük. Ez a csökkenés veszteséges, de a szem alig veszi észre a különbséget, mivel a fényességre érzékenyebb, mint a színekre. Viszont ez a teljes adatmennyiséget a felére tömöríti össze. Most mindhárom mátrix elemeiből kivonunk 128-at, hogy a 0 legyen az értelmezési tartomány közepén. Végül minden mátrixot 8 × 8-as blokkokra osztunk fel. Az *Y* mátrixnak 4800 blokkja lesz, a másik kettőnek egyenként 1200, ahogy a 7.45.(b) ábrán látszik.

A JPEG-kódolás 2. lépésében mind a 7200 blokkra külön-külön alkalmaznunk kell egy **DCT**-t (**Discrete Cosine Transformation** – **diszkrét koszinusztranszformáció**). Minden DCT kimenete egy DCT-együtthatókból álló 8 × 8-as mátrix. A (0, 0) elem a blokk átlaga. A többi elem azt mondja meg, hogy mennyi spektrális teljesítmény van az egyes térbeli frekvenciákban. Rendszerint ezek az elemek az origótól, a (0, 0)-tól való távolsággal arányosan rohamosan csökkennek, ahogy a 7.46. ábra is jelzi.

Amint a DCT kész van, a JPEG-kódolás továbblép a 3. lépésre, amelyet **kvantálásnak** (**quantization**) hívnak. Itt a kevésbé fontos DCT-együtthatókat kitöröljük. Ezt a



7.46. ábra. *(a) Az Y mátrix egy blokkja. (b) A DCT-együtthatók*

DCT-együtthatók

150	80	40	14	4	2	1	0
92	75	36	10	6	1	0	0
52	38	26	8	7	4	0	0
12	8	6	4	2	1	0	0
4	3	2	0	0	0	0	0
2	2	1	1	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Kvantálási táblázat

1	1	2	4	8	16	32	64
1	1	2	4	8	16	32	64
2	2	2	4	8	16	32	64
4	4	4	4	8	16	32	64
8	8	8	8	8	16	32	64
16	16	16	16	16	16	32	64
32	32	32	32	32	32	32	64
64	64	64	64	64	64	64	64

Kvantált együtthatók

150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

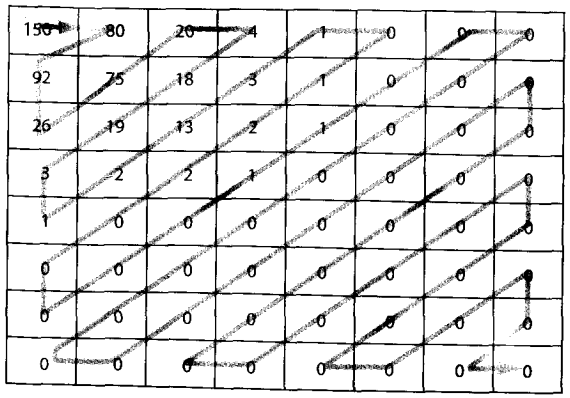
7.47. ábra. *A kvantált DCT-együtthatók számítása*

(veszteséges) transzformációt úgy végezzük, hogy a 8 × 8-as DCT-mátrixban minden együtthatót elosztunk egy táblázatból vett súllyal. Ha mindegyik súly értéke 1, akkor a transzformáció semmit sem csinál. Ha viszont a súlyok az origótól távolodva gyorsan csökkennek, akkor a nagyobb térbeli frekvenciák gyorsan kiesnek.

Erre a lépésre látható példa a 7.47. ábrán. Itt a kezdeti DCT-táblázat, a kvantálási táblázat és az eredmény látható, amelyet úgy kapunk, hogy minden DCT-elemet elosztunk a kvantálási táblázat megfelelő elemével. A kvantálási táblázatban található értékek nem képezik a JPEG-szabvány részét. Minden alkalmazásnak rendelkeznie kell egy saját táblázattal, ezáltal lehetséges a veszteség és a tömörítés arányának a szabályozása.

A 4. lépésben lecsökkentjük minden blokk (0, 0) értékét (amelyik a bal felső sarokban van) azáltal, hogy az előző blokk ugyanezen a helyen lévő elemétől való eltérésével helyettesítjük. Mivel ezek az elemek a saját blokkjaik átlagai, várhatóan csak lassan fognak változni, ezáltal a különbségi értékek kis számok lesznek. A többi értéknél nem számolunk eltérést.

Az 5. lépésben sorba rakjuk a 64 elemet és a listára futamhosszkódolást alkalmazunk. A blokk balról jobbra, és azután fentről lefele történő pásztázása nem gyűjtené össze a 0-kat, így egy cikcakkos pásztázási mintát alkalmaznak (lásd 7.48. ábra). Ebben a példában a cikcakk minta 38 egymás utáni 0-t eredményez a mátrix végén. Ezt a sorozatot



7.48. ábra. *A sorrend, amelyben a kvantált értékeket átviszik*

össze lehet vonni egyetlen számba, mely megmondja, hogy 38 nullánk van. Ezt a módszert **futamhosszkódolásnak** (**run-length encoding**) nevezik.

Most már van egy számsorozatunk, ami (a transzformált térben) leírja a képet. A 6. lépés Huffman-kódolja a számokat tárolás vagy átvitel céljából, azaz a gyakorabban előforduló számokhoz rövidebb kódszavakat rendel, mint a ritkábban előforduló számokhoz.

A JPEG bonyolultnak tűnhet, de ez csak azért van, mert *tényleg* bonyolult. Mégis megéri használni az akár 1:20 arányú tömörítés előnyei miatt. Egy JPEG-kép dekódolásához az algoritmus fordítottját kell lefuttatni. A dekódolás pont olyan hosszú, mint a kódolás, vagyis a JPEG nagyjából szimmetrikus. Ez nem minden tömörítő algoritmusra jellemző, amint azt rövidesen látni fogjuk.

Az MPEG-szabvány

Végre eljutottunk a dolgok lényegéig: az **MPEG-** (**Motion Picture Experts Group – mozgókép szakértői csoport**) szabványokig. Bár számos egyedi algoritmus létezik, ezek a szabványok határozzák meg a mozgóképek tömörítésére használt főbb algoritmusokat. Ezek 1993 óta nemzetközi szabványok. Mivel a filmek képeket és hangokat is tartalmaznak, az MPEG mind hangtömörítésre, mind képtömörítésre alkalmas. A hangok és az állóképek tömörítését már tanulmányoztuk, lássuk most tehát a mozgókép-tömörítést.

Az MPEG-1 szabványt (ami magába foglalja az MP3-at is) először 1993-ban jelentették meg, és azóta is széleskörűen használják. Az volt a cél, hogy a videomagnókkal azonos minőségű kimenetet eredményezzen, ami a 40:1 arányú tömörítés mellett körülbelül 1 Mb/s adatsebességet igényelt. Az ilyen mozgókép már alkalmas a széles körű internetes használatra a webhelyeken. Ne aggódjon, ha nem emlékszik a videomagnókra – az MPEG-1-et is a filmek CD-n történő tárolására használták, amikor még léteztek olyanok. Ha nem tudja, hogy mik azok a CD-k, akkor tovább kell lépnünk az MPEG-2-re.

Az 1996-ban megjelent MPEG-2 szabványt műsorszórásra alkalmas minőségű mozgókép tömörítésére tervezték. Mára nagyon elterjedt, minthogy ez képezi a DVD-n lévő kódolt mozgókép (ami elkerülhetetlenül megtalálja útját az internet felé) és a digitális televíziós műsorszórás (DVB) alapját is. A DVD-minőségű mozgóképeket általában 4-8 Mb/s adatsebességgel kódolják.

Az MPEG-4 szabvány két videoformátummal rendelkezik. Az első, 1999-ben megjelent formátum a mozgóképet objektumalapú ábrázolással kódolja. Ez lehetővé teszi a természetes és szintetikus képek, valamint másfajta média összekeverését, például egy időjárási térkép előtt álló meteorológus esetén. Ennek a felépítésnek köszönhetően könnyű elérni, hogy a programok kezelhessék a film adatait. A második, 2003-ban megjelent formátum **H.264** vagy **AVC** (**Advanced Video Coding – fejlett mozgóképkódolás**) néven ismert. Ennek célja, hogy a mozgóképet a korábbi kódolók adatsebességének felével kódolja azonos minőség mellett, hogy még jobban támogassa a mozgóképek hálózaton keresztül történő átvitelét. Ezt a kódolót használják a HDTV-adásokhoz a legtöbb Blu-ray lemezen.

Ezeknek a szabványoknak sok és változatos részlete van. A későbbi szabványok sokkal több képességgel és kódolási lehetőséggel is rendelkeznek, mint a korábbi szabványok. A részletekbe azonban nem fogunk belemenni. A mozgóképtömörítésben az idők során elért előrelépéseket legnagyobb részben az apró továbbfejlesztgetések eredményezték,

és nem a videotömörítésben bekövetkező alapvető változások. Ezért az átfogó koncepciókat fogjuk felvázolni.

Az MPEG a hangot és a mozgóképet is összetömöríti. Mivel a hang- és videokódolók függetlenül dolgoznak, felmerül a kérdés, hogy hogyan lehet a két folyamot a vevőnél szinkronizálni. A megoldás egyetlen óra alkalmazása, amely mindkét kódoló számára rendelkezésére bocsátja a pontos időt tartalmazó időbélyegeket. Ezeket az időbélyegeket tartalmazza a kódolt kimenet, és így eljutnak a vevőig, amely ezeket a hang- és képfolyamok szinkronizálásához használhatja fel.

Az MPEG-videotömörítés a filmekben előforduló kétféle, térbeli és időbeli redundanciát használja ki. A térbeli redundancia kihasználható úgy, hogy egyszerűen minden képet külön kódolnak JPEG-gel. Ezt a megközelítést alkalomszerűen használják, különösen akkor, amikor tetszőleges hozzáférés szükséges minden egyes képkockához, mint a videók szerkesztése közben. Ebben a módban JPEG-szintű tömörítés érhető el.

További tömörítés érhető el azt a tényt kihasználva, hogy az egymás után következő képkockák gyakran majdnem teljesen azonosak. Ez a hatás kisebb, mint elsőre tűnhet, mert sok filmkészítő 3-4 másodpercenként iktat be vágást (hogy mérje egy filmrészlet időtartamát, és számolja a vágásokat). Ennek ellenére még a 75 vagy több, nagymértékben hasonló képkockából álló futamok is nagy csökkentési lehetőséget kínálnak ahhoz képest, hogy az egyes képeket külön-külön JPEG-gel kódoljuk.

Az olyan jelenetekben, ahol a kamera és a háttér áll, és egy vagy több színész lassan mozog, majdnem minden képpont képről képre ugyanaz marad. Itt jól működne az, hogy minden képet kivonunk az előzőből, és a különbségre JPEG-kódolást alkalmazunk. Ám ez az eljárás látványosan csődöt mond olyan jelenetek esetében, ahol a kamera mozog, vagy ráközelít valamire. Valamilyen módon ezeket a mozgásokat kompenzálni kell. Az MPEG pontosan ezt teszi, és ez a lényegi különbség az MPEG és a JPEG között.

Az MPEG-kimenet három fajta képből áll:

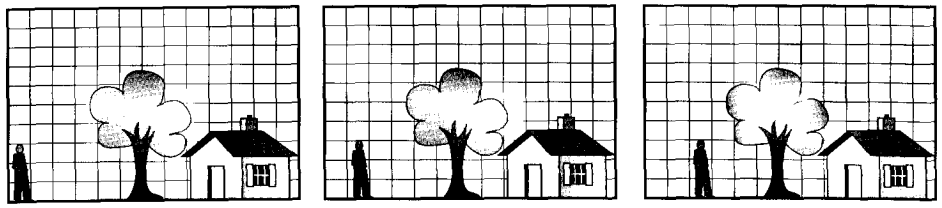
1. I (Intracoded) képek: önállóan tömörített állóképek.

2. P (Predictive) képek: blokkonkénti eltérés az előző képektől.

3. B (Bidirectional) képek: az előző és a jövőbeli képek közötti blokkonkénti eltérések.

Az I képek egyszerűen JPEG-kódolt állóképek. Ezek kódolása JPEG vagy más hasonló módszerrel történhet. Három ok miatt érdemes a kimeneti folyamban rendszeres időközönként (például másodpercenként egyszer vagy kétszer) I képeket megjeleníteni. Először is, az MPEG használható többesküldésre is, ahol a nézők tetszés szerint kapcsolódhatnak be az adásba. Ha minden kép függ az előzőktől, egészen az első képig visszamenőleg, bárki, aki az első képet elmulasztotta, nem tudná a további képeket dekódolni. Másodszor, ha valamely kép hibásan érkezne meg, nem lenne lehetséges a további dekódolás: innentől kezdve minden értelmetlen szemét lenne. Harmadszor, I képek nélkül az előre- vagy visszatekerésnél a dekódernek minden képet ki kellene számolnia, amelyen áthalad, hogy tudja annak a képnek teljes értékét, amelyen megáll.

Ezzel ellentétben a P képek a képek közti különbségeket kódolják. Ezek a **makroblokkokon** (**macroblocks**) alapulnak, amelyek a luminanciamátrixból például egy 16×16



7.49. ábra. *Három egymást követő kép*

képpontot, a krominanciamátrixokból pedig 8×8 képpontot fednek le. Egy makroblokkot úgy kódolnak, hogy az előző képen megkeresik ugyanazt, vagy egy ettől csak kevéssé eltérő makroblokkot.

A 7.49. ábrán látható egy olyan példa, ahol a P képek hasznosak lehetnek. Itt három egymás után következő képet látunk, amelyeknek ugyanaz a hátterük, csupán egy ember helyzete különböző. A hátteret tartalmazó makroblokkok pontosan illeszkedni fognak, de azoknak a makroblokkoknak a pozíciója, amelyek az embert tartalmazzák, valamilyen ismeretlen értékkel eltolódnak, és azokat meg kell keresni.

Az MPEG-szabványok nem határozzák meg, hogy hogyan kell a keresést véghezvinni, milyen messzire kell a keresést kiterjeszteni, és mi számít jó illeszkedésnek. Ezt minden megvalósításnak egyedül kell eldöntenie. Például egy megvalósítás az előző képen az aktuális pozícióban, és attól vízszintesen $\pm \Delta x$, függőlegesen $\pm \Delta y$ távolságig kereshet illeszkedést. Minden pozícióhoz ki kell számolni a luminanciamátrix illeszkedő elemeinek számát. A legmagasabb pontszámú pozíciót kiálthatnák ki nyertesnek, feltéve, hogy az valamilyen küszöbszint felett van. Egyébként a makroblokkot hiányzónak nevezik. Természetesen sokkal kifinomultabb algoritmusok is lehetségesek.

Ha egy makroblokkot megtalálnak, akkor veszik az aktuális és az előző kép értéke közötti különbséget, luminanciában is és krominanciákban is. Ezeket a különbségmátrixokat azután szokás szerint diszkrét koszinusz-transzformációnak, kvantálásnak, futamhosszkódolásnak és Huffman-kódolásnak vetik alá. A makroblokkra vonatkozó érték ezután a kimeneti folyamban úgy jelenik meg ekkor, mint a mozgásvektor (azaz, hogy mennyit mozdult el a makroblokk előző helyzetéhez képest mindkét irányban), amelyet különbségének kódolása követ. Ha a makroblokk nem szerepel az előző képen, akkor a jelenlegi értékét kódolják úgy, mintha az egy I képben lenne.

Nyilvánvaló, hogy ez az algoritmus nagymértékben aszimmetrikus. Egy megvalósítás akár minden lehetséges pozíciót kipróbálhat az előző képen, ha mindenképpen meg akarja találni az utolsó makroblokkot is, bárhol is legyen az. Ez a megközelítés minimalizálni fogja a kódolt MPEG-folyamot azon az áron, hogy nagyon lassú lesz a kódolás. Ez jó lehet egy filmkönyvtár egyszeri lekódolásához, de a valós idejű videokonferenciához nagyon nem alkalmas.

Hasonlóan, minden megvalósítás maga döntheti el, hogy mit vesz „megtalált" makroblokknak. Ez a szabadság lehetővé teszi a megvalósítás programozóinak, hogy versenyezzenek a minőség és a használt kereső algoritmus területén, de mindig az MPEG-nek megfelelő kimenetet állítanak elő.

Eddig az MPEG dekódolása magától értetődő. Az I képek dekódolása hasonló, mint a JPEG-képek dekódolása. A P képek dekódolásához a dekódernek el kell tárolni az előző

képeket, így képes lesz felépíteni a következő képet egy külön pufferben a teljesen kódolt makroblokkok és az előző képek óta történt változásokat tartalmazó makroblokkok alapján. Az új képet makroblokkról makroblokkra rakják össze.

A B képek hasonlók a P képekhez, azzal a különbséggel, hogy ezek lehetővé teszik, hogy a hivatkozott makroblokk a megelőző vagy rákövetkező képekben legyen. Ez a további szabadság javított mozgáskompenzációt tesz lehetővé. Akkor hasznos, amikor például a tárgyak más tárgyak előtt vagy mögött haladnak el. A B képek kódolásához a kódolónak egy képsorozatot kell egyszerre a memóriában tartania: a korábbi képeket, az aktuális, éppen kódolás alatt levő képet és a későbbi képeket. A dekódolás hasonlóképpen bonyolultabb, és némi késleltetést is okoz. Ennek az az oka, hogy egy adott B képet nem lehet dekódolni mindaddig, amíg a rákövetkező képek közül azok, amelyektől függ, nincsenek dekódolva. Tehát annak ellenére, hogy a B képek biztosítják a legjobb tömörítést, nem mindig használják azokat nagyobb bonyolultságuk és a pufferelés igénye miatt.

Az MPEG-szabványok ezeknek a technikáknak sok továbbfejlesztett változatát tartalmazzák a kiváló tömörítési szintek érdekében. Az AVC a mozgókép 50:1 arányt meghaladó mértékű tömörítésére használható, ami a hálózati sávszélességigényt is ugyanekkora mértékben csökkenti. Az AVC-vel kapcsolatos további információért lásd Sullivan és Wiegand [2005] munkáját.

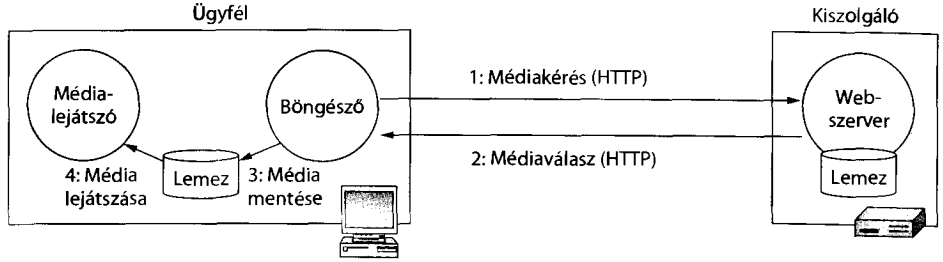
7.4.3. Tárolt média folyamszerű átvitele

Most térjünk át a hálózati alkalmazásokra! Ezek közül az első az olyan média folyamszerű átvitele, amelyet állományokban tárolnak. Ennek leggyakoribb példája a mozgóképek interneten keresztül történő megtekintése. Ez a **VoD (Video on Demand – igény szerinti videonézés)** egyik formája. Az igény szerinti videonézés többi formája egy, az internettől független szolgáltató-hálózatot (például kábelhálózat) használ a filmek továbbításához.

A következő szakaszban az élő médiaközvetítést, például az IPTV-műsorszórást és az internetes rádiót vizsgáljuk meg. Ezután a harmadik esetet jelentő valós idejű konferenciát tárgyaljuk. Ilyen például az IP-hálózaton keresztül történő beszédátvitel és a Skype-on lebonyolított videokonferencia. Ez a három eset egyre szigorúbb követelményeket támaszt a hangnak és mozgóképnek hálózaton keresztül történő továbbításával kapcsolatban, mert egyre növekvő figyelmet kell szentelnünk a késleltetésnek és a dzsitternek.

Az internet tele van zenei és videooldalakkal, amelyek tárolt médiaállományokat továbbítanak. Valójában a legegyszerűbb módja a tárolt média kezelésének az, ha *nem* folyamszerűen továbbítjuk. Képzeljük el, hogy létre akarunk hozni egy online videokölcsönző oldalt, hogy versenyre keljen az Apple iTunes-szal! Egy szokásos webhely lehetővé fogja tenni a felhasználóknak, hogy letöltsék, majd megnézzék a mozgóképeket (természetesen csak az után, hogy fizettek). A lépések sorozatát a 7.50. ábra mutatja. Ki fogjuk silabizálni ezeket, hogy összevethessük a következő példával.

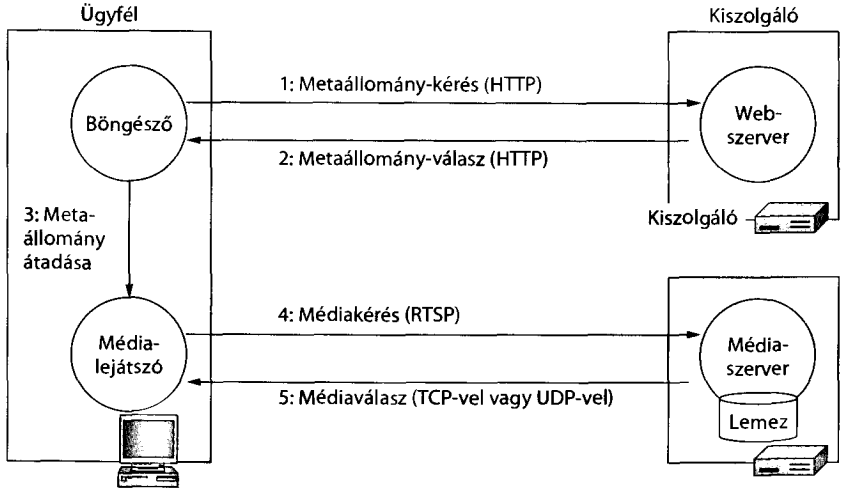
A böngésző akkor lép akcióba, amikor a felhasználó rákattint egy filmre. Az 1. lépésben elküld egy, a filmre vonatkozó HTTP-kérést annak a webszervernek, amelyre a film hiperhivatkozása mutat. A 2. lépésben a kiszolgáló előveszi a filmet (ami csak egy MP4, vagy valamilyen más formátumban lévő állomány), és visszaküldi azt a böngészőnek. A MIME-típus, például *video/mp4* felhasználásával a böngésző meghatározza, hogy hogyan kellene megjeleníteni az állományt. Ebben az esetben egy segédalkalmazásként feltüntetett médialejátszó szolgál erre a célra, de beépülő modul is lehetne. A 3. lépésben a böngésző az egész filmet kimenti egy lemezen elhelyezett ideiglenes állományba. Ezután elindítja a médialejátszót, átadva neki az ideiglenes állomány nevét. Végül a 4. lépésben a médialejátszó elkezdi beolvasni az állományt, és lejátszani a filmet.



7.50. ábra. *Médialejátszás a világhálón keresztül egyszerű letöltések segítségével*

Ez a megközelítés elvileg teljesen helyes. Le fogja játszani a filmet. Nincsenek megoldandó problémák sem a valós idejű hálózati átvitellel kapcsolatban, mert a letöltés egy egyszerű állományletöltés. Az egyetlen probléma az, hogy az egész filmet át kell vinni a hálózaton, mielőtt a lejátszás megkezdődhet. A legtöbb ügyfél nem kíván egy órát várni egy igény szerinti videonézésre. Ez a modell még a hangok számára is problémás lehet. Képzeljen el egy dalba történő belehallgatást az album megvásárlása előtt! Ha a dal 4 MB méretű, ami egy MP3 dal jellemző mérete, és a széles sávú kapcsolat sebessége 1 Mb/s, a felhasználót fél perc csend fogadja, mielőtt a belehallgatás elkezdődik. Ez a modell nem alkalmas arra, hogy túl sok albumot eladjanak.

A webhelyek a 7.51. ábrán látható felépítést alkalmazhatják annak érdekében, hogy a böngészők működésének megváltoztatása nélkül megkerüljék a problémát. A filmre



7.51. ábra. *Média folyamszerű átvitele a web és egy médiakiszolgáló felhasználásával*

hivatkozó oldal nem a tényleges filmállomány, hanem egy úgynevezett **metaállomány** (**metafile**), azaz egy nagyon rövid állomány, ami mindössze megnevezi az adott filmet (és valószínűleg más kulcsleírókat is tartalmaz). Egy egyszerű metaállomány lehet akár egyetlen ASCII-szöveg is, mint ez itt:

```
rtsp://joes-movie-server/movie-0025.mp4
```

A böngésző a szokásos módon megkapja az egysoros állományt az 1. és 2. lépésekben. Ezután elindítja a médialejátszót és átadja neki az egysoros állományt a 3. lépésben, ahogy megszokhattuk. A médialejátszó beolvassa a metafájlt és látja azt az URL-t, ahonnan a filmet meg kell kapnia. Kapcsolatba lép a *joes-movie-server*-rel, és elkéri a filmet a 4. lépésben. Ezután az 5. lépésben a médiakiszolgáló a filmet folyamszerűen továbbítja a médialejátszónak. Ennek a kialakításnak az az előnye, hogy a médialejátszó gyorsan elindul egy nagyon rövid metafájl letöltését követően. Miután ez megtörtént, a böngésző többé nem részese a folyamatnak. A médiát a médiakiszolgáló közvetlenül a médialejátszóhoz küldi, így az a teljes film letöltése előtt elkezdheti a film megjelenítését.

Két kiszolgálót látunk a 7.51. ábrán, mert a metafájlban megnevezett kiszolgáló gyakran nem azonos a webszerverrel. Valójában többnyire még csak nem is HTTP-kiszolgáló, hanem egy specializált médiakiszolgáló. Ebben a példában a médiakiszolgáló **RTSP**-t (**Real Time Streaming Protocol – valós idejű adatfolyam protokoll**) használ, ahogy azt az *rtsp* sémanév is jelzi.

A médialejátszónak négy főbb feladata van:

1. Kezelni a felhasználói csatlakozó felületet.
2. Kezelni az átviteli hibákat.
3. Kibontani (*expandálni*) a tartalmat.
4. Kiküszöbölni a dzsittert.

Manapság a legtöbb médialejátszónak cicomás felhasználói csatlakozó felülete van, ami néha a sztereo hifiberendezéseket utánozza, gombokkal, tekerőkkel, csúszkákkal és vizuális kijelzőkkel. Az ilyen lejátszóknál gyakran megtalálható a **cserélhető előlap**, az ún. **skin**, amit a felhasználó maga rakhat rá a lejátszóra. A médialejátszónak mindezt kezelnie kell, és együtt kell működnie a felhasználóval.

A többi feladat a hálózati protokollokkal kapcsolatos, illetve azoktól függ. Sorjában mindegyiken végig fogunk menni, kezdve az átviteli hibák kezelésével. A hibák kezelésének módja attól függ, hogy a média továbbításához olyan TCP-alapú protokollt használnak-e, mint a HTTP, vagy olyan UDP-alapút, mint az RTP. A gyakorlatban mindkettő használatos. Ha TCP-alapú szállítást használnak, akkor nincs hiba, amit a médialejátszó kijavíthatna, mert a TCP már megbízható szállítást biztosít. Ez könnyű módja a hibák kezelésének, legalábbis a médialejátszó számára, de ez a következő lépésben bonyolulttá teszi a dzsitter eltávolítását.

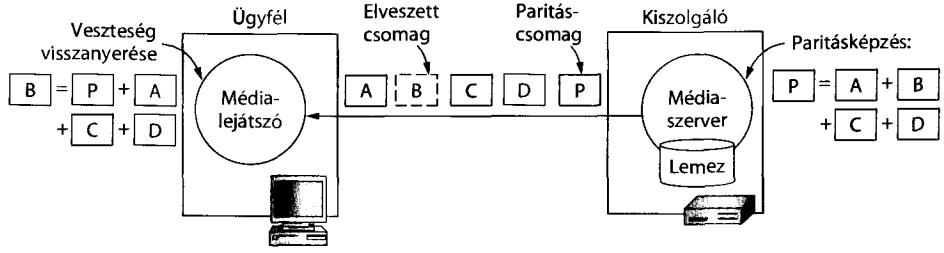
Másik lehetőségként az adatok mozgatásához olyan UDP-alapú átvitelt használhatnak, mint az RTP. Ezt a 6. fejezetben tanulmányoztuk. Ezeknél a protokolloknál nincsenek újraküldések. Így a torlódás következtében történő csomagvesztés vagy az átviteli hibák azt fogják eredményezni, hogy a média egy része nem érkezik meg. Ennek a problémának a megoldása a médialejátszóra hárul.

Lássuk, milyen nehézséggel állunk szemben! A csomagvesztés gond, mert az ügyfelek nem kedvelik a hosszú kimaradásokat kedvenc dalaikban és filmjeikben. Ez azonban nem akkora probléma, mint egy hagyományos állomány továbbítása közben történt csomagvesztés, mert a média egy kis darabjának elvesztése nem feltétlenül rontja le a felhasználónak tartott megjelenítést. Mozgóképek esetén a felhasználó valószínűleg észre sem veszi, ha valamelyik másodpercben 25 új képkocka helyett véletlenül csak 24 új képkocka van. A hangoknál a lejátszás során előforduló rövid kimaradásokat el lehet fedni az időben közeli hangokkal. A felhasználó valószínűleg nem veszi észre ezt a helyettesítést, hacsak nem *kifejezetten erre* figyel.

A fenti érvelés kulcsa azonban az, hogy a kimaradások nagyon rövidek. A hálózati torlódás vagy egy átviteli hiba általában egy egész csomag elvesztését okozza, és a csomagok gyakran kisebb löketekben vesznek el. Két stratégia használható a csomagvesztésnek a médiára gyakorolt hatásának csökkentésére: a FEC és az összefésülés. A következőkben mindegyiket ismertetjük.

Az **FEC** (**Forward Error Correction – előre irányuló hibajavítás**) egyszerűen annak a hibajavító kódolásnak az alkalmazási szinten történő felhasználása, amit a 3. fejezetben tanulmányoztunk. Erre egy példa a csomagok közötti paritásképzés [Shacham és McKenny, 1990]. Minden négy elküldött adatcsomaghoz létrehoznak és elküldenek egy ötödik, ún. **paritáscsomagot** (**parity packet**). Ez látszik a 7.52. ábrán, az *A*, *B*, *C* és *D* csomagokkal. A *P* paritáscsomag redundáns biteket tartalmaz, amelyek a négy adatcsomagban található bitek paritásai vagy „kizáró-vagy" (KIZÁRÓ VAGY) összegei. Remélhetőleg az öt csomagból álló csoportok nagy részének minden csomagja megérkezik. Amikor ez megtörténik, a vevő a paritáscsomagot egyszerűen figyelmen kívül hagyja. Akkor sincs baj, ha csak a paritáscsomag hiányzik.

Néha azonban egy adatcsomag elveszhet az átvitel során, mint a *B* csomag a 7.52. ábrán. A médialejátszó csak három adatcsomagot, az *A*-t, *C*-t és *D*-t, továbbá a *P* paritáscsomagot kapja meg. Az elveszett adatcsomagban lévő bitek helyreállíthatók a paritásbitek segítségével. Hogy konkrétak legyünk, a „+" jelet a „kizáró-vagy" vagy modulo 2 összeadás



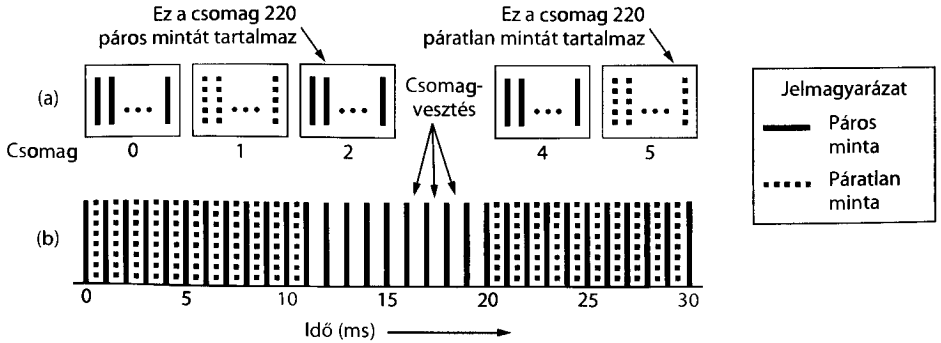
7.52. ábra. *A paritáscsomag felhasználása a csomagvesztés javítására*

jelölésére használva, a B csomag helyreállítható a következőképpen: $B=P+A+C+D$, felhasználva a „kizáró-vagy" tulajdonságait (például $X+Y+Y=X$).

Az FEC képes csökkenteni a médialejátszó által észlelt csomagvesztési szintet néhány elvesztett csomag javításával, de ez csak egy bizonyos szintig működik. Ha az ötelemű csoportból két csomag elveszik, semmit sem tudunk tenni az adatok visszaszerzése érdekében. Az FEC másik, figyelembe veendő tulajdonsága az ár, amit ennek a védelemnek az eléréséért kell fizetni. Minden négy csomagból öt lett, így a média sávszélességigénye 25%-kal nagyobb. A dekódolás késleltetési ideje is megnőtt, mert szükségünk lehet rá, hogy várjunk, amíg a paritáscsomag megérkezik, mielőtt helyreállíthatnánk az előtte érkezett adatcsomagot.

Van egy okos trükk is a fenti technikában. A 3. fejezetben azt írtuk a paritásról, hogy hibadetektálást biztosít. Mi itt hibajavítást végzünk. Hogyan képes elvégezni mindkét feladatot? A válasz az, hogy ebben az esetben ismert, hogy melyik csomag veszett el. Az elveszett adatot **törlődésnek** (**erasure**) nevezik. A 3. fejezetben, amikor megvizsgáltunk egy keretet, ami néhány bithibával érkezett, nem tudtuk, melyik bit hibásodott meg. Ennek az esetnek a kezelése nehezebb, mint a törlődéseké. Tehát a paritásképzés törlődések esetén hibajavítást biztosít, törlődések hiányában csak hibajelzést. Hamarosan látni fogjuk a paritásnak egy másik, váratlan előnyét, amikor elérünk a többesküldést tartalmazó forgatókönyvhöz.

A második stratégiát **összefésülésnek** (**interleaving**) nevezik. Ez a megközelítés azon alapul, hogy átvitel előtt a média sorrendjét összekeverik vagy összefésülik, vételnél pedig a sorrendet visszaállítják vagy kifésülik. Ily módon, ha egy csomag (vagy csomagok lökete) elvész, a sorba rendezés során a veszteség időben szét fog szóródni. Nem eredményez egyetlen, nagy kiesést a média lejátszásakor. Egy csomag tartalmazhat például 220 sztereo mintát, mindegyikben két darab 16 bites számmal, ami rendszerint 5 ms hosszú zenének felel meg. Ha a mintákat sorban küldenék el, az elvesztett csomag 5 ms hosszú kiesést jelentene a zenében. Ehelyett a mintákat úgy továbbítják, ahogyan az a 7.53. ábrán látható. Egy 10 ms-os intervallum minden páros mintáját elküldik egy csomagban, amelyet az összes páratlan mintát tartalmazó követ. A 3-as csomag elvesztése most nem 5 ms kimaradást jelent a zenében, hanem minden második minta elvesztését 10 ms-on keresztül. Ez a veszteség könnyen kezelhető, ha a médialejátszó az előző és a következő



7.53. ábra. *Amikor a csomagok váltakozó mintákat szállítanak, a csomagvesztés csak átmeneti felbontáscsökkenést okoz és nem időbeli kimaradást*

minták felhasználásával interpolálja a hiányzó értékeket. Az eredmény: kisebb átmeneti felbontás 10 ms-on keresztül, és nem egy észrevehető időbeli kimaradás a médiában.

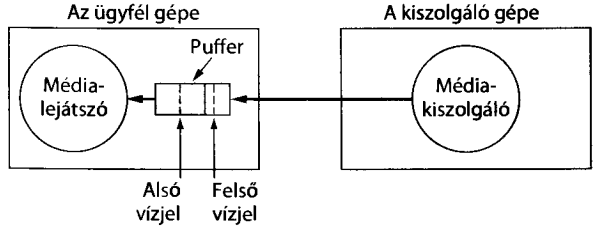
Ez az összefésülési séma csak tömörítetlen mintavétel esetén működik. Az összefésülés azonban (rövid időn keresztül, és nem egyedi mintákkal) tömörítés után is alkalmazható mindaddig, amíg valahogyan meg lehet találni a minta határait a tömörített folyamban. Az RFC 3119 egy olyan sémát ír le, amely tömörített hanggal működik.

Az összefésülés, amikor használható, egy jó megoldás, mert nincs szüksége nagyobb sávszélességre, mint ahogy azt az FEC-nél láttuk. Az összefésülés azonban, akárcsak az FEC, növeli a késleltetést, mert meg kell várni egy csomagokból álló csoport megérkezését (azért, hogy ki lehessen fésülni).

A médialejátszó harmadik feladata a tömörített tartalom kibontása (**decompressing**). Ez ugyan nagyon számításigényes, de eléggé magától értetődő feladat. A nehéz kérdés az, hogy hogyan dekódoljuk a médiát, ha a hálózati protokoll nem javítja ki az átviteli hibákat. Sok tömörítő sémában a későbbi adatot nem lehet kibontani, amíg a korábbi adat nincs kibontva, mert a későbbi adatot a korábbi adatra vonatkozóan kódolják. Egy UDP-alapú átvitel esetén előfordulhat csomagvesztés. Tehát a kódolási folyamatot úgy kell megtervezni, hogy a dekódolást a csomagvesztés ellenére is lehetővé tegye. E követelmény miatt használ az MPEG I, P és B képeket. Minden I kép a többi képtől függetlenül dekódolható, így bármely korábbi kép elvesztése után is helyre tud állni.

A negyedik teendő a minden valós idejű rendszer rémének tekintett dzsitternek a kiküszöbölése. Az általános módszer, amit a 6.4.3. szakaszban ismertettünk, egy lejátszási puffer használata. Minden folyamszerű átvitelt alkalmazó rendszer úgy indul, hogy körülbelül 5-10 másodpercnyi médiát pufferel, mielőtt a lejátszást megkezdené, amint azt a 7.54. ábra is mutatja. A lejátszáskor a média szabályosan folyik ki a pufferből, ezért a hang tiszta és a mozgókép egyenletes. A kezdeti késleltetés megadja az esélyt a puffernek arra, hogy megteljen az alsó vízjelig (**low-water mark**). Az elképzelés az, hogy az adatoknak innen kezdve eléggé szabályosan kell érkezniük ahhoz, hogy a puffer soha ne ürüljön ki teljesen. Ha ez megtörténne, a médialejátszás leállna. A pufferelés értelme az, hogy ha az adatok a torlódás miatt időnként lassan érkeznek meg, akkor a pufferben lévő média lehetővé teszi a lejátszás normális folytatását az új médiaadatok megérkezéséig és a puffer újbóli feltöltéséig.

Az, hogy mekkora pufferre van szükség és a médiakiszolgáló milyen gyorsan küldi a puffer feltöltéséhez az adatokat, a hálózati protokollokon múlik. Számos lehetőség kínálkozik. A tervezésnél az a legfontosabb tényező, hogy UDP-alapú vagy TCP-alapú továbbítást alkalmaznak.



7.54. ábra. *A médialejátszó puffereli a médiakiszolgálótól érkező bemenetet, és a lejátszás nem közvetlenül a hálózatról, hanem a pufferből történik*

Tételezzük fel, hogy olyan UDP-alapú továbbítást alkalmaznak, mint az RTP. Továbbá tételezzük fel azt is, hogy elegendő sávszélesség áll rendelkezésre ahhoz, hogy a csomagokat kis csomagvesztéssel és kis egyéb hálózati forgalom mellett küldik el a médiakiszolgálótól a médialejátszóhoz. Ebben az esetben a csomagokat pontosan akkora sebességgel lehet küldeni, amekkora a média lejátszási sebessége. Minden csomag áthalad a hálózaton, és a terjedési késleltetés után, nagyjából pont akkor érkezik meg a médialejátszóhoz, amikor annak a médiát le kell játszania. Nagyon kis puffer szükséges, mivel a késleltetés nem változik. Ha összefésülést vagy FEC-t használnak, nagyobb pufferre van szükség, hogy legalább azok a csomagok elférjenek, amelyek az összefésülést vagy FEC-t végrehajtják. Ez azonban csak kissé növeli a puffer méretét.

Sajnos ez a forgatókönyv két szempontból is légbőlkapott. Először is, a hálózati utak sávszélessége változó, így a médiakiszolgáló számára általában nem világos, hogy elegendő lesz-e a sávszélesség, mielőtt elkezdené a médiát folyamszerűen továbbítani. Egy egyszerű megoldás a média többféle felbontásban történő kódolása, és annak lehetővé tétele, hogy minden felhasználó olyan felbontást válasszon, amelyet az internetkapcsolata lehetővé tesz. Gyakran csak két szint létezik: csúcsminőség, mondjuk 1,5 Mb/s vagy nagyobb sebességgel kódolva, és gyenge minőség, mondjuk 512 kb/s vagy kisebb sebességgel kódolva.

Másodszor, mindig lesz egy kis dzsitter vagy ingadozás a médiamintáknak a hálózaton történő áthaladásának idejében. Ez a dzsitter két forrásból fakad. Gyakran van érzékelhető mennyiségű versengő forgalom a hálózaton – ezek egy részét maguk a több feladatot egyszerre végző felhasználók idézik elő a világháló böngészésével, miközben látszólag a folyamszerűen átvitt filmet nézik. Ez a forgalomban lüktetéseket okoz a média megérkezésekor. Továbbá, bennünket a videó képkockáinak és a hangmintáknak a megérkezése érdekel, nem a csomagok megérkezése. Tömörítés esetén különösen a videó képei lehetnek nagyobbak vagy kisebbek, a tartalomtól függően. Egy akciósorozat kódolásához általában több bitre van szükség, mint egy békés tájképéhez. Ha a hálózati sávszélesség állandó, az egységnyi idő alatt szállított média mennyisége változni fog. Minél nagyobb az ezekből a forrásokból származó dzsitter, vagyis a késleltetés ingadozása, annál magasabbra kell helyezni a puffer alsó vízjelét az alulcsordulás elkerülése érdekében.

Most képzeljük el, hogy olyan TCP-alapú átvitelt használnak a média elküldésére, mint a HTTP. A TCP azáltal, hogy újraküldéseket végez, és megvárja, amíg a csomagok a megfelelő sorrendben megérkeznek, meglehet, hogy jelentősen megnöveli a médialejátszó által megfigyelt dzsittert. Az eredmény az, hogy nagyobb pufferre és magasabb felső vízjelre van szükség. Van azonban egy előnye is. A TCP olyan gyorsan küldi az adatokat, ahogy azokat a hálózat szállítani képes. A média néha késhet, ha az elveszett adatokat javítani kell. Az idő nagy részében azonban a hálózat képes gyorsabban szállítani a médiát, mint ahogyan a lejátszó felhasználja azt. Ezekben az időszakokban a puffer töltődni fog, és megelőzi a jövőbeli alulcsordulásokat. Ha a hálózat jelentősen gyorsabb, mint az átlagos médiasebesség, amint ez gyakran megesik, akkor a puffer az indítást követően gyorsan feltöltődik úgy, hogy annak kiürülésétől egyhamar nem kell tartani.

TCP-t vagy UDP-t és akkora átviteli sebességet használva, amely meghaladja a lejátszási sebességet, felmerül a kérdés, hogy a médialejátszó és a médiakiszolgáló a lejátszási ponttól mekkora távolságig hajlandó előrefutni. Gyakran az egész állományt akarják letölteni.

A lejátszási ponttól messzire előremenni azonban sok olyan munka elvégzését jelenti, amire még nincs szükség, amely jelentős tárterületet igényelhet, és amely egyáltalán

nem szükséges a puffer alulcsordulásának kivédéséhez. Ennek elkerülésére az a megoldás, hogy a médialejátszó egy **felső vízjelet (high-water mark)** definiál a pufferben. A kiszolgáló alapvetően továbbra is csak ontja az adatokat egészen addig, amíg a puffer telítettsége el nem éri a felső vízjelet. Ekkor a médialejátszó felszólítja, hogy tartson szünetet. Mivel az adatok ezután egy ideig még tovább özönlenek, amíg a kiszolgáló meg nem kapja a szünetre vonatkozó kérést, ezért a felső vízjel és a puffer vége közt a távolságnak nagyobbnak kell lennie, mint a hálózatban a sávszélesség és a késleltetés szorzata. Miután a kiszolgáló leállt, a puffer elkezd ürülni. Amikor a szintje eléri az alsó vízjelet, a médialejátszó felszólítja a médiakiszolgálót, hogy kezdjen el újból adni. Az alulcsordulás elkerülése érdekében az alsó vízjelnek figyelembe kell vennie a hálózat sávszélesség-késleltetés szorzatát, amikor a médialejátszó felszólítja a médiakiszolgálót a média küldésének folytatására.

A média áramlásának elindítását és leállítását a médialejátszónak a távolból kell tudnia vezérelni – pontosan erre való az RTSP. Ezt az RFC 2326 definiálja, mely egy olyan eljárást biztosít, amellyel a lejátszó vezérelheti a kiszolgálót. A médiafolyam elindításán és leállításán kívül képes egy adott pozíció hátra- vagy előrekeresésére, meghatározott intervallumok lejátszására és gyorsabb vagy lassabb lejátszásra. Noha az RTSP nem gondoskodik az adatfolyamról, ez gyakran az RTP az UDP felett, vagy az RTP a TCP és HTTP felett.

Az RTSP fontosabb parancsait a 7.55. ábra sorolja fel. Egyszerű szöveges formátumúak, mint a HTTP-üzenetek, és általában TCP felett továbbítják ezeket. Az RTSP UDP felett is futhat, mivel minden parancsot nyugtázás követ (és így újra lehet küldeni, ha a nyugtázás elmaradt).

Annak ellenére, hogy a TCP nem tűnik alkalmasnak a valós idejű forgalom megvalósítására, a gyakorlatban gyakran használják. Ennek legfőbb oka az, hogy könnyebben képes átjutni a tűzfalakon, mint az UDP, különösen akkor, ha a HTTP portjával használják. Sok rendszergazda úgy állítja be a tűzfalakat, hogy azok megvédjék a hálózatukat a nem szívesen látott látogatóktól. Csaknem mindig engedélyezik egy távoli gép 80-as portjáról érkező TCP-összeköttetések áthaladását a HTTP és a webes forgalom számára. Ennek a portnak a blokkolása hamar boldogtalan kempingezőket eredményez. A többi port nagy része azonban blokkolva van, beleértve egyebek mellett az RTSP-t és az RTP-t is, amelyek többek között az 554-es és 5004-es portokat használják. Ezért a legkönnyeb módja annak, hogy a folyamszerűen továbbított médiát a tűzfalon keresztül

Parancs	Kiszolgáló tevékenysége
DESCRIBE	Felsorolja a média paramétereit
SETUP	Kiépít egy logikai csatornát a lejátszó és a kiszolgáló között
PLAY	Megkezdi az adatok küldését az ügyfélnek
RECORD	Megkezdi az ügyféltől érkező adatok fogadását
PAUSE	Ideiglenesen leállítja az adatküldést
TEARDOWN	Elengedi a logikai csatornát

7.55. ábra. *RTSP-parancsok a lejátszótól a kiszolgálónak*

megkapjunk, ha a webhely úgy tesz, mintha – legalábbis a tűzfal felől nézve – egy HTTP-kiszolgáló lenne, ami a szokásos HTTP-választ küldi.

A TCP-nek van néhány további előnye is. Mivel a TCP megbízható összeköttetést biztosít, a média teljes példányát adja az ügyfélnek. Ez megkönnyíti a felhasználó dolgát, ha egy korábban megtekintett lejátszási pontra kíván visszalépni, mivel nem kell az elveszett adatok miatt aggódnia. Végül, a TCP annyit fog pufferelni a médiából, amennyit csak lehet, és olyan gyorsan, ahogyan az csak lehetséges. Amikor a puffer olcsó (akkor olcsó, amikor a lemezt használják tárolásra), a médialejátszó letöltheti a médiát, miközben a felhasználó megtekinti azt. Amikor a letöltés megtörtént, a felhasználó megszakítás nélkül nézheti, még akkor is, ha megszakadt az összeköttetése. Ez a tulajdonság hasznos a mobiltelefonok számára, mert az összeköttetés gyorsan változhat mozgás közben.

A TCP hátránya a plusz indítási késleltetés (a TCP-összeköttetés létesítése miatt), és a magasabb alsó vízjel. Ez azonban ritkán okoz problémát, amíg a hálózati sávszélesség nagymértékben meghaladja a média sebességét.

7.4.4. Élő média folyamszerű továbbítása

Nem csak a rögzített mozgóképek örvendenek hihetetlen népszerűségnek a világhálón, az élő média folyamszerű továbbítása, közvetítése[4] is nagyon népszerű. Amint lehetővé vált a hang- és videoanyagok folyamszerű továbbítása az interneten, a kereskedelmi rádió- és televízióállomásoknak az az ötletük támadt, hogy adásukat az éter mellett az interneten is közvetíteni fogják. Nem sokkal később az egyetemi állomások is elkezdték kirakni műsorukat az internetre. Végül már egyes *egyetemi hallgatók* is elkezdték saját műsorukat sugározni az interneten.

Manapság az emberek és mindenféle méretű vállalatok is közvetítenek élő hangot és mozgóképet. Ez a terület az innováció melegágya a technika és a szabványok fejlődése miatt. A nagyobb televízióállomások az élő média folyamszerű továbbítását online jelenlétük megmutatására használják. Ezt **IPTV**-nek (**IP TeleVision** – **IP-televízió**) nevezik. Élő közvetítéseket rádióadók – például a BBC – adásának sugárzására is használnak. Ezt nevezik **internetes rádiónak** (**Internet radio**). Mind az IPTV, mind az internetes rádió olyan eseményekkel érik el a közönséget, melyek a divatbemutatóktól a futball-világkupáig és a Melbourne-i krikettpályáról közvetített élő nemzetközi krikettmérkőzésekig terjednek. A kábelszolgáltatók az IP-hálózaton keresztül történő élő közvetítés technikáját használják saját műsorszóró rendszerük kiépítéséhez. Széles körben alkalmazzák alacsony költségvetésű produkciókhoz, a pornótól a természetfilmekig. A mai technikával gyakorlatilag bárki gyorsan és alacsony költséggel indíthat élő közvetítést.

Az élő média folyamszerű továbbításának egyik megközelítése az, hogy a programokat lemezre mentik. A nézők csatlakozhatnak a kiszolgáló archívumaihoz, feliratkozhatnak bármelyik programra, és letölthetik azt meghallgatásra. A **podcast**[5] egy ilyen módon letöltött epizód. Időzített események esetén az is lehetséges, hogy a tartalmat

4 A továbbiakban a média közvetítése mindig folyamszerű továbbítást jelent. *(A lektor megjegyzése)*

5 A weben rendszertelenül (epizódonként) közzétett audio- vagy videoállományok sorozata. A szó az iPod és a webcast összevonásából származik. *(A lektor megjegyzése)*

közvetlenül az élő közvetítés után tárolják, és az archívum – mondjuk – csak fél órával, vagy még annyival sem lesz lemaradva az élő adástól.

Ez a megközelítés valójában pontosan megegyezik az imént tárgyalt folyamszerű médiaközvetítéssel. Egyszerűen megvalósítható, az összes eddig tárgyalt módszer használható hozzá, és a nézők az archívum teljes műsorválasztékából válogathatnak.

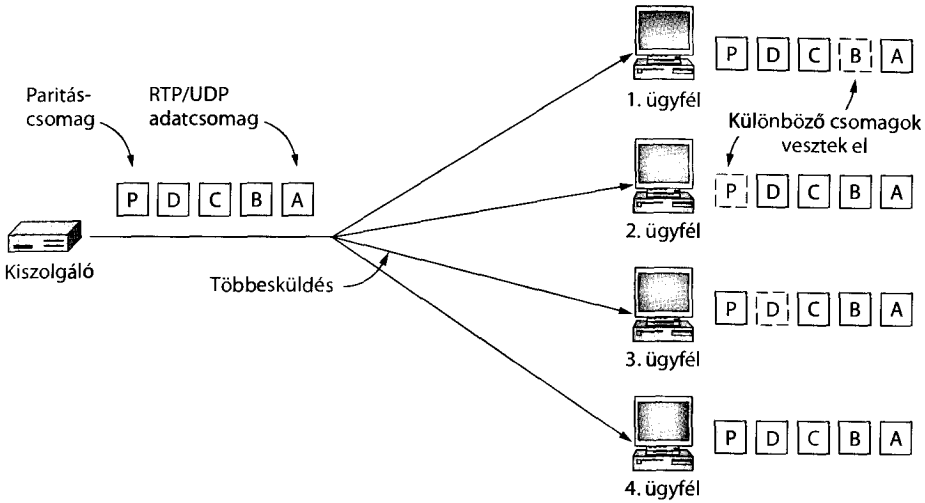
Egy ettől eltérő megoldás az élő internetes közvetítés. A nézők beállítanak egy éppen közvetített médiafolyamot, épp úgy, mintha bekapcsolnák a televíziót. A médialejátszók azonban olyan további képességekkel rendelkeznek, amelyek lehetővé teszik a felhasználó számára a lejátszás szüneteltetését vagy visszatekerését. Az élő média közvetítése tovább folytatódik, és a lejátszó továbbra is pufferelni fogja azt, ameddig erre a felhasználó kész. A böngésző szempontjából ez pontosan úgy néz ki, mint a tárolt média közvetítésének esete. A lejátszónak nem számít, hogy a tartalom egy állományból származik, vagy élőben közvetítik, és ezt a lejátszó általában nem is képes megmondani (kivéve, hogy egy élő közvetítésben nem lehet előre ugrani). Adottak a működés hasonlóságai, a korábban megtárgyaltak nagy része továbbra is használható, de van néhány jelentős különbség is.

Fontos, hogy az ügyfél oldalán a dzsitter kisimításához továbbra is szükség van pufferelésre. Valójában az élő közvetítésekhez gyakran nagyobb méretű pufferre van szükség (függetlenül attól a megfontolástól, hogy a felhasználó szüneteltetheti a lejátszást). Amikor egy állomány az, amelynek a tartalmát közvetítik, a média nagyobb sebességgel is érkezhet, mint a lejátszás sebessége. Így a puffer gyorsan megtelik, ami kiegyenlíti a hálózati dzsittert (és a lejátszó megállítja a folyamot, ha nem kíván több adatot pufferelni). Ezzel ellentétben az élő médiaközvetítést mindig pontosan akkora sebességgel továbbítják, mint amekkora sebességgel előállították, ami egyben megegyezik a lejátszás sebességével is. Ennél gyorsabban nem lehet elküldeni. Következésképpen a puffernek elég nagynak kell lennie a hálózati dzsitter teljes tartományának kezeléséhez. A gyakorlatban 10-15 másodperc indulási késleltetés általában megfelelő, tehát ez nem egy komoly probléma.

Egy további fontos különbség az, hogy az élőben közvetített eseményeknél ugyanazt a tartalmat rendszerint több százan vagy ezren nézik egyszerre. Ilyen körülmények között az élő közvetítések megvalósítására a természetes megoldás a többesküldés használata. Ez nem azonos a tárolt média közvetítésének esetével, mert ott a felhasználók általában minden pillanatban különböző tartalmakat kérnek le. A több felhasználóhoz közvetített média ekkor sok független közvetítési munkamenetből áll, amelyek történetesen egyszerre zajlanak.

A többesküldést alkalmazó közvetítés sémája a következőképpen működik. A kiszolgáló minden médiacsomagot egyszer küld el IP-többesküldés segítségével egy csoportcímre. A hálózat a csoport minden tagjához eljuttatja a csomag egy másolati példányát. Minden olyan ügyfél, amelyik meg akarja kapni a folyamot, korábban csatlakozott a csoporthoz. Az ügyfelek ezt az IGMP segítségével teszik meg ahelyett, hogy RTSP-üzenetet küldenének a médiakiszolgálónak. Ennek az az oka, hogy a médiakiszolgáló már az élő folyamot küldi (de nem az első felhasználó csatlakozása előtt). Arról gondoskodni kell, hogy a folyamot mindenki helyileg megkapja.

Mivel a többesküldés „egy-a-többhöz" típusú kézbesítési szolgáltatás, a médiát az UDP szállítási protokoll felett RTP-csomagok hordozzák. TCP csak egyetlen küldő és



7.56. ábra. *Többesküldéssel megvalósított médiaközvetítés paritáscsomaggal*

egyetlen vevő között működik. Mivel az UDP nem nyújt megbízható szolgáltatást, néhány csomag elveszhet. A médiaveszteség elfogadható szintre csökkentéséhez FEC-t vagy összefésülést használhatunk, ahogy azt már korábban is megtettük.

FEC használata esetén a többesküldésnél előnyös együttműködés van, ami a 7.56. ábra paritásképzést bemutató példáján látható. Ha a csomagokat többesküldéssel továbbítják, a különböző ügyfelek különböző csomagokat veszíthetnek el. Például az 1. ügyfél a *B* csomagot veszítette el, a 2. ügyfél a *P* paritáscsomagot, a 3. ügyfél a *D*-t, a 4. ügyfél pedig egyetlen csomagot sem vesztett el. Annak ellenére azonban, hogy az ügyfelek három különböző csomagot vesztettek el, ebben a példában minden egyes ügyfél képes visszaállítani az összes adatcsomagot. Mindössze arra van szükség, hogy egyetlen ügyfél se veszítsen el egynél több csomagot. Bármelyik is legyen az elvesztett csomag, paritás számításával visszaállítható. Nonnenmacher és mások [1997] munkája leírja, hogyan lehet ezt az elképzelést a megbízhatóság fokozására használni.

Egy sok ügyfelet kiszolgáló szerver számára egyértelműen a média RTP- és UDP-csomagokkal történő többesküldése a leghatékonyabb működési mód. Máskülönben a kiszolgálónak *N* ügyfél esetén *N* folyamot kell továbbítania, amely a kiszolgálónál nagyon nagy hálózati sávszélességet igényel a nagy folyamú események számára.

Talán meglepi önt, amikor azt tanulja, hogy az internet a gyakorlatban nem így működik. Ami általában történik, az az, hogy minden egyes felhasználó külön TCP-összeköttetést hoz létre a kiszolgálóval, és a média ezen az összeköttetésen keresztül folyamszerűen továbbítódik. Az ügyfél számára ez ugyanolyan, mint a tárolt média közvetítése. És ugyanúgy, mint a tárolt média közvetítésének esetében, számos oka van ennek a látszólag rossz választásnak.

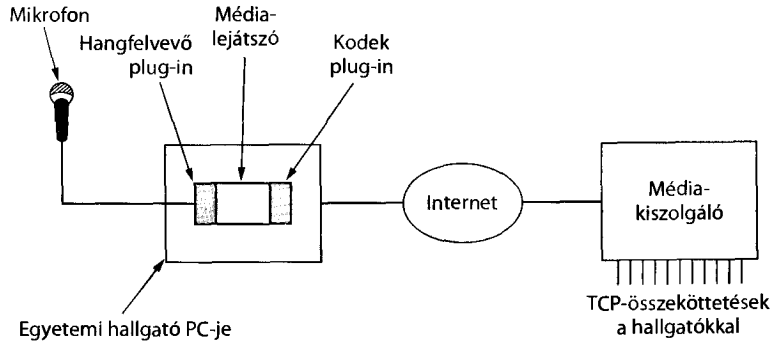
Az első ok az, hogy az IP-többesküldés nem érhető el széles körben az interneten. Néhány internetszolgáltató (ISP) és hálózat belül támogatja, de a hálózatok határain át általában ez a szolgáltatás nem elérhető, pedig szükség lenne erre a nagy távolságra történő közvetítésekhez. A további okok azok az előnyök, amelyekkel a TCP az UDP-vel

szemben rendelkezik, ahogyan azokat korábban már megtárgyaltuk. A TCP-vel megvalósított közvetítés csaknem minden ügyfelet elér az interneten, különösen akkor, amikor HTTP-nek álcázzák a tűzfalakon történő átjutás miatt, a média megbízható továbbítása pedig lehetővé teszi a felhasználók számára, hogy könnyen visszatekerjenek.

Van azonban egy fontos eset, amelyben az UDP és a többesküldés felhasználható a médiaközvetítéshez: ez a szolgáltató hálózatán belüli közvetítés. Például egy kábeltársaság dönthet úgy, hogy a hagyományos videoközvetítés helyett a tv-programokat IP-technikával juttatja el az ügyfelek beltéri egységeihez (set-top box). A video-műsorszórásra használt IP-technikát általában IPTV-nek nevezik, amint azt feljebb megtárgyaltuk. Mivel a kábeltársaság teljesen ellenőrzése alatt tartja a saját hálózatát, kiépítheti azt úgy, hogy támogassa az IP-többesküldést és legyen elegendő sávszélessége az UDP-alapú szétosztáshoz. Mindez az ügyfél számára láthatatlan, mivel az IP-technika csak a szolgáltató **saját hálózatán (walled garden – bekerített kert)** belül létezik.[6] A szolgáltatás szempontjából ez pontosan úgy néz ki, mint egy kábeltévé, de a szolgáltatás mögött IP áll, a beltéri egység egy UDP-t használó számítógép, a tévé pedig egyszerűen csak egy számítógéphez csatlakoztatott monitor.

Visszatérve az internet esetéhez, a TCP felett megvalósított élő közvetítés hátránya az, hogy a kiszolgálónak a média egymástól független másolatait kell elküldenie minden egyes ügyfélnek. Ez nem túl sok ügyfél esetén, különösen hangközvetítésnél, megvalósítható. A trükk az, hogy a kiszolgálót jó internetkapcsolatot biztosító helyre kell elhelyezni, ahol elegendő a sávszélesség. Ez általában egy adatközpontban lévő kiszolgáló bérlését jelenti egy tárhelyszolgáltatótól, és nem egy otthoni kiszolgáló csak széles sávú kapcsolattal történő használatát. A tárhelypiacon nagy a verseny, így ez nem feltétlenül költséges.

Valójában bárki, még egy egyetemista is könnyen felállít és működtet egy folyamszerűen továbbító médiakiszolgálót, például egy internetes rádióállomást. Ennek az állomásnak a főbb alkotóelemei a 7.57. ábrán láthatók. Az állomás lelke egy hagyományos PC tisztességes hangkártyával és mikrofonnal. Népszerű szoftvereket használnak a hang



7.57. ábra. *Egy egyetemista rádióállomása*

6 A témát részletesebben lásd a *http://mediapedia.hu/musorszolgaltato-portal* honlapon. *(A fordító megjegyzése)*

felvételéhez és különféle, például MP4-formátumba történő kódolásához, valamint rendszerint médialejátszókat használnak a hangfelvételek meghallgatásához.

A PC-n létrehozott hangfolyamot ezután az interneten keresztül egy jó hálózati kapcsolattal rendelkező médiakiszolgálóba töltik, hogy podcastokként tárolt állományokat közvetítsen, vagy élő közvetítést valósítson meg. A kiszolgáló a média szétosztását nagyszámú TCP-összeköttetés segítségével kezeli. A kiszolgáló egy webhely formájában felhasználói felületet is biztosít az adóállomásról szóló oldalakkal és a közvetítéssel elérhető tartalomra mutató hivatkozásokkal. Léteznek kereskedelmi szoftvercsomagok, melyek az összes részletet kezelik, de vannak nyílt forráskódú csomagok is, mint például az icecast.

Nagyon nagy számú ügyfél esetén azonban a TCP használata alkalmatlanná válik arra, hogy a médiát minden egyes ügyfélhez egyetlen kiszolgálóról küldje el. Ehhez egyetlen kiszolgálónak egyszerűen nincs elegendő sávszélessége. Nagy médiaközvetítő oldalak esetén a közvetítést földrajzilag szétszórtan elhelyezkedő kiszolgálók halmaza végzi, így egy ügyfél a legközelebbi kiszolgálóhoz csatlakozhat. Ez egy tartalomelosztó hálózat, amit ennek a fejezetnek a végén fogunk tanulmányozni.

7.4.5. Valós idejű konferenciahívás

A hanghívásokat valaha a nyilvános, kapcsolt távbeszélő-hálózaton továbbították, a hálózati forgalom elsősorban hangforgalomból állt, és csak itt-ott volt benne egy kis adatforgalom. Aztán jött az internet és a világháló. Az adatforgalom egyre csak nőtt, és 1999-re akkora lett az adatforgalom, mint a hangforgalom (mivel a hangot ma már digitalizálják, mindkettő mérhető bitekben). 2002-re az adatforgalom mérete egy nagyságrenddel meghaladta a hangforgalom méretét, és még mindig exponenciálisan nő, miközben a hangforgalom szinte stagnál.

Ennek a növekedésnek az lett a következménye, hogy a telefonos hálózat a feje tetejére állt. A hangforgalmat ma internetes technikával továbbítják, és a hálózati sávszélességnek csak egy csekély töredékét jelenti. Ezt a bomlasztó technikát **IP-n keresztül történő hangátvitelként (voice over IP, VoIP)** és **internettelefóniaként (Internet telephony)** ismerik.

Az IP-n keresztül történő hangátvitelt különféle formában használják, melyek mögött erős gazdasági tényezők működnek. (Magyarul: pénzt takarít meg, ezért az emberek használják.) Ennek egyik formája a hagyományos (régimódi?) telefonokra hasonlít, amely az Ethernetre csatlakozik, és hívásokat továbbít a hálózaton keresztül. Pehr Anderson egyetemi hallgató volt az M.I.T.-n, amikor barátaival létrehozta ennek az elgondolásnak a prototípusát egy osztályprojekt keretében. A munkára jó (4) osztályzatot kapott. Nem volt elégedett, ezért NBX néven 1996-ban céget alapított, elsőként alkalmazta ezt a fajta IP-n keresztül történő hangátvitelt, és három évvel később eladta a céget a 3Com-nak 90 millió dollárért. A vállalatok szeretik ezt a megoldást, mert lehetővé teszi számukra, hogy megszüntessék a különálló telefonvonalakat, és a hangátvitelt azzal a hálózattal valósítsák meg, amivel már egyébként is rendelkeznek.

Egy másik megoldás az IP-technikát nagy távolságú telefonos hálózat kiépítésére használja. Az olyan országokban, mint az Egyesült Államok, az ilyen szolgáltatás egymással versengő, nagy távolságú összeköttetéseket biztosító szolgáltatókon keresztül érhető el egy különleges előhívószám tárcsázásával. A hangmintákat a hálózatba injektált csomagokba helyezik, majd amikor a csomagok elhagyják a hálózatot, akkor kiveszik belőlük. Mivel az IP-berendezések sokkal olcsóbbak, mint a telekommunikációs berendezések, ez olcsóbb szolgáltatásokat eredményez.

Mellesleg az árkülönbségnek nem csak technikai okai vannak. A telefonszolgáltatás több évtizeden keresztül szabályozott monopólium volt, ami a telefontársaságoknak a költségeiket egy rögzített százalékkal meghaladó nyereséget garantált. Nem meglepő módon ez a költségek növelésére késztette őket, például sok-sok redundáns hardverrel rendelkeztek, melyet a nagyobb megbízhatósággal indokoltak (a telefonrendszer 40 év alatt összesen csak 2 órára állhatott le, vagy átlagosan évi 3 percre). Ezt a hatást gyakran „aranyozott telefonpózna szindrómaként" emlegetik. A szabályok fellazítása (dereguláció) óta ez a hatás természetesen csökkent, de a megörökölt berendezések még mindig léteznek. Az IT-ipar történetében soha nem fordult elő ilyesmi, így az mindig karcsú és fitt volt.

Mi azonban az IP-n keresztül történő hangátvitelnek arra a formájára fogunk koncentrálni, ami a felhasználók számára valószínűleg a legszembetűnőbb: amikor az egyik számítógépet arra használják, hogy felhívjon egy másik számítógépet. Ez a forma hétköznapivá vált, amint a PC-ket elkezdték mikrofonokkal, hangszórókkal, kamerákkal és a médiafeldolgozáshoz kellően gyors CPU-kkal szállítani, az emberek pedig elkezdtek otthonról, széles sávú átvitellel csatlakozni az internetre. Ennek jól ismert példája a Skype szoftver, amely 2003-ban jelent meg. A Skype és más társaságok is átjárókat biztosítanak annak érdekében, hogy a hagyományos telefonszámok felhívását, valamint az IP-címmel ellátott számítógépek felhívását könnyűvé tegyék.

A hálózati sávszélesség növekedésével a hanghívásokhoz a videohívások is csatlakoztak. Videohívások kezdetben csak a vállalatok területén belül történtek. A videokonferencia-rendszereket arra tervezték, hogy két vagy több helyszín között továbbítsa a mozgóképet, ami lehetővé teszi a különböző helyeken lévő vezetőknek, hogy lássák egymást, miközben a megbeszéléseiket tartják. Jó, széles sávú internet-összeköttetéssel és videotömörítő szoftverrel azonban otthoni felhasználók is videokonferenciázhatnak. Az olyan eszközök, mint a Skype, amelyek kezdetben csak hangátvitelt támogattak, ma már rutinszerűen mozgóképet is tartalmazó hívásokra is képesek, így barátok és családok láthatják és hallhatják is egymást szerte a világon.

A mi szempontunkból az internetes hang- vagy videohívás is egy médiaközvetítési probléma, de olyan, aminek sokkal több kikötésnek kell megfelelnie, mint egy tárolt állomány vagy egy élő esemény közvetítése. További megkötés a kis késleltetés, ami a kétirányú beszélgetéshez szükséges. A telefonhálózatok az elfogadható használathoz legfeljebb 150 ms egyirányú késleltetést engednek meg, e fölötti késleltetésnél a résztvevők elkezdik érzékelni azt, és ez bosszantja őket. (A nemzetközi hívásoknak akár 400 ms késleltetése is lehet, és ezen a ponton messze vannak a pozitív felhasználói élménytől.)

Ilyen kis késleltetést nehéz elérni. 5-10 másodpercnyi média pufferezése biztosan nem fog működni (ami az élő sportesemények közvetítésekor működne). Ehelyett az IP-n keresztül történő video- és hangátviteli rendszereket olyan műszaki megoldásokkal kell megvalósítani, amelyek minimalizálják a késleltetést. Ez a cél azt jelenti, hogy az UDP az egyértelmű választás a TCP helyett, mert a TCP újraküldések legalább egy körülfor-

dulási időnyi késleltetést jelentenek. A késleltetés bizonyos formái azonban nem csökkenthetők, még UDP-vel sem. Például a Seattle és Amszterdam közötti távolság közel 8000 km. A fénysebességű terjedési késleltetés az optikai kábelben ekkora távolságon 40 ms. Sok sikert a megdöntéséhez! A gyakorlatban, a hálózatban a terjedési késleltetés nagyobb lesz, mert nagyobb távolságot ölel fel (mivel a bitek nem követik a szélességi köröket), és továbbítási késleltetések lépnek fel, mert minden IP-útválasztó tárolja és továbbítja a csomagokat. Ez a rögzített késleltetés feléli az elfogadható késleltetés előirányzott mértékét.

A késleltetés másik forrása a csomagmérettel áll összefüggésben. Általában a nagy csomagok jelentik a hálózati sávszélesség kihasználásának legjobb módját, mert ezek hatékonyabbak. 64 kb/s mintavételezési sebességű hang esetében azonban egy 1 KB-os csomagot 125 ms ideig tartana megtölteni (és még tovább, ha a minták tömörítve vannak). Ez a késleltetés elfogyasztaná a teljes késleltetési előirányzat legnagyobb részét. Ráadásul, ha az 1 KB-os csomagot pontosan 1 Mb/s sebességű, széles sávú adatkapcsolaton keresztül küldik el, az átvitel 8 ms-ot fog igénybe venni. Aztán adjunk hozzá még 8 ms-ot, hogy a csomag a túlsó oldalon lévő széles sávú kapcsolaton is áthaladhasson. Nyilvánvaló, hogy nagy csomagokra ez nem fog működni.

Ehelyett a VoIP-rendszerek rövid csomagokat használnak a késleltetés csökkentése érdekében, a sávszélesség-kihasználás hatékonyságának rovására. A hangmintákat kisebb, általában 20 ms-os egységekbe zárják. 64 kb/s-os sebességen ez 160 bájt adatot jelent, tömörítéssel kevesebbet. Definíció szerint azonban az ebből a csomagolásból eredő késleltetés 20 ms lesz. Az átviteli késleltetés is kisebb lesz, mivel a csomag kisebb. A mi példánkban ez körülbelül 1 ms-ra csökkenne. A kis csomagok használatával egy Seattle-ből Amszterdamba tartó csomag legkisebb egyirányú késleltetése az elfogadhatatlan 181 ms-ról (40 + 125 + 16) az elfogadható 62 ms-ra (40 + 20 + 2) csökkent.

Eddig még nem beszéltünk a szoftver által okozott többletteherről, de ez is elhasználja a késleltetési előirányzat egy részét. Ez különösen igaz a mozgóképekre, mert általában tömörítésre van szükség ahhoz, hogy a mozgókép beleférjen a rendelkezésre álló sávszélességbe. Egy tárolt állomány közvetítésével ellentétben itt nincs idő számításigényes kódoló használatára a nagyfokú tömörítéshez. A kódolónak és a dekódolónak is gyorsan kell futnia.

A média mintáinak időben történő lejátszásához továbbra is szükség van pufferre (az érthetetlen beszéd és a szaggatott videó elkerülése érdekében), de a puffer méretét nagyon kis értéken kell tartani, mert a késleltetési előirányzatban megmaradó időt ezredmásodpercekben mérik. Amikor egy csomag megérkezéséhez túl hosszú idő szükséges, a lejátszó átlépi a hiányzó mintákat, és ilyenkor esetleg a környezetéből vett zajt játszik vagy megismétel egy képkockát, hogy elfedje a veszteséget a felhasználó elől. Kompromisszumot kell kötni a dzsitter kezeléséhez használt puffer mérete és a médiából elvesző mennyiség között. Egy kisebb puffer csökkenti a késleltetést, de több csomagvesztést eredményez a dzsitter miatt. Végül, ahogyan a puffer mérete csökken, a veszteség a felhasználó számára úgy válik egyre észrevehetőbbé.

A figyelmes olvasók észrevehették, hogy eddig ebben a szakaszban semmit sem mondtunk a hálózati rétegbeli protokollokról. A hálózat képes csökkenteni a késleltetést, vagy legalábbis a dzsittert, a szolgáltatásminőségi mechanizmusok használatával.

Az az oka annak, hogy ez a kérdés korábban nem merült fel, hogy a médiaközvetítések képesek jelentős késleltetéssel is működni, még élő közvetítés esetén is. Ha a késleltetés nem komoly gond, egy fogadó oldali puffer elegendő a dzsitter problémájának megoldásához. A valós idejű konferenciahívás esetén azonban általában fontos, hogy a hálózat csökkentse a késleltetést és a dzsittert, ezzel segítve a késleltetési előirányzat teljesülését. Az egyetlen olyan alkalom, amikor ez nem lényeges akkor van, amikor olyan nagy a hálózati sávszélesség, hogy mindenki jó minőségű szolgáltatást kap.

Az 5. fejezetben két szolgáltatásminőséget biztosító mechanizmust ismertettünk, amelyek segítenek elérni ezt a célt. Az egyik mechanizmus a DS (Differentiated Services – differenciált szolgáltatások), amelyben a csomagokat különböző osztályokhoz tartozóként jelölik meg, amelyeket különbözőképpen kezelnek a hálózaton. A VoIP-csomagoknak megfelelő jelzés az kis késleltetés. A gyakorlatban a rendszerek beállítják a DS kódpontot a *sürgős továbbítás* (Expedited Forwarding) osztálynak megfelelő jól ismert értékre, *kis késleltetés* (Low Delay) szolgáltatástípussal. Ez különösen hasznos a széles sávú hozzáférési adatkapcsolatoknál, mivel ezek a kapcsolatok hajlamosak rá, hogy torlódás alakuljon ki rajtuk, amikor a webforgalom vagy más forgalom verseng a kapcsolat használatáért. Egy adott stabil hálózati úton a késleltetést és a dzsittert a torlódás megnöveli. Minden 1 KB-os csomagnak egy 1 Mb/s sebességű adatkapcsolaton keresztül történő elküldéséhez 8 ms-ra van szüksége, és egy VoIP-csomag ezeknek a késleltetéseknek teszi ki magát, ha a webes forgalom mögötti várakozási sorban foglal helyet. Egy kis késleltetésjelzéssel azonban a VoIP-csomagok a sor elejére ugranak, megkerülve a webes csomagokat és csökkentve késleltetésüket.

A késleltetés csökkentését szolgáló második mechanizmus szerint meg kell bizonyosodni arról, hogy van elegendő sávszélesség. Ha a rendelkezésre álló sávszélesség változik vagy az átviteli sebesség ingadozik (mint a tömörített mozgóképnél), és néha nincs elegendő sávszélesség, várakozási sorok alakulnak ki, és megnövelik a késleltetést. Ez még DS-nél is előfordulhat. Az elegendő sávszélesség biztosítása érdekében le lehet foglalni a hálózat erőforrásait. Ezt a képességet az integrált szolgáltatások biztosítják. Ez sajnos nem terjedt el széles körben. Ehelyett a hálózatokat egy várható forgalomnagyságra méretezik, vagy a hálózati ügyfeleket egy adott szintű forgalomnak megfelelő szolgáltatási szintű szerződésekkel látják el. Az alkalmazásoknak ez alatt a szint alatt kell működniük, hogy elkerüljék a torlódások kialakulását és a szükségtelen késleltetések megjelenését. Alkalmi otthoni videokonferencia-hívások esetén a felhasználó a sávszélességi igények helyettesítéseként videominőséget választhat, vagy a szoftver ellenőrizheti a hálózati utat és automatikusan választhat egy megfelelő minőséget.

A fenti tényezők közül bármelyik azt okozhatja, hogy a késleltetés elfogadhatatlanná válik, ezért a valós idejű videokonferenciák esetén mindegyikre figyelmet kell fordítani. Az IP-hálózaton keresztül történő hangátvitel áttekintéséhez és ezeknek a tényezőknek az analizálásához lásd Goode [2002] munkáját.

Most, hogy megtárgyaltuk a médiaközvetítés útvonalán keletkező késleltetés problémáját, áttérünk a következő fő gondra, amit a konferenciahívó rendszereknek meg kell oldaniuk. Ez a hívások felépítésének és lebontásának problémája. Két protokollt vizsgálunk meg, amelyeket széles körben alkalmaznak erre a célra: a H.323-at és az SIP-t. A Skype egy másik fontos rendszer, de ennek belső működése szabadalmaztatott.

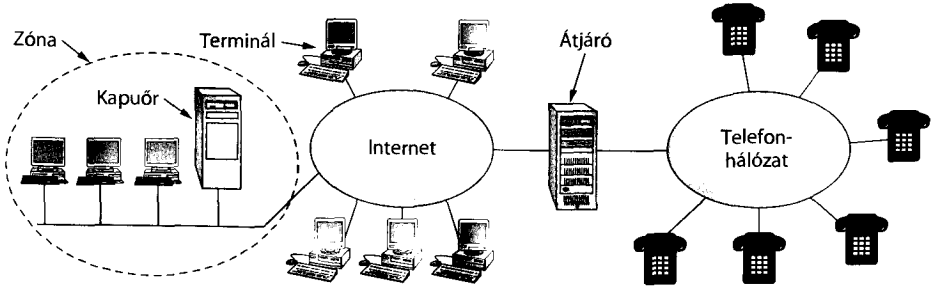
H.323

Egyvalami mindenki számára világos volt már az interneten lebonyolított hang- és videohívások előtt, mégpedig az, hogy ha minden egyes szállító saját protokollkészletet tervez, akkor a rendszer sosem fog működni. Ennek elkerülésére számos érdekelt fél jött össze az ITU védnöksége alatt, hogy szabványokat dolgozzanak ki. 1996-ban az ITU kiadta a **H.323**-as ajánlását, „Nem-garantált szolgáltatásminőség nyújtására szolgáló vizuális telefonrendszerek és helyi hálózati eszközök" címmel. Ilyen címet is csak a távbeszélőipar tud kitalálni. Az 1998-as átdolgozás során gyorsan „csomagalapú multimédia kommunikációs rendszerek"-re változtatták. A H.323 lett az első, széles körben elterjedt internetes telefonrendszerek alapja. Ez is maradt a legszélesebb körben alkalmazott megoldás, 2009-ben készült el a hetedik változata.

A H.323 inkább az internettelefónia architekturális áttekintése, mintsem egy konkrét protokoll. A beszédkódolás, hívásfelépítés, jelzések, adatátvitel stb. területén számos konkrét protokollra hivatkozik ahelyett, hogy maga határozna ezekben a kérdésekben. Az általános modellt a 7.58. ábra mutatja. Középen található az **átjáró** (**gateway**), mely az internetet a telefonhálózattal köti össze. Ez beszéli a H.323-protokollokat az internet felőli oldalon, és a PSTN-protokollokat a telefonos oldalon. Az egymással kommunikáló eszközöket **terminároknak** (**terminals**) nevezzük. Egy LAN-on lehet még **kapuőr** (**gatekeeper**) is, ami a hatáskörébe tartozó végpontokat vezérli, melyek összességét **zónának** (**zone**) nevezik.

A telefonhálózat számos protokollt vesz igénybe. Először is van egy protokoll a hang és mozgókép kódolására és dekódolására. A szabványos telefonok egyetlen, 64 kb/s adatsebességű digitális hangcsatornával történő ábrázolását (8000, 8-bites hangminta másodpercenként) az ITU **G.711** ajánlása definiálja. Az összes H.323 rendszernek támogatnia kell a G.711-et. Más beszédtömörítő protokollok is engedélyezettek, de ezek támogatása nem kötelező. Az ilyen protokollok különböző tömörítő algoritmusokat használnak és különböző kompromisszumot érnek el a minőség és a sávszélesség között. Mozgóképek esetén a fentebb ismertetett MPEG-formátumú tömörítéseket támogatják, beleértve a H.264-et is.

Mivel többfajta tömörítő algoritmus is engedélyezett, egy külön protokoll szükséges ahhoz is, hogy a terminálok megegyezzenek, hogy melyik algoritmust fogják használni. Ez a protokoll a **H.245**. Ez a kapcsolat egyéb szempontjairól, például az adatsebességről



7.58. ábra. *A H.323 architekturális modellje az internettelefónia számára*

Hang	Mozgókép	Vezérlés			
G.7xx	H.26x	RTCP	H.225 (RAS)	Q.931 (Jelzés)	H.245 (Hívás-vezérlés)
RTP					
UDP				TCP	
IP					
Adatkapcsolati réteg protokollja					
Fizikai réteg protokollja					

7.59. ábra. *A H.323 protokollkészlet*

való tárgyalást is lehetővé teszi. Az RTP-csatornák vezérléséhez RTCP szükséges. Kell még egy protokoll a kapcsolatok kiépítéséhez és lebontásához, a tárcsahangok biztosításához, a csengetési hangokhoz és a hagyományos telefónia többi részéhez. Itt az ITU **Q.931**-et használják. A terminároknak szükségük van egy olyan protokollra is, mellyel a kapuőrökhöz beszélhetnek (ha vannak ilyenek). Erre a **H.225** használatos, mely a **RAS** (**Registration/Admission/Status – regisztráció/beléptetés/státus**) nevű PC–kapuőr csatornát kezeli. Ez a csatorna lehetővé teszi, hogy a terminálok csatlakozzanak egy zónához, vagy hogy kilépjenek onnan, hogy sávszélességet kérjenek vagy adjanak vissza, frissítsék a státusukat stb. Végül kell egy protokoll a tényleges adatátvitelre is. Erre a célra RTP-t használnak, amit szokás szerint az RTCP kezel. Mindezen protokollok elhelyezkedését a 7.59. ábra mutatja.

Ahhoz, hogy lássuk, hogyan kapcsolódnak össze ezek a protokollok, vegyük azt az esetet, melyben egy PC terminál LAN-on (kapuőrrel) egy távoli telefont hív. A PC-nek először meg kell találnia a kapuőrt, ezért egy UDP kapuőr-felfedezési csomagot sugároz a 1718-as porton. Ha a kapuőr válaszol, a PC megtudja az IP-címét. Ezután elküld a kapuőrnek egy RAS-üzenetet egy UDP-csomagban, hogy regisztrálja magát nála. Ha ezt elfogadta, a PC elküld a kapuőrnek egy RAS-beléptetőcsomagot, melyben sávszélességet kér. A hívás csak akkor kezdődhet, ha ezt elfogadta. A sávszélesség előre való lefoglalása lehetővé teszi a kapuőr számára, hogy korlátozza a hívások számát. Így elkerülhető, hogy túl legyen jegyezve a kimeneti vonal, és biztosítható a szükséges szolgáltatásminőség.

Mellesleg a telefonrendszerek ugyanígy működnek. Amikor felemeljük a kagylót, egy jelzés fut be a helyi központba. Ha a központnak van elég szabad kapacitása egy másik híváshoz, előállítja a tárcsahangot. Ha nincs, akkor nem fogunk hallani semmit. Manapság a rendszer annyira túldimenzionált, hogy a tárcsahang csaknem azonnal megérkezik, de a telefonálás hajnalán ez gyakran néhány másodpercet vett igénybe. Tehát ha az unokák majd egyszer megkérdezik, „Miért vannak tárcsahangok?", most már tudja. Kivéve, ha akkorra már valószínűleg telefon sem lesz.

A PC erre kiépít egy TCP-összeköttetést a kapuőrrel, hogy megkezdje a hívásfelépítést. A hívásfelépítés a meglevő telefonhálózati protokollokat használja, melyek összeköttetés-alapúak, ezért van szükség a TCP-re. A telefonrendszerben ellenben nincs semmi olyasmi, mint az RAS, amivel a telefonok a jelenlétüket bejelenthetnék, ezért a

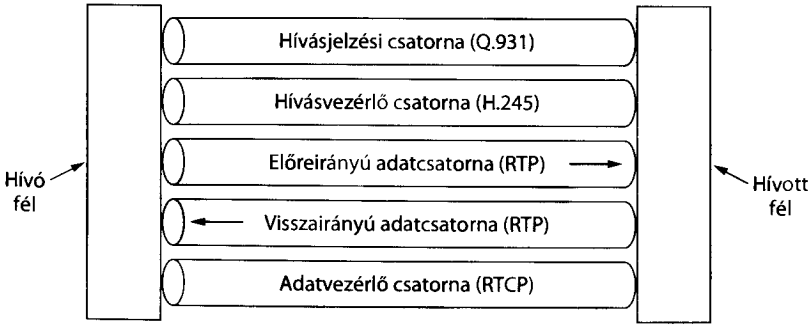
H.323 tervezői szabadon választhattak az UDP és a TCP közül az RAS számára, és végül a kevésbé költséges UDP-t választották.

Most, hogy már a sávszélességet is kiosztották, a PC elküldhet egy Q.931 *SETUP* üzenetet a TCP-összeköttetésen keresztül. Ez az üzenet meghatározza a hívott készülék telefonszámát (vagy egy IP-címet és egy portot, ha számítógépet hívtak). A kapuőr egy Q.931 *CALL PROCEEDING* üzenettel válaszol, hogy nyugtázza a kérés helyes vételét, majd továbbítja a *SETUP* üzenetet az átjárónak.

Az átjáró, ami félig számítógép, félig telefonos kapcsoló, egy hagyományos telefonhívás segítségével felhívja a keresett (hagyományos) telefont. Az a helyi központ, amelyre a telefont kötötték, megcsöngeti a készüléket, és visszaküld egy Q.931 *ALERT* üzenetet, hogy tudassa a PC-vel, hogy megkezdődött a csengetés. Amikor a másik oldalon felveszik a telefont, a helyi központ visszaküld egy Q.931 *CONNECT* üzenetet, hogy jelezze a PC-nek, hogy az összeköttetés kiépült.

Amint megvan az összeköttetés, a kapuőr kimarad a körből, de nem így az átjáró. Az ezután következő csomagok elkerülik a kapuőrt, és közvetlenül az átjáró IP-címére érkeznek. Ezen a ponton csak egy sima cső köti össze a két felet. Ez egy fizikai rétegbeli összeköttetés, mely pusztán biteket visz át, semmi több. Egyik fél sem tud semmit a másikról.

