```
                S: 220 xyz.com SMPT service ready
C: HELO abc.com
                S: 250 xyz.com says hello to abc. com
C: MAIL FROM: <elinor@abc.com>
                S: 250 sender ok
C: RCPT TO: <carolyn@xyz.com>
                S: 250 recipient ok
C: DATA
  S: 354 Send mail; end with "." on a line by itself
C: From: elinor@abc.com
C: To: carolyn@xyz.com
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@abc.com>
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: Earth orbits sun integral number of times
C:
C: Ez a bevezető. A felhasználói ügynök figyelmen kívül hagyja. Minden jót.
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: text/richtext
C:
C: Boldog szülinapot,
C: Boldog szülinapot,
C: Boldog szülinapot, kedves <bold> Carolyn </bold>
C: Boldog szülinapot
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
C:     access-type="anon-ftp";
C:     site="bicycle.abc.com";
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 xyz.com closing connection
```

7.47. ábra. *Egy üzenet küldése az elinor@abc.com-ról a carolyn@xyz com-ra*

Néhány probléma elkerülésére az 1451-es RFC-ben definiálták a kiterjesztett SMTP-t **(extended SMTP – ESMTP).** Azok a kliensek, amelyek ezt szeretnék használni elsőnek a *EHLO* parancsot kell kiadják a *HELO* helyett. He ezt visszautasítja, akkor a szerver egy általános szerver, és a kliensnek a szokásos módon kell működnie. Amennyiben az *ELHO*-t elfogadja a szerver, akkor az új parancsok és paraméterek is életbe lépnek. Folyamatban van ezeknek a parancsoknak és paramétereknek a szabványosítása.

E-levél átjárók

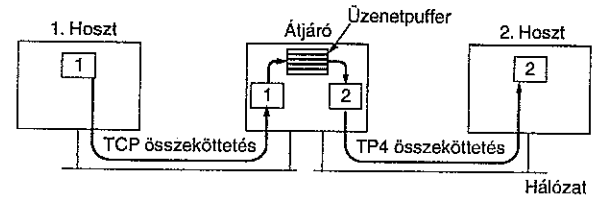
Az e-levél akkor működik a legjobban, ha mind a küldő, mind a fogadó az Interneten van, és támogatják a küldő és fogadó között a TCP összeköttetést. Azonban számos olyan gép, szeretne e-levelet fogadni, az Internetről, amely nincs az Internetre kötve.

Például néhány cég, biztonsági okokból, kifejezetten nem akar az Interneten lenni. Közülük néhányan ennek elérésére tűzfalakat (firewall) emelnek maguk és az Internet közé. Probléma lehet az is, ha a küldő csak a 822-es RFC-t beszéli, míg a fogadó csak az X.400-at vagy más szabadalmazott gyártóspecifikus levelező protokollt érti meg. A közvetlen kommunikáció lehetetlen, mivel ezekben mind különböznek az üzenet formátumok, illetve a protokollok.

Mindkét problémát megoldja az alkalmazási rétegbeli **e-levél átjárók (email gateway)** használata. A 7.48. ábrán az 1-es hoszt csak a TCP/IP-t és a 822-es RFC-t ismeri, míg a 2-es hoszt csak az OSI TP4-et és az X.400-at. Az 1-es hoszt által követendő folyamat az, hogy létesít egy TCP kapcsolatot az átjáróval, majd SMTP segítségével elküldi az (1-es) üzenetet. Az átjárón futó daemon ezután beteszi az üzenetet egy, a 2-es hosztnak szóló, üzenet-pufferbe. Később létrejön egy TP4 (az OSI megfelelője a TCP-nek) kapcsolat a 2-es hoszttal, és az üzenet az SMTP OSI megfelelőjével átkerül a 2-es hosztra. Az átjárónak mindössze annyit kell tennie, hogy kihámozza a bejövő üzenetet az egyik sorból, és beteszi azt egy másikba.

Egyszerűnek látszik, azonban nem az. Az első probléma az, hogy az Internet címek teljesen különbözőek az X.400 címektől. Egy bonyolult megfeleltetési mechanizmusra van köztük szükség. A második probléma, hogy az egyik rendszerben levő boríték vagy fejrész mezők a másikban nem feltétlenül léteznek. Például, ha az egyik rendszer prioritási osztályokat használ, a másikban viszont egyáltalán nem is létezik ilyesmi, akkor az egyik irányban ez a fontos információ elvész, míg a másik irányban valahogyan elő kell azt teremteni a semmiből.

Még ennél is nagyobb gond, hogy mi legyen a nem kompatíbilis szövegrészekkel. Mit kellene tennie egy átjárónak egy olyan Internetről érkezett üzenettel, melynek szövegrésze egy FTP-vel elérhető fájlra utal, miközben a célrendszer nem támogatja ezt a fogalmat? Mit tegyen, ha egy X.400 rendszer utasítja, hogy kézbesítsen egy levelet valamilyen címre, de ha az nem megy, akkor küldje el faxon? A fax használata nem része a 822-es RFC modelljének. Itt tehát nem létezik egyszerű megoldás. Az egyszerű formázatlan szöveges ASCII üzenetekhez az átjáró ésszerű ötlet, de ami ennél cifrább, arra nehezebben húzható rá.



7.48. ábra. *E-levél kézbesítése alkalmazási rétegbeli átjáróval*

Végső kézbesítés

Eddig azzal a feltételezéssel éltünk, hogy a felhasználók olyan gépeken dolgoznak, amelyek képesek e-levelek küldésére és fogadására. Egyre inkább nem ez a helyzet. Például számos cégnél a felhasználók asztali PC-kel dolgoznak, amelyek nincsenek az Internetre kötve, és nem képesek cégen kívüli e-levelet fogadni vagy küldeni. Ehelyett a cégnek egy vagy több e-levél szervere van, amelyek képesek e-levél küldésére és fogadására. Az üzenetek küldéséhez vagy fogadásához a PC-nek valamiféle kézbesítő protokoll segítségével kapcsolatot kell teremtenie e-levél szerverrel.

A távoli postaládákban tárolt üzenetek elhozására használt egyszerű protokoll a **POP3 (Post Office Protocol – postahivatal protokoll),** amelyet az 1225-ös RFC definiál. Ez olyan parancsokat tartalmaz, melyek segítségével a felhasználó kiléphet, belépheti, elhozhatja vagy letörölheti leveleit. A protokoll maga ASCII szövegekből áll és kicsit az SMPT hangulatát idézi. A POP3 arra jó, hogy a felhasználó elhozhatja leveleit a távoli postaládából, és a saját gépén tárolhatja, majd később elolvashatja őket.

Egy kifinomultabb kézbesítési protokoll az **IMAP (Interactive Mail Access Protocol – interaktív levélelérési protokoll),** amelyet az 1064-es RFC definiál. Azon felhasználók számára tervezték, akiknek több számítógépük van, esetleg egy munkaállomás az irodában, egy PC otthon és egy laptop az utazáshoz. Az IMAP mögött meghúzódó alapötlet az, hogy minden e-levél szerver egy adatbázist tart karban, amely minden számítógépről elérhető. Így tehát ellentétben a POP3-mal az IMAP nem másolja át az üzeneteket a felhasználó saját gépére, hiszen több ilyen is van.

Az IMAP-nak számos szolgáltatása van, mint pl. a képesség, hogy a leveleket nem érkezési sorrendben jelenítse meg, mint ahogyan az a 7.40. ábrán van, hanem tulajdonságok szerint (pl. Mutasd meg a Samtől érkezett első levelet). Ebben a megközelítésben a postaláda inkább egy relációs adatbázishoz hasonlít, mintsem üzenetek lineáris sorozatához.

Egy harmadik kézbesítési protokoll a **DMSP (– Distributed Mail System Protocol – elosztott levelezési rendszer protokoll),** amely a PCMAIL rendszer része, és az 1056-os RFC definiálja. Ez nem feltételezi, mint az IMAP vagy POP3, hogy minden levél egy szerveren van tárolva. Ehelyett lehetővé teszi, hogy a felhasználó letöltse a szerverről egy munkaállomásra, PC-ra vagy laptopra a leveleket, majd bontsa a kapcsolatot. Az e-leveleket szétkapcsolt állapotban lehet elolvasni és megválaszolni. Egy későbbi újbóli kapcsolatteremtés során az e-levelek átmásolódnak, és a rendszer újraszinkronizálódik.

Sok rendszer lehetővé teszi a beérkező e-levelek további feldolgozását, függetlenül attól, hogy az e-levelek közvetlenül a felhasználó munkaállomására vagy egy távoli szerverre kézbesítődnek. Egy kifejezetten értékes szerszám a felhasználó számára, hogy **szűrőket (filters)** definiálhat. Ezek a szabályok, akkor érvényesülnek, ha egy levél érkezik, vagy akkor, amikor a felhasználói ügynököt elindítják. Minden szabály egy feltételt és egy akciót tartalmaz. Például egy szabály azt tartalmazhatja, hogy minden Andrew S. Tanenbaumtól érkező levelet 24-es méretű piros villogó félkövér betűkkel kell megjeleníteni (vagy esetleg, megjegyzés nélküli automatikusan el kell dobni).

Egy másik gyakori szolgáltatás az a képesség, amely a leveleket (átmenetileg) más

címre továbbítja. Ez a cím akár egy kereskedelmi személyhívó szolgáltató által üzemeltetett számítógép is lehet, amely aztán a személyhívón a *Subject:* sorban levő üzenetet megjelenítésével rádión vagy műhold segítségével jelez a felhasználónak.

A végső kézbesítés ugyancsak gyakorta használt szolgáltatása a **távollétet jelző démon (vacation daemon)** telepítése. Ez egy program, amely megvizsgálja a beérkező levelet és a következő unalmas választ küldi a feladónak

Szia. Nyaralni mentem. Augusztus 24-én jövök meg. Minden jót.

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. A legtöbb távollétet jelző démon számon tartja, hogy kinek küldött már ilyen konzervlevelet, és többször nem ismétli azt meg. A jobbak azt is megnézik, hogy a beérkezett levelet nem egy levelezési lista alapján küldték-e, mert ha igen, akkor egyáltalán nem küld választ. (Azok, akik a nyári időszakban nagy levelezési listákra levelet küldenek, valószínűleg nem szeretnének mindenkitől nyaralásra vonatkozó terveket tartalmazó válaszokat kapni.)

A szerző nemrégiben egy egészen szélsőséges formájával találkozott a kézbesítést követő feldolgozásnak, amikor levelet küldött egy olyan személynek, aki naponta állítólag 600 levelet kap. Személyazonosságát itt nem hozzuk nyilvánosságra, mivel a könyv olvasóinak legalább a fele szintén küldene neki egy levelet. Nevezzük egyszerűen Johnnak.

John telepített egy e-levél robotot, amely minden levelet ellenőriz, hogy lássa, hogy az egy új levelezőtől jött-e. Ha igen, akkor visszaküld egy konzervválaszt, ami elmagyarázza, hogy John ezentúl már nem képes személyesen elolvasni minden e-levelet. Ehelyett készített egy személyes **FAQ (Frequently Asked Questions – gyakran feltett kérdések)** dokumentumot, ami megválaszolja a gyakori kérdéseket. Általában a hírcsoportoknak vannak FAQ-jaik, nem pedig személyeknek.

John FAQ-ja megadja a címét, a fax- és telefonszámát, és megadja, hogy hogyan lehet a cégét elérni. Megmagyarázza, hogyan lehet felkérni előadásra, és hogy hol találhatók a publikációi, valamint egyéb dokumentumok. Ezenkívül mutatókat ad az általa írt programokra, az általa vezetett konferenciákra, az általa kidolgozott szabványokra és így tovább. Lehetséges, hogy ez a megközelítés szükséges, de az is lehet, hogy egy ilyen személyes FAQ valójában csupán státuszszimbólum.

7.4.5. E-levéllel kapcsolatos személyiségi jogok

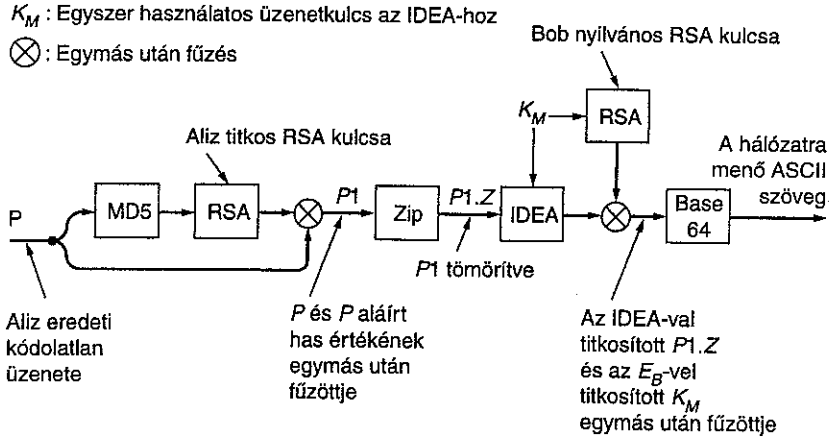
Ha egy e-levelet egy távoli helyre küldenek, akkor az általában egy tucat közbülső gépen megy keresztül. Mindegyik elolvashatja, és tárolhatja a levelet későbbi felhasználás céljából. Személyiségi jog nem létezik annak ellenére, hogy sokan azt hiszik, létezik (Weisband és Reining, 1995). Akárhogy is van, sokan szeretnék azt, hogy képesek legyenek olyan e-levelet küldeni, amit csak a szándékolt címzett tud elolvasni és senki más nem: sem a főnökük, sem a számítógépbetyárok, de még a hatóság sem. Ez a

szükség sok embert és csoportot rávitt arra, hogy alkalmazzák a korábban megvizsgált titkosítási elveket az e-leveleken, hogy ezzel biztonságos e-leveleket készíthessenek. A következő részben két elterjedten használt e-levél rendszert fogunk megvizsgálni, a PGP-t és a PEM-et. További információ tekintetében lásd (Kaufman és mások, 1995; Schneier, 1995; Stallings, 1995b; Stallings 1995c) műveit.

PGP – Elég jól biztosított személyiségi jog

Első példánk, a **PGP (Pretty Good Privacy – elég jól biztosított személyiségi jog)** tulajdonképpen egyetlen személy agyának szüleménye, Phil Zimmermanné (Zimmermann, 1995a, 1995b). Ez egy komplett e-levél biztonsági csomag, amely felkínálja a személyiségi jogok védelmét, a hitelességvizsgálatot, a digitális aláírásokat és a tömörítést egyben, méghozzá mindegyiket egyszerűen használható módon. Ezenkívül az egész csomag, a forráskódot is beleértve, szabadon terjeszthető az Interneten, a BBS-eken, valamint a kereskedelmi hálózatokon. A minőségének, az árának (nulla) és annak köszönhetően, hogy könnyen elérhető MS-DOS/Windows, UNIX és Macintosh rendszerekre, manapság már széles körben használják. Egy kereskedelmi változata is létezik azon cégek számára, akik a termékre vonatkozó támogatást igénylik.

Ez a csomag jó néhány vitának volt tárgya (Levy, 1993). Mivel az Interneten szabadon elérhető, ezért az USA kormánya kijelentette, hogy mivel idegen állampolgárok szabadon elérhetik, megszegi a hadianyag kivitelről szóló törvényt. Későbbi verziók már az Egyesült Államokon kívül készültek, hogy ezt a megszorítást elkerüljék. Egy másik probléma az RSA szabvány állítólagos megsértése volt, azonban ez a vita a 2.6-os változat kihozatalával elült. Akárhogyan is van, nem mindenkinek tetszik az ötlet, hogy az emberek titkolóznak előttük, ezért a PGP ellenségei mindig ott leselkednek az



7.49. ábra. *Üzenetküldés PGP-vel*

árnyékban, és arra várnak, hogy lecsaphassanak. Phil Zimmermann mottója szerint: „Ha a személyiségi jogok számkivetettek, akkor csak a számkivetetteknek lesznek személyiségi jogaik" („If privacy is outlawed, only outlaws will have privacy").

A PGP új titkosítási eljárások feltalálása helyett meglevőket alkalmaz. Erősen az RSA-n, az IDEA-n, valamint az MD5-ön alapul, amelyek mind kiállták az idők próbáját, és tervezésükben nem vettek részt, vagy nem befolyásolták őket hatósági ügynökségek, amelyek megpróbálták volna gyengíteni őket. Azok számára, akik nem bíznak a hatóságokban, ez nagy előnyt jelent.

A PGP támogatja a szövegtömörítést, a biztonságosságot, a digitális aláírásokat és kimerítő kulcskezelési szolgáltatásokat is nyújt. Ahhoz, hogy lássuk a PGP hogyan működik vizsgáljuk meg a 7.49. ábrán látható példát. Aliz biztonságosan el szeretne küldeni Bobnak egy aláírt szöveges üzenetet, *P*-t. Mind Aliznak, mind Bobnak vannak titkos (D_X) és nyilvános (E_X) RSA kulcsaik. Tegyük fel, hogy ismerik egymás nyilvános kulcsát. A kulcsok kezelésére később térünk ki.

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 (pl. 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 értékkel rendelkező 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 viszonykulcsnak nevez a PGP irodalom, de ez valójában egy névhiba, hiszen itt nincs szó viszonyról). Ezután megtörténik a *P1.Z* visszacsatolásos kódolása, K_M-mel és az IDEA-val. Valamint K_M is titkosításra kerül Bob nyilvános kulcsával. Ezután elkészül a két kód egymás után fűzöttjének base64 kódoltja, ahogyan azt a MIME-ról szóló részben láttuk. Az eredmény ezután csak betűket, számokat és a +, / és = szimbólumokat tartalmazza, tehát beletehető egy 822-es RFC 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. Kitömörítés 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, tömörí-

test és digitális aláírást biztosít, és mindezt sokkal hatékonyabban, mint a 7.23. ábrán látható séma.

A PGP három RSA kulcshosszúságot támogat. A felhasználótól függ, hogy kiválassza a legmegfelelőbbet. A hosszúságok a következőek:

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.

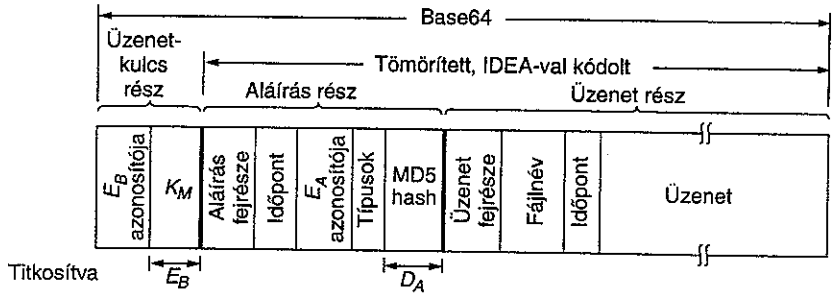
Szó volt egy negyedik kategóriáról: világegyetemi (2048 bit), amelyet senki vagy semmi az egész univerzumból nem tudna feltörni, de ezt egyelőre még nem fogadták el. Mivel az RSA csak két rövid számítás erejéig kell, valószínűleg célszerű, ha mindig mindenki katonasági erősségű kulcsokat használ, kivéve esetleg az öreg PC-XT-k esetében.

A PGP üzenet formátuma a 7.50. ábrán látható. Az üzenet három részből áll, amelyek rendre az IDEA kulcsot, az aláírást és az üzenetet tartalmazzák. A kulcsrész nem csak a kulcsot, hanem egy kulcsazonosítót is tartalmaz, hiszen a felhasználónak több kulcsa 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 típus információ 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 kulcsgyűrűt és a nyilvános kulcsgyűrűt. Az **egyéni kulcs gyűrű (private key ring)** egy vagy több titkos-nyilvános kulcspárt tartalmaz. A több kulcs megengedésé-



7.50. ábra. *Egy PGP üzenet*

nek oka az, hogy a felhasználónak lehetősége legyen a titkos kulcsait időnként, vagy ha úgy érzi, hogy kompromittálták, 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 üzenet azonosítók 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 kulcsgyűrű (public key ring)** a felhasználó levelezőpartnereinek nyilvános kulcsait tárolja. Ezek az üzenetekhez tartozó üzenetkulcsok visszakódolásához kellenek. A nyilvános kulcsgyűrű 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 élő-lánc támadást.

Az ilyen támadások megelőzésére, vagy legalábbis a következményeik minimalizálására, Aliznak tudnia kell, hogy mennyire bízhat meg a nyilvános kulcs gyűrűjében levő „Bob kulcsának" nevezett dologban. Ha tudja, hogy Bob személyesen adta azt oda egy lemezen, akkor a megbízhatósági értéket a legmagasabbra állíthatja.

A gyakorlatban azonban az emberek gyakran egy megbízható kulcs szervertől szerzik a nyilvános kulcsokat, amiből már működik néhány az Interneten. A kulcs szerverek egy nyilvános kulcs kérésére válaszul generálnak egy üzenetet, amely tartalmazza a nyilvános kulcsot, egy időbélyeget és a kulcs lejártának időpontját. Ezután ellátja a választ egy MD5-ös hash-sel és aláírja azt saját titkos kulcsával, így a kérdező ellenőrizni tudja, hogy kitől érkezett. A felhasználótól függ, hogy beállítson egy megbízhatósági szintet a helyi rendszergazda, a telefonszolgáltató, az ACM, az Ügyvédi közösség, a hatóság, vagy akárki, aki a kulcskarbantartással foglalkozik, által karbantartott kulcsok számára.

PEM – Megnövelt személyiségi jogokat biztosító levél

A PGP-vel ellentétben, amely eredetileg egy ember szüleménye, második példánk a **PEM (Privacy Enhanced Mail – megnövelt személyiségi jogokat biztosító levél)**, egy hivatalos Internet szabvány, amelyet négy RFC tartalmaz: az RFC 1421–1424. Nagy általánosságban a PEM ugyanazt a területet foglalja magába, mint a PGP: személyiségi jogok biztosítása és hitelességvizsgálat a 822-es RFC típusú e-levelezési rendszerekben. Vannak azonban benne mind technológiai, mind megközelítésbeli különbségek a PGP-vel szemben. A PEM-mel kapcsolatos további információk tekintetében lásd (Kent, 1993) művét.