Ezután a H.245 protokollt használják, hogy egyezkedjenek a hívás paramétereiről. Ez a protokoll a H.245 vezérlőcsatornát használja, mely mindig nyitva áll. Mindkét oldal azzal kezdi, hogy bejelenti a képességeit, például, hogy tud-e kezelni mozgóképeket (a H.323 erre is képes) vagy konferenciahívásokat, milyen kodekeket támogat stb. Amikor mindkét oldal megtudta, hogy mire képes a másik, két egyirányú adatcsatornát állítanak fel, és mindegyikhez egy kodeket és egyéb paramétereket rendelnek. Mivel a berendezések a két oldalon különbözők lehetnek, minden további nélkül lehetséges, hogy az előre- és visszairányú csatornákon különböző kodekeket használnak. A tárgyalások befejezése után megindulhat az adatfolyam az RTP felhasználásával. Ezt az RTCP kezeli, melynek a torlódáskezelésben van szerepe. Ha mozgókép is van, akkor az RTCP kezeli a hang/kép szinkronizációt. A különböző csatornákat a 7.60. ábra mutatja. Ha bármelyik fél lerakja a kagylót, a hívás befejeződött. Ezt követően a Q.931 hívásjelzési csatorna segítségével lebontják az összeköttetést, hogy felszabadítsák azokat az erőforrásokat, amelyekre többé nincs szükség.



7.60. ábra. *Logikai csatornák a hívó és a hívott fél között a hívás ideje alatt*

Amikor a hívás véget ért, a hívó PC egy RAS-üzenettel újból megkeresi a kapuőrt, hogy visszaadja a kapott sávszélességet, de kezdeményezhet esetleg egy újabb hívást is.

Még semmit nem szóltunk a H.323 részét képező szolgáltatásminőségről, annak ellenére, hogy azt mondtuk, ez fontos részét képezi a valós idejű konferenciahívás sikerre vitelének. Ennek az az oka, hogy a QoS kívül esik a H.323 körein. Ha a hívások alapjául szolgáló hálózat képes megbízható, dzsittermentes összeköttetést kiépíteni a hívó PC-től az átjáróig, akkor a hívás során jó lesz a szolgáltatásminőség, egyébként nem. A telefonos oldalon azonban a hívás minden része dzsittermentes lesz, mert a telefonhálózatot így tervezték meg.

SIP – viszonykezdeményező protokoll

A H.323-at az ITU dolgozta ki. Az internetes közösségből sokan tipikus telefontársasági terméknek: nagynak, bonyolultnak és rugalmatlannak tekintették. Következésképp az IETF is felállított egy bizottságot, hogy kidolgozzon egy egyszerűbb és még moduláris megoldást az IP-n keresztül történő hangátvitelre. A munka legfőbb eredménye az **SIP** (**Session Initiation Protocol – viszonykezdeményező protokoll**) lett. A legutolsó változatot, ami 2002-ben készült, az RFC 3261 írja le. A protokoll leírja, hogyan kell internetes telefonhívásokat, videokonferenciákat és más multimédiás összeköttetéseket felépíteni. A H.323 teljes protokollkészlet, míg az SIP csak egyetlen modul, de ezt arra tervezték, hogy jól együtt tudjon működni a meglevő internetes alkalmazásokkal. A telefonszámokat például URL-ként definiálja, hogy weboldalakon is fel lehessen tüntetni azokat, és egy kattintással lehessen hívást kezdeményezni (ugyanúgy, ahogyan a *mailto* sémával egy kattintással elő lehet hozni egy programot az e-levél küldéséhez).

Az SIP-vel két résztvevős (hagyományos telefonhívások), több résztvevős (mindenki beszélhet és hallható is) és többesküldéses (egy adó, több vevő) kapcsolatokat is létre lehet hozni. A kapcsolatokban lehet hang-, mozgókép- vagy adatforgalom is, ez utóbbi például a többszereplős, valós idejű játékoknál hasznos. Az SIP csak a kapcsolatok kiépítéséért, kezeléséért és lebontásáért felelős; az adatátvitelt más protokollok végzik, mint például az RTP/RTCP. Az SIP az alkalmazási réteg protokollja, és igény szerint futhat UDP vagy TCP fölött is.

Az SIP különféle szolgáltatásokat támogat, beleértve a hívott fél felkutatását (aki lehet, hogy nem az otthoni gépénél ül) és a hívott fél képességeinek meghatározását, valamint a hívásfelépítés és -bontás eljárásait. A legegyszerűbb esetben az SIP egy kapcsolatot épít fel a hívó és a hívott fél gépe között, először tehát ezt az esetet fogjuk tanulmányozni.

Az SIP-ben a telefonszámokat URL-ként ábrázolják az *sip* séma használatával. Az *sip:ilse@cs.university.edu* utalhat például egy Ilse nevű felhasználóra a *cs.university.edu* DNS-nevű hoszton. Az SIP URL-ek tartalmazhatnak IPv4-címeket, IPv6-címeket vagy tényleges telefonszámokat is.

Az SIP a HTTP mintájára készült szöveges protokoll. Az egyik fél elküld egy ASCII-szöveges üzenetet, mely egy metódus nevét tartalmazza az első sorban, amit további sorok követnek az átadandó paramétereket tartalmazó fejlécekkel. Sok fejlécet a MIME-ből vettek át, hogy az SIP együtt tudjon működni a meglevő internetes alkalmazásokkal. Az alapspecifikáció által definiált hat eljárást a 7.61. ábra sorolja fel.

Eljárás	Leírás
INVITE	A munkamenet létrehozását kéri
ACK	Megerősíti a munkamenet létrejöttét
BYE	A munkamenet lebontását kéri
OPTIONS	Lekérdezi egy hoszt képességeit
CANCEL	Visszavon egy függőben lévő kérést
REGISTER	Értesít egy átirányító kiszolgálót a felhasználó jelenlegi helyéről

7.61. ábra. *SIP-metódusok*

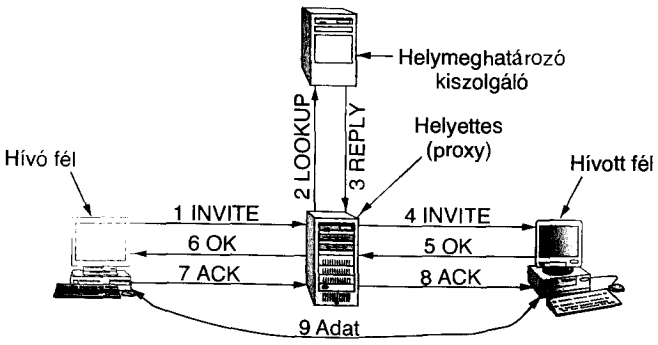
Egy munkamenet kiépítéséhez vagy a hívó hoz létre egy TCP-összeköttetést a hívott féllel, és azon küldi át az *INVITE* üzenetet, vagy pedig egy UDP-csomagban küldi át az *INVITE*-ot. A fejlécek mindkét esetben a második és az azt követő sorokban írják le az üzenet törzsének szerkezetét, mely a hívó képességeit, médiatípusait és formátumait tartalmazza. Ha a hívott fél fogadja a hívást, akkor egy HTTP-típusú válaszkóddal válaszol (ez egy háromjegyű szám, mely a 7.38. ábra csoportjait használja, például 200 jelenti az elfogadást). A válaszkód sora után a hívott fél is információt adhat a képességeiről, médiatípusairól és formátumairól.

Az összeköttetés egy „háromutas kézfogás" révén épül ki, vagyis a hívó végül egy *ACK* üzenet visszaküldésével nyugtázza a 200-as üzenet vételét.

Bármelyik fél kérheti a munkamenet bontását egy olyan üzenet elküldésével, mely a *BYE* eljárást tartalmazza. Ha a másik oldal ezt nyugtázza, akkor a viszony véget ér.

Az *OPTIONS* metódus egy gép képességeinek lekérdezésére való. Ezt rendszerint a munkamenet létrejötte előtt használják, hogy megtudják, hogy a kérdéses gép egyáltalán képes-e IP-n keresztül történő hangátvitelre vagy bármely adott viszonytípusra.

Az SIP lehetőséget ad egy otthonától éppen távol lévő felhasználó követésére és elérésére – ehhez a képességhez kötődik a *REGISTER* metódus. Ezt az üzenetet egy SIP helymeghatározó kiszolgálónak küldik el, ami nyomon követi, hogy ki hol tartózkodik



7.62. ábra. *Helyettes kiszolgáló és átirányítás használata SIP-vel*

éppen. Ettől a kiszolgálótól később le lehet kérdezni a felhasználó aktuális tartózkodási helyét. Az átirányítás műveletét a 7.62. ábra szemlélteti. Itt a hívó elküld egy *INVITE* üzenetet a helyettes kiszolgálónak, mely elrejti az esetleges átirányítást. A helyettes ezután megkeresi a felhasználó tartózkodási helyét, és oda küldi az *INVITE* üzenetet. Ezt követően átjátszóként működik a további üzenetek számára a „háromutas kézfogásban". A *LOOKUP* és *REPLY* üzenetek nem részei az SIP-nek; itt tetszés szerinti protokoll használható a helymeghatározó kiszolgálótól függően.

Az SIP-nek számos olyan funkciója van, amit most nem tárgyalunk, mint például a hívásvárakoztatás, hívásszűrés, titkosítás és hitelesítés. A protokollnak megvan az a képessége is, hogy hívásokat létesítsen egy számítógépről egy hagyományos telefonra, ha rendelkezésre áll egy megfelelő átjáró az internet és a telefonrendszer között.

A H.323 és az SIP összehasonlítása

A H.323 és az SIP is lehetővé teszi a két résztvevős és több résztvevős hívásokat mind számítógépes, mind telefonos végpontok között. Mindkettő támogatja a paraméterekkel kapcsolatos egyezkedéseket, a titkosítást és az RTP/RTCP-protokollokat. A különbségek és hasonlóságok összegzését a 7.63. ábra adja meg.

Tétel	H.323	SIP
Ki dolgozta ki	ITU	IETF
PSTN-nel való kompatibilitás	Igen	Nagyban
Kompatibilitás az internettel	Nem	Igen
Architektúra	Monolitikus	Moduláris
Teljesség	Teljes protokollkészlet	Az SIP csak a felépítést kezeli
Egyeztetés a paraméterekről	Igen	Igen
Hívásjelzés	Q.931 TCP felett	SIP TCP vagy UDP felett
Üzenetformátum	Bináris	ASCII
Médiaátvitel	RTP/RTCP	RTP/RTCP
Több résztvevős hívások	Igen	Igen
Multimédia-konferenciák	Igen	Nem
Címzés	URL vagy telefonszámok	URL
Hívásbontás	Explicit vagy TCP bontás	Explicit vagy időzítő lejárta
Közvetlen üzenetküldés	Nem	Igen
Titkosítás	Igen	Igen
A szabványok mérete	1400 oldal	250 oldal
Megvalósítás	Nagy és bonyolult	Mértéktartó, de problémás
Helyzet	Széles körben alkalmazott, különösen mozgóképekhez	Főleg a hangátvitel esetén jelent alternatívát

7.63. ábra. *A H.323 és az SIP összehasonlítása*

Bár a funkciókészletek hasonlók, a két protokoll filozófiája nagyban eltér. A H.323 egy tipikus, nehézsúlyú távbeszélő-ipari szabvány, amely meghatározza a teljes protokollkészletet, és pontosan definiálja, hogy mit szabad és mit nem. Ez a megközelítés ahhoz vezet, hogy a protokollok minden rétegben nagyon jól meghatározottak, ami megkönnyíti az átjárhatóságot is. Ennek az ára egy nagy, bonyolult és merev szabvány, amit nehéz a jövőbeli alkalmazásokhoz igazítani.

Az SIP ezzel szemben egy tipikus internetes protokoll, amely rövid ASCII-sorok cseréjével működik. Ez egy könnyűsúlyú modul, ami jól együtt tud működni más internetes protokollokkal, de kevésbé jól a már meglevő telefonos jelzési protokollokkal. Az IETF-féle IP-s hangátvitel különösen moduláris, rugalmas és könnyen igazítható az új alkalmazásokhoz. A hátránya az, hogy folyamatosan átjárhatósági problémák merülnek fel, amikor az emberek megpróbálják értelmezni a szabvány jelentését.

7.5. Tartalomszállítás

Az interneten minden a kommunikációról szokott szólni, akárcsak a telefonhálózat esetében. Régen egyetemi oktatók kívántak távoli gépekkel kommunikálni, a hálózaton keresztül bejelentkezni, hogy feladatokat oldjanak meg. Az emberek sokáig e-leveleket használtak az egymással történő kommunikációhoz, de ma már IP-hálózaton keresztül mozgókép- és beszédátvitelt is használnak. A web széles körű elterjedése óta azonban az internet egyre inkább nem a kommunikációról, hanem a tartalomról szól. Sokan használják a világhálót információ felkutatására, és hatalmas mennyiségben fordul elő az egyenrangú társak közötti (P2P) állománymegosztás, amit a filmekhez, a zenéhez és a programokhoz való hozzáférés ösztönöz. A tartalomra történő átállás annyira hangsúlyos, hogy az internet sávszélességének nagy részét ma tárolt mozgóképek továbbítására használják.

Mivel a tartalom szétosztásának feladata nem ugyanaz, mint a kommunikációé, ezért eltérő követelményeket is támaszt a hálózattal szemben. Például, ha Sanyi beszélni kíván Jancsival, felhívhatja IP-hálózaton keresztül a mobiltelefonját. A kommunikációnak egy bizonyos számítógéppel kell megtörténnie; nem lenne jó Pali számítógépét hívni. De ha Jancsi meg kívánja nézni csapatának legutóbbi krikettmérkőzését, bármelyik számítógépről szívesen fogadja a videoközvetítést, amelyik képes ezt a szolgáltatást nyújtani. Nem érdekli, hogy a számítógép vajon Sanyié vagy Palié vagy, ami még valószínűbb, egy ismeretlen kiszolgálóé az interneten. Azaz, a hely a tartalom szempontjából nem számít, kivéve, ha az befolyásolja a teljesítőképességet (és a jogszerűséget).

A másik különbség az, hogy néhány webhely, amely tartalmat szolgáltat, rettenetesen népszerűvé vált. Ilyen a YouTube. Ez lehetővé teszi a felhasználóknak, hogy megosszák a mindenféle elképzelhető témakörbe tartozó, saját készítésű mozgóképeiket. Sokan meg akarják ezt tenni. A többiek meg akarják nézni azokat. Úgy becsülik, hogy a YouTube ezekkel a sávszélesség-igényes videókkal adja ma az internetes forgalomnak akár 10%-át.

Nincs egyetlen olyan kiszolgáló sem, amely eléggé nagy teljesítőképességű vagy megbízható lenne ilyen megdöbbentő mértékű igény kezeléséhez. Ehelyett a YouTube és más nagy tartalomszolgáltatók megépítették a saját tartalomelosztó hálózatukat. Ezek a hálózatok a világon szétszórtan elhelyezkedő adatközpontokat használnak arra, hogy a rendkívül nagy számú ügyfelet jó teljesítménnyel és rendelkezésre állással szolgálják ki tartalommal.

A tartalomelosztáshoz használt technikák az idők során fejlődtek ki. A világháló növekedésének kezdeti szakaszában csaknem a népszerűsége lett a veszte. A tartalom iránti megnövekedett igény oda vezetett, hogy a kiszolgálók és a hálózat gyakran váltak túlterheltté. A www-t sokan már Világméretű Várakozásként (World Wide Wait) kezdték emlegetni.

A fogyasztói igényekre adott válaszként az internet magját nagyon nagy sávszélességgel látták el, és gyorsabb széles sávú összeköttetéseket alakítottak ki a hálózat szélén. Ez a sávszélesség volt a teljesítőképesség javításának kulcsa, de ez csak része a megoldásnak. A végtelen késleltetések csökkentése érdekében a kutatók különböző architektúrákat is kidolgoztak a sávszélességnek tartalomelosztáshoz történő használatára.

Az egyik architektúra a **CDN (Content Distribution Network – tartalomelosztó hálózat)**.[7] Ebben egy szolgáltató egy elosztott gépekből álló gyűjteményt hoz létre az interneten belül, és arra használja ezeket, hogy az ügyfeleket tartalommal szolgálja ki. Ez a nagy játékosok választása. Egy alternatív architektúra a **P2P- (Peer-to-Peer – egyenrangú)** hálózat. Ebben egy számítógépekből álló gyűjtemény egyesíti erőforrásait, hogy tartalmat szolgáltassanak egymásnak külön létesített kiszolgálók vagy bármilyen központi ellenőrzőpont nélkül. Ez az elképzelés megragadta az emberek képzeletét, mert a sok kis játékos együttműködve hatalmas ütést vihet be.

Ebben a szakaszban megvizsgáljuk az interneten történő tartalomelosztás problémáját és néhány gyakorlatban alkalmazott megoldást. Miután röviden megtárgyaltuk a tartalom népszerűségét és az internetes forgalmat, leírjuk, hogyan lehet nagy teljesítményű webszervereket építeni és gyorstárazást használni a webügyfeleknek nyújtandó teljesítőképesség javítása érdekében. Azután elérkezünk a tartalomelosztás két fő architektúrájához: a CDN-ekhez és a P2P-hálózatokhoz. Látni fogjuk, hogy ezek felépítése és tulajdonságai teljesen különbözők.

7.5.1. Tartalom és internetes forgalom

A jól működő hálózatok tervezéséhez és megépítéséhez szükségünk van annak a forgalomnak a megértésére, amit a hálózatoknak továbbítaniuk kell. Például, a tartalomra történő váltással a kiszolgálókat a vállalati irodákból az internetes adatközpontokba költöztették, amely nagy számú gépnek biztosít kiváló hálózati kapcsolatot. Manapság még egy kis kiszolgáló üzemeltetéséhez is könnyebb és olcsóbb egy internetes adatközpontban bérelni egy virtuális kiszolgálót, mint otthon vagy az irodában működtetni egy valódi gépet széles sávú internetkapcsolattal.

Szerencsére csak két olyan tényező van az internetes forgalommal kapcsolatban, amit lényeges tudni. Az első tényező az, hogy a forgalom gyorsan változik, nem csak részleteiben, hanem a teljes összetételében is. 1994 előtt a forgalom legnagyobb része

7 A CDN-re két elnevezés terjedt el. Az egyik a Content Distribution Network (tartalomelosztó hálózat), a másik Content Delivery Network (tartalomszállító hálózat). *(A lektor megjegyzése)*

hagyományos FTP-állományátvitel (programok és adathalmazok számítógépek közötti mozgatása) és e-levél volt. Majd megérkezett és exponenciálisan növekedett a világháló. A webforgalom már jóval a 2000. évi dotcom lufi előtt megelőzte az FTP és az e-levél forgalmat. 2000 körül elkezdődött a zene, majd a film állományok P2P-megosztása. 2003-ra az internet forgalmának nagy részét a P2P-forgalom tette ki, háttérbe szorítva a webet. Valamikor a 2000-es évek végén a YouTube-hoz hasonló helyek által tartalomelosztó módszerekkel közvetített mozgókép forgalma elkezdte felülmúlni a P2P-forgalmat. A Cisco azt jósolja, hogy 2014-re az összes internetes forgalom 90%-a ilyen vagy olyan formában mozgókép lesz [Cisco, 2010].

Nem mindig a forgalom nagysága számít. Például, noha az IP-hálózaton történő beszédátvitel még a Skype 2003-as indulása előtt fellendült, ez mindig csak egy apró jel lesz a diagramon, mert a hang sávszélesség-igénye két nagyságrenddel kisebb, mint a mozgóképé. A VoIP-forgalom azonban másképpen veszi igénybe a hálózatot, mert érzékeny a késleltetésre. Egy másik példa: az online közösségi hálózatok intenzíven növekedtek a Facebook 2004-es indulása óta. A Facebook 2010-ben először, több felhasználót ért el naponta a világhálón, mint a Google. Még a forgalmat félretéve is (és ez egy szörnyen nagy forgalom) fontosak az online közösségi hálózatok, mert megváltoztatják azt a módot, ahogyan az emberek az interneten keresztül kapcsolatba lépnek egymással.

A lényeg az, hogy gyorsan, és bizonyos rendszerességgel szeizmikus változások történnek az internetes forgalomban. Mi jön legközelebb? Kérjük, nézze majd meg ennek a könyvnek a következő kiadását, és tudatni fogjuk önnel.

A második lényeges tényező az internetes forgalommal kapcsolatban, hogy az nagyon aszimmetrikus. Sok számunkra ismerős tulajdonság értéke egy átlag körül csoportosul. Például a legtöbb felnőtt közel átlagos testmagasságú. Akad néhány magas és alacsony ember, de csak kevés nagyon magas vagy nagyon alacsony. Az ilyen tulajdonságok esetén lehet egy olyan testmagasság tartománnyal tervezni, ami nem nagyon nagy, de ennek ellenére felöleli a populáció többségét.

Az internet forgalma nem ilyen. Hosszú időn át ismert volt, hogy létezik néhány nagy forgalmú webhely, és rengeteg webhely sokkal kisebb forgalommal. Ez a sajátosság a hálózati nyelv részévé vált. A korai dokumentumok a forgalmat **csomagvonatokban** (**packet train**) fejezték ki, mert az volt az elképzelés, hogy a forgalom olyan, mintha hirtelen expresszvonatok mennének végig a kapcsolaton rengeteg csomaggal [Jain és Routhier, 1986]. Ezt később az **önhasonlóság** (**self-similarity**) fogalmával formalizálták, amire a mi szempontunkból úgy lehet tekinteni, mint a hálózati forgalomra, ami sok rövid és sok hosszú rést mutat, még akkor is, amikor különböző időskálákon tekintjük meg [Leland és mások, 1994]. A későbbi munkák a hosszan áramló forgalomról **elefántokként** (**elephants**), a rövid ideig áramló forgalomról pedig **egerekként** (**mice**) beszéltek. Az elképzelés az, hogy csak néhány elefánt van és sok egér, de az elefántok a fontosak, mert annyira nagyok.

Visszatérve a webtartalomhoz, az ugyanilyen fajta aszimmetria nyilvánvaló. A videokölcsönzők, a közkönyvtárak és más hasonló szervezetek tapasztalatai azt mutatják, hogy nem minden tétel egyformán népszerű. Kísérletileg, ha N film érhető el, akkor a k. legnépszerűbb filmre vonatkozó kérések számának hányada közelítőleg C/k, ahol C-t úgy számítják, hogy az összeget 1-re normálják, vagyis:

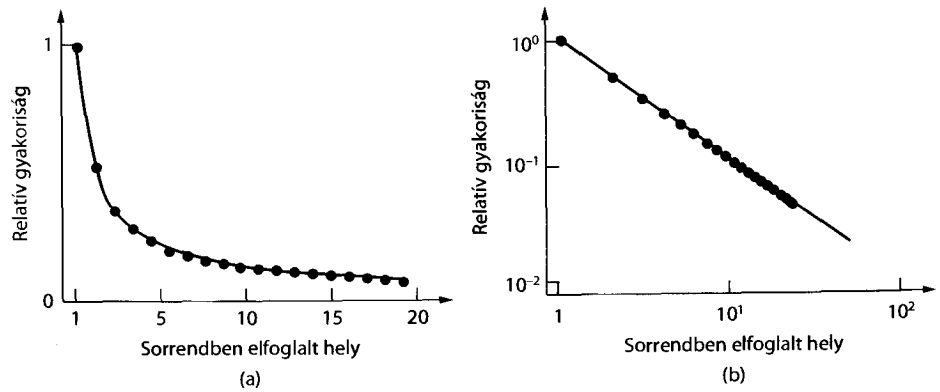
$$C = 1/(1 + 1/2 + 1/3 + 1/4 + 1/5 + \ldots + 1/N)$$

Ebből adódóan a legnépszerűbb film hétszer olyan népszerű, mint a hetedik legnépszerűbb. Ez az eredmény mint **Zipf-féle törvény** (**Zipf's law**) ismeretes [Zipf, 1949]. A nevét George Zipfről, a Harvard Egyetem nyelvtudomány professzoráról kapta, aki megfigyelte, hogy egy szó használatának gyakorisága egy nagyméretű szövegben fordítottan arányos annak sorrendben elfoglalt helyével. Például a 40. leggyakoribb szót kétszer annyi alkalommal használják, mint a 80. leggyakoribb szót, és háromszor annyi alkalommal, mint a 120.-at.

A Zipf-eloszlást a 7.64.(a) ábra mutatja. Ez azt az elképzelést ragadja meg, hogy kevés népszerű dolog van, és sok népszerűtlen. Az ilyen eloszlási forma felismeréséhez kényelmes az adatokat olyan diagramon megrajzolni, amelynek mindkét tengelye logaritmikus osztású, ahogyan az a 7.64.(b) ábrán látható. Az eredménynek egy egyenes vonalnak kell lennie.

Amikor megvizsgálták a weboldalak népszerűségét, kiderült, hogy az nagyjából követi Zipf törvényét [Breslau és mások, 1999]. A Zipf-eloszlás a **hatványfüggvényként** (**power law**) ismert eloszlások családjának egyik példája. A hatványfüggvények sok emberi jelenséggel kapcsolatban nyilvánvalók, mint például a városok népességének eloszlása és a vagyon eloszlása. Ezek ugyanúgy hajlamosak a kevés nagy játékos és a rengeteg kis játékos leírására, és ezek is egyenes vonalként jelennek meg egy log-log diagramon. Hamarosan felfedezték, hogy az internet topológiáját nagyjából le lehet írni hatványfüggvények segítségével [Faloutsos és mások, 1999]. Ezután a kutatók nekiálltak az internet összes elképzelhető tulajdonságát logaritmikus skálán ábrázolni, meglátván az egyenes vonalat felkiáltottak: „hatványfüggvény!"

Ami azonban jobban számít, mint az egyenes vonal a log-log diagramon, az ezeknek az eloszlásoknak a jelentése a hálózatok tervezése és használata szempontjából. A sokféle tartalom ismeretében, amelyek Zipf- vagy hatványfüggvény-eloszlásúak, alapvetőnek tűnik, hogy a webhelyek az interneten népszerűség szempontjából Zipf-szerűek. Ez viszont azt jelenti, hogy az átlagos webhely nem hasznos ábrázolás. A webhelyeket jobban jellemzi, hogy népszerűek vagy népszerűtlenek. Mindkét fajta webhely számít. A népszerű helyek nyilvánvalóan fontosak, mivel néhány népszerű webhely felelhet az internet forgalmának nagy részéért. Talán meglepő, de a népszerűtlen webhelyek is lényegesek.



7.64. ábra. *Zipf-eloszlás (a) lineáris skálán (b) log-log skálán*

Ennek az az oka, hogy a népszerűtlen helyekre irányuló összesített forgalommennyiség a teljes forgalomnak nagy hányadát teheti ki. Ennek az az oka, hogy oly sok népszerűtlen webhely létezik. Azt az elképzelést, hogy a sok népszerűtlen választás együtt fontos lehet, olyan könyvek népszerűsítették, mint például a *Hosszú farok* [Anderson, 2008a].

Az olyan csökkenő jellegű görbék, mint amilyen a 7.64.(a) ábrán látható, gyakoriak, de nem mind egyformák. Különösen azok a helyzetek mutatnak **exponenciális csökkenést (exponential decay)**, amelyekben a csökkenés sebessége arányos a megmaradó anyag mennyiségével (mint például az instabil radioaktív atomok esetén), és amelyek sokkal gyorsabban csökkennek, mint ami Zipf törvényéből következik. A t idő után maradó tételek – mondjuk atomok – számát általában az $e^{-t/\alpha}$ képlettel fejezik ki, ahol az α konstans határozza meg, hogy milyen gyors a csökkenés. Az exponenciális csökkenés és Zipf törvénye között az a különbség, hogy az exponenciális csökkenés esetén a farok vége biztonságosan figyelmen kívül hagyható, de Zipf törvénye esetén a farok teljes súlya jelentős, és nem hagyható figyelmen kívül.

Azért, hogy képesek legyünk hatékonyan dolgozni ebben az aszimmetrikus világban, mindkét típusú webhelyet képesnek kell lennünk felépíteni. A népszerűtlen webhelyeket könnyű kezelni. DNS használatával igazából sok különböző webhely mutathat az interneten belül ugyanarra a számítógépre, ami az összes ilyen webhelyet működteti. Másrészt, a népszerű webhelyek kezelése nehéz. Nem létezik olyan egyedülálló számítógép, ami akár távolról is kellően nagy teljesítményű lenne. Egyetlen számítógép használata esetén, ha az elromlik, több millió felhasználó számára teheti elérhetetlenné a webhelyet. Ezeknek a webhelyeknek a kezeléséhez tartalomelosztó rendszereket kell építenünk. Legközelebb ezek kérdéseivel foglalkozunk.

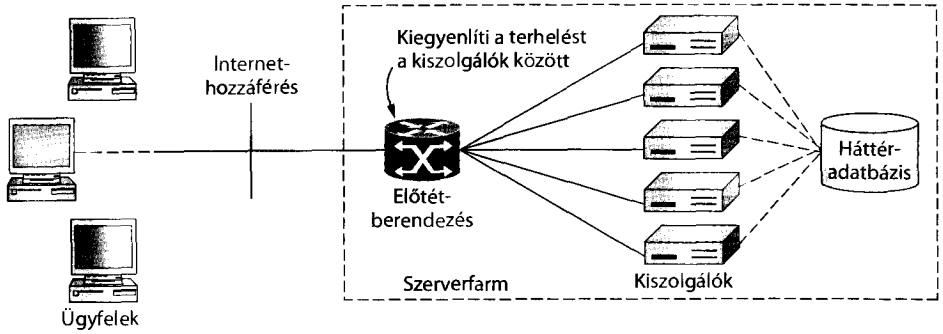
7.5.2. Szerverfarmok és webhelyettesek

Ahogyan azt egészen eddig láttuk, a web megvalósításaiban egyetlen kiszolgálógép volt, ami több ügyfélgéppel beszélgetett. Olyan nagy webhelyek létrehozásához, amelyek jó teljesítményt nyújtanak, felgyorsíthatjuk a műveletvégzést a kiszolgáló oldalán vagy az ügyfél oldalán. A kiszolgálóoldalon nagyobb teljesítményű webszerverek építhetők egy szerverfarm segítségével, amelyben egy számítógépfürt úgy működik, mintha egyetlen kiszolgáló lenne. Az ügyféloldalon jobb teljesítmény érhető el jobb gyorstárazási technikákkal. Különösen a helyettes gyorstárak (proxy cache) biztosítanak egy nagy, megosztott gyorstárat egy ügyfélcsoport számára.

Egymás után mindegyik megoldást ismertetni fogjuk. Jegyezzük meg azonban, hogy egyik megoldás sem elegendő a legnagyobb webhelyek létrehozásához. Azoknak a népszerű webhelyeknek van szükségük a következő szakaszban ismertetendő módszerekre, amelyek sok, különböző helyeken lévő számítógépet fognak össze.

Szerverfarmok

Nem számít, hogy egy számítógépnek mekkora a sávszélessége, mert csak annyi webkérést szolgálhat ki, amennyi még nem jelent túl nagy terhelést. Ebben az esetben az a



7.65. **ábra.** *Egy szerverfarm*

megoldás, hogy egynél több számítógépet kell használni egy webszerver létrehozásához. Ez vezet a 7.65. ábrán látható **szerverfarm** (server farm) modellhez.

Ebben a látszólag egyszerű modellben az jelenti a nehézséget, hogy a szerverfarmot képező számítógépek halmazának az ügyfelek számára egyetlen logikai webhelynek kell látszania. Ha nem látszik annak, akkor csak párhuzamosan működő, különböző webhelyeket hoztunk létre.

Számos lehetséges megoldás létezik arra, hogy a kiszolgálók halmazát egyetlen webhelynek látszóvá tegyük. Mindegyik megoldás feltételezi, hogy bármelyik kiszolgáló bármelyik ügyféltől érkező kérést képes kezelni. Ennek érdekében minden kiszolgálónak rendelkeznie kell a webhely másolati példányával. Ezért a kiszolgálókat úgy ábrázoltuk, hogy azokat szaggatott vonal köti össze a közös háttér adatbázissal.

Az egyik megoldáshoz a DNS-t kell használni a kéréseknek a szerverfarmban lévő kiszolgálók közötti szétterítésére. Amikor egy URL-re vonatkozó webkérést hajtanak végre, a DNS-kiszolgáló a webszerverek IP-címeinek egy körbe forgó listáját küldi vissza. Minden ügyfél megpróbálkozik egy IP-címmel, jellemzően a lista első helyén szereplővel. A hatás az, hogy a különböző ügyfelek különböző kiszolgálókkal veszik fel a kapcsolatot ugyanannak a webhelynek az elérése érdekében pontosan úgy, ahogy azt elterveztük. A DNS-módszer a CDN-ek központi eleme, és később még újra előkerül ebben a szakaszban.

A másik megoldás egy **előtét-berendezésen** (**front end**) alapul, ami szétszórja a beérkező kéréseket a szerverfarm készletében lévő kiszolgálók között. Ez még akkor is megtörténik, ha az ügyfél egyetlen segítségével lép érintkezésbe a szerverfarmmal. Az előtét-berendezés általában egy adatkapcsolati rétegbeli kapcsoló vagy egy IP-útválasztó, azaz egy eszköz, amely kereteket vagy csomagokat kezel. Minden megoldás, ami az előtét-berendezésre (vagy a kiszolgálókra) épül, a hálózati, szállítási vagy alkalmazási réteg fejléceibe kukkant, és nem szabványos módon használja azokat. Egy webkérés és egy webválasz TCP-összeköttetésként kerül továbbításra. A helyes működés érdekében az előtét-berendezésnek a webkérés minden csomagját ugyanannak a kiszolgálónak kell kiosztania.

Az előtét-berendezés egy egyszerű megvalósításában minden beérkező kérést üzenetszórással minden kiszolgálóhoz eljuttatnak. Minden egyes kiszolgáló a kéréseknek csak egy töredékére válaszol, egy előzetes megállapodás alapján. Például 16 kiszolgáló meg-

nézheti a forrás IP-címet, és csak akkor válaszolnak a kérésre, ha a forrás IP-cím utolsó 4 bitje egyezik a beállított szelektorukkal. A többi csomagot eldobják. Noha ez a bejövő sávszélességre nézve pazarló, a válaszok gyakran sokkal hosszabbak, mint a kérés, tehát közel sem olyan hatástalan, mint amilyennek látszik.

Egy általánosabb kivitel esetén az előtét-berendezés megvizsgálhatja a csomagok IP-, TCP- és HTTP-fejléceit, és tetszőleges módon hozzárendelheti azokat egy kiszolgálóhoz. A leképezést **terhelés-kiegyenlítő** (**load balancing**) házirendnek nevezik, mert célja a kiszolgálók közötti munkaterhelés kiegyenlítése. A házirend lehet egyszerű vagy összetett. Egy egyszerű házirend a kiszolgálókat egymás után sorban, vagy körbeforgó módon használhatja. Ennél a megközelítésnél az előtét-berendezésnek minden egyes kéréshez tartozó leképezésre emlékeznie kell, hogy az újabb csomagok, amelyek ugyanannak a kérésnek a részét képezik, ugyanahhoz a kiszolgálóhoz kerüljenek. Annak érdekében, hogy ezt a webhelyet megbízhatóbbá is tegyék, mint egy egyedüli kiszolgáló, az előtét-berendezésnek észre kell vennie, amikor a kiszolgálók meghibásodtak, és le kell állítani a kéréseknek a hozzájuk történő küldését.

A NAT-hoz hasonlóan, ez az általános konstrukció veszélyes, vagy legalábbis törékeny abban az értelemben, hogy az imént létrehoztunk egy eszközt, ami megsérti a rétegszerkezetű protokollok legalapvetőbb elvét, amely szerint vezérlési célokra minden rétegnek a saját fejlécét kell használnia, valamint nem lenne szabad megvizsgálnia és felhasználnia az adatmezőből származó információt semmilyen célra. Az emberek azonban terveznek ilyen rendszereket, és amikor ezek a jövőben tönkremennek a magasabb rétegekben végzett változtatások miatt, hajlamosak meglepődni. Az előtét-berendezés ebben az esetben egy kapcsoló vagy útválasztó, de a szállítási, vagy magasabb rétegbeli információ alapján cselekedhet. Az ilyen dobozt ún. **közbeépített doboznak** (**middlebox**) nevezik, mert beépíti magát a hálózati útvonal közepébe, ahol a protokoll veremnek megfelelően nincs semmi keresnivalója. Ebben az esetben az a legjobb, ha az előtét-berendezést a szerverfarm belső részének tekintjük, ami fel egészen az alkalmazási rétegig az összes réteget lezárja (és ezért azoknak a rétegeknek az összes fejrész információját használhatja).

Ennek ellenére, mint a NAT esetében is, ez a felépítés hasznos a gyakorlatban. A TCP-fejlécek megtekintésének az az oka, hogy jobban el lehet végezni a terhelés kiegyenlítés munkáját, mint csak egyedül az IP-információ alapján. Például egyetlen IP-cím képviselhet egy egész vállalatot, és sok kérést létesíthet. Csak a TCP vagy magasabb rétegbeli információ megtekintésével rendelhetőek ezek a kérések a különböző kiszolgálókhoz.

A HTTP-fejrészek megvizsgálásának oka némileg különböző. Sok webes tevékenység hozzáfér az adatbázisokhoz és frissíti azokat, mint például amikor egy vásárló utánanéz a legutóbbi vásárlásának. A kérést teljesítő kiszolgálónak le kell kérdeznie a háttér-adatbázist. Érdemes az ugyanattól a felhasználótól érkező későbbi kéréseket ugyanahhoz a kiszolgálóhoz irányítani, mert annak a kiszolgálónak a gyorstárában már van információ a felhasználóról. A legegyszerűbb mód annak előidézésére, hogy ez történjen, a websütik felhasználása (vagy más információ a felhasználó megkülönböztetéséhez) és a HTTP-fejrészek megvizsgálása a sütik megtalálása végett.

Utolsó megjegyzésként, bár úgy írtuk le ezt a felépítést, mint ami webhelyekhez használatos, szerverfarmot azonban másfajta kiszolgálókhoz is lehet építeni. Erre példa az UDP felett a médiát folyamszerűen továbbító szerverek. Az egyetlen szükséges változtatás, hogy az előtét-berendezésnek képesnek kell lennie az ezeknek a kéréseknek a következtében fellépő terhelés kiegyenlítésére (a kéréseknek más protokoll fejrész mezői lesznek, mint a webkéréseknek).

Webhelyettesek

A webkéréseket és webválaszokat HTTP segítségével küldik el. A 7.3. szakaszban ismertettük, hogy a böngészők hogyan tudják gyorstárazni a válaszokat és újra felhasználni azokat a jövőbeli kérések megválaszolására. A böngészők különféle fejrészmezőket és szabályokat használnak annak megállapításához, hogy egy weboldalnak a gyorstárban lévő másolata még mindig friss-e. Itt nem fogjuk megismételni azt az anyagot.

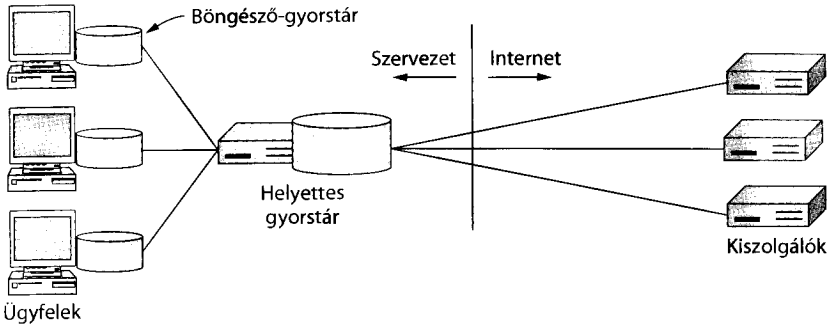
A gyorstárazás a válaszidő rövidítésével és a hálózati terhelés csökkentésével növeli a teljesítőképességet. Ha a böngésző képes önállóan megállapítani, hogy a gyorstárban lévő oldal friss, az oldal azonnal lekérhető a gyorstárból, a legkisebb hálózati forgalom nélkül. Még akkor is, ha a böngészőnek a kiszolgáló megerősítését kell kérnie, hogy az oldal még mindig friss, a válaszidő lerövidül és a hálózati forgalom csökken, különösen nagy oldalak esetén, mivel csak egy kis üzenetet kell elküldeni.

A legjobb azonban, amit egy böngésző tehet, hogy a gyorstárban tart minden weboldalt, amelyet a felhasználó korábban meglátogatott. A népszerűségről szóló értekezésünkből felidézheti, hogy létezik néhány népszerű oldal, amit sok ember rendszeresen látogat, és sok-sok népszerűtlen oldal is. A gyakorlatban ez korlátozza a böngésző gyorstárazásának hatékonyságát, mert sok olyan oldal van, amit az adott felhasználó csak egyszer látogatott meg. Ezeket az oldalakat mindig le kell kérni a kiszolgálóról.

Az egyik stratégia a gyorstárazás hatékonyabbá tételére a gyorstár több felhasználó közötti megosztása. Ily módon az egyik felhasználó számára lekért oldal visszaadható egy másik felhasználónak, amikor az ugyanazt kéri le. Böngészőoldali gyorstárazás nélkül mindkét felhasználónak le kellene kérnie ugyanazt az oldalt a kiszolgálóról. Természetesen ezt a megosztást nem lehet elvégezni titkosított forgalom esetén, olyan oldalaknál, amelyek hitelesítést igényelnek, és a nem gyorstárazható oldalaknál (például aktuális részvényárfolyamok), amelyeket programok adnak vissza. Különösen az egyre gyakoribb, programok által készített dinamikus oldalak esetén nem hatékony a gyorstárazás. Ennek ellenére sok olyan weboldal van, ami számos felhasználó számára látható és ugyanúgy néz ki, attól függetlenül, hogy melyik felhasználó kérte le (például képek).

A **webhelyettest** (**Web proxy**) arra használják, hogy megossza a gyorstárat a felhasználók között. A helyettes egy ügynök, ami valaki más, például a felhasználó érdekében jár el. Sokféle helyettes létezik. Például az ARP-helyettes ARP-kérésekre válaszol egy olyan felhasználó helyett, aki máshol van (és nem tud válaszolni magának). Egy webhelyettes webkéréseket hajt végre a felhasználóinak a nevében. Rendszerint a webválaszok gyorstárazását végzi, és mivel a gyorstárat megosztja a felhasználók között, emiatt a gyorstára lényegesen nagyobb, mint a böngészőé.

Amikor helyettest használnak, a tipikus kiépítés egy szervezet számára az, hogy egyetlen webhelyettest üzemeltetnek az összes felhasználójuk számára. A szervezet lehet egy vállalat vagy egy internetszolgáltató (ISP). Mindketten előnyre tehetnek szert felhasználóik webkéréseinek felgyorsításával és sávszélesség igényük csökkentésével. Míg az



7.66. ábra. *Egy helyettes gyorstár a webböngészők és webszerverek között*

átalánydíj, a használattól független díjszabás általános a végfelhasználóknál, a legtöbb vállalat és internetszolgáltató az általuk használt sávszélességnek megfelelően fizet.

Ezt a felépítést mutatja a 7.66. ábra. A helyettes használatához minden egyes böngészőt úgy állítottak be, hogy a weboldal lekéréseket a helyetteshez intézze az oldal valódi kiszolgálója helyett. Ha a helyettesnek megvan az oldal, akkor azonnal visszaküldi az oldalt. Ha nincs, akkor lekéri az oldalt a kiszolgálótól, hozzáadja a gyorstárhoz későbbi felhasználás végett, és elküldi az ügyfélnek, aki kérte.

Az ügyfelek azonkívül, hogy a webkéréseket a helyettesnek küldik a valódi kiszolgáló helyett, saját gyorstárazást is végeznek böngészőjük gyorstára felhasználásával. A helyettest csak az után kérdezik meg, hogy a böngésző megpróbálta saját gyorstárából kielégíteni a kérést. Azaz a helyettes a gyorstárazás második szintjét biztosítja.

További helyettesek is hozzáadhatók a gyorstárazás további szintjeinek biztosítása érdekében. Minden egyes helyettes (vagy böngésző) a **felmenő (upstream) helyettes** segítségével bonyolítja a kéréseit. Minden egyes felmenő helyettes gyorstárazást végez a **lemenő (downstream) helyettesek** (vagy böngészők) számára. Tehát a vállalati böngészők használhatják a vállalati helyettest, ami az internetszolgáltató helyettesét használja, ami közvetlenül kapcsolódik a webszerverhez. Az egyszintű helyettes gyorstárazás azonban, amit a 7.66. ábrán mutattunk be, a gyakorlatban gyakran elegendő a potenciális előnyök nagy részének kihasználásához. A problémát ismét a népszerűség hosszú farka jelenti. A webes forgalomról készült tanulmányok megmutatták, hogy a megosztott gyorstárazás addig különösen előnyös, amíg a felhasználók száma el nem éri körülbelül egy kisvállalat méretét (mondjuk, 100 főt). Amint az emberek száma tovább nő, a gyorstár megosztásának haszna marginálissá válik a népszerűtlen kérések miatt, melyek tárolóhely hiányában nem tarthatók a gyorstárban [Wolman és mások, 1999].

A webhelyettesek további előnyöket is nyújtanak, amelyek gyakran fontos tényezők a telepítésükről történő döntésben. Az egyik előny a tartalom szűrése. A rendszergazda beállíthatja úgy a helyettest, hogy az bizonyos webhelyeket helyezzen feketelistára, vagy másképpen szűrje meg az általa indított kéréseket. Például sok rendszergazda rosszallóan tekint a munkaidőben YouTube videót (vagy ami még rosszabb, pornót) néző alkalmazottakra, és a szűrőiket ennek megfelelően állítják be. A helyettesek másik haszna a magánélet védelme vagy az anonimitás, amikor a helyettes elrejti a felhasználó személyazonosságát a kiszolgáló elől.

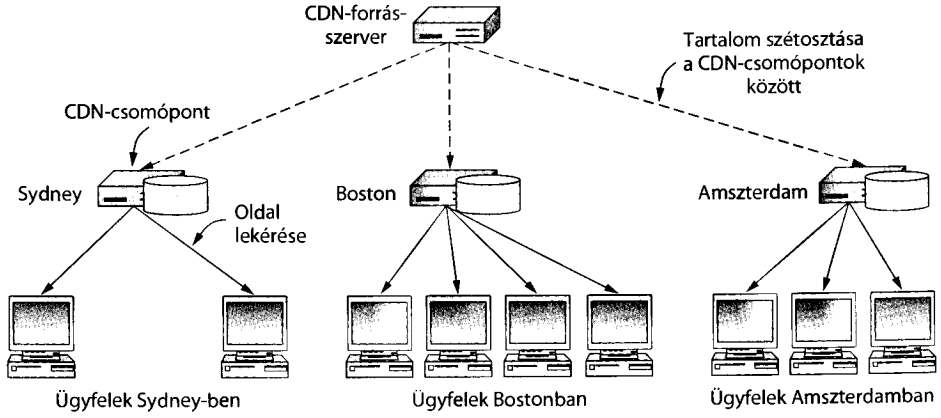
7.5.3. Tartalomszállító hálózatok

A szerverfarmok és a webhelyettesek segítik a nagy webhelyek kiépítését és a web teljesítőképességének növelését, de nem elegendők az igazán népszerű webhelyek számára, amelyeknek globális szinten kell tartalmat szolgáltatniuk. Ezekhez a webhelyekhez más megoldásra van szükség.

A **CDN-ek (Content Delivery Networks – tartalomszállító hálózatok)**[8] a hagyományos web gyorstárazás ötletét a feje tetejére állították. Ahelyett, hogy az ügyfelek a kért oldal másolati példányát egy közeli gyorstárban keresnék, maga a szolgáltató helyezi el az oldal másolatait a különböző helyeken lévő csomópontok halmazában, és arra utasítja az ügyfelet, hogy egy közeli csomópontot használjon kiszolgálóként.

A 7.67. ábra egy példát mutat arra az útra, amelyet az adatok követnek, amikor azokat CDN osztja szét. Ez egy fa. A CDN-ben lévő forrás kiszolgáló szétosztja a tartalom másolatait a CDN-ben lévő többi csomópontnak, amelyek ebben a példában Sydneyben, Bostonban és Amszterdamban találhatók. Ezt ábrázolják a szaggatott vonalak. Az ügyfelek ezután az oldalakat a CDN-ben legközelebb lévő csomóponttól kérik le. Ezt a folytonos vonalak jelzik. Így az összes Sydney-ben lévő ügyfél az oldalnak a Sydney-ben tárolt másolatát kéri le, és nem a forráskiszolgálótól kérik el az oldalt, ami talán Európában található.

Egy fastruktúra használatának három előnye van. Az első, hogy a tartalom elosztása több CDN-csomópont használatával annyi ügyfélre terjeszthető ki, amennyire csak szükséges, és a fában több szint is létrehozható, amikor a CDN-csomópontok közötti elosztás válik a szűk keresztmetszetté. Nem lényeges, hogy hány ügyfél van, a fastruktúra hatékony. A forráskiszolgáló nincs túlterhelve, mert a sok ügyféllel a CDN-csomópontok fájának segítségével beszélget; nem magának kell megválaszolnia egy oldalra vonatkozó minden egyes kérést. A második, hogy minden egyes ügyfél jó teljesítményű kiszolgálásban részesül azáltal, hogy az oldalakat egy közeli kiszolgálóról kéri le, és nem



7.67. ábra. *A CDN-tartalomelosztás fastruktúrája*

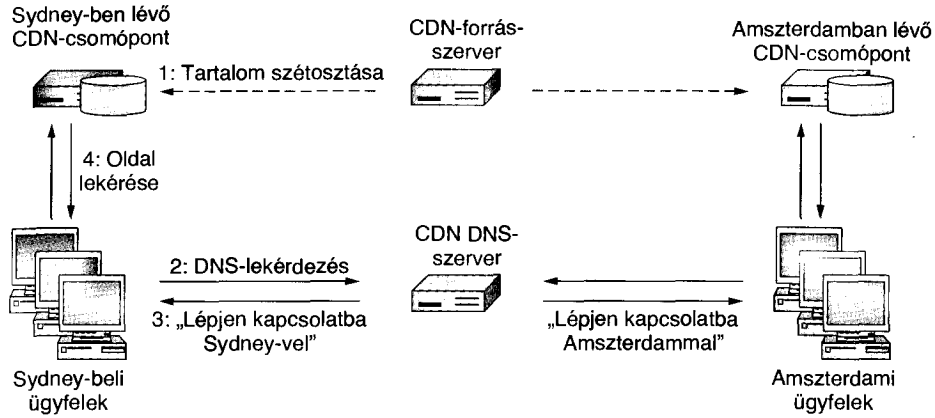
8 Ugyanaz, mint a tartalomelosztó hálózat. *(A lektor megjegyzése)*

egy távoliról. Ennek az az oka, hogy az összeköttetés létesítésének körbefordulási ideje rövidebb, a TCP a lassú indulást követően sokkal hamarabb felgyorsul a rövidebb körbefordulási idő miatt, és a rövidebb hálózati út kisebb valószínűséggel keresztez torlódásos területeket az interneten. Végül, a hálózatra helyezett teljes terhelést is a minimumon tartja. Ha a CDN-csomópontokat jól helyezik el, akkor egy adott oldalra irányuló forgalomnak a hálózat minden egyes részén csak egyszer kell áthaladnia. Ez fontos, mert végül valakinek fizetnie kell a hálózati sávszélességért.

Az elosztási fa használatának ötlete egyszerű. Ami kevésbé egyszerű, az az, hogyan kell az ügyfeleket szervezni ennek a fának a használatához. Például úgy tűnhet, hogy a helyettes kiszolgálók megoldást nyújtanak. A 7.67. ábrára nézve, ha minden ügyfelet a sydney-i, bostoni vagy amszterdami CDN-csomópont mint gyorstárazást végző webhelyettes használatára állítanának be, az elosztás a fának megfelelően történne. Ez a stratégia azonban a gyakorlatban hamar elbukna, három okból is. Az első ok az, hogy a hálózat egy adott részében lévő ügyfelek valószínűleg különböző szervezetekhez tartoznak, ezért feltehetőleg különböző webhelyetteseket is használnak. Emlékezzen rá vissza, hogy a gyorstárakat általában nem osztják meg a szervezetek között a nagyszámú ügyfél gyorstárazásából adódó korlátozott előnyök miatt, és biztonsági okokból sem. Másodszor, több CDN is létezhet, de minden ügyfél csak egy helyettes gyorstárat használ. Melyik CDN-t kellene az ügyfélnek helyettesként használnia? Végül, talán az összes között a leggyakorlatiasabb probléma az, hogy a webhelyetteseket az ügyfelek állítják be. Vagy beállítják azokat a CDN által nyújtott tartalomelosztás előnyeinek kiaknázására vagy nem, és a CDN ezzel kapcsolatban csak keveset tehet.

Egy másik egyszerű módja az egyszintű elosztási fa megvalósításának a **tükrözés** (**mirroring**) használata. Ennél a megközelítésnél a forráskiszolgáló ugyanúgy megismétli a tartalmat a CDN-csomópontokon, mint korábban. A különböző hálózati tartományokban lévő CDN-csomópontokat **tükröknek** (**mirrors**) nevezik. A forráskiszolgálón lévő weboldalak kifejezett hivatkozásokat tartalmaznak a különböző tükrökre, amelyek általában megmondják a felhasználóknak a tükrök helyét. Ez a felépítés lehetővé teszi a felhasználó számára, hogy kézzel válasszon ki egy közeli tükröt a tartalom letöltéséhez. A tükrözést általában úgy alkalmazzák, hogy elhelyeznek egy nagy szoftvercsomagot például, az Egyesült Államok keleti és nyugati partján, Ázsiában és Európában található tükrökön. A tükrözött helyek általában teljesen statikusak, és a helyek választéka hónapokon vagy éveken át állandó marad. Ezek kipróbált és bevizsgált módszerek. Az elosztás azonban a felhasználótól függ, mert a tükrök valóban különböző webhelyek, még akkor is, ha a hivatkozások összekötik őket.

A harmadik megoldás, amelyik átlép az előző két megoldás nehézségein, DNS-t használ és **DNS-átirányításnak** (**DNS redirection**) nevezik. Tételezzük fel, hogy az ügyfél le akarja kérni a *http://www.cdn.com/page.html* URL-lel azonosított oldalt. Az oldal lekéréséhez a böngésző a DNS-t fogja használni a *www.cdn.com*-nak IP-címre történő feloldásához. Ez a DNS-keresés a szokásos módon történik. A DNS-protokoll használata során a böngésző megtanulja a *cdn.com*-hoz tartozó névkiszolgáló IP-címét, azután kapcsolatba lép a névkiszolgálóval, hogy megkérje a *www.cdn.com* feloldására. Most jön az igazi csel. A névkiszolgálót a CDN működteti. Ahelyett, hogy minden egyes kérésre ugyanazt az IP-címet adná vissza, megnézi a kérést küldő ügyfél IP-címét, és különböző válaszokat ad vissza. A válasz annak a CDN-csomópontnak az IP-címe lesz, amelyik a



7.68. ábra. *Az ügyfelek közeli CDN-csomópontokhoz irányítása DNS használatával*

legközelebb található az ügyfélhez. Tehát, ha egy ügyfél Sydney-ben megkéri a CDN-névkiszolgálót, hogy oldja fel a *www.cdn.com*-ot, a névkiszolgáló a Sydney-ben lévő CDN-csomópont IP-címét adja vissza, de ha egy amszterdami ügyfél kéri ugyanezt, a névkiszolgáló az amszterdami CDN-csomópont IP-címét küldi vissza.

Ez a stratégia a DNS szemantikájának megfelelően teljesen legális. Korábban már láttuk, hogy a névkiszolgálók az IP-címek változó listáját küldhetik vissza. A névfeloldás után a Sydney-ben lévő ügyfél az oldalt közvetlenül a Sydney-ben található CDN-csomóponttól szerzi meg. Az ugyanazon a „kiszolgálón" lévő további oldalak is közvetlenül a Sydney-ben lévő CDN-csomópontról szerezhetők meg a DNS-gyorstárazás miatt. A lépések teljes sorozatát a 7.68. ábra mutatja.

A fenti folyamatban van egy összetett kérdés, hogy mit jelent a legközelebbi CDN-csomópont megtalálása, és hogyan kezdjünk hozzá. A legközelebbi fogalom definiálásakor nem a földrajz az igazán lényeges. Legalább két olyan tényező van, amit figyelembe kell venni egy ügyfélnek egy CDN-csomóponthoz történő hozzárendelésénél. Az egyik tényező a hálózati távolság. Az ügyfélnek a CDN-csomópontig egy rövid és nagy kapacitású hálózati útvonallal kell rendelkeznie. Ez a helyzet gyors letöltéseket fog eredményezni. A CDN-ek egy általuk korábban kiszámított leképezési táblázatot használnak az ügyfél IP-címe és annak hálózati helye közötti fordításhoz. A kiválasztott csomópont lehet, hogy a légvonalban legrövidebb távolságra levő lesz, de lehet, hogy nem. Ami fontos, az a hálózati út hosszúságának és az azon lévő kapacitáskorlátozások valamilyen kombinációja. A második tényező az a terhelés, ami már most is a CDN-csomópontra nehezedik. Ha a CDN-csomópontok túlterheltek, akkor lassan válaszolnak, pont úgy, mint az a túlterhelt webszerver, amelyet leginkább szeretnénk elkerülni. Tehát szükség lehet a CDN-csomópontok között a terhelés kiegyenlítésére, néhány ügyfél olyan csomóponthoz rendelésével, amelyek egy kicsit távolabb vannak, de kevésbé leterheltek.

A DNS-nek a tartalomelosztáshoz történő használatát elsőként az Akamai cég alkalmazta 1998-tól kezdődően, amikor a világháló a korai növekedés terhe alatt nyögött. Az Akamai volt az első nagy CDN, és iparági vezetővé vált. Az üzleti megoldásuk ösztönző szerkezete valószínűleg még annál az ötletnél is ügyesebb volt, hogy a DNS-t használják

az ügyfeleknek a közeli csomóponttal való összekapcsolására. A vállalatok fizetnek az Akamainak azért, hogy az úgy juttassa el a tartalmukat az ügyfelekhez, hogy a webhelyeik gyorsan reagáljanak, amit a felhasználók szeretnek használni. A CDN-csompontokat a hálózat jó összeköttetésű helyein kell elhelyezni, ami kezdetben azt jelentette, hogy az internetszolgáltatók hálózatán belül. Az internetszolgáltatók számára előnyös volt egy CDN-csomópont elhelyezése a hálózatukon belül, tudniillik a CDN-csomópont, akárcsak a helyettes gyorstárak, csökkenti azt a felfelé irányuló hálózati sávszélességet, amire szükségük van (és amiért fizetni kell). Ráadásul, a CDN-csomópont az internetszolgáltató ügyfelei számára a válaszolási képességet javítja, ami a szolgáltatót kedvezően tünteti fel a szemükben, versenyelőnyt biztosítva azokkal a szolgáltatókkal szemben, amelyeknek nincsen CDN-csomópontjuk. Ezek az előnyök (amelyek nem kerültek pénzbe az internetszolgáltatónak) azt eredményezték, hogy a szolgáltatók ész nélkül telepítik a CDN-csomópontokat. Tehát a tartalomszolgáltatónak, az internetszolgáltatónak és az ügyfeleknek mind hasznára válik, a CDN pedig pénzt keres. 1998 óta más vállalatok is beszálltak az üzletbe, így ez most már egy versenyképes iparág több szolgáltatóval.

A leírtakból az következik, hogy a legtöbb vállalat nem építi ki a saját CDN-jét. Ehelyett a tartalmuk tényleges továbbításához egy olyan CDN-szolgáltató szolgáltatásait veszik igénybe, mint az Akamai. Annak érdekében, hogy más vállalatok is használhassák a CDN szolgáltatásait, a képünkhöz hozzá kell adnunk még egy utolsó lépést.

Miután aláírták a szerződést a CDN-nel a tartalomnak egy webhely tulajdonosa nevében történő elosztásáról, a tulajdonos átadja a tartalmat a CDN-nek. Ezt a tartalmat

```
<html>
<head> <title> Bundás Videó </title> </head>
<body>
<h1> Bundás Videó terméklista </h1>
<p> Kattintson az alábbi ingyenes termékmintákra! </p>

<a href="koalak.mpg"> Mai koalák </a> <br>
<a href="kenguruk.mpg"> Vicces kenguruk </a> <br>
<a href="erszenyesek.mpg"> Kedves erszényes medvék </a> <br>
</body>
</html>
```

(a)

```
<html>
<head> <title> Bundás Videó </title> </head>
<body>
<h1> Bundás Videó terméklista </h1>
<p> Kattintson az alábbi ingyenes termékmintákra! </p>

<a href="http://www.cdn.com/bundasvideo/koalak.mpg"> Mai koalák </a> <br>
<a href=" http://www.cdn.com/bundasvideo/kenguruk.mpg"> Vicces kenguruk </a> <br>
<a href=" http://www.cdn.com/bundasvideo/erszenyesek.mpg"> Kedves erszényes medvék
</a> <br>
</body>
</html>
```

(b)

7.69. ábra. *(a) Az eredeti weboldal. (b) Ugyanaz az oldal a CDN-re hivatkozás után*

eljuttatják a CDN-csomópontokra. Ezenkívül a tulajdonos újraírja valamennyi weboldalát, amely a tartalomra hivatkozik. Ahelyett, hogy a saját webhelyükön lévő tartalomra hivatkoznának, az oldalak a CDN-en elérhető tartalomra mutatnak. A séma működésének példájaként tekintse meg a Bundás Videó weblapjának forráskódját a 7.69.(a) ábrán. Az előfeldolgozás után ez a 7.69.(b) ábra alakját ölti, és felkerül a Bundás Videó kiszolgálójára a *www.bundasvideo.com/index.html* címen.

Amikor a felhasználó begépeli böngészőjébe a *www.bundasvideo.com* URL-t, a DNS megadja a Bundás Videó weboldalának IP-címét, hogy a központi (HTML) oldalt a megszokott módon le lehessen tölteni. Amikor a felhasználó rákattint valamelyik hiperhivatkozásra, a böngésző kikeresteti a DNS-sel a *www.cdn.com* címét. A keresés során kapcsolatba lép a CDN DNS-kiszolgálójával, ami visszaadja a közelben lévő CDN-csomópont IP-címét. A böngésző ezután egy szokásos, például a */bundasvideo/koalak.mpg*-re vonatkozó HTTP-kérést küld a CDN-csomópontnak. Az URL meghatározza a visszaküldendő oldalt. Az út *bundasvideo*-val kezdődik, hogy a CDN-csomópont képes legyen megkülönböztetni az általa kiszolgált különböző cégekre vonatkozó kéréseket. Végül visszaadja a mozgóképet és a felhasználó megnézi az aranyos bundás állatokat.

A felosztás hátterében álló stratégia, miszerint a tartalmat a CDN-en helyezik el, a kezdőoldalakat pedig a tartalom tulajdonosánál, abból áll, hogy az irányítást a tartalom tulajdonosának adja, miközben a CDN-re bízza a nagy mennyiségű adat mozgatását. A legtöbb kezdőoldal kicsi, csak HTML-szövegből áll. Ezek az oldalak gyakran hivatkoznak nagy állományokra, például videókra és képekre. A CDN pontosan ezeket a nagy állományokat szolgáltatja, annak ellenére, hogy a CDN használata a felhasználó számára teljesen észrevehetetlen. Az oldal ugyanúgy néz ki, de gyorsabban megjelenik.

A webhelyek számára egy másik előnye is van a megosztott CDN használatának. Egy webhely iránti jövőbeli igényt nehéz megjósolni. Az igények gyakran hullámzóak, amit **hirtelen tömegnek** (**flash crowd**) neveznek. Egy ilyen hullám akkor keletkezhet, amikor a legújabb terméket forgalomba hozzák, divatbemutató vagy más esemény alkalmával, vagy ha a vállalat valahogy másképp bekerült a hírekbe. Még egy korábban ismeretlen, nem látogatott holtágat képező webhely is hirtelen az internet középpontjába kerülhet, ha hírértékűvé válik, és népszerű webhelyek hivatkoznak rá. Mivel a legtöbb webhely nincs felkészítve a forgalom erőteljes növekedésére, ezért amikor a forgalom áradata rájuk zúdul, sokan összeomlanak.

Egy hasonló eset volt a következő. A floridai kormányhivatal webhelye rendszerint annak ellenére nem egy forgalmas hely, hogy kikereshetők rajta floridai vállalatokról, közjegyzőkről, kulturális ügyekről, valamint az ottani szavazásról és választásról szóló információk is. Valamilyen furcsa okból kifolyólag 2000. november 7-én (a Bush és Gore között zajló elnökválasztás napja) hirtelen egy csomó embert érdekelt az ezen a webhelyen elhelyezett, választási eredményeket tartalmazó oldal. A hely hirtelen a világ egyik leglátogatottabb webhelyévé vált, és ennek következtében természetesen összeomlott. Ha CDN-t használt volna, valószínűleg túlélte volna a megpróbáltatásokat.

A CDN használatával egy webhely nagyon nagy tartalomkiszolgáló kapacitáshoz fér hozzá. A legnagyobb CDN-ek több tízezer kiszolgálót telepítettek a világ minden táján lévő országokban. Mivel (definíció szerint) egyszerre csak kevés webhelyen tapasztalható hirtelen tömeg, ezek a webhelyek felhasználhatják a CDN kapacitását a terhelés kezelésére, amíg a vihar elvonul. Azaz a CDN gyorsan meg tudja emelni a webhely kiszolgálási kapacitását.

Az előzőekben tárgyaltak az Akamai működésének egyszerűsített leírása. Sokkal több olyan részlet van, ami a gyakorlatban fontos. A példánkban bemutatott CDN-csomópontok általában gépfürtök (clusters). A DNS-átirányítás két szinten történik: az egyiken az ügyfeleket hozzárendelik a hozzávetőleges hálózati helyhez, a másikon szétszórják a terhelést az adott helyen lévő csomópontok között. A megbízhatóság és a teljesítőképesség is fontos. Annak érdekében, hogy egy ügyfél képes legyen a fürt egyik gépéről a másikra váltani, a második szinten a DNS-válaszok rövid élettatartammal (TTL) rendelkeznek, hogy az ügyfél rövid idő után megismételje a címfeloldást. Végül, az olyan statikus objektumok elosztására koncentráltunk, mint a képek és a videók, de a CDN-ek a dinamikus oldalak létrehozását, a folyamszerű médiaközvetítést és mást is támogatnak. A CDN-ekkel kapcsolatos további információért lásd Dilley és mások [2002] művét.

7.5.4. Egyenrangú társak hálózata

Nem mindenki képes egy világszerte 1000 csomópontból álló CDN-t felállítani a tartalmának elosztásához. (A jól fejlett és versenyképes tárhelyiparnak köszönhetően igazából nem nehéz világszerte 1000 virtuális gépet bérelni. A csomópontok beszerzése azonban a CDN felépítésének csak a kezdete.) Szerencsére sokunk számára létezik egy alternatíva, amelyet könnyű használni és óriási mennyiségű tartalom elosztására képes. Ez a P2P- (Peer-to-Peer – egyenrangú társak) hálózat.

A P2P-hálózatok 1999-től kezdve jelentek meg a színtéren. Első széles körű alkalmazásuk tömeges bűncselekmény lett: 50 millió Napster-felhasználó cserélt egymással szerzői jog által védett dalokat a jogtulajdonosok engedélye nélkül, amíg a bíróság nagy viták közepette le nem állította a Napster működését. Ennek ellenére az egyenrangú műszaki megoldásoknak számos érdekes és legális alkalmazása létezik. Más rendszerek fejlesztése folytatódott, amelyek iránt a felhasználók akkora érdeklődést mutattak, hogy a P2P-forgalom gyorsan lekörözte a webes forgalmat. Ma a BitTorrent a legnépszerűbb P2P-protokoll. Olyan széles körben alkalmazzák (engedélyezett és nyilvános) mozgóképek és más tartalmak megosztására, hogy a teljes internetforgalom nagy hányadát teszi ki. Ezt fogjuk megvizsgálni ebben a szakaszban.

A **P2P** (**Peer-to-Peer, egyenrangú társak**) állománymegosztó hálózat alapötlete az, hogy sok számítógép összeáll, és összerakják erőforrásaikat egy tartalomelosztó rendszer létrehozása érdekében. A számítógépek gyakran egyszerű otthoni számítógépek. Nem kell internetes adatközpontban lévő gépeknek lenniük. A számítógépeket társaknak (peer) nevezik, mert mindegyik képes felváltva egy másik társ ügyfeleként működni, lekérni annak tartalmát, és kiszolgálóként tartalmat szolgáltatni más társak számára. A P2P-rendszereket az teszi érdekessé, hogy a CDN-nel ellentétben itt nincs dedikált infrastruktúra. Mindenki részt vesz a tartalomelosztás feladatában, és gyakran nincs központi irányítás.

Sokan izgatottak a P2P műszaki megoldások összessége miatt, mert úgy látják, hogy az erőt ad a kicsiknek. Ennek nemcsak az az oka, hogy egy CDN üzemeltetése nagy vállalatot igényel, miközben egy P2P-hálózathoz bárki csatlakozhat egy számítógéppel. A P2P-hálózatok olyan ijesztően nagy kapacitással rendelkeznek a tartalmak elosztásához, ami összemérhető a legnagyobb webhelyekével.

Képzeljen el egy N átlagos felhasználóból álló P2P-hálózatot, amelyben mindenki 1 Mb/s-os széles sávú kapcsolattal rendelkezik. A P2P-hálózat összesített feltöltési kapacitása, vagy az a sebesség, amivel a felhasználók a tartalmat az internetre küldhetik, N Mb/s. A letöltési kapacitás, vagy a sebesség, amellyel a felhasználók a forgalmat fogadhatják, szintén N Mb/s. Minden felhasználó képes egyszerre fel- és letölteni is, mert mindkét irányban 1 Mb/s sebességű kapcsolattal rendelkeznek.

Nem nyilvánvaló, hogy ennek igaznak kell lennie, de kiderül, hogy a teljes kapacitás eredményesen használható a tartalom elosztására, még egy állomány egyetlen másolati példányának az összes többi felhasználóval történő megosztása esetén is. Hogy lássuk, mindez hogyan lehetséges, képzeljük el, hogy a felhasználókat egy bináris fába szervezték, amelyben minden nem-levél felhasználó két másik felhasználónak tud adatokat küldeni. A fa eljuttatja az állomány egyetlen másolatát az összes többi felhasználóhoz. A lehető legtöbb felhasználó feltöltési sávszélességének állandó használatához (ennélfogva a nagy állomány kis késleltetéssel történő elosztásához) a felhasználók hálózati tevékenységét csővezetékbe kell szerveznünk. Képzeljük el, hogy az állomány 1000 darabra van osztva. Minden felhasználó képes egy új darabot fogadni valahonnan a fa felső részéből, és a korábban fogadott darabot ezzel egyidejűleg lefelé küldi a fában. Ily módon, miután a csővezeték beindult, néhány (a fa mélységével megegyező) darab elküldése után minden nem-levél felhasználó szorgalmasan tölti fel az állományt a többi felhasználóhoz. Mivel a nem-levél felhasználók száma körülbelül $N/2$, ennek a fának a feltöltési sávszélessége $N/2$ Mb/s. Megismételhetjük ezt a trükköt, és létrehozhatunk egy másik fát, ami a levél és nem-levél csomópontok szerepének felcserélésével kihasználja a másik $N/2$ Mb/s feltöltési sávszélességet. Ez a szerkezet együttesen a teljes kapacitást kihasználja.

Ez az érvelés azt jelenti, hogy a P2P-hálózatok önskálázóak. A használható feltöltési kapacitásuk azokkal a letöltési igényekkel párhuzamosan nő, amelyeket a felhasználóik okozhatnak. Bizonyos értelemben mindig „kellően nagyok" anélkül, hogy szükségük lenne bármilyen dedikált infrastruktúrára. Ezzel ellentétben még egy nagy webhely kapacitása is rögzített, és vagy túl nagy, vagy túl kicsi lesz. Vegyünk egy mindössze 100 fürtből álló webhelyet, melyek közül mindegyik 10 Gb/s-ra képes. Kevés felhasználó esetén nem segít ez a roppant kapacitás. A webhely az N felhasználónak nem tud N Mb/s-nál nagyobb sebességgel információt nyújtani, mert a korlát a felhasználóknál van és nem a webhelyen. Több mint egymillió 1 Mb/s-os felhasználó esetén pedig a webhely nem tudja elég gyorsan kiszivattyúzni az adatokat az összes felhasználó folyamatos letöltésének fenntartásához. Ez nagyszámú felhasználónak tűnhet, de a nagy BitTorrent-hálózatok (például Pirate Bay) azt állítják, hogy több mint 10 000 000 felhasználójuk van. Ez a mi példánk esetében több mint 10 terabit/s!

Ezeket a hozzávetőleges számokat egy szemernyi (vagy inkább óriási) fenntartással kellene kezelnie, mert túlságosan leegyszerűsítik a helyzetet. A P2P-hálózatok számára jelentős kihívást jelent a sávszélesség jó kihasználása, amikor a felhasználók minden formában és méretben előfordulnak, valamint különböző feltöltési és letöltési kapacitásokkal rendelkeznek. Ezek a számok azonban jelzik a P2P-ben rejlő óriási potenciált.

Van egy másik oka is annak, hogy a P2P-hálózatok fontosak. A CDN-ek és más, központilag működtetett szolgáltatások a szolgáltatókat olyan helyzetbe hozzák, amelyben számos felhasználó személyes adatainak tárházával rendelkeznek, kezdve a böngészési

preferenciákkal és az emberek online vásárlásainak helyeivel, befejezve az emberek tartózkodási helyével és e-levél címével. Ez az információ felhasználható egy jobb, még inkább személyre szabott szolgáltatás nyújtásához, de az emberek megánéletébe történő betolakodásra is. Az utóbbi történhet szándékosan – mondjuk egy új termékkel kapcsolatban – vagy egy véletlen közzétételnek, veszélyeztetésnek a következtében. A P2P-rendszerek esetén nem létezhet olyan egyedüli szolgáltató, amelyik képes a teljes rendszer ellenőrzésére. Ez nem jelenti azt, hogy a P2P-rendszerek feltétlenül biztosítják a magánélet védelmét, mivel a felhasználók bizonyos mértékig megbíznak egymásban. Ez csak annyit jelent, hogy a magánélet védelmének egy másik formáját biztosítják, mint a központilag irányított rendszerek. Most fedezik fel a P2P-rendszerekben rejlő, az állománymegosztáson túlmutató szolgáltatások (például tárolás, folyamszerű médiaközvetítés) lehetőségét, és majd az idő megmutatja, hogy ez a lehetőség jelentős-e.

A P2P műszaki megoldások összességének a kifejlesztése óta két, egymással kapcsolatban lévő utat követ. A gyakorlatiasabb oldalon állnak azok a rendszerek, amelyeket mindennap használnak. A legismertebb ilyen rendszerek a BitTorrent-protokollon alapulnak. Az inkább elméleti oldalon élénk érdeklődés kíséri a DHT- (Distributed Hash Table – osztott hash-tábla) algoritmusokat, amelyek lehetővé teszik, hogy a P2P-rendszerek teljes egészében jó teljesítményt nyújtsanak, miközben egyáltalán nem támaszkodnak központosított alkotóelemekre. Mindkét megoldást megvizsgáljuk.

BitTorrent

A BitTorrent-protokollt Brahm Cohen fejlesztette ki 2001-ben, hogy a társak csoportja számára lehetővé tegye a gyors és könnyű állománymegosztást. Számos szabadon elérhető ügyfél létezik, amelyek ugyanúgy használják ezt a protokollt, mint ahogyan sok böngésző a HTTP-protokollt használja a webszerverekkel történő kommunikációja során. A protokoll nyílt szabványként elérhető a *www.bittorrent.org* címen.

Egy tipikus P2P-rendszerben, mint amilyet BitTorenttel alakítottak ki, minden felhasználó rendelkezik némi információval, ami számot tarthat a többi felhasználó érdeklődésére. Ez az információ lehet szabad szoftver, zene, mozgóképek, fényképek és így tovább. Három problémát kell megoldani ebben a környezetben a tartalom megosztásához:

1. Hogyan találja meg a társ azokat a további társakat, amelyek rendelkeznek azzal a tartalommal, amit le kíván tölteni?

2. Hogyan többszörözik meg a társak a tartalmat annak érdekében, hogy mindenkinek nagy sebességű letöltést nyújtsanak?

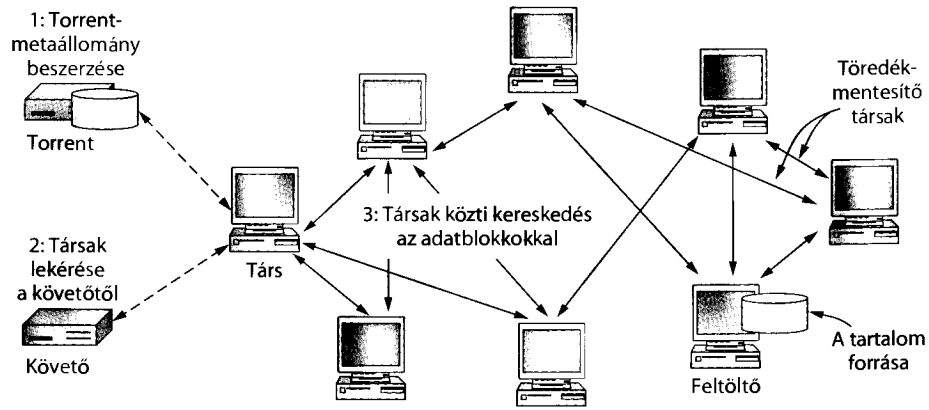
3. Hogyan ösztönzik egymást a társak a tartalom másokhoz történő feltöltésére és a tartalom maguk számára történő letöltésére?

Az első probléma azért létezik, mert – legalábbis kezdetben – nem minden társ rendelkezik az összes tartalommal. A BitTorrent azt a megközelítést alkalmazza, hogy minden tartalomszolgáltató létrehoz egy tartalomleírást, amit **torrentnek** (áradat) neveznek. A torrent sokkal kisebb, mint a tartalom, és ezt egy társ arra használja, hogy a többi társtól letöltött adatok integritását ellenőrizze vele. A többi felhasználónak, akik le akarják tölteni a tartalmat, először be kell szerezniük a torrentet, mondjuk úgy, hogy megtalálják azt egy, a tartalmat reklámozó weboldalon.

A torrent csak egy különleges formátumú állomány, ami kétfajta kulcsfontosságú információt tartalmaz. Az egyik a **követő** (**tracker**) neve, ami egy olyan kiszolgáló, ami elvezeti a társakat a torrent tartalmához. A másik fajta információ a tartalmat alkotó egyforma méretű darabok, vagy **adatblokkok, töredékek** (**chunk**s) listája. A különböző torrentekhez különböző méretű, általában 64 KB és 512 KB közötti adatblokkok használhatók. A torrentállomány minden egyes adatblokk nevét tartalmazza, amit az adatblokk SHA-1 hash-függvénnyel képzett 160 bites hash-értékével adnak meg. Az olyan kriptográfiai hash-függvényeket, mint például az SHA-1, a 8. fejezetben fogjuk tárgyalni. Egyelőre egy hosszabb és biztonságosabb ellenőrző összegként gondolhatunk a hash-értékre. Az adatblokkok méretét és a hash-értékeket figyelembevéve, a torrentállomány mérete legalább három nagyságrenddel kisebb, mint a tartalomé, tehát ez gyorsan továbbítható.

A torrentben leírt tartalom letöltéséhez egy társ először kapcsolatba lép a torrent követőjével. A **követő** (**tracker**) egy olyan kiszolgáló, ami egy listát tart fenn a tartalmat aktívan le- és feltöltő összes többi társról. A társaknak ezt a halmazát **bolynak** (**swarm**) hívják. A boly tagjai rendszeresen kapcsolatba lépnek a követővel, hogy bejelentsék, még mindig aktívak, valamint a boly elhagyásakor is kapcsolatba lépnek vele. Amikor egy új társ a bolyhoz történő csatlakozás miatt kapcsolatba lép a követővel, a követő beszámol neki a bolyban lévő többi társról. A torrent megszerzése és a követővel történő kapcsolatfelvétel az első két lépés a tartalom letöltése érdekében, ahogyan azt a 7.70. ábra mutatja.

A második probléma az, hogy hogyan osszuk meg úgy a tartalmat, hogy az lehetővé tegye az azonnali letöltéseket. Amikor a boly először létrejön, néhány társnak a tartalmat felépítő összes adatblokkal rendelkeznie kell. Ezeket a társakat hívják **feltöltőknek vagy magosztóknak** (**seeders**). A bolyhoz csatlakozó többi társnak nincsenek adatblokkjai; ezek azok a társak, akik letöltik a tartalmat.



7.70. ábra. *BitTorrent*

Amíg egy társ részt vesz egy bolyban, egyszerre tölt le hiányzó adatblokkokat a többi társtól, és tölt fel meglévő adatblokkokat olyan társakhoz, akik igénylik azokat. Ezt a kereskedést a tartalomelosztás utolsó lépéseként mutatja a 7.70. ábra. Idővel a társ több adatblokkot is összegyűjt, amíg le nem tölti a teljes tartalmat. A társ bármikor elhagyhatja a bolyt (és vissza is térhet). A letöltés befejeződését követően egy társ általában egy rövid ideig még a bolyban marad. Az érkező és távozó társak miatt a bolyban a lemorzsolódás aránya elég magas lehet.

Ahhoz, hogy a fenti módszer jól működjön, minden adatblokknak sok társnál kellene rendelkezésre állnia. Ha mindenki ugyanabban a sorrendben kapná meg az adatblokkokat, akkor valószínűleg sok társ következő adatblokkja függene a feltöltőktől. Ez szűk keresztmetszetet eredményezne. Ehelyett a társak kicserélik egymással azoknak az adatblokkoknak a listáját, amelyekkel rendelkeznek. Ezután kiválasztják azokat a ritka adatblokkokat, amelyeket nehéz fellelni a letöltéshez. Az ötlet az, hogy a ritka adatblokkok letöltésével létrejön róluk egy másolat, ami a többi társ számára megkönnyíti az adatblokk megtalálását és letöltését. Ha minden társ így tesz, rövid idő elteltével minden adatblokk széles körben elérhető lesz.

Talán a harmadik probléma a legérdekesebb. A CDN-csomópontokat kizárólag azért létesítették, hogy tartalmat szolgáltassanak a felhasználóknak. A P2P-csomópontokat azonban nem ezért hozták létre. Ezek a felhasználók számítógépei, és a felhasználók jobban érdekeltek lehetnek egy film megszerzésében, mint abban, hogy a többi felhasználó letöltését segítsék. Azokat a csomópontokat, amelyek erőforrásokat vesznek el egy rendszerből természetbeni hozzájárulás nélkül, **potyautasoknak (free-riders)** vagy **leszívóknak, piócáknak (leechers)** nevezik. Ha túl sok van belőlük, a rendszer nem működik jól. A korábbi P2P-rendszerekről tudni lehetett, hogy befogadták ezeket [Saroiu és mások, 2003], így a BitTorrent megpróbálta a számukat minimalizálni.

A BitTorrent ügyfelekben alkalmazott megközelítés az, hogy megjutalmazzák azokat a társakat, akik jó feltöltési viselkedést mutattak. Minden egyes társ véletlenszerűen próbálgatja a többi társat, adatblokkokat hoz el tőlük, miközben adatblokkokat tölt fel hozzájuk. A társ az adatblokkokkal történő kereskedést csak azzal a kisszámú társsal folytatja, akik jó letöltési teljesítményt biztosítanak, de közben véletlenszerűen más társaknál is próbálkozik, hogy jó partnerekre találjon. A társak véletlenszerű próbálgatása az újoncok számára azt is lehetővé teszi, hogy megszerezzék az első adatblokkokat, amelyekkel a többi társsal kereskedhetnek. Azokat a társakat, akikkel egy csomópont éppen adatblokkot, töredéket cserél, **töredékmentesítőnek (unchoked)** nevezik.

Ennek az algoritmusnak az a célja, hogy idővel olyan társakat találjon, amelyek feltöltési és letöltési sebessége egymáshoz illeszthető. Minél többet ad egy társ a többi társnak, annál többet várhat viszont. A társak csoportjának használata egy társnak segít abban, hogy telítésbe vigye a letöltési sávszélességét a legnagyobb teljesítmény elérése érdekében. Ezzel szemben, ha egy társ nem tölt fel adatblokkokat a többi társhoz, vagy nagyon lassan teszi azt, akkor előbb vagy utóbb meg fogják szakítani vele a kapcsolatot, vagyis **megfojtják (choked)**. Ez a stratégia gátolja azt az antiszociális viselkedést, amikor a társak bliccelnek a bolyban.

A megfojtó algoritmust néha úgy írják le, mint a **szemet-szemért (tit-for-tat)** stratégia megvalósítását, ami ösztönzi az együttműködést az ismételt érintkezések során. Ez azonban szigorúan véve nem gátolja meg az ügyfeleket abban, hogy kihasználják a

rendszert [Piatek és mások, 2007]. Ennek ellenére az a tény, hogy figyelmet szenteltek a problémára és a hétköznapi felhasználók számára megnehezítették a bliccelést, valószínűleg közrejátszott a BitTorrent sikerében.

Ahogyan azt az ismertetőnkből láthatja, a BitTorrentnek gazdag szókincse van. Léteznek torrentek, bolyok, leszívók, feltöltők és követők, valamint fékezés (snubbing), fojtás, leselkedés (lurking) és egyebek. További információért tekintse meg a BitTorrentről szóló rövid értekezést [Cohen, 2003] és a *www.bittorrent.org*-ról elindulva nézze meg a világhálót.

DHT-k – osztott hash-táblák

A P2P állománymegosztó hálózatok felbukkanása 2000 körül nagy érdeklődést váltott ki a kutatók közösségében. A P2P-rendszerek lényege az, hogy nélkülözik a CDN-ek és más rendszerek központilag felügyelt struktúráját. Ez jelentős előny lehet. A központilag felügyelt alkotóelemek szűk keresztmetszetté válnak, amint a rendszer nagyon nagyra nő, és ezek képezik az egyedüli meghibásodási pontot. A központi alkotóelemek vezérlési helyként is használhatóak (például a P2P-hálózat kikapcsolásához). A korai P2P-rendszerek azonban csak részlegesen voltak decentralizáltak, vagy ha teljesen decentralizáltak voltak, akkor nem voltak hatékonyak.

A BitTorrent imént ismertetett hagyományos formája társ-társ átviteleket és minden bolyhoz központosított követőt használ. Kiderült, hogy a P2P-rendszereknek a követő az a része, amit nehéz decentralizálni. A fő probléma az, hogy hogyan lehet kitalálni, melyik társnál van az a meghatározott tartalom, amit keresünk. Például minden felhasználónak lehet egy vagy több olyan adata, mint például dalok, fényképek, programok, állományok és így tovább, amiket a többi felhasználó olvasni akar. A többi felhasználó hogyan fogja ezeket megtalálni? Egyszerű készíteni egy katalógust (indexet) arról, hogy kinek mije van, de ez központosított. Az nem segít, ha minden társnak van saját katalógusa. Igaz, ez elosztott, azonban olyan sok munkát igényel az összes társ katalógusának naprakészen tartása (mert a tartalom mozog a rendszerben), hogy nem éri meg a fáradságot.

A kérdés, amivel a kutatói közösség küzdött, úgy hangzott, hogy vajon lehetséges-e olyan P2P-katalógusokat (indexeket) létrehozni, amelyek teljesen elosztottak és jó teljesítőképességgel rendelkeznek. A jó teljesítőképesség alatt három dolgot értünk. Először, minden csomópont csak egy kis mennyiségű információt őriz a többi csomópontról. Ez a tulajdonság azt jelenti, hogy nem lesz költséges a katalógus naprakészen tartása. Másodszor, minden csomópont gyorsan meg tud keresni bejegyzéseket a katalógusban. Máskülönben ez a katalógus nem túl hasznos. Harmadszor, minden csomópont képes egyszerre használni a katalógust, még akkor is, ha más csomópontok jönnek-mennek. Ez a tulajdonság azt jelenti, hogy a katalógus teljesítőképessége a csomópontok számával növekszik.

A kérdésre adott válasz „Igen" volt. Négy különböző megoldást találtak ki 2001-ben. Ezek a következők: Chord (azaz húr) [Stoica és mások, 2001], CAN [Ratnasamy és mások, 2001], Pastry (azaz tészta) [Rowstron és Druschel, 2001] és Tapestry (azaz faliszőnyeg) [Zhao és mások, 2004]. Nem sokkal ezután más megoldásokat is kidolgoztak, köztük a Kademlia-t, amit a gyakorlatban is használnak [Maymounkov és Mazieres, 2002].