A PEM-mel küldött üzenetek először egy kanonikus formát kapnak, így mindegyi-

kük ugyanazokat a konvenciókat használja a puha szóköz (white space) karakterek (pl. tabulátor, záró szóköz), a kocsi-vissza jel és a soremelés tekintetében. Ez a transzformáció azért kell, hogy semlegesítse azoknak az üzenetkézbesítő ügynököknek a hatását, amelyek módosítják azokat az üzeneteket, amelyek nem a nekik megfelelő formájúak. A kanonikus formára hozatal nélkül az ilyen módosítás befolyásolhatja az üzenetről a célállomáson készített hash értéket.

Ezután elkészül az üzenet hash, az MD2 vagy az MD5 segítségével. A módszer választható, csakúgy, mint a PGP-nél. Ezután elkészül az üzenet és a hozzáfűzött hash érték kódoltja a DES segítségével. Az 56 bites kulcs ismert gyengeségeinek tudatában ez a választás felettébb gyanús. A kódolt üzenetet ezután base64 formára lehet hozni, valamint el lehet küldeni a címzettnek. A PEM a levelezési listákat is támogatja.

Ahogyan a PGP-nél is, itt is minden üzenet egyszerhasználatos kulccsal van kódolva, ami az üzenet mellett szerepel. A kulcs vagy RSA-val, vagy a hármas DES, azaz az EDE használatával titkosítható. Gyakorlatilag mindenki az RSA-t használja, tehát mi is azzal foglalkozunk. Nem is tehetünk mást: a PEM nem mondja meg, hogyan kell a DES kulcsokat karbantartani.

A kulcskarbantartás sokkal szervezettebb, mint a PGP-nél. A kulcsokat az **igazoló hatóságok (certification authories)** igazolják, úgy hogy egy igazolás tartalmaz egy felhasználói nevet, egy nyilvános kulcsot és a kulcs lejárásának időpontját. Minden igazolást egy egyedi sorozatszám azonosít. Az igazolásokban van egy MD5 hash is, amit az igazoló hatóság saját titkos kulcsával aláír. Ezek az igazolások megfelelnek az ITU X.509-es nyilvános kulcsokra vonatkozó igazolásjavaslatnak, és mint ilyenek X.400-as címeket használnak, mint a korábbi példában szereplő Ken Smith címe.

A PGP hasonló sémát használ (az X.509 javaslat használata nélkül), de van egy hibája: Higgyen-e a felhasználó egy igazoló hatóságnak? A PEM úgy oldja meg ezt a problémát, hogy az úgynevezett **PCA-kkal (Policy Certification Authority – irányelvet igazoló hatóság)** igazolja az igazoló hatóságokat. Ezeket aztán az **IPRA (Internet Policy Registration Authority – internetes irányvonal nyilvántartó hatóság)**, igazolja, aki a legfelső bírája annak, hogy ki a huncut és ki a becsületes.

Minden PCA-nak definiálnia kell egy jelentkezési eljárást, és be kell azt jelentenie az IPRA-nak. Ezek az állásfoglalások azután nyilvánosságra kerülnek. Például egy PCA kijelentheti, hogy a birodalmába tartozó felhasználók mind személyesen megjelentek, és bemutatták a születési anyakönyvi kivonatukat, a jogosítványukat, az útlevelüket, két főbb hitelkártyájukat, hoztak egy élő tanút és nem utolsósorban megadták a nyilvános kulcsukat lemezen. Egy másik PCA esetleg elfogadhat e-leveles bejelentkezést idegenektől. Az irányelvek nyilvánosságra hozatalával a felhasználóknak lesz némi fogalmuk arról, hogy kiben mennyire lehet megbízni. Arra nézve nem történtek intézkedések, hogy kiderítsék vajon betartják-e a PCA-k az irányelveket.

Terv szerint háromféle igazoló hatóság kerülne bevezetésre. A szervezet a beosztottjainak adhatna igazolásokat. A legtöbb cég saját igazoló hatósággal rendelkezhetne. A lakossági a magánszemélyek számára lenne fenntartva, hasonlóan az Internet-szolgáltatókhoz, akik mindenkinek szolgáltatnak, aki fizet. És végül az anonim bejelentkezésekhez is lenne egy hatóság. Így most már érthető, hogy kellenek a pásztor szerepét játszó PCA-k.

Ez a séma ugyan szigorúan hierarchikus és bürokratikus, azonban megvan az az előnye a PGP-vel szemben, hogy az esetleges igazolás visszavonások a lehetőség szerint részlegesek. A visszavonás akkor szükséges, ha egy felhasználó meg szeretné változtatni a nyilvános kulcsát, mert pl. kompromittálták azt vagy ellopták, vagy kirabolták az igazoló hatóságát. A visszavonás úgy történik, hogy a felhasználó közli az igazoló hatóságával, hogy a kulcsát kompromittálták (vagy valószínűleg mindketten közlik egymással). Az igazoló hatóság ezután hozzáadja a most már érvénytelen kulcs sorozatszámát egy listához, amely a visszavont igazolásokat tartalmazza, aláírja és szétküldi azt.

Ha valaki a PEM-mel szeretne levelet küldeni egy másik felhasználónak, annak előbb meg kel vizsgálnia a legutóbbi visszavont kulcsok listáját, hogy megnézze, hogy a címzett átmeneti tárban levő kulcsa még érvényes-e. Ez az eljárás hasonlít ah-

Elem	PGP	PEM
Támogatja a titkosítást?	Igen	Igen
Támogatja a hitelességvizsgálatot?	Igen	Igen
Támogatja a digitális aláírásokat?	Igen	Igen
Támogatja a tömörítést?	Igen	Nem
Támogatja a kanonikus formára hozást?	Nem	Igen
Támogatja a levelezési listákat?	Nem	Igen
Base64 kódolást használ?	Igen	Igen
Jelenlegi titkosítási algoritmus	IDEA	DES
Kulcshossz az adattitkosításhoz (bitekben)	128	56
Jelenlegi kulcskezelő algoritmus	RSA	RSA vagy DES
A kulcskezeléshez való kulcsok hossza (bitekben)	384/512/1024	Változó
Felhasználói névtér	Felhasználó által definiált	X.400
Megfelel az X.509-nek?	Nem	Igen
Meg kell-e bízni valakiben?	Nem	Igen (IRPA)
Kulcsigazolás	Alkalomra szóló	IRPA/PCA/CA hierarchia
Kulcsvisszavonás	Bizonytalan	Jobb
A hallgatózók el tudják-e olvasni az üzeneteket?	Nem	Nem
A hallgatózók el tudják-e olvasni az aláírásokat?	Nem	Igen
Internet szabvány?	Nem	Igen
Tervezte	Kis csapat	Szabványügyi bizottság

7.51. ábra. *A PGP és a PEM egyfajta összehasonlítása*

hoz, amikor egy eladó végignézi az ellopott hitelkártyák listáját, mielőtt elfogadna egyet. A PEM kritikák megegyeznek abban, hogy az állandó ellenőrizgetés túl sok munkát jelent, ezért senki sem fog vesződni vele.

A 7.51. ábra felsorol néhány, a PGP és PEM közötti különbséget és hasonlóságot. Legtöbbjükről már volt szó, azonban egyes pontokhoz érdemes némi megjegyzést fűzni. A hitelességvizsgálat úgy tűnik, nagyobb szerepet kap a PEM-ben, hiszen ott kötelező, míg a PGP-nél választható. A PEM a hitelességvizsgálatra vonatkozó információt is a kódolt csomagon kívül szállítja, ami lehetővé teszi, hogy a hálózat is ellenőrizhesse az üzenetek eredetét. Ennek következményeképpen a hallgatózók, annak ellenére, hogy elolvasni nem tudják, azért feljegyezhetik, hogy ki kinek küld levelet.

A technikai különbségeket félretéve, meglepő különbség adódik a hozzájuk kapcsolódó kultúrában is. A PGP-nek, amely nem Internet szabvány, van Internetes kultúrája, míg a PEM-nek, ami egy hivatalos Internet szabvány, nincs. A PGP azon alapult, amit Dave Clark úgy nevez, hogy „nyers konszenzus és futó kód" („rough consensus and running code"). Valaki (Zimmermann) kitalált egy megoldás egy ismert problémára, rendesen megírta, és a forráskódot nyilvánosságra hozta, hogy mindenki használhassa. A PEM egy négyrészes nemzetközi szabványként kezdte, melynek részei a kinézetet definiáló ASN.1, a neveket definiáló X.400 és az igazolásokat definiáló X.509. Egy rideg, háromrétegű szervezeti hierarchiát használ a különböző féle igazoló hatóságokhoz, amelyekhez hivatalosan igazolt irányelvek tartoznak, és megköveteli, hogy mindenki megbízzon az IPRA-ban. A megvalósítások később jelentek meg és jóval elmaradtak a PGP-hez képest minőségben, mennyiségben, valamint abban, hogy nem állnak rendelkezésre a különböző felhasználói felületekhez. Röviden, a PGP úgy néz ki, mint egy tipikus Internet csomag, míg a PEM magán viseli az OSI szabványok tulajdonságait, amit az Internet népe nem szeret, de a PTT-k annál inkább. Ezt majd az olvasó is látni fogja.

7.5. USENET hírek

A számítógépes hálózatok egyik legnépszerűbb alkalmazása a hírcsoportok világméretű rendszere, amit **hálózati híreknek (net news)** neveznek. A hírekre gyakran **USENET** néven hivatkoznak, ami egy régi különálló UNIX-UNIX fizikai hálózatra utal vissza, amely egy **uucp** nevű programmal szállította a forgalmat. Manapság a forgalom nagy része az Interneten megy, de a USENET és az Internet nem ugyanaz. Némely Internetes gép nem kap híreket, van viszont olyan, amelyik anélkül kap, hogy rajta lenne az Interneten.

A következő részekben a USENET-et fogjuk bemutatni. Először megnézzük a felhasználó szemszögéből. Aztán megmutatjuk, hogyan valósították meg.

7.5.1. A USENET felhasználói szemszögből

A hírcsoport egy világméretű vitafórum valamely témában. Az ebben a témában érdekeltek „előfizethetnek" a hírcsoportra. Az előfizetők egy speciális felhasználói ügynökkel, a hírolvasóval, elolvashatják a cikkeket (üzeneteket), amelyek a hírcsoportba érkeznek. Az előfizetők küldhetnek is cikkeket a hírcsoportokba. Minden, a hírcsoportba küldött cikk automatikusan kézbesítődik minden előfizetőnek, akárhol is legyen a világon. A kézbesítés időtartama általában néhány másodperc és néhány óra között változik, attól függően, hogy a küldő és a fogadó milyen messze van a kitaposott úttól. A hírcsoport elvileg olyasmi, mint egy levelezési lista, de gyakorlatilag máshogy van megvalósítva. Úgy is felfogható, mint egy fejlett többpontos átvitel.

A hírcsoportok száma olyan nagy (valószínűleg több, mint 10 000), hogy a kezelhetőség kedvéért hierarchiába szervezik. A 7.52. ábra a „hivatalos" hierarchia legfelsőbb szintjét mutatja. Más hierarchiák is léteznek, de ezek általában regionális felhasználásra valók vagy nem angol nyelvűek. A többi hierarchia között van egy speciális, az *alt*. Az *alt* úgy viszonyul a hivatalosakhoz, mint egy piac az áruházhoz. Ez tulajdonképpen kaotikus, szabályozatlan összevisszasága a különböző témájú hírcsoportoknak, melyek közül néhány világszerte igen népszerű.

A *comp* csoportok voltak az eredeti USENET csoportok. Ezeket a csoportokat az informatikusok, a profi és hobbiszinten számítógéppel foglalkozók népesítik be. Mindegyik egy a számítógépekkel kapcsolatos témának van szentelve, legyen az szoftver vagy hardver.

A *sci* és a *humanities* csoportokat az olyan tudósok és amatőrök népesítik be, akik a fizika, kémia, biológia, Shakespeare stb. iránt érdeklődnek. Nem meglepő, hogy a *sci* hierarchia sokkal kiterjedtebb, mint a *humanities*, hiszen az azonnali elektronikus kommunikáció ötletét minden tudós kedveli, míg a humán beállítottságúak legalább is szkeptikusak a témával kapcsolatban. C. P. Snow-nak igaza volt.

A *news* hierarchia teljesen a hírrendszer körüli vitákkal és annak kezelésével foglalkozik. A rendszer adminisztrátorok itt kaphatnak segítséget, és itt vitatják meg az új hírcsoportok beiktatását is.

Név	Témák
Comp	Számítógépek, informatika és a számítógép ipar
Sci	A fizikai tudományok és műszaki tudományok
Humanities	Irodalom és humán tudományok
News	Magáról a USENET-ről folyó viták
Rec	Szabadidős tevékenységek, többek között sport és zene
Misc	Minden, ami máshova nem illeszkedik
Soc	Társadalom, és szociális kérdések
Talk	Támadó beszédek, hitviták, eszmecserék és viták mindenről
Alt	Alternatív fa, amely mindent magába foglal

7.52. ábra. *USENET hírcsoportok csökkenő jel-zaj viszony szerint*

Az eddig felsorolt hierarchiák professzionális vagy legalábbis akadémikus hangvételűek. Ezen változtat a *rec*, ami a szabadidős tevékenységekkel és hobbikkal foglalkozik. Azok azonban, akik ide írnak, szintén igen jól informáltak az érdeklődési körükben.

Ahogyan lefelé haladunk elérkezünk a *soc*-hoz, ami számos politikával, nemi témával, vallással, különböző népi szokásokkal és leszármazással foglalkozó hírcsoportot tartalmaz. A *talk*-ban számos vitatémát találhatunk, és azok népesítik be, akik inkább véleményükre támaszkodnak, mint a tudásukra. Az *alt* egy teljes alternatív fa, amely saját szabályai szerint működik.

Név	Téma
Comp.ai	Mesterséges intelligencia
Comp.database	Az adatbázisok tervezése és megvalósítása
Comp.lang.c	A C programozási nyelv
Comp.os.minix	A Tanenbaum-féle oktatási operációs rendszer, a MINIX
Comp.os.ms-windows.video	Mozgókép hardver és szoftver Windowshoz
Sci.bio.entomology.lepidoptera	Lepkékkel és molyokkal kapcsolatos kutatás
Sci.geo.earthquakes	Geológia, szeizmológia, és földrengések
Sci.med.orthopedics	Ortopéd sebészet
Humanities.lit.authors.shakespeare	Shakespeare színdarabjai és költészete
News.groups	Lehetséges új hírcsoportok
News.lists	USENET-tel kapcsolatos listák
Rec.arts.poems	Szabad költészet
Rec.food.chocolate	Nassolás
Rec.humor.funny	Hallotta a farmerről a viccet, aki
Rec.music.folk	A nép a népzenéről
Misc.jobs.offered	A szabad állásokról
Misc.health.diabetes	Mindennapi együttélés a diabétesszel
Soc.culture.estonia	Élet és kultúra Észtországban
Soc.singles	Egyedülálló emberek és érdeklődési körük
Soc.couples	A soc.singles-t végzettek
Talk.abortion	Semmi jel, csak zaj
Talk.rumors	Innen származnak a pletykák
Alt.alien.visitors	Itt kell jelenteni a repülő csészealjakat
Alt.bermuda.triangle	Ha ezt olvasod, különös módon el fogsz tűnni
Alt.sex.voyeurism	Nézd meg magad
Alt.tv.simpsons	Bart és a többiek

7.53. ábra. *A hírcsoportok egy kicsiny része*

Minden a 7.52. ábrán látható kategória rekurzívan alkategóriákra bomlik. Például a *rec.sports* a sportokkal foglalkozik, a *rec.sports.basketball* a kosárlabdával, a *rec.sports.basketball.women* pedig a női kosárlabdával. A 7.53. ábrán néhány példa látható a hírcsoportokra. Számos esetben a további hírcsoportokra a nyilvánvaló paraméterek lecserélésével lehet következtetni. Például a *comp.lang.c* a C programozási nyelvről szól, de a *.c* lecserélhető gyakorlatilag minden más programozási nyelvre, és ezzel megkapjuk az illető hírcsoportot.

Számos hírolvasó program létezik. Az e-levél olvasókhoz hasonlóan egyesek billentyűzet alapúak, mások pedig egér alapúak. Majdnem minden esetben a hírolvasó elindításkor egy fájlban utánanéz, hogy a felhasználó melyik hírcsoportokra van előfizetve. Aztán általában egysoros összefoglalók segítségével megjeleníti a még olvasatlan cikkeket az első hírcsoportban, és várja, hogy a felhasználó olvasás céljából kijelöljön egypárat. A kiválasztott cikkek ezután egyesével megjelennek. Elolvasás után le lehet őket törölni, el lehet menteni, ki lehet nyomtatni és így tovább.

A hírolvasók ezenkívül lehetővé teszik a felhasználó számára, hogy előfizessen vagy lemondjon hírcsoportokat. Az előfizetés megváltoztatása mindössze azon helyi fájl átszerkesztését igényli, amely azt tárolja, hogy a felhasználó melyik hírcsoportokra van előfizetve. Hogy párhuzamot találjunk, a hírcsoportokra való előfizetés olyan, mint egy televíziós adás figyelemmel kísérése. Ha valaki minden héten meg szeretne nézni egy műsort, akkor egyszerűen bekapcsolja a tv-t. Nem kell ezt valami központi hatóságnak bejelentenie.

A hírolvasók küldeni is képesek. A felhasználó megírja a cikket, majd parancsot ad vagy rákattint egy ikonra, és az üzenet már úton is van. Egy napon belül majdnem mindenkihez elér, aki az adott hírcsoportra előfizetett. Lehetőség van a **keresztülküldésre (crosspost)**, azaz arra, hogy valaki egy cikket egyszerre több hírcsoportnak küldjön egyetlen utasítással el. Lehetőség van továbbá a kézbesítés földrajzi korlátozására. Valószínűleg nem sok hasznát vennék Hong Kongban a keddi Kollokviumon a Stanford Egyetemen elhangzott bejelentésnek, ezért ennek elküldési határait Kaliforniára lehet csökkenteni.

A USENET szociológiája finoman fogalmazva egyedi. Ezelőtt még soha nem volt lehetőség rá, hogy egymást nem ismerő emberek ezrei világméretű vitákat folytassanak szerteágazó témákban. Például, most már elképzelhető, hogy valaki elküldje problémáját a hálózatra. Lehet, hogy a következő nap a küldő 18 megoldást kap problémájára, és egy kis szerencsével ebből csak 17 rossz.

Sajnos néhányan ezt az újonnan felfedezett hatalmukat arra használják, hogy emberek nagy csoportjával felelőtlenül kommunikáljanak. Ha valaki elküld egy levelet, amiben azt mondja: „A hozzád hasonlóakat le kellene lőni", akkor fellángolnak az indulatok, és általában gyalázkodó levelek tömege érkezik, amit **lángtengernek (flamewar)** hívnak.

Ez a helyzet egyedileg és kollektívan is megoldható. A felhasználók egyedileg telepíthetnek egy, törlendő leveleket tartalmazó, ún. **levélirtó fájlt (killfile)**, ami megadja, hogy egy bizonyos tárgyú vagy egy bizonyos személytől érkező leveleket megérkezés után még a megjelenítés előtt törölni kell.

A legtöbb hírolvasó lehetővé teszi az egyéni vitaszálak kiölését is. Ez a szolgáltatás jól jöhet, ha egy vita végtelen hurokba kezd kanyarodni.

Ha egy csoport kellő számú előfizetője kezdi megelégeli a hírcsoport szennyezését, akkor javasolhatják a hírcsoport felügyelet alá vételét. A **felügyelt hírcsoport (moderated newsgroup)** azt jelenti, hogy abba a hírcsoportba csak egy ember, a felügyelő, küldhet cikkeket. Minden a felügyelt hírcsoportra küldött levél automatikusan a felügyelőhöz kerül, aki a jókat elküldi, a rosszakat pedig eldobja. Egyes témákhoz tartozik mind felügyelt, mind felügyeletlen hírcsoport is.

Mivel a USENET-re naponta több ezren fizetnek elő olyanok, akik először látják, ugyanazok, a kezdőktől származó kérdések, újra és újra megfogalmazódnak. Annak érdekében, hogy az ilyen típusú forgalmat csökkentsék, számos hírcsoport létrehozott egy **FAQ (Frequently Asked Questions – gyakran feltett kérdések)** dokumentumot, amely megpróbálja megválaszolni a kezdő minden kérdését. Ezek közül néhány kiemelkedően megbízható és több mint 100 oldalas. A karbantartó általában egy- vagy kéthavonta szétküldi ezeket.

A USENET-en számos zsargon honos, mint pl. BTW (Ha már itt tartunk – By The Way), ROFL (Gurulok a röhögéstől – Rolling On the Floor Laughing) és IMHO (Szerény véleményem szerint – In My Humble Opinion). Számosan használnak kis ASCII szimbólumokat, amelyeket **tréfás közleményre utaló jelzéseknek (smileys)** vagy **érzelmet kifejező jelzéseknek (emoticons)** neveznek. Az érdekesebbek közül néhány a 7.54. ábrán látható. Ha a könyvet az óramutató járásával megegyezően 90 fokkal elforgatjuk, a legtöbbet azonnal megértjük. Egy 650 tréfás közleményre utaló jelzéseket tartalmazó könyv tekintetében lásd (Sanderson és Dougherty, 1993).

Annak ellenére, hogy a legtöbb ember a valódi nevét használja az üzenetekben, vannak néhányan, akik teljesen ismeretlenek szeretnének maradni, különösen akkor, amikor vitatott hírcsoportokra vagy társkereséssel kapcsolatban küldenek cikkeket. Ez vezetett az **ismeretlen továbbküldő (anonymous remailer)** szerverek létrehozásához, melyek olyan szerverek, amelyek fogadják a beérkező üzeneteket (akár cikkeket is), és megváltoztatják a *From:*, *Sender:* és *Reply-to:* mezőket úgy, hogy azok a továbbküldő szerverre mutassanak. A továbbküldők közül néhány számokat rendel a felhasználókhoz, és az ezekre a számokra érkezett e-leveleket is továbbküldik, így az embereknek lehetősége nyílik rá, hogy az ismeretlen üzenetekre is tudjanak válaszolni, valahogy így „SWF 25 keresi az SWM/DWM 20-30-at...". Nem valószínű, hogy ezek a továbbküldők akkor is meg tudják tartani titkukat, ha a helyi rendőrség meg akarja tudni egy felhasználó személyazonosságát (Barlow, 1995).