A megoldások **DHT (Distributed Hash Tables – osztott hash-táblák)** néven ismertek, mert egy katalógus (index) alapvető feladata az, hogy egy kulcsot egy értékhez rendeljen. Ez olyan, mint egy hash-tábla, és a megoldások természetesen elosztott változatok.

A DHT-k úgy végzik munkájukat, hogy szabályos szerkezetbe rendezik a csomópontok közötti kommunikációt, amint azt látni fogjuk. Ez a viselkedés meglehetősen különbözik azoknak a hagyományos P2P-hálózatoknak a viselkedésétől, amelyek bármilyen kapcsolatot használtak, ami csak létrejött a társak között. Ezért a DHT-ket **strukturált P2P-hálózatoknak (structured P2P networks)** nevezik. A hagyományos P2P-protokollok **strukturálatlan P2P-hálózatokat (unstructured P2P networks)** építenek ki.

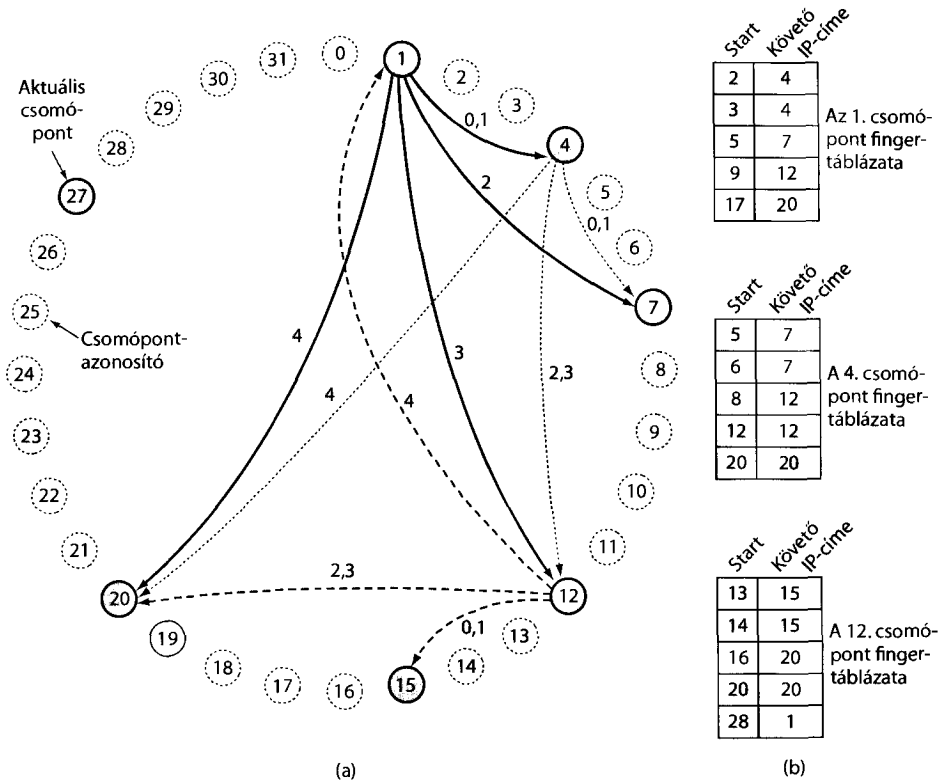
A Chord (húr) az a DHT-megoldás, amelyet ismertetni fogunk. Forgatókönyv gyanánt gondoljuk meg, hogy hogyan lehet lecserélni egy hagyományosan a BitTorrentben használt központosított követőt egy teljesen elosztott követőre! A Chord ennek a problémának a megoldására is használható. Ebben a forgatókönyvben az összefoglaló katalógus egy olyan lista, ami az összes bolyt tartalmazza, amelyekhez egy számítógép valamilyen tartalom letöltése érdekében csatlakozhat. A katalógusban történő kereséshez használt kulcs a tartalom torrentleírása. Ez egyedileg azonosít egy bolyt, amelyből a tartalom a tartalom összes adatblokkjának hash-értékeként tölthető le. A katalógusban lévő minden egyes kulcshoz tárolt érték a bolyhoz tartozó társak listája. Ezek a társak azok a számítógépek, amelyekkel kapcsolatba kell lépni a tartalom letöltéséhez. Valaki, aki le akar tölteni egy tartalmat, mint például egy filmet, csak a torrentleírással rendelkezik. A DHT által megválaszolandó kérdés az, hogy központi adatbázis hiányában ki fogja-e találni valaki, hogy (a több millió BitTorrent-csomópont közül) mely társaktól töltse le a filmet?

Egy Chord DHT n csomópontból áll. A mi forgatókönyvünk esetén ezek BitTorrentet futtató csomópontok. Minden csomópontnak van IP-címe, amelyen kapcsolatba lehet vele lépni. Az összefoglaló katalógust szétosztják a csomópontok között. Ebből az következik, hogy minden egyes csomópont a katalógus morzsáit és darabkáit tárolja, hogy azt mások is használhassák. A Chord legfontosabb része az, hogy egy látszólagos (virtuális) térben lévő azonosítókat használ a katalógus bejárásához, nem pedig a csomópontok IP-címeit, vagy a tartalom (mint például filmek) címeit. Elviekben az azonosítók egyszerű m-bites számok, amelyek egy kör mentén növekvő sorrendbe rendezhetők.

Annak érdekében, hogy egy csomópont címét azonosítóvá alakítsák, egy *hash* nevű hash-függvény segítségével leképezik azt egy m-bites számra. A Chord az SHA-1-et használja *hash*-ként. Ez ugyanaz a hash-függvény, amit a BitTorrent leírásakor említettünk. Meg fogjuk vizsgálni, amikor a kriptográfiát tárgyaljuk a 8. fejezetben. Egyelőre elegendő annyit mondanunk, hogy ez csak egy függvény, mely egy változó hosszúságú bájtfüzért kap paraméterül, és előállít belőle egy erősen véletlenszerű 160 bites számot. Ily módon felhasználhatjuk arra, hogy minden IP-címet 160 bites számmá alakítsunk, amelyet **csomópont-azonosítónak (node identifier)** nevezünk.

A 7.71.(a) ábrán megmutatjuk az azonosítók körét az $m = 5$ esetre. (Egyelőre hagyjuk figyelmen kívül a körben lévő húrokat.) Az azonosítók egy része megfelel a csomópontoknak, de a többségük nem. Ebben a példában az 1, 4, 7, 12, 15, 20 és 27 azonosítójú, sötétített karikák felelnek meg tényleges csomópontoknak; a többiek nem léteznek.

Definiáljuk most a *következő*(k) függvényt, mint a körben az óramutató járása szerint k-t követő első létező csomópont azonosítóját. Például *következő*(6) = 7, *következő*(8) = 12, és *következő*(22) = 27.



7.71. ábra. *(a) 32 csomópont-azonosítóból álló halmaz, körbe rendezve. A sötét karikák tényleges gépeknek felelnek meg. Az ívek az 1-es, 4-es és 12-es csomópontokból induló fingereket mutatják. Az íveken lévő címkék a táblázat indexei. (b) Példák a fingertáblázatokra*

Egy **kulcs (key)** is készül a tartalom nevéből a *hash* (például SHA-1) segítségével, ami 160-bites számot állít elő. A mi forgatókönyvünkben a tartalom neve a torrent. Vagyis annak érdekében, hogy megkapjuk a *torrent* (a torrentleíró állomány) kulcsát, kiértékeljük a *kulcs = hash(torrent)* kifejezést. Ez a számítás csak egy, a *hash*-re vonatkozó helyi eljáráshívás.

Új boly létrehozásához egy csomópontnak be kell szúrnia a katalógusba egy új (*torrent, saját IP-cím*) tartalmú kulcs–érték párt. Ennek elvégzéséhez a csomópont meghívja a *következő(hash(torrent))* függvényt, hogy tárolja a *saját IP-cím*-et. Ily módon a katalógus véletlenszerűen szétosztódik a csomópontok között. A hibatűrőség érdekében p darab különböző hash-függvényt is használhatnánk, hogy az adatot p számú csomópontban tároljunk, de a továbbiakban a hibatűrés lehetőségét nem tárgyaljuk.

Nem sokkal a DHT létrehozása után egy másik csomópont meg akar találni egy torrentet, hogy csatlakozhasson a bolyhoz és letölthesse a tartalmat. Egy csomópont úgy keresi meg a *torrentet*, hogy először előállítja annak **hash**-értékét a *kulcs* megszerzéséhez, másodszor a *következő(kulcs)*-ot használja, hogy megtalálja annak a csomópontnak az IP-címét, amelyik a megfelelő értéket tárolja. Az érték a bolyban lévő társak listája.

A csomópont hozzáadhatja saját IP-címét a listához, és kapcsolatba léphet a többi társsal a tartalom BitTorrent-protokollal történő letöltése érdekében.

Az első lépés könnyű, a második már nem az. Ahhoz, hogy egy bizonyos kulcshoz tartozó csomópont IP-címét megkaphassuk, minden csomópontnak fenn kell tartania bizonyos adminisztratív adatszerkezeteket. Ezek közül az egyik a körben következő csomópont IP-címe. Az 7.71. ábrán például a 4. csomópont után a 7., míg a 7. után a 12. csomópont következik.

A keresés a következők szerint folytatódhat. A kezdeményező csomópont elküld egy csomagot a sorban utána következőnek, mely tartalmazza a saját IP-címét és a keresett kulcsot. A csomag körbehalad a gyűrűn, amíg meg nem találja a keresett csomópont-azonosító követőjét. Ez a csomópont megvizsgálja, hogy van-e a keresett kulcsra vonatkozó információja. Ha van, közvetlenül visszajuttatja azt a kezdeményező csomóponthoz, aminek az IP-címét már ismeri.

Az összes csomópont közti lineáris keresés azonban igen kevéssé hatékony, ha a P2P-rendszer elég nagy, mivel a keresésnél érintett pontok átlagos száma $n/2$. A jóval gyorsabb keresés érdekében ezért minden csomópont karbantart egy úgynevezett **fingertáblázatot** (**finger table**). A táblázatnak m bejegyzése van, 0-tól $m-1$-ig indexelve, és mindegyik különböző, tényleges csomópontra mutat. A bejegyzéseknek két mezőjük van: *start* és a *következő*(*start*) csomópont IP-címe, ahogy azt a 7.71.(b) ábra példája mutatja három csomópontra. A k. csomópontban lévő i. bejegyzés mezőinek értéke:

$start = k + 2^i \ (modulo\ 2^m)$
a *következő*(*start*[*i*]) csomópont IP-címe

Figyeljük meg, hogy minden csomópont viszonylag kevés másik csomópont IP-címét tárolja, és hogy ezek többsége a csomópont-azonosítót tekintve elég közelinek mondható.

A fingertáblázat segítségével a k. csomópontban lévő *kulcs*ot a következőképp találhatjuk meg. Ha a *kulcs* a k és a *következő*(k) közé esik, akkor a *kulcs*ról információt tároló csomópont a *következő*(k), és a keresés véget ért. Ellenkező esetben megkeressük a fingertáblázatban azt a bejegyzést, melynek a *start* mezője a *kulcs* legközelebbi elődje. A keresési kérést ekkor közvetlenül az ebben a bejegyzésben található IP-címre küldjük azzal, hogy folytassa az a keresést. Mivel e csomópont már közelebb van a *kulcs*hoz, de még mindig alatta van, jó az esélye annak, hogy képes lesz visszatérni a válasszal legfeljebb néhány további kérdés után. Valójában minden keresés megfelezi a célig hátralévő távolságot, így megmutatható, hogy a keresések átlagos száma $\log_2 n$.

Első példaként keressünk rá a *kulcs* = 3 értékre az 1. csomópontban. Mivel az 1. csomópont látja, hogy a 3. közte és a követője, a 4. között van, a keresett csomópont a 4. Ezzel a keresés befejeződik, és visszatér a 4. csomópont IP-címével.

Második példaként keressük meg a *kulcs* = 16 értéket az 1. csomópontban. Mivel a 16. nem esik 1. és 4. közé, meg kell nézni a fingertáblázatot. A 16. legközelebbi elődje 9., így a kérést a 9. bejegyzéshez tartozó IP-címre küldjük, ami történetesen a 12. csomópont címe. A 12. csomópont sem tudja közvetlenül a választ, ezért megkeresi a 16.-at közvetlenül megelőző csomópontot, és megtalálja 14.-et, ami megadja a 15. csomópont IP-címét. Ezután elküldenek ennek egy lekérdezést. A 15. csomópont azt látja, hogy a

16. közé és az ő követője (a 20.) közé esik, így visszaküldi a 20. csomópont IP-címét a hívónak, aki továbbküldi azt az 1. csomópontnak.

Mivel a csomópontok bármikor bekapcsolódhatnak és távozhatnak is, a Chord-algoritmusnak valahogy ezeket a műveleteket is kezelnie kell. Feltesszük, hogy a rendszer a működése kezdetén még elég kicsi volt ahhoz, hogy a csomópontok közvetlenül információt cserélhessenek egymással, hogy felépítsék az első kört és a fingertáblázatokat. Ezt követően egy folyamat szükséges az adminisztrációhoz. Amikor egy új csomópont, r, csatlakozni akar, fel kell vennie a kapcsolatot egy létező csomóponttal és meg kell kérnie, hogy keresse meg számára a *következő*(r) IP-címét. Az új csomópont ezután meghívja *következő*(r) függvényt, hogy megtudja ki az ő elődje, majd mindkettejüket arra kéri, hogy illesszék be őt maguk közé a körbe. Például, ha a 7.71. ábrán a 24. csomópont csatlakozni akar, akkor megkéri valamelyik csomópontot, hogy keresse meg neki *következő*(24)-et, ami 27. Ekkor megkérdi 27.-et, hogy ki az ő elődje (20.). Végül mindkettejüket értesíti létezéséről, így a 20. a 24.-et követőjének fogja tekinteni, míg a 27. az elődjének. Ezenfelül a 27. csomópont átadja az új tagnak a 21. és 24. közé eső kulcsokat, melyek ezentúl hozzá fognak tartozni. Ezzel a csatlakozás lezárult.

Eközben azonban számos fingertáblázat érvénytelenné vált. Hogy ezt a hibát kijavítsák, minden csomópont futtat egy folyamatot a háttérben, mely időről időre meghívja a *következő* függvényt és újraszámol minden fingert. Ha ezen keresések közül valamelyik új csomópontra bukkan, a megfelelő fingerbejegyzés frissítődik.

Amikor egy csomópont szabályszerűen hagyja el a hálózatot, átadja kulcsait a követőjének, majd értesíti az elődjét a távozásáról, hogy az kapcsolódhasson a távozó követőjéhez. Ha azonban a csomópont váratlanul összeomlik, baj van, hiszen az elődjének nem lesz érvényes követője. Hogy kiküszöböljék ezt a problémát, a csomópontok nemcsak egy, hanem s darab közvetlen követőt tartanak számon, így katasztrófa esetén akár $s-1$ egymás után meghibásodott csomópontot is kihagyva még mindig képesek visszazárni a kört.

A DHT-k témakörében a kitalálásuk óta óriási mennyiségű kutatómunkát végeztek. Azért, hogy az olvasónak elképzelése legyen arról, mégis mennyire sokat kutatták, engedje meg, hogy feltegyünk egy kérdést: melyik minden idők legtöbbször hivatkozott hálózatokkal kapcsolatos tudományos cikke? Nehezen fog találni olyan cikket, amelyikre többször hivatkoztak, mint a megtermékenyítő Chord-cikkre [Stoica és mások, 2001]. A kutatásoknak e valóságos hegye ellenére a DHT-k alkalmazásai csak lassan kezdenek megjelenni. Néhány BitTorrent-ügyfél DHT-ket használ egy olyan teljesen elosztott követő kialakítása érdekében, mint amit leírtunk. Az olyan nagy kereskedelmi felhőszolgáltatások (cloud services), mint például az Amazon Dynamo-ja is tartalmaz DHT-módszereket [DeCandia és mások, 2007].

7.6. Összefoglalás

Az ARPANET névkezelése nagyon egyszerűen indult: egy ASCII-szöveges állomány felsorolta az összes hoszt nevét és a hozzájuk tartozó IP-címeket. Minden éjjel minden gép letöltötte ezt az állományt. De amikor az ARPANET átalakult internetté és mérete robbanásszerűen megnőtt, egy lényegesen kifinomultabb és dinamikus névkezelési sémára

volt szükség. Az, amit most használnak, egy hierarchikus séma, melyet körzetnévkezelő rendszernek neveznek. Ez az interneten elérhető összes gépet fák halmazába szervezi. A legfelső szinten találhatóak a jól ismert általános körzetek, köztük a *com* és az *edu*, valamint körülbelül 200 országhoz tartozó körzet. A DNS-t elosztott adatbázisként valósították meg, szerte a világon elhelyezett kiszolgálók segítségével. A DNS-kiszolgáló lekérdezésével egy folyamat IP-címre képezheti le egy internetkörzet nevét, amit arra használ, hogy a körzetbe tartozó számítógéppel kommunikáljon.

Az e-levél az internet első sikeres alkalmazása. Még ma is széles körben használja mindenki, a kisgyerekektől a nagyszülőkig. A világ e-levél rendszereinek többsége az immár az RFC 5321-ben és 5322-ben definiált levelezőrendszert használja. Az üzeneteknek egyszerű ASCII-fejrészeik vannak, és sokféle tartalom küldhető MIME használatával. Különböző felhasználói ügynökök, köztük webalkalmazások küldik be a leveleket az üzenettovábbító ügynököknek kézbesítés okán, és kérik le tőlük megjelenítés céljából. A beküldött leveleket az SMTP segítségével továbbítják, ami úgy működik, hogy egy TCP-összeköttetést épít ki a küldőtől a fogadó üzenettovábbító ügynökig.

A web az az alkalmazás, amire az emberek többsége úgy gondol, mintha az lenne az internet. Ez a rendszer eredetileg a (HTML-ben írt) hiperszöveg oldalak számítógépek közötti hivatkozásainak zökkenőmentes megvalósítására szolgált. Az oldalakat a böngészőtől a kiszolgálóig kiépített TCP-összeköttetéssel és HTTP használatával töltik le. Manapság a weben található tartalom nagy részét dinamikusan hozzák létre, vagy a kiszolgálón (például PHP-vel) vagy a böngészőben (például JavaScript-tel). Amikor ezt háttéradatbázissal kombinálják, a kiszolgálón létrehozott dinamikus oldalak olyan alkalmazásokat tesznek lehetővé, mint például az e-kereskedelem és a keresés. A böngészőben létrehozott dinamikus oldalak teljes értékű alkalmazásokká fejlődnek, mint például az e-levél ügynök, ami a böngészőben fut, és webprotokollokat használ a távoli kiszolgálókkal történő kommunikációhoz.

A gyorstárazást és a tartós összeköttetéseket széles körben alkalmazzák a web teljesítőképességének növelésére. A világháló mobil eszközökön történő használata a sávszélesség és a mobil készülékek számítási teljesítményének növekedése ellenére is kihívást jelenthet. A webhelyek gyakran a kis kijelzőjű eszközökhöz szabott oldalakat küldenek kisebb képekkel és kevésbé összetett navigálással.

A webprotokollokat egyre növekvő mértékben használják a gépek közötti kommunikációban. A tartalom leírására a HTML-lel szemben előnyben részesítik az XML-t, amit a gépek könnyen feldolgoznak. A SOAP egy RPC-mechanizmus, ami HTML használatával XML-üzeneteket küld.

A digitális hang és mozgókép az internet fő mozgatórugói 2000 óta. Az internet forgalmának többsége ma mozgókép. Közülük sokat webhelyek közvetítenek folyamszerűen különféle protokollok segítségével (beleértve az RTP/UDP- és RTP/HTTP/TCP-protokollokat). Az élő médiát sok fogyasztóhoz közvetítik. Ebbe beletartoznak az internetes rádiók és tévéállomások, amelyek mindenféle eseményt közvetítenek. A valós idejű hangot és mozgóképet valós idejű konferenciahívásra is használják. Sok hívás IP-hálózaton keresztül történő beszédátvitellel valósul meg a hagyományos telefonhálózat használata helyett, beleértve a videokonferenciát is.

Létezik néhány rendkívül népszerű webhely, és nagyon nagy számú kevésbé népszerű is. A népszerű webhelyek kiszolgálása érdekében kifejlesztették a tartalomelosztó hálózatokat. A CDN-ek a DNS-t használják az ügyfél egy közeli kiszolgálóhoz irányítására. A kiszolgálókat adatközpontokban helyezik el, szerte a világon. Egy másik lehetőségként a P2P-hálózatok a gépek egy gyűjteménye számára lehetővé teszik a tartalom, például filmek egymás közötti megosztását. Olyan tartalomelosztó kapacitást biztosítanak, ami a P2P-hálózatban található gépek számával skálázódik, és ami a legnagyobb webhelyekkel is versenyre kelhet.

7.7. Feladatok

1. Sok vállalati számítógépnek három különböző és globálisan egyedi azonosítója van. Melyek ezek?

2. A 7.4. ábrán a *laserjet* szó után nincs pont. Miért?

3. Képzeljen el egy olyan helyzetet, amelyben egy kiberterrorista eléri, hogy a világ összes DNS-kiszolgálója egyszerre omoljon össze! Hogyan változtatja ez meg valakinek az internet használatára való képességét?

4. A DNS TCP helyett UDP-t használ. Ha egy DNS-csomag elveszik, nincs automatikus helyreállítás. Okoz-e ez problémát, és ha igen, hogyan oldják azt meg?

5. János egy eredeti körzetnevet szeretne, és egy véletlenített programot használ arra, hogy előállítson magának egy másodlagos körzetnevet. Be akarja jegyeztetni ezt a körzetnevet a *com* általános körzetbe. Az előállított körzetnév 253 karakter hosszú. A *com* adminisztrátora meg fogja engedni ennek a körzetnévnek a bejegyzését?

6. Lehetséges-e az, hogy egy gépnek egyetlen DNS-neve, de több IP-címe legyen? Hogyan fordulhat ez elő?

7. A webhellyel rendelkező vállalatok száma az elmúlt években exponenciálisan nőtt. Ennek eredményeként több ezer vállalat regisztrálta magát a *com* körzetbe, ami a körzet legfelső szintű kiszolgálóját erős terhelésnek tette ki. Javasoljon megoldást a probléma enyhítésére anélkül, hogy megváltoztatná a névkezelési sémát (azaz új elsődleges körzetnevek bevezetése nélkül)! Az is elfogadható, ha a megoldása az ügyfél szoftverének megváltoztatását is megköveteli.

8. Egyes levelezőrendszerek egy *Tartalom visszaküldése (Content Return)* nevű fejrészmezőt is támogatnak. Ez azt adja meg, hogy vissza kell-e küldeni az üzenet törzsét abban az esetben, ha az üzenet nem kézbesíthető. Ez a mező vajon a borítékhoz vagy a fejrészhez tartozik?

9. Az elektronikus levelezőrendszerek használatához könyvtárszolgáltatások szükségesek, hogy ki lehessen keresni az emberek e-levél címeit. Az ilyen könyvtárak ösz-

szeállításához a neveket szabványos részekre (például családi név, keresztnév) kell bontani, hogy a keresés megoldható legyen. Elemezzen néhány problémát, amit meg kell oldani ahhoz, hogy egy ilyen világméretű szabvány elkészülhessen!

10. Egy nagy ügyvédi iroda, amelynek számos alkalmazottja van, minden egyes munkatársa számára egyedi e-levél címet biztosít. Minden dolgozó e-levél címe ilyen alakú: *<bejelentkezés>@ugyvediiroda.com*. Az iroda azonban nem határozta meg egyértelműen a bejelentkezés formátumát. Tehát néhány munkatárs a keresztnevét használja bejelentkezési névként, néhányan a vezetéknevüket, néhányan a monogramjukat, és így tovább. Az iroda most rögzített formátumot szeretne létrehozni, például:

keresztnev.vezeteknev@ugyvediiroda.com

amit az összes alkalmazott e-levél címéhez használhatnának. Hogyan lehetne ezt megoldani anélkül, hogy a hajót túlságosan meghimbálnánk?

11. Egy bináris állomány 4560 bájt hosszú. Milyen hosszú lesz akkor, ha base64 kódolást használunk, minden 110 elküldött bájt után és a végén egy CR+LF párral?

12. Nevezzen meg öt olyan MIME-típust, melyek nem szerepelnek ebben a könyvben! További információt böngészőjében vagy az interneten találhat.

13. Tegyük fel, hogy egy MP3-állományt szeretne elküldeni barátjának, de a barátja internetszolgáltatója minden egyes beérkező levél méretét 1 MB-ra korlátozza, miközben az MP3-állomány mérete 4 MB. Van-e valamilyen mód a helyzet kezelésére az RFC 5322 és MIME használatával?

14. Tételezzük fel, hogy János éppen most állított be egy olyan automatikus levéltovábbító mechanizmust az összes üzleti e-levelét fogadó munkahelyi e-levél címén, amelyik továbbítja leveleit a személyes e-levél címére, amit a feleségével közösen használ. János felesége erről nem tudott, és működésbe hozott egy távollétet jelző ügynököt a személyes felhasználói fiókukon. Mivel János továbbküldte a leveleit, ezért nem állított be távollétet jelző ügynököt a munkahelyi gépén. Mi történik akkor, amikor e-levél érkezik János munkahelyi e-levél címére?

15. Az RFC 5322-ben, akárcsak más szabványokban, pontos nyelvtanra van szükség, mely megmondja, hogy mit szabad és mit nem, hogy a különböző megvalósítások együtt tudjanak működni. Még az egyszerű elemeket is gondosan definiálni kell. Az SMTP-fejrészekben megengedett a white space (puha szóköz) karakterek használata a tokenek között. Adjon *két* elfogadható alternatív definíciót a tokenek közötti white space karakterekre!

16. A távollétet jelző ügynök vajon a felhasználói ügynök vagy az üzenettovábbító ügynök részét képzi-e? Természetesen a felhasználói ügynökkel együtt telepítik, de valóban a felhasználói ügynök küldi a válaszokat? Indokolja válaszát!

17. A Chord-algoritmus egy egyszerű változatában a társ-társ kikereséshez a keresések nem használnak fingertáblázatokat. Ehelyett egyenletesen helyezkednek el a kör mentén, mindkét irányban. Meg tudja egy csomópont pontosan jósolni, hogy melyik irányban kellene keresnie? Indokolja válaszát!

18. Az IMAP lehetővé teszi, hogy a felhasználók a távoli postaládájukból letölthessék a leveleiket. Ez azt jelenti, hogy a postaládák belső formátumát szabványosítani kell, hogy bármelyik ügyféloldali IMAP-program el tudja olvasni bármelyik levelezőszerver postaládáját? Indokolja válaszát!

19. Figyelje meg a Chord kört a 7.71. ábrán! Tételezzük fel, hogy a 18. csomópont hirtelen rákapcsolódik a hálózatra. Az ábra fingertáblázatai közül melyek érintettek és hogyan?

20. Mit használ a webes levelezés: POP3-mat, IMAP-ot, vagy egyiket sem? Ha az első két lehetőség közül az egyiket, akkor miért pont azt? Ha egyiket sem, akkor jellegénél fogva melyikhez áll a legközelebb?

21. Amikor a weboldalakat kiküldik, akkor MIME-fejrészek előzik meg azokat. Miért?

22. Lehetséges-e az, hogy amikor a felhasználó a Firefoxban kattint rá egy hivatkozásra, akkor elindul egy bizonyos segédalkalmazás, de ha ugyanarra a hivatkozásra Internet Explorerben kattint rá, akkor egy teljesen más segédalkalmazás indul el még akkor is, ha mindkét esetben ugyanaz a MIME-típus érkezik vissza? Indokolja válaszát!

23. Nem említettük ugyan a szövegben, de az URL használhatja az IP-címet is a DNS-név helyett. Ennek az információnak a segítségével magyarázza meg, miért nem végződhet egy DNS-név számjegyre!

24. Képzeljük el, hogy valaki a Stanford Egyetem Matematika Tanszékén épp most írt egy bizonyítást tartalmazó új dokumentumot, amit FTP-vel kíván eljuttatni kollégáihoz, hogy azok áttekinthessék. A dokumentumot az *ftp/pub/attekintendo/ujTetel.pdf* FTP-könyvtárba helyezi. Feltételezése szerint mi lesz a dokumentum URL-je?

25. A 7.22. ábrán a *www.aportal.com* a felhasználói beállításokat egy sütiben tartja számon. Ennek a sémának az a hátránya, hogy a sütik mérete 4 KB-ban korlátozott, így ha a beállítások terjedelmesek, például van köztük sok részvény, sportcsapat, újsághír-kategória, több város időjárása, akciók számos termékcsoportra vonatkozóan, és egyebek, akkor elérhető a 4 KB-os határ. Dolgozzon ki egy alternatívát a beállítások számontartására, melynél nem jelentkezik ez a probléma!

26. A Lajhár Bank az online banki műveleteket egyszerűbbé szeretné tenni lusta ügyfelei számára, ezért miután az ügyfél belép és azonosítja magát a jelszavával, a bank visszaküld neki egy sütit, ami az ügyfél banki azonosítóját tartalmazza. Ily módon az

ügyfélnek a későbbi online banklátogatások alkalmával nem kell azonosítania magát vagy a jelszavát begépelni. Mit gondol erről az ötletről? Működni fog? Jó ötlet?

27. (a) Figyelje meg a következő HTML-címkét:

```
<h1 title="ez a címsor"> CÍMSOR 1 </h1>
```

Milyen körülmények közt és hogyan használja a *TITLE* attribútumot a böngésző?
(b) Miben tér el a *TITLE* attribútum az *ALT* attribútumtól?

28. Hogyan lehet megoldani HTML-ben, hogy egy képre rá lehessen kattintani? Adjon egy példát!

29. Írjon olyan HTML-oldalt, amely egy hivatkozást tartalmaz a *felhasznalonev@KorzetNev.com* e-levél címre! Mi történik, amikor a felhasználó rákattint erre a hivatkozásra?

30. Írjon több hallgatót felsoroló XML-oldalt egy egyetemi adminisztrátor számára, amely tartalmazza mindegyikük nevét, címét és tanulmányi átlagát!

31. Mondja meg, hogy a következő alkalmazások esetén a PHP-szkript és a JavaScript közül (1) melyiket lehetséges és (2) melyiket célszerű alkalmazni, és miért!
(a) Tetszőleges kért hónap naptárának megjelenítése 1752 szeptemberével kezdődően.
(b) Az Amszterdamból New Yorkba tartó repülőjáratok menetrendjének megjelenítése.
(c) Polinom felrajzolása a felhasználó által megadott együtthatókból.

32. Írjon egy JavaScript-programot, mely egy 2-nél nagyobb egészről eldönti, hogy prímszám-e! Megjegyezzük, hogy a JavaScriptben megtalálhatók az if és while utasítások, ugyanazzal a szintaxissal, mint a Javában vagy a C-ben. A modulo operátor a %. *x* négyzetgyökét a *Math.sqrt(x)* paranccsal kaphatja meg.

33. Adott a következő HTML-oldal:

```
<html> <body>
<a href="www.info-source.com/welcome.html"> Az információért kattintson ide! </a>
</body> </html>
```

Ha a felhasználó a hiperhivatkozásra kattint, kiépül egy TCP-összeköttetés, és egy parancssorozatot küldenek át a kiszolgálónak. Soroljon fel minden átküldött sort!

34. A *Ha-változott-azóta* fejrész segítségével ellenőrizni lehet, hogy a gyorstárban lévő oldal még mindig érvényes-e. Az ellenőrzést nem csak HTML-t, hanem képeket, hangot, videót stb. tartalmazó oldalakra is el lehet végezni. Mit gondol, ennek a módszernek a hatékonysága JPEG-képek esetében a HTML-hez viszonyítva jobb vagy rosszabb? Jól gondolja meg, mit ért „hatékonyság" alatt, és indokolja válaszát!

35. Egy nagyobb sportesemény, például valamilyen népszerű sportág bajnoki döntőjének napján, sokan keresik fel az esemény hivatalos weboldalát. Ez is olyan „hirtelen tömeg" jelenségnek számít, mint a floridai elnökválasztás 2000-ben? Miért?

36. Van-e annak értelme, hogy egyetlen internetszolgáltató CDN-ként működjön? Ha igen, hogyan valósítható ez meg? Ha nem, mi a baj az ötlettel?

37. Tételezzük fel, hogy az audio CD-k nem használnak tömörítést. Hány MB adatot kell a kompakt lemeznek tartalmaznia, hogy képes legyen kétórányi zene lejátszására?

38. A 7.42.(c) ábrán a kvantálási zaj a 3 bites minták miatt következik be. Az első minta, a 0, még pontos, de a következő néhány nem az. Mi a periódus 1/32-énél, 2/32-énél és 3/32-énél levő minták százalékos hibaaránya?

39. Lehetne-e pszichoakusztikus modellt használni az internettelefóniához szükséges sávszélesség csökkentésére? Ha igen, lennének-e feltételei a modell alkalmazásának, és milyenek? Ha nem, miért nem?

40. Egy hangközvetítő kiszolgáló 100 ms-nyi „távolságra" van egy médialejátszótól. A kiszolgáló kimeneti sebessége 1 Mb/s. Ha a médialejátszónak 2 MB-os puffere van, mit tud mondani az alsó és felső vízjelek helyzetéről?

41. Az IP-n keresztül történő hangátvitel is ugyanazokkal a problémákkal szembesül a tűzfalak esetében, mint a hangközvetítés? Indokolja válaszát!

42. Mekkora bitsebességgel lehet tömörítetlen, 1200 × 800 képpontos, színes képeket képpontonként 16 bites színmélység és 50 kép/s sebesség esetén átvinni?

43. Egy MPEG-keretben előforduló 1 bites hiba befolyásolhat többet, mint azt a keretet, amelyben előfordult? Indokolja válaszát!

44. Vegyük egy 50 000 ügyfeles videokiszolgáló példáját, ahol minden ügyfél három filmet néz meg havonta. Tegyük fel, hogy a filmek kétharmadát este 9 és 11 óra között kell leadni. Mennyi filmet kell ez alatt az idő alatt egyszerre átvinnie? Ha minden film 6 Mb/s-ot igényel, hány OC-12-es összeköttetésre lesz szüksége a kiszolgálónak a hálózathoz?

45. Tegyük fel, hogy Zipf törvénye érvényes egy 10 000 filmet tartalmazó kiszolgálóhoz intézett kérésekre. Ha a kiszolgáló az 1000 legnépszerűbb filmet memóriában, a többi 9000-et lemezen tartja, adjon egy kifejezést az összes hozzáférés azon törtrészére, amely a memóriához irányul. Írjon egy kis programot, amely numerikusan kiértékeli ezt a kifejezést.

46. Egyes cyber-telepesek olyan körzetneveket foglaltak le, melyek ismert vállalati oldalak félregépelt változatai, mint például a *www.microsfot.com*. Soroljon fel legalább öt ilyen körzetet!

47. Sokan jegyeztettek be *www.valami.com* alakú körzetneveket, ahol a *valami* egy gyakori szó. A következő kategóriák közül mindegyikhez adjon meg öt webhelyet és jellemezze azokat röviden (például a *www.stomach.com* egy Long Island-i gasztroenterológus oldala)! A kategóriák: állatok, ételek, háztartási tárgyak, testrészek. Kérem, az utolsó kategóriánál szorítkozzon a derékon felüli testrészekre!

48. Írja át a 6.6. ábra kiszolgálóját úgy, hogy igazi webszerver legyen, és használja a HTTP 1.1 *GET* parancsát, valamint fogadja el a *Host* üzenetet is! A kiszolgáló tartson fenn egy gyorstárat a lemezről nemrég betöltött állományok számára, és a kéréseket a gyorstárból elégítse ki, amikor ez lehetséges!

8. Hálózati biztonság

Létezésének első pár évtizedében a számítógép-hálózatokat elsődlegesen egyetemi kutatók elektronikus levelek küldésére, illetve hivatalokban nyomtatók megosztására használták. Ilyen feltételek mellett a biztonság kérdésének nem sok figyelmet szenteltek. Napjainkban azonban, amikor hétköznapi emberek milliói használják a hálózatokat banki műveletek közben, vásárláshoz és adóbevallásuk elkészítéséhez, a hálózati biztonság kérdése komoly problémaként dereng fel a láthatáron. A most következő részekben a hálózati biztonságot több nézőpontból fogjuk megvizsgálni, megmutatva számos buktatóját, és ismertetve azokat az algoritmusokat és protokollokat, amelyek alkalmazásával a hálózatok biztonságosabbá tehetők.

A biztonság igen tág fogalom, és a támadási módok széles skálája elleni védekezést tartalmazza. Legegyszerűbb formája, amikor biztosítani szeretnénk, hogy kíváncsi emberek ne olvashassák el, és ami még rosszabb, ne módosíthassák különböző címzettekhez küldött üzeneteinket. Tartalmazza azokat az eseteket, amikor felhasználók olyan távoli szolgáltatásokat szeretnének elérni, amelyekhez nincs joguk. Foglalkozik azzal a kérdéssel, hogy hogyan döntsük el a következő, látszólag az adóhatóságtól érkezett üzenetről, hogy valóban az adóhatóságtól és nem a maffiától jött: „Fizess péntekig, különben...". A biztonság azokra a problémákra is választ keres, hogyan lehet azonosítani az elfogott és újra lejátszott üzeneteket, valamint azokat a személyeket, akik tagadják, hogy bizonyos üzeneteket elküldtek.

A legtöbb biztonsági problémát rosszindulatú emberek szándékosan okozzák, hogy előnyhöz jussanak, a figyelmet magukra vonják vagy másoknak kárt okozzanak. A leggyakoribb elkövetők közül mutat be párat a 8.1. ábra. Az ábra alapján világosnak kell lennie: egy hálózat biztonságossá tételéhez sokkal több kell, mint pusztán a programhibák kijavítása. Gyakran intelligens, specializálódott, megfelelő alapokkal rendelkező támadók eszén kell túljárni. Az is nyilvánvaló, hogy az egyszerű alkalmi támadókat megfékező intézkedések nem jelentenek számottevő akadályt a komoly betolakodóknak. A rendőrségi feljegyzések pedig azt mutatják, hogy a legtöbb károkozó támadást nem a telefonvonalat lehallgató kívülállók, hanem sértett alkalmazottak követik el. A biztonsági rendszereket tehát ennek tudatában kell megtervezni.

A hálózati biztonsággal kapcsolatos problémák nagyjából négy, szorosan összefüggő területre oszthatók: titkosság (secrecy vagy confidentiality), hitelesség (authentication), letagadhatatlanság (nonrepudiation) és sértetlenség (integrity). A titkosság azzal foglalkozik, hogy az információ ne juthasson illetéktelen kezekbe. Általában ez az, ami az

Támadó	Cél
Diák	Örömét leli mások elektronikus levelének elolvasásában
Betörő (Cracker)	Különböző biztonsági rendszereket tesz próbára, adatokat tulajdonít el
Kereskedelmi képviselő	Elhiteti, hogy egész Európát, és nem csak Andorrát képviseli
Üzletember	A konkurencia üzleti stratégiáját szeretné felderíteni
Elbocsátott alkalmazott	Bosszút áll, amiért kirúgták
Könyvelő	A vállalat pénzét sikkasztja el
Tőzsdei bróker	Egy vevőnek elektronikus levélben tett ígéretét szeretné letagadni
Szélhámos	Bankkártyaszámokat szerez meg, hogy azokat eladja
Kém	Az ellenség katonai vagy ipari titkait szeretné megismerni
Terrorista	Biológiai fegyverekkel kapcsolatos titkokat próbál megszerezni

8.1. ábra. *Néhány embertípus, akik biztonsági problémákat okoznak, és a motivációik*

embereknek a hálózati biztonságról az eszükbe jut. A hitelesség abban segít, hogy meggyőződjünk arról, kivel állunk kapcsolatban, mielőtt érzékeny adatokat fednénk fel vagy üzleti megállapodást kötnénk. A letagadhatatlanság aláírásokkal foglalkozik: hogyan bizonyítható, hogy ügyfeled elektronikus úton 10 millió darab bal kezes bokszkesztyűt rendelt 89 centért, amikor később azt állítja, hogy az ár 69 cent volt? Vagy az is lehet, hogy azt állítja, soha nem küldött semmilyen megrendelést. Végül, hogyan bizonyosodhatunk meg arról, hogy a kapott üzenet valóban az, amelyiket elküldték, nem pedig egy a rosszindulatú ellenfél által a továbbítás alatt módosított vagy kitalált küldemény.

Mindezek a kérdések (a titkosság, a hitelesség, a letagadhatatlanság és a sértetlenség biztosítása) a hagyományos rendszerekben is jelentkeznek, azonban némi eltéréssel. A titkosságot és a sértetlenséget ajánlott levéllel és az iratok lezárásával oldják meg. Egy postavonat kirablása ma már jóval nehezebb, mint Jesse James idején volt.

Ugyanígy, az emberek többnyire meg tudják különböztetni az eredeti dokumentumot a fénymásolattól, ami gyakran lényeges. Próbaképpen készíts másolatot egy eredeti csekkről. Próbáld meg beváltani a csekket hétfőn a bankodban. Azután kedden próbálkozz a fénymásolattal. Vizsgáld meg a bank viselkedésében beállt változást. Az elektronikus csekkeknél az eredeti és a másolat megkülönböztethetetlenek. Időbe telhet, míg a bankok használni kezdik ezeket.

Az embereket az arcuk, hangjuk és kézírásuk alapján azonosítunk. Aláírásunk bizonyítéka lehet a céges levélpapírra tett kézjegyünk, domború bélyegző vagy más egyéb. A hamisítványt többnyire leleplezik a kézírás-, papír- vagy festékszakértők. E lehetőségek közül egyik sem áll rendelkezésünkre az elektronikus világban. Nyilvánvalóan más megoldásokat kell találni.

Mielőtt maguk a módszerek tárgyalását megkezdenénk, érdemes egy kis időt szánni arra, hogy meghatározzuk, vajon mindegyik protokollréteghez hozzátartozik-e a

hálózati biztonság. Könnyen megeshet, hogy nem helyezhető kizárólag egyetlen helyre. Minden réteg hozzájárulhat valamivel. A fizikai rétegben a vezeték megcsapolását meghiúsíthatjuk a vezetéknek lepecsételt és argon gázzal feltöltött csőbe helyezésével. Minden kísérlet, amelynek során a csőbe szeretnének hatolni, a gáz elszökésével, és így nyomáscsökkenéssel jár, ami riasztást válthat ki. Néhány katonai rendszerben használják ezt a módszert.

Az adatkapcsolati rétegben kétpontos vonalon haladó kereteket kódolással titkosíthatjuk, amikor elhagyja az egyik gépet, és a titkosítást visszakódolhatjuk, visszafejthetjük, amikor a keret a másikra megérkezik. Minden lépést elvégezhetünk az adatkapcsolati rétegben anélkül, hogy a felsőbb rétegek tudnának róla. Ennek a megoldásnak akkor jelentkeznek a korlátai, amikor a csomagoknak több útválasztón kell keresztüljutniuk, mivel ekkor mindegyik eszközön vissza kell kódolni a csomagot, sebezhetővé téve útválasztón belüli támadásokkal szemben. Emellett nem teszi lehetővé, hogy csak egyes kapcsolatokat védjünk (például online vásárlást bankkártyával), másokat pedig nem. Mindezek ellenére az **adatkapcsolati titkosítás (link encryption)**, ahogy a fenti módszert nevezik, könnyen használható bármilyen hálózaton, és gyakran hatásos.

A hálózati rétegben tűzfalakat telepíthetünk, hogy egyes csomagokat a hálózaton belül, vagy másokat azon kívül tartsunk. Az IP-s biztonsági funkciók szintén ebben a rétegben működnek.

A szállítási rétegben teljes összeköttetéseket titkosíthatunk, végponttól végpontig, vagyis alkalmazási folyamattól alkalmazási folyamatig. A maximális biztonság eléréséhez ilyen végponttól végpontig terjedő eljárásokra van szükség.

Végül olyan kérdések is vannak – ilyen például a felhasználók hitelesítése vagy a letagadhatatlanság –, melyeket csak az alkalmazási rétegben lehet kezelni.

Mivel a biztonság egyik rétegbe sem illik igazából, nem illett bele a könyv egyik fejezetébe sem, ezért kapott egy saját fejezetet.

Jóllehet, ez a fejezet hosszú, műszaki jellegű és alapvető fontosságú, ugyanakkor pillanatnyilag nem teljesen a tárgyhoz tartozónak számít. Jól dokumentált ugyanis az a tény, hogy a legtöbb biztonsági hiba, például a bankokban, sokkal inkább a hanyag biztonsági eljárások és a nem hozzáértő alkalmazottak, valamint a megvalósításkor elkövetett számos hiba következménye, amelyek lehetővé teszik, hogy illetéktelen felhasználók távolról betörjenek, továbbá az ún. social engineering[1] támadások következménye, ahol az ügyfeleket trükkel ráveszik arra, hogy kiadják bankszámlájuk részleteit. Ezek a biztonsági problémák sokkal gyakoribbak, mint az, hogy ügyes bűnözők lehallgatják a telefonvonalakat és azután visszafejtik a titkosított üzeneteket. Ha valaki simán besétálhat bármelyik bankfiókba egy utcán talált bankkártyával, azt állítva, hogy elfelejtette a PIN-kódját, és erre ott rögtön kap egy újat (a jó ügyfélkapcsolatra törekvés nevében), akkor a világ

1 A social engineering azoknak a módszereknek, megoldásoknak az összessége, amelyek alkalmasak arra, hogy embereket manipuláljanak annak érdekében, hogy tőlük bizalmas adatokat nyerjenek, vagy rávegyék őket arra, hogy olyan cselekedeteket hajtsanak végre, amelyeket maguktól nem tennének meg. Jóllehet a tipikus módszerek között csalás, trükközés vagy becsapás szerepel információgyűjtés vagy számítógéphez történő hozzáférés céljából, a támadó ezeknek a módszereknek az alkalmazása során sohasem kerül az áldozattal személyes kapcsolatba. *(A lektor megjegyzése)*

összes rejtjelezésével sem lehet megelőzni a visszaéléseket. E tekintetben Ross Anderson [2008a] könyve igazi leleplezésnek minősül, mivel számos iparágból több száz biztonsági hibára ad példát, melyek közül szinte mind – finoman szólva – a hanyag üzleti gyakorlatnak vagy nemtörődömségnek a következménye. Mindazonáltal az a műszaki alap, amelyre az e-kereskedelem épül, ahol mindezeket a dolgokat jól csinálják, a kriptográfia.

A fizikai rétegbeli biztonságot leszámítva szinte minden biztonsági eljárás kriptográfiai elveken alapszik. A biztonság tárgyalását ennélfogva mi is a kriptográfia részletesebb tanulmányozásával kezdjük. A 8.1. szakaszban áttekintjük a főbb alapelveket, a 8.2–8.5. szakaszokban pedig a kriptográfiában használatos alapvető algoritmusok és adatszerkezetek közül vizsgálunk meg néhányat. Ezt követően részletesen is megnézzük, hogyan lehet ezeket az elveket a gyakorlatban is alkalmazni a hálózati biztonság elérésére. A sort néhány rövid, a technikáról és a társadalomról szóló gondolattal zárjuk.

Mielőtt nekivágnánk, még valamit meg kell említenünk: mi az, amit nem tárgyalunk? Nos, igyekeztünk nem belemélyedni az operációs rendszerek és az alkalmazások biztonsági kérdéseibe, és pusztán a hálózatok területére összpontosítani, bár gyakran nehéz meghúzni a határt. Könyvünkben nem esik szó például a biometriai alapú felhasználó-hitelesítésről, a jelszavas biztonságról, puffer-túlcsordulásos támadásokról, trójai falovakról, a bejelentkezéskori megtévesztésről, kódbeszúrásról, vírusokról, férgekről és hasonlókról. Ezeket témákat hosszasan tárgyalja a *Modern operációs rendszerek* [Tanenbaum, 2007] 9. fejezete. Az érdeklődő olvasó ugyanebben a könyvben a biztonság rendszervonatkozású kérdéseiről is olvashat. Most pedig kezdjük meg utazásunkat!

8.1. Kriptográfia

A **kriptográfia** (**cryptography**) elnevezés a görög „titkos írás" szavakból ered. Hosszú és színes története évezredekre nyúlik vissza. Ebben a szakaszban csak a legfontosabb részletei közül vázolunk fel párat, hogy háttér-információt biztosítsunk a későbbi szakaszokhoz. Aki a kriptográfia teljes történelmére kíváncsi, annak ajánljuk Kahn [1995] könyvét. A jelenleg használt legmodernebb biztonsági és kriptográfiai algoritmusokról, protokollokról és alkalmazásokról Kaufman és mások [2002] műve ad átfogó tájékoztatást. A biztonsági kérdések matematikai megközelítésével Stinson [2002] művében, egy kevésbé matematikai megközelítéssel pedig Burnett és Paine [2001] írásában találkozhatunk.

A szakértők különbséget tesznek a rejtjel és a kód között. A **rejtjel** (**cipher**) egy karakterről karakterre vagy bitről bitre történő átalakítást takar, mely nem veszi figyelembe az üzenet nyelvi szerkezetét. Ezzel szemben a **kód** (**code**) egy szót helyettesít egy másik szóval vagy szimbólummal. A kódok ma már nem használatosak, pedig igazán dicsőséges múltra tekinthetnek vissza. A valaha kidolgozott legsikeresebb kódot az amerikai hadsereg használta a II. világháborúban a Csendes-óceánon. Ők egész egyszerűen egymással beszélő navajo indiánokat állítottak hadrendbe, akik sajátos navajo szavakat használtak a katonai kifejezésekre, például *chay-dagahi-nail-tsaidi*-nak (szó szerint: teknősölő) mondták a páncéltörő fegyvert. A navajo nyelv erősen tonális, rendkívül bonyolult, és nincs írott formája. Japánban ráadásul senki nem tudott róla semmit.

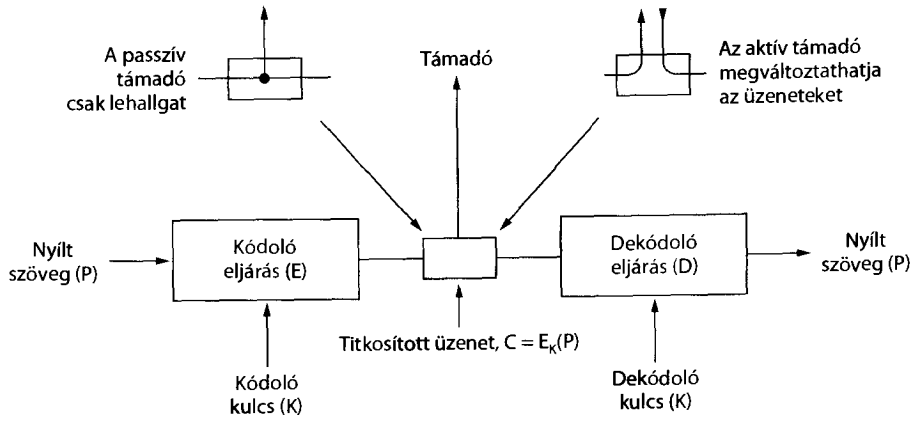
1945 szeptemberében a *San Diego Union* a következőket írta a kódról: „A japánok három éven át a tengerészgyalogosok minden partraszállásánál különös, kotyogó hangokra lettek figyelmesek, melyekbe más hangok is vegyültek: ezek egy tibeti szerzetes kiáltására és a forralt víz kitöltésének hangjára emlékeztettek." A japánok sosem fejtették meg a kódot, sok Navajo „kódbeszélőt" pedig magas katonai érdemrenddel tüntettek ki kiemelkedő szolgálatáért és bátorságáért. Az amerikaiak csendes-óceáni győzelmeiben döntő szerepet játszott az a tény, hogy az amerikaiaknak sikerült feltörniük a japán kódot, a japánok viszont sohasem tudták feltörni a Navajo-kódot.

8.1.1. Bevezetés a kriptográfiába

Eredetileg négy csoport alkalmazta és vitte tovább a kriptográfia mesterségét: a hadsereg, diplomáciai testületek, naplóírók és a szerelmesek. Ezek közül a hadseregnek volt a legfontosabb szerepe, és ez alakította ki leginkább ezt a területet. A katonai szervezetekben a titkosítandó üzeneteket régebben alacsonyan fizetett hivatalnokok kezébe adták, hogy azokat kódolják és küldjék tovább. Csupán az üzenetek óriási mennyisége is megakadályozta, hogy ezt a munkát pár elit specialista végezze el.

A számítógépek eljöveteléig a kriptográfia egyik legnagyobb korlátját a kódolást végző tisztségviselők azon képessége jelentette, hogyan képesek egyszerű eszközökkel, gyakran háborús körülmények között elvégezni a szükséges transzformációkat. További nehézséget okozott az egyik titkosító módszerről egy másikra való gyors áttérés, mivel ez sok ember átképzését vonta maga után. Mindezek ellenére a veszély, amit egy titkosító ügynök elfogása jelentett, szükségessé tette, hogy ilyen esetekben azonnal megtörténhessen a módszerbeli váltás. Ezek az egymásnak ellentmondó követelmények hozták létre a 8.2. ábrán látható modellt.

A kódolandó üzeneteket, melyeket **nyílt szövegnek** (**plaintext**) hívunk, egy olyan függvénnyel transzformálunk, melynek paramétere egy **kulcs** (**key**). A titkosító eljárás kimenetét, melyet **titkosított szövegnek** (**ciphertext**) hívunk, továbbítjuk gyakran



8.2. ábra. *A titkosító modell (szimmetrikus kulcsú titkosítás esetén)*

rádió vagy futár segítségével. Feltételezzük, hogy az ellenség vagy a támadó hallja és pontosan lemásolhatja a teljes, titkosított szöveget. Ennek ellenére, az eredeti címzettel ellentétben, a **támadó** (**intruder**) nem ismeri a visszakódoláshoz szükséges kulcsot, és így nem képes az üzenet egyszerű visszaalakítására. Néha a támadó nemcsak hallhatja a kommunikációs csatornát (passzív támadó), hanem a felvett üzeneteket később viszszajátszhatja, saját üzeneteit indíthatja útnak, vagy egyébként valódi üzeneteket változtathat meg, mielőtt azok a címzetthez megérkeznének (aktív támadó). A titkosírás megfejtésének mesterségét **kriptoanalízisnek** (**cryptoanalysis**) hívjuk. A titkosító eljárások kifejlesztésének tudománya (kriptográfia) és azok feltörése (kriptoanalízis) együttesen a **kriptológia** (**cryptology**) témakörét alkotják.

Gyakran hasznos lesz a továbbiakban, ha egységes jelölést vezetünk be a nyílt és titkosított üzenet, valamint a kulcs jelölésére. A $C=E_K(P)$ kifejezés azt jelenti, hogy a P nyílt szöveget a K kulcsot használva kódoljuk, és így a C titkosított üzenetet kapjuk. Hasonlóan a $P=D_K(C)$ a C üzenet nyílt szöveggé történő visszakódolását reprezentálja. Ezek után nyilvánvaló, hogy:

$$D_K(E_K(P))=P$$

A jelölés azt sejteti, hogy az E, valamint a D műveletek tulajdonképpen matematikai függvények, ami így is van. Az egyedüli trükk, hogy mindkettő kétparaméteres függvény, és az egyik paramétert (a kulcsot) alsó indexként tüntettük fel normál paraméter helyett, hogy élesen megkülönböztessük az üzenettől.

A kriptográfia alaptörvénye szerint feltételezzük a kriptoanalitikusról, hogy ismeri a kódoláshoz használt módszer algoritmusát. Más szavakkal, a kódfejtő pontosan tudja, hogy a titkosító (kódoló) eljárás (E) és a dekódoló eljárás (D) (lásd 8.2. ábra) hogyan működik. Egy új módszer kifejlesztéséhez, teszteléséhez és telepítéséhez szükséges erőfeszítések, melyeket akkor kell tennünk, ha módszerünket kitalálták, vagy úgy gondoljuk, hogy kitalálták, minden esetben célszerűtlenné tette, hogy a módszert titokban tartsuk, és feltételezzük azt, hogy valóban titok.

Itt lépnek be a képbe a kulcsok. A kulcs egy (aránylag) rövid karaktersorozatból áll, mely egyet választ ki a számos lehetséges kódolás közül. Az általános eljárással ellentétben, melyet legfeljebb néhány évente változtathatunk meg, a kulcsot olyan gyakran cserélhetjük, amilyen gyakran csak szükséges. Így az alapmodellünk egy stabil és mindenki által ismert eljárásból áll, melyet egy titkos és könnyen változtatható kulccsal paraméterezünk. Auguste Kerckhoff flamand hadikriptográfus 1883-ban megfogalmazott felismerése [Kerckhoff, 1883] után **Kerckhoff elvének** nevezzük azt az ötletet, mely szerint a kriptoanalitikus ismeri az algoritmust, és a titkosság kizárólag a kulcsokban rejlik. Azaz:

Kerckhoff elve: Minden algoritmusnak nyilvánosnak kell lennie; csak a kulcsok titkosak.

Az algoritmusok nyilvánosságát nem győzzük hangsúlyozni. A kereskedelemben az **ismeretlenség biztonsága** (**security by obscurity**) néven ismert fogalom, vagyis az, hogy az algoritmust megpróbáljuk titokban tartani, sosem vezet célra. Az eljárás publikálása viszont azt is lehetővé teszi, hogy a kriptográfus több elméleti szakemberrel ismertesse módszerét, akik aztán megpróbálják feltörni azt, hogy publikációkat írhassanak ravaszságuk demonstrálására. Ha számos szakértőnek 5 év próbálkozás után sem sikerül az algoritmus feltörése, akkor az már egészen megbízhatónak tekinthető.

Mivel az igazi titkosság a kulcsban rejlik, annak hossza alapvető fontosságú tervezési kérdés. Vegyünk egy egyszerű kombinációs zárat. Az általános alapelv az, hogy sorrendben számjegyeket adunk meg. Ezt mindenki tudja, de a kulcs titok. Két számjegy hosszúságú kulcs 100 lehetőséget rejt magában. Egy három számjegy hosszúságú esetén már 1000 lehetőség adódik, és hat számjegy alkalmazásakor elérjük az egymilliót. Minél hosszabb a kulcs, annál nagyobb **munkatényezővel** (**work factor**) kell számolnia a kódfejtőnek. Egy rendszer feltörésének munkaigénye – a kulcstér elemeinek végigpróbálásával – a kulcs hosszával exponenciálisan növekszik. A titkosság alapja egy erős (de nyilvános) algoritmus és egy hosszú kulcs. Ha meg szeretnéd akadályozni, hogy gyermeked elolvassa e-leveleidet, egy 64 bites kulcs megteszi. A kormányzati hivatalok sakkban tartásához azonban legalább 256 bites kulcs szükséges.

A kódfejtő nézőpontjából a kriptoanalízis problémája három területre osztható. Amikor számos titkos szöveggel, de egyetlen nyílt szöveggel sem rendelkezik, a **csak titkosított szövegalapú** (**chipertext only**) problémával áll szemben. Az újságok rejtvény oldalain megjelenő kriptogrammák ebbe a problémakörbe tartoznak. Amikor néhány nyílt szöveggel és azok titkosított párjaival rendelkezik a támadó, az **ismert nyílt szövegalapú** (**known plaintext**) problémát kell leküzdenie. Végül, amikor a kódtörőnek lehetősége van saját maga által választott szövegrészletek titkosított párjainak előállítására, a **választott nyílt szöveg típusú** (**chosen plaintext**) támadással találkozunk. Az újságok kriptogrammái triviális módon megfejthetők lennének, ha megengednénk az olyan típusú kérdéseket, hogy „Mi az ABCDEFGHIJKL titkosított párja?"

A kriptográfia területén kezdőnek számító emberek gyakran azt gondolják, hogy egy kód biztonságos, ha ellenáll a csak titkosított szöveg típusú támadásoknak. Ez a feltételezés nagyon naiv. A kódfejtő sok esetben jó becslést tud adni a nyílt szöveg bizonyos részeire. Bejelentkezéskor például sok számítógép kezdi azzal a válaszát, hogy „login: ". Néhány ilyen összetartozó nyílt szöveg – kódolt szövegpár birtokában – a kódtörő munkája jelentősen leegyszerűsödik. A biztonság eléréséhez tehát a kriptográfusnak óvatosnak kell lennie, és biztosítania kell, hogy a rendszer akkor is feltörhetetlen legyen, ha az ellenfél tetszőleges mennyiségű választott nyílt szöveget kódolhat.

A titkosító módszereket történelmileg két csoportba soroljuk: a helyettesítő típusú rejtjelezés és a keverő típusú rejtjelezés. Most ezeket tekintjük át röviden, hogy alapot adjanak a modern kriptográfia megértéséhez.

8.1.2. Helyettesítő titkosítók

Egy **helyettesítő titkosítóban** (**substitution cipher**) minden betű vagy betűcsoport egy másik betűvel vagy betűcsoporttal helyettesítődik a titkosság elérése érdekében. Az egyik legrégebbi ismert módszer a **Caesar-titkosító** (**Caesar cipher**), amely nevét Julius Caesarról kapta. A módszert alkalmazva az a betűből D lesz, a b-ből E, a c F-fé alakul ... és a z C-vé válik. Például az *attack* szó titkosított párja a *DWWDFN* szó lesz. Példáinkban a nyílt üzeneteket kisbetűvel, míg a kódolt változatokat csupa nagybetűvel fogjuk megadni.

A Caesar-titkosító kissé általánosabb változata megengedi, hogy a titkosított szöveg ábécéje k karakterrel legyen eltolva a korábbi fix 3 helyett. Ebben az esetben a k szám a ciklikus eltolást alkalmazó általános módszer mellett kulcsként viselkedik. A Caesar-titkosító talán becsaphatta a karthágóiakat, de azóta már senkit nem vezet félre.

Egy kicsit fejlettebb módszer, amikor a nyílt szöveg minden szimbólumához – legyen ez most az egyszerűség kedvéért pusztán 26 karakter – egy másik karaktert rendelünk. Vegyük a következő hozzárendelést:

Nyílt szöveg: a b c d e f g h i j k l m n o p q r s t u v w x y z
Kódolt szöveg: Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

A fenti általános módszert **egybetű-helyettesítéses titkosításnak (monoalphabetic substitution cipher)** nevezzük, ahol a kulcs az a 26 karakterből álló karaktersorozat, mely a nyílt szöveg teljes ábécéjének felel meg. A fenti kulcsot alkalmazva az *attack* nyílt üzenetet a kódoló a *QZZQEA* titkosított üzenetté transzformálja.

Első látásra ez a rendszer biztonságosnak tűnik, mivel a kódtörő – bár ismeri az általános módszert (betűhelyettesítés) – nem ismeri, hogy a szóba jöhető $26! \approx 4 \times 10^{26}$ lehetséges kulcs közül melyiket használtuk. Ellentétben a Caesar-kódolóval, minden egyes kulcs kipróbálása nem ígéretes megközelítés. Még ha egy kombináció kipróbálása 1 μs időt venne csupán igénybe, egy számítógépnek 10 000 évig tartana minden kulcs végigpróbálása.

Ennek ellenére meglepően kevés titkosított üzenet alapján a kód könnyűszerrel megfejthető. A támadás alapvetően a természetes nyelvek statisztikai tulajdonságait használja ki. Az angolban például az *e* a leggyakoribb karakter, sorrendben őt követik a *t*, *o*, *a*, *n*, *i* stb. A leggyakoribb betűpárok vagy más néven **betűkettősök (digrams)** a *th*, *in*, *er*, *re* és az *an*. A leggyakrabban előforduló három egymás mellett álló betű: **betűhármasok (trigrams)** a *the*, *ing*, *and* és az *ion*.

Az a kódfejtő, aki egybetű-helyettesítéses titkosító törésébe fog, a kódolt üzenetben szereplő betűk relatív gyakoriságának meghatározásával kezdi a munkáját. Ezek után próbálkozhat a leggyakoribb karakter *e*-vel való helyettesítésével, és a második leggyakoribbhoz a *t*-t rendelni. Ennek alapján megvizsgálhatja a betűhármasokat a *tXe* alakú mintákat keresve, mivel itt erős a gyanú, hogy az *X* helyén a *h* betű áll. Ehhez hasonlóan, ha a *thYt* minta gyakran felbukkan, az *Y* helyére célszerű az *a* betűt próbálni. Az így nyert információval nekiláthat az *aZW* betűhármasok felkutatásának, ami valószínűleg az *and* szót fogja lefedni. Gyakori karakterek, betűkettősök és betűhármasok megsejtésével és magánhangzó-mássalhangzó minták lehetséges kombinációinak ismeretével a kódtörő némi kísérletezéssel felépítheti a nyílt szöveget, betűről betűre.

Egy másik megközelítés szerint teljes szavakat, illetve kifejezéseket érdemes keresni. Példaképpen vizsgáljuk meg a következő titkosított üzenetet, amit egy könyvelőség küldött (ötös karaktercsoportokba szedve):

CTBMN	BYCTC	BTJDS	QXBNS	GSTJC	BTSWX	CTQTZ	CQVUJ
QJSGS	TJQZZ	MNQJS	VLNSX	VSZJU	JDSTS	JQUUS	JUBXJ
DSKSU	JSNTK	BGAQJ	ZBGYQ	TLCTZ	BNYBN	QJSW	

Egy ilyen jellegű cégtől érkező levélben gyakori szó lehet a *financial*. Felhasználva a szónak azt a tulajdonságát, hogy az *i* betű kétszer szerepel benne, közrefogva négy másikat. A titkos üzenetben ilyen távolságban levő karakterismétlődéseket fogunk keresni. A fenti szövegben ennek alapján 12 találatunk van: a 6., 15., 27., 31., 42., 48., 56., 66., 70., 71., 76. és 82. pozícióban. Ezek közül azonban csak kettőnél, a 31-nél és a 42-nél ismétlődik a következő karakter (ami az n-nek felelne meg) megfelelően. E kettő közül pedig csak a 31. pozícióban levőnél megfelelő a hipotetikus *a* karakterek ismétlődése. Ennek alapján erős a gyanúnk, hogy a 30. helyen a *financial* szó kezdődik. Ettől a ponttól kezdve, ha felhasználjuk a betűk előfordulásának relatív gyakoriságát, illetve az angol nyelv ezzel kapcsolatos statisztikai jellemzőit, továbbá közel teljes szavakat keresünk a befejezéshez, könnyű a kulcs megfejtése.

8.1.3. Keverő kódolók

A helyettesítő kódolók megtartották a nyílt szöveg karaktereinek sorrendjét, csak azokat más alakkal ruházták fel. Ezzel ellentétben a **keverő kódolók (transposition ciphers)** nem keresnek másik betűalakot, viszont az eredeti sorrendet átalakítják. A 8.3. ábra egy egyszerű keverő kódolót mutat be, az oszlopalapú keverőt. A titkosító kulcsként egy olyan szót vagy kifejezést használ, mely nem tartalmaz ismétlődő betűket. Példánk kulcsa a MEGABUCK lesz. A kulcs szerepe az oszlopok megszámozása lesz oly módon, hogy az első oszlopot az a kulcskarakter fogja kijelölni, amelyik az ábécében legelőször szerepel, a többi oszlop sorrendje is hasonlóképpen alakul. A nyílt üzenetet vízszintes sorokba írjuk, a titkosított üzenetet pedig függőlegesen olvassuk ki az oszlopok előbb megállapított sorrendjének megfelelően.

Egy keverő kódoló feltöréséhez a kódfejtőnek először rá kell jönnie, hogy egy ilyen típusú kóddal áll szemben. Az *E*, *T*, *A*, *O*, *I* karakterek gyakoriságát könnyen összevetheti a nem kódolt üzenetekben levő gyakoriságukkal. Ha ez nagy hasonlóságot mutat, akkor kétségkívül egy keverő kódolóval van dolga, mivel itt az egyes betűk önmaguknak felelnek meg, vagyis a gyakoriság eloszlása változatlan marad.

M	E	G	A	B	U	C	K
7	4	5	1	2	8	3	6
p	l	e	a	s	e	t	r
a	n	s	f	e	r	o	n
e	m	i	l	l	i	o	n
d	o	l	l	a	r	s	t
o	m	y	s	w	i	s	s
b	a	n	k	a	c	c	o
u	n	t	s	i	x	t	w
o	t	w	o	a	b	c	d

Nyílt szöveg

pleasetransferonemilliondollarsto
myswissbankaccountsixtwotwo

Titkosított üzenet

AFLLSKSOSELAWAIATOOSSCTCLNMOMANT
ESILYNTWRNNTSOWDPAEDOBUOERIRICXB

8.3. ábra. *A keverő kódoló*

A következő lépés az oszlopok számának meghatározása. Sok esetben egy – az üzenetben szereplő – szó vagy kifejezés kitalálható az üzenet környezetéből. Például elképzelhető, hogy a kódtörő gyanítja: a nyílt üzenetben a *milliondollars* kifejezés szerepel valahol. Az *MO, IL, LL, LA, IR* és *OS* betűkettősök előfordulásait keresve a titkos üzenetben ennek a kifejezésnek a deformált darabjait találhatja meg. A kódolt üzenetben szereplő *O* betű az *M*-et követi (fenti példánkban egymás alatt szerepelnek a 4. oszlopban), mivel a vizsgált kifejezésben a feltételezett kulcstávolságra vannak egymástól. Ha hét hosszúságú kulcsot használtunk volna, akkor az *MD, IO, LL, LL, IA, OR* és *NS* betűkettősök után kellene keresnünk. Valójában minden egyes kulcshosszhoz különböző betűkettős tartozik a titkosított üzenetben. A különböző lehetőségeket megvizsgálva a kódfejtő gyakran könnyen meghatározhatja a kulcshosszt.

A hátralevő feladat az oszlopok sorrendjének meghatározása. Ha az oszlopok száma (k) kicsi, az összes $k(k-1)$ db oszloppár megvizsgálható a betűkettősök előfordulási gyakorisága szempontjából. Annak a párnak a sorrendjét fogadjuk el, amely legjobban kielégíti az angol nyelv törvényszerűségeit. A megtalált pár után következő oszlopot további próbákkal határozhatjuk meg, azt a jelöltet választva, amelyik a legjobban megfelel a betűkettősök és betűhármasok szabályszerűségeinek. A megelőző oszlopot is hasonlóképpen határozzuk meg. Az eljárást a végleges oszlopsorrend meghatározásáig folytatjuk. Igen valószínű, hogy ezek után az eredeti üzenet felismerhetővé válik (például ha a *milloin* szót látjuk, világos, hogy hol a hiba).

Néhány keverő kódoló fix hosszúságú bemenetből ugyancsak kötött hosszúságú kimeneti blokkot készít. Ezeket a kódolókat egyszerűen leírhatjuk azzal a sorrenddel, ahogy a bemeneti karakterek a kimeneten megjelennek. A 8.3. ábrán bemutatott kódoló például egy 64 karakteres blokk-kódoló. Ennek kimenete: 4, 12, 20, 28, 36, 44, 52, 60, 5, 13, ..., 62. Más szavakkal: a negyedik bemeneti karakter (a) lesz a kimenet első karaktere, amit a bemenet 12. szimbóluma követ (f) és így tovább.

8.1.4. Egyszer használatos bitminta

Feltörhetetlen kódolót készíteni egyébként kifejezetten egyszerű; a módszer évtizedek óta ismert. Először is, válasszunk kulcsnak egy véletlen bitsorozatot! Ezután a kódolandó üzenetet szintén alakítsuk át bitsorozattá, mondjuk a karakterek ASCII-kódját felhasználva. Végül számoljuk ki a két sorozat KIZÁRÓ VAGY (XOR) művelettel adott eredményét bitről bitre. Az így kapott üzenet feltörhetetlen, mivel egy kellően hosszú üzenetmintában minden egyes karakter előfordulási valószínűsége azonos lesz, akárcsak a betűkettősöké és a betűhármasoké. Ez az **egyszer használatos bitminta (one-time pad)** néven ismert módszer ellenáll minden jelenlegi és jövőbeli támadásnak, függetlenül attól, hogy mennyi számítási kapacitással rendelkezik a támadó. Ennek oka az információelméletben keresendő: az üzenet ugyanis egyszerűen nem hordoz információt, mivel az összes lehetséges, adott hosszúságú nyílt üzenet egyformán valószínű lehet.

Az egyszer használatos bitminták alkalmazására a 8.4. ábra ad példát. Az „I love you" tartalmú 1-es üzenetet először 7 bites ASCII-formátumra alakítjuk. Ezután vesszük az egyszer használatos 1-es bitmintát, és KIZÁRÓ VAGY (XOR) kapcsolatba hozzuk az üzenettel, így kapjuk meg a titkosított szöveget. A kódtörő kipróbálhatja az összes lehetséges

1. üzenet	1001001	0100000	1101100	1101111	1110110	1100101	0100000	1111001	1101111	1110101	0101110
1. bitminta	1010010	1001011	1110010	1010101	1010010	1100011	0001011	0101010	1010111	1100110	0101011
Titkosított szöveg	0011011	1101011	0011110	0111010	0100100	0000110	0101011	1010011	0111000	0010011	0000101
2. bitminta	1011110	0000111	1101000	1010011	1010111	0100110	1000111	0111010	1001110	1110110	1110110
2. nyílt szöveg	1000101	1101100	1110110	1101001	1110011	0100000	1101100	1101001	1110110	1100101	1110011

8.4. ábra. *Titkosítás egyszer használatos bitminták segítségével. Bármilyen lehetséges nyílt szöveg előállítható a titkosított szövegből egy másik bitminta alkalmazásával*

bitmintát, hogy megnézze, melyik milyen nyílt szöveget eredményezne. Próbálkozhat például az ábrán megadott 2-es bitmintával, ami a 2-es nyílt szöveget adja, vagyis azt, hogy „Elvis lives", ami lehet elfogadható is, meg nem is (ez már túlmutat a könyv keretein). Valójában minden 11 karakteres ASCII nyílt szöveghez létezik egy azt előállító egyszer használatos bitminta. Ezt értjük azalatt, hogy nincs információ a titkosított szövegben, hiszen abból bármilyen, megfelelő hosszú üzenetet elő lehet állítani.

Az egyszer használatos bitminták elvileg nagyszerűek, de a gyakorlatban számos hátrányuk van. Először is, a kulcsot nem lehet megjegyezni, így mind a küldőnek, mind a fogadónak egy írott másolatot kell magánál tartania. Az írott kulcsok pedig egyértelműen nemkívánatosnak számítanak abban az esetben, ha van esély arra, hogy a támadók valamelyik másolatot megszerezzék. Ráadásul az elküldhető üzenet hosszát is korlátozza a rendelkezésre álló kulcs hossza. Ha egy titkos ügynök megüti a főnyereményt, és nagy mennyiségű adat birtokába jut, könnyen találhatja magát abban a helyzetben, hogy képtelen azt a központ felé továbbítani, mivel elfogyott a rendelkezésére álló kulcs. A módszer további problémája, hogy rendkívül érzékeny az elveszett vagy közbeékelődött karakterekre. Ha a küldő és a fogadó egy ponton kiesnek a szinkronból, akkor onnantól kezdve minden üzenet zagyvaságnak tűnik számukra.

A számítógépek megjelenésével az egyszer használatos bitminták mégis hasznosnak bizonyulhatnak egyes alkalmazásokban. A kulcs forrása lehet egy speciális DVD, mely több gigabájtnyi információt tárol. Ha ezt egy közönséges DVD-tokba rejtjük, és az elejére még pár percnyi mozgóképet is rögzítünk, akkor senki nem fog gyanakodni. Gigabites hálózati sebességek mellett persze kissé fárasztó lehet 30 másodpercenként új DVD-lemezt berakni. Mindennek tetejébe a DVD-ket személyesen kell elvinni a küldőtől a fogadóhoz, még azelőtt, hogy bármilyen üzenetet elküldenénk. Ez azonban nagyban csökkenti a módszer gyakorlati használhatóságát.

Kvantumkriptográfia

Érdekes módon talán mégis van egy megoldás az egyszer használatos bitminta hálózaton történő átvitelére. Ez pedig egy nagyon valószínűtlen forrásból származik: a kvantummechanika területéről. A munka még kísérleti fázisban van, de a kezdeti tesztek ígéretesek. Ha a módszert sikerül tökéletesíteni és hatékonyabbá tenni, akkor gyakorlatilag minden kriptográfia egyszer használatos bitmintákon fog alapulni, mivel azok bizonyíthatóan biztonságosak. A következőkben röviden elmagyarázzuk, hogyan működik a **kvantumkriptográfia (quantum cryptography)** módszere. Konkrétan a **BB84**

nevű protokollt ismertetjük, mely alkotóiról és közlésének évéről kapta nevét [Bennet és Brassard, 1984].

Az Aliz nevű felhasználó szeretne létrehozni egy, a Bob nevű felhasználóval közös, egyszer használatos bitmintát. Aliz és Bob történetünk két **főszereplője** (**principals**). Bob lehet például egy bankár, akivel Aliz üzletelni szeretne. Az elmúlt évtizedben gyakorlatilag az összes kriptográfiával foglalkozó cikkben és könyvben „Aliz" és „Bob" volt a főszereplő. A kriptográfusok szeretik a hagyományokat. Ha mi most „Andyt" és „Barbarát" használnánk főszereplőként, akkor senki nem hinne el a fejezetből egy szót sem. Maradjon hát Aliz és Bob.

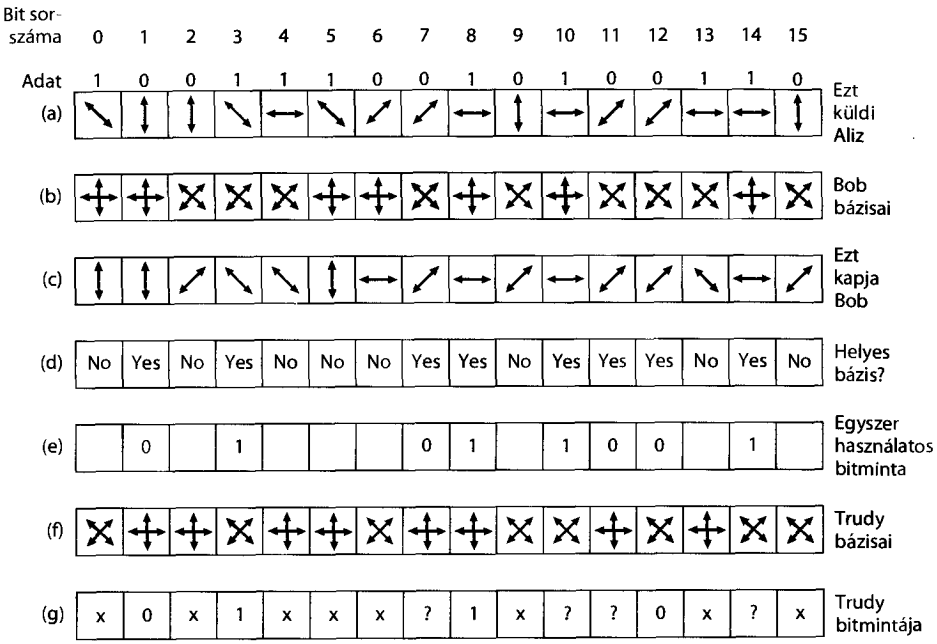
Ha Aliznak és Bobnak lenne közös egyszer használatos bitmintájuk, akkor azzal biztonságosan kommunikálhatnának. A kérdés csak az: hogyan állapodhatnak meg egy ilyen bitmintában anélkül, hogy előzőleg DVD-ket cseréltek volna? Feltehetjük, hogy Aliz és Bob egy fényvezető szál két végénél helyezkedik el, melyen keresztül fényimpulzusokat küldhetnek és fogadhatnak. Van azonban egy elszánt támadó is, Trudy, aki megvágja az üvegszálat, hogy beillesszen abba egy aktív csatolót. Trudy olvashatja a biteket, és hamis üzeneteket is küldhet mindkét irányban. Aliz és Bob helyzete immár reménytelennek tűnhet, de a kvantumkriptográfia új fényt vethet mindenre.

A kvantumkriptográfia azon a tényen alapul, hogy a fény bizonyos sajátos tulajdonsággal bíró, **fotonnak** (**photon**) nevezett kis részecskékben érkezik. A fényt ezenfelül polárszűrők segítségével polarizálni is lehet, amint azt a napszemüveget viselők és a fényképészek is jól tudják. Ha egy fénysugár (vagyis egy fotonfolyam) egy polárszűrőn halad keresztül, akkor az összes kilépő foton a szűrő tengelyének megfelelő irányban polarizált lesz (például vízszintesen). Ha ezt a sugarat most egy második polárszűrőn visszük keresztül, akkor az abból kilépő fény erőssége arányos lesz a szűrők tengelyei által bezárt szög koszinuszának négyzetével. Ha a tengelyek merőlegesek egymásra, akkor egyetlen foton sem jut át. A két szűrő abszolút helyzete nem lényeges, csak a tengelyeik által bezárt szög számít.

Az egyszer használatos bitminta előállításához Aliznak két polárszűrőkészletre van szüksége. Az első készlet egy függőleges és egy vízszintes szűrőből áll, ezt **egyenes vonalú** (**rectilinear**) **bázisnak** nevezzük. A bázis tulajdonképpen csak egy koordinátarendszer. A másik szűrőkészlet ugyanilyen, de 45 fokkal el van forgatva, vagyis az egyik szűrő tengelye a bal alsó sarokból a jobb felső sarok, a másik szűrő tengelye pedig a bal felső sarok felől a jobb alsó sarok irányába mutat. Ezt a készletet **átlós** (**diagonal**) **bázisnak** nevezzük. Aliznak tehát két bázisa van, melyeket tetszése szerint, gyorsan be tud illeszteni a sugara útjába. A valóságban Aliznak nem négy különálló szűrője van, hanem egy olyan kristálya, melynek polarizációját nagy sebességgel, elektronikus úton lehet a négy lehetséges irány közül bármelyikre átkapcsolni. Bob ugyanilyen felszereléssel rendelkezik. Az a tény, hogy mindkettőjüknek két bázisa van, alapvető fontosságú a kvantumkriptográfiában.

Aliz minden bázis esetében hozzárendeli az egyik irányhoz a 0-t, a másikhoz pedig az 1-et. Az alábbi példában feltesszük, hogy a függőlegest választja 0-nak és a vízszintest 1-nek. Ettől függetlenül a bal alsó–jobb felső irányhoz is hozzárendeli a 0-t és a bal felső–jobb alsó irányhoz az 1-et. Választásait nyílt szövegként küldi el Bobnak.

Aliz ezután kiválaszt egy egyszer használatos bitmintát, például egy véletlenszám-generátor segítségével (ez már egy önmagában is bonyolult kérdés). A mintát bitről bitre



8.5. ábra. *Példa a kvantumkriptográfiára*

átviszi Bobnak, és eközben minden egyes bitnél véletlenszerűen használja valamelyiket a két bázisa közül. A bitet úgy küldi el, hogy a fotonágyúja mindig az adott bitnél használt bázisnak megfelelően polarizált fotont bocsát ki. Választhatja például rendre az átlós, linearizált, linearizált, átlós, linearizált stb. bázisokat. Az 1001110010100110 ezen bázisokkal való elküldéséhez tehát a 8.5.(a) ábrán jelölt fotonokat kell kiadnia. A bitminta és a bázissorozatok együttesen minden egyes bit esetében egyértelműen meghatározzák, hogy milyen polarizációt kell használni. Az egyszerre egy foton segítségével elküldött biteket **kubiteknek** (**qubits**) nevezik.

Bob nem tudja, hogy milyen bázisokat kell használnia, ezért minden beérkező fotonhoz egyszerűen véletlenszerűen választ egyet, és azt használja, ahogy azt a 8.5.(b) ábra mutatja. Ha eltalálja a helyes bázist, megkapja a helyes bitet. Ha rossz bázist választ, akkor egy véletlenszerű bitet kap, mert ha egy foton egy, a saját polarizációjához képest 45 fokban polarizált szűrőn halad át, akkor egyforma valószínűséggel, véletlenszerűen fog átugrani a szűrővel megegyező, vagy az arra merőleges polarizációra. A fotonoknak ez a tulajdonsága alapvetőnek számít a kvantummechanikában. Egyes bitek tehát helyesek lesznek, mások véletlenszerűek, de Bob nem tudja, hogy éppen melyik melyik. Bob eredményei a 8.5.(c) ábrán láthatók.

Hogyan tudja meg Bob, hogy mely bázisokat találta el és melyeket nem? Úgy, hogy egy nyílt szövegben egyszerűen közli Alizzal, hogy milyen bázist használt az egyes bitekhez, Aliz pedig szintén egy nyílt szövegben elmondja neki, hogy melyiket találta el, és melyiket nem, amint azt a 8.5.(d) ábra mutatja. Ebből az információból mindketten felépíthetnek egy bitfüzért a helyes találgatásokról, amint ez a 8.5.(e) ábrán is látszik. Ez

a bitfüzér átlagosan fele olyan hosszú lesz, mint az eredeti bitfüzér, de mivel mindkét fél ismeri, felhasználhatják egyszer használatos bitmintaként. Aliznak mindössze annyit kell tennie, hogy a kívántnál valamivel több mint kétszer hosszabb bitfüzért ad le, így ő is és Bob is a kívánt hosszúságú bitmintát kapják. A probléma tehát meg van oldva.

De várjunk csak egy percet! Trudyról megfeledkeztünk. Tegyük fel, hogy ő nagyon kíváncsi a szereplők mondanivalójára, ezért elvágja az üvegszálat, és beilleszti a saját vevőjét és adóját. Balszerencséjére ő sem tudja, hogy melyik fotonhoz melyik bázist kell használni. Akárcsak Bob, ő is mindössze annyit tehet, hogy véletlenszerűen választ ki egyet minden fotonhoz. Választásaira a 8.5.(f) ábra mutat egy lehetséges példát. Amikor Bob később (nyílt szövegben) jelenti, hogy milyen bázisokat használt, és Aliz (szintén nyílt szövegben) elmondja neki, hogy mely tippek voltak helyesek, Trudy tudni fogja, hogy mely biteket vette jól és melyeket rosszul. Az ábrán eltalálta a 0., 1., 2., 3., 4., 6., 8., 12. és 13. bitet. Aliz 8.5.(d) ábrán látható válaszából azonban tudja, hogy csak az 1., 3., 7., 8., 10., 11., 12. és 14. bitek részei az egyszer használatos bitmintának. Ezek közül négy bitnél (1., 3., 8. és 12.) ő is helyesen tippelt, így rendelkezik a helyes bittel. A maradék négynél (7., 10., 11. és 14.) azonban rossz volt a tippje, és nem ismeri az akkor átvitt bitet. Bob tehát tudja már, hogy a bitminta 01011001-gyel kezdődik [8.5.(e) ábra], Trudynak viszont csak annyi van meg, hogy 01?1??0? [8.5.(g) ábra].

Aliz és Bob természetesen tudják, hogy Trudy megszerezhette a bitmintájuk egy részét, szeretnék tehát csökkenteni a Trudy rendelkezésére álló információ mennyiségét. Ezt egy transzformáció segítségével érhetik el. Feloszthatják például a bitmintájukat 1024 bites blokkokra, majd a blokkokat négyzetre emelhetik, és az így kapott 2048 bites számok egymás után fűzését használhatják bitmintának. Ha Trudy csak részlegesen ismeri az átvitt bitfüzért, akkor semmilyen úton sem képes előállítani annak négyzetét, vagyis nem ért el semmit. Azt a transzformációt, mely úgy alakítja át az eredeti bitmintát, hogy azzal csökkentse Trudy tudását, **személyiségi jogok erősítésének** (**privacy amplification**) nevezzük. A gyakorlatban a négyzetre emelés helyett olyan bonyolult átalakításokat használnak, melyekben minden kimeneti bit függ minden bemeneti bittől.

Szegény Trudy! Nemcsak hogy fogalma sincs a bitmintáról, de a jelenléte sem marad titokban. Továbbítania kell ugyanis minden vett bitet Bobnak, hogy elhitesse vele, hogy Alizzal beszél. A baj csak az, hogy ő legfeljebb a vett kubitet tudja leadni, azzal a polarizációval, mellyel ő vette azt, így az esetek kb. felében tévedni fog, ezzel pedig sok hibát okoz Bob bitmintájában.