Jel	Jelentés	Jel	Jelentés	Jel	Jelentés
:-)	Boldog vagyok	=!:-)	Abe Lincoln	:+)	Nagy orr
:-(	Mérges/szomorú vagyok	=):-)	Sam nagybácsi	:-))	Toka
:-I	Egykedvű vagyok	*<:-)	Mikulás	:-{)	Bajusz
;-)	Kacsintok	<:-(	Tökfej	#:-)	Összekuszált haj
:-(o)	Kiabálok	(-:	Ausztráliai	8-)	Szemüveges
:-(*)	Hányok	:-)X	Csokornyakkendős ember	C:-)	Okostojás

7.54. ábra. *Néhány tréfás közleményre utaló jel*

Ahogyan egyre többen fizetnek elő a USENET-re, állandó kereslet van az új, és egyre szűkebb témájú hírcsoportok iránt. Következésképpen kitaláltak egy eljárást új hírcsoportok létrehozására. Tegyük fel, hogy valaki szereti a svábbogarakat, és más svábbogárrajongókkal szeretne beszélgetni. Elküld egy üzenetet a *news.groups*-ra, amelyben leírja a javasolt hírcsoport nevét, mondjuk *rec.animals.wildlife.cockroaches*, és azt, hogy miért olyan fontos (a svábbogarak lenyűgözők; 3500 fajtájuk létezik; lehetnek pirosak, sárgák, zöldek, barnák és feketék; már jóval az első dinoszaurusz előtt megjelentek a földön; valószínűleg ezek voltak az első repülő állatok és így tovább). Arról is nyilatkozik, hogy felügyelt vagy nem felügyelt legyen-e a hírcsoport.

Ezután következik a vita. Amikor elcsitul, egy e-leveles szavazás történik. A szavazatokban kiemelik, hogy ki és hogy hogyan vélekedik (a csalás elkerülése végett). Amennyiben az igenek száma 2:1 arányban meghaladja a nemek számát, és legalább 100-zal több igen érkezik, mint nem, akkor a *news.groups* felügyelője szétküld egy üzenetet, melyben elfogadja az új hírcsoportot. Ez az üzenet jelzés a világ minden rendszeradminisztrátorának, hogy a hírcsoportra áldását adta a legfőbb hatalom, és hogy az mostantól hivatalos.

Az új csoport létrehozása az *alt* hierarchiában kevésbé formális, ez valójában ezen ok miatt létezik egyáltalán. Az ott levő hírcsoportok némelyike olyan közel áll a hivatalosan és morálisan eltűrhető határhoz, hogy sohasem szavazták volna meg őket nyilvános szavazással. Gyakorlatilag az ezeket támogató emberek egyszerűen kikerülték az általános eljárást, és létrehozták saját hierarchiájukat. Bárhogy is van, az *alt*-ban levő számos hierarchia tökéletesen konvencionális.

7.5.2. A USENET megvalósítása

A kisebb hírcsoportok közül néhányat levelezési listaként valósítottak meg. Az ilyen hírcsoportokra úgy küldhet valaki cikket, hogy elküldi a levelezési lista címére, és minden listabeli cím kap belőle egy másolatot.

Ha azonban egy nagy egyetem hallgatóinak csak a fele előfizetne az *alt.sex*-re, akkor a szerverek összeroppannának a bejövő e-levelek tömegének súlya alatt. Következésképpen a USENET megvalósítása általában nem levelezési listákon alapul. Ehelyett a beérkező leveleket minden helyen (egyetemeken, cégeknél, Internet szolgáltatóknál) egyetlen könyvtárban tárolják, mondjuk a *news*-ban, amelynek alkönyvtárai vannak a *comp*, *sci* stb. részére. Ezeknek aztán vannak alkönyvtáraik, mint pl. *news/comp/os/minix*. Minden beérkező hír a neki megfelelő könyvtárban tárolódik. A hírolvasók egyszerűen, ha szükség van rá, innen hozzák őket el. Ez azt jelenti, hogy minden helyen csak egy példányban kell tárolni a cikkeket, függetlenül az előfizetők számától. Néhány nap múlva a cikkek lejárnak, és a rendszer letörli őket a lemezről.

A USENET-re való felkerüléshez szükség van egy másik USENET-en levő hírtároló helyre, ami **hírforrásként (newsfeed)** szolgál. Úgy képzelhetjük el, hogy ez egy hírtároló helyekből, mint csomópontokból, álló irányított gráf. A gráf csomópontjait összekötő élek az átviteli vonalak. Ez a gráf a USENET. Vegyük észre, hogy az Internetre való csatlakozás nem szükséges, de nem is elégséges feltétele USENET-re való csatlakozásnak.

Minden hírtároló hely, amely leveleket szeretne kapni, időnként megkérdezheti a hírszolgáltatóját (hírszolgáltatóit), hogy a legutolsó kapcsolatteremtés óta érkezett-e új hír. Amennyiben igen, akkor azokat elhozza, és a *news* megfelelő alkönyvtárában tárolja. Ily módon terjed a hír a hálózatban. Ugyanúgy elképzelhető, hogy inkább a hírforrás, mint a fogadó a kezdeményező, és ő teremt kapcsolatot, amennyiben elegendő mennyiségű új hír gyűlt össze. Eredetileg a hírtároló helyek kérdezgették a hírforrásukat, manapság azonban ez inkább a második módszert alkalmazzák.

Nem minden hírtároló hely kap meg minden hírcsoportot. Ennek számos oka van. Először is, a teljes hírcsomag meghaladja az 500 MB-ot naponta, és rohamosan növekszik. Ahhoz, hogy mindet tárolni lehessen, elég sok szabad lemezterület kellene. Másodszor, az átviteli idő és költség is számít. 28,8 kb/s sebességgel 39 órába telne 24 órai hírtermés átvitele, és ehhez egy fenntartott telefonvonal is szükséges. Még 56,4 kb/s-mal is egy 20 órán keresztül fenntartott vonalra van szükség ahhoz, hogy mindent el lehessen hozni. Sőt, a teljes méret akkorára nőtt, hogy létrehoztak műholdas hírforrásokat is.

Harmadszor, nem minden hírtároló helyet érdekel minden levél. Például nem valószínű, hogy a finn vállalatoknál dolgozó embereket tömegesen érdekelné a *rec.arts.magna* (a japán humoros könyvekről szóló hírcsoport). Végül néhány hírcsoport túlságosan pikáns néhány rendszeradminisztrátor ízlése szerint, akik aztán a letagadhatatlan helyi érdeklődés ellenére le is tiltják őket. 1995. decemberében a világméretű CompuServe hálózat (átmenetileg) beszüntette minden „szex" hírcsoport szállítását, mert egy kisebb német hivatalnok úgy gondolta, hogy ez jó módja a pornográfia megfékezésének. Az ezt követő felháborodás előre látható, hirtelen és igen hangos volt.

A hírcikkeknek ugyanaz a formátumuk, mint a 822-es RFC-beli leveleknek, csak néhány fejrész mezővel ki vannak bővítve. Ez a tulajdonság könnyen szállíthatóvá, és a legtöbb meglevő e-levelező rendszerrel kompatíbilissé teszi azokat. A hírekben használatos fejrész mezőket az 1036-os RFC definiálja. A 7.55. ábrán látható egy hírcikk példa.

```
From: Vogel@nyu.edu
Message-Id: <54731@nyu.edu>
Subject: Madárészlelés
Path: cs.vu.nl!sun4nl!EU.net!news.sprintlink.net!in2.uu.net!pc144.nyu.edu!news
Newsgroups: rec.birds
Followup-To: rec.birds
Distribution: world
Nntp-Posting-host: nuthatch.bio.nyu.edu
References:
Organization: New York University
Lines: 4
Summary: Képzeljétek, mit láttam!

Épp most láttam egy struccot a 52-edik utca és az Ötödik sugárút sarkán New York-
ban. Most van a vándorlási időszakuk? Látta más is?

Jay Vogel
```

7.55. ábra. *Példa egy hírcikkre*

Nem árt néhány szót szólni a hírcikkek fejrész mezőiről. A *Path:* fejrész mező azokat a csomópontokat tartalmazza, amelyeken a cikk keresztül halad a feladótól a címzettig. Minden ugrásnál a továbbító gép a lista elejére írja saját nevét. Ez a lista megadja a feladóhoz visszavezető utat. A felkiáltójelek használata (kiejtés szerint: bumm [bang]) a USENET címekből ered, amelyek megelőzték a DNS címeket.

A *Newsgroups:* fejrész mező megadja, hogy melyik hírcsoportokba tartozik a cikk. Egynél több hírcsoport nevét is tartalmazhatja. Minden több hírcsoportra írt cikk tartalmazza minden megfelelő hírcsoport nevét. Mivel ide több hírcsoport neve kerülhet, ezért a *Followup-To:* mező megadja, hogy a válaszokat melyik hírcsoportra kell küldeni, hogy az ezt követő vita egy hírcsoportban folytatódjon.

A *Distribution:* fejrész mező azt adja meg, hogy a cikk milyen mesze jusson. Tartalmazhat egy vagy több állam- vagy országkódot, egy meghatározott hálózati hely címét vagy azt, hogy „world" („világ").

Az *Nntp-Posting-Host:* fejrész mező jelentése megegyezik az 822-es RFC-beli *Sender:* mező jelentésével. Megadja, hogy ténylegesen melyik gép küldte a levelet, annak ellenére, hogy esetleg egy másikon írták meg (az NNTP a hírtovábbító protokoll, ami alább kerül ismertetésre).

A *References:* fejrész mező azt jelenti, hogy ez válasz egy korábbi cikkre, és megadja annak azonosító számát (ID). Minden válasz jellegű cikkben kötelező, de az új vita kezdetét jelentő cikkekben tilos használni.

Az *Organization:* fejrész mező arra szolgálhat, hogy megadja, hogy a küldő melyik céggel, egyetemmel vagy ügynökséggel áll kapcsolatban. Azoknak a cikkeknek a végén, amelyek ezt a mezőt használják, gyakran találhatók felelősségelhárító megjegyzések, melyek szerint az nem a szervezet hibája, ha a cikk kissé ostoba.

A *Lines:* fejrész mező megadja, hogy milyen hosszú a szövegrész. Ebbe nem számít bele a fejrész mező és a szövegrészt elválasztó üres sor.

A *Subject:* fejrész mezők összekötik a vita fonalait. Számos hírolvasóban vannak olyan parancsok, amelyek lehetővé teszik a felhasználó számára, hogy a következő beérkezett levél helyett inkább a következő, témához tartozó levélre ugorhasson. A törlendő cikkeket tartalmazó fájlok is ezeket a mezőket használják ahhoz, hogy megtudják, mit kell visszautasítani.

Végül a *Summary:* fejrész mező általában egy összefoglalást ad a cikk tartalmáról. A kapcsolódó cikkekben a *Subject:* fejrész mező egy „Re:"-t követően az eredeti tárgyat tartalmazza.

NNTP – Hírcsoport átviteli protokoll

Vizsgáljuk most meg, hogy hogyan is terjednek a cikkek a hálózatban. Az eredeti algoritmus egyszerűen elárasztotta a USENET minden vonalát cikkekkel. Ez séma ugyan működött, de végül is a forgalom mérete használhatatlanná tette, így valami jobbat kellett keresni.

Az **NNTP (Network News Transfer Protocol – hírcsoport átviteli protokoll)** nevű protokoll váltotta fel, amelyet a 977-es RFC definiál. Az NNTP hasonlít a SMTP-hez, azaz kliens ASCII parancsokat ad egy szervernek, ami erre ASCII kódolá-

Parancs	Jelentés
LIST	Adj egy listát az általad tárolt hírcsoportokról és cikkekről
NEWSGROUPS date time	Adj egy listát a dátum/idő után keletkezett hírcsoportokról
GROUP grp	Adj egy listát az összes cikkről a grp csoportban
NEWNEWS grps date time	Adj egy listát az egy bizonyos csoportban levő új cikkekről
ARTICLE id	Küldj el egy bizonyos cikket
POST	Van itt egy cikk, amit el akarok küldeni
IHAVE id	Van egy id azonosítóval ellátott cikkem. Kéred?
QUIT	A kapcsolat bontása

7.56. ábra. *A hírcikkek továbbítására szolgáló alapvető NNTP parancsok*

sú számokkal jelezett válaszokat ad. A legtöbb USENET gép manapság az NNTP-t használja.

Az NNTP-t két céllal tervezték. Az első az volt, hogy lehetővé váljon a hírcikkek biztonságos kapcsolaton (TCP) keresztül történő terjesztése. A második az volt, hogy lehetővé tegyék azon felhasználók számára a hírek távoli olvasását, akiknek a gépe nem volt képes híreket fogadni. Mindkét céllal elterjedten használják, azonban mi azzal fogunk inkább foglalkozni, hogy hogyan terjednek a hírcikkek a hálózatban az NNTP segítségével.

Ahogyan azt fentebb említettük két általános lehetőség van. Az első a hírszívás (news pull), azt jelenti, hogy a kliens felhívja az egyik hírforrását, és új hírcikkek után érdeklődik. A második a híretetés (news push), azt jelenti, hogy a hírforrás felhívja a klienst és bejelenti, hogy új hírcikkei vannak. Az NNTP parancsok mindkét lehetőséget támogatják, azon felül, hogy a távoli üzenetolvasást is lehetővé teszik.

Az új hírcikkek megszerzéséhez a kliensnek először létre kell hozni egy TCP kapcsolatot a hírforrás 119-es portjával. E mögött a port mögött van az NNTP daemon, ami vagy egyfolytában fut, vagy szükség esetén automatikusan elindul. A kapcsolat megteremtése után a kliens és a szerver parancsok és válaszok sorozatának segítségével kommunikál. Ezek a parancsok és válaszok arra szolgálnak, hogy a kliens megkapja az összes általa igényelt hírcikket, de egyiket se hozza el kétszer, akárhány hírforrást is használ. A főbb, hírcikkek mozgatására szolgáló parancsok a 7.56. ábrán láthatók.

A *LIST* és a *NEWGROUPS* parancsok lehetővé teszik a kliens számára, hogy megtudja, hogy a hírforrás melyik csoportokat tárolja. Az előbbi a teljes listát megadja. Az utóbbi csak a megadott dátum és idő után keletkezetteket adja meg. Ha a kliens tudja, hogy a lista hosszú, akkor hatékonyabb, ha hírforrásonként számon tartja a csoportokat, és mindig csak a frissítések iránt érdeklődik. A válasz mindkét parancsra egy ASCII lista, amely soronként egy hírcsoportot tartalmaz, és minden hírcsoporthoz megadja a szerveren levő utolsó cikk számát, a szerveren levő első cikk számát, és egy jelzőt, amely megadja, hogy az adott hírcsoportra lehet-e cikket küldeni.


```
   S: 200 feeder.com NNTP server at your service (válasz a hír kapcsoltra)
C: NEWNEWS soc.couples 960901 030000 (van új hír a soc.couples-en?)
   S: 230 List of 2 articles follows
   S: <13281@psyc.berkeley.edu>      (az soc.couples-en az 1. üzenet a 2-ből a
                                      Berkeleyről jött)
   S: <162721@aol.com>               (az soc.couples-en az 2. üzenet a 2-ből az
                                      AOL-ról jött)
   S:                                (lista vége)
C: ARTICLE <13281@psyc.berkeley.edu> (kérem a Berkeleyről való cikket)
   S: 220 <13281@psyc.berkeley.edu> follows
   S: (itt az egész <13281@psyc.berkeley.edu> cikk átvitelre kerül)
   S:                                (cikk vége)
C: ARTICLE <162721@aol.com>          (kérem az AOL-ról való cikket)
   S: 220 <162721@aol.com> follows
   S: (itt az egész <162721@aol.com> cikk átvitelre kerül)
   S:                                (cikk vége)
C: NEWNEWS misc.kids 960901 030000 (van új cikk a misc.kids-en?)
   S: 230 List of 1 article follows
   S: <43222@bio.rice.edu>           (1 cikk a Rice-ról)
   S:                                (lista vége)
C: ARTICLE <43222@bio.rice.edu>      (kérem a Rice-ról való cikket)
   S: 220 <43222@bio.rice.edu> follows
   S: (itt az egész <43222@bio.rice.edu> cikk átvitelre kerül)
   S:                                (cikk vége)
C: NEWGROUPS 960901 030000
   S: 231 2 new groups follow
   S: rec.pets
   S: rec.nude
   S: .
C: NEWNEWS re.pets 0 0               (mutasd meg az összes cikket)
   S: 230 List of 1 article follows
   S: <124@fido.net>                 (1 cikk a fido.net-ről)
   S:                                (lista vége)
C: ARTICLE <124@fido.net>            (kérem a fido.net-ről való cikket)
   S: 220 <124@fido.net> follows
   S: (itt az egész cikk átvitelre kerül)
   S: .
C: POST
   S: 340                            (küldheted)
C: (article posted on wholesome.com sent here)
   S: 240                            (a cikk átjött)
C: IHAVE <5321@foo.com>
   S: 435                            (már megvan, ne küldd)
C: QUIT
   S: 205                            (Viszlát)
```

7.57. ábra. *Hogyan gyűjtheti be a cikkeket hírforrásától a wholesome.com*

Miután a kliens kiderítette, hogy a szerver mely hírcsoportokat tárolja, elkezdheti a cikkek után való kérdezősködést (pl. régi hírcsoportok esetében a *NEWGROUPS* használata után). Erre a célra a *GROUP* és *NEWNEWS* parancsokat szokás használni. Ugyanúgy, mint korábban, az előbbi itt is a teljes listát eredményezi, míg az utóbbi csak a megadott dátum és idő, általában az utolsó, e hírforrással létesített kapcsolat időpontja óta történt frissítésekről számol be. Az első paraméter az összes csoportra való utalásként tartalmazhat csillag jelet. Például a *comp.os.** az összes *comp.os*-el kezdődő hírcsoportot jelenti.

Miután a kliens összeállította a teljes listáját arra vonatkozóan, hogy melyik cikk melyik csoportban található (vagy még a teljes lista összeállítása előtt), megkezdheti azoknak a cikkeknek az átvitelét az *ARTICLE* paranccsal, amelyekre szüksége van. Miután minden szükséges cikket megszerzett, a kliens felajánlhatja a más hírforrásokból az *IHAVE* parancs segítségével szerzett cikkeket, valamint a helyileg a *POST* parancs segítségével feladott cikkeket. A szerver tetszés szerint elfogadhatja vagy visszautasíthatja ezeket. Amikor a kliens elkészült, felbonthatja a kapcsolatot a *QUIT* paranccsal. Ily módon minden egyes gép maga irányíthatja, hogy melyik cikket melyik hírforrástól szerzi meg, és így elkerülheti, hogy egy cikk két példányban szerepeljen.

Az NNTP működésére példaképpen képzeljünk el egy információszolgáltatót, a *wholesome.net*-et, aki mindenáron el akarja kerülni a vitákat, így csupán a *soc.couples* és a *misc.kids* hírcsoportokat ajánlja fel. A vezetés azonban nyílt szemléletű, és más olyan hírcsoportokat is hajlandó tárolni, amelyek valószínűleg senkit nem botránkoztatnak meg. Ezért informálódni szeretnének az új hírcsoportok felől, hogy ügyfeleik érdekében megalapozott döntéseket hozhassanak. A 7.57. ábrán látható egy lehetséges forgatókönyv a kliens *(wholesome.com)* és a hírforrásként működő szerver *(feeder.com)* párbeszédéről. Ez a forgatókönyv a hírszívás módszerét használja (azaz a kliens kezdeményezi a kapcsolatot és ő tudakozódik a cikkek után). A zárójelekben levő megjegyzések nem részei az NNTP protokollnak.

Ebben a kapcsolatban a *wholesome.com* először aziránt tudakozódik, hogy van-e új hír-cikk a *soc.couples*-ben. Mikor megtudja, hogy kettő van, mindegyiket elhozza, és *news/soc/couples*-ban külön fájlként tárolja azokat. Minden fájl a benne levő cikk számáról van elnevezve. Ezután a *wholesome.com* a *misc.kids* felől érdeklődik, és megtudja, hogy ott egy új cikk van. Elhozza azt, és *news/misc/kids* könyvtárba teszi.

Miután minden cikket megszerzett az általa tárolt hírcsoportokhoz, megnézi, hogy keletkezett-e új hírcsoport, és megtudja, hogy kettő is létrejött. Az egyik ígéretesnek tűnik, így elhozza az oda tartozó leveleket. A másik ijesztően néz ki, ezért onnan nem hozza el a leveleket. (A *wholesome.com* sokat költött az AI szoftverekre, hogy képes legyen ránézésre csupán a nevekből kideríteni, hogy mit érdemes elhozni és mit nem.)

Miután minden szükséges cikket megszerzett, a *wholesome.com* felajánl a *feeder.com*-nak egy új üzenetet, amit valaki náluk adott fel. A cikket elfogadja a *feeder.com*, és az átvitelre is kerül. Ezután a *wholesome.com* felajánl egy újabb cikket, amit a másik hírforrásától szerzett. A *feeder.com* visszautasítja ezt, mivel már rendelkezik a cikk egy példányával. Végül a *wholesome.com* szétbontja a kapcsolatot és a TCP összeköttetést is.

A híretető módszer hasonló. Azzal kezdődik, hogy a hírforrás felhívja a fogadásra váró gépet. A hírforrás általában számon tartja, hogy az ügyfelei mely hírcsoportokra

vannak előfizetve, és azzal kezdi, hogy bejelenti az első ilyen hírcsoportban levő első új cikket az *IHAVE* parancs segítségével. A lehetséges fogadó ekkor belenéz a helyi táblázataiba, hogy lássa, megvan-e már a kérdéses cikk, és ennek megfelelően elfogadhatja vagy visszautasíthatja azt. Ha elfogadja, akkor a cikk átvitelre kerül, és egyetlen pontot tartalmazó sor követi azt. Ezután a hírforrás bejelenti a következő cikket és így tovább, amíg minden hírcikk átvitelre nem került.

A probléma mind a hírszívásos, mind a híretetős módszerrel az, hogy a résztvevők mindkettőben időnként megállnak, és válaszra várakoznak. Általában 100 ms idő telik el egy válaszra való várakozás során. Ez napi 100 000 vagy több cikk esetén jelentős többletidőt tesz ki.

7.6. A Világháló (World Wide Web)

A Világháló egy keretszerkezet az Interneten gépek ezrein elszórva elhelyezkedő, összekapcsolt dokumentumok eléréséhez. 5 év alatt nagy energiájú fizikai adatok közlésére szolgáló módszerből olyan alkalmazássá vált, amelyre az emberek milliói mint „az Internet"-re gondolnak. Hihetetlen népszerűsége onnan ered, hogy kezdők által is könnyen használható grafikus interfésze van, és hatalmas információmennyiséget biztosít majdnem minden elképzelhető témáról, – az abakusztól a zoológiáig.

A Háló (amelyet **WWW**-ként is ismernek) története 1989-ben kezdődött a CERN-ben, a nukleáris kutatás európai központjában. A CERN-nek sok részecskegyorsítója van, amelyeknél a résztvevő európai országok tudósainak nagy csoportjai végeznek részecskefizikai kutatásokat. Ezen csapatok tagjai gyakran fél tucat vagy még több különböző országból valók. A legtöbb kísérlet nagyon összetett, és a berendezés megépítését sokévi előzetes tervezés előzi meg. A Háló abból az igényből nőtt ki, hogy a különböző országokban elszórva elhelyezkedő kutatók jelentések, tervek, rajzok, fényképek és más dokumentumok folyton változó gyűjteményét használva működhessenek együtt.

Az összekapcsolt dokumentumok hálózatának eredeti javaslata egy CERN fizikustól, Tim Berners-Leetől eredt 1989 márciusában. Az első (szöveg alapú) prototípus 18 hónappal később kezdte meg működését. 1991 decemberében nyilvános bemutatót tartottak a San Antonió-i Hypertext '91 konferencián Texasban. A következő év során folytatódott a fejlesztés, amely 1993 februárjában érte el tetőfokát az első grafikus interfész, a Mosaic kibocsátásával (Vetter és mások, 1994).

A Mosaic olyan népszerű volt, hogy a szerzője, Marc Andreessen elhagyta a Szuperszámítógépek Alkalmazásának Nemzeti Központját (National Center for Supercomputing Applications), ahol a Mosaicot kifejlesztették, és megalakított egy társaságot, a Netscape Communications Corp.-ot, amelynek a célja ügyfelek, kiszolgálók, és más Háló-szoftverek fejlesztése volt. Amikor a Netscape-et 1995-ben tőzsdére vitték, a befektetők 1,5 milliárd dollárt fizettek a részvényekért, nyilván azt gondolva, hogy ez lesz a következő Microsoft. Ez a rekord annál is meglepőbb, mivel a társaságnak csak egy terméke volt, nagy veszteséggel működött, és az ismertetőjében bejelentette, hogy belátható időn belül nem számít profitra.

1994-ben a CERN és az M.I.T. aláírtak egy megegyezést a Világháló (World Wide Web) Konzorcium felállításáról. A szervezet célja a Háló továbbfejlesztése, a protokollok szabványosítása és az állomások közti együttműködés elősegítése. Berners-Lee lett az igazgató. Azóta egyetemek és társaságok százai csatlakoztak a konzorciumhoz. Az M.I.T. működteti a konzorcium amerikai részét, az INRIA franciaországi kutatóközpont pedig az európai részét. Bár több könyv van a Hálóról, mint égen a csillag, naprakész információkat (természetesen) legjobban magán a Hálón lehet találni. A konzorcium honlapja a *http://www.w3.org* címen található. Az érdeklődő olvasók itt a konzorcium minden tevékenységét és dokumentumát lefedő oldalakhoz találnak élőkapcsokat.

A következő szakaszokban részletesen leírjuk, hogyan jelenik meg a Háló a felhasználó számára, és különösen, hogy hogyan működik belül. Mivel a Háló alapvetően egy ügyfélkiszolgáló (client-server) alapú rendszer, mind az ügyfél (vagyis a felhasználó), mind a kiszolgáló oldalát tárgyalni fogjuk. Ezek után megvizsgáljuk a nyelvet, amelyen a Háló-oldalak íródnak (HTML és Java). Végül azt vesszük szemügyre, hogy hogyan találhatunk meg valamely információt a Hálón.

7.6.1. Az ügyfél oldala

A felhasználók szemszögéből a Háló-dokumentumok hatalmas, világméretű gyűjteményéből áll, amelyeket röviden csak **oldalaknak (pages)** neveznek. Minden oldal tartalmazhat élőkapcsokat (mutatókat) más, vele összefüggő oldalakhoz, amelyek bárhol a világon elhelyezkedhetnek. A felhasználók (azáltal, hogy rájuk kattintanak), követhetik az élőkapcsokat, amelyek a hivatkozott oldalakra viszik őket. Ezt a folyamatot végtelen sokszor ismételhetik, és ezalatt esetleg sok száz összekapcsolt oldalon haladhatnak keresztül. Az olyan oldalakat, amelyek más oldalakra mutatnak, **hipertextnek (hypertext)** nevezzük.

Az oldalakat egy **böngészőnek (browser)** nevezett programmal tekinthetjük meg. Két népszerű böngésző a Mosaic és a Netscape. A böngésző elhozza a kívánt oldalt, értelmezi a szöveget és az abban előforduló formázóparancsokat, majd megfelelően formázva megjeleníti az oldalt a képernyőn. Egy példát láthatunk a 7.58.(a) ábrán. Mint sok Háló-oldal, ez is egy címmel kezdődik, némi információt tartalmaz, és az oldal karbantartójának e-levél címével végződik. Az olyan – **élőkapocsnak (hyperlink)** nevezett – szövegrészeket, amelyek összeköttetést biztosítanak más oldalakhoz, aláhúzással, különleges színnel való megjelenítéssel, vagy mindkettővel emelik ki. Egy élőkapocs követéséhez a felhasználó a kurzort a kiemelt területre viszi (az egeret vagy a kurzormozgató billentyűket használva), és kiválasztja (az egér gombjának lenyomásával vagy az ENTER leütésével). Bár léteznek nem grafikus böngészők is, mint például a Lynx, ezek nem annyira népszerűek, mint a grafikus böngészők, így az utóbbiakra fogunk összpontosítani. Hanggal vezérelhető böngészők is fejlesztés alatt állnak.

Azon felhasználók, akiket érdekel az Állatpszichológia Tanszék, többet is megtudhatnak róla, ha rákattintanak annak (aláhúzott) nevére. Ekkor a böngésző elhozza azt az oldalt, amelyre ez a név mutat, és megjeleníti, ahogy a 7.58.(b) ábrán is látszik. Itt az aláhúzott egységekre szintén rá lehet kattintani, hogy más oldalakat hozzunk be és

így tovább. Az új oldal lehet ugyanazon a gépen, mint az előző, vagy egy olyan gépen, amely a földgolyó másik felén helyezkedik el. A felhasználó ezt nem tudja. Az oldalak elhozását a böngésző végzi, a felhasználó bármilyen segítsége nélkül. Ha a felhasználó valaha visszatér a főoldalra, a már egyszer követett élőkapcsok esetleg szaggatott aláhúzással (és valószínűleg más színnel) jelennek meg, hogy megkülönböztethetők legyenek a még nem követett élőkapcsoktól. Vegyük észre, hogy a főoldalon az *Egyetemi Információ* sorra kattintva nem történik semmi. Nincs aláhúzva, ami azt jelenti, hogy ez csak szöveg és nem mutat másik oldalra.

A legtöbb böngésző számos gombot és szolgáltatást tartalmaz a webes navigáció

ÜDVÖZÖLJÜK AZ EAST PODUNK EGYETEM WWW HONLAPJÁN!

- Egyetemi információ
  - Felvételi tudnivalók
  - Az egyetem térképe
  - Útbaigazítás az egyetemhez
  - Az East Podunk Egyetem hallgatói állománya
- Tanszékek
  - Állatpszichológiai Tanszék
  - Alternatív Tanulmányok Tanszék
  - Mikrobiotikai Főzés Tanszék
  - Nem Hagyományos Tanulmányok Tanszék
  - Hagyományos Tanulmányok Tanszék

Webmaster@eastpodunk.edu

(a)

ÁLLATPSZICHOLÓGIAI TANSZÉK

- Tudnivalók a leendő fő szakirányosoknak
- Személyzet
  - A Tanszék tagjai
  - Végzett hallgatóink
  - Tanszéki dolgozók
- Kutatási projektek
- Álláslehetőségek
- A legnépszerűbb tanfolyamaink
  - A növényevők kezelése
  - Lómenedzsment
  - Hogyan tárgyaljunk a háziállatunkkal?
  - Felhasználóbarát kutyaházépítés
- Az összes tanfolyam listája

Webmaster@animalpsyc.eastpodunk.edu

(b)

7.58. ábra. *(a) Egy Háló-oldal. (b) Az Állatpszichológiai Tanszék-re kattintva elért oldal*

megkönnyítésére. Soknak van gombja az előző oldalra visszalépéshez, a következő oldalra előrelépéshez (amely csak azután működik, miután a felhasználó már arról visszalépett), és közvetlenül a felhasználó honlapjára lépéshez. A legtöbb böngészőnek van egy gombja vagy menüparancsa, amely egy adott oldalra könyvjelzőt helyez el, és egy másik is, amely a könyvjelzők listáját jeleníti meg, így biztosítva, hogy bármelyiket egy egérkattintással újra fel lehessen keresni. Az oldalakat lemezre is lehet menteni, vagy ki lehet nyomtatni. Általában számos lehetőség áll rendelkezésre a képernyő elrendezésének szabályozására és a felhasználói beállítások megadására. Kilenc böngésző összehasonlítását írja le (Berghel, 1996).

Rendes (nem aláhúzott) szövegen és (aláhúzott) hipertexten kívül a Háló-oldalak tartalmazhatnak ikonokat, vonalas rajzokat, térképeket, és fényképeket is, amelyek mutathatnak egy másik oldalra. Ezen elemek közül az egyikre kattintva, a böngésző elhozza a hivatkozott oldalt, és megjeleníti, ugyanúgy, mintha szövegre kattintottunk volna. A fényképekhez és térképekhez hasonló ábráknál a következő elhozott oldal attól is függhet, hogy az ábra melyik részére kattintottunk.

Nem minden oldal tekinthető meg a hagyományos módon. Néhány oldal például hangállományokat és/vagy videoklipeket is tartalmaz. Amikor a hipertextes oldalakat más médiafajtával keverik, az eredményt **hipermédiának (hypermedia)** nevezik. Néhány böngésző minden fajta hipermédiát meg tud jeleníteni, míg mások ilyenre nem képesek. Ehelyett ezek egy konfigurációs állományban néznek utána, hogyan kell a beérkezett adatot kezelni. Rendesen a konfigurációs állomány egy program nevét adja meg, amelyet a bejövő oldallal mint bemenettel le kell futtatni. Ezeket **külső megjelenítőnek (external viewer)** vagy **kisegítő alkalmazásnak (helper application)** nevezik. Ha nincs megjelenítő beállítva, a böngésző rendszerint megkéri a felhasználót, hogy válasszon egyet. Ha nem létezik megjelenítő, a felhasználó utasíthatja a böngészőt, hogy mentse a beérkező oldalt lemezre, vagy dobja el azt. A beszédet előállító kisegítő alkalmazások még a vak felhasználók számára is elérhetővé teszik a Hálót. Más kisegítő alkalmazások speciális Háló-nyelvekhez tartalmaznak értelmezőket, ezáltal lehetőség nyílik Háló-oldalakról programokat letölteni és futtatni. Ez a mechanizmus lehetővé teszi, hogy kibővítsük a Háló funkcionalitását.

Sok Háló-oldal nagy ábrákat tartalmaz, amelyek hosszú idő alatt töltődnek be. Például, egy tömörítetlen 640×480 (VGA), 24 bit/pixeles ábra körülbelül 4 percet vesz igénybe egy 28,8 kb/s-os modemes vonalon. Néhány böngésző úgy kezeli az ábrák lassú betöltését, hogy először a szöveget hozza el és jeleníti meg, majd a képeket hozza el. Ez a stratégia ad valami olvasnivalót a felhasználónak, amíg az ábrák jönnek, és lehetővé teszi, hogy a felhasználó megszüntesse a letöltést, ha az oldal nem elég érdekes ahhoz, hogy ellensúlyozza a várakozást. Egy másik stratégia egy olyan opció biztosítása, amellyel letiltható az ábrák automatikus letöltése és megjelenítése.

Néhány oldal szerzője úgy kísérli meg az esetlegesen unatkozó felhasználókat megbékíteni, hogy az ábrákat egy különleges módon jeleníti meg. Először az ábra gyorsan megjelenik durva felbontásban. Ezután a részleteket fokozatosan helyettesíti be. A felhasználó számára gyakran előnyösebb az, ha pár másodperc után látja az egész ábrát, még ha kis felbontásban is, mint ha azt látja, hogy az lassan épül fel a tetejétől, sorról sorra.

Néhány Háló-oldal űrlapokat tartalmaz, amelybe a felhasználónak kell beírnia az információt. Ezen űrlapok tipikus alkalmazásai: a felhasználó által megadott tétel kikeresése az adatbázisból, egy termék megrendelése, vagy közvélemény-kutatásban való részvétel. Más Háló-lapok térképeket tartalmaznak, amelyekre a felhasználók rákattinthatnak, hogy kinagyítsák azokat, vagy információt kapjanak valamely földrajzi területről. Az űrlapok és aktív térképek (amelyekre rá lehet kattintani) kezelése kifinomultabb feldolgozást igényel, mint egyszerűen elhozni egy ismert oldalt. Később látni fogjuk, hogyan valósítják meg ezeket a szolgáltatásokat.

Néhány böngésző a helyi lemezt használja az elhozott oldalak gyorstárazására. Mielőtt egy oldalt elhozna, ellenőrzi, hogy az jelen van-e a helyi gyorstárban. Amennyiben igen, akkor csak azt szükséges ellenőrizni, hogy az oldal változott-e. Ha nem, az oldalt nem kell még egyszer letölteni. Ennek eredményeképpen a VISSZA gombra kattintás, és az előző oldal megtekintése rendszerint nagyon gyors.

Egy gépnek közvetlenül az Interneten kell lennie ahhoz, hogy egy Háló-böngészőt futtasson, vagy legalább egy SLIP vagy PPP összeköttetéssel kell rendelkeznie egy routerhez vagy más géphez, amely közvetlenül az Internetre kapcsolódik. Ez a követelmény azért szükséges, mert a böngésző úgy hoz el egy oldalt, hogy létrehoz egy TCP összeköttetést az oldalt tartalmazó géphez, és ezután egy üzenetet küld át az összeköttetésen, az oldalt kérve. Egyetlen böngésző sem működik, ha nem tud egy tetszőleges géphez TCP összeköttetést felépíteni az Interneten.

Néha meglepő, mennyit megtesznek az emberek, hogy hozzáférjenek a Hálóhoz. Például egy társaság Web-by-Fax szolgáltatást kínál. Egy Internet-elérés nélküli ügyfél felhívja a Web-by-Fax szervert, és a telefon billentyűzetét használva bejelentkezik. Ezután bebillentyűz egy, a kért Háló-oldalt azonosító kódot, és ezt elfaxolják a hívó faxkészülékére.

7.6.2. A kiszolgáló oldala (szerver)

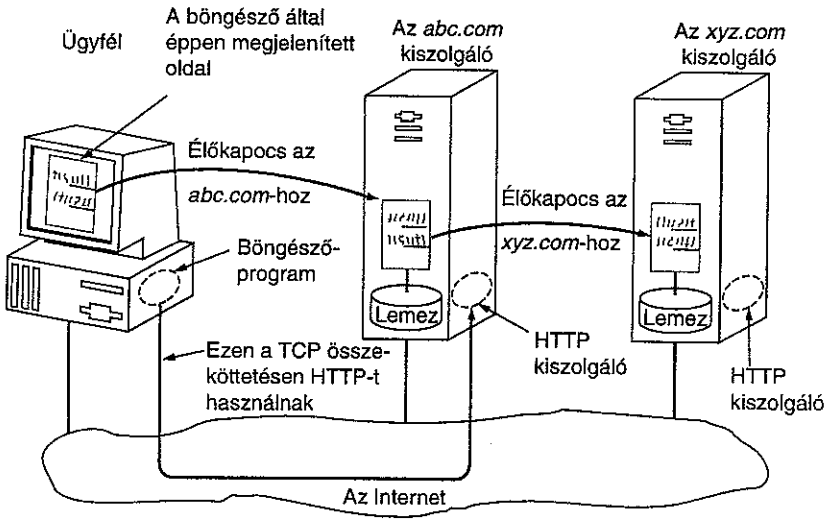
Minden Háló-állomásnak (Web site) van egy kiszolgáló folyamata, amely a 80-as TCP porton figyeli az ügyfelektől (amelyek rendszerint böngészők) beérkező hívásokat. Miután egy összeköttetés felépült, az ügyfél egy kérést és a kiszolgáló egy választ küld. Ezután a kapcsolatot lebontják. Azt a protokollt, amely az érvényes kéréseket és válaszokat határozza meg, HTTP-nek hívják. Valamilyen részletességgel később tanulmányozni fogjuk, de egy ezt használó egyszerű példa ésszerű elképzeléssel szolgálhat arról, hogyan is működnek a Háló-kiszolgálók. A 7.59. ábrán látható, hogyan illeszkednek a Háló-modell különböző részei egymáshoz.

Ehhez a példához képzeljük el, hogy a felhasználó éppen most kattintott rá valamely szövegrészletre, vagy esetleg egy ikonra, amely arra az oldalra mutat, amelynek a neve (illetve URL-je, Uniform Resource Locator, egységes erőforrás-meghatározó): *http://www.w3.org/hypertext/WWW/TheProject.html*. Az URL-eket ebben a fejezetben később tárgyaljuk. Most elég annyit tudni, hogy egy URL-nek három része van: a protokoll neve (*http*), a gép neve, ahol az oldal elhelyezkedik (*www.w3.org*), és annak az állománynak a neve, amely tartalmazza az oldalt (*hypertext/WWW/TheProject.html*). A felhasználó kattintása és az oldal megjelenítése közti lépések a következők:

1. A böngésző meghatározza az URL-t (látván, hogy mit választottak ki).
2. A böngésző megkérdezi a DNS-től a *www.w3.org* IP címét.
3. A DNS 18.23.0.23-mal válaszol.
4. A böngésző létrehoz egy TCP összeköttetést a 18.23.0.23 80-as portjával.
5. Ezután elküld egy *GET /hypertext/WWW/TheProject.html* parancsot.
6. A *www.w3.org* kiszolgáló elküldi a *TheProject.html* állományt.
7. A TCP kapcsolatot lebontják.
8. A böngésző megjeleníti a *TheProject.html*-ben levő összes szöveget.
9. A böngésző elhozza és megjeleníti a *TheProject.html*-ben levő összes ábrát.

Sok böngésző a képernyő alján egy állapotsorban megjeleníti, hogy pillanatnyilag éppen melyik lépést hajtja végre. Ily módon amikor a teljesítmény gyatra, a felhasználó láthatja, hogy ez azért van, mert a DNS nem válaszol, a kiszolgáló nem válaszol, vagy egyszerűen az oldal átvitele közben torlódás lépett fel a hálózaton.

Érdemes megfigyelni, hogy egy oldal minden beágyazott ábrájához (ikon, rajz, fénykép stb.) a böngésző egy új TCP összeköttetést hoz létre a megfelelő kiszolgáló-



7.59. ábra. *A Háló-modell részei*

hoz, hogy elhozza az ábrát. Szükségtelen mondani, hogy ha egy oldal sok ikont tartalmaz, és mindegyik ugyanazon a kiszolgálón helyezkedik el, akkor mindegyikhez egy új összeköttetés létrehozása, használata, és lebontása nem valami hatékony, de a megvalósítást egyszerűvé teszi. A protokoll jövőbeni módosításai foglalkozni fognak a hatékonysági kérdéssel. Egy javaslatot ad (Mogul, 1995).

```
C: telnet www.w3.org 80
T: Trying 18.23.0.23 ...
T: Connected to www.w3.org.
T: Escape character is '^]'
C: GET /hypertext/WWW/TheProject.html HTTP/1.0
  S: HTTP/1.0 200 Document follows
  S: MIME-Version: 1.0
  S: Server: CERN/3.0
  S: Content-Typoe: text/html
  S: Content-Length 8247
  S:
  S: <HEAD> <TITLE> The World Wide Web Consortium (W3C) </TITLE> < /HEAD>
  S: <BODY>
  S: <H1> <IMG ALIGN=MIDDLE ALT="W3C" SRC="Icons/WWW/w3c_96x67.gif">
  S: The World Wide Web Consortium </H1> <P>
  S:
  S: The World Wide Web is the universe of network-accessible information.
  S: The <A HREF="Consortium/"> World Wide Web Consortium </A>
  S: exists to realize the full potential of the web <P>
  S:
  S: W3C works with the global community to produce
  S: <A HREF="#Specifications> specifications </A> and
  S: <A HREF="#Reference"> reference software </A> .
  S: W3C is funded by industrial
  S: <A HREF="Consortium/Member/List.html"> members </A>
  S: but its products are freely available to all. <P>
  S:
  S: In this document:
  S: <menu>
  S: <LI> <A HREF="#Specifications"> Web Specifications and Development Areas </A>
  S: <LI> <A HREF="#Reference"> Web Software </A>
  S: <LI> <A HREF="#Community"> The World Wide Web and the Web Community </A>
  S: <LI> <A HREF="#Joining"> Getting involved with the W3C </A>
  S: </menu>
  S: <P> <HR>
  S: <P> W3C is hosted by the
  S: <A HREF="http://www.lcs.mit.edu/"> Laboratory for Computer Science </A> at
  S: <A HREF="http://web.mit.edu/"> MIT </A> and
  S: in Europe by <A HREF="http://www.inria.fr/"> INRIA </A> .
  S: </BODY>
```

7.60. ábra. *Egy példa-forgatókönyv egy Háló-oldal megszerzéséhez*

Mivel a HTTP, az SMTP-hez hasonlóan, egy ASCII protokoll, így egészen könnyű egy terminál előtt ülő személynek (a böngésző helyett) közvetlenül beszélni a Hálókiszolgálókkal. Mindössze egy összeköttetésre van szükség a kiszolgáló 80-as portján. Egy ilyen összeköttetés megszerzésének legegyszerűbb módja a Telnet program használata. A 7.60. ábrán egy forgatókönyv látható arról, hogyan lehet ezt véghezvinni. Ebben a példában a *C:*-vel jelölt sorokat a felhasználó (ügyfél) gépeli, a *T:*-vel jelölt sorokat a Telnet program állítja elő, és az *S:*-sel jelölt sorokat az MIT-n levő kiszolgáló állítja elő.