Amikor Aliz végül megkezdi az adatküldést, akkor egy hathatós hibajavító kódot alkalmaz. Bob szemében egy bithiba az egyszer használatos bitmintában ugyanolyan, mint egy 1 bites átviteli hiba, hiszen mindkét esetben rossz bitet vesz. Ha megfelelő mértékű hibajavítást alkalmaznak, akkor a hibák ellenére is vissza tudja állítani az eredeti üzenetet, és közben könnyen megszámolhatja azt is, hogy hány hibát javított ki. Ha ez a szám jóval több, mint a berendezés várható hibaaránya, akkor tudni fogja, hogy Trudy megcsapolta a vonalat, és ennek megfelelően cselekedhet (például szólhat Aliznak, hogy kapcsoljon át egy rádiós csatornára, vagy hívja a rendőrséget stb.). Ha Trudy képes lenne klónozni a fotonokat, vagyis lenne egy fotonja, amit megvizsgálhatna és egy ugyanolyan, amit elküldhetne Bobnak, akkor elkerülhetné a lebukást, de jelenleg nem ismeretes olyan módszer, amivel tökéletesen le lehetne másolni egy fotont. De még ha

Trudy képes lenne is fotonokat klónozni, az sem csökkentené a kvantumkriptográfia értékét az egyszer használatos bitmintákról való megegyezésben.

Jóllehet, a kvantumkriptográfia működőképességét egy 60 km-es üvegszálon már bebizonyították, a hozzá szükséges felszerelés azonban rendkívül bonyolult és drága. Az ötlet ennek ellenére ígéretes. A kvantumkriptográfiáról további információt Mullins [2002] művében találhatunk.

8.1.5. Két alapvető kriptográfiai elv

Bár a következőkben számos különböző kriptográfiai rendszert fogunk megismerni, ezek mégis megegyeznek abban, hogy két fontos elv szolgál alapjukként, amelyeket fontos megértenünk. Figyeljünk tehát oda, mert ha megszegjük, saját magunknak okozunk kárt.

Redundancia

Az első alapelv szerint a titkosított üzeneteknek némi redundanciát kell hordozniuk, vagyis olyan információt, ami nem szükséges az üzenet megértéséhez. Egy példán keresztül beláthatjuk ennek fontosságát. Vegyünk egy olyan céget, amelytől levélben rendelhetünk valamit. Legyen ez a Korszerű Játszótéri Bútorok (KJB) cég, mintegy 60 000 termékkel. Képzeljük el, hogy igen hatékonyan működnek, és a KJB programozói úgy döntenek, hogy minden megrendelőlevél alakja a következőképpen nézzen ki: 16 bájt hosszú ügyfél-azonosító, majd egy 3 bájtos adatmező (1 bájt a mennyiségnek, 2 bájt pedig a termékkód számára). Az utolsó 3 bájtot egy nagyon hosszú kulccsal titkosítják, amit csak a megrendelő és a KJB programozói ismernek.

Első látásra biztonságosnak tűnik a fenti módszer, és bizonyos értelemben az is, mivel passzív támadók nem tudják megfejteni az üzeneteket. Sajnálatos módon egy súlyos hiányossága van, ami használhatatlanná teszi. Tegyük fel, hogy egy frissen elbocsátott alkalmazott bosszút szeretne állni a cégen. Mielőtt távozna a vállalattól, magával viszi az ügyfelek listáját, vagy annak egy részét. A következő éjszaka elkészít egy programot, mely fiktív rendeléseket generál valódi ügyfél-azonosítókkal. Mivel nem ismeri a titkos kulcsokat, az utolsó három bájtot véletlenszerűen tölti fel, majd rendelések százait juttatja el a KJB -hez.

Amikor az üzenetek megérkeznek, a KJB számítógépei az ügyfelekhez rendelt kulcsok segítségével visszafejtik az üzenetet. A KJB szerencsétlenségére, mivel majdnem minden 3 bájtos kombináció értelmes, a számítógépek elkezdik a rendelések teljesítését. Bár furcsának tűnhet, hogy a vásárló 837 gyermekhintát és 540 homokozót rendelt, a programok csak annyit „gondolhatnak", hogy minden bizonnyal játszótér-hálózatot szeretne nyitni az ügyfél. Ily módon egy aktív támadó (az elbocsátott alkalmazott) nagy kárt okozhat még úgy is, hogy fogalma sincs a programja által generált üzenetek tartalmáról.

A veszély jelentősen csökkenthető, ha az üzenetekbe redundanciát csempészünk. Ha például a rendeléssel kapcsolatos részt 12 bájt hosszú mezőn tároljuk, az első 9 karakter helyére nullákat írva, a fenti támadás már nem jár sikerrel, mivel a volt alkalmazott nem képes valódinak tűnő üzenetek nagyszámú előállítására. A történet mondanivaló-

ja, hogy minden üzenetnek számottevő redundanciát kell hordoznia ahhoz, hogy aktív támadók ne áraszthassanak el bennünket valódinak tűnő hamis üzenettel.

A redundancia növelésével azonban a kódfejtőnek egyre könnyebb dolga lesz az üzenet feltörésekor. Tegyük fel, hogy a levélen keresztüli rendelés versenyképes üzletággá válik, és a KJB fő riválisa, a Játszótér Fejlesztő és Kivitelező (JFK) cég mindent megadna azért, hogy megtudja, hány homokozót ad el a KJB. Így aztán megcsapolja a KJB telefonvonalát. Az eredeti 3 bájtos sémát használva a kriptoanalízis reménytelen, mivel egy kulcs megsejtése után a kódtörő nem ellenőrizheti kellőképpen, hogy valóban kulcsot talált-e, mivel minden üzenet gyakorlatilag érvényes. A módosított 12 bájtos módszert alkalmazva azonban a kriptoanalízist végző támadónak könnyű dolga van egy kulcs ellenőrzésekor. Ezek szerint:

A kriptográfia 1. alapelve: Az üzeneteknek valamilyen redundanciát kell tartalmazniuk.

Más szóval, egy üzenet visszafejtésekor a vevőnek egy egyszerű vizsgálattal vagy egy kisebb számítással meg kell tudnia állapítani, hogy az vajon érvényes-e. Erre a redundanciára azért van szükség, hogy az aktív támadó ne küldhessen el mindenféle zagyvaságot, becsapva ezzel a vevőt, aki a zagyvaságot visszafejtve az így kapott „nyílt szöveg" alapján cselekedne. Másfelől viszont a nagyobb redundancia épp a passzív támadóknak kedvez, vagyis itt némi ellentét húzódik meg. Továbbá, a redundanciát sosem szabad az üzenet elején vagy végén, csupa nulla karakter beiktatásával elérni, mivel néhány kriptográfiai algoritmus az ilyen sorozatokból előre megjósolható kimenetet gyárt, ami leegyszerűsíti a kódfejtő munkáját. Sokkal jobb egy CRC-polinom, mint egy 0-kból álló sorozat, mivel a vevő azt egyszerűen ellenőrizheti, a kódtörőnek mégis több munkát jelent. Ennél is jobb a kriptografikus hash, melynek ötletét később fogjuk tanulmányozni. Pillanatnyilag gondoljunk arra, hogy ez olyan, mint egy jobb CRC.

Egy pillanatra visszatérve a kvantumkriptográfiára láthatjuk, hogy a redundancia ott is szerepet játszik. Bob egyszer használatos bitmintájában egyes bitek hibásak lesznek, köszönhetően annak, hogy Trudy elfogja a fotonokat. Bobnak tehát bizonyos redundanciára lesz szüksége a beérkező üzenetekben, hogy észrevegye a hibák meglétét. A redundancia egyik nagyon durva formája az, ha az üzenetet egyszerűen megismétlik. Ha a két másolat nem azonos, akkor Bob tudja, hogy vagy az üvegszál nagyon zajos, vagy valaki beavatkozott az átvitelbe. A kétszeres küldés persze túlzás: egy Hamming- vagy Reed–Solomon-kód sokkal hatékonyabb módot biztosít a hibajelzésre és a hibajavításra. Annak mindenesetre nyilvánvalónak kell lennie, hogy némi redundanciára szükség van ahhoz, hogy meg lehessen különböztetni egy érvényes üzenetet egy érvénytelentől, különösen egy aktív támadó jelenlétében.

Frissesség

A második kriptográfiai alapelv szerint valamilyen módon biztosítani kell, hogy minden vett üzenetnél ellenőrizni lehessen az üzenet frissességét, vagyis azt, hogy csak nemrég küldték-e el. Erre azért van szükség, hogy egy aktív támadó ne játszhasson vissza régi üzeneteket. Ha nem tennénk semmi óvintézkedést, akkor az elbocsátott alkalmazottunk

lehallgathatná a KJB telefonvonalát, és sorra megismételhetné az előzőleg elküldött érvényes üzeneteket. Még egyszer megfogalmazva:

A kriptográfia 2. alapelve: Kell egy módszer az ismétléses támadások meghiúsítására.

Egy lehetséges intézkedés például, ha minden üzenetet időbélyeggel látunk el, mely mondjuk 10 másodpercig érvényes. A vevő ezután egyszerűen megőrzi az üzeneteket kb. 10 másodpercig, és összehasonlítja az újonnan érkezetteket a régebbiekkel, hogy kiszűrhesse a másodpéldányokat. A 10 másodpercnél régebbi üzeneteket ki lehet dobni, mivel a több mint 10 másodperccel később elküldött ismétléseket rögtön visszautasítják azzal, hogy túl régiek. Az időbélyegek mellett más lehetőségek is vannak, ezeket a későbbiekben tekintjük át.

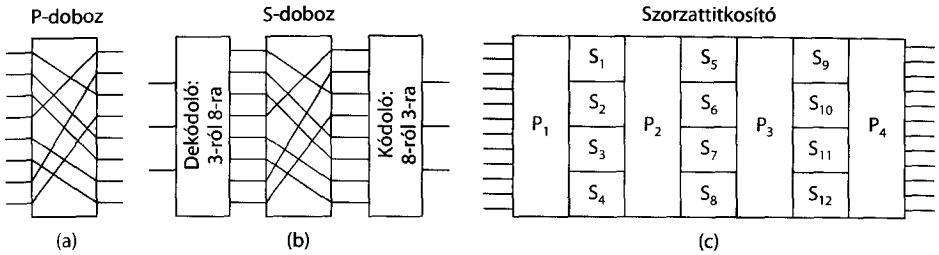
8.2. Szimmetrikus kulcsú algoritmusok

A modern kriptográfia ugyanazokat az alapötleteket használja, mint a hagyományos titkosítás: helyettesítést és keverést, de ma már máson van a hangsúly. A tradicionális kódkészítők egyszerű algoritmusokat használtak. Ma ennek az ellenkezője igaz: a cél olyan bonyolult és szövevényes algoritmusok megalkotása, hogy a kódfejtő még hatalmas mennyiségű, általa választott üzenet kódolt megfelelőjének birtokában se legyen képes abból bármit is kibogozni a kulcs nélkül.

A fejezetben elsőként tárgyalt algoritmuscsoport az úgynevezett **szimmetrikus kulcsú algoritmusok** (**symmetric-key algorithms**) csoportja lesz. Ezek onnan kapták a nevüket, hogy ugyanazt a kulcsot használták a titkosításhoz és a visszafejtéshez is. A 8.2. ábra a szimmetrikus kulcsú algoritmus használatát szemlélteti. Ezúttal konkrétan a **blokk-kódolókkal** (**block cipher**) fogunk foglalkozni, melyek egy n bites blokkban kapják a nyílt szöveget, és azt a kulcs segítségével egy n bites blokkba alakítják át titkosított szöveggé.

A kriptográfiai algoritmusokat (a sebesség érdekében) hardveresen és (a rugalmasság miatt) szoftveresen is meg lehet valósítani. Bár tárgyalásunk legnagyobb része a tényleges megvalósítástól független algoritmusokkal és protokollokkal foglalkozik, mégis érdekes lehet pár szót ejteni a kriptografikus hardverek felépítéséről. A helyettesítés és a keverés egyszerű elektronikus áramkörökkel is megvalósítható. A 8.6.(a) ábra egy **P-doboznak** (a P a permutációt jelöli) nevezett eszközt mutat, mellyel 8 bites bemeneti adatokon lehet végrehajtani a keverést. Ha a 8 bitet fentről lefelé sorszámmal látjuk el (01234567), akkor ennek a P-doboznak a kimenete a 36071245 lesz. Megfelelő belső huzalozással a P-doboz tetszőleges keverést elvégezhet, mindezt gyakorlatilag fénysebességgel, hiszen számítások nincsenek, csak jelterjedés van. Ez a tervezés Kerckhoff elvét követi: a támadó tudja, hogy az általános módszer a bitek permutálása. Amit nem tud, az az, hogy melyik bit hova kerül, hiszen ez maga a kulcs.

A helyettesítést ún. **S-dobozok** (az S a substitution szóból származik) végzik, ahogy azt a 8.6.(b) ábra mutatja. A fenti eszköz 3 bites nyílt üzenetekből ugyancsak 3 bites, titkosított szöveget gyárt. A 3 bitnyi bemenet egy vonalat választ ki az első fokozat nyolc



8.6. ábra. *A szorzattitkosítók alapvető elemei. (a) P-doboz. (b) S-doboz. (c) Szorzattitkosító*

kimenetéből, amin 1-es értéket állít be, a többin 0-t. A második lépcsőben egy P-doboz található. A harmadik fokozat a kiválasztott aktív bemeneti vonala alapján újra bináris alakú kódot generál. A bemutatott huzalozással, ha a bemenetre a 01234567 sorozatot adjuk, a kimeneten a 24506713 szekvencia fog megjelenni. Más szavakkal a 0-t 2-vel helyettesíti, az 1-et a 4-gyel stb. Megint igaz, hogy az S-dobozban található P-doboz megfelelő kialakításával bármilyen helyettesítést elvégezhetünk. Egy ilyen eszközt ráadásul a hardverbe is be lehet építeni, és így nagy sebességet lehet elérni, mivel a kódolók és dekódolók csak egy- vagy kétkapunyi (nanomásodpercnél kisebb) késleltetést jelentenek, és a jelterjedés ideje a P-dobozon keresztül jóval 1 pikomásodperc alatt maradhat.

Ezeknek az építőköveknek akkor mutatkozik meg az igazi erejük, amikor sorba kapcsoljuk őket, így létrehozva a 8.6.(c) ábrán látható **szorzat típusú titkosítót** (**product cipher**). Ennél a példánál első lépésben 12 bemeneti vonalat cserélünk fel. Elméletileg a második fokozatban elhelyezhetnénk egy olyan S-dobozt, amely egy 12 bites bemenethez egy 12 bites kimenetet rendelne. Ehhez azonban 212 = 4096 egymást keresztező vezetékre lenne szükség az eszköz belsejében. Ehelyett a bemenetet 4 db 3 bites csoportra bontjuk, mindegyiket egymástól függetlenül helyettesítünk. Bár ez a megoldás már nem annyira általános, de még mindig hatékony. Kellően nagyszámú fokozatot használva a kódolóban, a kapott kimenet rendkívül bonyolult függvénye lesz a bemenetnek.

A *k* bites bemenetből *k* bites kimenetet előállító szorzattitkosítók nagyon elterjedtek. A *k* értéke jellemzően 64 és 256 között mozog. A hardveres megvalósítások a 8.6.(c) ábra 7 fokozatával szemben rendszerint legalább 18 fizikai fokozatot tartalmaznak. A szoftveres megvalósítást egy legalább 8 iterációt tartalmazó ciklussal programozzák be, ahol minden iteráció egy S-doboz típusú helyettesítést hajt végre a 64–256 bites adatblokk egy alblokkján, melyet egy olyan permutáció követ, amely az S-dobozok kimenetét keveri össze. A művelet elején és végén gyakran van még egy speciális permutáció is. A szakirodalomban ezeket az iterációkat **köröknek** (**rounds**) nevezik.

8.2.1. DES – az adattitkosító szabvány

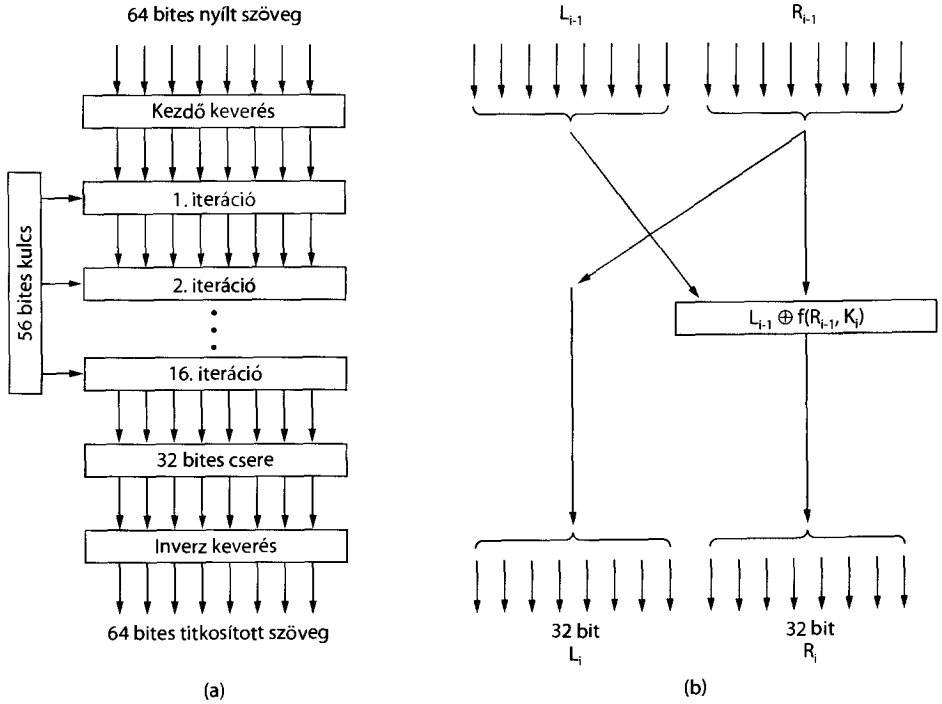
1977 januárjában az amerikai kormányzat egy az IBM által kifejlesztett szorzat típusú kódolót fogadott el szabványként a nem bizalmas információ számára. A kódoló, amit **DES** (**Data Encryption Standard** – **adattitkosító szabvány**) névre kereszteltek, az iparban is széles körben elterjedt a biztonsági termékek piacán. Eredeti formájában ma

már nem tekinthető biztonságosnak, de némi kiegészítéssel ma is használható. A DES működését tekintjük át a következőkben.

A DES vázlatos működése a 8.7.(a) ábrán követhető nyomon. A nyílt szöveget 64 bites blokkonként kódoljuk, ami során szintén 64 bites titkos üzeneteket kapunk. Az algoritmus, melynek paraméteréül egy 56 bites kulcs szolgál, 19 különálló fokozatból épül fel. Az első lépés egy kulcsfüggetlen keverés a 64 bites bemeneten. Az utolsó lépés ennek pontosan az inverz művelete. Az utolsót megelőző lépésben az első 32 bites részt felcseréljük a hátsó 32 bites résszel. A maradék 16 lépés működése ehhez hasonló, de mindegyik paraméteréül a kulcs különböző függvényekkel képzett értéke szolgál. Az algoritmus lehetővé teszi, hogy a dekódolást ugyanazzal a kulccsal végezhessük, mint a kódolást. Egyedül a lépések sorrendjét kell megfordítanunk.

A közbenső lépések egyikét mutatja részletesebben a 8.7.(b) ábra. Mindegyik ilyen fokozat két 32 bites bemenetből ugyancsak kettő 32 bites kimenetet produkál. A bal oldali kimenet egyszerűen a jobb oldali bemenet másolata. A jobb oldali kimenetet KIZÁRÓ VAGY (XOR) művelettel kapjuk, amit egyrészt a bal oldali bemenet, másrészt a jobb oldali bemenet, valamint a fokozathoz tartozó kulcsérték (K_i) alapján egy adott függvénnyel képzett érték között végzünk el. Az algoritmus szövevényessége ebben az adott függvényben rejlik.

A függvény négy lépésből áll, melyeket egymás után végzünk el. Első lépésben egy 48 bites számot képzünk (E) a 32 bites jobb oldal (R_{i-1}) kiterjesztésével, amit egy rögzített



8.7. ábra. *Az adattitkosító szabvány. (a) Általános vázlat. (b) Egy iteráció részletesebben. A bekarikázott + a* KIZÁRÓ VAGY *műveletet jelenti*

keverés és másolás segítségével kapunk. A második lépésben az E, illetve a (K_i) értékek között KIZÁRÓ VAGY műveletet hajtunk végre. Az így kapott eredményt 8 db 6 bites csoportra osztjuk, amiket aztán különböző S-dobozokba pumpálunk. Egy ilyen 64 lehetőséget magában hordozó inputból az egyes S-dobozok 4 bites kimenetet generálnak. Végül az így nyert 8 × 4 bitet egy P-dobozon engedjük keresztül.

Mind a 16 iterációs lépésben különböző kulcsokat használunk. Az algoritmus kezdetekor egy 56 bites keverést végzünk a kulcson. Mindegyik lépés megkezdése előtt a kulcsot két 28 bites részre particionáljuk, mindegyiket az iteráció sorszámának megfelelő számú bittel balra forgatva. A K_i-t ezekből a szegmensekből egy újabb 56 bites keverés során kapjuk meg. Az 56 bites kulcs egy 48 bites részét minden fokozatban még külön permutáljuk.

A DES megerősítésére olykor egy úgynevezett **fehérítés** (**whitening**) eljárást is használnak. Ez abból áll, hogy a DES alkalmazása előtt minden egyes nyílt szöveg blokkot KIZÁRÓ VAGY (XOR) kapcsolatba hoznak egy véletlenszerű 64 bites kulccsal, majd a titkosított szöveget az átvitel előtt egy másik 64 bites kulccsal is KIZÁRÓ VAGY kapcsolatba hozzák. A fehérítés egyszerűen eltávolítható ezen műveletek fordítottjainak elvégzésével (feltéve, hogy a vevő is rendelkezik a két fehérítő kulccsal). Az eljárás tulajdonképpen a kulcs hosszát növeli meg, sokkal időigényesebbé téve ezzel a teljes kulcstérben való keresést. Figyeljük meg, hogy minden egyes blokkra ugyanazt a fehérítő kulcsot alkalmazzuk (vagyis csak egy fehérítő kulcs van).

A DES-t megszületése óta viták övezik. Története egy, az IBM által készített és szabadalmaztatott kódolóval kezdődik, mely a Lucifer névre hallgatott. Az IBM által kifejlesztett titkosító azonban nem 56 bites, hanem 128 bites kulcsokkal dolgozott. Amikor az amerikai kormányzat kódolószabványt szeretett volna elfogadtatni a nem bizalmas információk titkosítására, „meghívta" az IBM-et, hogy „megvitassa" a problémát az NSA-val, a kormányzat kódfejtő szervezetével, mely a világ legnagyobb, matematikusokat és kriptográfiai szakembereket foglalkoztató cége. Az NSA körül annyi titok leng, hogy az iparban közszájon forog a következő tréfa:

Kérdés: Mit jelent az NSA rövidítés?
Válasz: Nincs Semmiféle Alakulat.

A viccet félretéve, az NSA a National Security Agency (Nemzeti Biztonsági Ügynökség) rövidítése.

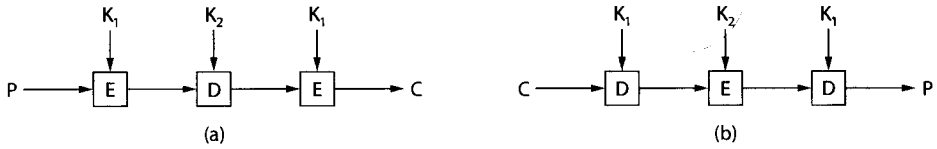
A tárgyalások után az IBM a 128 bitről 56 bitre csökkentette a kulcsok hosszát, és úgy döntött, hogy titokban tartja a DES tervezésével kapcsolatos információt. Sokan feltételezik, hogy a kulcshossz ilyetén való csökkentése azért történt, hogy az NSA még feltörhesse a DES-t, de bármely kisebb költségvetésű szervezet erre képtelen legyen. A tervezéssel kapcsolatos titkok pedig némelyekben azt a gyanút keltik, hogy kiskapu van elrejtve az algoritmusban, amelynek segítségével az NSA jóval könnyebben törheti fel a DES-kódot. A DES-t övező nyugtalanság csak fokozódott, amikor egy NSA-alkalmazott tapintatosan felkérte az IEEE-t, hogy ne tartson meg egy tervezett kriptográfiai konferenciát. Az NSA persze mindent tagadott.

1977-ben két stanfordi, kriptográfiával foglalkozó szakember, Diffie és Hellman [1977] egy, a DES feltörésére alkalmas gép terveit dolgozta ki, mely körülbelül 20 millió

dollárból megépíthető lett volna. Egy viszonylag kicsi üzenetdarabka és annak kódolt párja alapján a gép megtalálhatja a titkosításhoz használt kulcsot a kulcstér mind a 256 darab kulcsának végigpróbálásával egy napon belül. Ma már ilyen gép létezik, és kevesebb mint 10000 dollárból megépíthető [Kumar és mások, 2006].

Háromszoros DES

Az IBM már 1979-ben felismerte, hogy a DES-kulcs túlságosan rövid, és a biztonság növelésének érdekében kidolgoztak egy háromszoros kódolást használó eljárást [Tuchman, 1979]. Az általuk választott módszert, melyet azóta a 8732-es számú Nemzetközi Szabvány is tartalmaz, a 8.8. ábrán mutatjuk be. Itt két kulcsot és három fokozatot használnak. Az első lépcsőben a nyílt üzenetet a szokott módon a K_1 kulccsal kódoljuk. Második lépésben dekódolást végzünk, melyhez a K_2-t használjuk kulcsként. Végül az így kapott eredményt ismét az első kulccsal kódoljuk.



8.8. ábra. *(a) DES-t használó háromszoros titkosítás. (b) Dekódolás*

Ez a konstrukció rögtön két kérdést vet fel. Először is, miért használtunk csupán két kulcsot három helyett? Másodszor, miért alkalmaztuk az **EDE** (**Encrypt Decrypt Encrypt – kódol-dekódol-kódol**) algoritmust, és nem az **EEE** (**Encrypt Encrypt Encrypt – kódol-kódol-kódol**) sorrendet? A két kulcs oka az, hogy még a legparanoiásabb kriptográfusok is egyetértenek abban, hogy az elkövetkező időkben az üzleti alkalmazások számára a 112 bites kulcshossz bőven elegendő. (Márpedig a kriptográfusoknál a paranoia nem hibának, hanem kívánatos tulajdonságnak számít.) 168 bit alkalmazása csak felesleges többletköltséget jelentene a tárolás és a kulcscsere során, és ehhez képest nem járna túl sok haszonnal.

Az alkalmazott sorrendre, mely szerint először kódolunk, dekódolunk, majd ismét kódolunk, a régi, egykulcsos DES-rendszerekkel való kompatibilitás miatt volt szükség. Mind a kódolás, mind a dekódolás egy függvénynek tekinthető a 64 bites számok halmazán. Kriptográfiai szempontból mindkét leképezés egyforma erejű. Az EEE helyett az EDE sorrend alkalmazása viszont lehetővé teszi, hogy egy háromlépcsős kódoló egy hagyományos társával is együtt tudjon működni a $K_1 = K_2$ azonos kulcsok használatával. A háromszoros DES ezen tulajdonsága miatt alkalmas a fokozatos bevezetésre. Az elméleti kriptográfusoknak ez nem sokat jelent, de annál fontosabb az IBM és ügyfelei számára.

8.2.2. AES – a fejlett titkosító szabvány

Amikor a DES és a háromszoros DES is aktív életük vége felé közeledtek, a **NIST** (**National Institute of Standards and Technology – Nemzeti Szabványügyi és Technoló-**

giai Intézet), vagyis az amerikai Kereskedelmi Minisztérium kormányzati szabványok jóváhagyásával megbízott ügynöksége úgy döntött, hogy a kormánynak új kriptográfiai szabványra van szüksége a nem bizalmas információ titkosításához. Az NIST élénken emlékezett még a DES-t övező vitákra, és jól tudta, hogy ha csak úgy bejelentene egy új szabványt, akkor mindenki, aki egy kicsit is járatos a kriptográfiában, rögtön azt feltételezné, hogy az NSA egy kiskaput épített a szabványba, vagyis hogy az NSA minden azzal titkosított dolgot el tud olvasni. Ilyen feltételek mellett pedig valószínűleg senki nem használná a szabványt, és az szépen, csendben kimúlna.

A NIST ezért a kormányzati bürokráciában meglepően új megközelítést választott: egy kriptográfiai versenyt írt ki. 1997 januárjában a világ minden tájáról kutatókat hívtak meg, hogy javaslatokat tegyenek az új szabványra, mely az **AES (Advanced Encryption Standard – fejlett titkosító szabvány)** nevet fogja kapni. A verseny szabályai a következők voltak:

1. Az algoritmusnak szimmetrikus blokk-kódoláson kell alapulnia.
2. A teljes konstrukciónak nyilvánosnak kell lennie.
3. Támogatni kell a 128, 192 és 256 bit hosszú kulcsokat.
4. Az eljárás legyen hardveresen és szoftveresen is megvalósítható.
5. Az algoritmus legyen nyilvános vagy megkülönböztetések nélküli alapon engedélyezett.

Tizenöt komoly javaslat érkezett. Később nyilvános konferenciákat szerveztek, ahol előadták a javaslatokat, és arra buzdították a hallgatóságot, hogy próbáljanak azokban sebezhető pontokat találni. 1998 augusztusában a NIST öt döntőst választott ki, elsősorban a biztonság, a hatékonyság, az egyszerűség, a rugalmasság és a memóriaigények (ez a beágyazott rendszereknél fontos) alapján. Ezután tovább folytak a konferenciák és a feltörési kísérletek.

2000 októberében a NIST bejelentette, hogy a Rijndaelt választotta, amelyet Joan Daemen és Vincent Rijmen dolgozott ki. A Rijndael név (melyet nagyjából rájn-dollnak kell ejteni), a szerzők vezetéknevéből származik (Rijmen + Daemen). 2001 novemberében a Rijndael az amerikai kormányzati AES szabványává vált, amelyet a FIPS 197-ben (Federal Information Processing Standard – Szövetségi Információfeldolgozási Szabvány) tettek közzé. A verseny rendkívüli nyíltságának, a Rijndael műszaki tulajdonságainak, valamint annak a ténynek köszönhetően, hogy a győztes csapat két fiatal belga kriptográfusból állt (akik valószínűleg nem építettek be kiskaput csak az NSA kedvéért), a Rijndael a világ meghatározó kriptográfiai szabványa lett. Az AES-titkosítás és -megfejtés mostanra számos mikroprocesszor (például Intel) utasításkészletének része lett.

A Rijndael 128-tól 256 bitig terjedő kulcsokat és blokkokat támogat, 32 bites lépésekben. A kulcsok és a blokkok hosszúságát egymástól függetlenül lehet megválasztani. Az AES viszont rögzíti, hogy a blokknak 128, a kulcsnak pedig 128, 192 vagy 256 bitesnek kell lennie. Kétséges, hogy fog-e bárki is valaha 192 bites kulcsokat használni, így az

AES-nek gyakorlatilag két változata van: az egyik 128 bites blokkokat és 128 bites kulcsot, a másik pedig 128 bites blokkokat és 256 bites kulcsot alkalmaz.

Az algoritmus további részletezésénél csak a 128/128-as esetet vizsgáljuk, mert a kereskedelemben valószínűleg ez lesz az irányadó. A 128 bites kulcs révén $2^{128} \approx 3 \times 10^{38}$ kulcsot tartalmazó kulcstérhez jutunk. Ha az NSA-nak sikerülne egy egymilliárd párhuzamosan működő processzort tartalmazó gépet építenie, melyben minden processzor pikomásodpercenként egy kulcsot értékel ki, nos, egy ilyen géppel még akkor is 1010 évig tartana a kulcstér végignézése. Addigra pedig már a Nap is kiég, úgyhogy az akkori emberek már csak gyertyafénynél olvashatják el az eredményeket.

Rijndael

Matematikai szempontból a Rijndael a Galois-mezők elvén alapszik, ennek köszönhetően van néhány bizonyítható biztonsági tulajdonsága. Mi azonban tekinthetjük most egyszerű C kódnak is anélkül, hogy belemennénk a matematikai részletekbe.

A DES-hez hasonlóan a Rijndael is helyettesítést és keverést alkalmaz, szintén több körben. A körök száma a kulcs és a blokk méretétől függ: 128 bites kulcs és 128 bites blokkok esetén 10 kör van, majd nagyobb kulcsok és blokkok esetén egyre több, egészen 14-ig. A DES-sel ellentétben azonban itt minden művelet egész bájtokra vonatkozik, hogy hardveresen és szoftveresen is hatékony megvalósításokat lehessen készíteni. A kód vázlatát a 8.9. ábra mutatja. Megjegyezzük, hogy ez a kód csak illusztráció. A védelmi

```
#define LENGTH 16                       /* bájtok száma az adatblokkban vagy a kulcsban */
#define NROWS 4                         /* a sorok száma az állapotmátrixban */
#define NCOLS 4                         /* az oszlopok száma az állapotmátrixban */
#define ROUNDS 10                       /* az iterációk száma */
typedef unsigned char byte;             /* előjel nélküli 8 bites egész */

rijndael(byte plaintext[LENGTH], byte ciphertext[LENGTH], byte key[LENGTH])
{
  int r;                                /* a ciklus indexe */
  byte state[NROWS][NCOLS];             /* az aktuális állapot */
  struct {byte k[NROWS][NCOLS];} rk[ROUNDS + 1];   /* a körök kulcsai */

  expand_key(key, rk);                  /* a körök kulcsainak előállítása */
  copy_plaintext_to_state(state, plaintext);   /* aktuális állapot inicializálása */
  xor_roundkey_into_state(state, rk[0]);       /* a kulcs és az állapot KIZÁRÓ VAGY értéke */

  for (r = 1; r <= ROUNDS; r++) {
      substitute(state);                /* S-doboz alkalmazása minden egyes bájtra */
      rotate_rows(state);               /* az i. sor elforgatása i bájttal */
      if (r < ROUNDS) mix_columns(state);   /* keverési függvény */
      xor_roundkey_into_state(state, rk[r]);   /* a kulcs és az állapot KIZÁRÓ VAGY értéke */
  }
  copy_state_to_ciphertext(ciphertext, state);   /* visszatérés az eredménnyel */
}
```

8.9. ábra. *A Rijndael kódjának C nyelvű vázlata*

kód jó megvalósítása még további gyakorlatokat igényel, olyat, mint amilyen például a könnyen sérülő memória nullázása annak használata után. Lásd például Ferguson és mások munkáját [2010].

A *rijndael* függvénynek három paramétere van: a *plaintext (nyílt szöveg)*, mely egy 16 bájtból álló tömb a bemeneti adatokkal, a *ciphertext (titkosított szöveg)*, egy 16 bájtos tömb a végeredmény számára, és a *key (kulcs)*, maga a 16 bájtos kulcs. A számítások során az adatok aktuális állapotát a *state (állapot)* bájttömb tárolja, melynek mérete *NROWS*×*NCOLS*. 128 bites blokkok esetén ez a tömb 4×4 bájtot tartalmaz, így 16 bájton a teljes 128 bites adatblokk tárolható.

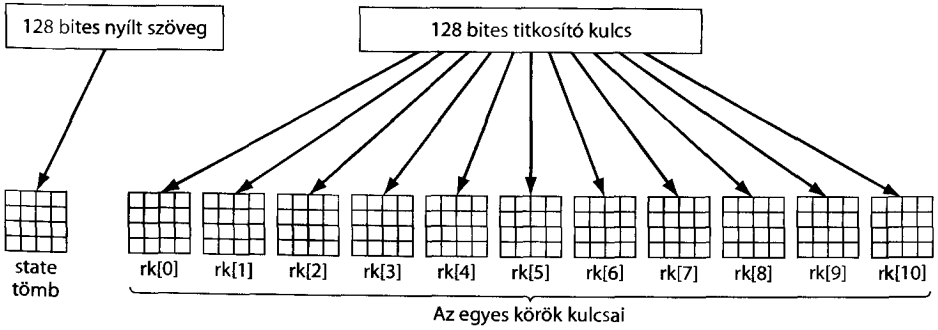
A *state* tömbbe kezdetben a nyílt szöveg kerül, amit később a számítás minden lépésében módosítanak. Egyes lépésekben bájtról bájtra történő helyettesítéseket végeznek, másokban a tömbön belül keverik a bájtokat, de más átalakításokat is végrehajtanak. Végül a *state* tömb tartalmát adják vissza titkosított szövegként.

A kód először kiterjeszti a kulcsot 11 darab, az állapotmátrixszal megegyező méretű tömbbe. Ezeket az *rk*-ban tárolják, ami egy tömbökből álló struktúra, melyben minden tömb egy állapotmátrixot tartalmaz. Az egyik tömböt a számítás elején, a többi tizet pedig a 10 kör során, egyenként alkalmazzák. A körökben használt kulcsokat meglehetősen bonyolult módon állítják elő a titkosító kulcsból, ezért ebben most nem mélyedünk el. Legyen elég annyit mondanunk, hogy a körök kulcsait ismételt forgatásokkal és a kulcsbitek különböző csoportjainak KIZÁRÓ VAGY (XOR) kapcsolatával állítják elő. A részleteket Daemen és Rijmen [2002] munkája tartalmazza.

A következő lépésben a nyílt szöveget átmásolják a *state* tömbbe, hogy az egyes körökben dolgozni lehessen vele. A másolás oszloponként történik: az első négy bájt kerül a 0. oszlopba, a következő négy az elsőbe, és így tovább. A sorok és az oszlopok is 0-tól vannak számozva, a körök viszont 1-től. A 12 darab 4 × 4 bájtos tömb kezdeti állapotát a 8.10. ábra szemlélteti.

Még egy lépés van a számítás törzse előtt: az *rk*[0]-t bájtról bájtra KIZÁRÓ VAGY (XOR) kapcsolatba hozzák a *state* tömbbel. Más szóval a *state*-ben található 16 bájtot egyenként kicserélik egy olyan bájtra, amely a saját maga és az *rk*[0] megfelelő bájtja KIZÁRÓ VAGY kapcsolatából adódik.

Most pedig eljött a fő mutatvány ideje! A ciklus 10 iterációt hajt végre, vagyis körönként egyet, és minden iteráció során átalakítja a *state* tömböt. Minden kör négy lépésből



8.10. ábra. *A state és az rk tömbök létrehozása*

áll. Az elsőben egy bájtról bájtra történő helyettesítést végeznek a *state* tömbön. Az egyes bájtokat az S-doboz indexelésére használják, hogy azok tartalmát az S-doboz megfelelő bejegyzésének tartalmával cseréljék fel. Ez tulajdonképpen egy közvetlen, egybetűhelyettesítéses kódolás. A DES-sel ellentétben a Rijndaelnak csak egyetlen S-doboza van.

A második lépés mind a négy sort balra forgatja. A 0. sor 0 bájttal fordul el (azaz nem változik), az 1. sor 1 bájttal, a 2. kettővel, a 3. pedig hárommal. Ez a lépés szétszórja az aktuális adatokat a blokkban, hasonlóan a 8.6. ábra permutációihoz.

A harmadik lépés az egyes oszlopokat a többitől függetlenül keveri össze. A keverést mátrixszorzással hajtják végre, melyben az új oszlop a régi oszlop és egy konstans mátrix szorzata lesz, ahol a szorzást a $GF(2^8)$ véges Galois-mező segítségével végzik el. Bonyolultnak hangzik ugyan, de létezik egy algoritmus, mely lehetővé teszi, hogy az új oszlop minden egyes elemét két darab táblázatból való kikeresés és három KIZÁRÓ VAGY művelet segítségével kiszámíthassuk [lásd Daemen és Rijmen, 2002, E függelék].

Végül a negyedik lépés az adott kör kulcsát KIZÁRÓ VAGY (XOR) kapcsolatba hozza a *state* tömbbel.

Mivel minden lépés megfordítható, a dekódolás egyszerűen az algoritmus visszafelé történő futtatásával végezhető el. Van ugyanakkor egy olyan trükk is, melynek segítségével a dekódolást is a kódoló algoritmussal végezhetjük el, más táblázatok felhasználásával.

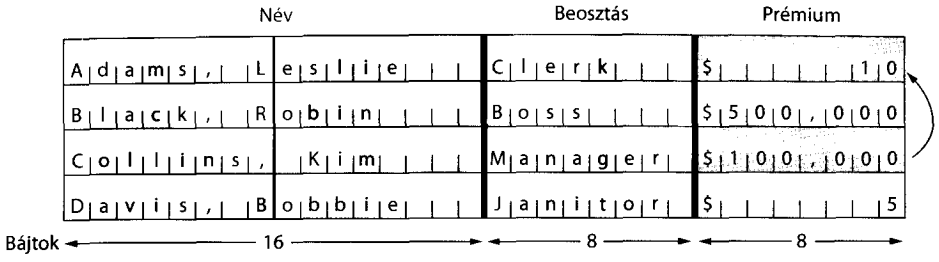
Az algoritmust nemcsak nagyfokú biztonságra, hanem nagy sebességre is tervezték. Egy 2 GHz-es gépen futó jó szoftveres megvalósítás elérheti a 700 Mb/s-os titkosítási sebességet, ami elég gyors ahhoz, hogy több mint 100 MPEG-2 videót lehessen valós időben titkosítani vele. A hardveres megvalósítások pedig még ennél is gyorsabbak.

8.2.3. Titkosítási módok

Bonyolultsága ellenére az AES (akárcsak a DES vagy bármelyik hasonló blokk-kódoló) alapjában véve csak egy egybetű-helyettesítéses kódoló, ami elég nagy karaktereket használ (128 bites karaktereket az AES-nél, illetve 64 biteseket a DES-nél). Ugyanaz a nyílt szövegblokk mindig ugyanazt a titkosított blokkot eredményezi. Ha például az *abcdefgh* nyílt szöveget ugyanazzal a DES-kulccsal kódoljuk 100-szor, akkor mind a 100-szor ugyanazt a titkosított szöveget kapjuk. A kódfejtő kihasználhatja ezt a tulajdonságot a kód feltörésére.

Elektronikus kódkönyv mód

64 bites blokkokat egyszerűbb ábrázolni, mint 128 biteseket, ezért a (háromszoros) DES példáján keresztül muttajuk be, hogyan lehet az egybetű-helyettesítéses titkosítók ezen tulajdonságát a kód részleges legyőzésére felhasználni. Ettől függetlenül az AES-nél szintén jelentkezik ez a probléma. A DES-t hosszú szövegek kódolására a legegyszerűbben úgy alkalmazhatjuk, hogy a szöveget felbontjuk egymást követő 8 bájtos (64 bites) blokkokra és azokat sorban ugyanazzal a kulccsal titkosítjuk. Az utolsó blokkot szükség esetén kiegészítjük, hogy elérje a 64 bites hosszt. Ezt az eljárást **ECB-módnak**



8.11. ábra. *Egy titkosítandó állomány nyílt szövege mint a 16 blokkos DES*

(**Electronic Code Book mode – elektronikus kódkönyv mód**) nevezzük, a régi kódkönyvek után, melyekben a nyílt szöveg minden szava fel volt sorolva, a hozzá tartozó kódszóval együtt (ez általában egy ötjegyű decimális szám volt).

A 8.11. ábrán egy adatállomány elejét találjuk, melyben egy cég az egyes dolgozóinak juttatandó éves prémiumokat tárolta. Az állomány 32 bájtos rekordok sorozatából áll, ahol minden rekord egy-egy alkalmazotthoz tartozik és a következő formátumú: 16 bájt név, 8 bájt a beosztás és 8 bájt a prémium mértéke. Mind a 16 db 8 bájt hosszú blokkot (0-tól 15-ig) DES segítségével kódoltunk.

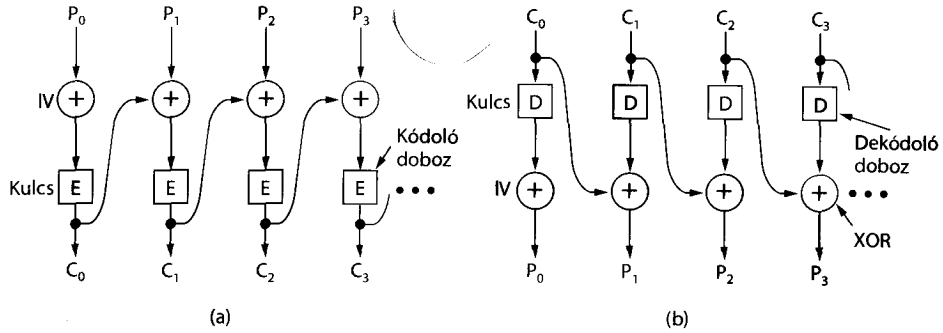
Leslie éppen összekülönbözött a felettesével, így nem sok jutalékra számíthat. Ezzel ellentétben Kim a főnök kedvence, amit mindenki tud. Az állomány Leslie kezébe kerül titkosítás után, de még mielőtt a bankba küldenék. Vajon kiegyenlítheti-e Leslie a prémiumok közötti különbséget csupán a titkosított állomány birtokában?

Minden különösebb gond nélkül! Csak annyit kell tennie, hogy a titkosított állomány 12. blokkját (ami Kim jutalékát tartalmazza) a 4. blokkba másolja (Leslie prémiumának helyére). Leslie nem ismeri a 12. blokk tartalmát, mégis egy vidámabb karácsonynak nézhet elébe. (Igazából a 8. blokkot is lemásolhatná, de azt sokkal könnyebben észrevennék, emellett Leslie sem annyira mohó.)

Titkosított blokkok láncolásának módja

Az ilyen típusú támadásokat az összes blokk-kódolóban kivédhetjük a kódolók különböző módon történő láncolásával, így a Leslie által is alkalmazott blokkcsere a visszafejtés után szemetet fog eredményezni a megváltoztatott blokk pozíciójától kezdődően. A láncolás egyik módja a **titkosított blokkok láncolása** (**cipher block chaining**). Ahogy azt a 8.12. ábrán is nyomon követhetjük, ennél a módszernél mindegyik nyílt szövegblokk és az azt megelőző titkosított blokk között KIZÁRÓ VAGY (XOR) műveletet hajtunk végre, mielőtt az adott blokkot kódolnánk. Ennek eredményeképpen ugyanaz a nyílt szövegrész már nem ugyanarra a titkosított blokkra fog leképeződni, így az algoritmusunk a továbbiakban már nem tekinthető csupán egy robusztus egybetű-helyettesítéses titkosítónak. Az első blokkhoz egy véletlenszerűen választott **inicializáló vektort** (**initialization vector**, IV) használunk párként, amit aztán a titkosított üzenettel együtt (de nyílt szövegként) továbbítunk.

Nézzük meg lépésről lépésre, hogy hogyan működik a 8.12. ábrán bemutatott blokkkódoló. Legelőször kiszámoljuk a $C_0 = E(P_0 \text{XOR} IV)$ értéket. Ezután a $C_1 = E(P_1 \text{XOR} C_0)$



8.12. ábra. *Titkosított blokkok láncolása. (a) Kódolás. (b) Dekódolás*

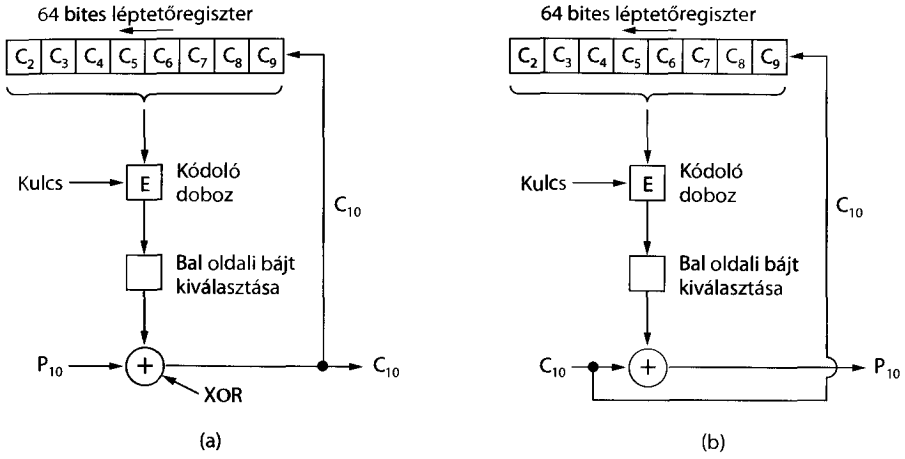
meghatározásával folytatjuk. Dekódoláskor a $P_0 = IV \text{ XOR } D(C_0)$ lépéssel kezdünk. Vegyük észre, hogy az *i.* szegmens kódolt megfelelője függ mindegyik 0-tól $i-1$-ig terjedő nyílt szövegblokktól, így ugyanazt a szövegrészt más-más alakra kódolja a titkosító attól függően, hogy hányadik blokkban szerepel. A Leslie által véghezvitt transzformáció a hozzá tartozó blokkoktól kezdődően badarságot fog eredményezni két blokkban a kimeneten. Egy agyafúrt biztonsági tisztviselő számára a rendellenesség helye nyilvánvalóvá teszi, hogy hol kezdje a vizsgálatot.

A titkosított blokkok láncolásának azon kedvező tulajdonsága, hogy ugyanaz a szegmens más-más szegmensekre képződik le, a kriptoanalízist is megnehezíti. Ez a legfőbb ok, ami miatt alkalmazzák.

Visszacsatolásos kódolási mód

A blokk-kódolók láncba fűzésének hátránya azonban az, hogy meg kell várnunk egy teljes 64 bites blokk megérkezését a dekódolás megkezdése előtt. Interaktív terminálóknál, ahol a felhasználók 8 karakternél rövidebb sorozat bevitele után is válaszra várhatnak, ez a módszer használhatatlanná válik. A bájtonként történő titkosítást az ún. **visszacsatolásos kódolóval** (**cipher feedback mode**) oldjuk meg, amit a 8.13. ábrán mutatunk be. Az ábrán azt az állapotot látjuk, amikor a kódoló a 0-tól 9-ig terjedő bájtokat már kódolta és elküldte. Amikor az ábrán látható módon megérkezik a 10. bájt, a DES-algoritmus a 64 bites shift regiszter tartalmából készíti el a 64 bites kódolt üzenetet. A kódolt szövegrész legbaloldalibb karaktere és a P_{10} között ezután egy KIZÁRÓ VAGY (XOR) műveletet hajtunk végre, amit aztán továbbítunk. A shift regiszter tartalmát 8 bittel balra mozgatjuk, melynek eredményeképpen a C_2 kipottyan a bal oldalon, a C_{10} pedig beül arra a helyre, amit ekkor hagy el a C_9. Vegyük észre, hogy a shift regiszter tartalma az eddig kódolásra került teljes nyílt szövegtől függ, így egy a kódolandó szövegben többször előforduló minta más-más kódolt párt kap, attól függően, hogy hol szerepel. Akárcsak a láncolt kódolóknál, itt is szükség van egy inicializáló vektorra, hogy elinduljon a játék.

A visszacsatolásos módban a dekódolás ugyanúgy működik, mint a kódolás. Tulajdonképpen a shift regiszter tartalmát inkább kódoljuk, mint dekódoljuk, mivel az a bájt, melyből a P_{10}-et állítjuk elő a C_{10}-zel való KIZÁRÓ VAGY művelet során, ugyanaz, mint



8.13. ábra. *Visszacsatolásos titkosító. (a) Kódolás. (b) Dekódolás*

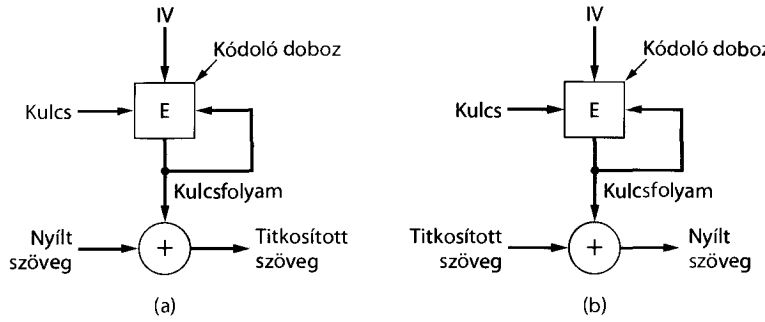
amit a C_{10} előállításához használunk a küldő oldalán, amikor a P_{10}-zel hajtjuk végre a KIZÁRÓ VAGY műveletet. A visszakódolás így tökéletesen működik, feltéve, ha a két shift regiszter azonos. A folyamatot a 8.13.(b) ábra szemlélteti.

Egy sajátságos mellékhatása a módszernek, hogy a titkosított szöveg egyetlen bitjének véletlenszerű megsérülése nyolc visszakódolt bájt deformálódását okozza, amelyekkel együtt szerepelt a shift regiszterben. Amint a hibás bit kikerül a dekódoló shift regiszterből, ismét hibátlan szöveget kódolhatunk vissza. Így egyetlen bit véletlen átbillenése lokálisan jelentkezik, és nem teszi tönkre a teljes hátralevő üzenetrészt, csupán csak annyi bitet, amennyi bit a shift regiszterben van.

Folyamtitkosítási mód

Vannak olyan alkalmazások, melyekben elfogadhatatlan, ha az átvitelben történt 1 bites hiba a nyílt szövegben egy 64 bites részt tesz tönkre. Az ilyen alkalmazások számára egy negyedik módszer, a **folyamtitkosítási mód** (**stream cipher mode**) jelentheti a megoldást. Ebben először egy inicializáló vektort (IV) kódolnak a kulcs segítségével, így kapnak egy kimeneti blokkot. Ezután ezt a kimeneti blokkot kódolják ugyanazon kulccsal, így jön létre a második kimeneti blokk. Ezt újfent kódolják, így áll elő a harmadik blokk és így tovább. A kimeneti blokkok (tetszőleges hosszúságú) sorozatát **kulcsfolyamnak** (**keystream**) nevezik, és egyszer használatos bitmintaként használják, vagyis KIZÁRÓ VAGY (XOR) kapcsolatba hozzák a nyílt szöveggel, és így áll elő végül a titkosított szöveg. A folyamatot a 8.14.(a) ábra mutatja be. Figyeljük meg, hogy az IV-t csak az első lépésben használjuk, azután mindig a kimenetet kódoljuk. Megfigyelhető még az is, hogy a kulcsfolyam független az adatoktól, vagyis ha kell, előre ki lehet számolni. A folyam ezenkívül az átviteli hibákra is teljesen érzéketlen. A dekódolás menetét a 8.14.(b) ábra mutatja.

A dekódolás úgy zajlik, hogy a vételi oldalon ugyanazt a kulcsfolyamot állítják elő. A kulcsfolyam csak az IV-től és a kulcstól függ, ezért nincsenek rá hatással a titkosított



8.14. ábra. *Folyamtitkosító. (a) Kódolás. (b) Dekódolás*

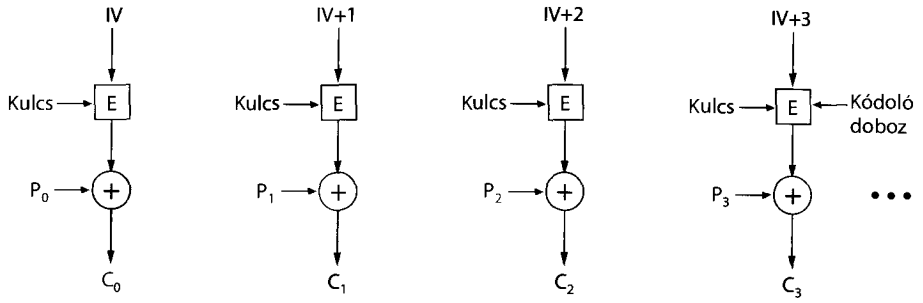
szöveg átvitelénél történt hibák. Ily módon az átvitt titkosított szövegben egy 1 bites hiba a visszafejtett nyílt szövegben is csak egy 1 bites hibát okoz.

Nagyon fontos, hogy sose használjuk kétszer ugyanazt a (kulcs, IV) párost egy folyamtitkosítóban, hiszen ha így tennénk, akkor mindkét alkalommal ugyanazt a kulcsfolyamot kapnánk, ezáltal pedig egy **kulcsfolyam-újrafelhasználásos támadás** (**keystream reuse attack**) veszélyének tennénk ki magunkat. Képzeljük el ugyanis, hogy a P_0 nyílt szövegblokkot egy kulcsfolyammal titkosítva megkapjuk a P_0 KIZÁRÓ VAGY K_0-t. Később egy másik nyílt szövegblokkot, Q_0-t is ugyanazzal a kulcsfolyammal titkosítunk, így kapjuk Q_0 KIZÁRÓ VAGY K_0-t. Ha a támadó megszerzi mindkét titkosított blokkot, akkor egy egyszerű KIZÁRÓ VAGY műveletet elvégezve megkapja P_0 KIZÁRÓ VAGY Q_0-t, vagyis megszabadul a kulcstól. A támadó tehát rendelkezik a két nyílt szövegblokk KIZÁRÓ VAGY értékével. Ha az egyik nyílt szöveget ismeri, vagy ki tudja találni, akkor meg tudja a másikat is. Bárhogy is legyen, a két nyílt szöveg KIZÁRÓ VAGY értékét meg lehet támadni az üzenetek statisztikai tulajdonságainak felhasználásával. Egy angol szöveg esetén például a folyam leggyakoribb karaktere valószínűleg két szóköz KIZÁRÓ VAGY értéke lesz, majd ezt követi a szóköz és az „e" betű KIZÁRÓ VAGY értéke stb. Egy szóval: a két nyílt szöveg KIZÁRÓ VAGY értékének birtokában a kódfejtőnek kiváló esélye van mindkét üzenet visszafejtésére.

Számláló mód

Az elektronikus kódkönyv módon kívül az összes módnál jelentkezik az a probléma, hogy nem lehetséges velük a titkosított adathoz való közvetlen hozzáférés. Tegyük fel például, hogy egy állományt átvisznek a hálózaton, majd titkosított formában tárolják a háttértáron. Ez ésszerű megoldás lehet, mondjuk akkor, ha a fogadó gép egy hordozható számítógép, amit esetleg ellophatnak. Ha tehát minden fontos állományt titkosított formában tárolunk, akkor nagyban csökkenthetjük a rossz kezekbe került számítógépekből kiszivárgó titkos információ által okozott károkat.

Csakhogy a háttértáron lévő állományokat sokszor nem szekvenciális úton érjük el; különösen igaz ez az adatbázisok állományaira. Ha egy állományt a titkosított blokkok láncolásával kódoltunk, akkor egy tetszőleges blokk eléréséhez először az összes azt meg-



8.15. ábra. *Titkosítás a számláló mód segítségével*

előző blokkot dekódolnunk kell, márpedig ez költséges megoldás. Emiatt egy újabb módot is kifejlesztettek: ez a **számláló mód (counter mode)**, melyet a 8.15. ábra szemléltet. Itt nem közvetlenül a nyílt szöveget titkosítjuk, hanem először egy konstanssal megnöveljük az inicializáló vektort, az eredményt kódoljuk, és az így kapott titkosított szöveget hozzuk KIZÁRÓ VAGY kapcsolatba a nyílt szöveggel. Ha minden újabb blokknál eggyel növeljük az inicializáló vektort, akkor az állomány tetszőleges blokkját könnyen dekódolhatjuk anélkül, hogy előtte az összes azt megelőző blokkot is dekódolnunk kellene.

A számláló mód hasznos módszer ugyan, mégis van egy gyengéje, amire érdemes rámutatni. Tegyük fel, hogy valamikor a későbbiekben ugyanazt a *K* kulcsot használjuk (más nyílt szöveggel, de ugyanazzal az IV-vel), és a támadó megszerzi mindkét titkosított szöveget. A kulcsfolyam mindkét esetben azonos, ezáltal a kódot ugyanolyan kulcsfolyam újrafelhasználásos támadás fenyegeti, mint amilyet a folyamtitkosítóknál láttunk. A kódfejtőnek mindössze annyi a dolga, hogy KIZÁRÓ VAGY kapcsolatba hozza a két titkosított szöveget, ezáltal megszabadul a kriptográfiai védelemtől és megkapja a két nyílt szöveg KIZÁRÓ VAGY értékét. A számláló mód ezen gyengéje persze nem jelenti azt, hogy maga az ötlet rossz. Ez mindössze annyit jelent, hogy mind a kulcsokat, mind az inicializáló vektort egymástól függetlenül és véletlenszerűen kell megválasztani. Ha véletlenül kétszer ugyanazt a kulcsot használnánk, de az IV különbözik a két esetben, akkor a nyílt szöveg még így is biztonságban marad.

8.2.4. Egyéb kódolók

Az AES (Rijndael) és a DES számítanak a legismertebb szimmetrikus kulcsú kriptográfiai algoritmusoknak és ipari szabványnak, ha csak a felelősségvállalást tekintjük. (Senki nem fog nekünk szemrehányást tenni azért, ha a termékeinkben az AES-t használjuk, és azt feltörik, de minden bizonnyal rossz néven veszik, ha egy nem szabványos titkosítót használunk, amelyet később feltörnek.) Érdemes ugyanakkor megemlíteni, hogy számos más szimmetrikus kulcsú titkosító módszert is kidolgoztak, melyek közül néhányat különböző termékekbe ágyazva is megtalálunk. A 8.16. ábra néhány gyakoribb módszert sorol fel. Ezeknek a kombinációját is használni lehet (például az AES-t a Twofish felett), hogy mindkét titkosítást fel kelljen törni az adatok kinyerése érdekében.

Kód	Szerző	Kulcshossz	Megjegyzés
DES	IBM	56 bit	Ma már túl gyengének számít
RC4	Ronald Rivest	1–2048 bit	Vigyázat: egyes kulcsok gyengék
RC5	Ronald Rivest	128–256 bit	Jó, de szabadalmaztatott
AES (Rijndael)	Daemen és Rijmen	128–256 bit	A legjobb választás
Serpent	Anderson, Biham, Knudsen	128–256 bit	Nagyon erős
Háromszoros DES	IBM	168 bit	Jó, csak kezd öregedni
Twofish	Bruce Schneier	128–256 bit	Nagyon erős; széles körben elterjedt

8.16. ábra. *Néhány gyakori szimmetrikus kulcsú kriptográfiai algoritmus*

8.2.5. Kriptoanalízis

Mielőtt elhagynánk a szimmetrikus kulcsú kriptográfia témáját, érdemes legalább említés szintjén tárgyalnunk négy kriptoanalitikai fejlesztést. Az első ilyen a **differenciális kriptoanalízis (differential cryptoanalysis)**, melyet Biham és Shamir [1993] dolgozott ki. A módszer tetszőleges blokk-kódoló elleni támadásra használható. Működésének alapja, hogy a kódolás során minimálisan eltérő szövegblokkok útját követik nyomon párhuzamosan. Sok esetben egyes bitminták sokkal nagyobb gyakorisággal fognak előfordulni, mint mások, ez a megfigyelés pedig valószínűségi alapon történő támadásokat tesz lehetővé.

A második figyelemre méltó módszer a **lineáris kriptoanalízis (linear cryptoanalysis)** [Matsui, 1994]. Ezzel mindössze 2^{43} ismert nyílt szöveg alapján feltörhető a DES. Úgy működik, hogy a nyílt és titkosított szövegek adott bitjei között KIZÁRÓ VAGY műveletet végez, és az eredményt vizsgálja. Ha elég gyakran ismételjük a műveletet, akkor az eredményben az 1-es és 0-s biteknek nagyjából egyenlő arányban kell megjelenniük. A kódolók azonban gyakran hajlamosak az egyik vagy másik irányba eltérést mutatni, és akármilyen kicsi is ez az eltérés, mégis felhasználható a feltörés munkaigényének csökkentésére. A részleteket Matsui dolgozata tartalmazza.

A harmadik módszer az elektromos teljesítményfelvétel elemzésével próbálja megtalálni a titkos kulcsokat. A számítógépeken rendszerint 3 V felel meg az 1-es bitnek, és 0 V a 0 bitnek. Ebből kifolyólag egy 1-es feldolgozásához több elektromos energia kell, mint egy 0 feldolgozásához. Ha a kriptográfiai algoritmus egy hurkot tartalmaz, melyben a kulcsbiteket sorban egymás után dolgozzák fel, és a támadó lecseréli a gép *n* GHz-es órajelét egy lassabb (például 100 Hz-es) órajelre, majd krokodilcsipeszeket rak a processzor táp- és földcsatlakozójára, akkor az egyes gépi utasítások által felvett teljesítmény pontosan megfigyelhető lesz. Ezekből az adatokból pedig meglepően egyszerű kikövetkeztetni a kulcsot. Az effajta kriptoanalízis ellen csak úgy lehet védekezni, ha az algoritmust gondosan, assembly nyelven kódolják, és biztosítják, hogy a teljesítményfelvétel független legyen a kulcstól és az egyes körök kulcsaitól is.

A negyedik módszer az időzítéselemzés. A kriptográfiai algoritmusok tele vannak if utasításokkal, melyek az egyes körök kulcsaiban tesztelik a biteket. Ha a then és else utasítások végrehajtása különböző időt vesz igénybe, akkor az órajel lelassításával és az egyes lépések időigényének figyelésével itt is ki lehet következtetni az egyes körök kulcsait. Ha pedig ismertek a körkulcsok, akkor általában az eredeti kulcs is kiszámítható. A teljesítmény- és időzítéselemzés egyszerre is alkalmazható, hogy még könnyebben eredményre jussunk. Ezek a módszerek kissé egzotikusnak tűnhetnek ugyan, a valóságban azonban igen hatékony eljárásoknak bizonyulnak, melyekkel bármilyen kódot fel lehet törni, melyet nem készítettek fel külön arra, hogy ellenálljon az ilyen próbálkozásoknak.

8.3. Nyilvános kulcsú algoritmusok

A legtöbb kriptográfiai rendszer gyenge pontja hosszú időn keresztül a kulcskiosztás problémája volt. Függetlenül a titkosító rendszer erejétől, ha a kulcs a támadó kezébe került, a rendszer teljesen védtelenné vált. Mivel a kriptológusok mindig elfogadták azt a nézetet, hogy a kódoláshoz és dekódoláshoz szükséges kulcsoknak azonosnak (vagy egymásból könnyűszerrel generálhatóknak) kell lenniük, és ezeket minden érintett felhasználóhoz el kell juttatni, feloldhatatlan problémának tűnt a kulcsok illetéktelen kezektől való megóvása, mivel azokat nem lehetett páncélszekrénybe zárni a kulcskiosztás szükségessége miatt.

1976-ban két stanfordi kutató, Diffie és Hellman [1976] egy merőben új kriptográfiai rendszert javasolt, amelyben a kódoló és dekódoló kulcsok nemcsak különbözők, de egymásból nem állíthatók elő. A módszerükben szereplő kódoló (E) és dekódoló (D) algoritmussal szemben három követelményt támasztottak. Ezek a feltételek a következők:

1. $D(E(P)) = P$

2. D előállítása E alapján rendkívül nehéz feladat legyen.

3. E feltörhetetlen legyen választott nyílt szöveg típusú támadással.

Az első szabály azt mondja, hogy ha a D műveletet alkalmazzuk a kódolt szövegre, $E(P)$-re, akkor az eredeti szöveget, P-t kell visszakapnunk. A második pont önmagáért beszél. A harmadik követelményre azért van szükség, mivel – ahogy ezt mindjárt látni fogjuk – a támadók kényük-kedvük szerint kísérletezhetnek majd a kódoló eljárással. Ilyen feltételek mellett semmi sem szól az ellen, hogy a kódoló kulcsot nyilvánossá tegyük.

A módszer a következőképpen működik. Tegyük fel, hogy Aliz titkos üzeneteket szeretne fogadni, ezért először kifejleszt két olyan algoritmust, melyek megfelelnek a fenti követelményeknek. Ezek után a kódoló algoritmust, illetve a titkosító kulcsot nyilvánosságra hozza, innen ered a **nyilvános kulcsú titkosítás (public-key cryptography)** elnevezés is. Aliz felrakhatja például a nyilvános kulcsát a webes honlapjára. Mi az E_A jelölést fogjuk alkalmazni arra a titkosító (encryption) algoritmusra, melyet Aliz nyilvános kulcsával paramétereztünk. Hasonlóképp, az Aliz egyéni kulcsával paraméterezett

(titkos) dekódoló (decryption) algoritmus jele D_A lesz. Bob ugyanúgy tesz, mint Aliz: ő is nyilvánosságra hozza E_B-t, de titokban tartja D_B-t.

Most pedig nézzük meg, hogyan oldhatjuk meg az Aliz és Bob között létrehozandó, titkosított csatorna problémáját, ha ők ketten még sohasem találkoztak azelőtt! Feltesszük, hogy mind Aliz, mind Bob titkosító kulcsa, azaz E_A és E_B is nyilvánosan olvasható állományokban vannak. Aliz veszi az első P üzenetét, kiszámítja $E_B(P)$-t, és az eredményt elküldi Bobnak. Bob ezt a D_B titkos kulcsa segítségével visszafejti [azaz kiszámítja $D_B(E_B(P)) = P$-t]. Rajta kívül senki más nem képes a titkosított üzenetet, $E_B(P)$-t elolvasni, mivel feltételezzük, hogy a kódolás erős védelmet nyújt, és D_B előállítása az ismert E_B alapján túl bonyolult feladat. Az R válasz elküldéséhez Bobnak az $E_A(R)$ üzenetet kell leadnia. Aliz és Bob ezek után biztonságosan kommunikálhatnak.

Érdemes talán egy megjegyzést tenni a szóhasználattal kapcsolatban. A nyilvános kulcsú módszerek minden felhasználótól két kulcsot követelnek meg: egy nyilvános kulcsot, melynek segítségével a világon bárki kódolhat számukra információt, és egy egyéni kulcsot, melyet tulajdonosa az üzenetek dekódolására használ. Ebben a könyvben következetesen *nyilvános* és *egyéni* kulcsként fogunk rájuk hivatkozni, hogy megkülönböztessük őket azoktól a *titkos* kulcsoktól, amelyeket a hagyományos szimmetrikus kulcsú kriptográfiában alkalmaznak.

8.3.1. RSA

Az egyetlen buktató, hogy a fenti három követelménynek eleget tevő eljárásokat kellene találnunk. A nyilvános kulcsú titkosítás nyilvánvaló előnyei miatt számos szakember komoly munkába kezdett, és sikerült mára néhány ilyen módszert kifejleszteniük. Az egyiket ezek közül egy, az M.I.T.-n dolgozó csoport [Rivest és mások, 1978] alkotta meg. Az algoritmus azóta a felfedezők (Rivest, Shamir, Adleman) kezdőbetűi alapján **RSA-algoritmus** néven vonult be a köztudatba. A módszer immár negyed százada túlélt minden egyes támadási kísérletet, tehát nagyon erősnek tekinthető. Ennek köszönhetően sok gyakorlati megoldás is ezen alapszik. A legfőbb hátránya, hogy a kielégítő biztonság érdekében legalább 1024 bites kulcsokat igényel (szemben a szimmetrikus kulcsú algoritmusok 128 bites kulcsaival), ami meglehetősen lassúvá teszi.

Az RSA a számelmélet tételein alapszik. Most röviden összefoglaljuk az algoritmus lépéseit, a részletek megismeréséhez az eredeti művet ajánljuk.

1. Válasszunk két nagy (jellemzően 1024 bites) prímszámot, p-t és q-t.

2. Számoljuk ki az $n = p \times q$ és a $z = (p-1) \times (q-1)$ számokat.

3. Válasszunk egy z-hez relatív prímet, jelöljük d-vel.

4. Keressünk egy olyan e számot, melyre $e \times d = 1 \bmod z$.

A fenti paraméterek előzetes meghatározása után megkezdhetjük a titkosítást. A nyílt szöveget, mint egyszerű bitsorozatot blokkokra osztjuk oly módon, hogy egy kódolandó

szegmens (P) a $0 \le P < n$ intervallumba essen. Ezt legkönnyebben úgy érhetjük el, ha az eredeti üzenet bitjeit k bit hosszú csoportokba szedjük, ahol k az a legnagyobb egész, melyre még fennáll a $2^k < n$ összefüggés.

A titkosítandó üzenetdarab (P) alapján kiszámítjuk $C = P^e \pmod n$ értéket. C visszakódolásához a $P = C^d \pmod n$ összefüggést használjuk. Bebizonyítható, hogy minden olyan P-re, mely a fenti tartományba esik, a megadott kódoló és dekódoló függvények egymás inverzei. A kódoláshoz az e és az n számokra van szükség, míg a visszafejtéshez a d-re és az n-re. Ennek alapján a nyilvános kulcs az (e, n) párból, míg az egyéni kulcs a (d, n) párból fog állni.

A módszer biztonsága a nagy számok faktorizálásának nehézségén alapszik. Ha a támadó képes lenne szorzatalakra hozni a mindenki által ismert n számot, megkapná a p-t és a q-t, ami alapján a z-t kiszámolhatná. A z érték ismeretében pedig az e és a d meghatározása az euklideszi algoritmussal elvégezhető. Szerencsére azonban az utóbbi 300 év próbálkozásai alapján a matematikusok többsége azon a véleményen van, hogy a nagy számok szorzatra bontása rendkívül nehéz feladat.

Rivest és kollégái szerint egy 500 digites szám számítógépes faktorizálása 10^{25} évet venne igénybe az összes lehetőséget kipróbálva. Mindkét esetben az általuk leghatékonyabb algoritmust és egy olyan számítógépet tételeztek fel, mely egy utasítást 1 μs alatt hajt végre. Ez még akkor is 10^{16} évet igényelne, ha egy millió lapka párhuzamosan futna és mindegyik utasítás végrehajtási ideje 1 ns lenne. Még ha folytatódna is a számítógépek sebességének évtizedenkénti egy nagyságrenddel való növekedése, akkor is sok év múlva kerülne elérhető közelségbe az 500 digites számok gyors faktorizálása. Utódainknak ekkor sem kell mást tenniük, mint a p és a q számokat hosszabbra választani.

Az RSA-algoritmus egy klasszikus iskolapéldáját mutatjuk be a 8.17. ábrán. A példa kedvéért kis számokat választottunk: $p = 3$ és $q = 11$, amiből $n = 33$ és $z = 20$ adódik. A $d = 7$ választás megfelelőnek tűnik, mivel 7-nek és 20-nak nincs közös osztója. A fenti értékekkel e könnyen kiszámolható a $7e = 1 \pmod{20}$ összefüggés alapján, amelyből $e = 3$. A P nyílt szöveg sifrírozott párja, C, a $C = P^3 \pmod{33}$ kifejezésből adódik. Az üzenet fogadója az eredeti szöveget a $P = C^7 \pmod{33}$ képlet segítségével kapja vissza. Az ábra a „SUZANNE" tartalmú üzenet kódolási lépéseit mutatja.

Mivel a példában szereplő prímeket készakarva kicsire választottuk, a P értékének is alacsonynak (33-nál kisebbnek) kell lennie, ezért egy kódolandó szövegblokk csak egy

Nyílt szöveg (P)		Titkos szöveg (C)			Dekódolás után	
Szimbólum	Számérték	P^3	$P^3 \pmod{33}$	C^7	$C^7 \pmod{33}$	Szimbólum
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	01	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	05	E
		A küldő számítása		A fogadó számítása		

8.17. ábra. *Példa az RSA-algoritmusra*

karaktert tartalmazhat. Ennek eredményeképpen egy egyszerű egybetű-helyettesítéses kódolót kapunk, ami nem túl hatásos. Ha ehelyett a p-t és q-t $\approx 2^{512}$ nagyságrendben választottuk volna meg, akkor az $n \approx 2^{1024}$ érték adódott volna, így minden blokk 1024 bitet vagy 128 darab 8 bites karaktert tartalmazhatott volna, szemben a DES 8 és az AES 16 karakterével.

Érdemes megjegyeznünk, hogy a fenti módon használt RSA hasonlóan viselkedik az ECB-módú szimmetrikus algoritmusokhoz abban a tekintetben, hogy ugyanaz a bemeneti blokk itt is mindig ugyanazt a kimeneti blokkot fogja eredményezni. Ennélfogva a titkosításnál célszerű valamilyen láncolást alkalmazni. A gyakorlatban azonban a legtöbb RSA-alapú rendszer a nyilvános kulcsokat csak egy, egyszer használatos viszonykulcs biztonságos szétosztására használja. A viszonykulcsot ezután hagyományos titkos kulcsként lehet felhasználni, például az AES- vagy a háromszoros DES-algoritmusokban. Az RSA túl lassú, ha nagy mennyiségű adatot kell titkosítani, de a kulcsok szétosztására széles körben használják.

8.3.2. Más nyilvános kulcsú eljárások

Bár az RSA igen széles körben alkalmazott, korántsem ez az egyetlen ismert nyilvános kulcsú algoritmus. Az első nyilvános kulcsú eljárás a hátizsák- (knapsack) módszer volt [Merkle és Hellman, 1978]. Tegyük fel, hogy az üzenetet küldeni szándékozó fél nagyszámú különböző súlyú tárggyal rendelkezik. Az üzenet kódolásához a tárgyak közül titokban kiválaszt párat, és azokat a zsákba teszi. A zsák összsúlyát, valamint a szóba jöhető tárgyak listáját ezek után nyilvánosságra hozza. A zsákban levő tárgyak listáját titokban tartja. Néhány magától értetődő korlátozás mellett azon tárgyak listájának előállítása, mely adott össztömeggel rendelkezik, algoritmikusan nehéz feladat, és ez képezi a módszer alapját.

Az algoritmus kifejlesztője, Ralph Merkle, olyannyira biztos volt a módszer feltörhetetlenségében, hogy 100 dolláros díjat ajánlott fel annak, akinek mégis sikerül. Adi Shamir (az RSA-ban szereplő „S" betű tulajdonosa) hamarosan jelentkezett a megoldással a 100 dollárért. Merkle nem riadt vissza, így nemsokára 1000 dolláros díjat tűzött ki feljavított algoritmusának feltöréséért. Ron Rivest (az RSA „R" betűje) nem késlekedett a megoldással, és bezsebelte a díjat. Merkle ezek után nem mert a következő változatra 10 000 dolláros tétet tenni, így az „A" betű (Leonard Adleman) hoppon maradt. Annak ellenére, hogy újra javítottak rajta, a hátizsák algoritmust nem tekintik biztonságosnak, és nem igazán alkalmazzák.

Más nyilvános kulcsú módszerek a diszkrét logaritmus számításának számítási igényére alapoznak [Rabin, 1979]. El Gamal [1985] és Schnorr [1991] erre az elvre épülő algoritmusokat fejlesztett ki.

Létezik még pár másfajta séma is, mint például Menezes és Vanstone [1993] elliptikus görbéken alapuló módszere, de az eljárások alapvetően két nagy kategóriába sorolhatók: az egyikbe tartoznak a nagy számok faktorizálásának nehézségén alapuló, a másikba pedig a diszkrét logaritmusok nagy prímekkel vett modulusát számoló módszerek. Ezen problémák megfejtése tényleg nagyon nehéznek tekinthetők – a matematikusok sok éven át dolgoztak rajtuk, és nem értek el semmilyen jelentős áttörést.

8.4. Digitális aláírások

Sok hivatalos, pénzügyi és más dokumentum hitelességvizsgálatát eldönti egy kézzel írott aláírás jelenléte vagy hiánya. A fénymásolatok ilyen szempontból szóba sem jönnek. Ahhoz azonban, hogy számítógépes üzenetkezelő rendszerek válthassák ki a papír- és tintaalapú, fizikai úton vándorló dokumentumokat, valamilyen új eljárásra van szükség, hogy a dokumentumokat hamisíthatatlan módon lehessen aláírni.

Nehéz feladat a kézi aláírások helyettesítésének problémájára megoldást találni. Alapvetően azt kell elérni, hogy legyen egy olyan rendszer, amelynek segítségével az egyik fél elküldhessen a másiknak egy aláírt üzenetet, mégpedig úgy, hogy közben teljesüljenek az alábbi feltételek:

1. A fogadó ellenőrizhesse a feladó valódiságát.
2. A küldő később ne tagadhassa le az üzenet tartalmát.
3. A fogadó saját maga ne rakhassa össze az üzenetet.

Az első követelmény például egy pénzügyi rendszerben szükséges. Mikor egy ügyfél számítógépe utasítja a bank számítógépét, hogy vegyen egy tonna aranyat, annak meg kell tudnia állapítani, hogy a számítógép, amely az utasítást adta, tényleg annak a cégnek a tulajdona, amelyiknek a bankszámláját terhelni kell. Más szóval, a banknak hitelesítenie kell az ügyfelet (és az ügyfélnek is hitelesítenie kell a bankot).

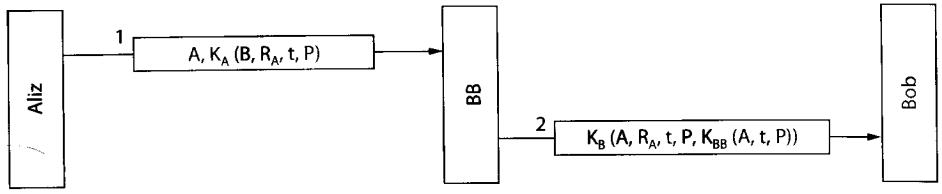
A második követelmény azért szükséges, nehogy a bankot kijátsszák. Tegyük fel, hogy a bank megveszi az egy tonna aranyat, és közvetlenül utána az arany ára drasztikusan leesik. Egy nem becsületes ügyfél esetleg beperelheti a bankot, mondván, hogy sohasem adott utasítást arany vásárlására. Amikor a bank a bíróságon bemutatja az üzenetet, az ügyfél letagadja, hogy ő küldte volna. Azt a tulajdonságot, hogy egyik szerződő fél sem tagadhatja le később az aláírását, **letagadhatatlanságnak** (**nonrepudiation**) nevezzük. Az alábbiakban tanulmányozandó digitális aláírási sémák segítenek biztosítani ezt a tulajdonságot.

A harmadik követelmény abban az esetben fontos, ha az arany ára megugrik, és a bank megpróbál egy olyan üzenetet konstruálni, amelyben az ügyfél egy tonna arany helyett egy rudat rendelt. Az effajta csalásnál a bank a maradék aranyat egyszerűen megtartja magának.

8.4.1. Szimmetrikus kulcsú aláírások

A digitális aláírás egyik megközelítésében adva van egy központi hitelességvizsgáló szerv, amely mindent tud, és amelyben mindenki megbízik, nevezzük mondjuk Nagy Testvérnek (Big Brother, *BB*). Ezután minden felhasználó választ egy titkos kulcsot, és saját kezűleg elviszi *BB* irodájába. Ily módon csak Aliz és *BB* ismeri Aliz titkos kulcsát, K_A-t és így tovább.

Ha Aliz a *P* aláírt, kódolatlan üzenetet szeretné elküldeni bankárjának, Bobnak, akkor előállítja a $K_A(B, R_A, t, P)$-t, ahol *B* Bob személyazonossága, R_A egy Aliz által választott



8.18. ábra. *Digitális aláírás a Nagy Testvér (BB) közreműködésével*

véletlen szám, *t* egy időbélyeg, ami a frissességet biztosítja, és a $K_A(B, R_A, t, P)$ az Aliz kulcsával, K_A-val kódolt üzenet. Aliz tehát a 8.18. ábrán látható módon elküldi az üzenetet. *BB* látja, hogy az üzenet Aliztól jött, visszakódolja, és az ábrán látható módon küld egy üzenetet Bobnak. A Bobnak küldött üzenet tartalmazza Aliz eredeti üzenetét és a $K_{BB}(A, t, P)$ aláírt üzenetet is. Bob ezután teljesíti Aliz kérését.

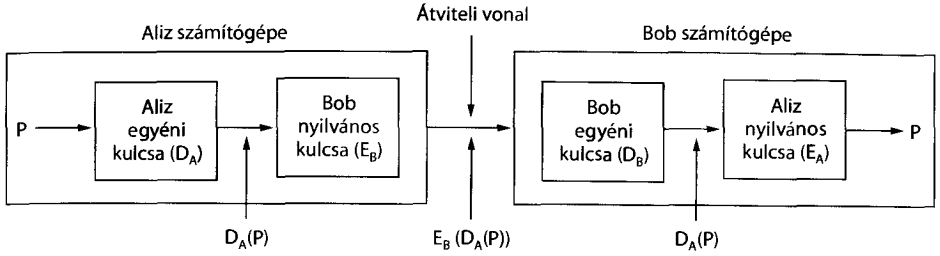
Mi történik, ha Aliz utóbb letagadja az üzenetet? Először is mindenki beperel mindenkit (legalábbis az Egyesült Államokban). Amikor az ügy végül is bíróságra kerül, és Aliz nyomatékosan tagadja, hogy a szóban forgó levelet ő küldte volna Bobnak, a bíró meg fogja kérdezni Bobot, hogy mivel tudja bizonyítani, hogy az üzenet Aliztól, és nem Trudytól jött. Bob először is rámutat, hogy *BB* nem fogad el üzenetet Aliztól, ha az nem a K_A-val van titkosítva, tehát Trudynak nincs lehetősége hamis levelet küldeni *BB*-nek Aliz nevében, hiszen *BB* rögtön észrevenné a csalást.

Bob ezek után drámai módon felmutatja az A bizonyítékot, $K_{BB}(A, t, P)$-t. Bob azt állítja, hogy ez egy üzenet *BB* aláírásával, ami bizonyítja, hogy Aliz elküldte *P*-t Bobnak. A bíró tehát megkéri *BB*-t (akiben mindenki megbízik), hogy kódolja vissza az A bizonyítékot. Mikor *BB* alátámasztja, hogy Bobnak igaza van, a bíró a pert Bob javára dönti el. Az ügy ezzel véget ért.

Egy lehetséges probléma a 8.18. ábrán látható aláírás protokollal az, hogy Trudy mindkét üzenetet képes visszajátszani. Ennek a problémának a minimalizálására mindent időbélyegekkel látnak el. Ezenkívül Bob megvizsgálhatja az összes nemrég érkezett üzenetet, hogy R_A szerepelt-e bennük. Ha igen, akkor az üzenet mint visszajátszás eldobható. Vegyük észre, hogy Bob visszautasít minden elég régi üzenetet az időbélyegek miatt. A gyors visszajátszás ellen Bob úgy védekezik, hogy ellenőrzi minden beérkező üzenetben R_A-t, hogy lássa, nem érkezett-e az utóbbi egy órában hasonló üzenet. Ha nem, akkor nyugodtan feltételezheti, hogy a kérés vadonatúj.

8.4.2. Nyilvános kulcsú aláírások

A szimmetrikus kulcsú kriptográfiát alkalmazó digitális aláírások alapvető szervezési problémája, hogy mindenkinek egységesen meg kell bíznia a Nagy Testvérben. A Nagy Testvér ráadásul minden aláírt levelet el tud olvasni. A legkézenfekvőbb jelöltek a Nagy Testvér szerepére a kormány, a bankok, a könyvelők és az ügyvédek lehetnek. Sajnos ezen szervezetek közül egyiknek sem sikerült teljes bizalmat keltenie az összes állampolgárban. Jó volna tehát, ha a dokumentumok aláírásához nem lenne szükség egy megbízható hatóság közreműködésére.



8.19. ábra. *Digitális aláírások nyilvános kulcsú titkosítás használatával*

Szerencsére ezen a téren jelentős eredményt kínálhat a nyilvános kulcsú titkosítás. Tegyük fel, hogy a nyilvános kulcsú titkosító algoritmusok a szokásos $D(E(P)) = P$ tulajdonság mellett szintén rendelkeznek az $E(D(P)) = P$ tulajdonsággal. (A feltételezés nem alaptalan, hiszen az RSA rendelkezik ezzel a tulajdonsággal.) Ezt feltételezve Aliz elküldhet egy aláírt kódolatlan üzenetet Bobnak, ha $E_B(D_A(P))$-t küldi el. Fontos észrevenni, hogy Aliz ismeri mind saját (egyéni) visszakódoló kulcsát, D_A-t, mind Bob nyilvános kulcsát, E_B-t, tehát képes egy ilyen üzenet megkonstruálására.

Amikor Bob megkapja ezt az üzenetet, átalakítja azt saját titkos kulcsával, és eredményül megkapja $D_A(P)$-t, ahogy azt a 8.19. ábra is mutatja. Ezt a szöveget egy biztonságos helyen tárolja, majd visszakódolja E_A segítségével, hogy megkapja az eredeti kódolatlan üzenetet.

Annak érdekében, hogy lássuk, hogyan is működik az aláírás, tételezzük fel, hogy Aliz ismételten tagadja, hogy P-t ő küldte volna Bobnak. Mikor az ügy a bíróság elé kerül, Bob bemutathatja mind P-t, mind $D_A(P)$-t. A bíró egyszerűen ellenőrizheti, hogy a Bob-féle, D_A-val titkosított üzenet tényleg érvényes azzal, hogy visszakódolja azt E_A-val. Mivel Bob nem ismeri Aliz egyéni kulcsát, ezért csakis úgy tehetett szert egy azzal titkosított üzenetre, hogy azt Aliz küldte neki. Amíg Aliz börtönben ül hamis eskü és csalás vádjával, rengeteg ideje lesz érdekes, új, nyilvános kulcsú algoritmusok kigondolására.