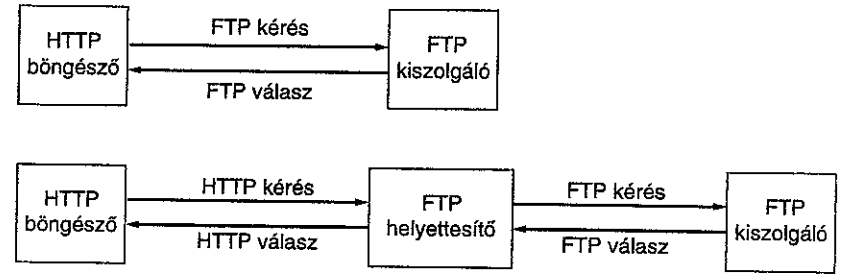
Az olvasókat arra bátorítjuk, hogy próbálják ki személyesen is ezt a forgatókönyvet (lehetőleg egy UNIX gépről, mivel néhány más rendszer nem adja vissza a kapcsolat állapotát). Mindenképpen vegyék figyelembe a *GET* sorában a szóközöket és a protokoll verzióját, valamint a *GET* sora után következő üres sort. Mellékesen az a szöveg, amelyet valójában kapni fognak, három ok miatt különbözni fog a 7.60. ábrán bemutatottól. Először is, mert az itt megadott példa kimenet rövidített és szerkesztett azért, hogy elférjen egy oldalra. Másodszor, mert némileg ki is van tisztítva, hogy a szerző ne jöjjön zavarba, aki kétségkívül arra ugyan számított, hogy a formázott oldalt emberek ezrei fogják megvizsgálni, de arra nem, hogy az azt előállító HTML-t valaki is elemezni fogja. Harmadszor, mert az oldal tartalmát állandóan módosítják. Mindezek ellenére ez a példa ésszerű elképzelést adhat a HTML működéséről.

Amit a példa mutat, az a következő. Az ügyfél, ebben az esetben egy személy, de rendesen egy böngésző, először összeköttetésbe lép egy bizonyos hoszttal, majd egy parancsot küld, amellyel egy meghatározott oldalt kér, és meghatároz egy adott használandó protokollt és használandó verziót (HTTP/1.0). A 7. sorban, a kiszolgáló egy állapotsorral válaszol, amely megadja az általa használt protokollt (amely ugyanaz, mint az ügyfélé), és a 200-as kódot, amely azt jelenti, minden rendben van. Ezt a sort egy RFC 822 MIME üzenet követi, amely fejrészéből az ábrán öt sor látható (számos továbbit helytakarékosság céljából kihagytunk). Ezután jön egy üres sor, amelyet az üzenet törzse követ. Egy kép küldésekor a *Content-Type* (tartalom típusa) mező a következő lehet:

```
Content-Type: Image/GIF
```

Ily módon a MIME típusok lehetővé teszik tetszőleges objektumok szabványos módon való elküldését. Ennek mellékhatásaként a MIME *Content-Transfer-Encoding* (a tartalom átviteli kódolása) mezőre nincs szükség, mert a TCP lehetővé teszi tetszőleges bájtfolyamok vagy akár képek küldését is módosítás nélkül. A példában használt, csúcsos zárójelek (<>) közé zárt parancsok jelentéséről e fejezetben később lesz szó.

Nem minden kiszolgáló beszéli a HTTP-t. Különösen sok régebbi kiszolgáló az FTP, Gopher vagy más protokollokat használja. Mivel nagy mennyiségű hasznos információ érhető el FTP és Gopher kiszolgálókon, a Háló egyik tervezési célja az volt, hogy ezeket az információkat a Háló felhasználóinak is elérhetővé tegyék. Egy megoldás az, hogy a böngészőnek ezeket a protokollokat kelljen használnia, amikor egy FTP vagy Gopher kiszolgálóval beszél. Tulajdonképpen némelyikük ezt a megoldást használja, de ha a böngészőknek minden protokollt meg kell érteniük, akkor szükségtelenül nagyok lesznek.



7.61. ábra. *(a) Egy böngésző, amely megérti az FTP-t. (b) Egy böngésző, amely nem*

Ehelyett gyakran egy másik megoldást használnak: a **helyettesítő kiszolgálókat (proxy server).** A helyettesítő kiszolgáló egyfajta átjáró, amely a böngésző felé HTTP, de a kiszolgáló felé FTP, Gopher vagy valamilyen más protokollt használ. Elfogadja a HTTP kéréseket, és lefordítja azokat mondjuk FTP kérésekké, így a böngészőnek nem kell megértenie a HTTP-n kívül semmilyen más protokollt. A helyettesítő kiszolgáló egy olyan program lehet, amely ugyanazon a gépen fut, mint a böngésző, de az is lehet, hogy egy külön gépen van valahol a hálózaton, és sok böngészőt szolgál ki. A 7.61. ábrán látható a különbség két böngésző közt, amelyek közül az egyik megérti az FTP-t, a másik pedig helyettesítő kiszolgálót használ.

Gyakran a felhasználók állíthatnak be a böngészőhöz helyettesítőket a böngésző által nem beszélt protokollokhoz. Ily módon megnövekszik az információforrások azon tartománya, amelyeket a böngésző el tud érni.

Azon kívül, hogy átjáróként szolgálnak ismeretlen protokolloknak, a helyettesítő kiszolgálóknak számos más fontos feladatuk is van, mint amilyen a gyorstárazás. Egy gyorstárazó helyettesítő kiszolgáló összegyűjti és megtartja mindazokat az oldalakat, amelyek rajta keresztülhaladtak. Amikor egy felhasználó egy oldalt kér, a helyettesítő kiszolgáló leellenőrzi, hogy megvan-e neki ez az oldal. Ha igen, megnézheti, hogy még mindig friss-e az oldal. Abban az esetben, ha az oldal még mindig friss, akkor továbbadja a felhasználónak, egyébként egy új másolatot hoz el.

Végül, egy szervezet elhelyezhet egy helyettesítő kiszolgálót a tűzfalán belülre, lehetővé téve a felhasználóknak, hogy elérjék a Hálót, de teljes Internet-elérés biztosítása nélkül. Ebben az elrendezésben a felhasználók a helyettesítő kiszolgálóval beszélnek, de a helyettesítő kiszolgáló lép kapcsolatba a távoli állomásokkal és hozza el az oldalakat a felhasználói számára. Ezt a mechanizmust például arra lehet használni, hogy a középiskolák letiltsák az olyan Háló-helyek elérését, amelyeket az igazgató a zsenge fiatal fejek számára nemkívánatosnak tart.

(Katz és mások, 1994; Kwan és mások, 1995) információt adnak a Háló egyik legnépszerűbb kiszolgálójáról (az NCSA HTTP démonjáról) és annak teljesítményéről.

HTTP – Hipertext átviteli protokoll

A Háló szabványos átviteli protokollja a **HTTP (HyperText Transfer Protocol – hipertext átviteli protokoll).** Minden interakció egy ASCII kérésből, és egy ezt követő RFC 822 MIME-szerű válaszból áll. Bár a szállítási rétegbeli összeköttetéshez a TCP használata nagymértékben elterjedt, a szabvány ezt formálisan nem követeli meg. Ha az ATM hálózatok elég megbízhatóak lesznek, a HTTP kéréseket és válaszokat akár AAL 5 üzenetekben is lehet szállítani.

A HTTP folyamatosan fejlődik. Számos változatot használnak, mások pedig fejlesztés alatt állnak. Az alább olvasható anyag viszonylag alapismereteket tartalmaz és valószínűtlen, hogy változzon, de néhány részlet a későbbi verziókban más lehet.

A HTTP protokoll két jól elkülöníthető részből áll: a böngészők által a kiszolgálókhoz intézett kérések és a másik irányban visszafelé haladó válaszok halmazából. Mindkettőt sorban szemügyre fogjuk venni.

A HTTP minden újabb verziója kétfajta kérést támogat: egyszerű és teljes kéréseket. Egy egyszerű kérés egyetlen *GET*-ből áll, amely a kívánt oldalt nevezi meg, a protokoll verziója nélkül. A válasz maga a nyers oldal, fejrészek, MIME és kódolás nélkül. Hogy lássák, hogy ez hogyan működik, próbáljanak meg létrehozni egy Telnet összeköttetést a *www.w3.org* 80-as portjával (ahogy a 7.60. ábra első sorában látható), majd gépeljék be a következő sort:

```
GET /hypertext/WWW/TheProject.html
```

ezúttal a HTTP/1.0 nélkül. Vissza fogják kapni az oldalt, a tartalom típusának megjelölése nélkül. Erre a mechanizmusra a visszairányú kompatibilitás miatt van szükség. Használata alább fog hagyni, ahogy a teljes kérésekre alapozott böngészők és kiszolgálók jobban elterjednek.

A teljes kéréseket arról lehet felismerni, hogy a protokoll verziószáma jelen van a *GET* sorában, amint a 7.60. ábrán is. A kérések több sorból is állhatnak, amelyeket egy üres sor követ, ezért volt szükség a 7.60. ábrán az üres sorra. Egy teljes kérés első sora tartalmazza a parancsot (amelyek közül a *GET* csak egy lehetőség), a kívánt oldalt, és a protokollt/verziót. A további sorok RFC 822 fejrészeket tartalmaznak.

Eljárás	Leírás
GET	Kérés egy Háló-oldal olvasására
HEAD	Kérés egy Háló-oldal fejrészének olvasására
PUT	Kérés egy Háló-oldal eltárolására
POST	Egy megnevezett erőforráshoz (Háló-oldalhoz) való hozzáfűzés
DELETE	A Háló-oldal eltávolítása
LINK	Két meglevő erőforrást kapcsol össze
UNLINK	Két erőforrás közt meglevő kapcsolatot bont le

7.62. ábra. *A beépített HTTP kérés-eljárások*

Bár a HTTP-t a Hálón való használatra tervezték, szándékosan a szükségesnél általánosabbra készítették, szem előtt tartva az eljövendő objektumorientált alkalmazásokat. Ennek okáért egy teljes kérés első sorának első szava egyszerűen az az **eljárás (method),** más néven parancs, amelyet le kell futtatni a Háló-oldalon (vagy egy általános objektumon). A beépített eljárásokat a 7.62. ábra sorolja fel. Amikor általános objektumokhoz kell hozzáférni, az objektumra jellemző eljárások is elérhetők lehetnek. A nevek érzékenyek a kisbetű/nagybetű különbségre, így a *GET* egy legális eljárás, de a *get* nem.

A *GET* eljárás arra kéri a kiszolgálót, hogy küldje el az oldalt (amely alatt a legáltalánosabb esetben objektumot értünk), MIME-ben megfelelően kódolva. Viszont, ha a *GET* kérést egy *Ha-módosították-az-alábbi-dátum-óta* (*If-modified-since*) fejrész követi, akkor a kiszolgáló csak akkor küldi el az adatot, ha a megadott dátum óta azt módosították. Ezt a mechanizmust használva egy böngésző, amelytől egy, a gyorstárban levő oldal megjelenítését kérik, ezt az oldalt feltételesen kérheti azt a kiszolgálótól, megadva az oldalhoz rendelt módosítási időt. Ha a gyorstárban szereplő oldal még mindig érvényes, a kiszolgáló csak egy állapotsort küld vissza, amely ezt a tényt jelenti be, így elkerülve az oldal ismételt átvitelével előálló többletterhet.

A *HEAD* eljárás csak az üzenet fejrészét kéri, a tulajdonképpeni oldal nélkül. Ez az eljárás használható arra, hogy megszerezzük egy oldalnak az utolsó módosítási idejét, hogy információt gyűjtsünk indexelési célokra, vagy egyszerűen leellenőrizzük egy URL érvényességét. Feltételes *HEAD* kérések nem léteznek.

A *PUT* eljárás a *GET* fordítottja: az oldal olvasása helyett írja azt. Ez az eljárás lehetővé teszi Háló-oldalak gyűjteményének felépítését távoli kiszolgálókon. A kérés törzse tartalmazza az oldalt. Ez lehet MIME használatával kódolt, amikor is a *PUT*-ot követő sorok tartalmazhatják a *Tartalom típusa* és hitelesítési fejrészeket, annak bizonyítására, hogy a hívónak valóban van engedélye a kívánt feladat végrehajtására.

A *PUT*-hoz némileg hasonló a *POST* eljárás. Ez is egy URL-t hordoz, de a meglevő adat felülírása helyett az új adatot „hozzáadja" ahhoz valamilyen általános értelemben. Ennek a parancsnak az alkalmazására példa egy üzenet postázása egy hírcsoportba, vagy egy állomány hozzáadása egy hirdetőtábla rendszerhez (BBS). Itt nyilván az volt a szándék, hogy a Háló átvegye a USENET hírrendszer feladatkörét.

A *DELETE* azt végzi, amit várhatunk: eltávolítja az oldalt. Itt is, mint a *PUT*-nál, nagy szerepet játszik a hitelesítés és az engedélyezés. Nincs arra garancia, hogy a *DELETE* sikerülni fog, mert még ha a távoli HTTP kiszolgáló le is akarja törölni az oldalt, az azt tartalmazó állománynak olyan attribútuma lehet, amely megtiltja a HTTP kiszolgáló számára a módosítást vagy eltávolítást.

A *LINK* és *UNLINK* eljárásokkal létező oldalak vagy más erőforrások közt hozhatunk létre kapcsolatokat.

Minden kérésre érkezik egy válasz, amely egy állapotsorból, és esetleg további információkból (mint egy Háló-oldal része vagy egésze) áll. Az állapotsor hordozhatja a 200-as (OK) kódot, vagy bármelyiket a sokfajta hibakód közül, például a 304 (nem módosított), 400 (rossz kérés), vagy 403 (tiltott) kódokat.

A HTTP szabvány meglehetősen részletesen leírja az üzenetek fejrészét és törzsét. Legyen elég annyi, hogy nagyon hasonlatosak az RFC 822 MIME üzenetekhez, így itt nem tárgyaljuk azokat.

7.6.3. Egy Háló-oldal megírása HTML-ben

A Háló-oldalakat egy **HTML (Hypertext Markup Language – hipertext jelölő nyelv)** nevű nyelven írják. A HTML lehetővé teszi, hogy a felhasználók olyan oldalakat készítsenek amelyek szöveget, grafikát, valamint más Háló-oldalakra mutatókat tartalmaznak. A HTML tanulmányozását ezekkel a mutatókkal kezdjük, mivel ez az a kötőanyag, amely egybetartja a Hálót.

Az URL-ek: egységes erőforrás-meghatározók

Már többször említettük, hogy a Háló-oldalak tartalmazhatnak mutatókat más Háló-oldalakra. Most ideje, hogy megnézzük, hogyan is valósítják meg ezeket a mutatókat. Amikor a Háló először létrejött, rögtön nyilvánvalóvá vált, hogy ha az egyik Háló-oldal a másikra akar mutatni, akkor az oldalak elnevezéséhez és megtalálásához mechanizmusok kellenek. Egészen pontosan, három kérdés volt, amelyekre válaszolni kellett, mielőtt egy kiválasztott oldalt meg lehetett volna jeleníteni:

1. Mi az oldal neve?

2. Hol helyezkedik el az oldal?

3. Hogyan lehet elérni az oldalt?

Ha minden oldalhoz valahogyan hozzárendelnénk egy címet, az oldalak azonosításánál nem lenne kétértelműség. Ennek ellenére a probléma nem oldódna meg. Az Egyesült Államokban majdnem mindenkinek van egy társadalombiztosítási száma, amely egy egyedi azonosító, mivel nincs két ember, akinek ugyanaz a száma. Viszont ha csak a társadalombiztosítási számot ismerjük, akkor nem tudjuk megtalálni a tulajdonosának lakcímét, és nyilván nem fogjuk tudni, hogy az illetőnek angolul, spanyolul vagy kínaiul kellene írnunk. A Háló alapvetően hasonló gondokkal küzd.

A választott megoldás úgy azonosítja az oldalakat, hogy mindhárom gondot egyszerre oldja meg. Minden oldalhoz hozzárendelnek egy **URL-t (Uniform Resource Locator – egységes erőforrás-meghatározó)**, amely tulajdonképpen az adott oldalnak a világon mindenütt használható neveként szolgál. Az URL-ek három részből állnak: a protokollból (amelyet sémának is neveznek), annak a gépnek a DNS nevéből, amelyen az oldal elhelyezkedik, és egy helyi névből, amely egyértelműen azonosítja a kérdéses oldalt. (Ez rendszerint egy állománynév azon a gépen, ahol az oldal elhelyezkedik.) Például a szerző tanszékének URL-je:

http://www.cs.vu.nl/welcome.html

Ez az URL három részből áll: a protokollból (*http*), a hoszt DNS nevéből (*www.cs.vu.nl*) és az állomány nevéből (*welcome.html*), amelyeket bizonyos írásjelek választanak el.

Sok állomás bír bizonyos beépített állománynév-rövidítésekkel. Például a *~user/*-t leképezhetik a *user* felhasználó WWW könyvtárára, azzal a szokásos feltételezéssel, hogy a könyvtárnév önmagában egy bizonyos állományt jelent abban, mondjuk az *index.html*-t. Így a szerző honlapja a következő címen érhető el:

http://www.cs.vu.nl/~ast/

bár a valóságban más az állomány neve. Sok állomásnál az üres állománynév alapértelmezésben a szervezet honlapjára mutat.

Most már világos kell legyen, hogyan is működik a hipertext. Ahhoz, hogy egy szövegrészre rá lehessen kattintani, az oldal írójának két tudnivalót kell biztosítani: a megjelenítendő szöveget, amire rá lehet kattintani, és azt az URL-t, ahova menni kell, ha a szöveget kiválasztják. Amikor a szöveget kiválasztják, a böngésző a DNS-t használva kikeresi a hoszt nevét. A hoszt IP címének ismeretében a böngésző létrehoz azzal egy TCP összeköttetést. Ezen az összeköttetésen keresztül elküldi az állomány nevét, a megadott protokolllt használva. Nyertünk. Jön vissza az oldal. Ez pontosan az, amit a 7.60. ábrán láttunk.

Ez az URL séma nyitott abban az értelemben, hogy egyszerűen lehet más, nem HTTP protokollt használni. Mi több, más elterjedt protokollokhoz is definiáltak URL-eket, és sok böngésző meg is érti azokat. Az elterjedtebb protokollok némileg egyszerűsített formája látható a 7.63. ábrán.

Menjünk végig röviden a listán. A *http* protokoll a Háló anyanyelve, amelyet a HTTP kiszolgálók beszélnek. A 7.62. ábra valamennyi eljárását támogatja a szükséges, objektumra jellemző eljárások mellett.

Az *ftp* protokollt használva lehet az FTP-vel, az Internet-állomány átviteli protokolljával állományokat elérni. Az FTP már több mint két évtizede létezik, és már beépült a köztudatba. A világon mindenhol vannak olyan FTP kiszolgálók, amelyre az Interneten keresztül bárki bejelentkezhet, és az FTP kiszolgálóra felhelyezett állományokat letöltheti. A Háló ezt nem változtatja meg, mindössze csak az állományok FTP-vel történő megszerzését könnyíti meg, mivel az FTP-nek némileg fapados interfésze van. Idővel az FTP valószínűleg el fog tűnni, mivel nincs különösebb előnye an-

Név	Használata	Példa
http	Hipertext (HTML)	http://www.cs.vu.nl/~ast
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Helyi állomány	/usr/suzanne/prog.c
news	Hírcsoport	news:comp.os.minix
news	Hírcsoport cikke	news:AA0134223112@cs.utah.edu
gopher	Gopher	gopher://gopher.tc.umn.edu/11/Libraries
mailto	E-levél küldése	mailto:kim@acm.org
telnet	Távoli bejelentkezés	telnet://www.w3.org:80

7.63. ábra. *Néhány szokásos URL*

nak, hogy egy állomás FTP kiszolgálót futtasson HTTP kiszolgáló helyett, amely mindent tud, amit az FTP kiszolgáló, és még többet is (bár vannak viták a hatékonyságról).

A *file* protokollt használva, vagy egyszerűbben azt megnevezve lehet egy helyi állományt egy Háló-oldalt elérni. Ez a megközelítés hasonló az FTP-hez, de nem követeli meg egy kiszolgáló meglétét. Természetesen ez csak helyi állományokra működik.

A *news* protokoll lehetővé teszi a Háló-kiszolgáló számára, hogy úgy hívjon le egy cikket a hírek közül, mintha az egy Háló-oldal lenne. Ez azt jelenti, hogy egy Háló-böngésző egyben hírolvasó is, sőt, sok böngészőnek van gombja vagy menüparancsa arra, hogy a USENET hírek olvasását még könnyebbé tegye, mint egy rendes hírolvasó.

A *news* protokollnak két formátumát támogatják. Az első formátum egy hírcsoportot nevez meg, és arra lehet használni, hogy egy előre beállított hírállomásról megkapjuk a cikkek listáját. A másodikhoz szükség van egy bizonyos cikk azonosítójának megadásához, ebben az esetben az *AA0134223112@cs.utah.edu*-ra. Ezután a böngésző elhozza a megadott cikket az előre beállított hírállomásról, az NNTP protokollt használva.

A *gopher* protokollt a Gopher rendszer használja, amelyet a Minnesotai Egyetemen terveztek, és az iskola atlétikai csapata, a Golden Gophers (Arany Hörcsögök) után neveztek el. (A „gopher" az angol szlengben a „go for" szinonimája is, ami kb. annyit tesz: menj és hozd el.) A Gopher, amely sok évvel megelőzte a Hálót, egy információbegyűjtő rendszer, amely elvében hasonló a Hálóhoz, de csak szöveget támogat, képek nélkül. Amikor egy felhasználó bejelentkezik egy Gopher kiszolgálóra, egy állományokból és könyvtárakból álló menü tárul eléje, amelyek bármelyike a világ bármely Gopher menüjéhez kapcsolódhat.

A Gopher nagy előnye a Hálóval szemben, hogy nagyon jól működik a még mindig szép számmal előforduló 25×80-as ASCII terminálokkal, és mivel a Gopher szövegalapú, nagyon gyors. Ennek következtében a világon több ezer Gopher kiszolgáló van. A *gopher* protokollt használva a Háló felhasználói elérhetik a Gophert, és minden Gopher menü mint egy Háló-oldal jelenik meg, amelyre rá lehet kattintani. Ha nem ismeri a Gophert, próbálja ki a 7.63. ábrán megadott példát, vagy kerestessen a kedvenc keresőjével a „gopher" szóra.

Bár a fenti példa nem mutatja, de lehetséges egy Gopher kiszolgálónak egy teljes lekérdezést is elküldeni, a *gopher+* protokollt használva. Ami megjelenik, az a távoli Gopher kiszolgáló lekérdezésének eredménye.

Az utolsó két protokoll nem igazán hasonlít a Háló-oldalak elhozására, és nem is minden böngésző támogatja ezeket, de azért hasznosak. A *mailto* protokoll lehetővé teszi, hogy a felhasználók egy Háló-böngészőből küldhessenek levelet. Ezt úgy lehet véghezvinni, hogy az OPEN gombra kattintunk, és egy olyan URL-t adunk meg, amely egy *mailto:*-ból és a címzett ezt követő e-levél címéből áll. A legtöbb böngésző erre úgy reagál, hogy egy űrlapot dob fel a képernyőre, amelyen rubrikák vannak a tárgy és a többi fejrészsor számára, továbbá hely van az üzenet begépeléséhez is.

A *telnet* protokoll arra használható, hogy egy távoli géppel on-line összeköttetést hozzunk létre. Ugyanúgy működik, mint a Telnet program, ami nem meglepő, mivel a legtöbb böngésző csak meghívja a Telnet programot, mint kisegítő alkalmazást. Gyakorlatképpen próbálja ki a 7.60. ábra forgatókönyvét újból, de ezúttal egy Háló-böngésző használatával.

Röviden, az URL-eket úgy tervezték, hogy az a felhasználóknak ne csak a Hálós navigációt, de az FTP, a Gopher, a hírek, az e-levél és a telnet használatát is lehetővé tegye. Ezáltal szükségtelenné válnak a többi szolgáltatás speciális felhasználói interfészű programjai, így majdnem minden Internet-elérés egyetlen programba, a Háló-böngészőbe integrálódik. Ha nem tudnánk, hogy ezt a sémát egy fizikus kutató tervezte, könnyen úgy vélhetnénk, hogy ez valamely szoftvercég hirdetési kampányának terméke.

Mindezen kellemes tulajdonságok ellenére, a Háló egyre növekvő használatával felmerült az URL séma egy öröklött gyengesége. Egy URL egy bizonyos hosztra mutat. Az olyan oldalakról, amelyekre sokan hivatkoznak, kívánatos sok másolatot létrehozni egymástól távol, hogy a hálózati terhelés csökkenjen. A gond csak az, hogy az URL nem ad semmi módot arra, hogy egy oldalra hivatkozzunk anélkül, hogy megmondanánk, hol is van az. Nem lehet azt mondani: „ Az xyz oldalt akarom, de nem érdekel, honnan szerzed meg." Ennek a problémának a megoldására, és az oldalak többszörözésének lehetővé tétele céljából az IETF **URI-k (Uniform Resource Identifier – általános erőforrás-azonosító)** egy rendszerén dolgozik. Egy URI-t úgy foghatunk fel, mint egy általánosított URL-t. Ez a tárgykör sok mostani kutatás témája.

Bár itt csak az abszolút URL-eket tárgyaltuk, léteznek relatív URL-ek is. Hasonló a különbség, mint a */usr/ast/foobar* abszolút állománynév és az egyszerű *foobar* közt, ha a szövegkörnyezet egyértelművé teszi az utóbbit.

HTML – Hipertext jelölőnyelv

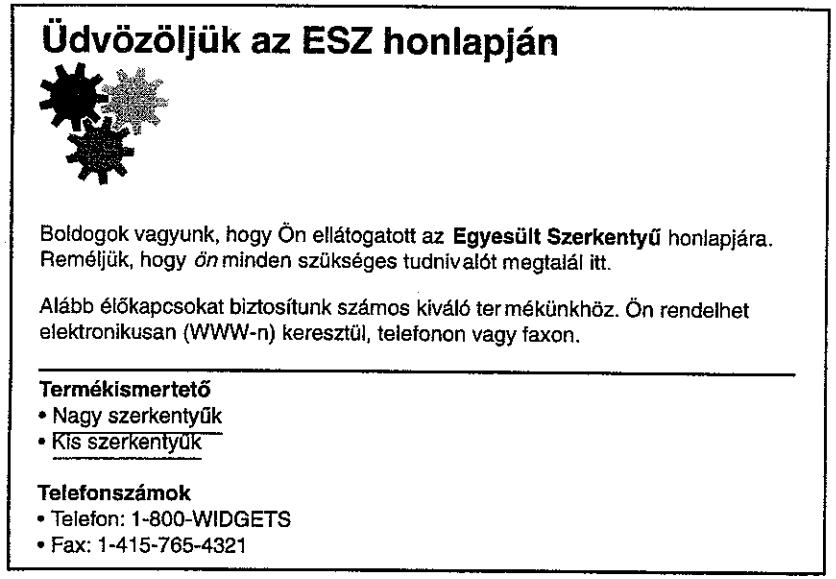
Most, hogy már értjük, hogyan működnek az URL-ek, ideje szemügyre vennünk magát a HTML-t. A HTML a 8879-es ISO szabvány, az **SGML (Standard Generalized Markup Language – szabványos általánosított jelölő nyelv)** alkalmazása, a hipertextre kihegyezve és a Hálóhoz alakítva.

Ahogy korábban említettük, a HTML egy jelölőnyelv, azaz egy olyan nyelv, amely leírja, miként kell a dokumentumot megformázni. A „jelölő" (markup) kifejezés a régebbi időkből származik, amikor a szerkesztők valóban megjelölték a dokumentumokat, hogy megmondják a nyomtatónak, amely akkoriban egy emberi lény volt, hogy mely betűkészleteket használja és így tovább. Ezért a jelölőnyelvek explicit formázóparancsokat tartalmaznak. Például a HTML-ben a <B> a félkövér mód kezdetét jelzi, a </B> pedig a végét. A jelölőnyelv használatának előnye a nem jelölő nyelvekkel szemben, hogy egyszerű hozzá böngészőt írni: a böngészőnek egyszerűen csak a jelölő parancsokat kell megértenie. A jelölőnyelvek további jól ismert példái a TeX és a troff.

A jelölőnyelven írt dokumentumokat szembeállíthatjuk a WYSIWYG (What You See Is What You Get) szövegszerkesztőkkel – mint pl. a MS-Word vagy a WordPerfect – előállított dokumentumokkal. Ezek a rendszerek az eltárolt állományaikba rejtett jelöléseket ágyazhatnak be, hogy később azokat reprodukálni tudják, de nem mindegyik így működik. Például a Macintoshon futó szövegszerkesztők a formázási információkat külön adatstruktúrákban tárolják, nem pedig a felhasználói állományokba beágyazott parancsként.

```
<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
élőkapcsokat biztosítunk számos kiváló termékünkhöz. Ön rendelhet elektronikusan
(WWW-n) keresztül, telefonon vagy faxon. <HR>
<H2> Termékismertető </H2>
<UL> <LI> <A HREF="http://widget.com/products/big"> Nagy szerkentyűk </A>
     <LI> <A HREF="http://widget.com/products/little"> Kis szerkentyűk </A>
</UL>
<H2> Telefonszámok </H2>
<UL> <LI> Telefon: 1-800-WIDGETS
     <LI> Fax: 1-415-765-4321
</UL> </BODY> </HTML>
```

(a)



(b)

7.64. ábra. *(a) Egy példa Háló-oldal HTML-je. (b) A formázott oldal*

Azáltal, hogy a jelölőparancsokat szabványosították, és beágyazták minden HTML állományba, bármelyik Háló-böngésző számára lehetővé vált, hogy bármely Háló-oldalt elolvashassa és újraformázhassa. A Háló-oldalak letöltés utáni újraformázásának képessége igen fontos, mert attól, hogy egy oldalt teljes képernyőn hoztak létre 24 bites színmélység és 1024×768-as felbontás mellett, még lehet, hogy egy 64×480 felbontású, 8 bites színmélységű képernyőn kell megjeleníteni egy kis ablakban. A saját WYSIWYG szövegszerkesztőnket nem használhatjuk a Hálón, mert a belső jelölőnyelveik (már amelyiknek van) más cégnél, más gépen és operációs rendszeren nem számítanak szabványnak. A különböző ablakméretek és felbontások miatt szükséges újraformázásokat sem kezelik. Viszont egyes szövegszerkesztők felkínálják és meg is csinálják azt a lehetőséget, hogy a dokumentumokat a gyártó saját formátuma helyett HTML-be mentik el.

A HTTP-hez hasonlóan, a HTML is állandóan változik. Amikor a Mosaic volt az egyetlen böngésző, akkor az általa értelmezett nyelv, a HTML 1.0 volt a de facto szabvány. Amikor új böngészők jöttek, szükség lett egy formális Internet szabványra, így megalkották a HTML 2.0 szabványt. A HTML 3.0 eredetileg azon kutatási törekvés következtében jött létre, hogy a HTML 2.0-hoz sok új képességet adjanak hozzá, mint például táblázatok, eszköztárak, matematikai formulák, továbbfejlesztett stíluslapok (az oldal elrendezésének és a szimbólumok jelentésének meghatározásához) stb.

A HTML hivatalos szabványosítását a WWW Konzorcium kezeli, de a különböző böngészőgyártók hozzárakták a maguk külön kiterjesztéseit. Ezek a gyártók azt remélik, hogy ezzel ráveszik az embereket arra, hogy az ő kiterjesztéseiket használva írják meg a Háló-oldalaikat, így ezen oldalak olvasói kénytelenek lesznek a gyártó böngészőjét használni, hogy helyesen tudják értelmezni azokat. Ez a tendencia nem könnyíti meg a HTML szabványosítását.

Alább egy rövid bevezetőt adunk a HTML-hez, csak egy általános kép kialakítása céljából. Bár természetesen bármely szokásos szövegszerkesztővel lehetséges HTML dokumentumokat írni, és sok ember így is tesz, de lehetőség van különleges HTML szerkesztők használatára is, amelyek elvégzik a munka nagy részét (de cserébe kevesebb beavatkozási lehetőséget adnak a felhasználónak a végeredmény részleteire nézve).

Egy szabályos Háló-oldal egy fejrészből és egy törzsből áll, amelyeket a <HTML> és </HTML> **címkék (tag),** vagy más néven formázóutasítások fognak közre, bár a legtöbb böngésző nem panaszkodik, ha ezek a címkék hiányoznak. Ahogy a 7.64. ábrán látszik, a fejrészt a <HEAD> és a </HEAD> címkék, a törzset pedig a <BODY> és </BODY> címkék fogják közre. Ezeket a parancsokat **direktíváknak (directives)** nevezik. A legtöbb HTML címkének ilyen a formátuma, azaz a <VALAMI> valami kezdetét, a </VALAMI> pedig annak végét jelzi. Könnyen elérhető számos más HTML példa is. A legtöbb böngészőnek van egy WIEV SOURCE (FORRÁS MEGJELENÍTÉSE) vagy hasonló menüparancsa. Ezt kiválasztva az aktuális oldal formázott képe helyett annak HTML forrása jelenik meg.

A címkéket nagy- és kisbetűvel is lehet írni. Ezért aztán a <HEAD> és a <head> ugyanazt jelenti, de az előbbi jobban kiemelkedik a szövegből, ha ember olvassa azt. A HTML dokumentum tulajdonképpeni elrendezése lényegtelen. A HTML értelmezők figyelmen kívül hagyják a fölös szóközöket és kocsi-vissza jeleket, mivel úgyis újra kell formázniuk a szöveget, hogy az beférjen az éppen aktuális megjelenítési területre. Ezért aztán ilyen fenti, ún. white space (puha szóköz) karaktereket tetszés szerint lehet adagolni a HTML dokumentumok olvashatóbbá tétele érdekében, és erre a legtöbbnek égető szüksége is van. Egy másik következmény, hogy az üres sorokat nem lehet a bekezdések elválasztására felhasználni, mivel ezeket egyszerűen nem veszik figyelembe. Egy erre szolgáló címkére van szükség.

Néhány címkének (névvel rendelkező) paraméterei is vannak. Például a következő címke,

```
<IMG SRC="ABC" ALT="foobar">
```

egy <IMG> címke, amely rendelkezik egy *SRC* nevű, *abc* értékű és egy *ALT* nevű, *foobar* értékű paraméterrel. A HTML szabvány minden címkéhez felsorolja, hogy mik a megengedett paraméterek, ha vannak ilyenek, és mit jelentenek. Mivel minden paraméternek van neve, a paraméterek megadásának sorrendje lényegtelen.

Technikailag a HTML dokumentumok az ISO 8859-1 Latin-1 karakterkészlettel íródnak, de azon felhasználók kedvéért, akiknek a billentyűzete csak az ASCII-t támogatja, az è-hez hasonló különleges karakterekhez léteznek escape sorozatok*. Mindegyik egy & jellel kezdődik, és egy pontosvesszővel végződik. Például az è a è-t állítja elő, az é pedig az é-t. Mivel a <, >, és & karakterek speciális jelentéssel bírnak, csak az escape sorozataikkal (&lt, &gt, &amp) lehet őket kifejezni.

A fejrész legfontosabb része a cím, amelyet <TITLE> és </TITLE> címkék határolnak, de jelen lehetnek bizonyos járulékos információk is. A cím maga nem jelenik meg az oldalon, de néhány böngésző az oldal ablakának címéül használja.

Most nézzünk meg pár további szolgáltatást a 7.64. ábrán bemutatottak közül. A 7.65. ábra bemutatja a 7.64. ábrán használt összes címkét, és még pár másikat. A címsorokat egy <H*n*> címke állítja elő, ahol *n* egy 1 és 6 közé eső szám. <H1> a legfontosabb címsor, <H6> pedig a legkevésbé fontos. A böngészőre van bízva, hogy ezeket megfelelően visszaadja a képernyőn. A kisebb számú címsorok tipikusan nagyobb és vastagabb betűtípussal kerülnek megjelenítésre. A böngésző különböző színeket is választhat minden címsor-szinthez. A <H1> címsorok rendszerint nagyok és félkövérek, és legalább egy üres sor van alattuk és felettük. A <H2> címsorok kisebb betűtípussal íródnak, és kevesebb üres hely van alattuk és felettük.

A <B> és <I> címkék a félkövér (boldface) és dőlt betűs (italics) módba váltáshoz használatosak. Ha a böngésző nem képes megjeleníteni a félkövér vagy dőlt betűket, valamilyen más módot kell használnia a visszaadásukra, mint például mindkettőre más-más színt vagy esetleg inverz megjelenítést. A fizikai stílusok, mint amilyenek a félkövér és dőlt betűk, helyett a szerzők logikai stílusokat is használhatnak, mint pl. <DN> (meghatározás), <EM> (gyenge hangsúlyozás), <STRONG> (erős hangsúlyozás), és <VAR> (egy program változója). A logikai stílusokat egy **stíluslapnak (style sheet)** nevezett dokumentumban határozzák meg. A logikai stílusok előnye, hogy egy meghatározás megváltoztatásával minden változó például dőlt betűsről egyenletes betűszélességűre cserélhető.

A HTML különféle mechanizmusokat biztosít listák létrehozására, akár egymásba ágyazott listákhoz is. Az <UL> címke egy sorrend nélküli listát indít. Az egyes tételek előtt, amelyeket a forrásban az <LI> címke jelöl, pontok (.) jelennek meg. Ennek a mechanizmusnak egy változata az <OL>, amelyik sorszámozott listákat készít. Amikor ezt a címkét használják, az <LI> tételeket a böngésző megsorszámozza. Egy harmadik lehetőség a <MENU>, amely rendszerint egy kompaktabb listát eredményez a

* A speciális karakterek listáját a szabvány megadja.

képernyőn, pontok és számok nélkül. Az eltérő kezdő- és végcímkéktől eltekintve, az <UL>, <OL>, és <MENU> címkék hasonló szintaxisúak és hasonló eredményt adnak.

A 7.65 ábrán feltüntetett listamechanizmusokon kívül két másik is van, amely röviden említést érdemel. A <DIR> használható rövid táblázatokhoz. A <DL> és </DL> párossal lehet definíciólistákat (szójegyzéket) készíteni két részből álló bejegyzésekkel, amely részeket a <DT> és <DD> címkék határozzák meg. Az első a név, a második a jelentése. Ezeket a szolgáltatásokat nagyban feleslegessé tette az (általánosabb és összetettebb) táblázatmechanizmus, amelyet alább írunk le.

A
, a <P> és a <HR> címkék mind egy-egy határt jelölnek a szöveg egyes részei közt. A pontos formátumot az oldalhoz hozzárendelt stíluslap határozza meg. A
 címke csak egy sortörést kényszerít ki. A böngészők rendszerint nem szúrnak be üres sort a
 után. Ezzel ellentétben a <P> egy bekezdés kezdetét jelöli, amely például beszúrhat egy üres sort és esetleg némi behúzást is. (Elméletben a </P> szolgál arra, hogy egy bekezdés végét jelölje ki, de ritkán használják. A legtöbb HTML szerző nem is tudja, hogy ilyen létezik.) Végül, <HR> egy törést kényszerít ki, és egy vízszintes vonalat húz keresztbe a képernyőn.

A HTML 1.0 nem volt képes táblázatokat vagy más megformázott információt megjeleníteni. Ami még rosszabb volt, hogy ha a HTML írója gondosan megformázott egy táblázatot a szóközök és kocsi-vissza jelek megfelelő használatával, a böngészők figyelmen kívül hagyták az egész elrendezést, és úgy jelenítették meg az oldalt,

Címke	Magyarázat
<HTML> ... </HTML>	Azt deklarálja, hogy a Háló-oldal HTML-ben íródott
<HEAD> ... </HEAD>	Az oldal fejrészét határolja
<TITLE> ... </TITLE>	A címet határozza meg (ez az oldalon nem jelenik meg)
<BODY> ... </BODY>	Az oldal törzsét határolja
<H*n*> ... </H*n*>	Egy *n* szintű címsort határol
<B> ... </B>	A ...-ot félkövérré teszi
<I> ... </I>	A ...-ot dőlt betűssé teszi
<UL> ... </UL>	Egy sorszám nélküli (pontozott) listát zárójelez
<OL> ... </OL>	Egy sorszámozott listát zárójelez
<MENU> ... </MENU>	Egy <LI> tételekből álló menüt zárójelez
<LI>	A lista egy tételének kezdete (nincs </LI>)
 	Egy törés kikényszerítése az adott ponton
<P>	Bekezdés kezdete
<HR>	Vízszintes vonal
<PRE> ... </PRE>	Előre megformázott szöveg, nem szabad újraformázni
<IMG SRC="...">	Egy kép betöltése az adott ponton
<A HREF="..."> ... </A>	Egy élőkapcsot definiál

7.65. ábra. *Néhány szokásos HTML címke. Némelyiknek további paraméterei is vannak*

mintha nem lenne megformázva. A <PRE> és </PRE> címkéket biztosították ahhoz, hogy a böngészőket meggátolják a gondosan elrendezett szöveg összekuszálásában. Ezek a böngészőt arra utasítják, hogy a köztük levő szöveget egyszerűen szó szerint, karakterről karakterre, minden változtatás nélkül jelenítse meg. Ahogy a táblázatot és a többi díszes elrendezési szolgáltatást egyre szélesebb körben valósítják meg, a <PRE> címke iránti igény el fog enyészni, kivéve a programlistákat, amelyekhez a legtöbb programozó csak a saját formázását hajlandó eltűrni.

A HTML lehetővé teszi a Háló-oldalakon a sorok közt ábrák használatát is. Az <IMG> címke mondja meg, hogy az oldalon az aktuális pozícióba egy képet kell betölteni. Ennek számos paramétere lehet. Az SRC paraméter megadja az ábra URL-jét (vagy URI-ját). A HTML szabvány nem szabályozza, hogy mely grafikus formátumok megengedettek. A gyakorlatban minden böngésző támogatja a GIF állományokat és sok támogatja a JPEG állományokat is. A böngészők támogathatnak más formátumokat is, de ez a kiterjesztés fordítva is elsülhet. Ha egy felhasználó olyan böngészőhöz szokott, amely támogatja mondjuk a BMP állományokat, akkor esetleg ilyeneket fog rakni a saját Háló-oldalaira, és később meg fog lepődni, amikor a többi böngésző egyszerűen figyelmen kívül hagyja gyönyörű művét.

Az <IMG> további paraméterei az *ALIGN*, amely a képnek az ábra szövegsorához vonatkoztatott elhelyezkedését szabályozza (*TOP* – felette, *MIDDLE* – a közepén, *BOTTOM* – alatta), az *ALT*, amely egy szöveget biztosít arra az esetre, ha a felhasználó kikapcsolta volna a képeket, és az *ISMAP*, egy jelző, amely jelzi, hogy az ábra egy aktív térkép.

Végül eljutottunk az élőkapcsokhoz, amelyek az <A> (anchor, horgony) és </A> címkéket használják. Az <IMG>-hez hasonlóan, az <A>-nak is különféle paraméterei vannak, többek közt a *HREF* (az URL), a *NAME* (az élőkapocs neve), és a *METHODS* (elérési eljárások). Az <A> és a </A> közti szöveg megjelenítésre kerül. Ha az élőkapcsot kiválasztjuk, akkor eljutunk egy másik oldalra. Lehetséges egy <IMG> ábrát is elhelyezni ide, amely esetben az ábrán kattintás szintén aktiválja az élőkapcsot.