Annak ellenére, hogy a nyilvános kulcsú titkosítás használata digitális aláírásokhoz egy elegáns módszer, azért vannak problémák, amelyek nem az algoritmus alapjaiból, hanem a futtatási környezetből adódnak. Csak egy a sok közül, hogy Bob csak addig tudja bizonyítani, hogy a levelet Aliz küldte, ameddig D_A titokban van. Ha Aliz felfedi titkos kulcsát, akkor az állítás már nem helytálló, hiszen bárki küldhette az üzenetet, akár maga Bob is.

A probléma akkor jelentkezhet, ha például Bob Aliz tőzsdeügynöke. Aliz utasítja Bobot, hogy vegyen meg egy bizonyos részvényt vagy kötvényt. Közvetlenül vásárlás után az árak drasztikusan leesnek. Aliz, hogy letagadhassa a Bobnak küldött üzenetet, elszalad a rendőrségre azzal, hogy betörtek otthonába, és lemásolták a kulcsát. Attól függően, hogy melyik országban él, lehet, hogy Aliz törvényesen felelősségre vonható, de az is lehet, hogy nem, főleg akkor, ha azt állítja, hogy a munkából hazaérkezvén fedezte fel a betörést, órákkal azután, amikor az állítólagosan történt.

Szintén probléma az aláírási rendszerrel kapcsolatban, ha Aliz úgy dönt, hogy lecseréli egyéni kulcsát, ami teljesen törvényes, és időről időre ajánlott is. Előfordulhat, később egy a fent leírt bírósági ügyben, hogy a bíró visszakódolja $D_A(P)$-t az aktuális E_A-val,

majd megállapítja, hogy nem P az eredmény. Bob ekkor igen kínosan érezné magát. Következésképpen valami hiteles szerv mégiscsak kell, hogy feljegyezzék a kulcscseréket és azok időpontjait.

Gyakorlatilag bármely nyilvános kulcsú titkosító algoritmus használható a digitális aláírásokhoz. A de facto hivatalos szabvány az RSA-algoritmus. Sok titkosítást használó termék alapul ezen. Mindazonáltal 1991-ben a NIST (National Institute of Standards and Technology) az El Gamal nyilvános kulcsú algoritmus egy variációjának használatát javasolta új **digitális aláírás szabványukban (Digital Signature Standard, DSS)**. Az El Gamal biztonságát nem a nagy számok prímtényezős felbontása, hanem a diszkrét logaritmus számolásának kivitelezhetetlenségéből nyeri.

Ahogyan az lenni szokott, ha a kormányzat szeretné a digitális titkosító szabványokat diktálni, most is nagy felhördülés volt. A DSS-t a következők miatt kritizálták:

1. Túlságosan titkos (az NSA az El Gamalra alapozva fejlesztette ki).
2. Túlságosan új (az El Gamal kimerítő vizsgálata még nem fejeződött be).
3. Túlságosan lassú (10–40-szer lassúbb a digitális aláírások ellenőrzése, mint az RSA-ban).
4. Nem elég biztonságos (fix 512 bit hosszúságú kulcs).

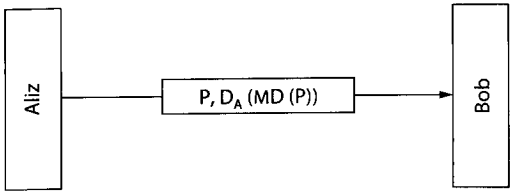
Egy ezt követő, átdolgozott kiadás révén a 4-es pont tárgytalanná vált, mert engedélyezték a maximum 1024 bites kulcsokat is. Az első két pont azonban még mindig érvényes.

8.4.3. Üzenetpecsétek

Az aláírások egyik kritikája, hogy gyakran párosítanak két eltérő funkciót: a hitelesítést és a titkosítást. Gyakran csak a hitelességvizsgálatra van szükség, és a titkosításra nem. Mivel a titkosítás lassú, ezért gyakran van igény aláírt kódolatlan dokumentumok küldésére. Az alábbiakban egy olyan hitelesítési módszert mutatunk be, amelyhez nem kell az egész üzenetet titkosítani

Ez a módszer egy egyirányú hash-függvényen alapszik, amely egy tetszőlegesen hosszú szövegből fix hosszúságú bitfüzért generál. Ez a hash-függvény, amelyet gyakran **üzenetpecsétnek (message digest)** neveznek, négy fontos tulajdonsággal bír:

1. Adott P-hez könnyen számolható $MD(P)$.
2. Adott $MD(P)$-hez gyakorlatilag lehetetlen H_0-t megtalálni.
3. Senki sem képes két különböző üzenetet generálni (P-t és P'-et), amelyekhez ugyanaz az üzenetpecsét tartozik ($MD(P) = MD(P)$).
4. A bemeneten még 1 bit megváltozása is teljesen más kimenetet eredményez.



8.20. ábra. *Digitális aláírások üzenetpecsétek használatával*

A 3. feltétel teljesítéséhez a pecsétnek legalább 128 bitesnek vagy még hosszabbnak kell lennie. A 4. feltételhez az szükséges, hogy a hash nagyon alaposan szétszórja a biteket, akárcsak az eddig látott szimmetrikus kulcsú algoritmusok esetében.

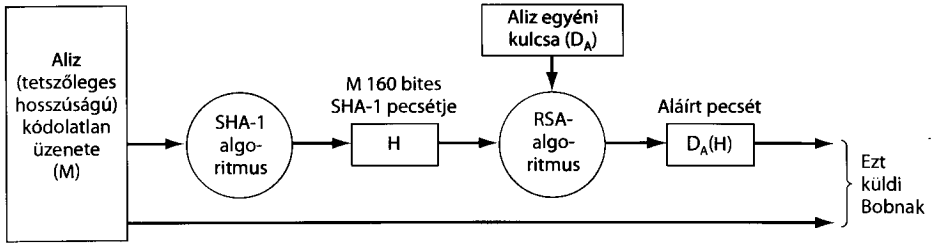
Az üzenetpecsétek kiszámítása egy szöveghez sokkal gyorsabban elvégezhető, mint ugyanazon szöveg kódolása egy nyilvános kulcsú algoritmussal, tehát az üzenetpecsétek használhatók a digitális aláírás algoritmusok gyorsítására. Ahhoz, hogy lássuk ez hogyan is működik, tekintsük ismét a 8.18. ábrát. Ahelyett, hogy *BB* a *P*-t $K_{BB}(A, t, P)$-vel írná alá, most kiszámolja az üzenetpecsétet, azaz *MD*-t kiszámolja *P*-re, aminek eredménye *MD*(*P*). *BB* ezek után $K_{BB}(A, t, P)$ helyett becsomagolja $K_{BB}(A, t, MD(P))$-t, mint ötödik elemet a K_B-vel titkosított listába, amelyet Bobnak küld.

Amennyiben vita támad, Bob előveheti mind *P*-t, mind $K_{BB}(A, t, MD(P))$-t. Miután ezt a Nagy Testvér visszakódolta a bíróságnak, Bob birtokában van a garantáltan eredeti *MD*(*P*)-nek és az állítólagos *P*-nek. Bárhogy is van, mivel gyakorlatilag lehetetlen, hogy Bob egy másik üzenetet találjon, aminek szintén ez a pecsétje, a bíró könnyen meggyőzhető arról, hogy Bob igazat mond. Az üzenetpecsétek ilyen módon történő használata mind a kódolási idő, mind az üzenet átvitel és tárolási költség szempontjából megtakarítást jelent.

Az üzenetpecsétek a nyilvános kulcsú titkosító rendszerekben is használhatók, amint az a 8.20. ábrán látható. Itt Aliz először kiszámolja az üzenetpecsétet saját szövegéhez. Ezt követően aláírja az üzenetpecsétet, és elküldi Bobnak mind az aláírt pecsétet mind a kódolatlan szöveget. Ha Trudy útközben kicseréli *P*-t, Bob észreveszi ezt, amint maga is kiszámolja *MD*(*P*)-t.

SHA-1 és SHA-2

Az üzenetpecsét függvényekre több változatot javasolnak. A legszélesebb körben használatos üzenetpecsét-függvény az **SHA-1 (Secure Hash Algorithm 1 – 1-es típusú biztonságos hash-algoritmus)** [NIST, 1993]. Ahogy a többi üzenetpecsét, ez is a hiányzó bitekkel operál egy meglehetősen bonyolult eljárásban, ahol minden bemeneti bit minden kimeneti bitet befolyásol. Az SHA-1-et az NSA dolgozta ki és a NIST hagyta jóvá a FIPS 180-1 szabványban. A bemeneti adatokat 512 bites blokkokban dolgozza fel, és egy 160 bites üzenetpecsétet állít elő. A 8.21. ábra azt mutatja be, hogyan küldhet Aliz egy nem titkos, de aláírt üzenetet Bobnak. Itt a nyílt szöveg képezi az SHA-1 algoritmus bemenetét, így áll elő a 160 bites SHA-1 pecsét. Aliz ezután aláírja a pecsétet a saját RSA-kulcsával, majd a nyílt szöveget az aláírt pecséttel együtt elküldi Bobnak.

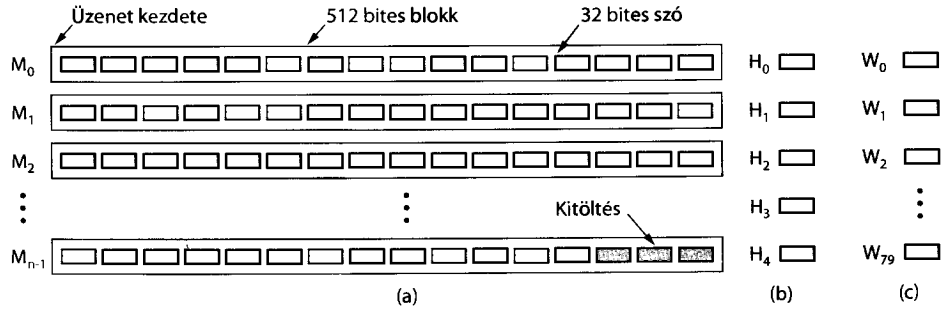


8.21. ábra. *Nem titkos, aláírt üzenetek előállítása SHA-1 és RSA alkalmazásával*

Miután Bob megkapta az üzenetet, maga is kiszámolja az SHA-1 pecsétet, valamint alkalmazza Aliz nyilvános kulcsát az aláírt pecsétre, hogy megkapja az eredeti pecsétet, *H*-t. Ha a két pecsét megegyezik, akkor az üzenet érvényesnek tekinthető. Bob könnyen észreveszi, ha Trudy bármit is átír az üzenetben, mivel Trudy sehogy sem képes átvitel közben úgy módosítani a (kódolatlan) üzenetet, hogy az új üzenet is a *H* pecsétet adja eredményül. A 8.21. ábrán látható sémát ezért széles körben használják olyan üzenetek esetén, melyeknek sérthetetlensége fontos, de tartalma nem titkos. Így viszonylag kis számítási költséggel biztosítható, hogy a nyílt szövegen végrehajtott bármilyen módosítást nagy valószínűséggel észlelni lehessen.

Most pedig nézzük át röviden, hogyan működik az SHA-1! Az algoritmus először egy 1-est tesz az üzenet végéhez, majd annyi 0-t, amennyi szükséges, de legalább 64-et, hogy az üzenet hossza 512 bit többszöröse legyen. Ezután az üzenet alsó 64 bitjének helyére az ottani bitek és az üzenet (kitöltés előtti) hosszát tartalmazó 64 bites szám VAGY kapcsolatának eredményét helyettesítik. A 8.22. ábrán az üzenet a jobb oldalról van kitöltve, mivel az angol szöveg és a számok balról jobbra olvasandók (vagyis általában a bal alsó sarkot tekintik a szám végének). A számítógépeknél ez az igazítás a felsővég gépeknek felel meg (ilyen például a SPARC, valamint az IBM 360 és az azt követő generációk), de az SHA-1 mindig az üzenet végét tölti ki, függetlenül a számábrázolás igazításától.

A számítás során az SHA-1 öt darab 32 bites változót kezel H_0-tól H_4-ig, ezekben gyűlik majd össze a pecsét. A változókat a 8.22.(b) ábra mutatja. A szabvány szerint ezeknek kezdeti érték gyanánt egy konstanst kell adni.



8.22. ábra. *(a) 512 bit többszörösére kitöltött üzenet. (b) A kimeneti változók. (c) A szavak tömbje*

Ezután M_0-tól M_{n-1}-ig sorban minden blokk feldolgozásra kerül. Az aktuális blokk 16 szavát először a W jelű, 80 szavas segédtömb elejére másolják, ahogy az a 8.22.(c) ábrán is látható. A W maradék 64 szavát a következő képlet segítségével töltik fel:

$$W_i = S^1(W_{i-3} \text{ XOR } W_{i-8} \text{ XOR } W_{i-14} \text{ XOR } W_{i-16}) \qquad (16 \le i \le 79)$$

ahol $S^b(W)$ a W 32 bites szó b bittel balra történő körkörös forgatását jelöli. Ezután A-tól E-ig öt segédváltozót inicializálnak a $H_0, \ldots, H_4$ értékeivel.

A tényleges számítást a következő pszeudo-C kóddal lehet leírni:

```
for (i = 0; i < 80; i++) {
  temp = S^5(A) + f_i(B, C, D) + E + W_i + K_i;
  E = D; D = C; C = S^30(B); B = A; A = temp;
}
```

ahol a K_i konstansokat a szabvány rögzíti. Az f_i keverőfüggvényeket a következőképp definiálták:

$$f_i(B,C,D) = (B \text{ ÉS } C) \text{ VAGY } (\text{NEM } B \text{ ÉS } D) \qquad (0 \le i \le 19)$$
$$f_i(B,C,D) = B \text{ XOR } C \text{ XOR } D \qquad (20 \le i \le 39)$$
$$f_i(B,C,D) = (B \text{ ÉS } C) \text{ VAGY } (B \text{ ÉS } D) \text{ VAGY } (C \text{ ÉS } D) \qquad (40 \le i \le 59)$$
$$f_i(B,C,D) = B \text{ XOR } C \text{ XOR } D \qquad (60 \le i \le 79)$$

Ha a ciklus mind a 80 iterációja véget ért, akkor az A–E változók értékeit hozzáadják a $H_0, \ldots, H_4$ értékeihez.

Miután így feldolgozásra került az első 512 bites blokk, jöhet a következő. A W tömböt újrainicializálják az új blokkból, de a H változók értékei megmaradnak. Ha ez a blokk is kész, jön a következő és így tovább, amíg az üzenet összes 512 bites blokkja bele nem kerül a fazékba. Amikor az utolsó blokk is készen van, a H változókban levő öt darab 32 bites szó adja meg a 160 bites kriptográfiai pecsétet. Az SHA-1 teljes C-kódját az RFC 3174 adja meg.

Az SHA-1 új változatát már kidolgozták, amelyik 224, 256, 384 és 512 bites pecsétet állít elő. Ezeket a változatokat együttesen SHA-2-nek nevezik. Ezek a pecsétek nemcsak hosszabbak az SHA-1-nél, de a hash-függvény is megváltozott annak érdekében, hogy kiküszöbölje az SHA-1 néhány potenciális gyengeségét. Az SHA-2-t széles körben még nem használják, de a jövőben minden bizonnyal fogják használni.

MD5

A teljesség kedvéért szólnunk kell egy másik népszerű pecsétről is. Ez az MD5 [Rivest, 1992], amelyik az ötödik az üzenetpecsétek sorozatában, és amelyet Ronald Rivest tervezett. Működése nagyon röviden a következő. Az üzenetet töltelékbitekkel 448 bit hosszúságra (modulo 512) növelik. Ezután ehhez az üzenet eredeti hosszát hozzáfűzi mint egy 64 bites egészet, hogy ez kiadja a teljes bemenetet, amelynek a hossza 512 bit többszöröse. Minden számítási ciklus vesz a bemenetről egy 512 bites blokkot, ezt gondosan összekeveri egy futó 128 bites puffer tartalmával. A jó eredmény érdekében belekever egy, a szinusz függvény értékeiből konstruált táblázatot is. Egy ismert függvény használata azt a célt szolgálja, hogy minden gyanút elkerüljön arra vonatkozóan, hogy a tervező egy ügyes kiskaput épített be, amelyen keresztül csak ő tud bemenni. Ez a folyamat folytatódik egészen addig, amíg az összes bemeneti blokk el nem fogy. A 128 bites puffer tartalma lesz az üzenetpecsét.

A több mint tíz éves állandó használat és tanulmányozás után az MD5-ben rejlő gyengeségek oda vezettek, hogy képesek vagyunk megtalálni az ütközéseket vagy az ugyanahhoz a hash-függvényhez tartozó különböző üzeneteket [Sotirov és mások, 2008]. Ez halálos döfés egy pecsétfüggvényre, mert ez azt jelenti, hogy képtelenek vagyunk a pecsétet biztonságosan használni egy üzenet hitelesítésére. Így a biztonsági kérdésekkel foglalkozó szakemberek az MD5-öt feltörhetőnek tartják. Helyettesíteni kell, ahol lehet, mással, és új rendszerekbe nem szabad ezzel tervezni. Ettől függetlenül, az MD5-tel még találkozhatunk a meglévő rendszerekben.

8.4.4. A születésnap-támadás

A titkosítás világában semmi sem az, aminek tűnik. Az ember azt hiheti, hogy egy m bites üzenetpecsét kicselezése nagyságrendileg 2^m műveletet igényel. Gyakorlatilag azonban sokszor $2^{m/2}$ művelet is elég, amennyiben a születésnap-támadást használjuk, amelyet Yuval [1979] publikált a ma már klasszikusnak számító cikkében a „Hogyan játsszuk ki Rabint"-ban.

Ennek a támadásnak az alapja egy, a matematikaprofesszorok által valószínűség-számítás előadásokon gyakran használt módszer. A kérdés: Hány embernek kell egy osztályban lenni ahhoz, hogy annak a valószínűsége, hogy két ember ugyanazon a napon született, meghaladja az 1/2-et? A legtöbb hallgató messze több mint 100-ra tippel megoldásként. Valójában azonban a valószínűség-számítás szerint csak 23. Anélkül, hogy precíz bizonyítást adnánk, szemléletesen 23 emberre $(23 \times 22)/2 = 253$ különböző párt alkothatunk, és minden pár találati valószínűsége 1/365. Ebben a megközelítésben nem is olyan meglepő a tény.

Általánosabban, ha a bemenet és a kimenet között létezik egy megfeleltetés, n bemenet (ember, üzenet stb.) és k lehetséges kimenet (születésnap, üzenetpecsét stb.) esetén, akkor $n(n-1)/2$ bemeneti pár van. Ha $n(n-1)/2 > k$, akkor annak az esélye, hogy legalább egy megfelelő párt találunk, elég nagy. Így megközelítőleg $n > \sqrt{k}$ esetén számíthatunk találatra. Ez az eredmény azt jelenti, hogy egy 64 bites üzenetpecsét nagy valószínűséggel feltörhető, ha generálunk 2^{32} db üzenetet, és keresünk kettőt, aminek ugyanaz az üzenetpecsétje.

Nézzünk meg egy gyakorlati példát. Az Állami Egyetem Informatika karán egy végleges kari oktatói állásra két jelentkező van, Tom és Dick. Tomot két évvel korábban vették fel, mint Dicket, először tehát őt bírálják el. Ha megkapja az állást, akkor Dick pórul jár. Tom tudja, hogy a kar elnöke, Marilyn, jó véleménnyel van a munkájáról, tehát megkéri, hogy írjon ajánlást a dékánnak, aki majd Tom ügyét el fogja bírálni. Minden levél bizalmassá válik, miután elküldték.

Marilyn utasítja titkárnőjét, Ellent, hogy írjon egy levelet a dékánnak, kiemelve azt, amit szeretne benne látni. Amikor a levél elkészül, Marilyn majd át fogja nézi, ki fogja számítani és alá fogja írni a 64 bites pecsétet, majd el fogja küldi a dékánnak. Ellen a levelet később e-mailben elküldheti. Tom számára a helyzet szerencsétlen, mert Ellen romantikus kapcsolatban van Dickkel, és át akarja verni Tomot, tehát az alább látható levelet írja, 32 szögletes zárójellel ellátott választási lehetőséggel.

Kedves Smith Dékán Úr,
Ez [*a levél* | *az üzenet*] [*őszinte* | *nyílt*] véleményemet tolmácsolja Tom Wilson professzorról, aki [*éppen* | *ebben az évben*] véglegesítésre *jelölt* | *van jelölve*]. [*Ismerjük egymást* | *Együtt dolgoztam*] Wilson professzorral [*már* | *majdnem*] hat éve. [*Kiváló* | *Kiemelkedő*] [*tehetséggel* | *képességekkel*] megáldott kutató, aki [*világszerte* | *nemzetközileg*] ismert [*briliáns* | *kreatív*] meglátásaiért [*sok* | *a legkülönbözőbb*] [*nehéz* | *bonyolult*] problémakörben.
Emellett [*nagyon* | *kiemelkedően*] [*tisztelt* | *csodált*] [*tanár* | *oktató*]. Hallgatói [*jó* | *kitűnő*] véleménnyel vannak [*óráiról* | *előadásairól*]. [*Tanszékünkön* | *Nálunk*] a [*legnépszerűbb* | *legkedveltebb*] [*tanár* | *előadó*].
[*Sőt mi több* | *Ezenfelül*] Wilson professzor [*tehetséges* | *hatékony*] alapítványi pályázó. [*Szerződései* | *alapítványi pénzei*] [*sok* | *számot tevő*] pénzt hoztak [*nekünk* | *a tanszékünknek*]. Ez a [*pénz* | *támogatás*] [*lehetővé tette* | *engedte meg*] számunkra, hogy sok [*speciális* | *fontos*] programokkal [*foglalkozhassunk* | *dolgozhassunk*], [*mint például* | *többek között*] az ön State 2000 programjával. Ezen pénzek nélkül nem [*lennénk képesek* | *tudnánk*] folytatni tevékenységünket, ami pedig mindkettőnknek [*fontos* | *lényeges*]. Azt tanácsolom, hogy véglegesítse őt.

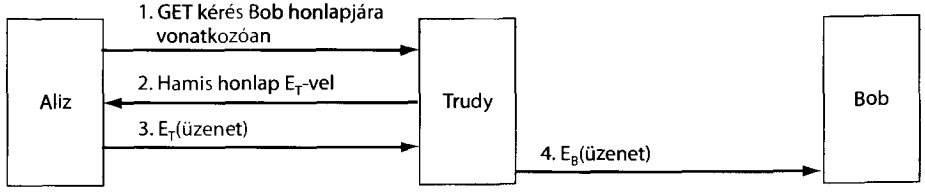
Tom pechére, amint Ellen befejezte a levél fogalmazását és begépelését, rögtön ír egy másikat is.

Kedves Smith Dékán Úr,
Ez [*a levél* | *az üzenet*] [*őszinte* | *nyílt*] véleményemet tolmácsolja Tom Wilson professzorról, aki [*éppen* | *ebben az évben*] véglegesítésre [*jelölt* | *van jelölve*]. [*Ismerjük egymást* | *Együtt dolgoztam*] Professzor Wilsonnal [*már* | *majdnem*] hat éve. [*Gyenge* | *szegényes*] képességekkel bíró kutató, és nem nagyon ismert a [*szakterületén* | *tevékenységi körében*]. Kutatásai [*ritkán* | *csak elvétve*] tartalmaznak jó [*meglátásokat* | *észrevételeket*] [*napjaink* | *az aktuális*] problémáinak [*fő* | *kulcs*] gondolatával kapcsolatban.
Továbbá nem kifejezetten [*kedvelt* | *tisztelt*] [*előadó* | *tanár*]. [*Előadásairól* | *Óráiról*] [*rossz* | *pocsék*] véleménnyel vannak hallgatói. [*Tanszékünkön* | *Nálunk*] a legnépszerűtlenebb [*előadó* | *tanár*], aki [*főleg* | *elsősorban*] azon [*tulajdonságáról* | *hajlamáról*] ismert [*nálunk* | *tanszékünkön*], hogy [*zavarba* | *kellemetlen helyzetbe*] hozza azokat a diákokat, akik kérdezni [*mernek* | *merészelnek*] óráin.
[*Sőt mi több* | *Ezenfelül*] Tom [*gyenge* | *átlagon aluli*] pénzszerző. [*Ösztöndíjai* | *Szerződései*] csak [*kevés* | *elenyésző*] pénzt hoznak [*nekünk* | *tanszékünknek*]. Ha nem találunk gyorsan új [*pénzbevételi* | *anyagi*] forrást elképzelhető, hogy le kell állítanunk néhány fontos programunkat, mint például az ön State 2000 programját. Sajnos ezen [*körülmények* | *helyzet*] mellett nem tudom [*becsülettel* | *tiszta lelkiismerettel*] ajánlani [*véglegesítésre* | *végleges pozícióra*].

Ezek után Ellen beprogramozza a számítógépét, hogy az éjszaka számolja ki mind a 232 üzenetpecsétet mindkét üzenethez. Jó esély van rá, hogy az első levél pecsétjei közül egy megegyezik a második levél egyik pecsétjével. Ha nem, akkor hozzáadhat még egy-két lehetőséget, és éjszaka újra próbálkozhat. Tegyük fel, hogy talál két egyezőt. Nevezzük a jó levelet *A*-nak, a rosszat pedig *B*-nek.
Ellen ezután jóváhagyás végett elektronikus levélben elküldi az *A* levelet Marilynnek. A *B* levelet titokként kezeli és senkinek nem mutatja meg. Marilyn természetesen jóváhagyja azt, és kiszámítja a 64 bites üzenetpecsétet, aláírja azt, és az aláírt levelet elküldi Smith dékán úrnak. Függetlenül ettől, Ellen a *B* levelet elektronikus levélben elküldi a dékánnak (nem pedig az *A*-t, amire megkérték).
A dékán, miután megkapta az elektronikus levelet és az aláírt pecsétet, lefuttatja az üzenetpecsét algoritmust a *B* levélen és látja, hogy az megegyezik a Marilyn által küldöttel, és kirúgja Tomot. A dékán nem veszi észre, hogy Ellen volt az, aki két levelet készített ugyanazzal az üzenetpecsétteI, és küldte azt neki másként, mint ahogy azt Marilyn látta és hitelesítette. (Egy lehetséges végkifejlet: Ellen elmondja Dicknek, hogy mit tett. Dick megdöbben, és szakít vele. Ellen dühbe jön és bevallja bűnét Marilynnek. Marilyn felhívja a dékánt. Tomot végül is véglegesítik.) Az SHA-1-nél a születésnap-támadás keresztülvihetetlen, mert még a másodpercenként 1 billió üzenetpecsétes sebességgel is 32 000 évig tartana kiszámolni mind a 280 darab pecsétet két levélhez, melyek közül mindegyikben 80 variáció lehetséges, és még utána sem garantált a találat. Egy 1 millió chipből álló felhő párhuzamos működése esetén a 32 000 év lecsökkenne 2 hétre.

8.5. A nyilvános kulcsok kezelése

A nyilvános kulcsú kriptográfia lehetővé teszi, hogy azok is biztonságosan kommunikálhassanak, akik nem rendelkeznek közös kulccsal. Lehetőség nyílik továbbá az üzenetek aláírására egy megbízható harmadik fél jelenléte nélkül is. Végül, az aláírt üzenetpecsétek segítségével a vett üzenetek sértetlensége is könnyen és biztonsággal ellenőrizhető.
Van azonban egy probléma, ami felett egy kicsit átsiklottunk: ha Aliz és Bob nem ismerik egymást, akkor hogyan szerzik meg egymás nyilvános kulcsát, hogy megkezdhessék a kommunikációt. A kézenfekvő megoldás – ti. hogy mindenki tegye fel a nyilvános kulcsát a weboldalára – a következők miatt nem működik. Tegyük fel, hogy Aliz meg akarja keresni Bob nyilvános kulcsát Bob weboldalán. Hogyan teszi ezt meg? Először is begépeli Bob URL-jét. A böngészője ekkor kikeresi Bob honlapjának DNS-címét, majd



8.23. ábra. *Így zavarhatja meg Trudy a nyilvános kulcsú titkosítást*

elküld egy *GET* kérést, amint azt a 8.23. ábra mutatja. Sajnálatos módon Trudy elfogja ezt a kérést, és egy hamis honlappal válaszol, ami valószínűleg Bob honlapjának másolata lesz, de nem Bob, hanem Trudy nyilvános kulcsát fogja tartalmazni. Amikor Aliz ezután kódolja az első üzenetét E_T-vel, Trudy visszafejti azt, elolvassa, újra titkosítja Bob nyilvános kulcsával, majd elküldi Bobnak, akinek fogalma sincs arról, hogy Trudy olvassa a beérkező üzeneteit. Sőt még ennél is rosszabb, hogy Trudy módosíthatja is az üzeneteket, mielőtt újra kódolná azokat Bob számára. Nyilvánvalóan szükség van tehát valamilyen eljárásra a nyilvános kulcsok biztonságos cseréjéhez.

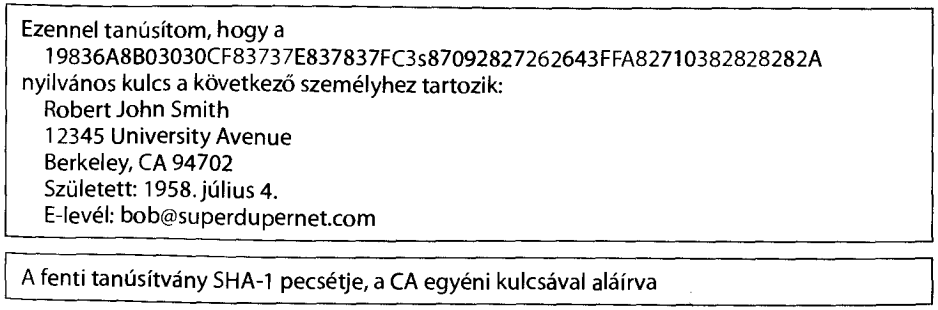
8.5.1. Tanúsítványok

Első kísérletünk a nyilvános kulcsok biztonságos szétosztására az lehetne, hogy egy **KDC**-t (**Key Distribution Center – kulcselosztó központ**) kellene felállítani, mely 24 órás online üzemben, kérésre adná a nyilvános kulcsokat. Ezzel a megoldással több probléma is van, például nem skálázható, így a kulcselosztó központ hamar szűk keresztmetszetté válna. Ráadásul, ha a központ egyszer meghibásodna, akkor hirtelen az egész internetes biztonság odalenne.

Emiatt egy másik megoldást fejlesztettek ki, egy olyat, melyben a kulcselosztó központnak nem kell folyamatosan online lennie, sőt egyáltalán nem is kell online lennie. A központ itt csak annyit tesz, hogy hitelesíti az egyes személyekhez, vállalatokhoz és más szervezetekhez tartozó nyilvános kulcsokat. Az olyan szervezeteket, melyek a nyilvános kulcsokat hitelesítik, **CA**-nak (**Certification Authority – tanúsító hatóság**) nevezzük.

Példának okáért tegyük fel, hogy Bob szeretné lehetővé tenni, hogy Aliz és más emberek biztonságosan kommunikálhassanak vele. Elmehet tehát a CA-hoz, ahol benyújtja a nyilvános kulcsát az útlevelével vagy a jogosítványával együtt, és egy tanúsítványt kér. A CA erre egy, a 8.24. ábrán láthatóhoz hasonló tanúsítvány állít ki, amit saját kulcsának segítségével egy SHA-1 pecséttel lát el. Bob ezután befizeti a CA által kért díjat, és megkapja a floppylemezt, ami a tanúsítványt és az aláírt pecsétet tartalmazza.

A tanúsítvány alapvető feladata az, hogy hozzárendelje a nyilvános kulcsot a főszereplő (személy, vállalat stb.) nevéhez. Maguk a tanúsítványok nem titkosak és nem védettek. Bob úgy is dönthet például, hogy felrakja az új tanúsítványát a weboldalára, és



8.24. ábra. *Egy lehetséges tanúsítvány és az aláírt pecsétje*

a főoldalon elhelyez egy ilyen linket: „Kattintson ide, és megkapja a nyilvános kulcsú tanúsítványomat". Kattintás után tehát megkapnánk a tanúsítványt és az aláírásblokkot (a tanúsítvány aláírt SHA-1 pecsétjét) is.

Menjünk most újra végig a 8.23. ábra forgatókönyvén! Mit tehet Trudy, amikor elkapja Aliz Bob honlapjára vonatkozó kérését? Rárakhatja saját tanúsítványát és aláírásblokkját a hamis oldalra, de ha Aliz meglátja a tanúsítványt, rögtön tudni fogja, hogy nem Bobbal beszél, mert abban nem szerepel Bob neve. Trudy röptében is módosíthatja Bob honlapját, kicserélve Bob nyilvános kulcsát a sajátjával. Csakhogy amikor Aliz lefuttatja az SHA-1 algoritmust a tanúsítványon, akkor olyan pecsétet kap, ami nem egyezik meg azzal, amit akkor kap, ha a CA jól ismert nyilvános kulcsát alkalmazza az aláírásblokkra. Trudy nem rendelkezik a CA egyéni kulcsával, ezért sehogy sem tud olyan aláírásblokkot előállítani, amelyik az ő nyilvános kulcsát magában foglaló módosított weboldal pecsétjét tartalmazná. Ily módon Aliz biztos lehet abban, hogy Bob nyilvános kulcsával rendelkezik, nem pedig Trudyével vagy valaki máséval. Emellett, ahogy ígértük, ez a séma nem igényli azt, hogy a CA online ellenőrizhető legyen, ezáltal megszűnik egy esetleges szűk keresztmetszet.

A tanúsítványok szokásos feladata a nyilvános kulcsok és a főszereplők egymáshoz rendelése, de ezenkívül arra is fel lehet használni őket, hogy egy nyilvános kulcshoz egy **attribútumot** (**attribute**) rendeljenek. Egy tanúsítvány például azt is kimondhatja: ez a nyilvános kulcs olyasvalakihez tartozik, aki már elmúlt 18 éves. Használható tehát például annak igazolására, hogy a kulcs tulajdonosa már nem kiskorú, így engedélyezett számára a nem gyerekeknek szánt tartalomhoz való hozzáférés stb. Mindez a tulajdonos személyazonosságának felfedése nélkül lehetséges. Egy ilyen tanúsítványt az illető jellemzően egy olyan weboldalnak, főszereplőnek vagy folyamatnak küld el, amelynél fontos az életkor. Az adott oldal, főszereplő vagy folyamat erre előállít egy véletlen számot, és titkosítja azt a tanúsítványban található nyilvános kulccsal. Ha a tulajdonosnak sikerül ezt dekódolnia és az eredményt visszaküldenie, akkor az azt bizonyítja, hogy az illető tényleg rendelkezik a tanúsítványban kiállított attribútummal. Vagy egy másik megoldás lehet az is, hogy a véletlen számot egy viszonykulcs előállítására használják fel, mely az azt követő üzenetváltást fogja biztosítani.

A tanúsítvány például egy objektumorientált elosztott rendszerben is tartalmazhat attribútumot. Rendszerint minden objektumnak több metódusa van. Az objektum tulajdonosa minden ügyfelének adhat egy tanúsítványt, mely egy olyan bittérképet tartalmaz, ami megadja, hogy az adott ügyfél melyik metódusokat hívhatja meg. Magát a bittérképet pedig egy nyilvános kulcshoz lehet rendelni egy aláírt tanúsítvány segítségével. Ha a tanúsítvány tulajdonosa igazolni tudja, hogy rendelkezik a megfelelő egyéni kulccsal, akkor végrehajthatja a bittérkép által megjelölt műveleteket. Itt is megtalálhatjuk tehát azt a tulajdonságot, hogy a tulajdonos kilétének ismeretére nincs szükség, ez pedig jól jön azokban a helyzetekben, ahol fontosak a személyiségi jogok.

8.5.2. X.509

Ha mindenki a saját típusú tanúsítványát szeretné aláíratni a CA-val, akkor a különböző formátumok kezelése hamar gondot okozna. Épp ezért kidolgoztak egy tanúsítványokra vonatkozó szabványt, amit az ITU is elfogadott. A szabvány neve **X.509**, és az interneten

Mező	Jelentése
Verzió	Az X.509 verziója
Sorszám	Ez a szám a tanúsító hatóság nevével együtt egyértelműen azonosítja a tanúsítványt
Aláírás algoritmusa	A tanúsítvány aláírásához használt algoritmus
Kiállító	A tanúsító hatóság X.500-as neve
Érvényességi időszak	Az érvényességi időszak kezdete és vége
Alany neve	Az az entitás, akinek a kulcsát tanúsítják
Nyilvános kulcs	Az alany nyilvános kulcsa, és az azt használó algoritmus azonosítója
Kiállító azonosítója	Opcionális azonosító, mely egyértelműen azonosítja a tanúsítvány kiállítóját
Alany azonosítója	Opcionális azonosító, mely egyértelműen azonosítja a tanúsítvány alanyát
Kiegészítések	Számos kiegészítést definiáltak
Aláírás	A tanúsítvány aláírása (a tanúsító hatóság egyéni kulcsával aláírva)

8.25. ábra. *Az X.509-es tanúsítvány legfontosabb mezői*

széles körben használják. Szabványosításának kezdete, vagyis 1988 óta már három verziót élt meg, mi ezek közül a harmadikat tárgyaljuk.

Az X.509-re erős hatással volt az OSI világa, olyannyira, hogy át is vett néhányat az OSI legrosszabb tulajdonságaiból (például a névkezelést és a kódolást). Meglepő módon az IETF is egyetértett az X.509-cel, bár a gépek címzésétől kezdve a szállítási protokollokon át az e-levél formátumokig szinte minden más területen általában figyelmen kívül hagyta az OSI-t, és a maga útját járta. Az IETF-féle X.509-et az RFC 5280 írja le.

Az X.509 lényegében a tanúsítványok leírására ad módot. A tanúsítványok legfontosabb mezőit a 8.25. ábra sorolja fel. Reméljük, az ábra megjegyzéseiből mindenkinek sikerül képet alkotni az egyes mezők funkcióiról, aki pedig még többet szeretne tudni, az tanulmányozza magát a szabványt vagy az RFC 2459-öt.

Például, ha Bob a kölcsönökért felelős részlegben dolgozik a Pénzes Banknál, akkor az X.500-as címe valami ilyesmi lehet:

```
/C=US/O=PenzesBank/OU=Kolcson/CN=Bob/
```

ahol a *C* az országot, *O* a szervezetet, *OU* a szervezeti egységet, a *CN* pedig a hétköznapi nevet jelenti. A CA-kat és más entitásokat is hasonló módon nevezik meg. Az X.500-as nevek egyik legnagyobb problémája az, hogy ha Aliz a *bob@penzesbank.com* címmel szeretné felvenni a kapcsolatot, akkor az X.500-as nevet tartalmazó tanúsítványból nem biztos, hogy egyértelműen ki tudja deríteni, hogy a tanúsítvány vajon arra a Bobra vonatkozik-e, akire ő gondolt. Szerencsére a 3. verziótól kezdve már DNS-neveket is lehet használni az X.500-as nevek helyett, ez a probléma tehát idővel megszűnhet.

A tanúsítványokat az OSI **ASN.1 (Abstract Syntax Notation 1 – absztrakt szintaxisjelölés 1)** szerint kódolják, ami voltaképpen egy C struktúrához hasonló, de nagyon különös és terjengős jelölésmód. Az X.509-ről többet is olvashatunk Ford és Baum [2000] munkájában.

8.5.3. Nyilvános kulcs infrastruktúrák

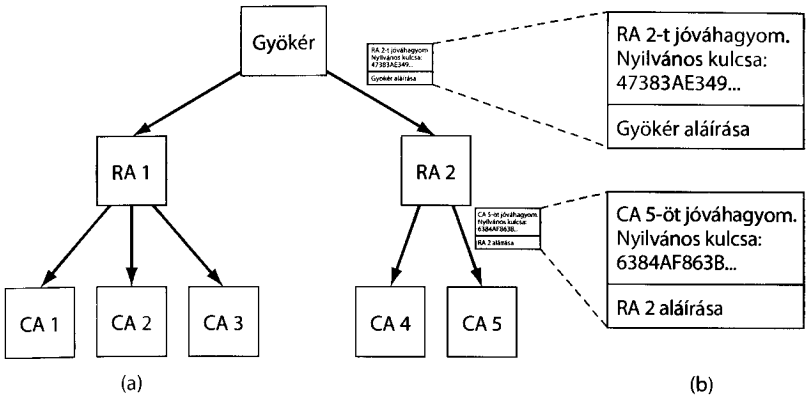
Nyilvánvalóan nem kivitelezhető az, hogy egyetlen CA adja ki a világ összes tanúsítványát, hiszen egy ilyen központ összeomlana a nagy terhelés miatt, és a hibák gócpontja is lenne. Elképzelhető lenne egy olyan megoldás is, mely szerint egyetlen nagy szervezet több CA-t is működtetne, amelyek mind ugyanazt az egyéni kulcsot használnák a tanúsítványok aláírására. Ez megoldaná a terhelés és a hibák problémáját, de újabb gonddal járna: megjelenne a kulcsszivárgás. Ha szerte a világon több tucatnyi kiszolgáló rendelkezne a CA egyéni kulcsával, akkor nagyban megnőne annak az esélye, hogy a kulcsot ellopják vagy valami egyéb módon kiszivárog. Nagyon kockázatos dolog lenne tehát egyetlen központi CA-t üzemeltetni, hiszen a kulcs gyanúba keveredése az egész világ elektronikus biztonsági infrastruktúráját romba dönthetné.

Ráadásul nagy kérdés is, hogy melyik szervezet üzemeltetné a CA-t. Nehéz elképzelni egy olyan hatóságot, amit a világon mindenhol törvényesnek és megbízhatónak tartanak. Az emberek egyes országokban ragaszkodnának ahhoz, hogy a kormány legyen ez a szerv, máshol meg azt követelnék, hogy ne a kormány legyen.

Ezen okok miatt egy másfajta módja alakult ki a nyilvános kulcsok hitelesítésének. Az eljárás általános neve a **PKI (Public Key Infrastructure – nyilvános kulcs infrastruktúra)**. Ennek általános működését foglaljuk össze ebben a szakaszban. A témában sok új javaslat is született, a részletek tehát változhatnak még az idővel.

A PKI-nek számos összetevője van, köztük felhasználók, CA-k, tanúsítványok és könyvtárak. A PKI lényegében annyit tesz, hogy módot ad ezen összetevők szervezetbe rendezésére, és szabványokat definiál a különféle dokumentumok és protokollok számára. A PKI egyik különösen egyszerű formája a CA-k hierarchiája, amit a 8.26. ábra mutat be. A példában három szintet ábrázoltunk, de a gyakorlatban lehet ennél több vagy kevesebb szint is. A legfelső szintű CA, a gyökér a második szintű CA-kat hitelesíti, melyeket **RA**-nak (**Regional Authorities – regionális hatóságok**) nevezünk, mert egy földrajzi régiót (például országot vagy kontinenst) foghatnak át. Ez az elnevezés ugyanakkor nem szabványos, sőt valójában a fa egyéb szintjeinek neve sem igazán az. Az RA-k pedig a tényleges CA-kat hitelesítik, melyek aztán kiadják az X.509 tanúsítványokat a szervezeteknek és a magánszemélyeknek. Amikor a gyökér engedélyez egy új RA-t, akkor kiállít egy X.509 tanúsítványt, mely kimondja, hogy az RA-t jóváhagyták, valamint tartalmazza az új RA nyilvános kulcsát. A gyökér ezután aláírja a tanúsítványt, és átadja az RA-nak. Hasonlóképpen, amikor egy RA hagy jóvá egy új CA-t, akkor kiállít és aláír egy tanúsítványt, mely megerősíti a jóváhagyást és tartalmazza a CA nyilvános kulcsát.

Példánk PKI-je a következőképp működik. Tegyük fel, hogy Aliz kommunikálni szeretne Bobbal, ezért szüksége van Bob nyilvános kulcsára. Keres tehát egy, a kulcsot tar-



8.26. ábra. *(a) Hierarchikus PKI. (b) Tanúsítványok láncolata*

talmazó tanúsítványt, és talál is egyet, melyet a CA 5 írt alá. Aliz azonban még sohasem hallott a CA 5-ről, felőle az akár Bob 10 éves kislánya is lehetne. Elmehet tehát a CA 5-höz, és azt mondhatja: „Kérem, igazolja a jogosultságát!" A CA 5 erre az RA 2-től kapott tanúsítvánnyal válaszol, ami tartalmazza a CA 5 nyilvános kulcsát. Aliz a CA 5 nyilvános kulcsának birtokában meggyőződhet róla, hogy Bob tanúsítványát tényleg a CA 5 írta alá, tehát hiteles.

Feltéve persze, hogy az RA 2 szerepét nem Bob 12 éves fia játssza el. A következő lépés tehát az, hogy Aliz az RA 2-t kéri fel a jogosultságának igazolására. Válaszul egy tanúsítványt kap a gyökér aláírásával, mely tartalmazza az RA 2 nyilvános kulcsát. Aliz tehát már biztos lehet benne, hogy tényleg Bob nyilvános kulcsával rendelkezik.

De honnan ismeri Aliz a gyökér nyilvános kulcsát? Most jön a trükk! Feltételezzük, hogy a gyökér nyilvános kulcsát mindenki ismeri. Elképzelhető például, hogy a böngészőjébe előre beépítették a gyökér kulcsát.

Bob persze nagyon rendes, és nem akar Aliznak ennyi fáradságot okozni. Tudja, hogy Aliz ellenőrizni fogja a CA 5-öt és az RA 2-t, ezért hogy Aliznak ne legyen rá gondja, ő maga gyűjti össze a két tanúsítványt, és a sajátjával együtt átadja azokat Aliznak. Aliz a gyökér nyilvános kulcsának ismeretében ellenőrizheti a legfelső szintű tanúsítványt, majd az abban található nyilvános kulcs segítségével a másodikat is. Ily módon Aliznak senkivel nem kell felvennie a kapcsolatot ahhoz, hogy elvégezhesse az ellenőrzést. A tanúsítványok pedig mind alá vannak írva, ezért könnyen észre lehetne venni, ha valaki megpróbálná átírni a tartalmukat. A gyökérhez ilyen módon visszavezető tanúsítványok láncát néha **bizalmi láncnak** (**chain of trust**) vagy **tanúsítvány-útvonalnak** (**certification path**) is nevezik. Az eljárást széles körben alkalmazzák a gyakorlatban.

Természetesen még mindig kérdéses, hogy ki fogja üzemeltetni a gyökeret. A megoldás az, hogy ne egy gyökér legyen, hanem több, és mindegyikhez saját RA-k és CA-k tartozzanak. A modern böngészőkbe valójában több mint 100 gyökér nyilvános kulcsa van eleve beépítve – ezekre **bizalmi horgony** (**trust anchor**) néven is szoktak hivatkozni. Ily módon tehát nincs szükség egyetlen, világszerte bizalmat élvező hatóságra.

Ez viszont azt a kérdést veti fel, hogy hogyan tudja eldönteni a böngésző gyártója, hogy a beépített bizalmi horgonyok közül melyik megbízható és melyik nem. Végeredményben minden azon múlik, hogy a felhasználó mennyire bízik meg abban, hogy a böngésző gyártója okosan választ, és nem fogad el minden olyan horgonyt, ami hajlandó lenne megfizetni a bekerülés díját. A böngészők többsége lehetővé teszi, hogy a felhasználó megvizsgálhassa a gyökerek kulcsait (általában a gyökér által aláírt tanúsítványok formájában), és letörölhesse azokat, melyek gyanúsnak tűnnek.

Könyvtárak

A PKI további kérdése, hogy hol tároljuk a tanúsítványokat (és azok valamely bizalmi horgonyhoz visszavezető láncát). Az egyik lehetőség az, hogy minden felhasználó maga tárolja a saját tanúsítványát. Ez a megoldás biztonságos ugyan (az aláírt tanúsítványokat nem lehet módosítani anélkül, hogy észrevennék a turpisságot), de egyben kényelmetlen is. Egy másik javasolt alternatíva szerint a DNS-t is lehetne tanúsítványkönyvtárnak használni. Ha a kapcsolatfelvétel előtt Aliznak úgyis ki kell keresnie Bob IP-címét a DNS segítségével, akkor miért ne lehetne az IP-címmel együtt mindjárt Bob teljes tanúsítványláncát is visszaadni?

Egyesek szerint ez jelenti a megoldást, mások azonban szívesebben látnának külön erre a célra kijelölt könyvtárkiszolgálókat, melyeknek csak az X.509-es tanúsítványok kezelése lenne a feladatuk. Az ilyen könyvtárakban az X.500-as nevek tulajdonságai alapján lehetne kereséseket végezni. Egy ilyen könyvtárszolgáltatás elméletileg válaszolni tudna például egy ilyen kérésre: „Adja meg az összes olyan Aliz nevű személy listáját, aki egy kereskedelmi osztályon dolgozik valahol az USA-ban vagy Kanadában!"

Visszavonás

A való világ is tele van tanúsítványokkal: ilyenek például az útlevelek vagy a jogosítványok is. Ezeket a tanúsítványokat esetenként vissza is lehet vonni – egy jogosítványt például be lehet vonni ittas vezetésért vagy más közlekedési vétségekért. A digitális világban is jelentkezik ez a probléma: a kiállító szerv olykor dönthet úgy, hogy visszavonja a tanúsítványt, mert az azt birtokló személy vagy szervezet valamilyen módon visszaélt vele. A tanúsítvány visszavonható akkor is, ha kitudódott az alany egyéni kulcsa, vagy ami még rosszabb, ha a CA egyéni kulcsa keveredett gyanúba. A PKI-nek tehát foglalkoznia kell a visszavonás kérdésével. A visszahívás lehetősége bonyolítja a dolgot.

Az első lépés ebben az irányban az, hogy az egyes CA-k bizonyos időközönként kiadnak egy **CRL**-t (**Certificate Revocation List – tanúsítvány-visszavonási lista**), amely megadja az adott CA által visszavont tanúsítványok sorszámainak listáját. Mivel a tanúsítványoknak lejárati idejük is van, a CRL-eknek csak a még nem lejárt tanúsítványok sorszámait kell tartalmazniuk. A tanúsítványok ugyanis automatikusan érvénytelenné válnak, ha az idejük lejár, nincs szükség tehát a lejárt és a ténylegesen visszahívott tanúsítványok megkülönböztetésére. Akár így történt, akár úgy, többé nem használhatók.

Sajnos a CRL-ek bevezetése azt is maga után vonja, hogy a tanúsítványt használni készülő felhasználónak meg kell szereznie a CRL-t, hogy megnézze, visszavonták-e az adott tanúsítványt. Ha igen, akkor nem szabad használni. Lehetséges azonban, hogy a

tanúsítvány nem szerepel a listán, de épp a lista kiadása után visszavonták. Az igazság kiderítésének egyetlen módja tehát az, hogy megkérdezzük a CA-t. Ha később újra használni akarjuk az adott tanúsítványt, akkor megint a CA-hoz kell fordulni, hiszen lehet, hogy épp csak pár másodperce történt a visszavonás.

Tovább bonyolítja a helyzetet, hogy a visszavont tanúsítványt akár újra érvénybe is helyezhetik, például ha azért lett visszavonva, mert a tulajdonosa nem fizetett be valamilyen díjat, de később rendezte a tartozását. A visszavonások (és az esetleges újra érvénybe helyezések) kezelése miatt épp a tanúsítványok egyik legjobb tulajdonságáról kell lemondanunk, mégpedig arról, hogy a CA-val való kapcsolatfelvétel nélkül is lehetett használni azokat.

Hol lenne célszerű a CRL-eket tárolni? Jó ötlet lenne ugyanoda tenni őket, ahol maguk a tanúsítványok is vannak. Az egyik stratégia szerint a CA időnként magától kiadná a CRL-eket, a könyvtárszolgáltatások pedig a CRL-ek alapján egyszerűen törölnék a visszavont tanúsítványokat. Ha a tanúsítványokat éppen nem könyvtárakban tároljuk, akkor a CRL-eket a hálózat számos más kényelmesen elérhető helyén is el lehet helyezni. Mivel a CRL maga is aláírt dokumentum, könnyen észrevehető lenne, ha valaki esetleg belenyúlna a tartalmába.

Ha a tanúsítványoknak hosszú az élettartama, akkor a CRL-ek is hosszúak lesznek. Ha a hitelkártyák például 5 évig érvényesek, akkor az esedékes visszavonások száma sokkal nagyobb lesz, mint ha 3 havonta mindig új kártyákat adnának ki. A hosszú CRL-eket általában úgy kezelik, hogy néha kiadnak egy nagy listát, és gyakrabban kiadják a lista aktuális változásait. Ezzel csökken a CRL-ek szétosztásához szükséges sávszélesség is.

8.6. A kommunikáció biztonsága

Ezzel befejeztük a terület eszközeinek tanulmányozását. Érintettük a legfontosabb eljárásokat és protokollokat is. A fejezet hátralevő része arról szól, hogyan alkalmazzák ezeket a módszereket a gyakorlatban a hálózati biztonság érdekében. A fejezet végén megosztunk néhány gondolatot a biztonság társadalmi vonatkozásairól is.

A következő négy szakaszban a kommunikáció biztonságát vesszük szemügyre, azaz megnézzük, hogyan kerülhetnek át a bitek a forrástól a címzett csomópontba titkosan és módosítás nélkül, valamint hogyan lehet a hívatlan biteket kapun kívül tartani. Ezek a problémák persze semmi esetre sem fedik le a hálózatok biztonságának összes kérdését, de kétségkívül a legfontosabbak között vannak, érdemes tehát a vizsgálódást ezektől elkezdeni.

8.6.1. IPsec

Az IETF éveken át tudatában volt annak, hogy a biztonság hiányzik az internetről. Ezt a hiányt azonban nem volt könnyű pótolni, mert nagy viszály tört ki annak kapcsán, hogy hová is tegyék a biztonsági funkciókat. A legtöbb biztonsági szakértő úgy véli, hogy az igazi biztonságot úgy lehet elérni, ha a titkosítás és a sértetlenség ellenőrzése a végpontok között zajlik (vagyis az alkalmazási rétegben). Azaz, a forrásfolyamat titkosítja és/vagy sértetlenségi védelemmel látja el az adatokat, majd elküldi a célfolyamatnak, ahol dekódolják és/vagy ellenőrzik azokat. Ha a két folyamat között vezető úton bármilyen csalás történik (akár valamelyik operációs rendszerben is), akkor azt észlelni lehet. Ezzel a megközelítéssel az a baj, hogy így az összes alkalmazást meg kell változtatni a biztonság érdekében. Ennek fényében a második legjobb megközelítés az, ha a titkosítást a szállítási rétegbe, vagy egy új, a szállítási és az alkalmazási réteg között lévő rétegbe helyezzük. Ezáltal még mindig végponttól végpontig fog terjedni a védelem, de nem lesz szükség az alkalmazások megváltoztatására.

Az ellenkező nézet szerint a felhasználók nem értenek a biztonsághoz, és nem lesznek képesek megfelelő módon használni azt, a meglévő programokat pedig senki nem akarja majd módosítani, ezért a hálózati rétegnek kellene hitelesíteni és/vagy titkosítani a csomagokat, a felhasználók bevonása nélkül. Sokévi elkeseredett küzdelem után ez a nézet végül elég támogatást kapott ahhoz, hogy sikerüljön kidolgozni egy hálózati réteg biztonsági szabványát. A tábor egyik érve egyébként az volt, hogy a hálózati rétegbeli titkosítás még nem gátolja meg a tudatosan biztonságra törekvő felhasználókat abban, hogy a saját útjukat járják, a többieknek pedig jó hasznára lehet bizonyos mértékig.

A viszály végét az **IPsec (IP security – IP-s biztonság)** nevű tervezet jelentette, melyet többek között az RFC 2401, 2402 és 2406 ír le. Nem minden felhasználó vágyik titkosításra (mivel annak nagy a számításigénye), de ezt a funkciót mégsem tették opcionálissá. Ehelyett úgy döntöttek, hogy titkosítani mindig kell, de megengedték egy „null" algoritmus használatát is. A null algoritmus leírását, továbbá egyszerűségének, könnyű megvalósíthatóságának és nagy sebességének méltatását az RFC 2410 tartalmazza.

Maga a teljes IPsec egy többféle szolgáltatásból, algoritmusból és felbontásból álló keretrendszer. A többféle szolgáltatást az indokolja, hogy nem mindenki akarja az összes szolgáltatás állandó használatának terhét magára venni, ezért az egyes szolgáltatások „à la carte" is kérhetők. A legfontosabbak a titkosság, a sértetlenség és a visszajátszásos támadások elleni védelem (ahol a támadó visszajátssza a párbeszédet). Ezek mind a szimmetrikus kulcsú kriptográfián alapulnak, mivel a nagy teljesítmény itt létfontosságú.

A többféle algoritmus azért használható, mert attól, hogy ma biztonságosnak gondolunk egy algoritmust, a jövőben még feltörhetik azt. Azáltal, hogy az IPsec független az algoritmusoktól, a keretrendszer még akkor is fennmaradhat, ha valamelyik algoritmust valamikor később feltörik.

A többféle felbontás pedig azért alkalmazható, hogy legyen lehetőség például egy külön TCP-összeköttetés, vagy két hoszt között menő összes forgalom, vagy két biztonságos útválasztó között menő összes forgalom stb. védésére.

Az IPsec-kel kapcsolatban kissé meglepőnek tűnhet az a tény, hogy – annak ellenére, hogy az IP-rétegben van – összeköttetés-alapú. Valójában persze ez nem is olyan meglepő, hiszen bármiféle biztonság eléréséhez először egy kulcsban kell megállapodni, majd azt egy ideig használni – lényegében ez is egyfajta összeköttetés. Sok csomag esetén az összeköttetések még azok kezelésének költségeit is csökkentik. Az összeköttetéseket az IPsec környezetében **SA**-nak (**Security Association – biztonsági kapcsolat**) nevezik. Az SA egy szimplex összeköttetés a két végpont között, melyhez egy biztonsági azonosítót is rendeltek. Ha mindkét irányban biztonságos forgalomra van szükség, akkor két biztonsági kapcsolatot kell alkalmazni. Az ilyen biztonságos összeköttetéseken utazó csomagok hordozzák azokat a biztonsági azonosítókat, melyeket a kulcsok és más fontos információ kikeresésére használnak a csomag megérkezésekor.

Műszaki szempontból az IPsec-nek két fő része van. Az első két új fejrészt ír le, melyek a csomagokban a biztonsági azonosítót, a sértetlenséget biztosító adatokat és az egyéb információt hordozza. A másik rész, az **ISAKMP (Internet Security Association and Key Management Protocol – internetes biztonsági kapcsolat- és kulcskezelő protokoll)** a kulcsok kezelésével foglalkozik. A továbbiakban nem foglalkozunk az ISAKMP-vel, mivel (1) rendkívül bonyolult és (2) a legfontosabb protokollja, az **IKE (Internet Key Exchange – internetes kulcscsere)** súlyos hibákat rejt, ezért ki kell cserélni – lásd Perlman és Kaufman [2000] írását.

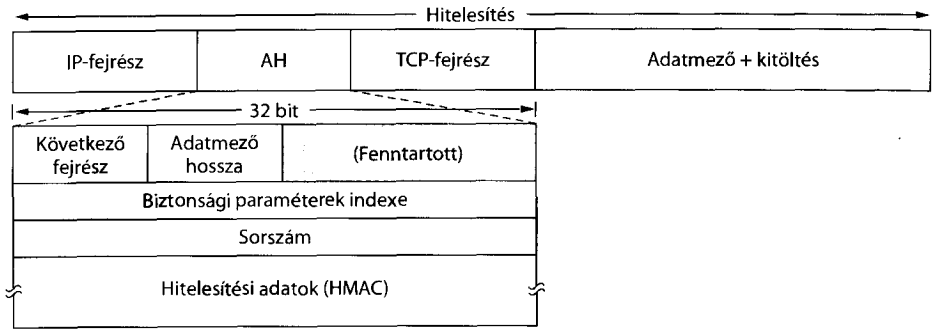
Az IPsec-et a következő két módon lehet használni. A **szállítási módban (transport mode)** az IPsec-fejrészt közvetlenül az IP-fejrész után illesztik be. Az IP-fejrész protokollmezőjét megváltoztatják oly módon, hogy jelezze azt, hogy egy IPsec-fejrész követi a rendes IP-fejrészt (a TCP-fejrészt megelőzően). Az IPsec-fejrész tartalmazza a biztonsági információt, mindenekelőtt az SA-azonosítót, egy új sorszámot, és esetlegesen az adatmező sértetlenségét ellenőrző adatokat.

Az **alagútmódban (tunnel mode)** a teljes IP-csomag fejrésszel együtt belekerül egy új IP-csomag törzsébe, mely egy teljesen új IP-fejrészt fog hordozni. Az alagútmód akkor hasznos, amikor az alagút nem a végső címzett állomásnál ér véget. Egyes esetekben az alagút egy biztonsági átjáró-számítógépben végződik, például egy vállalati tűzfalban. Egy VPN-nél (Virtual Private Network – virtuális magánhálózat) ez a szokásos eset. Így a biztonsági átjáró fogja a rajta áthaladó csomagokat be- és kicsomagolni. Ha az alagút ennél a biztonságos gépnél ér véget, akkor a vállalati LAN-on lévő gépeknek nem kell tudniuk az IPsec-ről, elég, ha a tűzfal ismeri azt.

Az alagútmód akkor is hasznos, ha egy kötegnyi TCP-összeköttetést fogunk össze és kezelünk egyetlen titkosított folyamként, mivel a támadó így nem láthatja meg, hogy ki kinek hány csomagot küldött. Néha ugyanis az is értékes információnak számít, ha tudjuk, hogy hová mennyi forgalom megy. Ha például egy katonai válság idején a Pentagon és a Fehér Ház között menő forgalom erősen visszaesik, de valamelyest megnő a Pentagon és egy mélyen a coloradói Sziklás-hegységben lévő katonai támaszpont közötti forgalom, akkor a támadó már ebből is levonhat némi következtetést. A (titkosított vagy hagyományos) csomagok áramlási mintájának elemzését **forgalomanalízisnek (traffic analysis)** nevezzük. Az alagútmód bizonyos mértékig segít meghiúsítani az ilyen kísérleteket. Ennek a módnak az a hátránya, hogy egy külön IP-fejrészt ad a csomaghoz, jelentősen megnövelve ezzel annak méretét. A szállítási mód ezzel szemben nincs ilyen nagy hatással a csomagméretre.

Az első új fejrész az **AH (Authentication Header – hitelesítési fejrész)**. Ez biztosítja a sértetlenség ellenőrzését és a visszajátszások elleni védelmet, de nem biztosít titkosságot (vagyis nem titkosítja az adatokat). Az AH használatát szállítási módban a 8.27. ábra szemlélteti. Az IPv4-nél ezt a fejrészt az (opciókkal együtt vett) IP-fejrész és a TCP-fejrész közé helyezik. Az IPv6-nál ez csak egy újabb kiegészítő fejrészt jelent, és eszerint is kezelik. Az AH formátuma valójában közelít a szabványos IPv6 kiegészítő fejrész formátumához. Ahogy az az ábrán is látható, az adatmezőt esetlegesen ki kell tölteni valamilyen adott hosszra, hogy a hitelesítő algoritmus működhessen.

Vizsgáljuk most meg az AH fejrészt! A *Következő fejrész (Next header)* mező az IP *Protokoll (Protocol)* mezőjének előző értékét tartalmazza, miután azt 51-re cserélték, annak jelzésére, hogy egy AH fejrész fog következni. Az esetek többségében ide a TCP



8.27. ábra. *Az IPsec hitelesítési fejrész szállítási módban, IPv4 esetén*

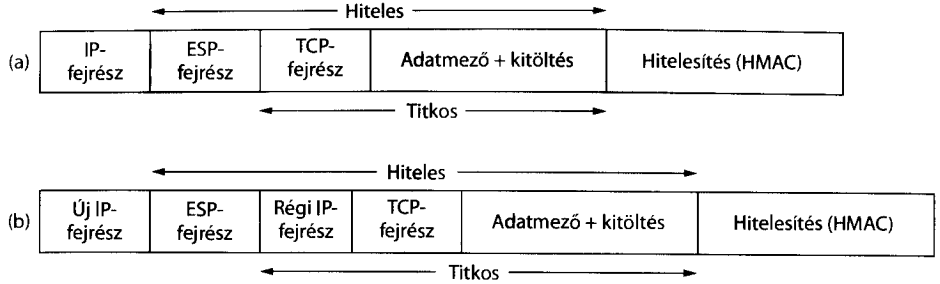
kódja (6) kerül. Az *Adatmező hossza (Payload length)* az AH fejrészben lévő 32 bites szavak száma mínusz kettő.

A *Biztonsági paraméterek indexe (Security parameters index)* maga az összeköttetés-azonosító. Ezt a feladó helyezi el, és a vevő adatbázisának egy konkrét rekordjára utal vele. Az összeköttetést leíró egyéb információ mellett ez a rekord tartalmazza a felhasznált közös kulcsot. Ha a protokollt nem az IETF, hanem az ITU fejlesztette volna ki, akkor ezt a mezőt most *Virtuális áramkör számá*-nak neveznék.

A *Sorszám (Sequence number)* mezőt az SA csomagjainak megszámozására használják. Minden csomag egyedi számot kap, még újraküldés esetén is. Más szóval, az újraküldött csomag más számot kap, mint az eredeti (még akkor is, ha a TCP sorszáma ugyanaz marad). Ez a mező a visszajátszásos támadások elleni védelmet szolgálja. Ezek a sorszámok nem fordulhatnak körbe. Ha elfogyott mind a 232, akkor egy új SA-t kell kiépíteni a párbeszéd folytatásához.

Végül elérkeztünk a *Hitelesítési adatokhoz (Authentication data)*, ami egy változó hosszúságú mező, mely az adatmező digitális aláírását tartalmazza. Amikor kiépítenek egy SA-t, a két oldal megállapodik arról, hogy melyik aláírási algoritmust fogják használni. Nyilvános kulcsú kriptográfiát többnyire nem alkalmaznak itt, hiszen a csomagokat rendkívül gyorsan kell feldolgozni, az ismert nyilvános kulcsú algoritmusok viszont túlságosan lassúak. Az IPsec szimmetrikus kulcsú kriptográfián alapul, továbbá az adó és a vevő az SA kiépítése előtt megegyeznek a közös kulcsról, az aláírás kiszámításánál így a közös kulcsot használják. Az egyik egyszerű megoldás szerint a pecsétet a csomag és a közös kulcs együttesére számítják ki. (A közös kulcsot természetesen nem viszik át.) Az ilyen sémákat **HMAC**-nek **(Hashed Message Authentication Code – hash-elt üzenethitelesítő kód)** nevezik. Sokkal gyorsabb ezt kiszámítani, mint először az SHA-1-et, majd az RSA-t futtatni az eredményen.

Az AH fejrész nem teszi lehetővé az adatok titkosítását, ezért többnyire akkor célszerű alkalmazni, amikor titkosságra nincs szükség, de a sértetlenség ellenőrzésére igen. Az AH egyik említésre méltó tulajdonsága az, hogy a sértetlenségi próba az IP-fejrész egyes mezőit is védi, mégpedig azokat, melyek nem változnak a csomag útválasztóról útválasztóra való haladása során. Az *Élettartam (Time to live)* mező például minden ugrásnál változik, ezért azt nem lehet bevonni a sértetlenségi védelembe. A forrás csomópont IP-címére viszont kiterjed az ellenőrzés, ezáltal a támadó nem tudja meghamisítani a csomag eredetét.



8.28. ábra. *(a) ESP szállítási módban. (b) ESP alagútmódban*

Az IPsec alternatívája az **ESP (Encapsulating Security Payload – beágyazott biztonsági adatmező)**. A 8.28. ábra ennek a használatát mutatja be szállítási és alagútmód esetén.

Az ESP-fejrész két darab 32 bites szóból áll. Ezek az AH-nál már látott *Biztonsági paraméterek indexe* és a *Sorszám* mezők. Ezeket általában egy harmadik szó (ami azonban műszakilag már nem része a fejrésznek), a titkosításnál használt *Inicializáló vektor (Initialization vector)* követi, hacsak nem használunk „null" titkosítást, mert ebben az esetben ez kimarad.

Az AH-hoz hasonlóan az ESP is biztosít HMAC sértetlenség-ellenőrzést, de ezt nem a fejrészben, hanem az adatmezőt követő mezőben teszi, amint azt a 8.28. ábra is mutatja. A HMAC hátravitelének a hardveres megvalósításnál van előnye. A HMAC-t ugyanis elég akkor kiszámolni, amikor a bitek már mennek kifelé a hálózati illesztőn, majd az eredményt hozzáilleszteni a sorozat végéhez. Az Ethernet és más LAN-ok ezért nem fejrészben, hanem farokrészben helyezik el a CRC ellenőrző összegüket. Az AH-nál viszont előbb pufferelni kell a csomagokat, ki kell számítani az aláírást, és csak azután lehet küldeni; ezáltal pedig esetlegesen kevesebb csomagot lehet elküldeni másodpercenként.

Tekintve, hogy az ESP mindent tud, amit az AH, sőt még többet is, ráadásul még hatékonyabb is, adódik a kérdés: mi szükség van akkor az AH-ra egyáltalán. A válasz lényegében történeti okokra vezethető vissza. Eredetileg az AH csak a sértetlenséget, az ESP pedig csak a titkosítást kezelte. Később az ESP-hez is hozzáadták a sértetlenségi védelmet, az AH-t tervező emberek viszont nem akarták, hogy az összes munkájuk kárba menjen. Egyetlen elfogadható érvük volt csak, miszerint az AH az ESP-vel ellentétben az IP-fejrész egyes mezőit is védi, de ez nem igazán nyomós érv. Egy másik gyenge érvük az volt, hogy egy olyan termék, mely az AH-t támogatja, de az ESP-t nem, esetleg könnyebben kap majd exportengedélyt, hiszen nem képes a titkosításra. Az AH valószínűleg fokozatosan meg fog szűnni a jövőben.

8.6.2. Tűzfalak

Az a képesség, hogy bármely számítógépet bárhol, bármely másik számítógéphez csatlakoztatni lehet, nem csak áldás. Azoknak, akik otthon ülve barangolnak az interneten, jó móka. A vállalati biztonsági menedzsereknek azonban rémálom. A legtöbb társaság nagy mennyiségű bizalmas információt tart online módon a hálózaton: üzleti titkokat,

termékfejlesztési terveket, marketingstratégiákat, pénzügyi elemzéseket stb. Súlyos következményekkel járhat ezeknek az információknak a felfedése egy versenytárs előtt.

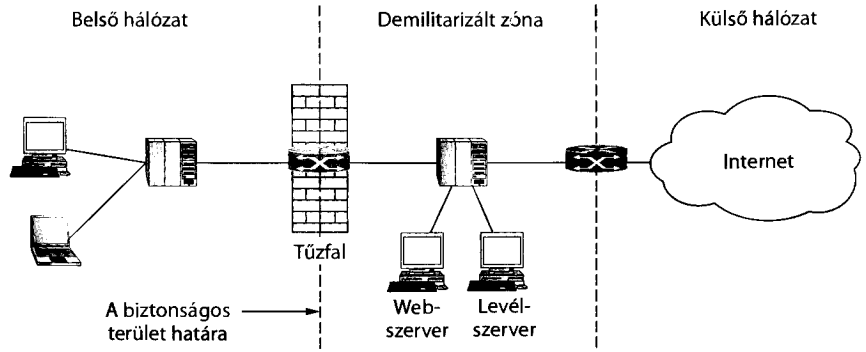
Az információ kiszivárgásának veszélye mellett fennáll az információ beszivárgásának a veszélye is. Különösen a vírusok, férgek és más digitális kártevők lékelhetik meg a biztonságot, pusztíthatnak el értékes adatokat és vesztegethetik el az adminisztrátorok idejének nagy részét, amikor azok összetakarítják a rendetlenséget, amit hagytak. Ezeket gyakran figyelmetlen alkalmazottak hozzák be, akik valami csillogó új játékot akarnak játszani.

Következésképpen olyan módszerekre van szükség, melyek segítségével a „jó" biteket bent, a „rosszakat" pedig kint tarthatjuk. Az egyik lehetséges módszer az IPsec. Ez a megközelítés a biztonságos helyek között mozgó adatokat védi, ámde semmit sem tesz azért, hogy megakadályozza a digitális kártevők és hackerek betörését a vállalati LAN-ra. A tűzfalak tanulmányozásával azonban megláthatjuk, hogyan lehet elérni ezt a célt is.

A **tűzfal (firewall)** egyszerűen a régi középkori biztonsági intézkedések modern adaptációja: egy mély árok ásása a vár körül. Ez a tervezés mindenkit, aki a várba be- vagy onnan kilépett, arra kényszerített, hogy egyetlen felvonóhídon haladjon keresztül, ahol a kapuőrség meg tudta figyelni. A hálózatokkal is lehetséges ugyanez a trükk: egy társaságnak számos LAN-ja lehet, tetszőleges módon összekötve, de minden, a társaságtól ki- vagy befelé irányuló forgalomnak egy elektronikus felvonóhídon (tűzfalon) kell keresztülhaladnia, mint ahogy a 8.29. ábrán látszik. Semmilyen más út nem létezik.

A tűzfal egy **csomagszűrőként** viselkedik. Megvizsgál minden egyes bejövő és kimenő csomagot. Azok a csomagok, amelyek megfelelnek a hálózati adminisztrátor által kialakított szabályoknak, feltételeknek, szabályosan továbbítódnak. Amelyek elbuknak a teszten, azok teketória nélkül eldobásra kerülnek.

A szűrési feltételeket jellemzően olyan szabályok vagy táblázatok segítségével adják meg, amelyek felsorolják, hogy mely források és célok elfogadhatók, melyeket kell blokkolni. Ezeken kívül vannak még alapértelmezési szabályok arra vonatkozóan, hogy mit kell csinálni más gépektől jövő vagy azokhoz menő csomagokkal. A szokásos TCP/IP-felállásban, a forrás és cél megadható IP-címekkel és portszámokkal. Például a 25-ös TCP-port a levelezéshez, míg a 80-as a HTTP-hez tartozik. Bizonyos portok egyszerűen blokkolhatók. Például egy vállalat blokkolhatja az összes IP-címmel összefüggésben a 79-es TCP-portot.



8.29. ábra. *Egy tűzfal, amely egy belső hálózatot véd*

Ez egyébként a valamikor népszerű, a felhasználók elektronikus levelezési címének lekérdezésére szolgáló Finger szolgáltatáshoz tartozott, napjainkban már kevésbé használják.

Más portok nem blokkolhatók olyan könnyen. A nehézség abból ered, hogy a hálózati adminisztrátorok biztonságra törekednek, de nem vághatják el a külvilággal folytatott kommunikációt. Ez a megoldás egyszerűbb és jobb lenne biztonsági szempontból, de végeláthatatlan felhasználói panaszkodáshoz vezetne. Ebben a helyzetben ügyes megoldás a 8.29. ábrán látható **DMZ (DeMilitarized Zone – demilitarizált övezet)**. A DMZ a vállalati hálózat olyan része, mely kívül esik a biztonsági határon. Ide bármilyen forgalom bejöhet. A demilitarizált zónában elhelyezett gépeket, például egy webszervert, az internethez csatlakozó gépek elérhetik, a vállalat weboldalát böngészhetik. Ezek után a tűzfalat be lehet úgy állítani, hogy blokkolja a 80-as TCP-portra irányuló bemeneti forgalmat, vagyis az internetről a belső hálózat gépei nem támadhatók ezen a porton. Hogy a webszerver menedzselhetőségét lehetővé tegyük, a tűzfalnak lehet olyan szabálya, amely megengedi a webszerverhez kapcsolódást a belső gépekről.

A tűzfalak egyre fejlettebbek lettek a támadókkal folytatott fegyverkezési versenyben. Eredetileg a tűzfalak egyetlen szabályhalmazt alkalmaztak minden egyes csomagra. Nehéznek bizonyult azonban olyan szabályokat írni, amelyek fenntartják a hasznos funkciókat, de blokkolják az összes nemkívánatos forgalmat. Az **állapottal rendelkező tűzfalak (stateful firewall)** hozzárendelik a csomagokat az összeköttetésekhez és a TCP/IP-fejlécmezők segítségével nyilvántartják az összeköttetések állapotát. Ez lehetővé teszi például olyan szabályok írását, melyek megengedik, hogy egy külső webszerver csomagokat küldjön egy belső hosztnak, de csak akkor, ha a belső hoszt kezdeményezte az összeköttetést a külső webszerverrel. Ilyen szabályok nem működhetnek egy állapottal nem rendelkező megoldásnál, ahol vagy átmegy, vagy eldobásra kerül minden olyan csomag, amely a külső webszervertől jön.

A tűzfalaknál az állapottal rendelkező feldolgozás utáni következő fejlettségi szint az **alkalmazásszintű átjárók (application-level gateway)** megvalósítása. Ennél a fajta feldolgozásnál a tűzfal belenéz a csomagok belsejébe a TCP-fejlécen túl is, azt kutatva, hogy mit csinál az adott alkalmazás. Ezzel a képességgel elkülöníthető a webböngészésre használt HTTP-forgalom az egyenrangú társak közötti (peer-to-peer) fájlcserélésre használt HTTP-forgalomtól. Az adminisztrátorok írhatnak olyan szabályokat, amelyek megkímélik a vállalatot az egyenrangú társak közötti fájlcseréléstől, de megengedik az üzletvitelhez fontos webböngészést. Megvizsgálható például a kimenő vagy bejövő forgalom tartalma, hogy megakadályozzák érzékeny dokumentumok kijuttatását a vállalattól.

A fenti fejtegetésből világosnak kell lennie, hogy a tűzfalak megsértik a protokollok szabványos rétegzettségét. Ezek az eszközök a hálózati rétegben működnek, de a szállítási és alkalmazási réteget is figyelembe veszik a szűréshez. Ez gyengévé teszi őket. Például a tűzfalak hajlamosak a szabványos portszámozási konvenciókra támaszkodni a csomagban szállított forgalom típusának meghatározásához. A szabványos portkiosztást ugyan gyakran használják, de nem minden számítógép és nem minden alkalmazás. Néhány peer-to-peer alkalmazás dinamikusan választ portokat, hogy nehezebb legyen felfedezni (és blokkolni). Az IPsec titkosítása és más megoldások elrejtik a magasabb rétegek információit a tűzfalak elől. Végezetül egy tűzfal nem tud kommunikálni azokkal a számítógépekkel, melyek forgalma keresztülmegy rajta, vagyis nem tudja közölni, hogy milyen szabályok érvényesek, és hogy miért bontotta le közöttük az összeköttetést.

Egyszerűen úgy tűnik, mint ha vezetékszakadás lenne. Ezekből az okokból kifolyólag a tiszta hálózati megoldások hívei a tűzfalakra az internetarchitektúra szennyfoltjaiként tekintenek. Az internet azonban a számítógép számára veszélyes hely. A tűzfalak segítenek ezen a problémán, tehát egyelőre nagy valószínűséggel maradnak.

Egy sereg biztonsági probléma merül föl azonban még akkor is, ha a tűzfal tökéletesen van konfigurálva, biztonsági problémák sokasága áll fenn. Például, ha a tűzfal úgy van beállítva, hogy csak bizonyos hálózatokból engedjen be csomagokat (például a vállalat másik telephelyeiről), egy behatoló a tűzfalon kívülről hamis forráscímek használatával megkerülheti ezt az ellenőrzést. Ha pedig egy belső alkalmazott titkos dokumentumokat akar kijuttatni, titkosíthatja vagy akár le is fényképezheti azokat, és a fotókat JPEG-fájlként küldve kijátszhatja az e-mail szűrőket. Azt a tényt pedig még nem is említettük, hogy bár a támadások háromnegyede a tűzfalon kívülről jön, a tűzfalon belülről érkező támadások, amelyek például elégedetlen alkalmazottak részéről érkeznek, tipikusan a legkártékonyabbak [Verizon, 2009].

A tűzfalakkal kapcsolatos további probléma, hogy egyszeres védelmi vonalat jelentenek. Ha ezt a védelmet áttörik, akkor minden gát szakad. Ezért a tűzfalakat gyakran egy többszintű védelem részeként alkalmazzák. Például egy tűzfal védheti a belső hálózatot, és minden gép futtathatja a saját tűzfalát. Azok az olvasók, akik úgy gondolják, hogy egyetlen biztonsági ellenőrző pont elég, biztosan nem repültek egy menetrend szerinti nemzetközi járattal mostanában.

Mindennek tetejébe a támadásoknak egy teljesen más csoportja is létezik, amellyel szemben a tűzfalak tehetetlenek. A tűzfal alapötlete az, hogy megakadályozza a támadók be-, valamint a titkos adatok kijutását. Sajnos vannak azonban olyan emberek is, akiknek nincs jobb dolguk, mint hogy megpróbáljanak egyes oldalakat térdre kényszeríteni. Ezt úgy érik el, hogy olyan nagy számban zúdítják az egyébként legális csomagjaikat a céljukra, hogy az összeomlik a terhelés alatt. Ha a támadó például egy webhelyet akar megbénítani, akkor elküldhet egy TCP *SYN* csomagot, mely az összeköttetés kiépítésére szolgál. A webhely ezután megpróbál egy táblázatbejegyzést rendelni az összeköttetéshez, és válaszul elküld egy *SYN + ACK* csomagot. Ha a támadó nem felel, a bejegyzés foglalt marad még egy pár másodpercig, mielőtt lejárna az időzítése. Ha tehát a támadó több ezer ilyen kérelmet küld el, akkor a táblázat összes bejegyzése foglalt lesz, és legális összeköttetéseket sem lehet majd létesíteni. Az ilyen támadásokat, melyekben a támadó célja nem az adatok ellopása, hanem a célgép megbénítása, **DoS (Denial of Service – szolgáltatás megtagadása)** típusú támadásoknak nevezzük. Ráadásul a kérelmet tartalmazó csomagok általában hamis forráscímet hordoznak, így a támadót nem könnyű megtalálni. A nagyobb webhelyek elleni DoS-támadások gyakoriak az interneten.

Ennél is rosszabb változat az, amikor a támadó már világszerte több száz másik számítógépbe tört be, majd mindegyiket arra utasítja, hogy egyszerre indítsanak támadást ugyanazon célpont ellen. Ez a megközelítés nemcsak a támadó tűzerejét növeli meg, de csökkenti a megtalálásának esélyét is, hiszen a csomagok nagyszámú, gyanútlan felhasználóhoz tartozó gépről érkeznek. Az ilyen támadások neve **DDoS (Distributed Denial of Service – szolgáltatás elosztott megtagadása)**. Az effajta támadások ellen nehéz védekezni. Ha a megtámadott gép még gyorsan fel is tudja ismerni a hamis kérést, akkor is időbe telik neki feldolgoznia és eldobnia azt, és ha kellően sok kérés érkezik másodpercenként, akkor a processzor minden idejét azok feldolgozásával fogja eltölteni.

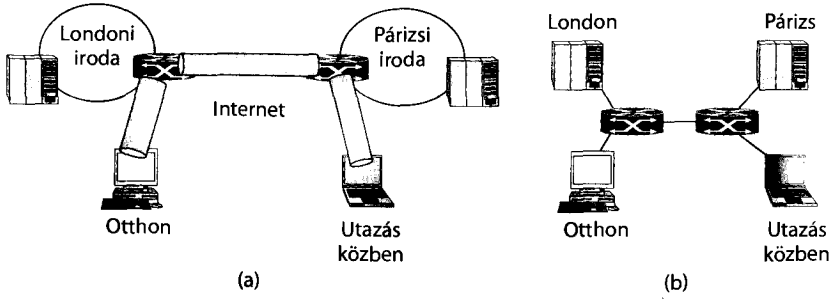
8.6.3. Virtuális magánhálózatok

Sok vállalatnak van irodája és telephelye több városban elszórtan, olykor akár különböző országokban is. A nyilvános adathálózatok előtti régi szép időkben gyakori volt, hogy az ilyen vállalatok a telefontársaságoktól béreltek vonalakat, hogy összekössék néhány vagy az összes telephelyüket. Egyes vállalatok még ma is ezt teszik. A vállalati számítógépekből és bérelt telefonvonalakból álló hálózatokat **magánhálózatnak** (**private network**) nevezzük.

A magánhálózatok jól működnek és nagyon biztonságosak. Ha a bérelt vonalakon kívül nem alkalmaznak más összeköttetést, akkor semmilyen forgalom nem szivároghat ki a vállalat telephelyeiről, a támadóknak pedig fizikailag meg kell csapolniuk a vonalakat a betöréshez, ami azért nem olyan egyszerű. A magánhálózatokkal csak az a gond, hogy egyetlen T1-es vonal bérleti díja is több ezer dollárt tesz ki havonta, a T3-as vonalak pedig még ennél is sokkal drágábbak. Amikor az internet és a nyilvános adathálózatok megjelentek, sok vállalat szerette volna átvinni az adat- (és esetleg a hang-) forgalmát a nyilvános hálózatra, mégpedig a magánhálózatok nyújtotta biztonság feladása nélkül.

Ez az igény hamar elvezetett a VPN-ek (**Virtual Private Networks – virtuális magánhálózatok**) kifejlesztéséig, amelyek a nyilvános hálózatok tetejébe épülnek, mégis rendelkeznek a magánhálózatok legtöbb tulajdonságával. A „virtuális" nevet azért kapták, mert pusztán illúziónak számítanak éppúgy, mint ahogyan a virtuális áramkörök sem igazi áramkörök és a virtuális memória sem igazi memória.

Az egyik népszerű megoldás az, hogy a VPN-eket közvetlenül az interneten keresztül kell kiépíteni. Elterjedt megoldás, hogy minden irodát felszerelnek egy tűzfallal, az irodák között páronként létrehoznak egy alagutat az interneten keresztül, ahogy az a 8.30.(a) ábrán látható. További előnye az interneten keresztül történő összeköttetésnek, hogy az alagutak igény szerint kialakíthatók egy otthon tartózkodó vagy utazó személy számítógépének bevonásával mindaddig, amíg ő internetkapcsolattal rendelkezik. Ez sokkal rugalmasabb, mint a bérelt vonalak, még a VPN-en lévő gépek szempontjából is, a topológia látszólag megegyezik a privát hálózattal, ahogy az a 8.30.(b) ábrán látható. Amikor a rendszer feláll, a tűzfalaknak páronként egyeztetniük kell SA-paramétereiket, többek között a szolgáltatásokat, működésmódokat, algoritmusokat és kulcsokat. Ha az alagutak kialakítására IPsec-et használnak, akkor lehetőség van bármely irodapárok közti forgalom összevonására (aggregálására) egyetlen hitelesített, titkosított SA-ba,



8.30. ábra. *(a) Egy virtuális magánhálózat (b) Topológia, ahogy az belülről látszik*

vagyis biztosítható az integritásellenőrzés, a titkosítás és jelentős immunitás a forgalom elemzése ellen. Sok tűzfal rendelkezik beépített VPN-képességekkel. Néhány hagyományos útválasztó is tudja ezt, de mivel a tűzfalak elsődleges feladata a biztonság megteremtése, természetes, hogy az alagutak végpontjai a tűzfalaknál vannak, tökéletesen elválasztva egymástól a vállalatot és az internetet. Így a tűzfalak, a VPN-ek és az ESP-vel alagútmódban működő IPsec természetes együttest alkotnak és a gyakorlatban széles körben használják.

Az SA kialakítása után megindulhat a forgalom. Az internetes útválasztók számára a VPN-alagúton utazó csomagok is ugyanolyanok, mint bármely más csomag. Ezekben a csomagokban az egyetlen rendhagyó dolgot az jelenti, hogy az IP-fejrész után egy IPsec-fejrész található bennük, de mivel az ilyen kiegészítő fejrészek nincsenek hatással a továbbítás folyamatára, a útválasztók nem foglalkoznak ezzel a fejrésszel sem.

A másik egyre népszerűbb megközelítés az, amikor az ISP alakítja ki a VPN-t. MPLS használatával (az 5. fejezetben tárgyalt módon) a VPN-forgalomhoz tartozó utak az ISP hálózatán keresztül felépíthetők a vállalat irodái között. Ezek az utak elválasztják a VPN-forgalmat a másfajta internetforgalomtól, és ugyanakkor egy adott sávszélesség vagy más szolgáltatásminőségi jellemző garantálható.