Példának vegyük az alábbi HTML részletet:

```
<A HREF="http://www.nasa.gov"> A NASA honlapja </A>
```

Amikor egy, ezzel a részlettel bíró oldal megjelenik, a képernyőn a következő fog feltűnni:

A NASA honlapja

Ha a felhasználó ezek után rákattint a szövegre, a böngésző azonnal elhozza és megjeleníti azt az oldalt, amelynek URL-je: *http://www.nasa.gov*.

Egy második példának most vegyük a következőt:

```
<A HREF="http://www.nasa.gov"> <IMG SRC="shuttle.gif" ALT="NASA"> </A>
```

Amikor ez megjelenik, az oldal egy képet fog mutatni az űrsiklóról (space shuttle).

Ha rákattintunk erre a képre, akkor a NASA honlapjára váltunk, ugyanúgy, mintha az előző példában az aláhúzott szövegre kattintottunk volna. Ha a felhasználó letiltotta az ábrák automatikus megjelenítését, akkor a NASA szöveg fog megjelenni a kép helyén.

Az <A> címke elfogad egy *NAME* nevű paramétert is, hogy az oldalon belülről lehessen erre hivatkozni. Például néhány Háló-oldal egy tartalomjegyzékkel kezdődik, amelyre rá lehet kattintani. A tartalomjegyzék egy elemére kattintva a felhasználó az oldal vonatkozó részére fog ugrani.

A HTML 2.0 nem tartalmazta azt a szolgáltatást, amelyet számos oldal szerzője hiányolt, nevezetesen, hogy lehessen olyan táblázatokat létrehozni, amelyek bejegyzéseire rákattintva élőkapcsokat lehet aktiválni. Következésképpen sok munkát végeztek

```
<HTML> <HEAD> <TITLE> Egy példaoldal egy táblázattal </TITLE> </HEAD>
<BODY>
<TABLE BORDER=ALL RULES=ALL>
<CAPTION> Néhány eltérés a HTML verziók közt </CAPTION>
<COL ALIGN=LEFT>
<COL ALIGN=CENTER>
<COL ALIGN=CENTER>
<COL ALIGN=CENTER>
<TR> <TH>Item <TH>HTML 1.0 <TH>HTML 2.0 <TH>HTML 3.0
<TR> <TH> Aktív térképek és képek <TD> <TD> X <TD> X
<TR> <TH> Egyenletek <TD> <TD> <TD> X
<TR> <TH> Űrlapok <TD> <TD> X <TD> X
<TR> <TH> Élőkapcsok <TD> X <TD> X <TD> X
<TR> <TH> Képek <TD> X <TD> X <TD> X
<TR> <TH> Listák <TD> X <TD> X <TD> X
<TR> <TH> Eszköztárak <TD> <TD> <TD> X
<TR> <TH> Táblázatok <TD> <TD> <TD> X
</TABLE> </BODY> </HTML>
```

(a)

Néhány eltérés a HTML verziók közt

Tétel	HTML 1.0	HTML 2.0	HTML 3.0
Aktív térképek és képek		X	X
Egyenletek			X
Űrlapok		X	X
Élőkapcsok	X	X	X
Képek	X	X	X
Listák	X	X	X
Eszköztárak			X
Táblázatok			X

(b)

7.66. ábra. *(a) Egy HTML táblázat. (b) Ennek a táblázatnak egy lehetséges megjelenítése*

azért, hogy a HTML 3.0-ban legyenek táblázatok is. Alább egy nagyon rövid bevezetőt adunk a táblázatokhoz, csak hogy az ízét érezni lehessen.

Egy HTML táblázat egy vagy több sorból áll, amelyek közül mindegyik egy vagy több **cellából (cell)** áll. A cellák sok mindent tartalmazhatnak, például szöveget, számokat vagy akár más táblázatokat is. A cellákat egybe lehet vonni, így például egy címsor több oszlopot is átfoghat. Az oldal szerzőjének van némi beleszólása az elrendezésbe, beleértve az elhelyezést, a keret stílusát és a cellahatárolókat, de a táblázatok megjelenítésénél a végső szót a böngészők mondják ki.

Egy HTML tábladefiníció látható a 7.66.(a) ábrán. Ennek egy lehetséges visszaadását mutatja a 7.66.(b) ábra. Ez a példa csak egy párat mutat meg a HTML táblázatok alapvető szolgáltatásai közül. A táblázatok a <TABLE> címkével kezdődnek. További információ is megadható a tábla általános tulajdonságainak leírására.

A <CAPTION> címke használható képaláírás biztosítására. Minden sor egy <TR> (Table Row, táblázatsor) címkével kezdődik. Az egyes cellákat a <TH> (Table Header, táblázat fejrésze) és a <TD> (Table Data, táblázatadat) jelöli. A megkülönböztetés lehetővé teszi a böngészők számára, hogy különbözőképpen adják vissza azokat, ahogy a példában is használtuk.

A táblázatokban számos egyéb címke is használatos, például olyanok, amelyekkel a cellák vízszintes és függőleges elrendezését, egy cellán belüli igazítást, a keretet, a cellák csoportosítását, egységeket és egyebeket szabályozhatunk.

Űrlapok

A HTML 1.0 alapvetően egyirányú volt. A felhasználók lehívhattak oldalakat a tartalomszolgáltatóktól, de bonyolult volt a másik irányba információt visszaküldeni. Ahogy egyre több és több kereskedelmi szervezet kezdte el használni a Hálót, a kétirányú forgalom iránt nagy igény jelentkezett. Sok társaság akarta azt például, hogy képes legyen rendeléseket felvenni a termékeire a Háló oldalain keresztül. A szoftvercégek szoftvert akartak terjeszteni a Háló útján, és az ügyfelekkel elektronikusan akarták kitöltetni a regisztrációs kártyákat. A Háló-keresést ajánló társaságok pedig azt akarták, hogy az ügyfeleik begépelhessék a keresési kulcsszavakat.

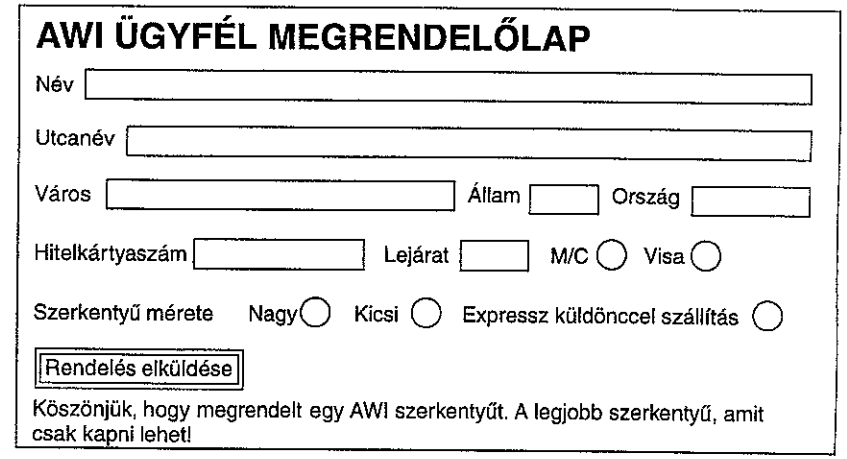
Ezek a követelések vezettek oda, hogy HTML 2.0-tól kezdve az **űrlapokat (forms)** is belevették a szabványba. Az űrlapok dobozokat vagy gombokat tartalmazhatnak, amelyek lehetővé teszik a felhasználók számára, hogy információt írhassanak be, vagy a felkínált lehetőségek közül választhassanak, majd az információt visszaküldhessék az oldal tulajdonosának. Erre a célra az <INPUT> (bevitel) címkét használják. Ennek sokfajta paramétere van, amelyek meghatározzák a megjelenített doboz méretét, természetét, és használatának módját. Ennek a legközönségesebb formái a felhasználó szövegét befogadó üres mezők, a kipipálható dobozok, az aktív térképek, és a SUBMIT gombok. A 7.67. ábra példája ezen választékból mutat be párat.

Kezdjük az űrlapok tárgyalását azzal, hogy végigmegyünk ezen az példán. Mint minden űrlapot, ezt is a <FORM> és </FORM> címkék fogják közre. A címkébe nem zárt szöveg egyszerűen megjelenik. Minden szokásos címke (mint pl. a <B>) megengedett az űrlapon belül. Ezen az űrlapon három fajta bemeneti dobozt használtunk.

Az első fajta bemeneti doboz a „Név" szöveg után következik. A doboz 46 karakter hosszú, és a felhasználótól egy szöveget várunk bele, amelyet azután a *customer* változóban tárolunk el későbbi feldolgozás céljából. A <P> címke arra utasítja a böngészőt, hogy a következő szöveget és dobozokat a következő sorban helyezze el, még

```
<HTML> <HEAD> <TITLE> AWI ÜGYFÉL MEGRENDELŐ LAP </TITLE> </HEAD>
</BODY>
<H1> Ügyfél megrendelő lap </H1>
<FORM ACTION="http://widget.com/cgi-bin/widgetorder" METHOD=POST>
Név <INPUT NAME="customer" SIZE=46> <P>
Utcanév <INPUT NAME="address" SIZE=40> <P>
Város <INPUT NAME="city" SIZE=20> <P>
Állam <INPUT NAME="state" SIZE=4> <P>
Ország <INPUT NAME="country" SIZE=10> <P>
Hitelkártyaszám <INPUT NAME="cardno" SIZE=10>
Lejárat <INPUT NAME="expires" SIZE=4>
M/C <INPUT NAME="cc" TYPE=RADIO VALUE="mastercard">
VISA <INPUT NAME="cc" TYPE=RADIO VALUE="visacard"> <P>
Szerkentyű mérete Nagy <INPUT NAME="product" TYPE=RADIO
VALUE="expensive">
Kicsi <INPUT NAME="product" TYPE=RADIO VALUE="cheap">
Expressz küldönccel szállítás <INPUT NAME="express" TYPE=CHECKBOX> <P>
<INPUT TYPE=SUBMIT VALUE="Rendelés elküldése"> <P>
Köszönjük, hogy megrendelt egy AWI szerkentyűt. A legjobb szerkentyű, amit
csak kapni lehet!
</FORM> </Body> </HTML>
```

(a)



(b)

7.67. ábra. *(a) Egy megrendelőlap HTML-je. (b) A megformázott oldal*

akkor is, ha lenne még hely az aktuális sorban. A <P> és más elrendezési címkék használatával az oldal szerzője szabályozhatja az űrlap kinézetét a képernyőn.

Az űrlap következő sora a felhasználó utcanevét kérdezi, 40 oszlop szélesen, szintén egy sorban egyedül. Az ezután következő sor a városra, az államra és az országra kérdez rá. Itt nem használtunk a mezők között <P> címkéket, így a böngésző mindet egy sorban fogja megjeleníteni, ha elférnek. Amennyit a böngésző tud, aszerint ez a bekezdés egyszerűen hat tételt tartalmaz: három karakterláncot és három dobozt váltakozva. Egymás után jeleníti meg ezeket, balról jobbra, mindannyiszor új sort kezdve, valahányszor az aktuális sorba nem fér ki a következő tétel. Ezért elképzelhető, hogy egy 1024×768-as képernyőn mindhárom karakterlánc és a hozzájuk tartozó dobozok is egy sorban fognak megjelenni, de egy 640×480-as képernyőn esetleg két sorra oszlanak. A legrosszabb esetben az „Ország" szó az egyik sor végén lesz, a hozzá tartozó doboz pedig a következő sor elején. Nincs arra mód, hogy közöljük a böngészővel, hogy a dobozt mindenképpen a szöveg mellé helyezze el.

A következő sor a hitelkártya számát és lejáratának dátumát kérdezi meg. A hitelkártyaszámokat csak megfelelő biztonsági intézkedések megtétele után lenne szabad átvinni az Interneten. Például néhány – de nem az összes – böngésző titkosítja a felhasználó által elküldött információt. Még ekkor is, a biztonságos kommunikáció és a kulcsgazdálkodás bonyolult dolog, és sokfelől támadható, ahogy azt korábban láttuk.

A lejárat dátuma után egy új szolgáltatással találkozunk: a rádiógombokkal. Ezeket akkor használják, amikor két vagy több lehetőség közül kell választani. Itt az elméleti modell egy autórádió, amelynek féltucat gombja van az állomások kiválasztására. A böngésző ezeket a dobozokat olyan formában jeleníti meg, amely lehetővé teszi a felhasználó számára, hogy kattintva (vagy a billentyűzet használatával) kiválaszthassa azokat vagy megszüntesse a kiválasztást. Ha az egyikre rákattintanak, az az ugyanabban a csoportban levő összes többit kikapcsolja. A vizuális megjelenítés a használt grafikus interfésztől függ. A böngészőn áll, hogy egy olyan formát válasszon, amely konzisztens a Windows-zal, a Motiffal, az OS/2 Warppal, vagy az éppen használt ablakozó rendszerrel. A szerkentyű méretére szintén két rádiógombot használunk. A két csoportot a *NAME* mezejükkel különböztetjük meg, és a statikus <RADIOBUTTON> és </RADIOBUTTON> címkékkel.

A *VALUE* paramétereket a lenyomott rádiógomb jelzésére használják. 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 *CHECKBOX* típusú doboz jelképez. Ezt 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. Például, amikor a felhasználó az Elektro-Pizza Háló-oldaláról rendel pizzát, akkor választhat szardíniát *és* hagymát *és* ananászt (ha így szereti), de nem választhat kis *és* közepes *és* nagy pizzát egyszerre. A pizzafeltéteket három külön *CHECKBOX* típusú gomb jelképezné, míg a pizza mérete rádiógombok egy halmaza lenne.

A rádiógombok némileg kényelmetlenek olyan nagyon hosszú listák esetén, amelyekből egy elemet kell kiválasztani. Ezért hozták létre a <SELECT> és </SELECT> címkéket, amelyek címkék közötti választási lehetőségek felsorolása található, a rádiógombok szemantikájával (hacsak nincs megadva a *MULTIPLE* (többszörös) paraméter, amelynél a szemantika megegyezik a kijelölhető dobozok szemantikájával). Néhány böngésző a <SELECT> és </SELECT> közt levő tételeket mint egy előugró menüt ábrázolja.

Eddig már az <INPUT> címke kétfajta beépített típusát is láttuk: a *RADIO* és *CHECKBOX* típusokat. Sőt, tulajdonképpen már egy harmadikat is láttunk már, a *TEXT*-et (szöveg). Mivel ez a típus az alapértelmezés, nem bajlódtunk azzal, hogy a *TYPE=TEXT* paramétert is megadjuk, de megtehettük volna. Két másik típus a *PASSWORD* és a *TEXTAREA*. Egy *PASSWORD* (jelszó) doboz ugyanolyan, mint egy *TEXT* doboz, kivéve, hogy a begépelt karakterek nem jelennek meg. A *TEXTAREA* (szöveges terület) doboz megegyezik a *TEXT* dobozzal, csak több sort is tartalmazhat.

Visszatérve a 7.67. ábra példájához, most a *SUBMIT* gombbal kapcsolatos példával találkozunk. Amikor erre rákattintunk, az űrlapban levő felhasználói információ visszakerül ahhoz a géphez, amely az űrlapot biztosította. Az összes többi típushoz hasonlóan a *SUBMIT* is egy fenntartott szó, amelyet a böngésző megért. A *VALUE* (érték) karakterlánc itt a gomb címkéje, és ez jelenik meg. Minden doboznak lehet értéke, de csak itt volt szükségünk erre a szolgáltatásra. A *TEXT* dobozok esetében a *VALUE* mező tartalma megjelenik az űrlappal együtt, de a felhasználó azt átszerkesztheti vagy kitörölheti. A *CHECKBOX* és *RADIO* dobozoknak szintén lehet kezdőértéket adni, ez egy *CHECKED* nevű mezővel tehető meg (mivel a *VALUE* csak a szöveget adja meg, de nem jelöl ki semmilyen előnyben részesített választási lehetőséget).

A böngésző megérti a *RESET* gombot is. Amikor erre rákattintunk, visszaállítja az űrlapot az eredeti állapotába.

Érdemes még két további típust megemlíteni. Az első a *HIDDEN* (rejtett) típus. Ez csak kimenetként szolgál, erre nem lehet rákattintani vagy módosítani. Például, amikor olyan oldalak sorozatán haladunk keresztül, amelyeken döntéseket kell hozni, a korábban meghozott döntések lehetnek *HIDDEN* típusúak, azok megváltoztatásának megelőzésére.

Az utolsó típusunk az *IMAGE* (ábra), amely az aktív térképekhez használatos (és más olyan ábrákhoz, amelyekre rá lehet kattintani). Amikor a felhasználó kattint a térképen, a kiválasztott pixel (x, y) koordinátái (vagyis az egér aktuális helyzete) változókban tárolódik, és az űrlap további feldolgozásra automatikusan visszakerül a tulajdonosához.

Az űrlapokat három módon lehet elküldeni: az elküldő gombot használva, rákattintva egy aktív térképre, vagy ENTER-t ütve egy egytételes *TEXT* űrlapon. Amikor egy űrlapot elküldenek, valamit cselekedni kell. A cselekvést a <FORM> címke paraméterei határozzák meg. Az *ACTION* paraméter meghatározza az URL-t (vagy URI-t), amelyet értesíteni kell a beadásról, és a *METHOD* paraméter mondja meg, melyik eljárást kell használni. Ezen (és az összes többi) paraméter sorrendje nem lényeges.

Az, hogy hogyan küldik vissza az űrlap változóit az oldal tulajdonosának, a *METHOD* paraméter értékétől függ. *GET* esetében csak csalással lehet értékeket visszaadni: hozzáfűzik azokat az URL-hez, egy kérdőjellel elválasztva. Ez a megközelítés könnyen eredményezhet több ezer karakter hosszú URL-eket. Ennek ellenére ezt az eljárást gyakran használják, mert egyszerű.

Ha a *POST* eljárást használják, akkor az üzenet törzse tartalmazza az űrlap változóit és azok értékeit. Az & jel használatos a mezők elválasztására, a + jelenti a szóközjelet. Például, a szerkentyűk űrlapjára egy válasz lehet a következő:

```
vásárló=John+Doe&cím=100+Main+St.&város=White+Plains
&állam=NY&ország=USA&kártyaszám=1234567890&lejárat=6/98&cc=mastercard&
termék=olcsó&express=be
```

Ezt a karakterláncot egyetlen sorként küldenék vissza a kiszolgálónak, és nem három sorként. Ha egy *CHECKBOX* nincs kiválasztva, akkor kimarad a karakterláncból. A kiszolgáló feladata, hogy értelmezze ezt a karakterláncot.

Szerencsére már létezik egy módszer az űrlapok adatainak kezelésére, amit **CGI-nek (Common Gateway Interface – közös átjáró interfész)** neveznek. Nézzünk a használatára egy szokásos példát. Tegyük fel, hogy valakinek van egy érdekes adatbázisa (pl. a Háló-oldalak egy indexe kulcsszó és téma szerint), és ezt a Háló felhasználói számára elérhetővé akarja tenni. CGI-vel ezt úgy teheti meg, hogy egy scriptet (vagy programot) ír, amely határfelületet (vagyis átjárót) képez az adatbázis és a Háló között. Ennek a scriptnek ad egy URL-t, szokás szerint a *cgi-bin* könyvtárban. A HTTP kiszolgálók tudják (vagy közölni lehet velük), hogy amikor egy, a *cgi-bin* könyvtárban elhelyezkedő oldalon kell egy eljárást meghívniuk, akkor az állomány nevét mint egy végrehajtható script vagy program nevét kell értelmezniük, és azt el kell indítaniuk.

Előbb vagy utóbb egy felhasználó meg fogja nyitni a szerkentyű-scriptünkhöz társított űrlapot, és azt megjeleníti. Miután kitöltötte az űrlapot, rá fog kattintani a SUBMIT gombra. Ez a cselekedet arra készteti a böngészőt, hogy egy TCP összeköttetést hozzon létre az űrlap *ACTION* paraméterében megadott URL-hez, vagyis a *cgi-bin* könyvtárban levő scripthez. Ezután a böngésző elvégzi az űrlap *METHOD*-ja által meghatározott műveletet, amely rendszerint a *POST*. Ennek a műveletnek az eredménye az lesz, hogy a script elindul, és (a TCP összeköttetésen keresztül, a standard bemenetére) megkapja a fent megadott hosszú karakterláncot. Ezen kívül számos környezeti változó is értéket kap, például a *CONTENT_LENGTH*, amely azt adja meg, milyen hosszú a bemeneti karakterlánc.

Ezen a ponton a legtöbb scriptnek értelmeznie kell a bemenetét, hogy azt egy kényelmesebb formába alakíthassa. Ezt a célt úgy érheti el, hogy meghív egyet a számos elérhető könyvtári vagy script eljárás közül. Ezután a script úgy kommunikálhat az adatbázisával, ahogyan csak óhajt. Például az aktív térképek rendszerint CGI scripteket használnak arra, hogy különböző cselekedeteket hajtsanak végre attól függően, hogy hova mutatott a felhasználó.

A CGI scriptek az űrlapokból származó információ elfogadása mellett kimenetet is előállíthatnak, és sok más dolgot is csinálhatnak. Ha egy élőkapocs egy CGI scriptre mutat, amikor azt a kapcsot aktivizálják, a script elindul, és számos környezeti változó is értéket kap, hogy valami információt adjon a felhasználóról. Ez a script azután kiír egy állományt (vagyis egy HTML oldalt) a kimenetére, amelyet elszállítanak a böngészőnek, és az értelmezi. Ez a mechanizmus lehetővé teszi, hogy a script egyéni Háló-oldalakat állíthasson elő helyben.

Jó-e vagy rossz, mindenesetre néhány Háló-állomás, amely lekérdezésekre válaszol, rendelkezik egy hirdetésekből álló adatbázissal is, amelyeket szelektíven el lehet helyezni az éppen készülő Háló-oldalon, attól függően, hogy mit keres a felhasználó. Ha a felhasználó a „gépkocsi" szóra keresett rá, akkor egy General Motors hirdetés jelenhet meg, míg egy „üdülés" iránti keresés egy United Airlines hirdetést eredményezhet. Ezekben a hirdetésekben rendszerint képek és szöveg van, amelyre rá lehet kattintani.