A virtuális magánhálózatok legfontosabb előnye, hogy az alkalmazási szoftverek számára átlátszó. A tűzfal építi fel és menedzseli az SA-kat. Az egyetlen személy, aki ennek a beállításnak a tudatában van, a rendszergazda, akinek be kell állítania és kezelnie kell a biztonsági átjárókat, vagy az az ISP-adminisztrátor, akinek az MPLS-utakat kell beállítania. Mindenki más számára ez egy bérelt vonalakból álló privát hálózatnak látszik. A VPN-ekkel kapcsolatos további információért lásd Lewis [2006] művét.

8.6.4. Vezeték nélküli biztonság

Meglepően egyszerű olyan rendszert tervezni, amely a VPN-ek és a tűzfalak révén elméletileg teljesen biztonságos, a gyakorlatban mégis olyan lyukas, mint a szita. Ez a helyzet állhat elő például akkor, ha a gépek egy része vezeték nélkül, rádiós úton kommunikál, mindkét irányban megkerülve ezzel a tűzfalat. A 802.11-es hálózatok hatótávolsága gyakran néhány száz méteres is lehet, így ha valaki kémkedni akar egy vállalatnál, elég, ha egyszerűen beáll a dolgozói parkolóba reggel, bennhagyja a kocsiban a 802.11-gyel felszerelt hordozható számítógépét, ami mindent rögzít, amit csak hall; majd a nap végén továbbáll. A merevlemez késő délutánra tele lesz értékes holmival. Elméletileg persze ilyesmi nem történhet meg. Bár elméletileg nem történhetnek meg a bankrablások sem.

A biztonsági problémák jó része a vezeték nélküli bázisállomások (hozzáférési pontok) gyártóinak azon törekvéseire vezethető vissza, hogy termékeiket minél inkább felhasználóbaráttá tegyék. Ezek az eszközök rendszerint azonnal működni kezdenek, ahogy a felhasználó kiveszi őket a dobozból és csatlakoztatja őket a villamos hálózathoz – gyakorlatilag bárminemű biztonság nélkül, az egész vételkörzetben szétkürtölve ezzel a titkokat. Ha a készüléket ezután csatlakoztatják az Ethernetre, akkor hirtelen a parkolóban is megjelenik az Ethernet teljes forgalma. A vezeték nélküli rendszerek a kémek valóra vált álmai: adatok ingyen, minden erőfeszítés nélkül. Mondani sem kell tehát, hogy a biztonság a vezeték nélküli rendszerekben még fontosabb, mint a vezeté-

kesekben. Ebben a szakaszban megnézzük néhány módját annak, hogyan kezelhetik a vezeték nélküli hálózatok a biztonságot. További információval Nichols és Lekkas [2002] műve szolgál.

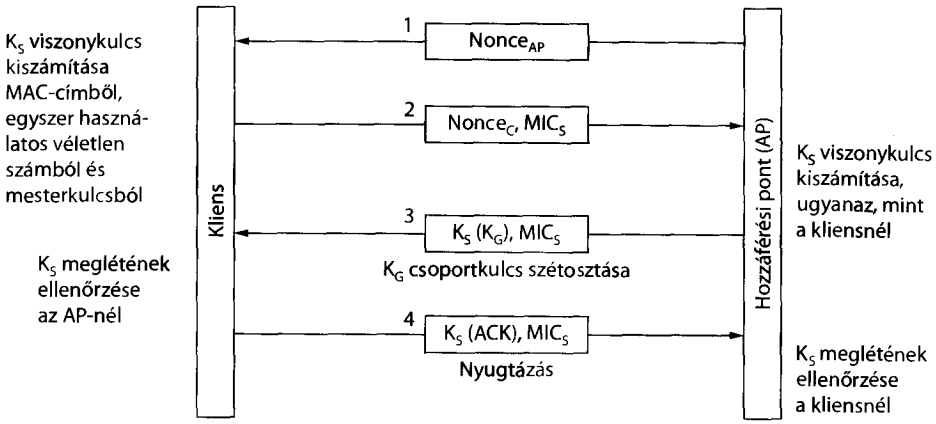
A 802.11 biztonsága

A 802.11 szabvány egy része, amely eredetileg 802.11i néven szerepelt, előír egy adatkapcsolati rétegben lévő biztonsági protokollt, mely megakadályozza, hogy a vezeték nélküli csomópontok olvassák vagy zavarják a vezeték nélküli hálózat másik csomópontpárjai közötti üzeneteket. Ez a megoldás fut még **WPA2 (Wi-Fi Protected Access 2 – Wi-Fi védett hozzáféréssel 2)** márkanéven is. A sima WPA egy átmeneti megoldás, mely a 802.11i egy részhalmazát implementálja. Használata kerülendő, jobb a WPA2.

Mielőtt bemutatjuk a 802.11i-t, meg kell jegyeznünk, hogy a **WEP (Wired Equivalent Privacy – vezetékessel egyenértékű titkosság)** leváltására jött létre, ami a 802.11 első generációs biztonsági protokollja volt. A WEP-et egy hálózati szabványokkal foglalkozó bizottság tervezte, merőben más eljárással, mint ahogy például a NIST az AES-t tervezte. Az eredmény katasztrofális volt. Mi volt a baj vele? Mint kiderült, biztonsági szempontból nagyon sok minden. Például a WEP úgy titkosította az adatokat, hogy azokat a biztonság érdekében KIZÁRÓ VAGY kapcsolatba hozta egy folyamtitkosító kimenetével. Sajnos a gyenge kulcskiosztás azt jelentette, hogy gyakori volt ennek a kimenetnek az újrafelhasználása. Ez azután triviális feltörési módokhoz vezetett. Egy másik példa, hogy az integritásellenőrzést 32 bites CRC-re alapozták. Ez ugyan hatékony kódolás átviteli hibák detektálására, de kriptográfiai szempontból egyáltalán nem erős védelem a támadók elriasztására.

Ezek és más tervezési hiányosságok a WEP-et könnyen kijátszhatóvá tették. A WEP feltörhetőségének első gyakorlati demonstrációja Adam Stubblefieldtől származik, aki akkor gyakornok volt az AT&T-nél [Stubblefield és mások, 2002]. Képes volt megírni és tesztelni egy Fluhrer és társai által vázolt támadást [Fluhrer és mások, 2001] egy hét alatt, melyből a legtöbb idő arra ment el, hogy rávegye a menedzsmentet a kísérlethez szükséges Wi-Fi-kártya megvásárlására. Manapság a WEP-jelszavakat egy percen belül feltörő szoftverek ingyenesen elérhetők, és a WEP használata erősen ellenjavallt. Bár megakadályozza az alkalmi hozzáférést, de valódi biztonságot egyáltalán nem nyújt. Amikor világossá vált, hogy a WEP komolyan feltörhető, sürgősen felállították a 802.11i csoportot, amely 2004 júliusára elő is állított egy hivatalos szabványt.

Most pedig bemutatjuk a 802.11i-t, amely megfelelően beállítva és használva valóban biztonságot nyújt. A WPA2 használatára két szokásos forgatókönyv létezik. Az első egy vállalati felállás, ahol a vállalatnak van egy különálló hitelesítő szervere felhasználóneveket és jelszavakat tartalmazó adatbázissal, amelynek segítségével eldönthető, hogy jogosult-e a vezeték nélküli kliens a hálózat használatára. Ebben az elrendezésben a kliensek szabványos protokollokat használnak hitelesítésre a hálózat felé. A fő szabvány a **802.1X**, amelynek segítségével a hozzáférési pont biztosítja a kliens és a hitelesítő szerver közötti párbeszédet, és értesül az eredményről, valamint az **EAP (Extensible Authentication Protocol – kiterjeszthető hitelesítő protokoll)** (RFC 3748), mely leírja, hogyan működjön együtt a kliens és a hitelesítő szerver. Az EAP valójában egy keretrendszer, és a protokollüzeneteket más szabványok definiálják. Nem merülünk bele azonban ezeknek az üzenetváltásoknak a részleteibe, mert nem sokkal járulnak hozzá az áttekinthetőséghez.



8.31. ábra. *A 802.11i kulcsmeghatározási kézfogás*

A másik forgatókönyv egy otthoni felállás, amelyben nincs hitelesítő szerver. Ehelyett van egy közös jelszó, amelyet a kliensek a vezeték nélküli hálózat elérésére használnak. Ez az elrendezés sokkal egyszerűbb, mint a hitelesítő szerver használata, ezért is használják otthon, illetve kis vállalkozásoknál, de kevésbé biztonságos. A fő különbség az, hogy a hitelesítő szerver használatánál a forgalom titkosítására minden kliens a többi kliens számára nem ismert kulcsot kap. Egyetlen közös, megosztott jelszó segítségével minden kliens számára különböző kulcsok származtathatók, de mivel az összes kliensnek ugyanaz a jelszava, származtathatják egymás kulcsait, ha akarják.

A forgalom titkosítására szolgáló kulcsok egy hitelesítési kézfogás részeként számítódnak ki. A kézfogásra közvetlenül az után kerül sor, hogy a kliens vezeték nélküli hálózathoz kapcsolódik és hitelesíti magát a hitelesítő szerverrel, ha van ilyen. A kézfogás kezdetén a kliens vagy rendelkezik a közös hálózati jelszóval, vagy a hitelesítő szerverhez használható saját jelszóval. Ez a jelszó felhasználásra kerül egy mesterkulcs származtatására. A mesterkulcsot azonban nem használják közvetlenül a csomagok titkosításánál. A gyakorlatban az a szabványos kriptográfiai megoldás, hogy egy viszonykulcsot (session key) kell meghatározni minden használati időszakra, a kulcsot meg kell változtatni a különböző munkamenetek között, és a mesterkulcsot a lehető legkevesebb megfigyelési lehetőségnek kell kitenni. A kézfogás alkalmával a viszonykulcs az, amelyet kiszámítanak.

A viszonykulcs négyutas meghatározása látható a 8.31. ábrán. Először az AP (access point – hozzáférési pont) küld egy egyszer használatos véletlen számot azonosítás végett. A biztonsági protokollokban a pontosan egyszer használt véletlen számokat **nonce**[2]-nak hívják, ami többé-kevésbé a „number used once" rövidítése. A kliens szintén választ egy saját nonce-t. A kliens ezután az egyszer használatos számok, a saját és a hozzáférési pont MAC-címe, valamint a mesterkulcs alapján meghatározza a K_S viszonykulcsot.

2 A nonce ebben a könyvben egy egyszer használatos véletlen szám, amely az alkalmazásban lehet tőszám, sorszám, üzenetszám, a titkosító kulcs része stb. *(A lektor megjegyzése)*

A viszonykulcsot részekre bontják, melyeket különböző célokra használnak, de ezek elhanyagolható részletek. A kliensnek most már vannak viszonykulcsai, de a hozzáférési pontnak még nincs. Ezért a kliens elküldi az egyszer használatos véletlen számát (nonce) az AP-nak, ami ugyanezen számításokkal meghatározza ugyanezeket a viszonykulcsokat. Az egyszer használatos véletlen számok elküldhetők nyílt szövegként, mivel a kulcsok nem határozhatók meg belőlük további titkos információ nélkül. A kliens üzeneteit egy a viszonykulcson alapuló integritásellenőrző védi, amelynek neve **MIC (Message Integrity Check – üzenetintegritás-ellenőrző)**. Az AP a viszonykulcs kiszámítása után ellenőrizheti, hogy az MIC rendben van-e, és így az üzenet valóban a klienstől származik-e. Az MIC csupán egy másik elnevezés az üzenethitelesítő kódra, ahogy az előfordul egy HMAC-ban is. Az MIC kifejezést gyakran használják hálózati protokollok helyett, az MAC (Medium Access Control) rövidítéssel való potenciális keveredés elkerülése érdekében.

Az utolsó két üzenetben az AP átadja a K_G csoportkulcsot a kliensnek, és az nyugtázza az üzenetet. Ezek az üzenetküldések és nyugtázások lehetővé teszik, hogy a kliens és az AP kölcsönösen ellenőrizze, hogy a másiknak megfelelő viszonykulcsa van-e. A csoportkulcsot üzenetszórásra és többesküldésre használják a 802.11 LAN-on. Mivel a kézfogás eredményeként minden kliensnek saját titkosító kulcsai lesznek, az AP nem használhatja azokat az összes kliensnek szóló csomagszórásra; minden kliensnek külön el kellene küldeni ezeket a csomagokat a saját kulcsaikkal titkosítva. Ehelyett szétosztanak egy közös kulcsot, így az üzenetszórásos forgalmat csak egyszer kell elküldeni, és azt minden kliens venni tudja. Ezt frissíteni kell minden olyan esetben, ha kliensek elhagyják a hálózatot vagy csatlakoznak hozzá.

Végül eljutunk ahhoz a részhez, amikor a kulcsok ténylegesen gondoskodnak a biztonságról. A 802.11i-ben két protokollt lehet használni üzenettitkosság, integritás és hitelesítés biztosítására. A WPA-hoz hasonlóan, a **TKIP (Temporary Key Integrity Protocol – időleges kulcsú integritási protokoll)** is átmeneti megoldás volt. A régi és lassú 802.11 kártyák biztonságának fokozására tervezték, hogy a WEP-nél egy kicsit biztonságosabb megoldással firmware javításként lehessen kijönni a piacra. Ez is hibásnak bizonyult azonban, így jobban járunk a másik javasolt protokoll, a **CCMP** használatával. Mit jelent a CCMP? Ez a látványosan hosszú név, a Counter mode with Cipher block chaining Message authentication Protocol (titkos blokkláncolású számláló módú üzenethitelesítő protokoll) rövidítése. Mi csak CCMP-nek fogjuk hívni, Ön annak hívja, aminek akarja.

A CCMP elég lényegre törően működik. AES-titkosítást használ 128 bites kulcs- és blokkhosszal. A kulcs a viszonykulcsból származik. A titkosság biztosítására az üzenetek az AES számlálómódjával kerülnek titkosításra. Emlékezzünk vissza a 8.2.3. fejezetben a titkosító módokról elmondottakra. Ezek a módok megakadályozzák, hogy titkosításkor ugyanaz az üzenet mindig ugyanazon bitsorozattá kódolódjon. A számláló mód egy számlálót kever a titkosításhoz. Az integritás biztosítása érdekében, az üzenet a fejlécmezőkkel együtt keresztülmegy titkosított blokkláncolásos kódoláson, így az utolsó 128 bites blokk megtartható MIC-nek. Ezután mind a számlálómóddal tikosított üzenet, mind az MIC átvitelre kerül. Ezt a titkosítást a kliens és az AP is elvégezheti, illetve ellenőrizheti csomag érkezésekor. Az üzenetszóráshoz és a többesküldéshez ugyanez az eljárás használható a csoportkulccsal.

A Bluetooth biztonsága

A Bluetooth-nak lényegesen rövidebb a hatótávolsága, mint a 802.11-nek, ezért ezt nem lehet a parkolóból megtámadni, de a biztonság ettől még itt is kérdés marad. Képzeljük el például, hogy Aliz számítógépe egy vezeték nélküli Bluetooth-billentyűzettel van felszerelve. Ha nem gondoskodunk a biztonságról, és Trudy épp a szomszédos irodában van, akkor mindent elolvashat, amit Aliz begépel, beleértve Aliz kimenő e-leveleit is. Megszerezhet még minden olyan dokumentumot is, amit Aliz a mellette lévő Bluetooth-nyomtatóra küldött (például beérkezett e-leveleket és bizalmas jelentéseket). Szerencsére a Bluetooth gondosan kidolgozott biztonsági sémával rendelkezik, hogy visszatartsa a világ Trudyjait. A séma főbb jellemzőit az alábbiakban összegezzük.

A 2.1-es és későbbi Bluetooth-verzióknak négy biztonsági működésmódja van, a védelem teljes hiányától a teljes adattitkosításig és adatintegritás-ellenőrzésig. Akárcsak a 802.11-nél, a kikapcsolt biztonsági funkciók mellett (a régi eszközöknél ez az alapértelmezés) nincs semmiféle védelem. A legtöbb felhasználó nem kapcsolja be a biztonsági szolgáltatásokat mindaddig, amíg komoly sérelmet nem szenved, azután már bekapcsolja. A mezőgazdaság területén ez a megközelítés úgy ismeretes, hogy valaki akkor csukja be a pajta ajtaját, miután már a ló kiszökött.

A Bluetooth több rétegben is nyújt biztonsági szolgáltatást. A fizikai rétegben a frekvenciaugrás biztosít egy kis védelmet, de mivel minden pikohálózatba belépő Bluetooth-eszközzel közölni kell ezt az ugrássorozatot, ez a sorozat nyilvánvalóan nem titkos. A valódi biztonság akkor kezdődik, amikor az újonnan érkezett szolga csatornát igényel a mesterhez. A 2.1-es Bluetooth előtt feltételezték, hogy van két olyan eszköz, amelyek egy közös titkos kulcsban előre megállapodtak. Bizonyos esetekben ezt a gyártók előre behuzalozták az eszközökbe (például egy együtt árusított fejhallgató és mobiltelefon esetében). Máskor az egyik eszköznek (például a fejhallgatónak) behuzalozott kulcsa van, amit a másik eszköznek (például mobiltelefonnak) a felhasználó decimális számként bebillentyűzhet. Ezeket a közös kulcsokat **tolvajkulcsoknak (passkey)** hívják. Sajnos a tolvajkulcsok gyakran „1234" vagy valami más könnyen megjósolható értékre vannak rögzítve, de minden esetben négy decimális számjegyből állnak, mindössze 10^4 választást téve lehetővé. Az egyszerű biztonsági párosításnál a Bluetooth 2.1-nél az eszközök hatjegyű kódot választanak, ami nehezebben kitalálhatóvá teszi a kulcsot, de még mindig messze van a biztonságostól.

A csatorna kiépítéséhez mind a szolga, mind a mester ellenőrzi, hogy a másik ismeri-e a tolvajkulcsot. Ha igen, megbeszélik, hogy a csatorna titkosított legyen-e, legyen-e integritásellenőrzés, vagy esetleg mindkettő. Ezután megállapodnak egy véletlen 128 bites viszonykulcsban, melynek néhány bitje publikus lehet. A kulcs gyengítésének az az oka, hogy megfeleljen bizonyos országok kormányzati előírásainak, ahol megakadályozzák olyan kulcsok exportját vagy használatát, amelyek hosszabbak, mint amit a kormány fel tud törni.

A titkosításra egy $\mathbf{E_0}$-nak nevezett folyamtitkosítást használnak; az integritásellenőrzésre a **SAFER+** használatos. Mindkettő hagyományos szimmetrikus kulcsú blokktitkosító. A SAFER+-t benevezték ugyan az AES-titkosítást feltörő versenyre, de az első körben kizárták, mert lassabb volt a többi pályázónál. A Bluetooth-t előbb véglegesítették, mint ahogy az AES-titkosításról megszületett a döntés; ellenkező esetben valószínűleg Rijndael-t használna.

A tényleges titkosítás a 8.14. ábrán látható folyamtitkosítót használja, a nyílt szöveget KIZÁRÓ VAGY (XOR) kapcsolatba hozza a kulcsfolyammal a titkos szöveg előállítására. Sajnos az E_0-nak magának (az RC4-hez hasonlóan) végzetes gyengeségei lehetnek [Jakobsson és Wetzel, 2001]. Bár írásunk idejéig még nem törték fel, kísérteties hasonlósága az A5/1 titkosításhoz, melynek látványos bukása az összes GSM-telefonforgalmat veszélyeztette, okot adhat az aggodalomra [Biryukov és mások, 2000]. Az embereket (beleértve e könyv szerzőit is) néha ámulatba ejti, hogy a kriptográfusok és a kriptográfiai elemzők közötti állandó macska-egér harcból oly gyakran az elemzők kerülnek ki győztesen.

További kérdést vet fel az a tény is hogy a Bluetooth csak az eszközöket hitelesíti, nem a felhasználókat. Így egy eltulajdonított Bluetooth-eszköz a tolvajnak hozzáférést biztosíthat a felhasználó pénzügyi és egyéb adataihoz. Másrészt viszont az is igaz, hogy a Bluetooth a felsőbb rétegekben is tartalmaz biztonsági funkciókat, ezért az adatkapcsolati szintű védelem áttörése után is nyújt némi biztonságot, különösen az olyan alkalmazásoknál, ahol egy PIN-kódot kell valamilyen billentyűzetről kézzel begépelni a tranzakciók végrehajtására.

8.7. Hitelességvizsgáló protokollok

A **hitelességvizsgálat** (**authentication**) olyan módszer, amivel egy folyamat ellenőrizheti, hogy kommunikációs partnere valóban az-e, akinek lennie kell, nem pedig egy csaló. Egy távoli folyamat azonosságának ellenőrzése meglehetősen nehéz, egy rosszakaratú, aktív támadó jelenlétében és titkosításon alapuló komplex protokollokat igényel. Ebben a részben megvizsgálunk néhány olyan hitelességvizsgáló protokollt, amelyek a nem megbízható számítógépes hálózatokban használatosak.

Félreértésből néhányan keverik a hitelesség- és jogosultságvizsgálat fogalmát. A hitelességvizsgálat azzal foglalkozik, hogy valóban a megfelelő folyamattal történik-e a kommunikáció. A jogosultságvizsgálat pedig a folyamat hatáskörének eldöntésére vonatkozik. Például egy fájlszerverhez fordul egy ügyfélfolyamat, és azt kéri: „Én Scott folyamata vagyok, és le akarom törölni a *cookbook.old* fájlt." A fájlszerver szempontjából két kérdés merül fel:

1. Ez tényleg Scott folyamata-e (hitelességvizsgálat)?

2. Scott letörölheti-e a *cookbook.old* fájlt (jogosultságvizsgálat)?

A kérést csak azután lehet teljesíteni, miután mindkét kérdésre egyértelműen igenlő a válasz. A hangsúly az első kérdésen van. Miután a szolgáltató tudja, kivel beszél, a jogosultságvizsgálathoz már csak meg kell nézni egy helyi táblázat megfelelő bejegyzését. Éppen ezért ebben a részben a hitelességvizsgálatra koncentrálunk.

A hitelességvizsgáló protokollok a következő általános modellt használják. Aliz azzal kezdi, hogy elküld egy üzenetet vagy Bobnak, vagy egy megbízható **kulcselosztó központnak** (**Key Distribution Center**, **KDC**). Feltesszük, hogy a központ mindig szavahihető. Ezután számos üzenetváltás történik mindkét irányban. Az elküldött üzeneteket Trudy elfoghatja, módosíthatja vagy visszajátszhatja, hogy becsapja Alizt és Bobot, vagy hogy egyszerűen csak megzavarja a munkájukat.

Mindazonáltal, amikor a protokoll tökéletes, Aliz biztos lehet, hogy Bobbal beszél, és Bob is biztos lehet, hogy Alizzal beszél. Ezenkívül, a protokollok többségében létrejön egy kettejük kapcsolatára jellemző titkos **viszonykulcs** (**session key**), amit ezután a párbeszédhez használnak. A gyakorlatban, hatékonysági okokból, minden adatforgalmat titkos kulcsú titkosítással kódolnak annak ellenére, hogy a nyilvános kulcsú titkosítás széles körben használatos magában a hitelességvizsgáló protokollban és a viszonykulcs létrehozásánál.

A véletlenszerűen választott friss viszonykulcs azért hasznos, mert használatával minimalizálni lehet a felhasználó saját titkos vagy nyilvános kulcsával kódolt adatforgalmat, ezáltal csökkentve a támadó által megkaparintható titkosított szöveget, és mert minimális lehet a kár, ha egy folyamat összeomlik, és a memóriamentés rossz kezekbe kerül. A viszony felépítése után minden visszamaradó kulcsot gondosan ki kell nullázni.

8.7.1. Osztott titkos kulcson alapuló hitelességvizsgálat

Első protokollunkhoz feltételezzük, hogy Aliznak és Bobnak már van egy osztott titkos kulcsa, K_{AB}. (A protokollok formális leírásában Alizt A, Bobot pedig B helyettesíti.) A felek a megosztott kulcsot megbeszélhetik telefonon vagy személyesen, de semmiképpen sem a (nem megbízható) hálózaton.

Ez a protokoll is azt az alapötletet használja, amit számos más hitelességvizsgáló protokoll: az egyik résztvevő egy véletlen számot küld a másiknak, aki azt speciális módon transzformálja, majd az eredményt visszaküldi. Az ilyen típusú protokollokat **kihívás-válasz** (**challenge-response**) protokolloknak nevezzük. Ebben és az ezt követő hitelességvizsgáló protokollokban a következő jelölések érvényesek:

A, B Aliz és Bob azonosítója.

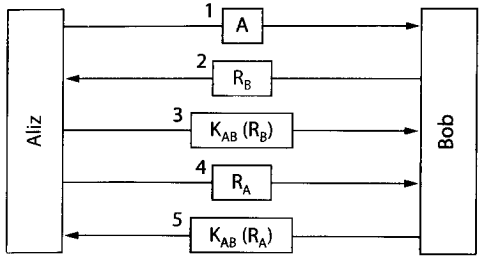
R_i-k a kihívások, ahol az i index a kihívót azonosítja.

K_i-k kulcsok, ahol i a tulajdonosra vonatkozik.

K_S a viszonykulcs.

Az első osztott kulcson alapuló hitelességvizsgáló protokollunk üzenetváltásait a 8.32. ábra mutatja. Az első üzenetben Aliz elküldi azonosítóját, A-t Bobnak, mégpedig Bob által értelmezhető formában. Bob természetesen nem tudhatja, hogy az üzenet Aliztól vagy Trudytól jött-e, ezért kiválaszt egy nagy véletlen számot, R_B-t, amit kihívásként visszaküld az állítólagos Aliznak a 2-es számú, kódolatlan üzenetben. Ezután Aliz a Bobbal közös titkos kulcs segítségével kódolja az üzenetet, majd a titkosított szöveget, $K_{AB}(R_B)$-t visszaküldi Bobnak a 3-as számú üzenetben. Amikor Bob megkapja ezt az üzenetet, már biztos lehet benne, hogy az Aliztól jött, hiszen Trudy nem ismeri K_{AB}-t, így ő nem generálhatott ilyen üzenetet. Továbbá, mivel R_B véletlenszerűen lett kiválasztva egy nagy halmazból (mondjuk 128 bites számok közül), nagyon kicsi az esélye annak, hogy Trudy egy korábbi viszonyban már láthatta az R_B-t és az arra adott választ. Éppen ilyen valószínűtlen az is, hogy bármely kihívásra csak úgy ki tudná találni a helyes választ.

Ekkor Bob már biztos benne, hogy Alizzal beszél, de Aliz még nem biztos semmiben. Amit Aliz tud, az még nem biztosítja arról, hogy Trudy nem hallgatta le az 1-es üzenetet,



8.32. ábra. *Kétirányú hitelességvizsgálat kihívás-válasz protokollal*

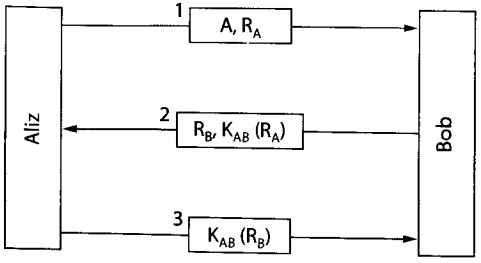
és nem ő küldte vissza R_B-t. Lehet, hogy Bob az éjszaka meghalt. Ahhoz, hogy megtudja, kivel beszél, Aliz választ egy nagy véletlen számot, R_A-t, és a kódolatlan 4-es üzenetben elküldi Bobnak. Ha Bob $K_{AB}(R_A)$-val válaszol, akkor Aliz már tudja, hogy Bobbal beszél. Ha ezek után szeretnének megbeszélni egy viszonykulcsot, akkor Aliz kitalál egyet, és elküldi Bobnak K_{AB}-vel titkosítva.

A 8.32. ábra protokollja öt üzenetet tartalmaz. Nézzük meg, hogy egy kis fejtöréssel sikerül-e megspórolnunk ezek közül néhányat! A 8.33. ábra egy lehetséges megközelítést mutat. Itt Aliz nem vár Bobra, hanem rögtön kezdeményezi a kihívás-válasz protokollt. Hasonlóképp, Bob is rögtön elküldi a saját kihívását az Aliznak küldött válaszában. Ezzel az egész protokollt öt helyett három üzenetre lehet rövidíteni.

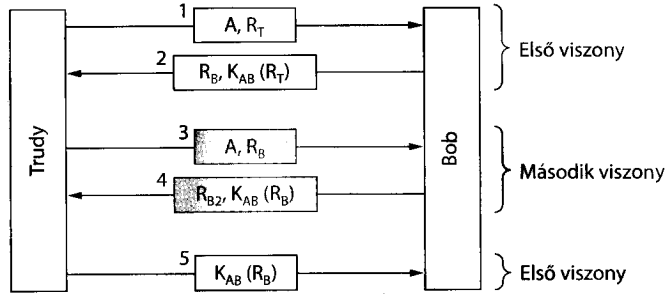
Vajon ez az új protokoll jobb-e, mint az eredeti? Egy szempontból igen: rövidebb. Sajnos azonban rosszabb is. Bizonyos körülmények között Trudy legyőzheti ezt a protokollt, a **visszatükrözéses támadás (reflection attack)** segítségével. Gyakorlatilag Trudy betörhet, ha lehetséges egyszerre több viszonyt létrehozni Bobbal. Ez lehet a helyzet például, ha Bob egy bank, amely kész több szimultán, a pénzkiadó automatáktól jövő hívás fogadására.

Trudy visszatükrözéses támadása a 8.34. ábrán látható. Először is Trudy Aliznak adja ki magát, és elküldi R_T-t. Bob szokás szerint válaszol saját kihívásával, R_B-vel. Most Trudy tehetetlen. Mit csinálhatna? Nem tudja $K_{AB}(R_B)$-t.

Kezdeményezhet egy másik kapcsolatot a 3-as üzenettel, amiben elküldheti azt az R_B-t mint kihívást, amit a 2-es üzenetben kapott. Bob gyanútlanul titkosít, és visszaküldi $K_{AB}(R_B)$-t a 4-es üzenetben. Ezzel Trudy megkapta a hiányzó információt, és be tudja fejezni az első kapcsolat felépítését, a másodikat pedig megszakítja. Bob meg van



8.33. ábra. *Rövidített kétirányú hitelességvizsgálat protokoll*



8.34. ábra. *Visszatükrözéses támadás*

győződve róla, hogy Trudy Aliz, és mikor a bankszámla egyenlegét lekérdezi, szó nélkül megadja a választ. Nem kételkedik akkor sem, mikor arra kéri, hogy mindent utaljon át egy titkos svájci bankszámlára.

A történet tanulsága:

Egy korrekt hitelességvizsgáló protokoll készítése nehezebb, mint amilyennek látszik.

A következő négy általános szabály gyakran segít a tervezőknek a gyakori hibák kiküszöbölésében:

1. A kezdeményezőnek előbb kell igazolnia magát, mint a fogadónak. Ez magakadályozza Bobot abban, hogy értékes infirmációt adjon ki, mielőtt Trudy egyáltalán igazolná kilétét.

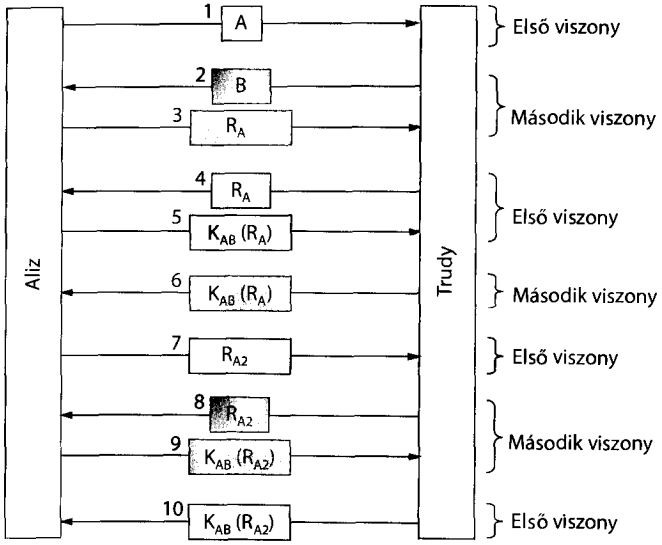
2. A kezdeményezőnek és a fogadónak különböző kulcsokat kell használni a hitelesítéshez még akkor is, ha ehhez két megosztott kulcs K_{AB} és K'_{AB} kell.

3. A kezdeményezőnek és a fogadónak különböző halmazokból kell választania a kihívást. Például a kezdeményezőnek páros, a fogadónak pedig páratlan számokat kelljen használnia.

4. A protokollnak ellenállónak kell lennie az olyan támadásoknak, melyek egy második, párhuzamos viszonyt használnak fel, és az egyik viszonyban megszerzett információt a másikban hasznosítják.

Elég, ha csak egy szabályt nem tartunk be, és a protokollunk máris sok esetben feltörhető lesz. Példánkban mind a négy szabályt megsértettük, ez pedig katasztrofális következményekkel járt.

Most pedig menjünk egy kicsit vissza és vegyük jobban szemügyre a 8.32. ábrát! Biztos, hogy a protokollt nem veszélyezteti a visszatükrözéses támadás? Nos, az attól függ. Erre a kérdésre elég körülményes választ adni. Trudynak azért sikerült visszatükrözéses támadással legyőzni a protokollunkat, mert meg tudott nyitni egy második viszonyt Bobbal, és rá bírta venni arra, hogy a saját kérdéseit válaszolja meg. Vajon mi történik



8.35. ábra. *Visszatükrözéses támadás a 8.32. ábra protokollja ellen*

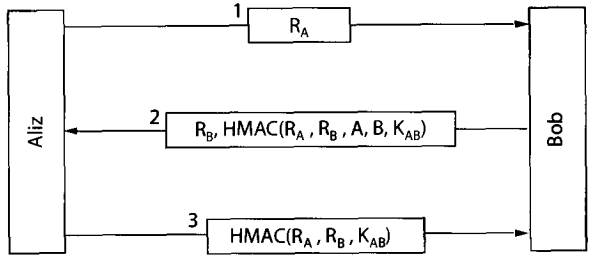
akkor, ha Aliz sem egy gép előtt ülő ember, hanem egy általános célú számítógép, mely több viszonyt is tud létesíteni? Nézzük meg, mit tehet ilyenkor Trudy.

Trudy támadását a 8.35. ábra segítségével érthetjük meg. Aliz először közli személyazonosságát az 1-es üzenetben. Trudy ezt elfogja, új viszonyt kezd a 2-es üzenettel, azt állítván, hogy ő Bob. A 2. viszonyhoz tartozó üzeneteket sötét színnel jelöltük. Aliz a 3-as üzenetben így válaszol: „Azt mondod, te vagy Bob? Bizonyítsd be!" Itt Trudy megakad, hiszen nem tudja bizonyítani, hogy ő lenne Bob.

Mit tehet ilyenkor Trudy? Visszatér az első viszonyhoz, ahol neki kell kihívást küldenie, és elküldi a 3-as üzenetben kapott R_A-t. Aliz készségesen válaszol erre az 5-ös üzenetben, ezzel megadja azt az információt, amire Trudynak a 2. viszonyban a 6-os üzenet elküldéséhez szüksége van. Ezen a ponton Trudy gyakorlatilag révbe ért, hiszen sikeresen megválaszolta Aliz kihívását a 2. viszonyban. Ekkor megszakíthatja az 1. viszonyt, a 2. viszony hátralévő részében pedig bármilyen régi számot elküldhet, és máris lesz egy hitelesített viszonya Alizzal.

Csakhogy Trudy még ennél is gonoszabb, és még nagyobb bajt akar keverni. Ahelyett, hogy elküldene egy régi számot, és ezzel befejezné a 2. viszonyt, megvárja, míg Aliz elküldi a 7-es üzenetet, vagyis az 1. viszonyhoz tartozó kihívását. Trudy természetesen megint nem tudja, hogyan válaszoljon, ezért ismét a visszatükrözéses támadást alkalmazza, és 8-as üzenetként az R_{A2}-t küldi vissza. Aliz minden további nélkül kódolja R_{A2}-t a 9-es üzenetben. Trudy ekkor visszavált az 1. viszonyhoz, és a 10-es üzenetben elküldi Aliznak az általa kért számot, amit egyszerűen csak ki kell másolnia Aliz 9-es üzenetéből. Ezen a ponton Trudynak már két, teljesen hitelesített viszonya van Alizzal.

Ennek a támadásnak az eredménye némiképp eltér a 8.34. ábrán látott háromüzenetes protokoll támadásának eredményétől. Trudynak ezúttal két hitelesített összeköttetése lesz Alizzal, míg az előző példában egy hitelesített összeköttetése volt Bobbal. Itt is



8.36. ábra. *Hitelesítés HMAC-kódolók felhasználásával*

igaz az, hogy ha betartottuk volna az előbb tárgyalt általános szabályokat a hitelesítési protokollokra vonatkozóan, akkor ez a támadás kivédhető lett volna. Az ilyenfajta támadásokról és az ellenük való védekezés módjairól Bird és mások [1993] adnak részletes leírást. Írásukban azt is megmutatják, hogyan lehet szisztematikus úton megalkotni olyan protokollokat, melyek bizonyíthatóan helyesen működnek. Mindazonáltal még a legegyszerűbb ilyen protokoll is elég komplikált, ezért mi most egy másik protokollosztályt mutatunk be, mely szintén hatásos.

A Bird és mások [1993] által leírt új hitelességvizsgáló protokollt a 8.36. ábra mutatja be. A protokoll az IPsec tárgyalásánál látott HMAC-hez hasonló kódot használ. Aliz első üzenetében elküldi Bobnak az R_A egyszer használatos véletlen számot. Bob erre saját véletlen számával, R_B-vel válaszol, amit egy HMAC-kóddal együtt küld vissza. Ehhez előbb létrehoz egy adatszerkezetet, amely Aliz véletlen számából, a saját véletlen számából, kettejük személyazonosságaiból, és a közös titkos kulcsból, K_{AB}-ből áll. Ezt az adatszerkezetet aztán a HMAC-be hash-eli, például az SHA-1 használatával. Amikor Aliz megkapja a 2-es üzenetet, ismerni fogja R_A-t (hiszen azt ő választotta), a nyílt szövegként érkezett R_B-t, a két személyazonosságot, és a K_{AB} titkos kulcsot is, hiszen azt mindig is ismerte. Mindebből ő is kiszámíthatja a HMAC-t. Ha az megegyezik az üzenetben található HMAC-vel, akkor biztos lehet benne, hogy Bobbal beszél, hiszen Trudy nem ismeri K_{AB}-t, így azt sem tudhatja, milyen HMAC-t kellene küldenie. Aliz ezután egy olyan HMAC-vel válaszol Bobnak, ami csak a két véletlen számot tartalmazza.

Feltörheti-e vajon Trudy ezt a protokollt? Nem, mivel egyik felet sem tudja rávenni arra, hogy egy általa választott értéket kódoljon vagy hash-eljen, mint ahogy azt a 8.34. és a 8.35. ábránál tette. Itt mindkét HMAC-kód a küldő fél által választott értéket tartalmazza, azt a választást pedig Trudy nem tudja befolyásolni.

Ezt az ötletet nemcsak HMAC-kódok használatával lehet megvalósítani. Gyakran használják azt a megoldást is, hogy az elemek sorozatára nem HMAC-kódokat számolnak ki, hanem az egyes elemeket sorban egymás után kódolják a titkosított blokkok láncolásának módszerével.

8.7.2. Osztott kulcs létesítése: a Diffie–Hellman-kulcscsere

Eddig feltételeztük, hogy Aliznak és Bobnak van egy osztott titkos kulcsa. Mi van, ha még sincs? Hogyan beszélhetnek meg egyet? Egy lehetséges módszer az lenne, ha Aliz

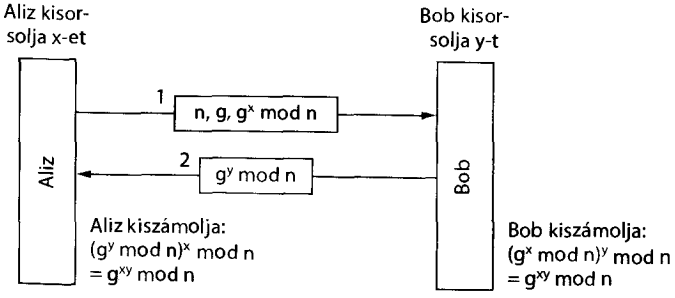
felhívná Bobot telefonon, és megmondaná neki a sajátját, de ő valószínűleg azt mondaná: „Honnan tudjam, hogy tényleg Aliz vagy és nem Trudy?" Vagy megbeszélhetnének egy találkozót, ahova mindketten elviszik az útlevelüket, a jogosítványukat, és három közismert hitelkártyát, de mivel elfoglalt emberek, valószínűleg hónapokba telne, mire mindkettejüknek megfelelő időpontot találnának. Hihetetlenül hangzik, de szerencsére létezik egy megoldás, mely segítségével vadidegenek is megbeszélhetnek egy osztott titkos kulcsot fényes nappal, annak ellenére, hogy Trudy minden üzenetet gondosan rögzít.

A protokollnak, amely lehetővé teszi, hogy idegenek is megbeszélhessenek egy osztott titkos kulcsot, **Diffie–Hellman-kulcscsere (Diffie–Hellman key exchange)** a neve [Diffie és Hellman, 1976], és a következőképpen működik. Aliznak és Bobnak meg kell egyeznie két nagy számban, n-ben és g-ben, ahol n és $(n-1)/2$ is prímszám, és néhány követelmény teljesül g-re is. Ezek a számok nyilvánosak lehetnek, azaz bármelyikük választhat egy n-et és g-t, majd nyíltan elküldheti a másiknak. Ezután Aliz kisorsol egy nagy (mondjuk 1024 bites) számot, x-et, és ezt titokban tartja. Hasonlóképpen Bob is kisorsol egy nagy titkos számot, y-t.

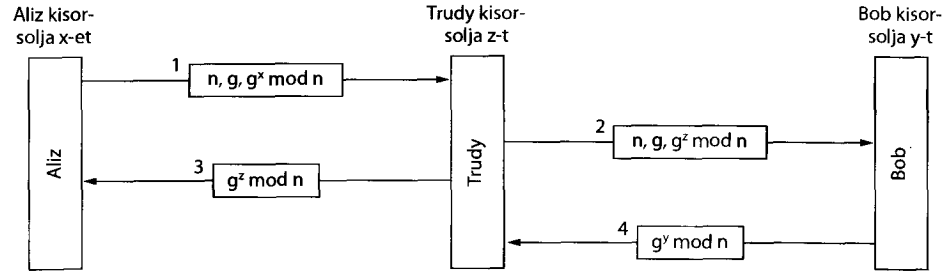
Aliz kezdeményezi a kulcscsere-protokollt azzal, hogy elküldi Bobnak egy üzenetben $(n,g,g^x \bmod n)$-et (lásd 8.37. ábra). Bob válaszol Aliznak, és elküldi $g^y \bmod n$-et. Aliz a kapott számot az x-edik hatványra emeli, így megkapja $(g^y \bmod n)^x \bmod n$-et. Bob hasonló módon előállítja $(g^x \bmod n)^y \bmod n$-et. A modulusos aritmetika szabályai szerint mindkét kifejezés megegyezik $(g^{xy} \bmod n)$-nel. Íme Aliz és Bob ezzel megbeszélték megosztott kulcsukat, $(g^{xy} \bmod n)$-et.

Természetesen Trudy mindkét üzenetet látta. Ismeri tehát g-t és n-et az 1. üzenetből. Ha ki tudná számolni x-et és y-t, akkor meghatározhatná a titkos kulcsot. A probléma az, hogy $g^x \bmod n$-ből nem tudja meghatározni x-et. Nem ismert olyan használható algoritmus, amely nagy prímszám modulo diszkrét logaritmusát állítja elő.

A fenti példa konkretizálása érdekében az $n=47$ és $g=3$ (gyakorlatban használhatatlan) számokat használjuk. Aliz $x=8$-at sorsol, míg Bob $y=10$-et. Ezeket mindketten titokban tartják. Aliz üzenete Bobnak (47, 3, 28), mert $3^8 \bmod 47 = 28$. Bob üzenete Aliznak (17). Aliz kiszámolja $17^8 \bmod 47 = 4$-et. Bob kiszámolja $28^{10} \bmod 47 = 4$-et. Aliz és Bob függetlenül megállapította, hogy a titkos kulcs most 4. Trudynak meg kell oldani a $3^x \bmod 47 = 28$ egyenletet, ami kimerítő kereséssel valósítható meg kis számokra, de nem megy, ha a számok mind százbites nagyságrendűek. Minden eddig ismert algoritmus egyszerűen túl sok időt venne igénybe, még egy nagy teljesítményű párhuzamos szuperszámítógéppel is.



8.37. ábra. *Diffie–Hellman-féle kulcscsere*



8.38. ábra. *Az ún. élőlánc- vagy közbeékelődéses támadás*

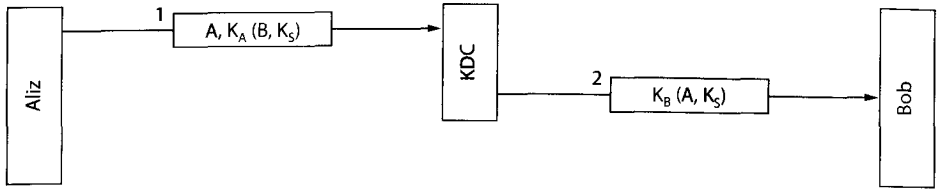
A Diffie–Hellman-kulcscsere algoritmus eleganciájának ellenére felmerül egy probléma: honnan tudja Bob, amikor megkapja a (47, 3, 28) számhármast, hogy ez tényleg Aliztól származik-e, és nem Trudytól. Nincs rá mód, hogy megbizonyosodjon efelől. Sajnos Trudy kihasználhatja ezt a tényt, hogy megtévessze mind Alizt, mind Bobot, amint azt a 8.38. ábra mutatja. Miközben Aliz és Bob külön-külön kisorsolja x-et és y-t, addig Trudy is kiválasztja saját véletlen számát, z-t. Aliz elküldi az 1-es üzenetet, amit Bobnak szán. Trudy ezt elfogja, és egy 2-es üzenetet küld Bobnak, a helyes g és n számokkal (amelyek amúgy is nyilvánosak), de x helyett saját véletlen számával, z-vel. Ezenkívül a 3-as üzenetben visszaüzen Aliznak. Később Bob elküldi a 4-es üzenetet Aliznak, amit Trudy ismét elfog.

Ekkor mindenki alkalmazza a modulos aritmetikát. Aliz titkos kulcsként $g^{xy} \bmod n$-et számol, és Trudy is ezt kapja (Alizzal történő üzenetváltásokhoz). Bob eredménye $g^{yz} \bmod n$, és Trudy is ezt kapja (Bobbal való üzenetváltásokhoz). Aliz azt hiszi, hogy Bobbal beszél, és létrehoz egy viszonykulcsot (Trudyval). Bob is így tesz. Trudy elfog és tárol, vagy tetszés szerint módosít minden üzenet, amit Aliz a titkosított viszonyon küld, majd (ha akarja) továbbküldi Bobnak. A másik irányban is hasonló a helyzet. Trudy minden üzenetet lát, és tetszés szerint módosíthat, miközben Aliz és Bob mindketten abban az illúzióban vannak, hogy egy biztonságos csatorna van köztük. Ezért ez a fajta támadás **közbeékelődéses támadás (man-in-the-middle attack)** néven ismert. Másik neve **élőlánc-támadás (bucket brigade attack)**, mert némileg emlékeztet a hajdani önkéntes tűzoltókra, akik élő láncot alkottak a tűzoltókocsitól a tűzig, és úgy adogatták egymásnak a vödröket.

8.7.3. Hitelességvizsgálat kulcselosztó központ alkalmazásával

Egy idegennel megosztott titok megbeszélése kis híján sikerült már, de tulajdonképpen mégsem. Ha jól belegondolunk, nem is biztos, hogy érdemes. Ezzel a módszerrel, ha n partnerrel tartjuk a kapcsolatot, n kulcsot kell tárolnunk. Egy népszerű embernek a kulcsok karbantartása komoly terhet jelenthet, főleg, ha minden kulcsot különálló műanyag chipkártyán kell tárolnia.

Egy másik megközelítés, ha bevezetünk egy megbízható kulcselosztó központot (Key Distribution Center, KDC). Ebben a modellben minden felhasználónak csak egy a KDC-vel megosztott kulcsa van. A hitelességvizsgálat és a kulcskarbantartás így a



8.39. ábra. *Első kísérlet KDC-t használó hitelességvizsgáló protokollra*

KDC-n keresztül történik. A legegyszerűbb, két felhasználót és egy megbízható KDC-t tartalmazó hitelességvizsgáló protokoll látható a 8.39. ábrán.

A protokoll alapötlete egyszerű: Aliz kisorsol egy viszonykulcsot, K_S-t, és megüzeni a KDC-nek, hogy K_S-t használva Bobbal szeretne beszélni. Ez az üzenet titkosítva van azzal a K_A titkos kulccsal, amit Aliz (csak) a kulcselosztó központtal oszt meg. A KDC visszakódolja az üzenetet, és kiveszi belőle Bob azonosítóját és a viszonykulcsot. Ezután létrehoz egy új üzenetet, ami tartalmazza Aliz azonosítóját és a viszonykulcsot, majd elküldi Bobnak. A titkosítás a Bob és a KDC között megosztott K_B kulccsal történik. Bob az üzenetet visszakódolva megtudja a viszonykulcsot, és azt, hogy Aliz szeretne beszélni vele.

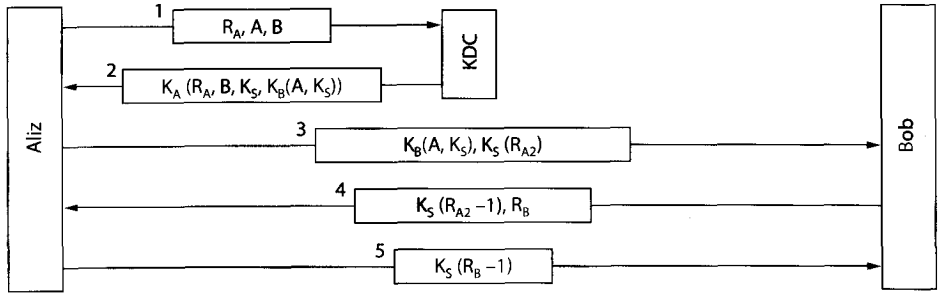
A hitelességvizsgálat itt ingyen történik. A KDC tudja, hogy az 1-es üzenet biztosan Aliztól származik, hiszen azt más nem tudta volna titkosítani Aliz titkos kulcsával. Hasonlóan, Bob tudja, hogy a 2-es üzenet biztosan a KDC-től jött, amiben megbízik, hiszen más nem tudja az ő titkos kulcsát.

Sajnos ennek a protokollnak van egy súlyos hibája. Trudy pénzszűkében van, így keres valami legális Aliz által igényelt szolgáltatást, kecsegtető ajánlatot tesz, és megkapja a munkát. A munka végeztével Trudy udvariasan megkéri Alizt, hogy banki átutalással fizessen. Aliz tehát megbeszél egy viszonykulcsot bankárával, Bobbal. Ezután egy üzenetben arra kéri Bobot, hogy utaljon át pénzt Trudy számlájára.

Eközben Trudy visszatér régi énjéhez, és a hálózatot kémleli. Felveszi mind a 8.39. ábrán látható 2-es üzenetet, mind az azt követő pénzátutalási kérelmet. Később visszajátssza mindkettőt Bobnak. Bob megkapja őket, és azt hiszi: „Aliz biztosan megint felbérelte Trudyt. Ezek szerint érti a dolgát." Bob ismét átutalja a pénzösszeget Aliz számlájáról Trudyéra. Valamikor az 50-edik üzenetpáros vétele környékén Bob kirohan irodájából, hogy megkeresse Trudyt, és felajánljon neki egy nagyobb hitelt, hogy fejleszthesse a nyilvánvalóan sikeres vállalkozását. Ezt a problémát nevezik **visszajátszásos támadásnak (replay attack)**.

Jó pár megoldás van a visszajátszásos támadás kivédésére. Egyik lehetőség, hogy minden üzenetet időbélyeggel látunk el. Ha valaki egy idejemúlt üzenetet kap, eldobhatja azt. Ezzel a megközelítéssel az a gond, hogy az órák sohasem teljesen szinkronizáltak egy hálózatban, ezért az időbélyeg egy bizonyos időtartamig érvényes kell legyen. Trudy ebben az időtartamban még sikeresen visszajátszhatja az üzenetet.

A második megoldás az egyszer használatos, egyedi véletlen üzenetszám beillesztése minden üzenetbe, amit általában **egyszer használatos véletlen számnak (nonce)** neveznek. Így minden résztvevőnek tárolnia kell az elhasznált üzenetszámokat, és kidobni a régi üzenetszámmal érkező leveleket. Az üzenetszámokat azonban örökre meg kell



8.40. ábra. *Needham–Schroeder-hitelességvizsgáló protokoll*

őrizni, nehogy Trudy egy 5 éves üzenetet próbáljon visszajátszani. Továbbá, ha egy gép összeomlik, és elveszti az üzenetszámok listáját, ismét sebezhetővé válik a visszajátszásos támadással szemben. Az időbélyegek és az üzenetszámok kombinálhatók úgy, hogy csak korlátos ideig kelljen tárolni az üzenetszámokat, természetesen azonban így sokkal komplikáltabb lesz a protokoll.

A hitelességvizsgálat kifinomultabb megközelítése a többutas kihívás-válasz protokoll, amelynek közismert változata a **Needham–Schroeder-féle hitelességvizsgáló protokoll** [Needham–Schroeder, 1978], ennek egy változata látható a 8.40. ábrán.

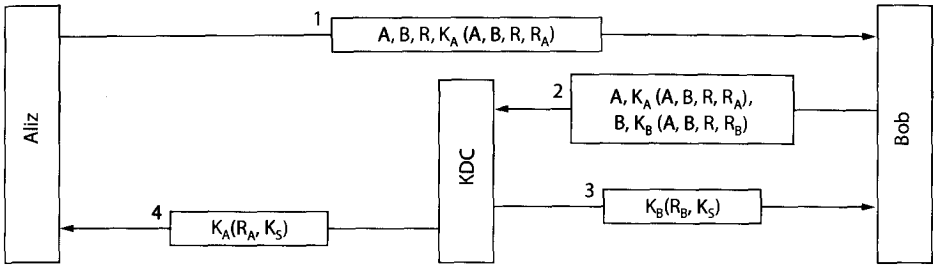
A protokoll első lépésében Aliz megüzeni a kulcselosztó központnak, hogy Bobbal szeretne beszélni. Ez az üzenet tartalmaz egy nagy véletlen számot, R_A-t, mint egyszer elküldött, egyedi üzenetszámot. A KDC visszaküldi a 2-es üzenetben Aliz véletlen számát, R_A-t, egy viszonykulcsot és egy jegyet, amit elküldhet Bobnak. R_A azért szükséges, hogy Aliz biztosítva legyen afelől, hogy az üzenet friss, nem pedig egy visszajátszás. Bob azonosítója szintén mellékelve van arra az esetre, ha Trudynak az az ötlete támadna, hogy kicseréli az 1-es üzenetben B-t a sajátjára azért, hogy a KDC a 2-es üzenetben a jegyet K_T-vel titkosítsa, és nem K_B-vel. A K_B-vel titkosított jegyet a titkosított üzenet tartalmazza, nehogy Trudy az Alizhoz vezető visszaúton kicserélhesse valamivel.

Aliz ezután elküldi a jegyet Bobnak, egy új véletlen szám, R_{A2} kíséretében, amit K_S-sel titkosít. A 4-es üzenetben Bob visszaküldi $K_S(R_{A2}-1)$-et, hogy biztosítsa Alizt arról, hogy az igazi Bobbal beszél. $K_S(R_{A2})$ visszaküldése nem lenne jó, hiszen azt Trudy egyszerűen kilophatja a 3-as üzenetből.

A 4-es üzenet kézhezvétele után Aliz biztos benne, hogy Bobbal beszél, valamint, hogy eddig nem történhetett visszajátszás. Hiszen pár ezredmásodperccel ezelőtt generálta R_{A2}-t. Az 5-ös üzenet célja, hogy Bob is meggyőződjön róla, hogy tényleg Alizzal beszél, és itt sem történt visszajátszás. Azzal, hogy mindkét fél generált egy kihívást, és válaszolt is egyre, a visszajátszásos támadás lehetősége kizárható.

Annak ellenére, hogy ez a protokoll elég megbízhatónak látszik, mégis van egy gyenge pontja. Ha Trudy egyszer megszerez egy régi viszonykulcsot kódolatlan formában, akkor a 3-as üzenet visszajátszásával egy viszonyt kezdeményezhet Bobbal, és elhitetheti vele, hogy ő Aliz [Denning és Sacco, 1981]. Ezúttal kifoszthatja Aliz bankszámláját anélkül, hogy törvényesen valaha is dolgozott volna neki.

Needham és Schroeder később publikáltak egy protokollt, amely megoldja ezt a problémát [Needham és Schroeder, 1987]. Ugyanannak a folyóiratnak ugyanabban a szá-



8.41. ábra. *(Egyszerűsített) Otway–Rees-féle hitelességvizsgáló protokoll*

mában Otway és Rees [1987] szintén ismertetett egy rövidebb protokollt, ami szintén kiküszöböli ezt a hibát. A 8.41. ábrán az **Otway–Rees-protokoll** egy kismértékben módosított változata látható.

Az Otway–Rees-protokollban Aliz először előállít egy véletlen számpárost, R-et, ami egy általános azonosító, és R_A-t, amit Aliz majd kihívásként használ Bob felé. Mikor Bob megkapja ezt az üzenetet, létrehoz egy új üzenetet Aliz üzenetének titkosított részéből, és egy hasonlót saját magától. A K_A-val és K_B-vel titkosított mindkét rész azonosítja Alizt és Bobot, tartalmazza az általános azonosítót, és tartalmaz egy kihívást.

A KDC ellenőrzi, hogy R mindkét részben ugyanaz-e. Lehet, hogy nem, hiszen Trudy megváltoztathatta R-et az 1-es üzenetben, vagy kicserélhette a 2-es üzenet egy részét. Amennyiben az R-ek megegyeznek, a KDC elhiszi, hogy a Bob által üzent kérés érvényes. Ezután generál egy viszonykulcsot, és két példányban titkosítja azt, egyszer Aliznak, egyszer pedig Bobnak. Mindegyik üzenet tartalmazza a fogadó véletlen számát biztosítékul arra nézve, hogy nem Trudy hozta létre azokat. Ezek után mind Aliz, mind Bob birtokában van ugyanannak a viszonykulcsnak, és elkezdhetnek kommunikálni. Mikor először cserélnek adatot tartalmazó üzenetet, mindketten láthatják, hogy a másiknak is birtokában van K_S tökéletesen azonos másolata, ezzel a hitelességvizsgálat befejeződött.

8.7.4. Hitelességvizsgálat Kerberos alkalmazásával

Sok valódi rendszerben (köztük a Windows 2000-ben is) használják a **Kerberos** hitelességvizsgáló protokollt, mely a Needham–Schroeder egyik variációján alapul. Egy sokfejű kutyaszörnyetegről kapta nevét, amely a görög mitológiában Hadész bejáratát őrizte (feltételezhetően a nemkívánatosak távoltartása végett). A Kerberost az M.I.T.-n fejlesztették ki azért, hogy a munkaállomások felhasználói biztonságosan hozzáférhessenek a hálózati erőforrásokhoz. Legfontosabb különbsége a Needham–Schroeder-hez képest az a feltételezés, hogy az órák meglehetősen jól szinkronizáltak. A protokoll sok változtatáson ment keresztül. A V5 változat az, amelyet az iparban széles körben használnak és az RFC 4120 definiálja. Az előző változatot, a V4-et nyugdíjazták, miután komoly hiányosságokat találtak benne [Yu és mások, 2004]. A V5 tulajdonképpen a V4 feljavítása egy sor kisebb protokollváltoztatással és néhány továbbfejlesztett lehetőséggel, például hogy a továbbiakban nem kapcsol át a ma már elavult DES-re. További információért lásd Neuman és Ts'o [1994] munkáját.

A Kerberos három további szervert is igénybe vesz az Aliz (egy kliens-munkaállomás) mellett:

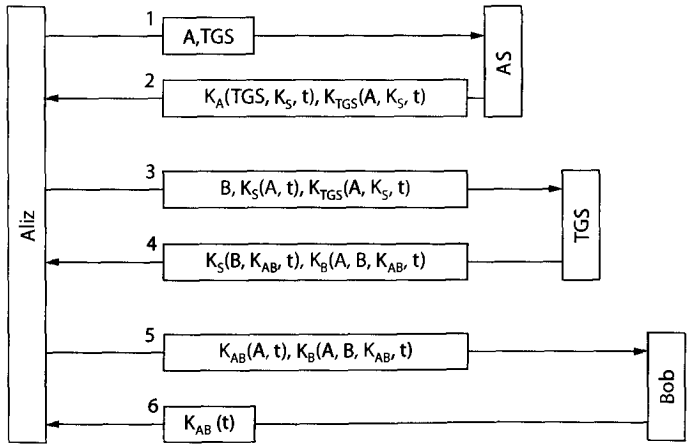
1. hitelességvizsgáló szervert (Authentication Server, AS): ellenőrzi a felhasználót belépéskor,
2. jegyadó szervert (Ticket-Granting Server, TGS): „személyazonosságot igazoló jegyeket" ad,
3. Bobot, a szervert: elvégzi az Aliz által kért munkát.

A hitelességvizsgáló szerver (AS) a KDC-hez hasonlóan, minden felhasználóhoz fenntart egy titkos jelszót. A jegyadó (TGS) feladata, hogy olyan jegyeket adjon, amelyek biztosítják a valódi szervereket, hogy a jegytulajdonos tényleg az, akinek mondja magát.

Egy viszony kezdeményezéséhez Aliz odaül egy tetszőleges nyilvános munkaállomáshoz és begépeli nevét. A munkaállomás nyílt szövegként kódolatlanul elküldi Aliz nevét és a TGS nevét az AS-hez (a 8.42. ábrán az 1-es üzenetet). Válaszul kap egy viszonykulcsot és egy jegyet, $K_{TGS}(A, K_S, t)$, a TGS számára. A viszonykulcsot Aliz titkos kulcsával titkosítja úgy, hogy csak Aliz tudja megfejteni. A munkaállomás csak akkor kérdezi meg Aliz jelszavát, amikor a 2-es üzenet megérkezik – előbb nem. A jelszót használja ezután a K_A előállításához annak érdekében, hogy a 2-es üzenetet megfejtse és megkapja a viszonykulcsot

Ezután a munkaállomás felülírja Aliz jelszavát, hogy az legfeljebb néhány ezredmásodpercig legyen a munkaállomásban. Ha Trudy megpróbál belépni Aliz helyett, az általa begépelt jelszó nem lesz jó, és ezt a munkaállomás észreveszi, hiszen a 2-es üzenet szerkezetileg helytelen lesz.

Aliz, miután belépett, megmondhatja a munkaállomásnak, hogy Bobbal, a fájlszerverrel szeretne kapcsolatot teremteni. A munkaállomás ezután a TGS-nek küldött 3-as üzenetben kér egy Bobbal való viszonyában használható jegyet. A kulcselem itt a



8.42. ábra. *A Kerberos V5 működése*

$K_{TGS}(A, K_S)$, ami a TGS titkos kulcsával van titkosítva, és arra szolgál, hogy bizonyítsa, hogy a küldő tényleg Aliz. A TGS azzal válaszol, hogy létrehoz Aliznak egy Bobbal használható viszonykulcsot, K_{AB}-t. Ennek kétféle változata megy vissza. Az első csak K_S-sel van titkosítva, hogy Aliz el tudja olvasni. A második egy másik jegy, amely Bob kulcsával, K_B-vel van titkosítva, így Bob el tudja olvasni.

Trudy lemásolhatja a 3-as üzenetet, és megpróbálhatja újra használni, de ezt meghiúsítja az üzenethez kapcsolódó titkosított t időbélyeg. Trudy nem tudja lecserélni az időbélyeget egy frissebbel, mert nem ismeri K_S-t, a viszonykulcsot, amelyet Aliz a TGS-sel való beszédhez használ. Még ha Trudy gyorsan visszajátssza is a 3-as üzenetet, akkor is csak egy újabb másolatát szerezhetné meg a 4-es üzenetnek, amit már elsőre sem volt képes visszakódolni, és másodszorra sem fog tudni.

Most már Aliz elküldheti K_{AB}-t Bobnak, hogy létrehozzon egy viszonyt. Ez az üzenetváltás is időbélyeget tartalmaz. A válasz biztosítja Alizt afelől, hogy tényleg Bobbal és nem Trudyval beszél.

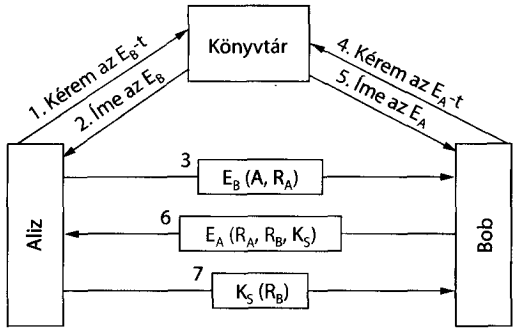
Az üzenetváltások sorozatának lezajlása után Aliz K_{AB} védelme alatt kommunikálhat Bobbal. Ha később úgy dönt, hogy egy másik szerverrel, Carollal is beszélnie kell, egyszerűen megismétli a 3-as üzenetet azzal a különbséggel, hogy ezúttal B helyett C-t küld. A TGS azonnal válaszol, megküldve a K_C-vel titkosított jegyet, amit Aliz elküldhet Carolnak, akit ez biztosít arról, hogy az Alitól jött.

Ennek a sok munkának az az értelme, hogy most már Aliz minden hálózaton elérhető szerverhez biztonságosan hozzákapcsolódhat, és a jelszavát sohasem kell a hálózaton keresztül küldenie. Sőt annak csak néhány ezredmásodpercre kell a saját munkaállomásában lennie. Vegyük észre azonban, hogy minden szerver saját maga végzi a jogosultságvizsgálatot. Mikor Aliz megmutatja a jegyét Bobnak, ez Bobot csupán arról győzi meg, hogy azt ki küldte. Pontosabban szólva, hogy Aliz mit tehet, azt Bob dönti el.

Minthogy a Kerberos fejlesztői nem várták, hogy az egész világ egyetlen hitelességvizsgáló szerverben bízzon meg, ezért gondoskodtak róla, hogy több **birodalom** (**realm**) létezhessen, külön-külön saját AS-sel és TGS-sel. Aliz úgy szerezhet jegyet egy távoli szerverhez, ha saját TGS-étől kér egy jegyet, amit a távoli TGS is elfogad. Ha a távoli TGS-t számon tartja a helyi TGS (hasonlóan a helyi szerverekhez), akkor a helyi TGS ad Aliznak egy jegyet, amely a távoli TGS-re érvényes. Ezután elintézheti ügyét a távoli birodalomban is, például szerezhet jegyet ottani szerverekhez. Vegyük észre azonban, hogy ahhoz, hogy két különböző birodalombeli fél együttműködhessen, mindkettőnek meg kell bíznia a másik TGS-ében. Különben nem tudnak együttműködni.

8.7.5. Hitelességvizsgálat nyilvános kulcsú titkosítással

A kölcsönös hitelességvizsgálat megoldható nyilvános kulcsú titkosítás használatával is. Induljunk ki abból, hogy Aliznak szüksége van Bob nyilvános kulcsára. Ha adott egy PKI, és annak van egy olyan könyvtárszolgáltatása, amely a nyilvános kulcsokról ad ki tanúsítványokat, akkor Aliz elkérheti Bob tanúsítványát. Ezt a kérést jelöltük a 8.43. ábrán az 1-es üzenettel. A 2-es üzenetben lévő válasz egy X.509 tanúsítvány, amely tartalmazza Bob nyilvános kulcsát. Miután ellenőrizte az aláírást, Aliz egy újabb üzenetben elküldi Bobnak az azonosítóját és egy egyszer használatos véletlen számot (nonce).



8.43. ábra. *Kölcsönös hitelességvizsgálat nyilvános kulcsú titkosítás használatával*

Amikor Bob megkapja ezt az üzenetet, még fogalma sincs arról, hogy az vajon Aliztól vagy Trudytól jött-e, de belemegy a játékba, és elkéri a könyvtárszolgáltatástól Aliz nyilvános kulcsát (4-es üzenet), amit hamarosan meg is kap (5-ös üzenet). Ezután elküldi Aliznak a 6-os üzenetet, mely tartalmazza az Aliz-féle R_A-t, saját véletlen számát, R_B-t, és a javasolt viszonykulcsot, K_S-et.

Mikor Aliz megkapja a 6-os üzenetet, visszakódolja azt saját egyéni kulcsának segítségével. Megtalálja benne az R_A-t, ami jóleső melegséggel tölti el: az üzenet csakis Bobtól jöhetett, mivel Trudy nem találhatja ki az R_A-t. Továbbá biztosan friss, hiszen épp az előbb küldte el az R_A-t Bobnak. Aliz tehát a 7-es üzenet elküldésével beleegyezik a viszonyba. Amikor Bob meglátja az imént generált viszonykulccsal titkosított R_B-t, tudni fogja, hogy Aliz megkapta a 6-os üzenetet és ellenőrizte az R_A-t. Bob most egy boldog ember.

Hogyan próbálhatja Trudy megfúrni ezt a protokollt? Összeeszkábálhat egy 3-as üzenetet, és cselesen ráveheti Bobot arra, hogy próbára tegye Alizt, de Aliz ekkor egy olyan R_A-t fog kapni, amit nem ő küldött, és nem folytatja tovább. Trudy nem tud hamis 7-es üzenetet visszaküldeni Bobnak, mert nem ismeri sem R_B-t, sem K_S-et, és nem is tudja megállapítani ezeket Aliz egyéni kulcsa nélkül. Nincs szerencséje.

8.8. Az elektronikus levelek biztonsága

Amikor egy e-levelet egy távoli helyre küldünk, akkor az általában egy tucat közbülső gépen megy keresztül. Mindegyik elolvashatja és tárolhatja a levelet későbbi felhasználás céljából. A gyakorlatban személyiségi jog nem létezik annak ellenére, hogy sokan azt hiszik, hogy igenis létezik. Akárhogy is van, sokan szeretnék azt, hogy képesek legyenek olyan e-levelet küldeni, amelyet csak a szándékolt címzett tud elolvasni, és senki más nem: sem a főnöük, de még a hatóság sem. Ez a vágy sok embert arra ösztönöz, hogy biztonságos e-levél előállítása érdekében használja a korábban megvizsgált titkosító módszereket. A következő szakaszokban a PGP nevű, széles körben használt, biztonságos e-levél rendszert fogjuk tanulmányozni, valamint röviden megemlítünk még egy másik rendszert is, az S/MIME-t. A biztonságos e-levelezésről további információval Kaufman és mások [2002], valamint Schneier [1995] szolgálnak.

8.8.1. PGP – elég jól biztosított személyiségi jog

Első példánk, a **PGP (Pretty Good Privacy – elég jól biztosított személyiségi jog)** tulajdonképpen egyetlen személy, név szerint Phil Zimmermann agyának szüleménye [Zimmermann, 1995a, 1995b]. Zimmermann a személyiségi jogok szószólója. Mottója: „Ha a személyiségi jogok védelmét törvénytelenné teszik, akkor csak a törvényen kívülieknek lesznek személyiségi jogaik". Az 1991-ben kiadott PGP egy komplett e-levél biztonsági csomag, mely biztosítja a személyiségi jogok védelmét, a hitelességvizsgálatot, a digitális aláírásokat és a tömörítést is egyben, méghozzá mindezt egyszerűen használható módon. Ezenfelül az egész csomag – a forráskódokat is beleértve – szabadon terjeszthető az interneten. A rendszert ma már széles körben használják, köszönhetően a minőségének, az árának (nulla), és annak, hogy UNIX, Linux, Windows és Mac OS platformokon is könnyen elérhető.

A PGP a titkosításhoz az **IDEA (International Data Encryption Algorithm – nemzetközi adatkódoló algoritmus)** nevű blokk-kódolót használja, ami 128 bites kulcsokkal dolgozik. Az algoritmust Svájcban fejlesztették ki abban az időben, amikor a DES-re már rossz szemmel néztek, de az AES-et még nem dolgozták ki. Az IDEA az elgondolását tekintve hasonlít a DES-hez és az AES-hez: több, egymást követő körben keveri össze a biteket; a keverőfüggvények részletei viszont különböznek a DES-nél és az AES-nél használtaktól. A kulcskezelés RSA-t, a sértetlenségvédelem pedig MD5-öt használ – ezeket a témákat már korábban tárgyaltuk.

A PGP már a meglétének első napja óta a viták középpontjában áll [Levy, 1993]. Zimmermann-nak eszébe sem jutott megakadályozni az embereket abban, hogy felrakják a PGP-t az internetre, ahonnan világszerte bárki elérheti azt, ezért az amerikai kormány kijelentette, hogy Zimmermann megsértette a hadianyagok kiviteléről szóló törvényt. A kormány 5 éven át folytatta Zimmermann ellen a vizsgálatot, amivel végül is felhagyott, feltehetően két okból. Először is, Zimmermann nem saját maga rakta fel a PGP-t az internetre, ezért az ügyvédje azt állította, hogy *ő* nem vitt ki semmit (és ezek után nincs nagy jelentősége annak a kérdésnek, hogy vajon egy weboldal megalkotása kivitelnek minősül-e egyáltalán). Másodszor, a kormány végül eljutott addig a felismerésig, hogy a per megnyeréséhez arról kellene meggyőznie az esküdtszéket, hogy egy letölthető, személyiségijog-védő programot tartalmazó weboldal azon fegyverkereskedelmi törvény hatálya alá esik, mely a tankok, tengeralattjárók, katonai repülőgépek, nukleáris fegyverek és ehhez hasonló hadianyagok kivitelét tiltja. A kormány dolgát a sajtó évekig tartó negatív propagandája sem könnyítette meg.

Kis kitérőként megjegyezzük, hogy a kiviteli szabályok valóban finoman szólva is bizarrak. A kormány illegális kivitelnek tartotta azt, hogy Zimmermann felrakott egy kódot a weboldalára, és ezért öt éven át zaklatta őt. Másfelől viszont, ha valaki a C nyelven íródott teljes PGP-forráskódot kiadná könyv formájában (jó nagy betűkkel szedve, és minden oldalon egy ellenőrző összeggel, hogy könnyen be lehessen olvasni lapolvasóval), majd exportálná a könyvet, akkor az ellen semmi kifogása se lenne a kormánynak, mert a könyvek nem számítanak hadianyagnak. A kard erősebb, mint a toll, legalábbis Uncle Sam számára.

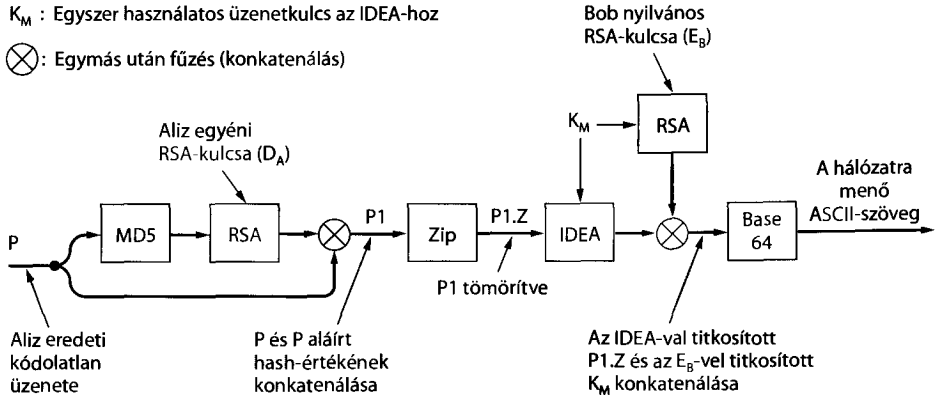
A PGP mindemellett a szabadalmak megsértésének problémájába is belekeveredett. Az RSA Security Inc., az RSA szabadalmának birtokosa azt állította, hogy a PGP a szabadalmat sértő módon használja az RSA-t, de ez a vita a 2.6-os változat megjelenésével elült. A PGP ráadásul az IDEA révén egy másik szabadalmaztatott titkosító algoritmust is használ, először ez okozott gondokat.

Mivel a PGP forráskódja nyílt, különböző magánszemélyek és csoportosulások is módosították már, így számos új verzió jött létre. Ezek közül egyesek a hadianyagtörvény megkerülését szolgálták, mások a szabadalmaztatott algoritmusok használatának elkerülését célozták meg, és megint mások zárt forráskódú, kereskedelmi terméket akartak csinálni a PGP-ből. A hadianyagtörvényt ugyan mára már egy kicsit liberalizálták (különben az AES-et használó termékeket nem lehetett volna kivinni az USA-ból), és az RSA szabadalma is lejárt 2000 szeptemberében, ezen problémák mégis azt hagyták örökségül, hogy ma különböző nevek alatt számos inkompatibilis PGP-verzió van forgalomban. Az alábbi tárgyalás a klasszikus PGP-re szorítkozik, mely a legrégebbi és legegyszerűbb változat. Az RFC 2440 egy másik népszerű változatot, az Open PGP-t írja le. Egy további változat a GNU Privacy Guard.

A PGP szándékosan használ meglevő kriptográfiai algoritmusokat ahelyett, hogy újakat találna ki. Főleg olyan algoritmusokra alapoz, amelyek a legalaposabb próbákat is kiállták már, és tervezésüket nem irányította vagy befolyásolta egyetlen kormányhivatal sem, hogy megpróbálja gyengíteni őket. Ez nagy előnyt jelent azok szemében, akik hajlamosak bizalmatlanul tekinteni a kormányra.

A PGP támogatja a szövegtömörítést, a titkosítást és a digitális aláírásokat, valamint átfogó kulcskezelési szolgáltatásokat is nyújt, de furcsa módon e-levél szolgáltatásokat nem. Inkább egyfajta előfeldolgozónak tekinthető, mely a nyílt szöveget veszi bemenetnek, és base64-alapú kódolt szöveget állít elő kimenetként. Ezt a kimenetet aztán természetesen már el lehet küldeni e-levélben. Egyes PGP-megvalósítások utolsó lépésként meg is hívják a felhasználói ügynököt, hogy ténylegesen elküldjék az üzenetet.

A PGP működését a 8.44. ábra segítségével tanulmányozhatjuk. Itt Aliz biztonságosan szeretne elküldeni Bobnak egy aláírt, szöveges üzenetet, *P*-t. Aliznak és Bobnak is van egyéni (D_X) és nyilvános RSA-kulcsa (E_X). Tegyük fel, hogy mindketten ismerik egymás nyilvános kulcsát! A kulcsok kezelésének módjára később térünk ki.



8.44. ábra. *Üzenetküldés PGP-vel*

Aliz azzal kezdi, hogy meghívja a számítógépén levő PGP programot. A PGP először legyártja a P hash-értékét az MD5 segítségével, majd kódolja az eredményt Aliz titkos RSA-kulcsával, (D_A)-val. Mikor Bob megkapja az üzenetet, visszakódolhatja a hash-értéket Aliz nyilvános kulcsával, és ellenőrizheti, hogy az megfelelő-e. Ha netalán valaki más (például Trudy) ebben a stádiumban meg is tudná szerezni a hash-értéket, és visszakódolná Aliz nyilvános kulcsával, az MD5 ereje akkor is biztosítaná, hogy kivitelezhetetlen legyen egy másik ugyanilyen hash-értékű levél generálása.

Ezután egyetlen üzenetbe, *P1*-be, egymás után belekerül a kódolt hash-érték és az eredeti üzenet, majd a ZIP-program, amely a Ziv–Lempel-eljárást használja [Ziv és Lempel, 1977], tömöríti ezt. Nevezzük e lépés kimenetét *P1.Z*-nek.

A PGP ezután egy véletlen karakterlánc begépelését kéri Aliztól. Mind a tartalom, mind a beírás sebességének figyelembevételével elkészül egy 128 bites IDEA-üzenetkulcs, K_M (amit a PGP-irodalom viszonykulcsnak nevez, de ez valójában egy névhiba, hiszen itt nincs szó viszonyról). Ezután megtörténik a *P1.Z* visszacsatolásos kódolása, K_M-mel és az IDEA-val. Továbbá K_M is titkosításra kerül Bob nyilvános kulcsával, E_B-vel. Ezután ennek a két komponensnek az egymás után fűzése (konkatenálása) és base64 kódolása következik, ahogyan azt a MIME-ről szóló részben láttuk. Az eredmény ezután csak betűket, számokat és a +, / és = szimbólumokat tartalmazza, tehát beletehető egy RFC 822-es formájú üzenet szövegrészébe, és várhatóan módosítás nélkül megérkezik.

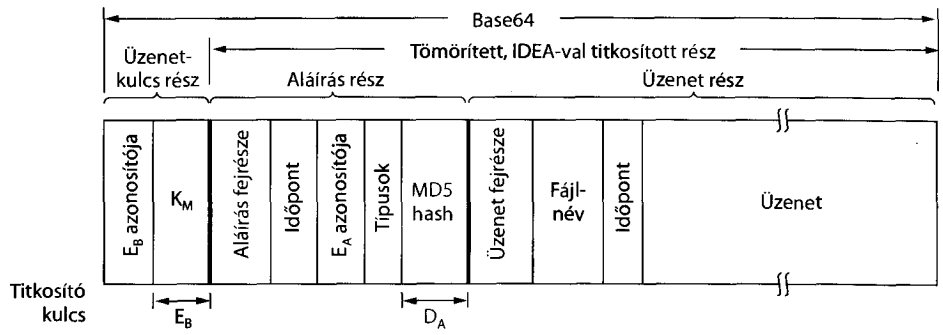
Mikor Bob megkapja az üzenetet, visszakódolja a base64 kódot, és visszakódolja az IDEA-kulcsot is titkos RSA-kulcsával. A kulcs segítségével visszakódolja az üzenetet *P1.Z* alakba. A tömörítés kicsomagolása után Bob elkülöníti a szöveget a kódolt hash-értéktől, majd visszakódolja a hash-értéket Aliz nyilvános kulcsával. Ha a hash-érték megegyezik azzal, amit saját kezűleg számolt ki az MD5-tel, akkor tudja, hogy ez a helyes P üzenet, és tudja, hogy Aliztól jött.

Érdemes megjegyezni, hogy az RSA itt csak két helyen kell: a 128 bites MD5 hash-érték kódolásánál és a 128 bites IDEA-kulcs kódolásánál. Az RSA ugyan lassú, de itt nem nagy mennyiségű adatot, hanem csak 256 bitet kell kódolnia. Továbbá, mind a 256 bit szerfelett véletlen, így Trudy részéről már az is kemény munkát igényel, hogy megállapítsa egy tippelt kulcsról, hogy helyes-e. A kódolás nagy része IDEA-val megy, ami nagyságrendekkel gyorsabb az RSA-nál. A PGP tehát biztonságot, tömörítést és digitális aláírást biztosít, és mindezt sokkal hatékonyabban, mint a 8.19. ábrán látható séma.

A PGP három RSA-kulcshosszúságot támogat. A felhasználótól függ, hogy kiválassza a legmegfelelőbbet. A hosszúságok a következők:

1. Mindennapi (348 bit): sok pénzzel rendelkezők feltörhetik.
2. Kereskedelmi (512 bit): a hárombetűs szervezetek esetleg fel tudják törni.
3. Katonasági (1024 bit): senki a világon nem töri fel.
4. Földön kívüli (2048 bit): semmilyen más bolygóról érkezett lények sem törhetik fel.

Mivel az RSA csak két rövid számítás erejéig kell, mindenkinek érdemes minden esetben földön kívüli erősségű kulcsokat használnia.



8.45. ábra. *Egy PGP-üzenet*

A 8.45. ábra egy klasszikus PGP-üzenet formátumát mutatja. Emellett számos más formátum is használatban van. Az üzenet három részből áll, melyek az IDEA-kulcsot, az aláírást, illetve magát az üzenet törzsét tartalmazzák. A kulcsos rész nem csak egy kulcsot, hanem egy kulcsazonosítót is tartalmaz, mivel a felhasználóknak több nyilvános kulcsuk is lehet.

Az aláírás rész tartalmaz egy fejrészt, amellyel most nem foglalkozunk. A fejrészt egy időbélyeg, a küldő nyilvános kulcsának azonosítója, ami a hash-aláírás visszakódolására használható, némi típusinformáció a használt algoritmusokról (hogy lehetővé tegye az MD6 és az RSA2 használatát, ha feltalálják őket) és a kódolt hash-érték követi.

Az üzenetrész is tartalmaz egy fejrészt, valamint tartalmazza az alapértelmezett fájlnevet, ha a fogadó a fájlt lemezre írja, egy üzenet-létrehozási időbélyeget és végül az üzenetet magát.

A kulcskarbantartásra nagy figyelmet szenteltek a PGP-ben, mivel ez minden biztonsági rendszer Achilles-sarka. Minden felhasználó két helyi adatbázist tart fent: az egyéni kulcskarikát és a nyilvános kulcskarikát. Az **egyéni kulcskarika** (**private key ring**) egy vagy több titkos-nyilvános kulcspárt tartalmaz. A több kulcs megengedésének oka az, hogy a felhasználónak lehetősége legyen a titkos kulcsait időnként, vagy ha veszélyeztetve érzi magát, lecserélni anélkül, hogy érvénytelenítené az éppen előkészületben vagy küldés alatt álló leveleit. Minden párhoz tartozik egy azonosító, így a feladó megadhatja a címzettnek, hogy melyik nyilvános kulcsot használta. Az üzenetazonosítók a nyilvános kulcs alsó 64 helyi értéken levő bitekből állnak. A felhasználók felelősek az azonosítókból származó konfliktusok elkerüléséért. A lemezen levő titkos kulcsok egy speciális (tetszőleges hosszúságú) jelszóval vannak titkosítva, hogy védettek legyenek a hátbatámadásoktól.

A **nyilvános kulcskarika** (**public key ring**) a felhasználó levelezőpartnereinek nyilvános kulcsait tárolja. Ezek az üzenetekhez tartozó üzenetkulcsok visszakódolásához szükségesek. A nyilvános kulcskarika minden bejegyzése tartalmazza nemcsak a nyilvános kulcsot, hanem a hozzá tartozó 64 bites azonosítót és egy jelzést arra nézve, hogy a felhasználó mennyire bízik meg a kulcsban.

A következő problémáról van itt szó. Tegyük fel, hogy a nyilvános kulcsok egy szabad elérésű adatbázisban vannak nyilvántartva. Egy módja annak, hogy Trudy el tudja olvasni Bob titkos e-leveleit az, hogy betör az adatbázisba, és kicseréli valamire Bob nyilvános

kulcsát. Miután Aliz később elhozza Bob állítólagos titkos kulcsát, Trudy véghezvihet egy élőlánctámadást.

Az ilyen támadások megelőzésére, vagy legalábbis a következményeik minimalizálására, Aliznak tudnia kell, hogy mennyire bízhat meg a nyilvános kulcskarikájában levő „Bob kulcsának" nevezett dologban. Ha tudja, hogy Bob személyesen adta azt oda egy lemezen, akkor a megbízhatósági értéket a legmagasabbra állíthatja. A nyilvános kulcsok kezelésének ez a decentralizált, felhasználó által vezérelt megközelítése különbözteti meg a PGP-t a központosított PKI sémáktól.

Az emberek ettől függetlenül időnként mégis egy megbízható kulcsszervertől szerzik be a nyilvános kulcsokat. Az X.509 szabványosítását követően emiatt a PGP is támogatni kezdte az ilyen tanúsítványokat a hagyományos nyilvános kulcskarika eljárás mellett. A PGP összes jelenlegi változata támogatja az X.509-et.

8.8.2. S/MIME

Az IETF vállalkozása az e-levelek biztonságának terén az **S/MIME (Secure/MIME – biztonságos MIME)**, melyet a 2632-től 2643-ig számozott RFC-k írnak le. Ez hitelesítést, sértetlenséget, titkosságot és letagadhatatlanságot biztosít, mindezt meglehetősen rugalmas módon, többféle kriptográfiai algoritmus alkalmazásával. A névből adódóan nem meglepő az a tény, hogy az S/MIME jól illeszkedik a MIME-höz, és mindenfajta üzenet védelmét lehetővé teszi. Definiáltak különféle új MIME-fejrészeket, például a digitális aláírások tárolására.

Az S/MIME-nek nincs olyan merev tanúsítvány-hierarchiája, mely egyetlen gyökérből indulna ki, amely korábban az egyik olyan politikai probléma volt, ami az egyik korábbi rendszer, a PEM (Privacy Enhanced Mail) végzete lett. A felhasználóknak ehelyett több bizalmi horgonyuk lehet. Amíg egy tanúsítványt vissza lehet követni valamilyen bizalmi horgonyig, addig a felhasználó bízik benne, így az érvényesnek tekinthető. Az S/MIME az eddig vizsgált szabványos algoritmusokat és protokollokat használja, ezért részletesebben most nem tárgyaljuk. Aki többre kíváncsi, tanulmányozza az RFC-ket.

8.9. A web biztonsága

Épp most néztünk át két olyan területet, ahol szükség van biztonságra: a kommunikációt és az e-levelezést. Úgy is gondolhatunk ezekre, mint levesre és előételre. Most pedig itt a főfogás ideje: elérkeztünk a webes biztonsághoz. A web az a hely, ahol manapság a legtöbb Trudy lófrál és végzi a piszkos dolgait. A következő szakaszokban megnézünk néhány, a webes biztonsággal kapcsolatos problémát és kérdést.

A webes biztonság területét nagyjából három részre tagolhatjuk. Először is: hogyan lehet az objektumokat és az erőforrásokat biztonságosan megnevezni? Másodszor: hogyan lehet biztonságos, hitelesített összeköttetéseket felépíteni? Harmadszor: mi történik akkor, ha egy weboldal egy darabka futtatható kódot küld el az ügyfélnek? Miután megnéztünk néhány lehetséges fenyegetést, rátérünk mindezekre a kérdésekre.

8.9.1. Fenyegetések

A weboldalak biztonsági problémáiról szinte minden héten olvashatunk az újságokban. A helyzet valóban elég ijesztő. Példaként lássunk pár eddig megtörtént esetet! Először is, az ún. „cracker"-ek már számos szervezet honlapját támadták meg és cserélték ki új honlapra, saját ízlésük szerint. (A bulvársajtó előszeretettel nevezi a számítógépekbe betörő embereket „hacker"-eknek, de a szakmabeliek közül sokan az igazán nagyszerű programozóknak tartják fenn ezt a fogalmat. A betörőket mi is inkább „cracker"-eknek fogjuk nevezni.) A feltört helyek közt vannak a Yahoo, az amerikai hadsereg, a CIA, a NASA és a New York Times oldalai is. A crackerek a legtöbb esetben csak valamilyen vicces szöveget helyeztek el az oldalon, és a webhelyet pár órán belül ki lehetett javítani.

Most viszont nézzünk meg néhány sokkal komolyabb esetet! Számos webhelyet kényszerítettek térdre szolgáltatás megtagadása (Denial of Service, DoS) típusú támadásokkal, melyekben a cracker elárasztja forgalmával a webhelyet, képtelenné téve azt a legális kérések kiszolgálására. A támadást gyakran egyidejűleg több olyan gépről viszik végbe, melyekbe a cracker előzőleg már betört (DDoS-támadások). Az ilyen támadások annyira mindennaposak, hogy már be sem kerülnek a hírekbe, a megtámadott webhelyeknek mégis több ezer dolláros veszteséget okozhatnak az elveszített üzletek miatt.

1999-ben egy svéd cracker feltörte a Microsoft Hotmail webhelyét, és elkészített egy tükrözött webhelyet, melyen egy Hotmail-felhasználó nevét megadva bárki elolvashatta az adott felhasználó összes aktuális és archivált e-levelét.

Egy másik esetben egy Maxim nevű, 19 éves orosz cracker tört be egy e-kereskedelmi webhelyre, és ellopott 300 000 hitelkártyaszámot. Ezután felvette a kapcsolatot a hely tulajdonosaival, és azt mondta nekik, hogy ha nem fizetnek neki 100 000 dollárt, akkor szétküldi az összes kártyaszámot az interneten. A tulajdonosok nem engedtek a zsarolásnak, ő pedig tényleg szétküldte a kártyaszámokat, komoly kárt okozva ezzel sok ártatlan áldozatnak.

Megint más módszert alkalmazott egy 23 éves kaliforniai diák, aki egy hamis sajtóközleményt küldött egy hírügynökségnek, amelyben azt állította, hogy az Emulex társaság nagy negyedéves veszteség bejelentése előtt áll, a vezérigazgató pedig azonnali lemondásra készül. A vállalat részvényeinek árfolyama órákon belül 60%-ot zuhant, ami a részvényeseknek 2 milliárd dollárnál is nagyobb veszteséget okozott. Az elkövető negyedmillió dollárt keresett a dolgon, mivel közvetlenül a bejelentés előtt eladta a részvényeit. Ez az eset ugyan nem egy webhely feltörése volt, mégis nyilvánvaló, hogy hasonló eredménnyel járna az, ha bármely nagyvállalat honlapján helyezne el valaki ilyen bejelentést.

Sajnos sok oldalon át folytathatnánk még a sort. Most azonban ideje lesz szemügyre vennünk néhány műszaki kérdést is a webes biztonságra vonatkozóan. A különböző típusú biztonsági problémákról további információval szolgálnak Anderson [2008a], Stuttard és Pinto [2007], valamint Schneier [2004] munkái. Az interneten keresgélve is rengeteg konkrét eset leírására bukkanhatunk.

8.9.2. Biztonságos névkezelés

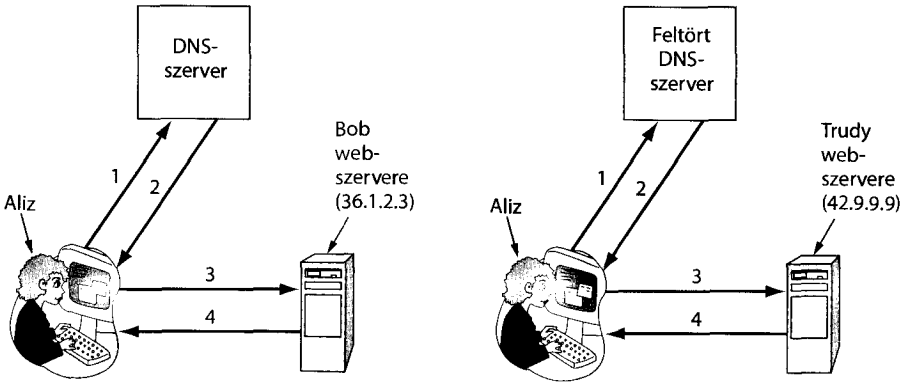
Kezdetnek vegyünk egy nagyon alapvető példát: tegyük fel, hogy Aliz szeretné meglátogatni Bob weboldalát. Begépeli hát Bob URL-jét a böngészőjébe, és pár másodperccel

később meg is jelenik egy weboldal. De vajon valóban Bob oldala az? Talán igen, talán nem. Lehet, hogy megint itt van Trudy a régi trükkjeivel. Elfoghatja például Aliz összes kimenő csomagját, és megvizsgálhatja azokat. Ha elfog egy olyan HTTP *GET* kérést, mely Bob webhelye felé tart, akkor maga is elmehet Bob weboldalára, hogy megszerezze az oldalt, amit aztán tetszése szerint módosíthat, és így, meghamisítva adhat vissza Aliznak. Aliznak minderről fogalma sem lesz. Még ennél is rosszabb, hogy Trudy átírhatja Bob e-kereskedésének árait, hogy a nagyon vonzónak tűnő ajánlatokkal rávegye Alizt arra, hogy adja meg a hitelkártyaszámát „Bobnak", amikor valamilyen árut vásárol.

Az ilyen klasszikus közbeékelődéses támadások egyik hátránya az, hogy Trudynak olyan helyzetben kell lennie, hogy el tudja fogni Aliz kimenő forgalmát, és meg tudja hamisítani a beérkező forgalmát. A gyakorlatban ehhez vagy Aliz, vagy Bob telefonvonalát kell megcsapolnia, mivel a fényvezetőszálas gerinchálózathoz elég nehéz hozzáférni. A vezetékek aktív megcsapolása kétségkívül járható út, de meglehetősen sok munkával is jár, és Trudy a ravaszsága mellett elég lusta is. Aliz becsapásának azonban vannak egyszerűbb módjai is.

A DNS megtévesztése

Tegyük fel, hogy Trudy be tud törni a DNS-rendszerbe, vagy esetleg épp Aliz internetszolgáltatójának DNS-gyorstárába, és lecseréli Bob IP-címét (ami mondjuk 36.1.2.3) a saját IP-címére (mondjuk 42.9.9.9). Ez a következő támadás előtt nyitja meg az utat. A 8.46.(a) ábra a támadás feltételezett lezajlását mutatja. Itt (1) Aliz elkéri a DNS-től Bob IP-címét, (2) megkapja azt, (3) elkéri Bob honlapját, és (4) azt is megkapja. Miután



8.46. ábra. *(a) Normális helyzet. (b) A DNS feltörésén és Bob rekordjának módosításán alapuló támadás*

Trudy Bob DNS-rekordjában Bob IP-címét a sajátjára cserélte ki, a 8.46.(b) ábra szituációjához jutunk. Itt, amikor Aliz kikeresi Bob IP-címét, akkor valójában Trudyét kapja meg, így minden Bobnak szánt forgalma Trudyhoz fog menni. Trudy ezután anélkül is véghezviheti a közbeékelődéses támadást, hogy bármilyen vezeték megcsapolásával bajlódnia kéne. Ehelyett most a DNS-kiszolgálót kell feltörnie, és egy rekordot kell megváltoztatnia, ami már sokkal könnyebb feladat.

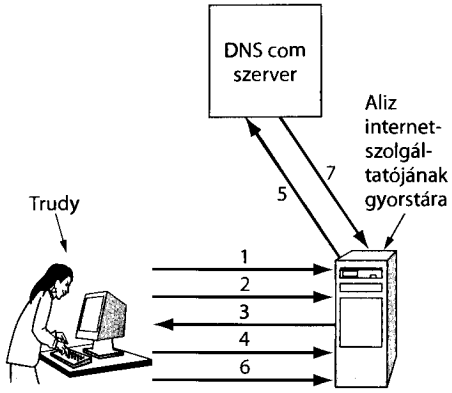
De hogyan tudja Trudy becsapni a DNS-t? Mint kiderül, ez viszonylag egyszerű feladat. Röviden összefoglalva, Trudy ráveheti Aliz internetszolgáltatójának DNS-kiszolgálóját arra, hogy szétküldjön egy kérdést Bob IP-címére vonatkozóan. A DNS-kiszolgálónak azonban sajnos igazából nincs módja arra, hogy ellenőrizze, ki adta a választ, mivel a DNS UDP-t használ. Trudy úgy aknázhatja ki ezt a tulajdonságot, hogy meghamisítja a várt választ, és ezzel hamis IP-címet juttat be a DNS-kiszolgáló gyorstárába. Az egyszerűség kedvéért feltételezzük, hogy Aliz szolgáltatójának eredetileg nincs bejegyzése Bob weboldalához, a *bob.com*-hoz. Ha mégis van, akkor Trudy megvárhatja, míg annak érvényességi ideje lejár, és később újra próbálkozhat (vagy esetleg más trükkökhöz is folyamodhat).

Trudy azzal kezdi a támadást, hogy elküld egy keresési kérést Aliz internetszolgáltatójának, melyben a *bob.com* IP-címére kérdez rá. Mivel ehhez a DNS-névhez nem tartozik bejegyzés, a gyorstáras kiszolgáló megkérdezi a *com* körzet legfelsőbb szintű kiszolgálóját. Csakhogy Trudy megelőzi a *com* kiszolgálót, és egy hamis választ küld vissza, mely szerint „a *bob.com* címe 42.9.9.9", ami valójában a saját IP-címe. Ha a hamis válasz ér vissza először Aliz szolgáltatójához, akkor az kerül bele a gyorstárba, az igazi választ pedig egy idejétmúlt kérésre vonatkozó kéretlen válasznak fogják tekinteni, és visszautasítják. Azt a támadást, amikor egy DNS-kiszolgálót vesznek rá arra, hogy egy hamis IP-címet vegyen fel, **DNS-megtévesztésnek** (**DNS spoofing**) nevezzük. Az ilyen, szándékosan hamis IP-címet tartalmazó gyorstár neve **mérgezett gyorstár** (**poisoned cache**).

A dolog azért valójában nem ennyire egyszerű. Először is, Aliz internetszolgáltatója megvizsgálja, hogy a válasz valóban a legfelső szintű kiszolgáló IP-címét tartalmazza-e. Trudy azonban azt ír a forrás címmezőjébe, amit csak akar, így ezt a vizsgálatot könnyedén kijátszhatja, hiszen a legfelső szintű kiszolgálók IP-címének mindenki által ismertnek kell lennie.

Másodszor, a DNS-kiszolgálóknál minden kérés egy sorszámot hordoz, hogy a kiszolgálók tudják, melyik kéréshez melyik válasz tartozik. Aliz internetszolgáltatójának megtévesztéséhez Trudynak ismerni kell annak aktuális sorszámát. A sorszámot úgy lehet a legkönnyebben megtudni, ha Trudy saját magának is regisztrál egy körzetet, mondjuk, a *trudy-a-tamado.com*-ot. Tegyük fel, hogy ennek az IP-címe is 42.9.9.9. Trudy egy DNS-kiszolgálót is felállít újdonsült körzetében, a *dns.trudy-a-tamado.com* név alatt. Ez szintén a 42.9.9.9-es IP-címet használja, hiszen Trudynak csak egy számítógépe van. A következő feladat az, hogy Aliz szolgáltatójának tudomására hozza a saját DNS-kiszolgálójának meglétét. Ezt könnyen elérheti: csak annyit kell tennie, hogy lekérdezi Aliz szolgáltatójától a *valami.trudy-a-tamado.com* nevet, aminek hatására Aliz szolgáltatója meg fogja kérdezni a legfelső szintű *com* kiszolgálót, hogy megtudja, ki Trudy új körzetének kiszolgálója.

Ha a *dns.trudy-a-tamado.com* már biztos helyen van Aliz szolgáltatójának gyorstárában, megkezdődhet az igazai támadás. Trudy ekkor lekérdezi Aliz szolgáltatójától



1. valami.trudy-a-tamado.com kikeresése (hogy bekerüljön a szolgáltató gyorstárába)
2. A www.trudy-a-tamado.com kikeresése (hogy megkapjuk a szolgáltató következő sorszámát)
3. Kérés a www.trudy-a-tamado.com-ra vonatkozóan (amely a szolgáltató következő sorszámát, az n-et hordozza)
4. A bob.com címének kikeresése, mint a villám (hogy a szolgáltató lekérdezze a com szervert az 5. lépésben)
5. Legitim kérés a bob.com-ra vonatkozóan, sorszám = n + 1
6. Trudy hamis válasza: Bob címe 42.9.9.9, sorszám = n + 1
7. Igazi válasz (visszautasítva, túl késő)

8.47. ábra. *Hogyan téveszthető meg Trudy Aliz internetszolgáltatóját*

a *www.trudy-a-tamado.com* címét. A szolgáltató erre természetesen elküld egy kérést Trudy DNS-kiszolgálójának. Ez a kérés hordozza azt a sorszámot, amit Trudy keres. Trudy erre mint a villám, megkéri Aliz szolgáltatóját, hogy keresse ki neki Bob címét, majd rögtön meg is válaszolja saját kérdését egy hamis válasszal, mely állítólagosan a legfelső szintű *com* kiszolgálótól jön, és így szól: „a *bob.com* címe 42.9.9.9". Ez a hamis válasz eggyel nagyobb sorszámot hordoz, mint az, amelyet Trudy épp azelőtt kapott meg. Ha már itt tart, elküldhet még egy hamis választ, kettővel nagyobb sorszámmal, sőt akár egy tucatnyit is, egyre nagyobb sorszámokkal – valamelyik biztosan megfelelő lesz, a többit meg egyszerűen eldobják. Amikor Trudy hamis válasza megérkezik, bekerül a gyorstárba, az ezután beérkező igazi választ pedig visszautasítják, hiszen akkor már nem lesz függőben lévő kérdés.

Ha Aliz most keresi ki a *bob.com* címét, akkor a 42.9.9.9-et, vagyis Trudy címét kapja válaszul. Trudy tehát kényelmesen, a saját nappalijában ülve vitt véghez egy sikeres közbeékelődéses támadást. A támadás lépéseit a 8.47. ábra szemlélteti. Ez a speciális támadás kivédhető úgy, hogy a DNS-szerver véletlenszám-azonosítókat használ lekérdezései alkalmával, és nem sorszámokat. Úgy tűnik azonban, hogy minden alkalommal, amikor egy rést betömnek, egy másik keletkezik. Pontosabban, az azonosítók csak 16 bitesek, így ezeken végighaladni könnyű, amikor a számítógép az, amelyik a találgatásokat végzi.

Biztonságos DNS

Ezt az egy konkrét támadást ki lehet védeni azzal, ha a DNS-kiszolgálók véletlenszerű azonosítókat használnak az egyszerű sorszámozás helyett, de úgy tűnik, amint betömünk egy lyukat, felbukkan egy másik. Az igazi gond itt az, hogy a DNS-t akkor fejlesztették ki, amikor az internet még csak néhány száz egyetem kutatási céljait szolgálta, és sem Aliz, sem Bob, sem Trudy nem volt hivatalos erre a partira. A biztonság akkor még nem volt téma; a cél az volt, hogy az internet egyáltalán működjön. Ez a környezet azonban drasztikusan megváltozott az évek során, az IETF így 1994-ben felállított egy munkacsoportot, hogy a DNS-t alapjaitól kezdve biztonságossá tegyék. Ez a (most is futó) projekt a **DNSsec** (**DNS security – DNS-biztonság**) néven ismert, az első eredményeit az RFC 2535-ben tették közzé. A DNSsec-et sajnos még nem sikerült teljeskörűen telepíteni, ezért számos DNS-kiszolgáló még mindig védtelen a megtévesztéses támadásokkal szemben.

A DNSsec az elgondolásait tekintve rendkívül egyszerű. A megoldás nyilvános kulcsú kriptográfián alapul. Minden (a 7.5. ábra értelmében vett) DNS-zóna rendelkezik egy nyilvános/egyéni kulcspárral. A DNS-kiszolgálók minden kiküldött információt aláírnak a származási zóna egyéni kulcsával, így a vevő ellenőrizheti a hitelességet.

A DNSsec három alapvető szolgáltatást nyújt:

1. az adatok származási helyének bizonyítása,
2. nyilvános kulcsok kiosztása,
3. tranzakciók és kérések hitelesítése.

Az első szolgáltatás a legfontosabb: ez ellenőrzi, hogy a visszaadott adatokat jóváhagyta-e a zóna tulajdonosa. A második a nyilvános kulcsok biztonságos tárolását és kinyerését teszi lehetővé. A harmadikra a visszajátszásos és a megtévesztéses támadások elleni védelemnél van szükség. Figyeljük meg, hogy titkosságot biztosító szolgáltatás nincs, hiszen a DNS-ben lévő összes információ nyilvánosnak tekinthető. Mivel a DNSsec fokozatos bevezetése várhatóan több évig is eltart majd, elengedhetetlenül fontos, hogy a biztonságos kiszolgálók együtt tudjanak működni a nem biztonságos társaikkal, vagyis magát a DNS-protokollt nem lehet megváltoztatni. Lássunk most tehát néhány részletet!

A DNS-rekordok RRSet (**Resource Record Set – erőforrás-rekordkészlet**) nevű halmazokba vannak csoportosítva, ahol az ugyanolyan nevű, osztályú és típusú rekordok kerülnek egy halmazba. Egy RRSet tartalmazhat több *A* rekordot is, például akkor, ha egy DNS-név egy elsődleges és egy másodlagos IP-címre is leképezhető. Az RRSet-ek számos új rekordtípussal is kiegészültek (lásd lejjebb). Minden RRSet egy kriptográfiai hash-el van ellátva (például MD5 vagy SHA-1 alkalmazásával). A hash-t a zóna egyéni kulcsával írják alá (például RSA segítségével). Az ügyfeleknek átvitt adategységek tehát aláírt RRSet-ek. Egy aláírt RRSet vételekor az ügyfél ellenőrizheti, hogy azt tényleg a származási zóna egyéni kulcsával írták alá. Ha az aláírás megegyezik, akkor az adatokat elfogadják. Mivel minden RRSet tartalmazza a saját aláírását, az RRSet-eket bármilyen gyorstárban tárolni lehet, még a nem megbízható kiszolgálókon is, és ez sem veszélyezteti a biztonságot.

A DNSsec számos új rekordtípust vezet be. Ezek közül az első a *KULCS (KEY)* rekord. Ez tartalmazza egy zóna, felhasználó, hoszt vagy más főszereplő nyilvános kulcsát, az aláíráshoz használt kriptográfiai algoritmust, az átvitelnél használt protokollt és még néhány egyéb bitet. A nyilvános kulcsot tisztán, önmagában tárolják. X.509 tanúsítványokat nem használnak a nagy helyigényük miatt. Az algoritmus mezőben az 1 jelenti az MD5/RSA-aláírásokat (ez a javasolt választás), más értékek pedig más kombinációknak felelnek meg. A protokollmező az IPsec vagy más biztonsági protokoll használatára utalhat, ha egyáltalán van ilyen.

A második új rekordtípus a *SIG (ALÁÍRÁS)* rekord. Ez a *KEY* rekord által meghatározott algoritmussal aláírt hash-t tartalmazza. Az aláírás az RRSet összes rekordjára

vonatkozik, beleértve az összes *KEY* rekordot is, kivéve magát a *SIG* rekordot. A rekord tartalmazza még az aláírás érvényességének kezdetét és végét is, valamint az aláíró nevét és pár más elemet is.

A DNSsec-et úgy tervezték, hogy a zóna egyéni kulcsát offline lehessen tartani. A zóna adatbázisának tartalmát naponta egyszer vagy kétszer manuálisan (például egy CD-ROM-on) át lehet vinni egy hálózatba nem kötött gépre, mely az egyéni kulcsot tárolja. Ezen a gépen az összes RRSet-et alá lehet írni, és az így előállt *SIG* rekordot vissza lehet szállítani a zóna elsődleges kiszolgálójára CD-ROM-on. Ily módon az egyéni kulcsot egy széfbe zárt CD-ROM-on lehet tárolni, melyet csak akkor tesznek be a hálózatba nem kötött gépbe, amikor az aznapi új RRSet-eket kell aláírni. Az aláírás elvégzése után a kulcs összes másolatát kitörlik a memóriából és a merevlemezről, és a CD-ROM-ot visszateszik a széfbe. Ez az eljárás az elektronikus biztonságot a fizikai biztonságra vezeti vissza, annak kezeléséhez pedig már jobban értenek az emberek.

Az, hogy az RRSet-eket előre aláírják, nagyban meggyorsítja a kérések megválaszolását, hiszen nem kell menet közben a kriptográfiával bajlódni. Ennek ára az, hogy nagy tárterületre van szükség a merevlemezen a DNS-adatbázisok összes kulcsának és aláírásának tárolásához. Egyes rekordok mérete akár az eredeti méret tízszeresére is növekedhet az aláírásnak köszönhetően.

Amikor az ügyfélfolyamat egy aláírt RRSet-et kap, akkor először alkalmaznia kell a származási zóna nyilvános kulcsát, hogy visszafejtse a hash-t, majd saját magának is ki kell számítania a hash-t, és össze kell hasonlítania a két értéket. Ha megegyeznek, akkor az adatok érvényesnek tekinthetők. Ez az eljárás azonban felveti azt a kérdést, hogy hogyan kapja meg az ügyfél a zóna nyilvános kulcsát? Az egyik lehetséges megoldás az, ha a kulcsot egy megbízható kiszolgálótól szerezzük be, egy biztonságos összeköttetésen keresztül (például IPsec használatával).

A gyakorlatban azonban inkább azzal számolhatunk, hogy az ügyfelek előre fel lesznek szerelve az összes legfelső szintű körzet nyilvános kulcsával. Ha Aliz most szeretné meglátogatni Bob webhelyét, akkor elkérheti a DNS-től a *bob.com* RRSet-jét, mely tartalmazni fogja Bob IP-címét, és egy *KEY* rekordot Bob nyilvános kulcsával. Ezt az RRSet-et a legfelső szintű *com* körzet fogja aláírni, ezért érvényességét Aliz könnyedén ellenőrizheti. Egy ilyen RRSet lehetséges tartalmára mutat példát a 8.48. ábra.

Bob nyilvános kulcsának ellenőrzött másolatával felvértezve Aliz már megkérdezheti Bob DNS-kiszolgálójától (melyet Bob üzemeltet), hogy mi a *www.bob.com* IP-címe. Ez az RRSet Bob egyéni kulcsával lesz aláírva, így Aliz ellenőrizheti a Bob által visszaadott

Körzetnév	Élettartam	Osztály	Típus	Érték
bob.com.	86400	IN	A	36.1.2.3
bob.com.	86400	IN	KEY	3682793A7B73F731029CE2737D...
bob.com.	86400	IN	SIG	86947503A8B848F5272E53930C...

8.48. ábra. *Példa a* bob.com *RRSet-jére. A* KEY *rekord tartalmazza Bob nyilvános kulcsát. A* SIG *rekord az* A *és a* KEY *rekordoknak a legfelső szintű* com *kiszolgáló által aláírt hash-e, mely a hitelességük ellenőrzésére szolgál*

RRSet-en lévő aláírás érvényességét. Ha Trudynak sikerül is valahogy egy hamis RRSet-et bejuttatnia bármelyik gyorstárba, Aliz akkor is könnyedén észreveheti, hogy az nem hiteles, mivel a benne szereplő *SIG* rekord helytelen lesz.

A DNSsec emellett arra is ad kriptográfiai eljárást, hogy egy választ egy bizonyos kérdéshez lehessen kötni, megelőzve ezzel azt a fajta megtévesztést, amivel Trudy állt elő a 8.47. ábra példájában. Ez a megtévesztést megakadályozó (opcionális) óvintézkedés a válaszhoz hozzáadja a lekérdező üzenet hash-ét, melyet a válaszadó egyéni kulcsával írnak alá. Mivel Trudy nem ismeri a legfelső szintű *com* kiszolgáló egyéni kulcsát, ezért nem tudja Aliz ISP-jének a lekérdezésre adandó választ meghamisítani. Azt minden további nélkül elérheti ugyan, hogy az ő válasza érkezzen meg előbb, de Aliz ISP-jének DNS-kiszolgálója úgyis vissza fogja utasítani, mivel a hash-elt lekérdezésre vonatkozó aláírása érvénytelen lesz.

A DNSsec néhány további rekordtípust is támogat. A *CERT (TANÚSÍTVÁNYOK)* rekord például a tanúsítványok tárolására használható (például X.509 tanúsítványokhoz). Ezt a rekordot azért biztosították, mert egyesek a DNS-ből egy PKI-t szeretnének kialakítani. Hogy ez valóban be is következik-e, azt még nem lehet tudni. A DNSsec tárgyalását mindenesetre ezzel befejezzük. További részleteket az RFC 2535 tartalmaz.

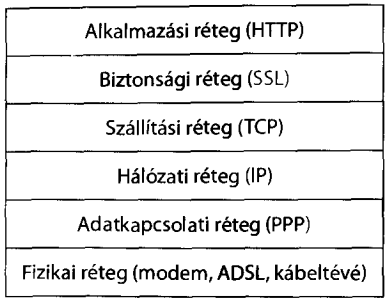
8.9.3. SSL – a biztonságos csatlakozóréteg

A biztonságos névkezelés kezdetnek jó, de a web biztonsága ennél jóval több dolgot takar. A következő lépést a biztonságos összeköttetések jelentik. Most tehát azt fogjuk megvizsgálni, hogy a biztonságos összeköttetéseket hogyan lehet létrehozni.

Amikor a web betört a köztudatba, kezdetben csak statikus oldalak terjesztésére használták. Néhány vállalatnak azonban nem sokkal később az az ötlete támadt, hogy üzleti tranzakciók céljából is használják a webet, hogy online módon lehessen például hitelkártyával vásárolni, banki műveleteket végezni vagy tőzsdézni. Az ilyen alkalmazások miatt jelent meg a biztonságos összeköttetések iránti igény. Az igény kielégítésére 1995-ben a Netscape Communications Corp., az akkoriban domináns böngészőgyártó az **SSL** (**Secure Sockets Layer – biztonságos csatlakozóréteg**) nevű biztonsági csomag bemutatásával állt elő. A szoftvert és annak protokollját ma már széles körben használják, például a Firefox, a Safari és az Internet Explorer, érdemes tehát közelebbről is szemügyre vennünk a részleteit.

Az SSL biztonságos összeköttetést hoz létre két csatlakozó között, és többek között a következő lehetőségeket kínálja:

1. paraméterek egyeztetése az ügyfél és a kiszolgáló között,
2. kölcsönös hitelesítés az ügyfél és a kiszolgáló között,
3. titkos kommunikáció,
4. az adatok sértetlenségének biztosítása.



8.49. ábra. *Egy SSL segítségével böngésző otthoni felhasználó által használt rétegek és protokollok*

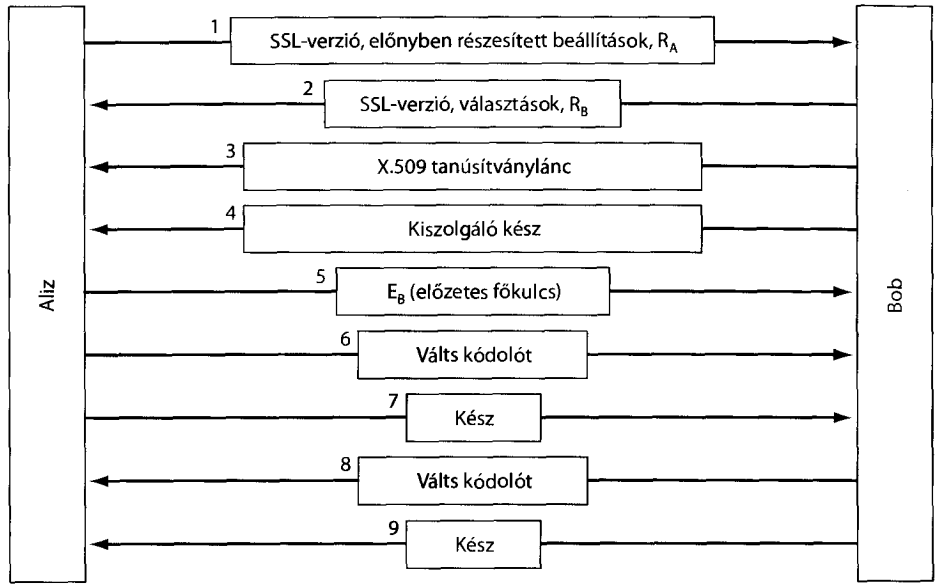
Ezekkel az elemekkel már eddig is találkoztunk, nem igényelnek tehát bővebb magyarázatot.

Az SSL elhelyezkedését a szokásos protokollkészletben a 8.49. ábra szemlélteti. Az SSL gyakorlatilag egy új réteget jelent, mely az alkalmazási és a szállítási réteg közé ékelődik be. Ez fogadja a böngészőből érkező kéréseket, és átadja azokat a TCP-nek a kiszolgálóhoz való továbbításra. A biztonságos összeköttetés kiépítése után az SSL fő feladata a tömörítés és a titkosítás kezelése. Ha a HTTP-t SSL fölött használják, akkor **HTTPS**-nek (**Secure HTTP – biztonságos HTTP**) nevezik, még akkor is, ha maga a protokoll továbbra is csak a szabványos HTTP. Ez néha egy új porton (443) keresztül érhető el a hagyományos (80) helyett. Megjegyezzük, hogy az SSL-t nem csak webböngészőkkel lehet használni, de ez a leggyakoribb alkalmazási módja. Köcsönös hitelesítést is biztosít.

Az SSL-protokoll több verziót is megélt már. Az alábbiakban csak a 3. verziót fogjuk tárgyalni, mely mind közül a legelterjedtebb. Az SSL számos különböző algoritmust és opciót támogat. Az opciók között van a tömörítés megléte vagy hiánya, az alkalmazandó kriptográfiai algoritmusok, valamint néhány, a kriptográfia kivitelének korlátozásaival kapcsolatos lehetőség. Ez utóbbiakat főként annak biztosítására szánták, hogy komoly kriptográfiát csak akkor lehessen használni, amikor az összeköttetés mindkét vége az Egyesült Államokban van. A kulcsok hossza minden egyéb esetben 40 bitre van korlátozva, ami a kriptográfusok szemében tulajdonképpen vicc. A Netscape azért kényszerült ilyen megszorítást alkalmazni, mert csak így kapta meg a kiviteli engedélyt az amerikai kormánytól.

Az SSL két alprotokollból áll: az egyik a biztonságos összeköttetés kiépítésére, a másik pedig annak használatára szolgál. Kezdjük a kiépítés folyamatának vizsgálatával! Az ezért felelős alprotokollt a 8.50. ábra mutatja be. Ez az 1-es üzenettel kezdődik, melyben Aliz elküld egy kérést Bobnak az összeköttetés kiépítésére vonatkozóan. A kérés megadja az Aliz által használt SSL verzióját, valamint az Aliz által előnyben részesített tömörítési módot és kriptográfiai algoritmusokat. Tartalmaz még ezenkívül egy R_A véletlen számot is, mely később kerül felhasználásra.

Most Bob következik. Ő a 2-es üzenetben választ egyet az Aliz által felkínált algoritmusok közül, és elküldi a saját véletlen számát, R_B-t, a 3-as üzenetben pedig elküld egy tanúsítványt, mely a saját nyilvános kulcsát tartalmazza. Ha a tanúsítvány nem hordozza valamely jól ismert hatóság aláírását, akkor elküld még mellé egy tanúsítványláncot, melynek segítségével már el lehet jutni egy ilyen hatósághoz. Minden böngésző, így Aliz



8.50. ábra. *Az SSL összeköttetést felépítő alprotokolljának egyszerűsített változata*

böngészője is eleve tartalmaz mintegy 100 nyilvános kulcsot, így ha Bob ki tud építeni egy olyan láncot, mely ezek közül valamelyikhez kapcsolódik, akkor Aliz képes lesz Bob nyilvános kulcsának ellenőrzésére. Ezen a ponton Bob egyéb üzeneteket is elküldhet (például elkérheti Aliz nyilvános kulcsának tanúsítványát). Ha Bob elkészült, elküldi Aliznak a 4-es üzenetet, hogy tudassa vele: rajta a sor.

Aliz erre véletlenszerűen kiválaszt egy 384 bites **előzetes főkulcsot** (**premaster key**), majd elküldi azt Bobnak az ő nyilvános kulcsával titkosítva (5-ös üzenet). Az adatok titkosítására használt tényleges viszonykulcsot a mindkét véletlen számmal kombinált előzetes főkulcsból származtatják, bonyolult módon. Az 5-ös üzenet beérkezése után Aliz és Bob is ki tudja már számolni a viszonykulcsot. Épp ezért Aliz meg is kéri Bobot arra, hogy álljon át az új kódolásra (6-os üzenet), majd azt is közli vele, hogy végzett a kiépítési alprotokollal (7-es üzenet). Bob ezután nyugtázza Aliz üzeneteit (8-as és 9-es üzenet).

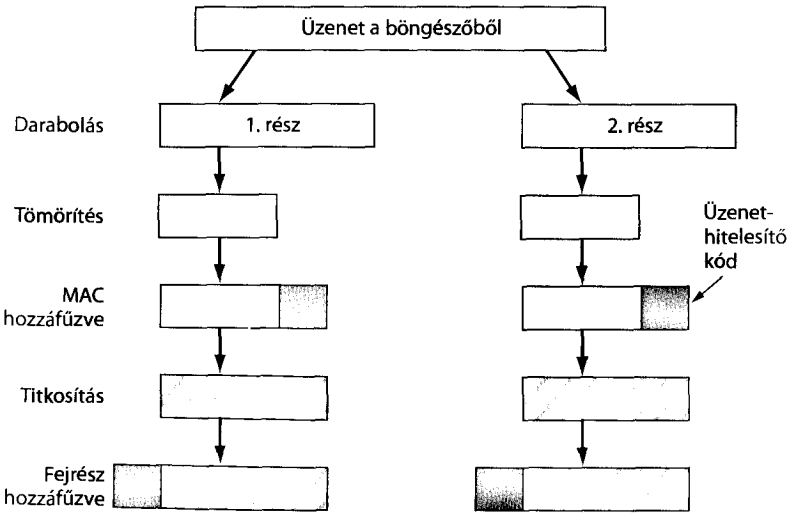
Most az a helyzet állt elő, hogy Aliz tudja ugyan, hogy kicsoda Bob, de Bob nem tudja, hogy ki Aliz (hacsak nincs Aliznak nyilvános kulcsa és egy ahhoz tartozó tanúsítványa, ami elég valószínűtlen egy magánszemély esetében). Bob első üzenetében ezért jó esélyel arra fogja kérni Alizt, hogy lépjen be egy előzőleg rögzített belépési név és jelszó segítségével. A beléptetés protokollja ugyanakkor már nem tartozik az SSL hatáskörébe. A lényeg az, hogy akárhogy is történjék a beléptetés, utána megkezdődhet az adatátvitel.

Mint már említettük, az SSL többféle kriptográfiai algoritmust is támogat. Ezek közül a legerősebb algoritmus a titkosításhoz háromszoros DES-et használ három különböző kulccsal, a sértetlenség biztosításához pedig SHA-1-et. Ez a kombináció viszonylag lassú, ezért leginkább a banki és ehhez hasonló alkalmazásokban használják, ahol a legmagasabb fokú biztonságra van szükség. Az átlagos e-kereskedelmi alkalmazásokban

a titkosításhoz 128 bites kulcsú RC4-et, a hitelesítéshez pedig MD5-öt használnak. Az RC4 a 128 bites kulcsot kezdőértéknek (seed) veszi, és belső használatra egy sokkal nagyobb számot állít elő belőle. A későbbiekben aztán ezzel a belső számmal készíti el a kulcsfolyamot. A kulcsfolyamot KIZÁRÓ VAGY kapcsolatba hozzák a nyílt szöveggel, így egy olyan klasszikus folyamkódolót kapnak, amilyet a 8.14. ábrán láthattunk. Az exportra szánt verziók szintén 128 bites kulcsú RC4-et használnak, de a bitek közül 88-at nyilvánosságra hoznak, hogy a kódot könnyen fel lehessen törni.

A tényleges átvitelre egy másik alprotokollt használnak, amit a 8.51. ábra mutat be. A böngészőből érkező üzeneteket itt először legfeljebb 16 KB-os egységekre darabolják. Ha a tömörítés engedélyezett, akkor ezután minden egységet külön tömörítenek. Ezt követően a két véletlen számból (nonce) és az előzetes főkulcsból származó titkos kulcsot konkatenálják a tömörített szöveggel, az eredményből pedig hash-t képeznek a megegyezés szerinti hash-algoritmussal (ez rendszerint az MD5). Ezt a hash-t aztán üzenethitelesítő kódként (message authentication code, MAC) minden egyes részhez hozzáfűzik. A tömörített részeket az MAC-vel együtt ezután a megállapodás szerinti szimmetrikus titkosító algoritmussal kódolják (általában KIZÁRÓ VAGY kapcsolatba hozzák az RC4 kulcsfolyammal). Végül egy fejrészt illesztenek az egyes részekhez, és átviszik azokat a TCP-összeköttetésen.

Itt azonban elkel egypár óvatosságra intő szó. Tudjuk, hogy az RC4-ről kiderült, hogy van néhány gyenge kulcsa, melyeket könnyen vissza lehet fejteni, ezért az SSL biztonsága is ingoványos talajon áll [Fluhrer és mások, 2001]. Az olyan böngészőket tehát, melyekben a kódoló készletet a felhasználó választhatja meg, mindig a 168 bites kulcsú háromszoros DES és az SHA-1 használatára kell beállítani, még akkor is, ha ez a kombináció lassabb az RC4-nél és az MD5-nél. Vagy ami még ennél is jobb, a felhasználó frissítse a böngészőt, hogy az támogassa az SSL utáni megoldást, amelyet rövidesen leírunk. Az SSL további problémája, hogy a főszereplők nem mindig rendelkeznek tanúsítvá-



8.51. ábra. *Adatátvitel SSL segítségével*

nyokkal, vagy ha igen, még akkor sem mindig ellenőrzik, hogy a felhasznált kulcsok megfelelnek-e azoknak.

1996-ban a Netscape Communications Corp. átadta az SSL-et az IETF-nek szabványosítás céljából. Az eredmény a **TLS** (**Transport Layer Security – szállítási rétegbeli biztonság**) protokoll lett, melyet az RFC 2246 ír le.

A TLS az SSL 3. változatára épül. Az SSL-en csak viszonylag kisebb változtatásokat hajtottak végre, ami ahhoz mégis elég volt, hogy az SSL 3. verziója és a TLS ne tudjon együtt működni. Megváltoztatták például azt a folyamatot, mely az előzetes főkulcsból és a két véletlen számból állítja elő a viszonykulcsot, mégpedig azért, hogy erősebb kulcsot kapjanak (vagyis olyat, amit nehezebb megfejteni). Emiatt az inkompatibilitás miatt a legtöbb böngésző megvalósítja mindkét protokollt úgy, hogy a TLS visszavedlik SSL-re az egyeztetés során, ha ez szükséges. Erre a megoldásra úgy hivatkoznak, mint SSL/TLS. Az első TLS-megvalósítások 1999-ben jelentek meg az 1.2 változattal, amelyet 2008 augusztusában definiáltak. Ez támogatja az erősebb titkosító eljárásokat (például AES). Az SSL piacon elfoglalt helye erős maradt, bár a TLS fokozatosan ki fogja szorítani

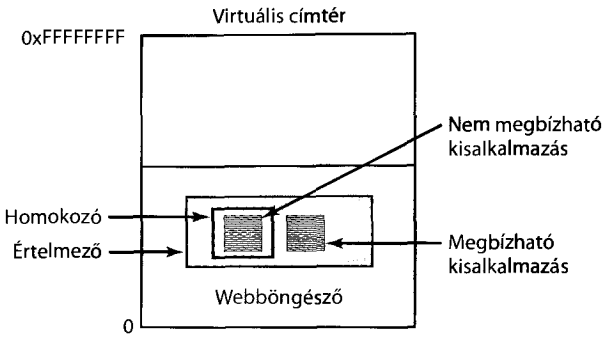
8.9.4. A hordozható kódok biztonsága

A névkezelésen és az összeköttetéseken kívül egyéb területei is vannak a webes biztonságnak. A kezdeti időkben a weboldalak pusztán statikus HTML-állományok voltak, így nem tartalmaztak még futtatható kódot sem. Manapság viszont gyakran találunk bennük apró programokat, köztük Java-kisalkalmazásokat, ActiveX-vezérlőket és JavaScripteket. Az ilyen **hordozható kódok (mobile code)** letöltése és futtatása nyilvánvalóan komoly biztonsági kockázatokat rejt, melyek minimalizálására számos eljárást javasoltak. Most röviden áttekintünk néhány olyan kérdést, melyet a hordozható kódok vetnek fel, majd megnézünk néhány megközelítést ezek kezelésére.

A Java-kisalkalmazások biztonsága

A Java-kisalkalmazások (appletek) apró Java-programok, amelyeket egy veremorientált, **JVM (Java Virtual Machine – Java virtuális gép)** nevű gépi nyelvre fordítottak le. Ezeket egy weboldalon is el lehet helyezni, ekkor az oldallal együtt ezek is letöltődnek. A letöltés után a kisalkalmazások egy böngészőn belüli JVM-értelmezőbe kerülnek, amint azt a 8.52. ábra szemlélteti.

Az értelmezett kódok futtatásának előnye a lefordított kódokkal szemben, hogy az előbbinél az értelmező a végrehajtás előtt minden utasítást megvizsgál. Ez lehetőséget nyújt annak ellenőrzésére, hogy az utasítás címe érvényes-e. Ezenfelül a rendszerhívásokat is elkapják és értelmezik. Az ilyen hívások kezelése a biztonsági politikától függ. Például, ha egy kisalkalmazás megbízható (például a helyi lemezről származik), akkor annak rendszerhívásait minden további nélkül végre lehet hajtani. Ha azonban nem megbízható (például az internetről érkezett), akkor egy úgynevezett **homokozóba (sandbox)** lehet zárni, hogy korlátok közé szorítsuk működését, és meggátoljuk abban, hogy hozzáférhessen a rendszer erőforrásaihoz.



8.52. ábra. *A kisalkalmazásokat a webböngésző is értelmezheti*

Amikor egy kisalkalmazás egy rendszererőforráshoz próbál hozzáférni, akkor a hívását egy biztonsági felügyelőnek adják át jóváhagyásra. A felügyelő a helyi biztonságpolitika fényében vizsgálja meg a hívást, majd eldönti, hogy elfogadja vagy visszautasítja-e azt. Ily módon lehetőség nyílik annak biztosítására, hogy a kisalkalmazások csak bizonyos erőforrásokhoz férhessenek hozzá. Sajnos az az igazság, hogy ez a biztonsági modell gyengén működik, és folyton-folyvást hibák bukkannak fel benne.

ActiveX

Az ActiveX-programok Pentium processzorokra írt bináris programok, melyeket weboldalakba lehet ágyazni. Ha a böngésző egy ilyennel találkozik, akkor megnézi, hogy szabad-e futtatni, és ha igen, akkor futtatja azt. Nincs semmilyen értelmezés, se homokozó, az ilyen programoknak tehát éppen annyi lehetőségük van, mint más felhasználói programoknak, így esetleg nagy károkat is okozhatnak. Az összes biztonság egyedül annak eldöntésében összpontosul, hogy futtatható-e az ActiveX-vezérlő vagy sem.

A Microsoft egy olyan eljárást választott ezen döntés meghozatalára, mely a **kódok aláírásának** (**code signing**) elvén alapszik. Minden ActiveX-vezérlőt egy digitális aláírás kísér – ez egy, a kódból képzett hash, melyet annak készítője nyilvános kulcsú kriptográfia segítségével aláírt. Amikor egy ActiveX-vezérlő felbukkan, a böngésző először az aláírást ellenőrzi, hogy meggyőződjön róla, hogy nem nyúltak hozzá a kódhoz az átvitel során. Ha az aláírás megfelelő, akkor a böngésző megnézi a belső táblázatában, hogy a program készítője megbízható-e, vagy van-e tőle induló bizalmi lánc egy megbízható szerzőhöz. Ha a szerző megbízható, akkor a programot futtatják; különben nem. A Microsoft ActiveX-vezérlők ellenőrzésére szolgáló rendszerét **Authenticode**-nak nevezik.

Érdemes összehasonlítani a Java és az ActiveX megközelítését. A Java nem tesz kísérletet arra, hogy kiderítse, ki írta a kisalkalmazást. Ehelyett egy futási időben működő értelmező biztosítja, hogy a kisalkalmazás ne tegyen semmi olyat, amit a gép tulajdonosa szerint a kisalkalmazások nem tehetnek meg. Ezzel szemben az aláírt kódoknál nem próbálják meg felügyelni a hordozható kód futási idejű viselkedését. Ha a kód megbízható forrásból érkezett, és nem módosították útközben, akkor futhat. Meg sem kísérlik megvizsgálni azt, hogy a kód vajon rosszindulatú vagy sem. Az eredeti szerző

szándékosan is megírhatja úgy a kódot, hogy az formázza le a merevlemezt, majd törölje a flash ROM tartalmát, hogy a számítógépet soha többé ne lehessen elindítani. Ha ez a szerző ezután megkapja a megbízhatósági tanúsítványt, akkor a kódot futtatni fogják, az pedig tönkreteszi a számítógépet (hacsak nem tiltották le az ActiveX-vezérlőket a böngészőben).

Sokan érzik úgy, hogy ijesztő dolog egy ismeretlen szoftvercégben megbízni. A probléma szemléltetésére egy seattle-i programozó megalapított egy szoftvercéget, majd megbízhatónak tanúsíttatta azt, amit nem nehéz elérni. Írt ezután egy olyan ActiveX-vezérlőt, mely szabályszerűen lekapcsolta a számítógépet, és elkezdte a vezérlőjét széles körben terjeszteni. Sok számítógépet sikerült lekapcsolnia, de azokat egyszerűen újra lehetett indítani, tehát nem okozott kárt, mindössze megpróbálta felhívni a világ figyelmét a problémára. A hivatalos reakció az volt, hogy visszavonták a tanúsítványát az adott vezérlőre vonatkozóan. Ez véget vetett ugyan a közvetlen kellemetlenségeket okozó rövidke időszaknak, de a felszín alatt rejlő problémát még mindig kihasználhatja egy rosszindulatú programozó [Garfinkel és Spafford, 2002]. Mivel nincs arra mód, hogy az esetleg hordozható kódokat is előállító több ezer céget felügyeljék, a kódok aláírásának módszere valójában egy ketyegő időzített bomba.

JavaScript

A JavaScriptnek nincs semmilyen formális biztonsági modellje, történelmét pedig végigkísérik a biztonsági lyukakkal teli megvalósítások. A biztonságot minden gyártó más módon kezeli. A Netscape Navigator 2. verziója például egy, a Javáéhoz hasonló modellt alkalmazott, a 4. verzióban viszont már felhagytak ezzel, és áttértek a kódaláírásos modellre.

Az alapvető probléma az, hogy egy idegen kód futtatása a számítógépen jó eséllyel csak a bajok forrása lehet. A biztonság szempontjából olyan ez, mintha az ember egy betörőt hívna meg a házába, aztán próbálná alaposan szemmel tartani, nehogy kiszökjön a konyhából a nappaliba. Ha valami váratlan következik be, és az ember figyelmét egy pillanatra elvonják, akkor csúnya dolgok történhetnek. A feszültséget itt az okozza, hogy a hordozható kódok látványos grafikát és gyors interakciót tesznek lehetővé, és sok webtervező ezt jóval fontosabbnak tartja, mint a biztonságot, különösen akkor, ha valaki másnak a gépe van veszélyben.

Böngésző-kiterjesztések

Ugyanúgy, mint a weblapok kóddal történő kiterjesztésének, a **böngészők kiterjesztésének**, a **kiegészítéseknek** (**add-on**) és a **beépülő moduloknak** (**plug-in**) is óriási piaca van. Ezek számítógépes programok, amelyekkel a webböngészők több funkciót tudnak ellátni. A beépülő modulokkal értelmezni lehet vagy meg lehet jeleníteni bizonyos típusú tartalmakat, például PDF-dokumentumokat vagy Flash-animációkat. A kiterjesztések és kiegészítések a böngészőt vagy új képességekkel látják el, amilyen például a jobb jelszókezelés, vagy együttműködési módokat biztosítanak weblapokkal például azáltal, hogy megjelölik azokat vagy hogy lehetővé teszik megadott termékek könnyű vásárlását.

Egy kiterjesztés, egy kiegészítés vagy egy beépülő modul telepítése nem bonyolultabb annál, mint böngészéssel elérni a kívánt objektumot, és rákattintani a hiperhivatkozásra a programok telepítéséhez. Ennek a tevékenységnek az eredményeként az interneten keresztül kód fog letöltődni és a böngészőbe települni. Az összes ilyen programot keretrendszerekben írják, amelyek különbözők lehetnek attól a böngészőtől függően, amelyet feljavítanak. Első közelítésben azonban ezek a böngésző hiteles számítási bázisának részévé válnak. Azaz, ha a kód, amelyet telepítünk hibás, az egész böngésző veszélybe kerülhet.

Van még két további nyilvánvaló hibalehetőség. Az első az, hogy a program esetleg rosszindulatúan viselkedik, például azzal, hogy összegyűjt személyes információkat és elküldi azt egy távoli szerverre. Mindenesetre a böngésző úgy tudja, hogy a felhasználó éppen ezért telepítette a kiterjesztést. A második probléma az, hogy a beépülő modulok a böngészőt képessé teszik arra, hogy új típusú tartalmat értelmezzen. Ez a tartalom gyakran maga egy teljesen kiterjesztett programnyelv. A PDF és a Flash jó példa erre. Amikor a felhasználó PDF-fel oldalakat vagy Flash-sel tartalmat néz, a beépülő modulok a böngészőjükben végrehajtják a PDF- vagy Flash-kódot. Ennek a kódnak biztonságosnak kellene lennie, azonban gyakran vannak benne gyenge biztonsági pontok, amelyek kihasználhatók. Mindezek miatt a kiegészítéseket és a beépülő modulokat csak akkor kell telepíteni, amikor szükség van rájuk, és csak megbízható szállítótól szabad ezeket letölteni.

Vírusok

A vírusok a hordozható kódok egy másik formáját jelentik. A fenti példáktól eltérően az ember semmilyen formában nem hívja meg a vírusokat a számítógépére. Egy vírus és a hagyományos hordozható kód között az a különbség, hogy a vírusokat úgy írják meg, hogy reprodukálják önmagukat. Amikor egy weboldalról, egy e-levél mellékletből vagy valami más úton-módon megérkezik egy vírus, akkor általában azzal kezdi a tevékenységét, hogy megfertőzi a háttértáron található futtatható programokat. Ha ezek után egy ilyen fertőzött programot futtatunk, akkor az irányítás átkerül a vírushoz, ami rendszerint megpróbálja átterjeszteni magát más gépekre is, például úgy, hogy másolatokat küld magáról az áldozat e-levél címjegyzékében található összes személynek. Egyes vírusok a merevlemez rendszerindító szektorát (boot sector) fertőzik meg, így a gép indításakor a vírus máris futni fog. A vírusok rendkívül súlyos problémát jelentenek az interneten, és már eddig is több milliárd dollárnyi kárt okoztak. A problémára egyértelmű megoldás nem létezik. Talán az olyan teljesen új generációs operációs rendszerek enyhítenének a gondokon, melyek biztonságos mikrokerneleken alapulnak, és szigorúan elkülönítik egymástól az egyes felhasználókat, folyamatokat és erőforrásokat.

8.10. Társadalmi kérdések

Az internet és annak biztonságtechnikája olyan terület, ahol a társadalmi kérdések, a közcélú szabályozás és a technika frontálisan ütköznek egymással, ez pedig gyakran súlyos következményekkel jár. A következőkben ennek három vetületét fogjuk röviden

megvizsgálni: a személyiségi jogok, a szólásszabadság és a szerzői jogok kérdéseit. Talán mondani sem kell, hogy leírásunkban éppen csak érinteni tudjuk ezen témák felszínét. Bővebb információval Anderson [2008a], Garfinkel és Spafford [2002] és Schneier [2004] munkái szolgálnak. Az internet is tele van a témával kapcsolatos anyagokkal. Elég, ha beírjuk valamelyik keresőbe a „személyiségi jogok", a „cenzúra" vagy a „szerzői jogok" szavakat (angolul: „privacy", „censorship" és „copyright"). Könyvünk honlapja a *http://www.pearsonhighered.com/tanenbaum* címen szintén tartalmaz néhány hivatkozást.

8.10.1. A személyiségi jogok védelme

Vannak az embereknek egyáltalán személyiségi jogaik? Jó kérdés. Az USA alkotmányának 4. kiegészítése megtiltja a kormánynak, hogy alapos ok nélkül kutasson az emberek otthonaiban, illetve a papírjaik vagy ingóságaik között, valamint azon feltételeket is korlátozza, melyek mellett házkutatási parancsot lehet kiadni. A személyiségi jogok ily módon tehát már több mint 200 éve napirenden vannak a nyilvánosság előtt, legalábbis az USA-ban.

Az elmúlt évtized azonban változásokat hozott ezen a téren: a kormányok immár könnyebben meg tudják figyelni az állampolgáraikat – ugyanakkor az állampolgárok is könnyebben meg tudják akadályozni az ilyen megfigyeléseket. A 18. században, ha a kormány át akarta kutatni egy polgár papírjait, akkor ki kellett küldenie egy lovas rendőrt a polgár farmjára, hogy az megkövetelje bizonyos dokumentumok bemutatását. Ez meglehetősen körülményes eljárás volt. Manapság a telefontársaságok és az internetszolgáltatók készségesen megcsapolják a vezetékeiket, ha egy házkutatási parancsot dugnak az orruk alá. Ez nagyban megkönnyíti a szegény rendőr életét, akit immár a lóról való leesés veszélye sem fenyeget.

A kriptográfia megváltoztatja ezt a világot. Ha valaki veszi a fáradságot ahhoz, hogy letöltse és telepítse a PGP-t, és jól őrzött földön kívüli erősségű kulcsokat használ, akkor gyakorlatilag biztos lehet abban, hogy az ismert univerzumban senki nem fogja tudni elolvasni az e-leveleit, házkutatási parancs ide vagy oda. A kormányok jól tudják ezt, és nincs ínyükre a dolog. A személyiségi jogok igazi védelme azt jelenti, hogy nemcsak a különféle bűnözőket, hanem az újságírókat és a politikai ellenfeleket is sokkal nehezebb megfigyelni. Ennek eredményeként egyes kormányok korlátozzák vagy tiltják a kriptográfiai eszközök használatát, illetve kivitelét. Franciaországban például 1999 előtt mindenfajta kriptográfia alkalmazása tilos volt – kivéve, ha a kormány is megkapta a kulcsokat.

A francia példa nem egyedi. Az amerikai kormány 1993 áprilisában jelentette be azt a szándékát, mely szerint egy hardveres kriptoprocesszort, egy ún. **lehallgató chip**-et (**clipper chip**) készül bevezetni, ami minden hálózati kommunikáció szabványos része lesz. Azt mondták, hogy ezzel az állampolgárok személyiségi jogait kívánják biztosítani. Azt is megemlítették, hogy a chip révén a kormány képes lesz minden forgalmat dekódolni a **kulcszálog** (**key escrow**) séma alkalmazásával, ami a kormány számára hozzáférést biztosít az összes kulcshoz. Megígérték persze azt is, hogy megfigyeléseket csak érvényes házkutatási parancsok birtokában végeznek. Mondani sem kell, hogy mindez hatalmas vitákat váltott ki: a személyiségi jogok szószólói teljesen elítélték, a törvények

végrehajtásáért felelős hivatalnokok pedig magasztalták a tervet. A kormány végül visszavonult, és elvetette az ötletet.

Az elektronikus személyiségi jogok témájában rengeteg információt találhatunk az Electronic Frontier Foundation (Elektronikus Határ Alapítvány) webhelyén, a *www.eff.org* cím alatt.

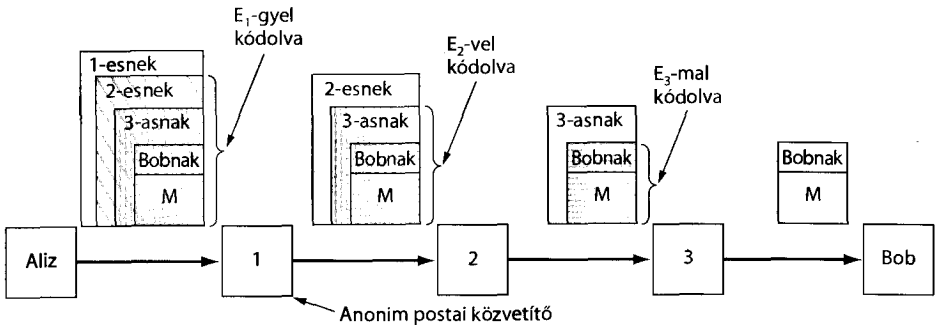
Anonim postai közvetítők

A PGP, az SSL és a hasonló technológiák lehetővé teszik, hogy két fél biztonságos és hiteles kommunikációt folytathasson, mentesülve bármilyen harmadik fél felügyeletétől és beavatkozásától. A személyiségi jogokat azonban néha azzal lehet a legjobban védeni, ha *nem* alkalmazunk hitelesítést, vagyis valójában anonim kommunikációt folytatunk. Az anonimitás a kétpontos üzenetváltásoknál éppúgy kívánatos lehet, mint a hírcsoportoknál.

Tekintsünk néhány példát! Először is, a tekintélyuralmi rendszerekben élő politikai ellenzék sokszor anonim módon szeretne kommunikálni, hogy elkerülje a börtönt vagy a halálbüntetést. Másodszor, számos vállalati, oktatási, kormányzati és egyéb intézményben történt visszaélésre meg nem nevezett források derítenek fényt, akik többnyire meg akarják őrizni névtelenségüket, hogy elkerülhessék a megtorló intézkedéseket. Harmadszor, a népszerűtlen társadalmi, politikai vagy vallási nézeteket képviselő emberek esetenként jobban szeretnek egymással e-levelek vagy hírcsoportok útján, személyazonosságuk felfedése nélkül kommunikálni. Negyedszer, vannak, akik az alkoholizmusról, az elmebetegségről, a szexuális zaklatásról, a gyermekekkel szemben elkövetett erőszakról vagy egy üldözött kisebbséghez való tartozásról folytatott eszmecserében nem szeretnének a nyilvánosság elé kerülni. Természetesen számtalan további példa is létezik még.

Nézzünk most meg egy konkrét esetet! Az 1990-es években az egyik nem történelmi egyházi csoport bírálói egy **anonim postai közvetítő (anonymous remailer)** útján küldték el nézeteiket egy USENET hírcsoportba. Ez a kiszolgáló lehetővé tette, hogy a felhasználók álneveket hozzanak létre, és a kiszolgálón keresztül elküldött leveleket ezen álnevek alatt továbbították, így senki nem tudta kideríteni, hogy honnan jött az eredeti üzenet. Egyes levelek olyan dolgokat hoztak nyilvánosságra, amelyek a vallási csoport szerint üzleti titok, illetve szerzői jogvédelem tárgyát képezték. A csoport ezért közölte a helyi hatóságokkal, hogy nyilvánosságra hozták az üzleti titkaikat és megsértették a szerzői jogaikat. Ezek mind bűncselekménynek minősültek ott, ahol a kiszolgáló üzemelt. A dologból bírósági ügy lett, végül a kiszolgáló üzemeltetőjét kötelezték arra, hogy átadja a tárolt leképezési információt, így fény derült a leveleket elküldő személyek valós személyazonosságára. (Mellesleg nem ez volt az első eset, hogy egy vallást a titkainak kisziárogtatása bosszantson: William Tyndale-t 1536-ban azért ítélték máglyahalálra, mert lefordította a Bibliát angolra.)

Az internetes közösség jelentős részét mélységesen felháborította a bizalmasság ilyen fokú megsértése. Mindenki azt a következtetést vonta le, hogy nem sokat ér az az anonim postai közvetítő, amely tárolja a valós e-levél címek és az álnevek közti leképezéseket (ezek az úgynevezett 1-es típusú közvetítők). Ez az eset sokakat ösztönzött arra, hogy olyan anonim közvetítőket tervezzenek, melyekkel szemben tehetetlenek a bírósági határozatok.



8.53. ábra. *Aliz így küld üzenetet Bobnak 3 közvetítőn keresztül*

Ezek az újfajta, **cypherpunk**[3] **postai közvetítőnek** (**cypherpunk remailer**) is nevezett rendszerek a következőképp működnek. A felhasználó előállít egy e-levél üzenetet az összes ahhoz tartozó RFC 822-es fejrésszel együtt (kivéve a *Feladó:* mezőt, természetesen), titkosítja azt a közvetítő nyilvános kulcsával, majd elküldi a közvetítőnek. Ott a külső RFC 822-es fejrészeket eltávolítják, a tartalmat dekódolják és az üzenetet újraküldik. A közvetítőben nincsenek sem felhasználói fiókok, sem naplóállományok, így még ha később le is foglalják a kiszolgálót, akkor sem marad nyoma azoknak az üzeneteknek, melyek keresztülmentek rajta.

Az anonimitásra törekvő felhasználók közül sokan több anonim közvetítőn keresztül fűzik láncba kéréseiket, mint ahogy azt a 8.53. ábra is mutatja. Itt Aliz egy „nagyon nagyon nagyon" anonim Valentin-napi üdvözlőlapot szeretne küldeni Bobnak, ezért három közvetítőt használ. Összeállítja tehát az *M* üzenetet, és rárak egy olyan fejrészt, mely Bob e-levél címét tartalmazza. Ezután az egészet titkosítja a 3-as közvetítő nyilvános kulcsával, E_3-mal (ezt az ábrán vízszintes vonalkázással jelöltük). Mindehhez hozzárak egy olyan fejrészt, mely a 3-as közvetítő e-levél címét tartalmazza nyílt szövegként. Az ábrán az így kapott üzenetet láthatjuk a 2-es és 3-as doboz között.

Következő lépésként Aliz az előbbi üzenetet titkosítja a 2-es közvetítő nyilvános kulcsával, E_2-vel (ezt függőleges vonalkázással jelöltük), majd az eredmény elé helyez egy kódolatlan fejrészt, mely a 2-es közvetítő e-levél címét hordozza. Az így előállt üzenetet láthatjuk a 8.53. ábrán az 1-es és 2-es doboz között. Végül Aliz az egész eddigi üzenetet titkosítja az 1-es közvetítő nyilvános kulcsával, E_1-gyel, és elérak egy kódolatlan fejrészt az 1-es közvetítő e-levél címével. Ez az üzenet látható az ábrán Aliz jobbján, és ez lesz az, amit Aliz végül is elküld.

Amikor az üzenet megérkezik az 1-es közvetítőhöz, eltávolítják a külső fejrészt. Az üzenet törzsét dekódolják, majd elküldik a 2-es közvetítőnek. Ugyanilyen lépéseket hajt végre a másik két közvetítő is.

3 A cypherpunk a cypher (titkosítás) és a punk (ellenszenves viselkedésű fiatal) összevonásából származó kifejezés. Itt: olyan személy, aki titkosítást alkalmaz a számítógép-hálózat használatakor annak érdekében, hogy személyiségi jogait biztosítsa, elsősorban a hatóságokkal szemben. *(A lektor megjegyzése)*

Bár a végső üzenettől rendkívül nehéz visszajutni Alizig, sok közvetítő még további óvintézkedéseket is foganatosít. Visszatarthatják például az üzeneteket egy véletlenszerűen megválasztott ideig, az üzenet végére rakhatnak valamilyen szemetet, illetve el is távolíthatják azt, és át is rendezhetik az üzenetek sorrendjét. Mindezt azért, hogy minél nehezebb legyen megállapítani, hogy a közvetítőből kilépő üzenetek melyik oda beérkező üzenethez tartoznak, vagyis hogy ne lehessen forgalomelemzést végezni. Mazières és Kaashoek [1998] egy korszerű anonim e-levél rendszerről adnak leírást.

Az anonimitás nem csak az e-levelekre korlátozódik. Vannak olyan szolgáltatások is, melyek anonim barangolást tesznek lehetővé a weben, felhasználva a rétegezett útvonal ugyanazon formáját, amelyiken egy csomópont csak a következő csomópontot ismeri a láncban. Ezt a módszert **hagyma-útválasztásnak (onion routing)** hívják, mert minden csomópont lefejt egy újabb réteget a hagymáról annak meghatározására, hogy hová kell küldenie a csomagban lévő szöveget. A felhasználó ezeknél úgy állítja be a böngészőjét, hogy az a névtelenítő szolgáltatást helyettesként (proxy) használja. A TOR (The Onion Router) közismert példa az ilyen rendszerre [Dingledine és mások, 2004]. Ettől kezdve minden HTTP-kérés a névtelenítő hálózaton megy keresztül, vagyis az kéri el az oldalakat, és az is adja vissza azokat. A webhely a kérés forrásaként nem a felhasználót, hanem a névtelenítő hálózat egy kimeneti csomópontját látja. Mindaddig, amíg a névtelenítő hálózat tartózkodik a naplózástól, ily módon utólag senki nem tudja megállapítani, hogy ki kérte az oldalt.

8.10.2. Szólásszabadság

A személyiségi jogok védelme azokra a magánszemélyekre vonatkozik, akik korlátozni akarják azokat az információkat, amit mások megtudhatnak róluk. A második nagyon fontos társadalmi kérdés a szólásszabadság, valamint annak ellentéte, a cenzúra, mely arról szól, hogy a kormányok korlátozni akarják azt, hogy mit olvashatnak és mit tehetnek közzé az emberek. A több millió oldalt tartalmazó web a cenzorok paradicsoma lett. Az adott rezsim természetétől és ideológiájától függően a következő tartalmú oldalak kerülhetnek tilalom alá:

1. az olyan anyagok, melyek nem valók gyermekeknek és fiatalkorúaknak,
2. a különféle etnikai, vallási, szexuális és egyéb csoportok ellen irányuló gyűlöletkeltés,
3. információ a demokráciáról és a demokratikus értékrendről,
4. a kormány verziójának ellentmondó leírások a történelmi eseményekről,
5. használati útmutatók a zárak feltöréséhez, fegyverek gyártásához, üzenetek titkosításához stb.

A reakció általában az, hogy betiltják az ilyen nemkívánatos oldalakat.

Mindez néha kiszámíthatatlan következményekkel járhat. Egyes nyilvános könyvtárak például webszűrőket telepítettek a számítógépeikre, hogy azok a pornográf oldalak

kiszűrése révén gyermekbaráttá váljanak. A szűrők visszautasítják a feketelistájukon szereplő oldalakat, de a többi oldalon is rákeresnek a csúnya szavakra, mielőtt megjelenítenék azokat. A Virginia állambeli Loudoun megyében előfordult olyan eset is, hogy a szűrő letiltotta egy, a mellrákról adatokat gyűjtő törzslátogató keresését is, mert megtalálta benne a „mell" szót. A látogató erre beperelte Loudoun megyét. Ugyanakkor Kalifornia államban, Livermore-ban egy szülő azért indított pert egy közkönyvtár ellen, mert az *nem* telepített szűrőt, miután a 12 éves fiát pornográf képek nézegetésén kapták. Mit tehet ilyenkor egy könyvtár?

Sokan megfeledkeznek arról, hogy a world wide web valóban *világméretű* hálózat, vagyis a teljes világot behálózza, mint azt a neve is mutatja. Nem minden ország ért azonban egyet abban, hogy mit kellene megengedni a weben. 2000 novemberében például egy francia bíróság arra utasította a kaliforniai székhelyű Yahoo!-t, hogy tiltsa le a francia felhasználók elől a weboldalán található náci emléktárgyak árverését, mert az ilyen tárgyak birtoklása sérti a francia törvényeket. A Yahoo! az amerikai bírósághoz fordult a fellebbezésével, ami végül pártját is fogta, de ezzel még korántsem rendeződött annak a kérdése, hogy hol kinek a törvényei érvényesek.

Gondoljunk csak bele! Mi történne, ha valamelyik Utah állambeli bíróság arra kötelezné Franciaországot, hogy tiltsa be a borral foglalkozó weboldalakat, csak azért, mert azok nem felelnek meg Utah sokkal szigorúbb, alkoholra vonatkozó törvényeinek? Vagy mi lenne, ha Kína azt követelné, hogy tiltsák be az összes demokráciával kapcsolatos oldalt, mivel azok nem az Állam érdekeit szolgálják? Vajon Irán vallási törvényei a sokkal liberálisabb svédekre is vonatkoznak? Letilthatja Szaúd-Arábia a nők jogaival foglalkozó weboldalakat? Ez az egész téma egy valóságos Pandora szelencéje.

Álljon itt egy fontos, idevágó megjegyzés John Gilmore-tól: „A hálózat a cenzúrát károsnak tekinti, és megkerüli azt." Ennek egy konkrét megvalósítása az **örökkévalóság-szolgáltatás (eternity service)** [Anderson, 1996]. Itt a szolgáltatás célja annak biztosítása, hogy az egyszer már kiadott információt ne lehessen visszavonni vagy átírni, amint az Sztálin idejében volt szokás a Szovjetunióban. A szolgáltatást igénybe vevő felhasználó meghatározza, hogy meddig szeretné megőriztetni az anyagait, befizet egy, az anyag terjedelmével és a megőrzés idejével arányos díjat, majd feltölti az anyagot. Ezek után senki nem lesz képes azt módosítani vagy eltávolítani, még maga a kezdeményező felhasználó sem.

Hogyan lehet megvalósítani egy ilyen szolgáltatást? A legegyszerűbb megoldás az, ha egy egyenrangú (peer-to-peer) rendszert alkalmazunk, amelyben az egyes dokumentumok több tucatnyi résztvevő kiszolgálóra is felkerülnek. A kiszolgálók üzemeltetőit azzal ösztönzik a rendszerbe való belépésre, hogy átadják nekik a szolgáltatás díjának bizonyos hányadát. A lehető legnagyobb rugalmasság érdekében a kiszolgálókat célszerű különböző törvénykezési hatáskörök alá eső területeken szétszórni. 10 véletlenszerűen kiválasztott kiszolgáló listáját például több helyen is lehetne biztonságos módon tárolni, így ha ezek közül valamelyik kompromittálódna, a többi még mindig megmaradna. A dokumentumot mindenáron megsemmisíteni próbáló hatóság így sosem lehetne biztos abban, hogy megtalálta az összes másolatot. A rendszer önjavító képességekkel is rendelkezhetne abban az értelemben, hogy ha kiderülne, hogy egyes másolatok megsemmisültek, akkor a fennmaradó helyek megpróbálhatnának új tárhelyeket találni az elveszettek pótlására.

Az örökkévalóság-szolgáltatás volt az első javaslat a cenzúrának is ellenállni képes rendszerekre. Azóta érkeztek más javaslatok is, melyeket néhány esetben meg is valósítottak. Ezekben számos új szolgáltatás is megtalálható, mint például a titkosítás, a névtelenség és a hibatűrőség. A tárolandó állományokat gyakran több részre darabolják fel, és az egyes részeket is több kiszolgálón tárolják. Ilyen rendszerek például a Freenet [Clarke és mások, 2002], a PASIS [Wylie és mások, 2000] és a Publius [Waldman és mások, 2000]. Ezektől eltérő munkákról is beszámol Serjantov [2002].

Sok országban egyre inkább próbálják szabályozni az immateriális javak kivitelét is. Ilyen javak lehetnek például a weboldalak, szoftverek, tudományos publikációk, e-levelek, telefonos ügyfélszolgálatok és mások. Az Egyesült Királyság több száz évre visszanyúló hagyományokkal rendelkezik a szólásszabadság terén, most mégis komolyan fontolgatja az erős korlátozások bevezetését. Ezek értelmében például a cambridge-i egyetemen egy brit professzor és a külföldi diákja között zajló szakmai megbeszélés a szabályozott export hatálya alá esne, és kormányengedélyhez lenne kötve [Anderson, 2002]. Az ilyen célkitűzések persze meglehetősen vitathatók. Szükségtelen említeni, hogy sok ember az ilyen politikát szélsőségesnek tekinti.

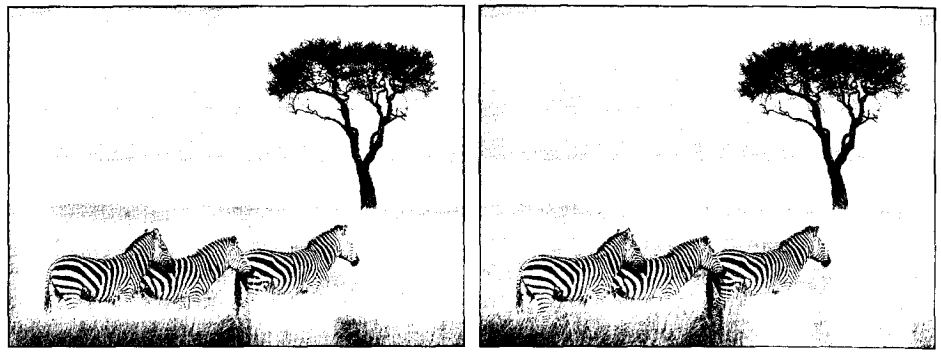
Szteganográfia

Azokban az országokban, ahol virágzik a cenzúra, az ellenzékiek gyakran a technikát hívják segítségül annak megkerülésére. A kriptográfia alkalmazásával lehet ugyan titkos üzeneteket küldeni (bár valószínűleg törvénytelenül), de ha a kormány úgy gondolja, hogy Aliz maga a Gonosz, akkor az a puszta tény, hogy ő Bobbal kommunikál, már Bobot is ugyanebbe a kategóriába juttathatja. Úgy tűnik, az elnyomó kormányok jól értik a tranzitivitás elvét még akkor is, ha híján vannak a matematikusoknak. Megoldást jelenthetnek az anonim postai közvetítők, de ha azokat az országon belül betiltják, a külföldre menő üzenetekhez pedig a kormány kiviteli engedélye szükséges, akkor ezek sem sokat segítenek. De ott van még a web!

Azok, akik titokban szeretnének kommunikálni, gyakran magát a tényt is megpróbálják elrejteni, hogy ott egyáltalán bármiféle kommunikáció folyik. Az üzenetek elrejtésének tudományát **szteganográfiának** (**steganography**) nevezzük, az „álcázott írás" görög szavak után. Az eljárást valójában már az ókori görögök is használták. Hérodotosz egy olyan tábornokról számol be, aki leborotváltatta a hírvivője fejét, rátetováltatta az üzenetét, majd megvárta, hogy a haj újra kinőjön, és csak azután küldte el a hírvivőt. A modern eljárások lényegében ugyanezt az elgondolást alkalmazzák, csak nagyobb a sávszélességük és kisebb a késleltetésük.

Egy hasonló példát láthatunk, ha megnézzük a 8.54.(a) ábrát. Ezt a fényképet a szerző készítette Kenyában. Három zebra látható rajta, amint épp egy akácfát bámulnak. Úgy tűnik, hogy a 8.54.(b) ábra ugyanezt a három zebrát és az akácfát ábrázolja, de ez a kép tud valami egész különlegeset is – magában hordozza öt Shakespeare-dráma (a Hamlet, a Lear király, a Macbeth, A velencei kalmár és a Julius Caesar) teljes, eredeti szövegét. Ezek a színművek több mint 700 KB-nyi szöveget jelentenek összesen.

Hogyan működik ez a szteganografikus eljárás? Az eredeti színes kép 1024 × 768 képpontból áll. Minden képpontot három darab 8 bites szám ír le: egy a vörös, egy a zöld,



(a) (b)

8.54. ábra. *(a) Három zebra és egy fa. (b) Három zebra, egy fa és William Shakespeare öt színművének teljes szövege*

egy pedig a kék szín intenzitását határozza meg. A képpont tényleges színét e három szín lineáris szuperpozíciója adja meg. A szteganografikus kódoló az egyes RGB-színek legalsó bitjét használja rejtett csatornaként. Ily módon minden képpont három bit titkos információt hordozhat: egyet a vörös, egyet a zöld és egyet a kék értékben. Egy ilyen méretű képben tehát akár 1024 × 768 × 3 bit, azaz 294 912 bájt titkos információ tárolható.

Az öt darab teljes szövege egy rövid megjegyzéssel együtt 734 891 bájtot tesz ki. Ezt először egy hétköznapi tömörítő algoritmus segítségével kb. 274 KB-os méretre tömörítettük, majd az eredményt IDEA-val kódoltuk, és beillesztettük az egyes színértékek legalsó bitjének helyére. Látható (pontosabban nem látható), hogy ennek az információnak a megléte teljesen észrevehetetlen. Az eredeti méretű, színes képeken ugyanúgy nem lehet észrevenni semmit, a szem ugyanis nemigen tud különbséget tenni a 21 bites és a 24 bites színek között.

Az itt látható alacsony felbontású, fekete-fehér képekből nem lehet igazán jól megítélni, hogy mennyire hatékony ez az eljárás. A szteganográfia működésének még jobb megértését segítendő, a szerző elkészített egy bemutatót, mely tartalmazza a 8.54.(b) ábrán látható képet nagy felbontásban, teljes színmélységgel, benne az öt színművel. A bemutató, valamint a képek és szövegek kombinálására szolgáló eszközök megtalálhatók a könyv webhelyén.

Az ellenzékiek tehát úgy használhatják fel a szteganográfiát az észrevétlen kommunikáció céljára, hogy készítenek például egy weboldalt tele politikailag korrekt képekkel. Az oldalon lehetnek fotók a Nagy Vezérről, helyi sporteseményekről, tévés személyiségekről, filmsztárokról stb. A képekben természetesen csak úgy hemzsegnének a szteganografikus üzenetek. Hiába gyanítaná valaki, hogy a képek üzeneteket rejtenek: ha ugyanis az üzeneteket előbb még tömörítik és titkosítják is, akkor utána már roppant nehéz megkülönböztetni azokat a fehér zajtól. Erre a célra természetesen csak frissen szkennelt képek jöhetnek szóba; ha csak úgy az internetről másolunk le egy képet, és annak változtatjuk meg a bitjeit, akkor máris marad egy veszélyes árulkodó jel utánunk.

Szteganografikus üzeneteket korántsem csak képek hordozhatnak. Éppen ilyen jól működik a hanghordozókba való elrejtés is. Rejtett információ szállítható egy IP-telefonos hívásban is a csomagkésleltetés változtatásával, a hang torzításával, vagy még a

csomag fejrészében is [Lubacz és mások, 2010]. Sőt még HTML-állományokban lévő címkék elrendezése és sorrendje is hordozhat információt.

Mi a szólásszabadság összefüggésében vizsgáltuk ugyan a szteganográfiát, de az eljárásnak még számos más felhasználási területe is van. Gyakori felhasználási mód például az is, amikor a képek tulajdonosai olyan titkos üzeneteket kódolnak képeikbe, melyek a tulajdonosi jogaikat nyilvánítják ki. Ha egy ilyen képet ellopnak és kiraknak egy webhelyre, a tulajdonos a bíróság előtt felfedheti a szteganografikus üzenetet, hogy bebizonyítsa: övé a kép. Ezt a módszert, melyet Piva és mások [2002] műve bővebben is tárgyal, **vízjelezésnek (watermarking)** nevezik.

A szteganográfiáról többet is megtudhatunk Wayner [2008] munkájából.

8.10.3. A szerzői jogok

A személyiségi jogok és a cenzúra mellett a szerzői jogok terén is találkozik a technika a közcélú szabályozással. A **szerzői jog (copyright)** lényege az, hogy a **szellemi tulajdon (Intellectual Property, IP)** alkotóit, köztük az írókat, festőket, zeneszerzőket, zenészeket, fotóművészeket, filmművészeket, koreográfusokat és másokat megilleti szellemi tulajdonuk kizárólagos felhasználási joga egy bizonyos ideig, ami rendszerint az alkotó élete plusz 50 vagy 75 év a közös tulajdon esetén. Ha egy mű szerzői jogvédelme lejárt, akkor átkerül köztulajdonba, és tetszése szerint bárki felhasználhatja vagy továbbadhatja. A Gutenberg-projekt (*www.promo.net/pg*) például több ezer köztulajdonban lévő művet rakott fel a webre (például Shakespeare-től, Twaintől, Dickenstől). 1998-ban az amerikai kongresszus az Amerikán belüli szerzői jogokat Hollywood kérésére újabb 20 évvel meghosszabbította, mert az ottaniak azt állították, hogy ha nem lesz hosszabbítás, akkor a későbbiekben senki nem fog alkotni semmit. Erre viszont azt hozhatnánk fel, hogy a szabadalmak is csak 20 évig tartanak, az emberek mégis találnak még fel dolgokat.

A szerzői jogok akkor kerültek igazán előtérbe, amikor a Napster nevű zenecserélgető szolgáltatás tagjainak száma elérte az 50 milliót. Bár maga a Napster nem másolt le semmilyen zenét, a bíróság mégis úgy vélekedett, hogy azzal, hogy egy központi adatbázisban tárolja azt, hogy kinek melyik szám van meg, valójában mások szabálysértéseit segíti elő. Azt persze senki nem állítja komolyan, hogy a szerzői jog rossz ötlet lenne (bár sokan gondolják úgy, hogy a védett időszak túl hosszú, és a nagyvállalatokat részesíti előnyben a közérdekkel szemben), a zeneelosztó-rendszerek új generációja azonban máris alapvető etikai kérdéseket vet fel.

Tekintsünk például egy olyan P2P-hálózatot, melyben az emberek legális állományokat (köztulajdonban lévő zenét, házi videókat, nem kereskedelmi titoknak minősülő vallási röpiratokat stb.) osztanak meg egymással, és esetleg néhány olyat is, melyet szerzői jog véd. Tegyük fel, hogy az ADSL vagy a kábeltévé révén mindenki állandóan online kapcsolatban van. Minden gépen van egy tartalomjegyzék arról, hogy mi van a merevlemezen, és van ott még egy lista is a többi tagról. Ha valaki egy konkrét dolgot keres, akkor megnézheti egy véletlenszerűen kiválasztott tagnál, hogy az rendelkezik-e az adott dologgal. Ha nem, akkor meg lehet nézni még az összes tagnál, aki az előző tag listáján szerepelt, aztán az azok listáján szereplő tagoknál, és így tovább. A számítógépek nagyon jók az ilyesmiben. Ha megvan a keresett elem, akkor egyszerűen le kell csak másolni.

Ha a szóban forgó művet szerzői jog védi, akkor az azt igénylő tag jó eséllyel törvénysértést követ el (bár a nemzetközi tranzakcióknál kérdéses, hogy melyik ország törvényei a mérvadók). De mi van azzal, akitől a lemásolt adatok származnak? Valóban bűnnek számít az, ha a kifizetett és legálisan letöltött zenét az ember olyan helyen tárolja a merevlemezén, ahol mások is hozzáférhetnek? Ha valakinek van egy faháza vidéken, amit nem zár be, és egy szellemitulajdon-tolvaj besurran oda egy hordozható számítógéppel és egy lapolvasóval, lemásol egy szerzői jogvédelem alá eső könyvet, majd kilopózik, akkor a háztulajdonos a bűnös azért, mert nem védte meg valaki más szerzői jogait?

A szerzői jogok egén azonban még tovább gyülekeznek a felhők. Jelenleg óriási csata folyik Hollywood és a számítógépipar között. Az előbbi szigorúbb védelmet követel mindenfajta szellemi tulajdonnak, az utóbbi pedig nem akar Hollywood rendőrévé válni. Az amerikai kongresszus 1998 októberében fogadta el a **DMCA**-t (**Digital Millennium Copyright Act – Digitális Millennium Szerzői Jogi Törvény**), mely bűncselekménynek minősíti a szerzői joggal védett művek védelmi mechanizmusainak kijátszását, illetve a másoknak való segítségnyújtást a védelem kijátszásában. Az Európai Unióban is hasonló szabályozást készülnek bevezetni. Azt gyakorlatilag senki nem gondolja, hogy a távol-keleti kalózoknak engedélyezni kellene a jogvédett alkotások másolását, de azért sokan vannak, akik úgy vélik, hogy a DMCA teljesen felborítja az egyensúlyt a jogtulajdonosok érdekei és a közérdek között.

Következzék egy idevágó példa! 2000 szeptemberében egy feltörhetetlen, online zeneárusító-rendszer kiépítésével megbízott zeneipari konzorcium versenyt hirdetett, melyben arra hívta fel a vállalkozó kedvűeket, hogy törjék fel a rendszert (és valóban pontosan ez a teendő minden új biztonsági rendszer bevezetésénél). A princetoni Edward Felten professzor által irányított, számos egyetem biztonsági szakértőit magában foglaló csapat felvette a kesztyűt, és feltörték a rendszert. Ezután írtak egy publikációt a tapasztalataikról, és beadták azt a USENIX biztonsági konferenciára, ahol a szakértők átnézték és elfogadták a művet. A publikáció bemutatása előtt azonban Felten levelet kapott az RIAA-tól (Recording Industry Association of America – Amerikai Lemezgyártók Szövetsége), azzal a fenyegetéssel, hogy pert indítanak a szerzők ellen a DMCA alapján, ha kiadják a publikációt.

Válaszul maguk a szerzők indítottak pert, melyben a szövetségi bíróságot kérték fel annak eldöntésére, hogy vajon a biztonsági kutatásokkal foglalkozó tudományos dolgozatok kiadása még mindig legálisnak minősül-e. Félvén attól, hogy a végső ítélet ellenük fog szólni, az ipar visszavonta a fenyegetését, a bíróság pedig ejtette Felten ügyét. Nem fér hozzá kétség, hogy az ipar visszavonulását a saját érveinek gyengesége ösztönözte: elvégre ők hívtak meg embereket arra, hogy feltörjék a rendszerüket, aztán meg perrel fenyegetőztek azért, mert egyesek elfogadták a kihívásukat. A visszavonulás után a publikációt végül is kiadták [Craver és mások, 2001]. Gyakorlatilag azonban biztosra vehető egy újabb összecsapás.

Mindezek közben a kalóz zenedarabok és filmek táplálták a P2P-hálózatok tömeges elterjedését. Ezt a szerzői jogok tulajdonosai nem fogadták örömmel, akik a DMCA közreműködését vették igénybe, hogy tegyen valamit ellenük. Nincsenek automatikus rendszerek, amelyekkel a P2P-hálózatok felkutathatók lennének, és ezután gyors figyelmeztetést lehetne küldeni a hálózati operátoroknak és azoknak a felhasználóknak, akiket a szerzői jogok megsértésével gyanúsítanak. Az USA-ban ezeket a figyelmeztetéseket úgy nevezik, hogy **DMCA eltávolítási felszólítás** weblap eltüntetésére (**DMCA**

takedown notices). Ez a kutatás tulajdonképpen egy fegyverkezési verseny, mert nehéz biztonsággal elkapni a szerzői jogok megsértőit. Még az is lehet, hogy a nyomtatónkat félreismertük, és az is egy bűnöző [Piatek és mások, 2008].

Ehhez a témához kapcsolódik **a tisztességes használat elve** (**fair use doctrine**) is, melyet különböző országokban székelő bíróságok mondtak ki. Az elv szerint a jogvédelem alá eső művek vásárlóit bizonyos korlátozott jogok illetik meg a mű másolását illetően, beleértve a mű tudományos célból való idézését, oktatási anyagként való felhasználását, és bizonyos esetekben a személyes használatra szánt biztonsági másolatok készítését is (arra az esetre, ha az eredeti hordozót valami baj érné). A tisztességes használat kritériumai között olyan kérdések vannak, hogy például (1) a felhasználás kereskedelmi célú-e, (2) az egész mű hány százalékát másolják le, és (3) milyen hatással van a másolás a mű eladására. Mivel azonban a DMCA és az Európai Unió hasonló törvényei tiltják a másolásvédelmi sémák megkerülését, ezek a törvények a tisztességes használatot is tiltják. A DMCA valójában a felhasználók történelmi jogait csorbítja azért, hogy a tartalomkínálóknak nagyobb hatalmat adjon. A végső leszámolás elkerülhetetlennek tűnik.

Egy újabb fejlemény, amely még a DMCA-t is felülmúlja a jogtulajdonosok és a felhasználók közötti egyensúly megbontásában, a **bizalmas számítógép-használat** (**trusted computing**), amelyet olyan neves ipari testületek támogatnak, mint a Microsoft és az Intel által vezetett **TCG** (**Trusted Computing Group – bizalmas számítógép-használat csoport**). Ennek ötlete az, hogy az operációs rendszer alatti szinten többféle módon gondosan szemmel tartaná a felhasználó tevékenységét (például hogy hallgat-e kalózfelvételeket) annak érdekében, hogy a nemkívánatos tevékenységeket letiltsa. Ezt egy kis chippel valósítják meg, amelyet **TPM**-nek (**Trusted Platform Module – bizalmas platform modul**) neveznek, amelybe nehéz belenyúlni. A legtöbb manapság eladott PC-t felszerelnek egy ilyen TMP-vel. A rendszer olyan szoftvert futtat, amelyet a tartalom tulajdonosa írt a PC manipulálására úgy, hogy a felhasználó ne tudja megváltoztatni. Ez felveti azt a kérdést, hogy ki bízzon a bizalmas számítógép-használatban. Minden bizonnyal ez nem a felhasználó. Mondani sem kell, hogy egy ilyen séma beláthatatlan társadalmi következményekkel jár. Szép dolog az, hogy az iparág végre figyelmet szentel a biztonságnak, de elég siralmas, hogy ez a figyelem kizárólag a szerzői jogok védelmére irányul ahelyett, hogy foglalkozna a vírusokkal, crackerekkel, behatolókkal és egyéb olyan biztonsági kérdésekkel, melyek az embereket aggasztják.

Egy szóval, a törvényhozók és az ügyvédek még évekig el lesznek foglalva azzal, hogy egyensúlyba hozzák a jogtulajdonosok gazdasági érdekeit és a közérdeket. A kibertér sem különbözik a valós tértől: egy-egy csoport itt is mindig a másik ellen fordul, ami aztán hatalmi harcokat, pereskedést és végül (remélhetőleg) valamilyen megoldást eredményez, legalábbis addig, amíg valami újabb bomlasztó technika meg nem jelenik.

8.11. Összefoglalás

A kriptográfia olyan eszköz, melyet az információ titokban tartására, valamint sértetlenségének és hitelességének megőrzésére használhatunk. Minden modern kriptográfiai rendszer Kerckhoff elvén alapszik, mely egy közismert algoritmusból és egy titkos kulcsból indul ki. Sok kriptográfiai algoritmus használ helyettesítésekből és keverésekből álló bonyolult átalakításokat, hogy a nyílt szövegből kódolt szöveget állítson elő. Ha viszont a kvantumkriptrográfia a gyakorlatban is jól alkalmazható lesz, akkor az egyszer használatos bitminták használatával valóban feltörhetetlen titkosító rendszereket kaphatunk.

A kriptográfiai algoritmusokat szimmetrikus kulcsú algoritmusokra és nyilvános kulcsú algoritmusokra oszthatjuk. Az előbbiek úgy készítik el a kódolt szöveget, hogy a kulcs függvényében több körön át szórják szét mindenfelé a biteket. A jelenlegi legnépszerűbb szimmetrikus kulcsú algoritmusok az AES (Rijndael) és a háromszoros DES. Ezeket lehet használni elektronikus kódkönyv módban, titkosított blokkok láncolásával, folyamkódoló módban, számláló módban és még egyéb módokban is.

A nyilvános kulcsú algoritmusokat az jellemzi, hogy más kulcsot használnak a kódoláshoz és a dekódoláshoz, valamint hogy a dekódoló kulcsot nem lehet megkapni a kódoló kulcsból. Ezen tulajdonságoknak köszönhetően a nyilvános kulcsot szabadon lehet terjeszteni. A legfontosabb nyilvános kulcsú algoritmus az RSA, ami abból a tényből meríti erejét, hogy nagyon nehéz feladat nagy számokat szorzatra bontani.

A jogi, kereskedelmi és egyéb dokumentumokat alá kell írni. Ennek megfelelően különféle sémákat dolgoztak már ki a digitális aláírásokhoz, mind szimmetrikus, mind nyilvános kulcsú algoritmusok felhasználásával. Az aláírandó üzenetekből rendszerint egy hash-t képeznek – például az SHA-1 algoritmus segítségével –, és nem az eredeti üzenetet, hanem a hash-t írják alá.

A nyilvános kulcsokat tanúsítványok segítségével lehet kezelni. Ezek olyan dokumentumok, amelyek egy szereplőt egy nyilvános kulcshoz kötnek. A tanúsítványokat vagy egy megbízható hatóság, vagy pedig olyasvalaki írja alá, akinek az aláírását egy megbízható hatóság (esetleg rekurzív módon) jóváhagyta. A bizalmi lánc gyökerét előre ismerni kell, de a böngészőkbe általában több gyökér tanúsítványa is előre be van építve.

Ezek a kriptográfiai eszközök a hálózati forgalom biztonsága érdekében is hasznosíthatók. Az IPsec a hálózati rétegben működik, és két hoszt között titkosítja a csomagfolyamot. A tűzfalak a szervezetekhez belépő, illetve az onnan kilépő forgalmat figyelik, többnyire a felhasznált protokoll és port alapján. A virtuális magánhálózatok a régi bérelt vonalas hálózatokat szimulálják, hogy bizonyos kívánatos biztonsági tulajdonságokat nyújtsanak. Végül, a vezeték nélküli hálózatokban komoly biztonsági megoldásokra van szükség, nehogy mindenki elolvassa az összes üzenetet, továbbá az ilyen szolgáltatást nyújtó protokollokra, mint amilyen a 802.11i.

Amikor két fél kiépít egy viszonyt, hitelesíteniük kell egymást, és ha szükséges, meg kell állapodniuk egy közös viszonykulcsról. Számos hitelesítési protokoll létezik: van, amelyik egy megbízható harmadik félhez fordul, de van, amelyik például a Diffie–Hellman-protokollt, a Kerberost vagy a nyilvános kulcsú kriptográfiát használja.

Az e-levelek biztonságát az ebben a fejezetben látott módszerek kombinálásával lehet elérni. A PGP például tömöríti az üzeneteket, aztán titkosítja azokat egy titkos kulccsal és elküldi a titkos kulcsot a vevő nyilvános kulcsával. Ezenfelül hash-t is képez az üzenetből, és az aláírt hash-t is elküldi, hogy az üzenet sértetlensége ellenőrizhető legyen.

A web biztonsága szintén fontos téma, mely a biztonságos névkezeléssel kezdődik. A DNSsec használatával megelőzhető a DNS-megtévesztéses támadás. A legtöbb e-kereskedelmi weboldal SSL/TLS-t használ az ügyfél és a kiszolgáló közötti biztonságos, hitelesített kapcsolat kiépítésére. Számos eljárás létezik a hordozható kódok kezelésére

is, különösen fontos ezek közül az ún. homokozók alkalmazása, illetve a kódok aláírása.

Az internet sok olyan kérdést vet fel, melyben a technika szoros kölcsönhatásban van a közcélú szabályozással. Ilyen kérdések merülnek fel többek között a személyiségi jogok, a szólásszabadság és a szerzői jogok terén.

8.12. Feladatok

1. Fejtse meg a következő egyábécés helyettesítéses titkosítást. A nyílt szöveg, amely csak betűkből áll, Lewis Carroll egy jól ismert költeményének részlete.

   mvyy bek mnyx n yvjjyr snijrh invq n muvjvdt je n idnvy
   jurhri n fehfevir pyeir oruvdq ki ndq uri jhrnqvdt ed zb jnvy
   Irr uem rntrhyb jur yeoijrhi ndq jur jkhjyri nyy nqlndpr
   Jurb nhr mnvjvdt ed jur iuvdtyr mvyy bek pezr ndq wevd jur qndpr
   mvyy bek, medj bek, mvyy bek, medj bek, mvyy bek wevd jur qndpr
   mvyy bek, medj bek,mvyy bek,medj bek, medj bek wevd jur qndrpr

2. Az affin titkosítás az egyábécés helyettesítéses titkosítás egyik változata, amelyben egy m méretű ábécé betűi először leképeződnek a 0-tól m-1-ig terjedő egészekre. Következésképp, minden egyes nyílt szövegű betűt reprezentáló egész transzformálódik a hozzá tartozó titkos szövegű betűt reprezentáló egészre. A titkosítási függvény egy betűre $E(x)=(ax+b) \bmod m$, ahol m az ábécé mérete, és a és b titkosító kulcsok és társprímek. Trudy kitalálta, hogy Bob affin titkosítással egy titkos szöveget állított elő. Szerzett a tikos szövegről egy másolatot, és kitalálta, hogy a titkos szövegben a leggyakrabban előforduló betű az „R", a második leggyakoribb betű pedig a „K". Mutassa meg, Trudy hogyan tudja feltörni a kódot, és visszaállítani a nyílt szöveget.

3. Fejtse meg a következő oszlopos keverőkódos rejtjelezést. A nyílt szöveget egy népszerű számítógépes tankönyvből vettük, így a „computer" valószínűleg előfordul benne. A nyílt szöveg kizárólag betűkből áll (szóközök nélkül). A rejtett szöveget az olvashatóság érdekében ötkarakteres blokkokra tördeltük.

   aauan cvlre turnn dltme aeepb ytust iceat npmey iicgo gorch srsoc
   nntii imiha oofpa gsivt tpsit lbolr otoex

4. Aliz keverőkódos titkosítást használt Bobnak írandó üzenetei titkosítására. A biztonság növelésére a keverőkódolóhoz készült titkosító kulcsot helyettesítő kódolással titkosította, és a titkosított titkosító kulcsot a számítógépén tartotta. Trudy módot talált arra, hogy hozzájusson a titkosított keverőkódos titkosító kulcshoz. Vajon Trudy képes megfejteni Aliz Bobnak írt üzeneteit? Miért igen vagy miért nem?

5. Keressen egy olyan 77 bites egyszer használatos bitmintát, mely a 8.4. ábra kódolt szövegéből a „Hello World" szöveget állítja elő!

6. Tegyük fel, hogy Ön egy kém és kényelemből segítségként egy végtelen számú könyvet tartalmazó könyvtárral rendelkezik. Az Ön operátorának ugyancsak van egy ilyen könyvtára az ő segítésére. Ön megengedte a *Gyűrűk ura* használatát egyszer használatos bitmintaként. Magyarázza meg, hogyan tudná használni ezeket a vagyontárgyakat egy végtelen hosszú egyszer használatos bitminta előállításához.

7. A kvantumkriptográfiához egy olyan fotonágyú szükséges, mely igény szerint képes egyetlen darab, egy bitet hordozó fotont kilőni. Számítsa ki azt, hogy hány fotonból áll egy bit egy 250 Gb/s-os fényvezetőszálas összeköttetésen! Felteheti, hogy a foton hossza megegyezik a hullámhosszával, ami ebben a példában legyen 1 mikron. A fény sebessége az üvegszálban 20 cm/ns.

8. Ha kvantumkriptográfiát alkalmazunk, és Trudy elfogja, majd újragenerálja a fotonokat, akkor néhányat biztosan elront majd, és emiatt hibák jelennek meg Bob egyszer használatos bitmintájában. Átlagosan hányadrésze lesz hibás a Bob bitmintájában lévő biteknek?

9. Egy alapvető kriptográfiai alapelv szerint minden üzenetnek tartalmaznia kell valamilyen redundanciát. Viszont azt is tudjuk, hogy a redundancia segít a támadónak megállapítani azt, hogy a tippelt kulcs helyes-e. Vegyük most a redundancia két különböző formáját! Az egyikben a nyílt szöveg első n bitje egy ismert mintát tartalmaz. A másikban az üzenet utolsó n bitje tartalmazza az üzenet hash-ét. A biztonság szempontjából vajon egyenértékű ez a két megoldás? Fejtse ki válaszát!

10. A 8.6. ábrán a P-dobozok váltakoznak az S-dobozokkal. Bár ez az elrendezés esztétikailag kellemes, biztonságosabb-e, mint először venni az összes P-dobozt, majd az összes S-dobozt?

11. Tervezzen meg egy DES-támadást arra alapozva, hogy a nyílt szöveg csak ASCII nagybetűkből és szóközből, vesszőből, pontból, pontosvesszőből, kocsi-vissza jelből és soremelés jelből áll. Semmit nem tudunk a nyílt szöveg paritásbitjeiről.

12. A szövegben kiszámítottuk, hogy egy olyan, egymillió processzort alkalmazó kódtörő géppel, mely nanomásodpercenként egy kulcsot elemezhetne, az AES 128 bites verziójának feltörése 10^{16} évet venne igénybe. Számítsuk ki, hogy mennyi ideig fog tartani egy kulcs elemzése, ha a kódtöréshez szükséges idő egy évre csökken, bár ez természetesen még mindig elég hosszú idő. A cél eléréséhez olyan számítógépekre van szükségünk, amelyek 10^{16}-szor gyorsabbak. Ha a Moore-törvény (a számítási teljesítmény 18 hónaponként megkétszereződik) továbbra is érvényes, hány évre lesz szükség a kódtöréshez, mielőtt egy párhuzamos számítógép lecsökkenthetné a kódtörés idejét egy évre?

13. Az AES támogatja a 256 bites kulcsokat. Hányféle kulcs lehet egy AES-256 rendszerben? Nézzen utána, hogy található-e a fizikában, a kémiában vagy a csillagászatban hasonló nagyságrendű szám! Az internetet is használja fel a nagy számok keresésében! Vonjon le következtetéseket a kutatásából!

14. Tegyük fel, hogy egy üzenetet a DES rejtett szövegű blokkokat egymás után fűző módjával titkosítottunk. A rejtett szöveg C_i blokkban egy bit véletlenül 0-ról 1-re változik az átvitel során. Mennyi nyílt szöveg fog összezavarodni ennek eredményeképpen?

15. Vegyük megint a rejtett szövegű blokkokat egymás után fűző módot. Egy 0 bit 1-re változtatása helyett egy külön 0-t szúrunk be a titkos szöveg folyamába a C_i blokk után. Mennyi nyílt szöveg fog emiatt összezavarodni?

16. Hasonlítsa össze a rejtett szövegű blokkokat egymás után fűző módot a rejtett szöveget visszacsatoló móddal egy nagy állomány átviteléhez szükséges titkosító műveletek szempontjából. Melyik a hatékonyabb és mennyivel?

17. Az RSA nyilvános kulcsú titkosító rendszert használva $a = 1$, $b = 2$, ... $y = 25$, $z = 26$.
    (a) Ha $p = 5$ és $q = 13$, akkor sorolja fel d öt érvényes értékét.
    (b) Ha $p = 5$, $q = 31$ és $d = 37$, találja meg e-t,
    (c) $p = 3$, $q = 11$ és $d = 9$ használatával keresse meg az e-t és titkosítsa a „hello" szöveget.

18. Aliz és Bob RSA nyilvános kulcsú titkosítást használ annak érdekében, hogy egymással kommunikálni tudjanak. Trudy kitalálja, hogy Aliz és Bob a prímszámok közül egyet megosztva arra használ, hogy az ő nyilvános kulcspárjuk n számát meghatározza. Más szóval, Trudy kitalálta, hogy $n_a = p_a \times q$ és $n_b = p_b \times q$. Trudy hogyan tudja felhasználni ezt az információt Aliz kódjának feltörésére?

19. Vegyük a 8.15. ábrán bemutatott számláló mód használatát, de legyen $IV = 0$. Veszélyezteti-e a 0 érték használata a kód biztonságát általános értelemben?

20. A 8.20. ábrán láthattuk, hogyan küldhet Aliz Bobnak egy aláírt üzenetet. Ha Trudy kicseréli a P-t, azt Bob észreveszi. De mi történik akkor, ha Trudy a P-t és az aláírást is kicseréli?

21. A digitális aláírásoknak van egy potenciális gyenge pontja a lusta felhasználóknak köszönhetően. Vegyük azt az esetet, amikor egy e-kereskedelmi tranzakcióban megkötnek egy szerződést, és a felhasználót arra kérik, hogy írja alá a szerződés SHA-1 hash-ét. Ha a felhasználó nem ellenőrzi tételesen, hogy a szerződés és a hash megfelel egymásnak, akkor figyelmetlenségből egy másik szerződést is aláírhat. Tegyük fel, hogy a maffia megpróbálja kihasználni ezt a rést, hogy némi pénzhez jusson! Felállítanak tehát egy fizetős weboldalt (például pornográf tartalommal, szerencsejátékokkal stb.), és az új ügyfelektől elkérik a hitelkártyájuk számát. Ezután átküldenek egy szerződést, mely szerint a felhasználó igénybe kívánja venni a szolgáltatásukat, és hitelkártyával fog fizetni, majd megkérik a felhasználót, hogy írja alá a szerződést. Teszik mindezt annak tudatában, hogy a legtöbb felhasználó egyszerűen csak aláírja a szerződést anélkül, hogy meggyőződne arról, hogy a szerződés és a hash rendben van-e. Mutassa meg, hogyan tud a maffia gyémántokat vásárolni egy törvényes internetes gyémántkereskedőtől úgy, hogy a vételárat a gyanútlan felhasználók nyakába varrja!

22. Egy matematika előadáson 25 hallgató van. Feltéve, hogy az összes hallgató az év első felében született – január 1. és június 30 között –, mi a valószínűsége annak, hogy legalább két hallgató ugyanazon a napon született? Tegyük fel, hogy senki sem született szökőnapon, vagyis 181 lehetséges születésnap van.

23. Miután Ellen bevallotta Marilynnek, hogy megtréfálta őt Tom birtoklásának kérdésében, Marilyn ezt a problémát azzal oldotta meg, hogy a további üzenetek tartalmát egy diktafonnak mondja el, és az új titkárnő csak begépeli azokat. Ezután Marilyn úgy tervezte, hogy megvizsgálja a határidőnaplójában levő üzeneteket, miután begépelték azokat, hogy leellenőrizze, hogy pontosan az ő szavait tartalmazzák-e. Használhatja-e az új titkárnő a születésnap-támadást üzenetek hamisításához, és ha igen, hogyan? *Tipp:* igen, használhatja.

24. Tekintsük a 8.23. ábrát, ahol Aliznak nem sikerült megszereznie Bob nyilvános kulcsát! Tegyük fel, hogy Bob és Aliz már megállapodtak egy közös titkos kulcsról, de Aliz még mindig szeretné megkapni Bob nyilvános kulcsát. Van-e ezek után arra mód, hogy biztonságban hozzájuthasson? Ha igen, hogyan?

25. Aliz nyilvános kulcsú kriptográfia útján szeretne kommunikálni Bobbal. Kiépít tehát egy összeköttetést valakivel, aki reményei szerint maga Bob. Elkéri partnerétől a nyilvános kulcsát, amit az kódolatlan formában küld el neki egy X.509 tanúsítvánnyal együtt, melyet a gyökér CA írt alá. Aliznak megvan már a gyökér CA nyilvános kulcsa. Milyen lépéseket hajt végre Aliz annak ellenőrzésére, hogy valóban Bobbal beszél-e? Felteheti, hogy Bobnak nem számít, kivel beszél (például Bob egyfajta nyilvános szolgáltatás).

26. Tegyük fel, hogy egy rendszer fastruktúrába szervezett CA-kon alapuló PKI-t használ! Aliz Bobbal szeretne kommunikálni, ezért kiépít felé egy kommunikációs csatornát, majd kap tőle egy tanúsítványt, amit az X CA írt alá. Aliz soha nem hallott még az X-ről. Milyen lépéseket kell ekkor megtennie annak ellenőrzésére, hogy valóban Bobbal beszél?

27. Lehet-e az AH-t használó IPsec-et szállítási módban használni akkor, ha az egyik gép egy NAT doboz mögött helyezkedik el? Indokolja válaszát!

28. Aliz üzenetet kíván küldeni Bobnak SHA-1 hash alkalmazásával. Konzultál Önnel a megfelelő használandó aláíró algoritmussal kapcsolatban. Mi az Ön javaslata?

29. Mondjon egy okot arra, hogy miért célszerű egy tűzfalat a bejövő forgalom megvizsgálására alkalmazni! Mondjon egy okot arra is, hogy miért célszerű egy tűzfalat a kimenő forgalom megvizsgálására alkalmazni! Mit gondol, várhatóan eredményesek lesznek ezek a vizsgálatok?

30. Tételezzük fel, hogy egy cég VPN-t használ telephelyeinek inteneten keresztül történő összekötésére. Jim, a cég egyik alkalmazottja a VPN-t használja főnökével, Maryvel való kommunkációjához. Írjon le egy olyan fajta kommunikációt kettejük között, amelyhez nem szükséges titkosítás vagy más biztonsági mechanizmus, továbbá egy olyan típusú kommunikációt, amelyhez titkosítás vagy másfajta biztonsági mechanizmus szükséges. Magyarázza meg válaszát.

31. A 8.34. ábrán látható protokollban változtasson meg kismértékben egy üzenetet, hogy ellenállóvá tegye azt a válaszolgató támadással szemben. Indokolja meg, miért működik a változtatás!

32. A Diffie–Hellman-kulcscserélést használjuk Aliz és Bob közt egy titkos kulcs létrehozásához. Aliz a következőt küldi Bobnak: (227, 5, 82). Bob a következővel válaszol: (125). Aliz x titkos száma 12. Mi lesz a titkos kulcs? Bob titkos száma $y = 3$. Mutassa meg, Hogy Aliz és Bob hogyan számítja ki a titkos kulcsot.

33. Két felhasználó Diffie–Hellman-kulcscserélést használ, jóllehet még sohasem találkoztak, titkot sem cseréltek, és tanúsítványuk sincs.
    (a) Magyarázza meg, hogy ez az algoritmus mennyire érzékeny a közbeékelődéses támadásra.
    (b) Ez az érzékenység hogyan változna meg, ha n és g titok lenne?

34. A 8.39. ábra protokolljában miért küldik el A-t nyílt szövegben a titkosított viszonykulccsal együtt?

35. A Needham–Schroeder-protokollban Aliz két kihívást hoz létre, R_A-t és R_{A2}-t. Ez túlzásnak tűnik. Nem lenne elég egy?

36. Tegyük fel, hogy egy szervezetnél Kerberost használnak a hitelesítéshez. Milyen hatással van egy AS vagy egy TGS meghibásodása a biztonságra és a szolgáltatások elérhetőségére?

37. Aliz a 8.43. ábrán látható nyilvános kulcsú hitelesítési algoritmust használja Bobbal történő kommunikációjának hitelesítésére. Amikor azonban elküldi a 7. üzenetet, elfelejti titkosítani R_B-t. Trudy ekkor megtudja R_B értékét. Aliznak és Bobnak meg kell ismételni a hitelesítési eljárást új paraméterekkel annak érdekében, hogy biztosítsák a titkos kommunkációt? Magyarázza meg válaszát.

38. A 8.43. ábra hitelesítési protokolljában a 7. üzenetben az R_B-t a K_S-sel titkosítottuk. Szükséges ez a titkosítás, vagy elegendő lett volna azt nyílt szövegben visszaküldeni? Magyarázza meg válaszát.

39. A vásárlás helyén levő terminálóknak, amelyek mágnescsíkos kártyákat és PIN-kódokat használnak, van egy végzetes hibájuk: egy rosszindulatú kereskedő módosíthatja a kártyaolvasóját, hogy a kártyán levő összes információt és a PIN-kódot is elfogja és tárolja, hogy a jövőben további (hamis) tranzakciókat postázhasson. A vásárlás helyén levő terminálok következő generációja olyan kártyákat fog használni, amelyeken teljes CPU, billentyűzet és egy kis képernyő is van. Gondoljon ki ehhez a rendszerhez egy olyan protokollt, amelyet a rosszindulatú kereskedők nem törhetnek fel.

40. Vajon többesküldéssel elküldhető egy PGP-üzenet? Milyen korlátozásokat kell alkalmazni?

41. Tegyük fel, hogy az interneten mindenki használ PGP-t. Lehetséges-e ekkor egy tetszőleges internetcímre egy PGP-üzenetet küldeni úgy, hogy azt minden érintett helyesen dekódolni tudja? Fejtse ki válaszát!

42. A 8.47. ábrán látható támadás egy lépésről megfeledkezik. Erre a lépésre nincs szükség az eredményes támadáshoz, de ha megteszik, akkor csökkenhet az utólagos lebukás veszélye. Mi ez a hiányzó lépés?

43. Az SSL adatátviteli protokoll két véletlen számot és egy előzetes főkulcsot használ. Van-e itt egyáltalán szerepe a véletlen számok alkalmazásának, és ha igen, mi az?

44. Tekintsünk egy képet, amely 2048×512 képpontból áll. Titkosítani akarunk egy 2,5 MB méretű állományt. Az állománynak mekkora darabját tudja titkosítani ebben a képben? Mekkora darabját tudná titkosítani ennek az állománynak, ha az eredeti méret negyedére tömörítené? Mutassa be a számításait.

45. A 8.55.(b) ábra öt Shakespeare-mű szövegét tartalmazza ASCII-formátumban. Szöveg helyett vajon zenét is el lehetne rejteni a zebrák közé? Ha igen, hogyan működne ez a megoldás, és mennyi zenét lehetne elrejteni a képben? Ha nem, miért nem?

46. Ön kapott egy 60 MB-os szövegfájlt, amelyik titkosítandó szteganográfia alkalmazásával úgy, hogy a képfájlban levő színek kis helyi értékű bitjeit használjuk erre. Mekkora méretű képre lenne szükség ennek az állománynak a titkosításához? Mekkora méretű képre lenne szükség, ha előbb a fájlt harmadára tömörítenénk? A válaszát képpontban adja meg, és mutassa meg a számításait. Tételezzük fel, hogy a kép képaránya 3:2, például 3000×2000 képpont.

47. Aliz gyakran használta az 1-es típusú anonim postai közvetítőt. Sok üzenetet küldött a kedvenc hírcsoportjába, az *alt.fanclub.alice* csoportba, és mindenki tudta, hogy azok tőle jönnek, mivel mind ugyanazt az álnevet hordozta. Trudy nem tudta megszemélyesíteni Alizt, feltéve, hogy a közvetítő helyesen működött. Miután az 1-es típusú közvetítőket mindenhol megszüntették, Aliz áttért egy *cypherpunk* közvetítőre, és új társalgást indított a hírcsoportjában. Javasoljon Aliznak egy módszert annak megakadályozására, hogy Trudy az ő nevében küldjön új üzeneteket a hírcsoportba!

48. Keressen az interneten egy érdekes ügyet a személyiségi jogok témakörében, és írjon róla egyoldalas beszámolót!

49. Gyűjtsön információt az internet segítségével egy olyan bírósági ügyről, ahol a tisztességes használat elve és a szerzői jogok kerültek szembe egymással! Írjon egyoldalas beszámolót a kutatása eredményéről!

50. Írjon egy olyan programot, mely úgy kódolja a bemenetét, hogy KIZÁRÓ VAGY (XOR) kapcsolatba hozza azt egy kulcsfolyammal. Keresse meg vagy írja meg a lehető legjobb véletlenszám-generátort a kulcsfolyam előállítására! A program működjön szűrőként: vegye a nyílt szöveget a szabványos bemenetén és adja ki a kódolt szöveget a szabványos kimenetén (és fordítva). A program egy paramétert használjon: legyen ez a kulcs, mely a véletlenszám-generátort inicializálja.

51. Írjon olyan eljárást, mely egy adatblokk SHA-1 hash-értékét számítja ki! Az eljárásnak két paramétere lehet: egy mutató a bemeneti pufferre és egy mutató a 20 bájtos kimeneti pufferre. Az SHA-1 pontos specifikációját megtalálhatja, ha az interneten rákeres a FIPS 180-1 kifejezésre, ami a teljes specifikációt tartalmazó dokumentum.

52. Írjon egy olyan függvényt, amely elfogad egy ASCII-karakterekből álló folyamot és titkosítja ezt a bemenetet egy helyettesítő kódolóval, amelyik titkosított blokkok láncolása módban üzemel. A blokk mérete legyen 8 bájt. A programnak be kell venni a nyílt szöveget a szabványos bemenetéről és ki kell nyomtatnia a titkosított szöveget a szabványos kimenetén. Ennek a problémának a megoldásához bármilyen elfogadható rendszert választhat annak meghatározására, hogy a bemeneti sorozat végét megtalálja, és/vagy amikor töltelékezést kell alkalmazni a blokk teljessé tételére. Bármilyen kimeneti formátumot választhat mindadddig, amíg az egyértelmű. A programnak két paraméter kell fogadnia:

    1. Egy pointert a vektor inicializálására, és

    2. Egy k számot, amely a helyettesítéses titkosítás eltolását mutatja, úgy hogy minden ASCII-karaktert úgy kódolunk, hogy helyette az ábécében k-val előtte lévő karaktert használjuk.

    Például, ha $x = 3$, akkor A kódolva D lesz, B kódolva E, és így tovább. Tegyen elfogadható feltételezéseket az ASCII-karakterkészlet utolsó karakterének megtalálását illetően. Legyen biztos abban, hogy világosan dokumentálásra kerül a kódban bármilyen feltételezés, amelyet Ön a bemenettel és a titkosító algoritmussal kapcsolatban tesz.

53. Ennek a programnak az a célja, hogy az RSA mechanizmusának egy jobb megértését adja. Írjon egy függvényt, amelyik paraméterként elfogad p és q prímszámokat, kiszámítja a nyilvános és egyéni RSA-kulcsokat e paraméterek felhasználásával, és kiadja az n, z, d és e értékeket mint kimenetet a szabványos kimeneten. A függvénynek ugyancsak el kell fogadnia egy ASCII-karakterekből álló sorozatot és titkosítania kell ezt a bemenetet a kiszámított RSA-kulcsok felhasználásával. A programnak be kell venni a nyílt szöveget a szabványos bemenetéről és ki kell nyomtatnia a titkosított szöveget a szabványos kimenetén. A titkosításnak karakteralapúnak kell lenni, azaz venni kell minden karaktert a bemeneten és titkosítani kell függetlenül a beneteten lévő többi karaktertől. Ennek a problémának a megoldásához bármilyen elfogadható rendszert választhat annak meghatározására, hogy a bemeneti sorozat végét megtalálja. Bármilyen kimeneti formátumot választhat mindaddig, amíg az egyértelmű. Legyen biztos abban, hogy világosan dokumentálásra kerül a kódban bármilyen feltételezés, amelyet Ön a bemenettel és a titkosító algoritmussal kapcsolatban tesz.

9. Ajánlott olvasmányok és irodalomjegyzék

Befejeztük a számítógép-hálózatok tanulmányozását, de ez csak a kezdet. Sok érdekes témával nem tudtunk az azokat megillető részletességgel foglalkozni, míg másokat teljes egészében kihagytunk helyhiány miatt. Ebben a fejezetben a további olvasáshoz ajánlott műveket és az irodalomjegyzéket tesszük közzé, azon olvasók kedvéért, akik folytatni szeretnék a számítógép-hálózatok tanulmányozását.

9.1. Javaslatok a továbbolvasáshoz

A számítógép-hálózatok minden területéhez bőséges irodalom áll rendelkezésre. Két folyóirat is van, mely rendszeresen közöl cikkeket ebben a témában: az *IEEE/ACM Transactions on Networking*, és az *IEEE Journal on Selected Areas in Communications*.

Az ACM Special Interest Groups on Data Communications (SIGCOMM) és a Mobility of Systems, Users, Data, and Computing (SIGMOBILE) kiadványai sok érdekes dolgozatot közölnek, különös tekintettel a sürgető témákra. Ezek a *Computer Communication Review* és a *Mobile Computing and Communications Review*.

Az IEEE három olyan magazint is kiad, melyek felméréseket, oktatási anyagokat és esettanulmányokat tartalmaznak a hálózatok témaköréből: ezek az *IEEE Internet Computing*, az *IEEE Network Magazine* és az *IEEE Communications Magazine* címet viselik. Az első kettő az architektúra, a szabványok és a szoftverek területére összpontosít, míg a harmadik inkább a kommunikációs technikák felé hajlik (üvegszálak, műholdak stb.).

Ezeken kívül sok, évente vagy kétévente megrendezett konferencia van, melyre számos cikk érkezik a hálózatok témaköréből. Ilyenek például a *SIGCOMM* konferencia, az *NSDI* (Symposium on Networked Systems Design and Implementation), a *MobiSys* (Conference on Mobile Systems, Applications, and Services) és az *OSDI* (Symposium on Operating Systems Design and Implementation).

A következőkben kiegészítő olvasmányokra teszünk néhány javaslatot, a könyv fejezetei szerinti bontásban. Az ajánlatok többsége könyvekből vett fejezetek néhány oktatóanyaggal és felméréssel. A teljes hivatkozási lista a 9.2. szakaszban található.

9.1.1. Bevezetés és általános művek

Comer: *The Internet Book*, 4. kiadás

Bárkinek, aki könnyen érthető bevezetőt keres az internethez, érdemes megnéznie. Comer leírja az internet történetét, növekedését, technikáját, protokolljait és szolgáltatásait úgy, hogy a kezdők is megérthetik, de a könyv olyan sok anyagot tartalmaz, hogy a szakmabeli olvasók érdeklődésére is számot tarthat.

Computer Communication Review, 25th Anniversary Issue, 1995. január

Ez a speciális kiadás összegyűjti az 1995-ig megjelent legfontosabb cikkeket, amelyek az internet kifejlesztésével kapcsolatosak. Olyan cikkeket tartalmaz, mint amilyen a TCP kifejlesztésének bemutatása, a többesküldés, a DNS, az Ethernet és a teljes architektúra.

Crovella és Krishnamurthy: *Internet Measurement*

Hogyan tudjuk meg, hogy mennyire jól működik az internet? Erre a kérdésre a válasz nem triviális, mert senki nem az internet felelőse. Ez a könyv leírja azt a technikát, amelyet az internet működésének mérésére fejlesztettek ki, a hálózati infrastruktúrától kezdve az alkalmazásokig.

IEEE Internet Computing, 2000. január–február

Az *IEEE Internet Computing* új évezredben megjelent első száma pontosan azt hozza, amit várunk tőle: megkéri azokat az embereket, akik az előző évezredben létrehozták az internetet, hogy próbálják meg kitalálni, vajon hová jut az a következő évezredben. A szakértők között olyan nevek szerepelnek, mint Paul Baran, Lawrence Roberts, Leonard Kleinrock, Stephen Crocker, Danny Cohen, Bob Metcalfe, Bill Gates, Bill Joy és mások. A legjobb lenne talán 500 évet várni, és csak *azután* elolvasni a jóslatokat.

Kipnis: *Beating the System: Abuses of the Standards Adoption Process*

A szabványosítási bizottságok próbálnak tisztességesek és a gyártóktól függetlenek maradni munkájuk során, de sajnos akadnak olyan vállalatok, melyek megpróbálnak visszaélni ezzel a rendszerrel. Többször is előfordult már például az, hogy egy vállalat segített kidolgozni valamilyen szabványt, majd annak elfogadása után bejelentette, hogy a szabvány valójában a vállalat egyik saját szabadalmán alapul. A vállalat ezek után más cégek számára tetszése szerint engedélyezhette vagy megtagadhatta a szabvány használatát attól függően, hogy kedvelte-e az adott céget vagy sem; mindezt pedig a saját maga által megszabott árakon tehette. Ez a cikk kiváló kezdetet jelent a szabványosítás sötét oldalának megismeréséhez.

Hafner és Lyon: *Where Wizzards Stay Up Late*
Naughton: *A Brief History of the Future*

Ki fedezte fel az internetet? Sok ember igényelte már ennek elismerését. És joggal, mivel különféle úton-módon sok embernek van benne a munkája. Paul Baran volt az, aki egy jelentést írt a csomagkapcsolásról, voltak emberek különböző egyetemeken, akik megtervezték az ARPANET architektúráját, voltak emberek a BBN-nél, akik az

első IMP-ket programozták, ott volt Bob Kahn és Vincent Cerf, aki kidolgozta a TCP/IP-t, és így tovább. Ezek a könyvek elmondják az internet történetét, legalábbis 2000-ig, tele anekdotákkal.

9.1.2. A fizikai réteg

Bellamy: *Digital Telephony*, 3. kiadás

Ez a tekintélyes könyv tartalmaz mindent, amit Ön valaha is tudni akart a távbeszélő rendszerről, sőt még annál is többet. Különösen érdekesek az átvitelről és a nyalábolásról, valamint a digitális kapcsolásról, a fényvezető szálakról, a mobiltelefonról és a DSL-ről szóló fejezetek.

Hu és Li: *Satellite-Based Internet: A Tutorial*

A műholdas internet-hozzáférés sokban különbözik a földi vonalakon keresztül történő eléréstől. Itt nem csak a nagy késleltetésről van szó, hiszen az útválasztás és a kapcsolás módja is eltérő. A szerzők ebben a dolgozatban a műholdas internet-hozzáféréshez kapcsolódó kérdések közül vizsgálnak meg néhányat.

Joel: *Telecommunications and the IEEE Communications Society*

Ez a cikk tömör, de meglepően átfogó formában írja le a távközlés történetét a telegráftól kezdve egészen a 802.11-ig, valamint foglalkozik még a rádió, a telefonok, az analóg és digitális kapcsolás, a tenger alatti kábelek, a digitális átvitel, az ATM, a televíziós műsorszórás, a műholdak, a kábeltévé, a fényvezető szálak, a mobiltelefonok, a csomagkapcsolás, az ARPANET és az internet témájával is.

Palais: *Fiber Optic Communication*, 5. kiadás

Az üvegszálakról szóló könyvek általában a szakemberekhez szólnak, de ez jobban érthető, mint a legtöbb. Foglalkozik hullámterelőkkel, fényforrásokkal, fényérzékelőkkel, csatolókkal, modulációval, zajjal és még sok más témával.

Su: *The UMTS Air Interface in RF Engineering*

Ez a könyv részletes áttekintést ad egy alapvető 3G mobiltelefon-rendszerről. Központi kérdésként tárgyalja a rádiós interfészt, vagy a vezeték nélküli protokollokat, amelyeket a mobilkészülék és a hálózati infrastruktúra között használnak.

Want: *RFID Explained*

Ez egy olvasmányos könyv arról, hogy a nem szokványos RFID-technika fizikai rétege hogyan működik. Az RFID minden vonatkozását tartalmazza, beleértve a lehetséges alkalmazásait is. Az RFID megvalósításának néhány való életből vett példája és az ezekből nyert tapasztalatok szintén megtalálhatók.

9.1.3. Az **adatkapcsolati réteg**

Kasim: *Delivering Carrier Ethernet*

Manapság az Ethernet nem csupán egy lokális hálózati technika. Az új divat az, hogy az Ethernetet mint nagy távolságú adatkapcsolatot használják a szolgáltatói Ethernethez. Ez a könyv esszék formájában a téma mélységeit tárja fel.

Lin és Costello: *Error Control Coding*, 2. kiadás

A hibák kijelzése és kijavítása egy megbízható számítógép-hálózat szempontjából központi kérdés. Ez a népszerű tankönyv elmagyarázza a legfontosabb kódokat az egyszerű Hamming-kódoktól a sokkal bonyolultabb kis sűrűségű paritásellenőrző kódokig. Mindezt megkísérli a minimálisan szükséges algebrai eszközökkel megtenni.

Stallings: *Data and Computer Communications*, 9. kiadás

A könyv második része foglalkozik az adatátvitellel és a különféle adatkapcsolatokkal, benne a hibajelzés, az ismétléssel történő hibajavítás és a forgalomszabályozás kérdéseivel.

9.1.4. A közeg-hozzáférési alréteg

Andrews és mások: *Fundamentals of WiMAX*

Ez az összefoglaló könyv a WiMAX-technika meghatározó leírását adja a széles sávú vezeték nélküli technika gondolatától a vezeték nélküli technika többszörös hozzáférésű rendszeren keresztül OFDM-mel és többszörös antennával történő használatáig. Oktatóanyag stílusban a legtöbb elérhető eljárásról informál, amely erről a nehéz anyagról található.

Gast: *802.11 Wireless Networks*, 2. kiadás

Ha valaki egy olvasmányos bevezetésre vágyik a 802.11 protokollokról és technikáról, akkor az itt kezdje. A MAC-alréteggel kezdődik, ezután bevezet a különféle fizikai rétegekbe és a biztonsági kérdésekbe is. A második kiadás azonban nem eléggé új ahhoz, hogy túl sokat szóljon a 802.11n-ről.

Perlman: *Interconnections*, 2. kiadás

Perlman könyve a hidak, útválasztók és az általában vett útválasztás hiteles, de szórakoztató tárgyalását adja. A szerző számos hálózati területen a világ egyik legnagyobb szaktekintélyének számít; ő dolgozta ki az IEEE 802 feszítőfás hídjainak algoritmusait is.

9.1.5. A hálózati réteg

Comer: *Internetworking with TCP/IP*, 1. kötet, 5. kiadás

Comer alapművet írt a TCP/IP-protokollkészletről. Több mint a fele az IP-vel és a hozzá kapcsolódó hálózati rétegbeli protokollokkal foglalkoznak. A többi fejezet elsősorban a felsőbb rétegekkel foglalkozik, ezeket is érdemes elolvasni.

Grayson és mások: *IP Design for Mobile Networks*

A tradicionális telefonhálózat és az internet ütköztetése mobiltelefon-hálózattal, amelyet belülről IP-vel valósítanak meg. A könyv arról szól, hogyan kell olyan hálózatot tervezni, amelyik mobiltelefon-szolgáltatást támogató IP-protokollt használ.

Huitema: *Routing in the Internet*, 2. kiadás

Ha mindazt tudni akarja az internetes útválasztásról, amit csak lehet, akkor ez a könyv Önnek szól. Mind a jól kiejthető algoritmusokat (például RIP és CDIR), mind a ki nem ejthetőket (például OSPF, IGRP és BGP) nagy részletességgel tárgyalja. Az újabb fejlesztések nincsenek benne, mert ez egy régi könyv, de ami benne van, azt nagyon jól megmagyarázza.

Koodli és Perkins: *Mobile Inter-networking with IPv6*

Két fontos hálózati réteg fejlesztését mutatja be egy kötetben: az IPv6-ot és a Mobil IP-t. Mindkét téma leírása kiváló. Perkins volt a mobil IP motorja.

Nucci és Papagiannaki: *Design, Measurement and Management of Large-Scale IP Networks*

Leírtuk már azt, hogy a hálózat hogyan működik, de azt nem, hogy azt Ön hogyan tervezné, telepítené és felügyelné, ha Ön volna az ISP. Ez a könyv ezt a rést tömi be, megvizsgálva a forgalomtervezés modern módszereit, és azt, hogy az ISP a hálózat felhasználásával hogyan nyújtja szolgáltatásait.

Perlman: *Interconnections*, 2. kiadás

A 12–15. fejezetekben a szerző az egyedi és többescímzéssel kapcsolatos útválasztó algoritmusok tervezését írja le WAN-ok és LAN-ok számára. A könyv legjobb része azonban a 18. fejezet, amelyben a szerző a hálózati protokollokkal kapcsolatos, több évtizedes tapasztalatait informális és vicces tárgyalási módban adja közre. Protokoll tervezőknek érdemes elolvasni.

Stevens: *TCP/IP Illustrated*, 1. kötet

A 3–10. fejezetek az IP és az ahhoz kapcsolódó protokollok (ARP, RARP és ICMP) átfogó tárgyalását adják, példákkal illusztrálva.

Varghese: *Network Algorithms*

Sok időt töltöttünk el azzal, hogy elmondtuk azt, hogy az útválasztók és más hálózati eszközök hogyan működnek együtt. Ez a köny ettől különbözik: arról szól, hogy az útválasztókat hogyan kell tervezni annak érdekében, hogy csomagokat továbbítsanak óriási sebességgel. A szerző kitalálója olyan okos algoritmusoknak, amelyeket a gyakorlatban használnak annak érdekében, hogy nagy sebességű hálózati elemeket szoftverben vagy hardverben valósítsanak meg.

9.1.6. A szállítási réteg

Comer: *Internetworking with TCP/IP*, 1. kötet, 5. kiadás

Mint fent említettük, Comer alapművet írt a TCP/IP-protokollkészletről. A könyv második fele az UDP-ről és a TCP-ről szól.

Farrell és Cahil: *Delay- and Disruption-Tolerant Networking*

Ez a rövid könyv az, amelyet el kell olvasni az architektúra, a protokollok és az olyan alkalmazások mélyebb megismerése érdekében, amelyeket „kihívás-hálózatnak" nevezznek, mivel veszélyes körülmények között kell működniük. A szerzők részt vettek a DTN kifejlesztésében az IETF DTN Kutatási Csoportban.

Stevens: *TCP/IP Illustrated*, 1. kötet

A 17–24. fejezetek példákkal illusztrált összefoglaló tárgyalást adnak a TCP-ről.

9.1.7. Az alkalmazási réteg

Berners-Lee és mások: *The World Wide Web*

Az a személy (és néhány CERN-beli kollégája) ad itt áttekintést a webről és annak jövőjéről, aki feltalálta azt. A cikk a web architektúrájára, az URL-ekre, a HTTP-re, a HTML-re, valamint a jövőbeli lehetséges fejlődési irányokra összpontosít, és összehasonlítja a webet más elosztott információs rendszerekkel is.

Held: *A Practical Guide to Content Delivery Networks*, 2. kiadás

Ez a könyv egyszerű eszközökkel magyarázza el azt, hogyan működik a CDN, kihangsúlyozza a tervezés és a helyes működés során figyelembe vett gyakorlati szempontokat.

Hunter és mások: *Beginning XML*, 4. kiadás

Nagyon sok könyv van, amelyik a HTML-ről, az XML-ről és a web szolgáltatásairól szól. Ez az 1000 oldalas könyv tartalmaz mindent, amit Ön erről tudni szeretne. Elmagyarázza nem csak azt, hogyan kell HTML- vagy XML-leírást készíteni, de azt is, hogyan kell olyan webes szolgáltatásokat fejleszteni, amelyek előállítanak és megváltoztatnak XML-t az AJAX, SOAP és más, gyakorlatban használt technikákkal.

Krishnamurty és Rexford: *Web Protocols and Practice*

Keresve sem találnánk olyan könyvet, mely ennél átfogóbban részletezné a web összes aspektusát. Amint azt várhatjuk, szó esik az ügyfelekről, kiszolgálókról, helyettesekről és a tárgyorsításról, de külön fejezetek szólnak a webes forgalomról és annak méréséről, valamint a web továbbfejlesztésére irányuló jelenlegi kutatásokról is.

Simpson: *Video Over IP*, 2. kiadás

A szerző széles körű bemutatást végez azzal kapcsolatban, hogyan lehet az IP-technikát mozgóképeknek hálózaton keresztüli továbbításához használni mind az interneten, mind a privát hálózatokon. Érdekessége a könyvnek az, hogy a videón keresztüli hálózattanulásra összpontosít.

Wittemburg: *Understanding Voice Over IP Technology*

Ez a könyv a beszéd IP-hálózaton keresztül történő továbbításáról szól, hangadatok IP-protokollokkal történő szállításától és a szolgáltatásminőség kérdéseitől kezdve az SIP és H.323 protokollkészletig bezárólag. A szükséges módon részletezi az anyagot, de elérhető és emészthető egységekben.

9.1.8. Hálózati biztonság

Anderson: *Security Engineering*, 2. kiadás

Ez a könyv bizonyos mértékig a szerző *Why Cryptosystems Fail* című könyvének 600 oldalas változata. A *Secrets and Lies*-hoz képest sokkal inkább műszaki megközelítést használ, de mégsem annyira műszaki, mint a *Network Security* (lásd alább). A szerző az alapvető biztonsági eljárások bemutatása után egész fejezeteket szentel a különböző alkalmazásoknak, többek között olyan területeken, mint a banki rendszerek, a nukleáris létesítmények irányítása, a biztonságos nyomtatás, a fizikai biztonság, az elektronikus hadviselés, a távközlési biztonság, az e-kereskedelem és a szerzői jogok védelme. A könyv harmadik része a rendszerek irányelveiről, menedzsmentjéről és kiértékeléséről szól.

Ferguson és mások: *Cryptography Engineering*

Sok könyv elmondja azt, hogyan működnek a népszerű titkosító algoritmusok. Ez a könyv azt mondja el, hogyan kell a titkosítást használni – miért tervezték a titkosító protokollokat olyannak, amilyenek, és hogyan kell ezeket összerakni egy rendszerben úgy, hogy elérhessük titkosítással kapcsolatos célunkat. Ez egy meglehetősen teljes könyv, amelyet fontos elolvasni mindazoknak, akik olyan rendszereket terveznek, amelyek a titkosítástól függnek.

Fridrich: *Steganography in Digital Media*

A szteganográfia visszanyúlik az ókori Görögországig, ahol a viaszt leolvasztották az üres fatáblákról, így titkos üzeneteket lehetett elhelyezni az alatta lévő fán, mielőtt a viaszt újra felvitték rá. Manapság az interneten lévő videó, hang és más tartalmak nyújtanak különböző szállítási lehetőségeket titkos üzenetek számára. Az információ képekben történő elrejtésére és megtalálására szolgáló különféle modern módszereket tárgyal ez a könyv.

Kaufman és mások: *Network Security*, 2. kiadás

Azoknak, akik a hálózati biztonságért felelős algoritmusok és protokollok műszaki részletei iránt érdeklődnek, érdemes legelőször is ehhez a mérvadó és szellemes könyvhöz fordulniuk. Titkos és nyilvános kulcsú algoritmusok és protokollok, üzenet-hashek, hitelesítés, Kerberos, PKI, IPsec, SSL/TLS, elektronikus levelezés biztonsága – a mű mindezen témákat számos példával illusztrálva gondosan és jelentős terjedelemben részletezi. A biztonsági folklórról szóló 26. fejezet igazi gyöngyszemnek számít. A biztonság világában az ördög az apró részletekben rejtőzik. Ez a fejezet a való életből vett tanácsai révén sokat segíthet azoknak, akik egy ténylegesen is használatba állítandó biztonsági rendszert szeretnének tervezni.

Schneier: *Secrets and Lies*

Ha az elejétől a végéig elolvasta az Applied Crypthography c. könyvet, akkor már mindent tud a kriptográfiai algoritmusokról, amit csak tudni lehet. Ha azonban ezt a művet is végigolvassa (amit sokkal gyorsabban megtehet), akkor azt is tudni fogja, hogy a kriptográfiai algoritmusokkal még korántsincs vége a történetnek. A legtöbb biztonsági rés ugyanis nem a hibás algoritmusoknak vagy a túlságosan rövid kulcsoknak köszönhető, hanem a biztonsági környezet hiányosságainak. A könyv a veszélyekről, támadásokról, védekezési módokról, ellentámadásokról és egyebekről szóló példák végtelen sorával szolgál. A mű a legtágabb értelemben vett számítógépes biztonság lenyűgöző, nem műszaki jellegű tárgyalását adja.

Skoudis és Liston: *Counter Hack Reloaded*, 2. kiadás

A hackereket úgy lehet a legjobban megállítani, ha mi magunk is hackerként gondolkodunk. A könyv azt mutatja be, hogy hogyan látják a hackerek a hálózatot, és amellett érvel, hogy a biztonságnak egy utólagosan kiépített, egyetlen konkrét technikára épülő funkció helyett az egész rendszer szerves részét kell képeznie. A szerző szinte az összes gyakori támadástípust tárgyalja, beleértve azt az emberi tényezőn alapuló változatot is, mely azt használja ki, hogy sok felhasználó nem igazán van tisztában a számítógépes biztonsági óvintézkedésekkel.

9.2. Irodalomjegyzék

ABRAMSON, N.: Internet Access Using VSATs, *IEEE Commun. Magazine*, vol. 38, pp. 60–68, July 2000.

AHMADI, S.: An Overview of Next-Generation Mobile WiMAX Technology, *IEEE Commun. Magazine*, vol. 47, pp. 84–88, June 2009.

ALLMAN, M.–PAXSON, V.: On Estimating End-to-End Network Path Properties, *Proc. SIGCOMM '99 Conf.*, ACM, pp. 263–274, 1999.

ANDERSON, C.: *The Long Tail: Why the Future of Business is Selling Less of More*, rev. upd. ed., New York: Hyperion, 2008a.

ANDERSON, R. J.: *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd ed., New York: John Wiley & Sons, 2008b.

ANDERSON, R. J.: Free Speech Online and Offline, *IEEE Computer*, vol. 25, pp. 28–30, June 2002.

ANDERSON, R. J.: The Eternity Service, *Proc. Pragocrypt Conf.*, CTU Publishing House, pp. 242–252, 1996.

ANDREWS, J.–GHOSH, A.–MUHAMED, R.: *Fundamentals of WiMAX: Understanding Broadband Wireless Networking*, Upper Saddle River, NJ: Pearson Education, 2007.

ASTELY, D.–DAHLMAN, E.–FURUSKAR, A.–JADING, Y.–LINDSTROM, M.–PARKVALL, S.: LTE: The Evolution of Mobile Broadband, *IEEE Commun. Magazine*, vol. 47, pp. 44–51, Apr. 2009.

BALLARDIE, T.–FRANCIS, P.–CROWCROFT, J.: Core Based Trees (CBT), *Proc. SIGCOMM '93 Conf.*, ACM, pp. 85–95, 1993.

BARAN, P.: On Distributed Communications: I. Introduction to Distributed Communication Networks, *Memorandum RM-420-PR*, Rand Corporation, Aug. 1964.

BELLAMY, J.: *Digital Telephony*, 3rd ed., New York: John Wiley & Sons, 2000.

BELLMAN, R. E.: *Dynamic Programming*, Princeton, NJ: Princeton University Press, 1957.

BELLOVIN, S.: The Security Flag in the IPv4 Header, RFC 3514, Apr. 2003.

BELSNES, D.: Flow Control in the Packet Switching Networks, *Communications Networks*, Uxbridge, England: Online, pp. 349–361, 1975.

BENNET, C. H.–BRASSARD, G.: Quantum Cryptography: Public Key Distribution and Coin Tossing, *Int'l Conf. on Computer Systems and Signal Processing*, pp. 175–179, 1984.

BERESFORD, A.–STAJANO, F.: Location Privacy in Pervasive Computing, *IEEE Pervasive Computing*, vol. 2, pp. 46–55, Jan. 2003.

BERGHEL, H. L.: Cyber Privacy in the New Millennium, *IEEE Computer*, vol. 34, pp. 132–134, Jan. 2001.

BERNERS-LEE, T.–CAILLIAU, A.–LOUTONEN, A.–NIELSEN, H.F.–SECRET, A.: The World Wide Web, *Commun. of the ACM*, vol. 37, pp. 76–82, Aug. 1994.

BERTSEKAS, D.–GALLAGER, R.: *Data Networks*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1992.

BHATTI, S. N.–CROWCROFT, J.: QoS Sensitive Flows: Issues in IP Packet Handling, *IEEE Internet Computing*, vol. 4, pp. 48–57, July–Aug. 2000.

BIHAM, E.–SHAMIR, A.: Differential Fault Analysis of Secret Key Cryptosystems, *Proc. 17th Ann. Int'l Cryptology Conf.*, Berlin: Springer-Verlag LNCS 1294, pp. 513–525, 1997.

BIRD, R.–GOPAL, I.–HERZBERG, A.–JANSON, P. A.–KUTTEN, S.–MOLVA, R.–YUNG, M.: Systematic Design of a Family of Attack-Resistant Authentication Protocols, *IEEE J. on Selected Areas in Commun.*, vol. 11, pp. 679–693, June 1993.

BIRRELL, A. D.–NELSON, B. J.: Implementing Remote Procedure Calls, *ACM Trans. on Computer Systems*, vol. 2, pp. 39–59, Feb. 1984.

BIRYUKOV, A.–SHAMIR, A.–WAGNER, D.: Real Time Cryptanalysis of A5/1 on a PC, *Proc. Seventh Int'l Workshop on Fast Software Encryption*, Berlin: Springer-Verlag LNCS 1978, pp. 1–8, 2000.

BLAZE, M.–BELLOVIN, S.: Tapping on My Network Door, *Commun. of the ACM*, vol. 43, p. 136, Oct. 2000.

BOGGS, D.–MOGUL, J.–KENT, C.: Measured Capacity of an Ethernet: Myths and Reality, *Proc. SIGCOMM '88 Conf.*, ACM, pp. 222–234, 1988.

BORISOV, N.–GOLDBERG, I.–WAGNER, D.: Intercepting Mobile Communications: The Insecurity of 802.11, *Seventh Int'l Conf. on Mobile Computing and Networking*, ACM, pp. 180–188, 2001.

BRADEN, R.: Requirements for Internet Hosts—Communication Layers, RFC 1122, Oct. 1989.

BRADEN, R.–BORMAN, D.–PARTRIDGE, C.: Computing the Internet Checksum, RFC 1071, Sept. 1988.

BRANDENBURG, K.: MP3 and AAC Explained, *Proc. 17th Intl. Conf.: High-Quality Audio Coding*, Audio Engineering Society, pp. 99–110, Aug. 1999.

BRAY, T.–PAOLI, J.–SPERBERG-MCQUEEN, C.–MALER, E.–YERGEAU, F.–COWAN, J.: Extensible Markup Language (XML) 1.1 (Second Edition), W3C Recommendation, Sept. 2006.

BRESLAU, L.–CAO, P.–FAN, L.–PHILLIPS, G.–SHENKER, S.: Web Caching and Zipf-like Distributions: Evidence and Implications, *Proc. INFOCOM Conf.*, IEEE, pp. 126–134, 1999.

BURLEIGH, S.–HOOKE, A.–TORGERSON, L.–FALL, K.–CERF, V.–DURST, B.–SCOTT, K.–WEISS, H.: Delay-Tolerant Networking: An Approach to Interplanetary Internet, *IEEE Commun. Magazine*, vol. 41, pp. 128–136, June 2003.

BURNETT, S.–PAINE, S.: *RSA Security's Official Guide to Cryptography*, Berkeley, CA: Osborne/McGraw-Hill, 2001.

BUSH, V.: As We May Think, *Atlantic Monthly*, vol. 176, pp. 101–108, July 1945.

CAPETANAKIS, J. I.: Tree Algorithms for Packet Broadcast Channels, *IEEE Trans. on Information Theory*, vol. IT–5, pp. 505–515, Sept. 1979.

CASTAGNOLI, G.–BRAUER, S.–HERRMANN, M.: Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits, *IEEE Trans. on Commun.*, vol. 41, pp. 883–892, June 1993.

CERF, V.–KAHN, R.: A Protocol for Packet Network Interconnection, *IEEE Trans. on Commun.*, vol. COM–2, pp. 637–648, May 1974.

CHANG, F.–DEAN, J.–GHEMAWAT, S.–HSIEH, W.–WALLACH, D.–BURROWS, M.–CHANDRA, T.–FIKES, A.–GRUBER, R.: Bigtable: A Distributed Storage System for Structured Data, *Proc. OSDI 2006 Symp.*, USENIX, pp. 15–29, 2006.

CHASE, J. S.–GALLATIN, A. J.–YOCUM, K. G.: End System Optimizations for High-Speed TCP, *IEEE Commun. Magazine*, vol. 39, pp. 68–75, Apr. 2001.

CHEN, S.–NAHRSTEDT, K.: An Overview of QoS Routing for Next-Generation Networks, *IEEE Network Magazine*, vol. 12, pp. 64–69, Nov./Dec. 1998.

CHIU, D.–JAIN, R.: Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks, *Comput. Netw. ISDN Syst.*, vol. 17, pp. 1–4, June 1989.

CISCO: Cisco Visual Networking Index: Forecast and Methodology, 2009–2014, Cisco Systems Inc., June 2010.

CLARK, D. D.: The Design Philosophy of the DARPA Internet Protocols, *Proc. SIGCOMM '88 Conf.*, ACM, pp. 106–114, 1988.

CLARK, D. D.: Window and Acknowledgement Strategy in TCP, RFC 813, July 1982.

CLARK, D. D.–JACOBSON, V.–ROMKEY, J.–SALWEN, H.: An Analysis of TCP Processing Overhead, *IEEE Commun. Magazine*, vol. 27, pp. 23–29, June 1989.

CLARK, D. D.–SHENKER, S.–ZHANG, L.: Supporting Real-Time Applications in an Integrated Services Packet Network, *Proc. SIGCOMM '92 Conf.*, ACM, pp. 14–26, 1992.

CLARKE, A. C.: Extra-Terrestrial Relays, *Wireless World*, 1945.

CLARKE, I.–MILLER, S. G.–HONG, T. W.–SANDBERG, O.–WILEY, B.: Protecting Free Expression Online with Freenet, *IEEE Internet Computing*, vol. 6, pp. 40–49, Jan.–Feb. 2002.

COHEN, B.: Incentives Build Robustness in BitTorrent, *Proc. First Workshop on Economics of Peer-to-Peer Systems*, June 2003.

COMER, D. E.: *The Internet Book*, 4th ed., Englewood Cliffs, NJ: Prentice Hall, 2007.

COMER, D. E.: *Internetworking with TCP/IP*, vol. 1, 5th ed., Englewood Cliffs, NJ: Prentice Hall, 2005.

CRAVER, S. A.–WU, M.–LIU, B.–STUBBLEFIELD, A.–SWARTZLANDER, B.–WALLACH, D. W.–DEAN, D.–FELTEN, E. W.: Reading Between the Lines: Lessons from the SDMI Challenge, *Proc. 10th USENIX Security Symp.*, USENIX, 2001.

CROVELLA, M.–KRISHNAMURTHY, B.: *Internet Measurement*, New York: John Wiley & Sons, 2006.

DAEMEN, J.–RIJMEN, V.: *The Design of Rijndael*, Berlin: Springer-Verlag, 2002.

DALAL, Y.–METCLFE, R.: Reverse Path Forwarding of Broadcast Packets, *Commun. of the ACM*, vol. 21, pp. 1040–1048, Dec. 1978.

DAVIE, B.–FARREL, A.: *MPLS: Next Steps*, San Francisco: Morgan Kaufmann, 2008.

DAVIE, B.–REKHTER, Y.: *MPLS Technology and Applications*, San Francisco: Morgan Kaufmann, 2000.

DAVIES, J.: *Understanding IPv6*, 2nd ed., Redmond, WA: Microsoft Press, 2008.

DAY, J. D.: The (Un)Revised OSI Reference Model, *Computer Commun. Rev.*, vol. 25, pp. 39–55, Oct. 1995.

DAY, J. D.–ZIMMERMANN, H.: The OSI Reference Model, *Proc. of the IEEE*, vol. 71, pp. 1334–1340, Dec. 1983.

DECANDIA, G.–HASTORIN, D.–JAMPANI, M.–KAKULAPATI, G.–LAKSHMAN, A.–PILCHIN, A.–SIVASUBRAMANIAN, S.–VOSSHALL, P.–VOGELS, W.: Dynamo: Amazon's Highly Available Key-value Store, *Proc. 19th Symp. on Operating Systems Prin.*, ACM, pp. 205–220, Dec. 2007.

DEERING, S. E.: SIP: Simple Internet Protocol, *IEEE Network Magazine*, vol. 7, pp. 16–28, May/June 1993.

DEERING, S.–CHERITON, D.: Multicast Routing in Datagram Networks and Extended LANs, *ACM Trans. on Computer Systems*, vol. 8, pp. 85–110, May 1990.

DEMERS, A.–KESHAV, S.–SHENKER, S.: Analysis and Simulation of a Fair Queueing Algorithm, *Internetwork: Research and Experience*, vol. 1, pp. 3–26, Sept. 1990.

DENNING, D. E.–SACCO, G. M.: Timestamps in Key Distribution Protocols, *Commun. of the ACM*, vol. 24, pp. 533–536, Aug. 1981.

DEVARAPALLI, V.–WAKIKAWA, R.–PETRESCU, A.–THUBERT, P.: Network Mobility (NEMO) Basic Support Protocol, RFC 3963, Jan. 2005.

DIFFIE, W.–HELLMAN, M. E.: Exhaustive Cryptanalysis of the NBS Data Encryption Standard, *IEEE Computer*, vol. 10, pp. 74–84, June 1977.

DIFFIE, W.–HELLMAN, M. E.: New Directions in Cryptography, *IEEE Trans. on Information Theory*, vol. IT–2, pp. 644–654, Nov. 1976.

DIJKSTRA, E. W.: A Note on Two Problems in Connexion with Graphs, *Numer. Math.*, vol. 1, pp. 269–271, Oct. 1959.

DILLEY, J.–MAGGS, B.–PARIKH, J.–PROKOP, H.–SITARAMAN, R.–WHEIL, B.: Globally Distributed Content Delivery, *IEEE Internet Computing*, vol. 6, pp. 50–58, 2002.

DINGLEDINE, R.–MATHEWSON, N.–SYVERSON, P.: Tor: The Second-Generation Onion Router, *Proc. 13th USENIX Security Symp.*, USENIX, pp. 303–320, Aug. 2004.

DONAHOO, M.-CALVERT, K.: *TCP/IP Sockets in C*, 2nd ed., San Francisco: Morgan Kaufmann, 2009.

DONAHOO, M.-CALVERT, K.: *TCP/IP Sockets in Java*, 2nd ed., San Francisco: Morgan Kaufmann, 2008.

DONALDSON, G.-JONES, D.: Cable Television Broadband Network Architectures, *IEEE Commun. Magazine*, vol. 39, pp. 122–126, June 2001.

DORFMAN, R.: Detection of Defective Members of a Large Population, *Annals Math. Statistics*, vol. 14, pp. 436–440, 1943.

DUTCHER, B.: *The NAT Handbook*, New York: John Wiley & Sons, 2001.

DUTTA-ROY, A.: An Overview of Cable Modem Technology and Market Perspectives, *IEEE Commun. Magazine*, vol. 39, pp. 81–88, June 2001.

EDELMAN, B.-OSTROVSKY, M.-SCHWARZ, M.: Internet Advertising and the Generalized Second-Price Auction: Selling Billions of Dollars Worth of Keywords, *American Economic Review*, vol. 97, pp. 242–259, Mar. 2007.

EL GAMAL, T.: A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, *IEEE Trans. on Information Theory*, vol. IT-1, pp. 469–472, July 1985.

EPCGLOBAL: *EPC Radio-Frequency Identity Protocols Class- Generation- UHF RFID Protocol for Communication at 860-MHz to 960-MHz Version 1.2.0*, Brussels: EPCglobal Inc., Oct. 2008.

FALL, K.: A Delay-Tolerant Network Architecture for Challenged Internets, *Proc. SIGCOMM 2003 Conf.*, ACM, pp. 27–34, Aug. 2003.

FALOUTSOS, M.-FALOUTSOS, P.-FALOUTSOS, C.: On Power-Law Relationships of the Internet Topology, *Proc. SIGCOMM '99 Conf.*, ACM, pp. 251–262, 1999.

FARRELL, S.-CAHILL, V.: *Delay- and Disruption-Tolerant Networking*, London: Artech House, 2007.

FELLOWS, D.-JONES, D.: DOCSIS Cable Modem Technology, *IEEE Commun. Magazine*, vol. 39, pp. 202–209, Mar. 2001.

FENNER, B.-HANDLEY, M.-HOLBROOK, H.-KOUVELAS, I.: Protocol Independent Multicast-Sparse Mode (PIM-SM), RFC 4601, Aug. 2006.

FERGUSON, N.-SCHNEIER, B.-KOHNO, T.: *Cryptography Engineering: Design Principles and Practical Applications*, New York: John Wiley & Sons, 2010.

FLANAGAN, D.: *JavaScript: The Definitive Guide*, 6th ed., Sebastopol, CA: O'Reilly, 2010.

FLETCHER, J.: An Arithmetic Checksum for Serial Transmissions, *IEEE Trans. on Commun.*, vol. COM-0, pp. 247–252, Jan. 1982.

FLOYD, S.-HANDLEY, M.-PADHYE, J.-WIDMER, J.: Equation-Based Congestion Control for Unicast Applications, *Proc. SIGCOMM 2000 Conf.*, ACM, pp. 43–56, Aug. 2000.

FLOYD, S.-JACOBSON, V.: Random Early Detection for Congestion Avoidance, *IEEE/ACM Trans. on Networking*, vol. 1, pp. 397–413, Aug. 1993.

FLUHRER, S.-MANTIN, I.-SHAMIR, A.: Weakness in the Key Scheduling Algorithm of RC4, *Proc. Eighth Ann. Workshop on Selected Areas in Cryptography*, Berlin: Springer-Verlag LNCS 2259, pp. 1–24, 2001.

FORD, B.: Structured Streams: A New Transport Abstraction, *Proc. SIGCOMM 2007 Conf.*, ACM, pp. 361–372, 2007.

FORD, L. R., Jr.-FULKERSON, D. R.: *Flows in Networks*, Princeton, NJ: Princeton University Press, 1962.

FORD, W.-BAUM, M. S.: *Secure Electronic Commerce*, Upper Saddle River, NJ: Prentice Hall, 2000.

FORNEY, G. D.: The Viterbi Algorithm, *Proc. of the IEEE*, vol. 61, pp. 268–278, Mar. 1973.

FOULI, K.-MALER, M.: The Road to Carrier-Grade Ethernet, *IEEE Commun. Magazine*, vol. 47, pp. S30–S38, Mar. 2009.

FOX, A.-GRIBBLE, S.-BREWER, E.-AMIR, E.: Adapting to Network and Client Variability via On-Demand Dynamic Distillation, *SIGOPS Oper. Syst. Rev.*, vol. 30, pp. 160–170, Dec. 1996.

FRANCIS, P.: A Near-Term Architecture for Deploying Pip, *IEEE Network Magazine*, vol. 7, pp. 30–37, May/June 1993.

FRASER, A. G.: Towards a Universal Data Transport System, *IEEE J. on Selected Areas in Commun.*, vol. 5, pp. 803–816, Nov. 1983.

FRIDRICH, J.: *Steganography in Digital Media: Principles, Algorithms, and Applications*, Cambridge: Cambridge University Press, 2009.

FULLER, V.-LI, T.: Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan, RFC 4632, Aug. 2006.

GALLAGHER, R. G.: A Minimum Delay Routing Algorithm Using Distributed Computation, *IEEE Trans. on Commun.*, vol. COM-5, pp. 73–85, Jan. 1977.

GALLAGHER, R. G.: Low-Density Parity Check Codes, *IRE Trans. on Information Theory*, vol. 8, pp. 21–28, Jan. 1962.

GARFINKEL, S.-SPAFFORD, G. (közreműködésével): *Web Security, Privacy, and Commerce*, Sebastopol, CA: O'Reilly, 2002.

GAST, M.: *802.11 Wireless Networks: The Definitive Guide*, 2nd ed., Sebastopol, CA: O'Reilly, 2005.

GERSHENFELD, N.-KRIKORIAN, R.-COHEN, D.: The Internet of Things, *Scientific American*, vol. 291, pp. 76–81, Oct. 2004.

GILDER, G.: Metcalfe's Law and Legacy, *Forbes ASAP*, Sepy. 13, 1993.

GOODE, B.: Voice over Internet Protocol, *Proc. of the IEEE*, vol. 90, pp. 1495–1517, Sept. 2002.

GORALSKI, W. J.: *SONET*, 2nd ed., New York: McGraw-Hill, 2002.

GRAYSON, M.-SHATZKAMER, K.-WAINNER, S.: *IP Design for Mobile Networks*, Indianapolis, IN: Cisco Press, 2009.

GROBE, K.-ELBERS, J.: PON in Adolescence: From TDMA to WDM-PON, *IEEE Commun. Magazine*, vol. 46, pp. 26–34, Jan. 2008.

GROSS, G.-KAYCEE, M.-LIN, A.-MALIS, A.-STEPHENS, J.: The PPP Over AAL5, RFC 2364, July 1998.

HA, S.-RHEE, I.-LISONG, X.: CUBIC: A New TCP-Friendly High-Speed TCP Variant, *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 64–74, June 2008.

HAFNER, K.-LYON, M.: *Where Wizards Stay Up Late*, New York: Simon & Schuster, 1998.

HALPERIN, D.-HEYDT-BENJAMIN, T.-RANSFORD, B.-CLARK, S.-DEFEND, B.-MORGAN, W.-FU, K.-KOHNO, T.-MAISEL, W.: Pacemakers and Implant-

able Cardi ac Defibrillators: Software Radio Attacks and Zero-Power Defenses, *IEEE Symp. on Security and Privacy*, pp. 129–142, May 2008.

HALPERIN, D.–HU, W.–SHETH, A.–WETHERALL, D.: 802.11 with Multiple Antennas for Dummies, *Computer Commun. Rev.*, vol. 40, pp. 19–25, Jan. 2010.

HAMMING, R.W.: Error Detecting and Error Correcting Codes, *Bell System Tech. J.*, vol. 29, pp. 147–160, Apr. 1950.

HARTE, L.–KELLOGG, S.–DREHER, R.–SCHAFFNIT, T.: *The Comprehensive Guide to Wireless Technology*, Fuquay-Varina, NC: APDG Publishing, 2000.

HAWLEY, G. T.: Historical Perspectives on the U.S. Telephone Loop, *IEEE Commun. Magazine*, vol. 29, pp. 24–28, Mar. 1991.

HECHT, J.: *Understanding Fiber Optics*, Upper Saddle River, NJ: Prentice Hall, 2005.

HELD, G.: *A Practical Guide to Content Delivery Networks*, 2nd ed., Boca Raton, FL: CRC Press, 2010.

HEUSSE, M.–ROUSSEAU, F.–BERGER-SABBATEL, G.–DUDA, A.: Performance Anomaly of 802.11b, *Proc. INFOCOM Conf.*, IEEE, pp. 836–843, 2003.

HIERTZ, G.–DENTENEER, D.–STIBOR, L.–ZANG, Y.–COSTA, X.–WALKE, B.: The IEEE 802.11 Universe, *IEEE Commun. Magazine*, vol. 48, pp. 62–70, Jan. 2010.

HOE, J.: Improving the Start-up Behavior of a Congestion Control Scheme for TCP, *Proc. SIGCOMM '96 Conf.*, ACM, pp. 270–280, 1996.

HU, Y.–LI, V. O. K.: Satellite-Based Internet: A Tutorial, *IEEE Commun. Magazine*, vol. 30, pp. 154–162, Mar. 2001.

HUITEMA, C.: *Routing in the Internet*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1999.

HULL, B.–BYCHKOVSKY, V.–CHEN, K.–GORACZKO, M.–MIU, A.–SHIH, E.–ZHANG, Y.–BALAKRISHNAN, H.–MADDEN, S.: CarTel: A Distributed Mobile Sensor Computing System, *Proc. Sensys 2006 Conf.*, ACM, pp. 125–138, Nov. 2006.

HUNTER, D.–RAFTER, J.–FAWCETT, J.–VAN DER LIST, E.–AYERS, D.–DUCKETT, J.–WATT, A.–MCKINNON, L.: *Beginning XML*, 4th ed., New Jersey: Wrox, 2007.

IRMER, T.: Shaping Future Telecommunications: The Challenge of Global Standardization, *IEEE Commun. Magazine*, vol. 32, pp. 20–28, Jan. 1994.

ITU (INTERNATIONAL TELECOMMUNICATION UNION): *ITU Internet Reports 2005: The Internet of Things*, Geneva: ITU, Nov. 2005.

ITU (INTERNATIONAL TELECOMMUNICATION UNION): *Measuring the Information Society: The ICT Development Index*, Geneva: ITU, Mar. 2009.

JACOBSON, V.: Compressing TCP/IP Headers for Low-Speed Serial Links, RFC 1144, Feb. 1990.

JACOBSON, V.: Congestion Avoidance and Control, *Proc. SIGCOMM '88 Conf.*, ACM, pp. 314–329, 1988.

JAIN, R.–ROUTHIER, S.: Packet Trains—Measurements and a New Model for Computer Network Traffic, *IEEE J. on Selected Areas in Commun.*, vol. 6, pp. 986–995, Sept. 1986.

JAKOBSSON, M.–WETZEL, S.: Security Weaknesses in Bluetooth, *Topics in Cryptology: CT-RSA 2001*, Berlin: Springer-Verlag LNCS 2020, pp. 176–191, 2001.

JOEL, A.: Telecommunications and the IEEE Communications Society, *IEEE Commun. Magazine*, 50th Anniversary Issue, pp. 6–14 and 162–167, May 2002.

JOHNSON, D.–PERKINS, C.–ARKKO, J.: Mobility Support in IPv6, RFC 3775, June 2004.

JOHNSON, D. B.–MALTZ, D.–BROCH, J.: DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks, *Ad Hoc Networking*, Boston: Addison-Wesley, pp. 139–172, 2001.

JUANG, P.–OKI, H.–WANG, Y.–MARTONOSI, M.–PEH, L.–RUBENSTEIN, D.: Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet, *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 96–107, Oct. 2002.

KAHN, D.: *The Codebreakers*, 2nd ed., New York: Macmillan, 1995.

KAMOUN, F.–KLEINROCK, L.: Stochastic Performance Evaluation of Hierarchical Routing for Large Networks, *Computer Networks*, vol. 3, pp. 337–353, Nov. 1979.

KARN, P.: MACA—A New Channel Access Protocol for Packet Radio, *ARRL/CRRL Amateur Radio Ninth Computer Networking Conf.*, pp. 134–140, 1990.

KARN, P.–PARTRIDGE, C.: Improving Round-Trip Estimates in Reliable Transport Protocols, *Proc. SIGCOMM '87 Conf.*, ACM, pp. 2–7, 1987.

KARP, B.–KUNG, H. T.: GPSR: Greedy Perimeter Stateless Routing for Wireless Networks, *Proc. MOBICOM 2000 Conf.*, ACM, pp. 243–254, 2000.

KASIM, A.: *Delivering Carrier Ethernet*, New York: McGraw-Hill, 2007.

KATABI, D.–HANDLEY, M.–ROHRS, C.: Internet Congestion Control for Future High Bandwidth-Delay Product Environments, *Proc. SIGCOMM 2002 Conf.*, ACM, pp. 89–102, 2002.

KATZ, D.–FORD, P. S.: TUBA: Replacing IP with CLNP, *IEEE Network Magazine*, vol. 7, pp. 38–47, May/June 1993.

KAUFMAN, C.–PERLMAN, R.–SPECINER, M.: *Network Security*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 2002.

KENT, C.–MOGUL, J.: Fragmentation Considered Harmful, *Proc. SIGCOMM '87 Conf.*, ACM, pp. 390–401, 1987.

KERCKHOFF, A.: La Cryptographie Militaire, *J. des Sciences Militaires*, vol. 9, pp. 5–38, Jan. 1883 and pp. 161–191, Feb. 1883.

KHANNA, A.–ZINKY, J.: The Revised ARPANET Routing Metric, *Proc. SIGCOMM '89 Conf.*, ACM, pp. 45–56, 1989.

KIPNIS, J.: Beating the System: Abuses of the Standards Adoption Process, *IEEE Commun. Magazine*, vol. 38, pp. 102–105, July 2000.

KLEINROCK, L.: Power and Other Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications, *Proc. Intl. Conf. on Commun.*, pp. 43.1.1–43.1.10, June 1979.

KLEINROCK, L.–TOBAGI, F.: Random Access Techniques for Data Transmission over Packet-Switched Radio Channels, *Proc. Nat. Computer Conf.*, pp. 187–201, 1975.

KOHLER, E.–HANDLEY, H.–FLOYD, S.: Designing DCCP: Congestion Control without Reliability, *Proc. SIGCOMM 2006 Conf.*, ACM, pp. 27–38, 2006.

KOODLI, R.–PERKINS, C. E.: *Mobile Inter-networking with IPv6*, New York: John Wiley & Sons, 2007.

KOOPMAN, P.: 32-Bit Cyclic Redundancy Codes for Internet Applications, *Proc. Intl. Conf. on Dependable Systems and Networks.*, IEEE, pp. 459–472, 2002.

KRISHNAMURTHY, B.–REXFORD, J.: *Web Protocols and Practice*, Boston: Addison-Wesley, 2001.

KUMAR, S.–PAAR, C.–PELZL, J.–PFEIFFER, G.–SCHIMMLER, M.: Breaking Ciphers with COPACOBANA: A Cost-Optimized Parallel Code Breaker, *Proc. 8th Cryptographic Hardware and Embedded Systems Wksp.*, IACR, pp. 101–118, Oct. 2006.

LABOVITZ, C.–AHUJA, A.–BOSE, A.–JAHANIAN, F.: Delayed Internet Routing Convergence, *IEEE/ACM Trans. on Networking*, vol. 9, pp. 293–306, June 2001.

LAM, C. K. M.–TAN, B. C. Y.: The Internet Is Changing the Music Industry, *Commun. of the ACM*, vol. 44, pp. 62–66, Aug. 2001.

LAOUTARIS, N.–SMARAGDAKIS, G.–RODRIGUEZ, P.–SUNDARAM, R.: Delay Tolerant Bulk Data Transfers on the Internet, *Proc. SIGMETRICS 2009 Conf.*, ACM, pp. 229–238, June 2009.

LARMO, A.–LINDSTROM, M.–MEYER, M.–PELLETIER, G.–TORSNER, J.–WIEMANN, H.: The LTE Link-Layer Design, *IEEE Commun. Magazine*, vol. 47, pp. 52–59, Apr. 2009.

LEE, J. S.–MILLER, L. E.: *CDMA Systems Engineering Handbook*, London: Artech House, 1998.

LELAND, W.–TAQQU, M.–WILLINGER, W.–WILSON, D.: On the Self-Similar Nature of Ethernet Traffic, *IEEE/ACM Trans. on Networking*, vol. 2, pp. 1–15, Feb. 1994.

LEMON, J.: Resisting SYN Flood DOS Attacks with a SYN Cache, *Proc. BSDCon Conf.*, USENIX, pp. 88–98, 2002.

LEVY, S.: Crypto Rebels, *Wired*, pp. 54–61, May/June 1993.

LEWIS, M.: *Comparing, Designing, and Deploying VPNs*, Indianapolis, IN: Cisco Press, 2006.

LI, M.–AGRAWAL, D.–GANESAN, D.–VENKATARAMANI, A.: Block-Switched Networks: A New Paradigm for Wireless Transport, *Proc. NSDI 2009 Conf.*, USENIX, pp. 423–436, 2009.

LIN, S.–COSTELLO, D.: *Error Control Coding*, 2nd ed., Upper Saddle River, NJ: Pearson Education, 2004.

LUBACZ, J.–MAZURCZYK, W.–SZCZYPIORSKI, K.: Vice over IP, *IEEE Spectrum*, pp. 42–47, Feb. 2010.

MACEDONIA, M. R.: Distributed File Sharing, *IEEE Computer*, vol. 33, pp. 99–101, 2000.

MADHAVAN, J.–KO, D.–LOT, L.–GANGPATHY, V.–RASMUSSEN, A.–HALEVY, A.: Google's Deep Web Crawl, *Proc. VLDB 2008 Conf.*, VLDB Endowment, pp. 1241–1252, 2008.

MAHAJAN, R.–RODRIG, M.–WETHERALL, D.–ZAHORJAN, J.: Analyzing the MAC-Level Behavior of Wireless Networks in the Wild, *Proc. SIGCOMM 2006 Conf.*, ACM, pp. 75–86, 2006.

MALIS, A.–SIMPSON, W.: PPP over SONET/SDH, RFC 2615, June 1999.

MASSEY, J. L.: Shift-Register Synthesis and BCH Decoding, *IEEE Trans. on Information Theory*, vol. IT-5, pp. 122–127, Jan. 1969.

MATSUI, M.: Linear Cryptanalysis Method for DES Cipher, *Advances in Cryptology—Eurocrypt 1993 Proceedings*, Berlin: Springer-Verlag LNCS 765, pp. 386–397, 1994.

MAUFER, T. A.: *IP Fundamentals*, Upper Saddle River, NJ: Prentice Hall, 1999.

MAYMOUNKOV, P.–MAZIERES, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric, *Proc. First Intl. Wksp. on Peer-to-Peer Systems*, Berlin: Springer-Verlag LNCS 2429, pp. 53–65, 2002.

MAZIERES, D.–KAASHOEK, M. F.: The Design, Implementation, and Operation of an Email Pseudonym Server, *Proc. Fifth Conf. on Computer and Commun. Security*, ACM, pp. 27–36, 1998.

MCAFEE LABS: *McAfee Threat Reports: First Quarter 2010*, McAfee Inc., 2010.

MENEZES, A. J.–VANSTONE, S. A.: Elliptic Curve Cryptosystems and Their Implementation, *Journal of Cryptology*, vol. 6, pp. 209–224, 1993.

MERKLE, R. C.–HELLMAN, M.: Hiding and Signatures in Trapdoor Knapsacks, *IEEE Trans. on Information Theory*, vol. IT-4, pp. 525–530, Sept. 1978.

METCALFE, R. M.: Computer/Network Interface Design: Lessons from Arpanet and Ethernet, *IEEE J. on Selected Areas in Commun.*, vol. 11, pp. 173–179, Feb. 1993.

METCALFE, R. M.–BOGGS, D. R.: Ethernet: Distributed Packet Switching for Local Computer Networks, *Commun. of the ACM*, vol. 19, pp. 395–404, July 1976.

METZ, C: Interconnecting ISP Networks, *IEEE Internet Computing*, vol. 5, pp. 74–80, Mar.–Apr. 2001.

MISHRA, P. P.–KANAKIA, H.–TRIPATHI, S.: On Hop by Hop Rate-Based Congestion Control, *IEEE/ACM Trans. on Networking*, vol. 4, pp. 224–239, Apr. 1996.

MOGUL, J. C.: IP Network Performance, in *Internet System Handbook*, D.C. Lynch and M.Y. Rose (eds.), Boston: Addison-Wesley, pp. 575–575, 1993.

MOGUL, J.–DEERING, S.: Path MTU Discovery, RFC 1191, Nov. 1990.

MOGUL, J.–MINSHALL, G.: Rethinking the Nagle Algorithm, *Comput. Commun. Rev.*, vol. 31, pp. 6–20, Jan. 2001.

MOY, J.: Multicast Routing Extensions for OSPF, *Commun. of the ACM*, vol. 37, pp. 61–66, Aug. 1994.

MULLINS, J.: Making Unbreakable Code, *IEEE Spectrum*, pp. 40–45, May 2002.

NAGLE, J.: On Packet Switches with Infinite Storage, *IEEE Trans. on Commun.*, vol. COM-5, pp. 435–438, Apr. 1987.

NAGLE, J.: Congestion Control in TCP/IP Internetworks, *Computer Commun. Rev.*, vol. 14, pp. 11–17, Oct. 1984.

NAUGHTON, J.: *A Brief History of the Future*, Woodstock, NY: Overlook Press, 2000.

NEEDHAM, R. M.–SCHROEDER, M. D.: Using Encryption for Authentication in Large Networks of Computers, *Commun. of the ACM*, vol. 21, pp. 993–999, Dec. 1978.

NEEDHAM, R. M.–SCHROEDER, M. D.: Authentication Revisited, *Operating Systems Rev.*, vol. 21, p. 7, Jan. 1987.

NELAKUDITI, S.–ZHANG, Z.-L.: A Localized Adaptive Proportioning Approach to QoS Routing, *IEEE Commun. Magazine* vol. 40, pp. 66–71, June 2002.

NEUMAN, C.–TS'O, T.: Kerberos: An Authentication Service for Computer Networks, *IEEE Commun. Mag.*, vol. 32, pp. 33–38, Sept. 1994.

NICHOLS, R. K.–LEKKAS, P. C.: *Wireless Security*, New York: McGraw-Hill, 2002.