7.6.4. A Java

A HTML-lel le lehet írni a statikus Háló-oldalak megjelenését, beleértve a táblázatokat és a képeket is. A cgi-bin hozzátoldással még egy korlátozott mértékű kétirányú együttműködés is lehetséges (űrlapok stb.). Ám a HTML-ben írt Háló-oldalakkal történő gyors együttműködés nem lehetséges. Hogy lehetőség legyen nagymértékben interaktív Háló-oldalak létrehozására is, egy más mechanizmusra van szükség. Ebben a szakaszban egy ilyen mechanizmust, a Java nyelvet és annak értelmezőjét írjuk le.

A Java akkor jött létre, amikor a Sun Microsystemsnél néhány ember egy új, információorientált vásárlói segédletek írására alkalmas új nyelvet akart kifejleszteni. Később a Világháló (World Wide Web) irányába tolódott el a hangsúly. Bár a Java sok ötletet és némi szintaxist is kölcsönvett a C-től és a C++-tól, azért ez egy új objektumorientált nyelv, amely egyikkel sem kompatíbilis. Néha ezt úgy is mondják, hogy a Java távolról a Smalltalkra, de közelről megnézve a C-re vagy a C++-ra hasonlít.

A fő ötlet, amivel a Javát interaktív Háló-oldalakhoz használhatjuk, az az, hogy egy Háló-oldal hivatkozhat egy kis, **kisalkalmazásnak (applet)** nevezett Java programra. Amikor a böngésző ehhez elér, akkor a kisalkalmazás letöltődik az ügyfél gépére, és ott biztonságos módon végrehajtódik. Szerkezetileg lehetetlenné kell tenni, hogy a kisalkalmazás bármely olyan állományt írjon vagy olvasson, amelyhez nincs joga. Azt is lehetetlenné kell tenni, hogy a kisalkalmazás vírusokat hozzon be, vagy bárhogy máshogy kárt okozzon. Ezen okok miatt, továbbá hogy elérjék a különböző gépek közti hordozhatóságot, a kisalkalmazásokat megírásuk és hibamentesítésük után bájtkódra fordítják le. A Háló-oldalak ezekre a bájtkódú programokra hivatkoznak, hasonlóan, mint az ábrákra. Amikor egy kisalkalmazás megérkezik, egy értelmező egy biztonságos környezetben végrehajtja.

Mielőtt belemennénk a Java nyelv részleteibe, érdemes pár szót szólni arról, hogy mire is jó ez az egész Java rendszer, és miért akarnak az emberek Java kisalkalmazásokat helyezni a Háló-oldalaikra. Egyrészt a kisalkalmazások segítségével a Háló-oldalak interaktívak lehetnek. Például egy Háló-oldal tartalmazhat egy amőba-, othello- vagy sakktáblát, és játszhat egy játékot a felhasználóval. A játékot lejátszó program egyszerűen a Háló-oldallal együtt letöltésre kerül. Egy másik lehetőségként összetett űrlapok (táblázatok) jeleníthetők meg, amelyeken a felhasználó kitölt bizonyos mezőket, és azonnal látja a számítások eredményét.

Teljességgel lehetséges, hogy hosszú távon azt a modellt, miszerint az ember programokat vásárol, majd telepíti és helyileg futtatja azokat, felváltja egy másik modell, amelyben az ember Háló-oldalakon kattintgat, és letölti a kisalkalmazásokat azért,

hogy azok neki dolgozzanak, esetleg egy távoli kiszolgálóval vagy adatbázissal együtt. A jövedelemadó bevallására szolgáló űrlap kézzel vagy egy külön programmal való kitöltése helyett az emberek rákattinthatnak az IRS (az USA adóhivatala) honlapjára, és letöltenek egy adózással kapcsolatos kisalkalmazást. Ez a kisalkalmazás esetleg feltesz pár kérdést, majd kapcsolatba lép a kérdéses személy munkaadójával, bankjával és brókerével, hogy megszerezze a kívánt fizetési, kamat- és osztalékinformációkat, kitölti az adóívet, majd megjeleníti azt jóváhagyás és elküldés végett.

A kisalkalmazások futtatására egy másik ok az, hogy ezek lehetővé teszik Háló-oldalakhoz animáció és hang hozzáadását anélkül, hogy külső megjelenítőkre lenne szükség. A hangot mint háttérzenét le lehetne játszani, amikor az oldal betöltődik, vagy amikor egy bizonyos esemény bekövetkezik (pl. a macskára kattintás nyávogást eredményezhet). Ugyanez igaz az animációra is. Mivel a kisalkalmazás helyben fut, felül tudja írni a képernyő (hozzá tartozó) részét, amilyen módon csak akarja, és teheti mindezt nagyon nagy sebességgel (egy távoli cgi-bin shell scripthez viszonyítva).

A Java rendszernek három része van:

1. Egy Java forrásból bájtkódot előállító fordító.

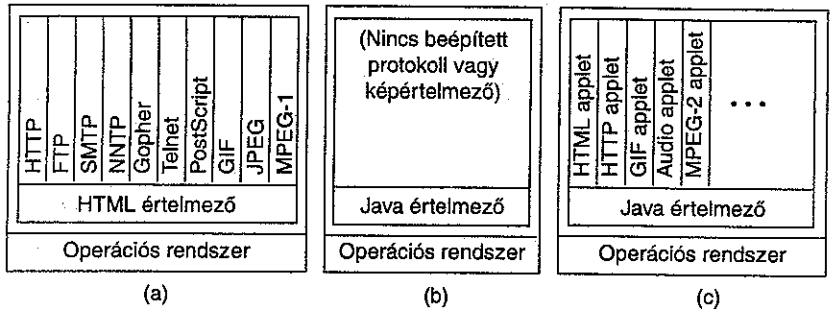
2. Egy böngésző, amely értelmezni tudja a kisalkalmazásokat.

3. Egy bájtkódértelmező.

A fejlesztő megírja a kisalkalmazást Javában, majd lefordítja azt bájtkódra. Hogy ezt a lefordított kisalkalmazást be lehessen ágyazni egy Háló-oldalra, kitaláltak egy új címkét, az <APPLET>-et. Egy jellemző felhasználás például:

```
<APPLET CODE=game.class WIDTH=100 HEIGHT=200> </APPLET>
```

Amikor a böngésző meglátja az <APPLET> címkét, elhozza a lefordított *game.class* kisalkalmazást az aktuális Háló-oldal állomásáról (vagy ha jelen van egy másik paraméter, a *CODEBASE*, akkor az ezáltal megadott URL-ről). Ezután a böngésző átadja a



7.68. ábra. *(a) Egy első generációs böngésző. (b) Egy Java alapú böngésző induláskor. (c) A (b) ábra böngészője, miután egy ideje már fut*

kisalkalmazást a helyi bájtkódértelmezőnek végrehajtásra (vagy ha van belső értelmezője, akkor saját maga értelmezi a kisalkalmazást). A *WIDTH* és *HEIGHT* paraméterek megadják a kisalkalmazás alapértelmezett ablakának méretét képpontokban.

Bizonyos értelemben az <APPLET> címke hasonlít az <IMG> címkére. Mindkét esetben a böngésző elhoz egy állományt, majd azt átadja egy (esetleg belső) értelmezőnek, amely megjeleníti a képernyő egy behatárolt területén. Ezután folytatja a Háló-oldal feldolgozását.

Azon alkalmazások számára, amelyeknek nagyon nagy teljesítményre van szükségük, néhány Java értelmezőnek van egy olyan képessége, hogy a bájtkódú programokat röptében szükség szerint gépi nyelvre fordítja.

Ennek a modellnek egy következménye az, hogy a Java alapú böngészők olyan módon is bővíthetők, ahogy az első generációs böngészők nem. Az első generációs böngészők rendszerint HTML értelmezők, amelyeknek a különféle protokollok (HTTP 1.0, FTP stb.) megértéséhez beépített moduljaik, a különféle képformátumokhoz pedig dekódereik vannak. Egy példa látható erre a 7.68.(a) ábrán. Ha valaki feltalál vagy népszerűvé tesz egy új formátumot, mint például a hangformátumok vagy az MPEG–2, ezek a régi böngészők nem képesek az ezeket tartalmazó oldalakat beolvasni. A legjobb esetben is a felhasználónak kell egy megfelelő külső megjelenítőt keresnie, letöltenie és telepítenie.

Egy Java alapú böngészőnél más a helyzet. Induláskor a böngésző tulajdonképpen egy üres Java virtuális gép, ahogy a 7.68.(b) ábrán látszik. A HTML és HTTP kisalkalmazásokat betöltve képessé válik szabványos Háló-oldalak beolvasására. Ahogy az új protokollokra és dekódolókra szükség van, ezek osztályait dinamikusan betölti, esetleg a hálózaton keresztül a Háló-oldalakon meghatározott állomásokról. Egy idő után a böngésző a 7.68.(c) ábrához hasonlóan nézhet ki.

Ezért ha valaki feltalál egy új formátumot, akkor mindösszesen annyit kell tennie, hogy egy, azt kezelni képes kisalkalmazás URL-jét beilleszti egy Háló-oldalra, és a böngésző automatikusan el fogja hozni és be fogja tölteni a kisalkalmazást. Nincs olyan elsőgenerációs böngésző, amely képes lenne röptében új külső megjelenítők letöltésére és telepítésére. Az a képesség, hogy a kisalkalmazásokat dinamikusan lehet betölteni, lehetővé teszi, hogy az emberek könnyen kísérletezhessenek új formátumokkal anélkül, hogy először vég nélküli szabványosítási találkozókon kellene egy megegyezés elérése céljából részt venniük.

Ez a kiterjeszthetőség a protokollokra is vonatkozik. Néhány alkalmazás számára különleges protokollok kellenek, például biztonságos protokollok a banki és kereskedelmi alkalmazásokhoz. A Java segítségével ezeket a protokollokat dinamikusan, szükség szerint lehet betölteni, és nincs szükség az unverzális szabványosításra. Ahhoz, hogy az X céggel tudjunk kommunikálni, csak a protokoll kisalkalmazását kell letölteni. Az Y céggel történő kommunikációhoz az ő protokoll kisalkalmazását kell megszerezni. Nincs arra szükség, hogy X és Y megegyezzen egy szabványos protokollban.

Bevezetés a Java nyelvbe

A fent felsorolt célok egy típusbiztos, objektumorientált nyelvhez vezettek el, amelyik beépített többszálúsággal rendelkezik, és nincsenek meghatározatlan vagy rendszerfüggő tulajdonságai. Itt a Java egy nagyban leegyszerűsített leírása következik, csak hogy rá lehessen érezni a lényegre. Sok szolgáltatást, részletet, lehetőséget és speciális esetet kihagytunk a tömörség kedvéért. A nyelv teljes specifikációja, és még sok minden más, természetesen magán a Hálón érhető el, a *http://java.sun.com* címen. Javás oktatási anyagokat lásd: (Campione és Walrath, 1996). A teljes történetet lásd: (Arnold és Gosling, 1996; Gosling és mások, 1996). A Java és a Microsoft által adott válasz (Blackbird) összehasonlítását lásd: (Singleton, 1996).

Ahogy fentebb említettük, közelről megnézve a Java hasonlít a C-re és a C++-ra. A nyelvtani szabályok például meglehetősen hasonlóak (pl. a fordítási egységeket puha szóköz karakterek választják el, és új sor bármely két fordítási egység közé elhelyezhető). Megjegyzések a C szintaxis szerint (/* ... */) és a C++ szintaxis szerint (// ...) is elhelyezhetők.

A Javának nyolc egyszerű adattípusa van, ezeket a 7.69. ábra sorolja fel. Minden típusnak egy meghatározott mérete van, a helyi megvalósítástól függetlenül. Ezért a C-től eltérően, ahol egy egész szám az azt futtató gép architektúrájától függően lehet 16, 32 vagy 64 bites, egy Java INT mindig 32 bites, nem több és nem kevesebb, mindegy, milyen fajta gépen fut az értelmező. Ez a konzisztencia szükséges, mivel ugyanannak a kisalkalmazásnak futnia kell 16, 32 és 64 bites gépeken is, és mindegyiken ugyanazt az eredményt kell adnia.

Az aritmetikai típusú változók (az első 6 típus) kombinálhatók a szokásos számtani műveletek (beleértve a ++ és -- műveleteket is) alkalmazásával, és összehasonlíthatók a szokásos relációjelekkel (<, <=, ==, !=). A típusok közti átalakítás megengedett, ahol ez értelmes.

A Java az ASCII helyett 16 bites Unicode karaktereket használ, így a karaktérváltozók két bájt hosszúak. Az első 127 Unicode karakter a visszafele kompatibilitás céljából ugyanaz, mint az ASCII-ban. Ezek felett foglal helyet néhány grafikus szimbólum, majd az orosz, arab, héber, japán (kanji, hiragana és katakana írásmódú), és gya-

Típus	Méret	Leírás
Byte	1 bájt	Egy –128 és +127 közti egész
Short	2 bájt	Egy előjeles 2 bájtos egész
Int	4 bájt	Egy előjeles 4 bájtos egész
Long	8 bájt	Egy előjeles 8 bájtos egész
Float	4 bájt	Egy 4 bájtos IEEE lebegőpontos szám
Double	8 bájt	Egy 8 bájtos IEEE lebegőpontos szám
Boolean	1 bit	Az egyedüli értékek az igaz és a hamis
Char	2 bájt	Egy Unicode karakter

7.69. ábra. *Az alapvető Java adattípusok*

korlatilag az összes többi nyelvben szükséges karakterek jönnek. Az ASCII-ban nem létező karakterek úgy írhatók, mint egy \u és négy hexadecimális számjegy. Például a Gurajati nullát \u0ae6 jelképezi.

A Java lehetővé teszi egydimenziós tömbök deklarálását. Például a következő sor:

```
int[] table;
```

egy table nevű tömböt deklarál, de nem foglal le annak helyet. Ezt később lehet megtenni, mint a C++-ban, például így:

```
table=new int[1024];
```

amely egy 1024 bejegyzésből álló tömbnek foglal helyet. Nem szükséges (és nem is lehetséges) visszaadni olyan tömböket, amelyekre nincs már szükség: a hulladékgyűjtő rutin visszaveszi azokat. Ezért a nagy hibalehetőségű *malloc* és *free* könyvtári rutinokra nincs szükség a tárkezeléshez. A tömböknek kezdőértéket lehet adni, és magasabb dimenziószám esetén tömbökből álló tömbök használhatók, mint a C-ben. A karakterláncok rendelkezésre állnak, de egy osztályban definiálják őket, ahelyett, hogy csak nulla bájttal végződő karaktertömbök lennének.

A Java vezérlőutasítások a 7.70. ábrán láthatóak. Az első kilencnek tulajdonképpen ugyanaz a szintaktikája és szemantikája, mint a C-ben, kivéve azt, hogy ahol egy Boolean kifejezést vár a nyelv, ott ezt meg is követeli. A break és continue utasítások most már elfogadnak címkéket, amelyek azt mondják meg, hogy a felcímkézett hurkok közül melyikből kell kilépni vagy melyiket kell folytatni.

A következő két utasítás megvan a C++-ban, de a C-ben nincs. A throw és try utasítások a kivételkezelésre vonatkoznak. A Java sokfajta szabványos kivételt (exception)

Utasítás	Leírás	Példa
Értékadás	Értékadás	n=i+j;
If	Booleanválasztás	if(k<0) k=0; else k=2*k;
Switch	Egy eset kiválasztása	switch (b) case 1:n++; case 2: n--;)
For	Iteráció	for(i=0;i<n;i++) a[i]=b[i];
While	Ismétlés	while(n<k) n+=i;
Do	Ismétlés	do {n=n+n} while (n<m);
Break	Utasításból kilépés	break label;
Return	Visszatérés	return n;
Continue	Következő iteráció	continue label;
Throw	Kivétel előidézése	throw new IllegalArgumentException();
Try	Kivétel érvényessége	try {...} catch (Exception e) {return –1};
Synchronized	Kölcsönös kizárás	synchronized void update(int s) {...}

7.70. ábra. *A Java utasításai. A { ... } jelölés egy kódblokkot jelöl*

definiál, mint például a nullával való osztás, és lehetőséget ad a programozóknak, hogy definiálják és előidézzék a saját kivételeiket. A programozók kezelőket írhatnak a kivételek elfogására, szükségtelenné téve ezáltal az állandó ellenőrzést, hogy minden rendben van-e (például egy állomány olvasása közben). A throw utasítás egy kivételt idéz elő, a try utasítás pedig meghatározza azt az érvényességi kört, amelyen belül a kivételkezelőket összerendeli egy olyan kódrészlettel, amelyben egy kivétel előfordulhat.

A synchronized utasítás új a Javában, és azért kellett, mert a Java programok több szálat is ellenőrzés alatt tarthatnak. Ez az utasítás használatos arra, hogy behatárolják a kód egy részletét (vagy egy egész eljárást), amelyben egyszerre csak egy aktív szál tartózkodhat, a versenyhelyzetek elkerülése végett. Az ilyen kódrészleteket rendszerint **kritikus tartományoknak (critical regions)** nevezik. Amikor a synchronized utasítás végrehajtásra kerül, az azt végrehajtó szálnak meg kell szereznie a kritikus tartományhoz rendelt zárat, végre kell hajtania a kódot, majd el kell engednie a zárat. Ha a zár nem elérhető, a szál addig vár, amíg az fel nem szabadul. Ily módon egész eljárásokat őrizve és feltételváltozókat használva, a programozók rendelkezésére áll a monitorprogramok minden lehetősége (Hoare, 1974).

A Java programokat paraméterekkel is meg lehet hívni. A parancssor feldolgozása hasonló a C-hez, kivéve, hogy a paramétertömb neve *argv* helyett *args*, és *args*[0] az első paraméter, nem pedig a program neve. A 7.71. ábra egy kis Java programot mutat, amely a faktoriálistáblát számítja ki, csak annak demonstrálására, hogy hogyan is néz ki egy kis Java program.

Annak ellenére, hogy a C++ és a Java mindketten a C-n alapuló objektumorientált nyelvek, néhány dologban mégis különböznek. Néhány tulajdonságot kivettek a Javából, hogy típusok szempontjából biztonságossá vagy könnyebben olvashatóvá tegyék. Ezek közt van a #define, a typedef, az enum, a union, a struct, az operátor overloading,

```
class Faktoriális {/* Ez a program egyetlen osztályból és annak két metódusából áll*/

  public static void main (int argc, String args[]) { // főprogram
  long i, f, lower = 1, upper = 20;        // négy long változó deklarálása

  for (i = lower; i <= upper; i++)         // ciklus alsótól a felsőig
      f = factorial(i);                    // f = i!
      System.out.println(i + " " + f);     // i és f kiíratása
  }
}

  static long factorial (long k) {         // rekurzív faktoriális függvény
      if (k == 0)
          return 1;                        // 0! = 1
      else
          return k * factorial(k-1);       // k! = k * (k-1)!
  }
}
```

7.71. ábra. *Egy Java program, amely a faktoriálisokat írja ki 0!-tól 20!-ig*

az explicit mutatók, a globális változók, az önálló függvények és a friend függvények. Mondani sem kell, hogy a goto utasítást az elavult programnyelvi szolgáltatások számára fenntartott különleges helyre száműzték. Más szolgáltatásokat viszont bevettek, hogy több hatalmat adjanak a nyelvnek, mint a hulladékgyűjtés, a többszálúság, az objektum interfészek és a csomagok.

Objektumorientáltság a Javában

A hagyományos eljárásalapú nyelvekben, mint amilyen a Pascal vagy a C, egy program változók és eljárások halmaza, minden általános rendező elv nélkül. Ezzel ellentétben az **objektumorientált nyelvekben (object-oriented languages)** (majdnem) minden dolog objektum. Egy **objektum (object)** rendszerint némi belső (vagyis rejtett) állapotváltozóból és néhány nyilvános **metódusnak (method)** nevezett eljárásból áll. A programoknak, amelyek az objektumot használják, a metódusokat illik meghívni az objektum állapotának manipulálásához (és kikényszeríthető, hogy ennek így is kelljen történnie). Ily módon az objektum írója szabályozhatja azt, hogy a programok hogyan használhatják az objektumon belüli információkat. Ezt az elvet **beágyazásnak (encapsulation)** nevezik, és ez az alapja minden objektumorientált programozásnak.

A Java megpróbálja mindkét oldalról a legjobbakat átvenni. Használható hagyományos eljárásalapú nyelvként vagy objektumorientált nyelvként is. Például a 7.71. ábra Java példáját ugyanilyen jól, és lényegében ugyanilyen módon meg lehetett volna írni C-ben is. Ezen nézőpontból a Java egy részhalmaza a C egy feljavított változatának tekinthető. Viszont Háló-oldalak írásához jobb, ha a Javára mint egy objektumorientált nyelvre gondolunk, így ebben a szakaszban az objektumorientáltságát fogjuk tanulmányozni.

Egy Java program egy vagy több **csomagból (package)** áll, amelyek közül mindegyik néhány osztálydefiníciót tartalmaz. A csomagokat el lehet érni egy hálózaton keresztül távolról is, így azoknak a csomagoknak, amelyek bárki által elérhetők, egyedi nevekkel kell rendelkezzenek. Rendszerint hierarchikus neveket használnak, amelyek a gép DNS nevének fordítottjával kezdődnek, mint pl:

```
EDU.univ.cs.catie.games.chess
```

Egy **osztály (class)** definíciója egy sablon, amely az objektumpéldányokat körvonalazza. Ezek közül mindegyik ugyanolyan állapotváltozókkal és metódusokkal rendelkezik, mint az osztály összes többi objektumpéldánya, viszont különböző objektumokon belül az állapotváltozók értékei függetlenek. Ezért az osztályok olyanok, mint a süteményformák: ők maguk nem sütemények, de arra használják őket, hogy szerkezetükben azonos süteményeket vágjanak ki velük. Minden süteményforma más-más süteményt vág ki. Ha már egyszer elkészültek, a különböző sütemények (objektumok) egymástól függetlenek.

A Java objektumokat dinamikusan lehet végrehajtás közben létrehozni, például:

```
objektum=new OsztályNév()
```

Ezeket az objektumokat a heap-ben tárolják, és a hulladékgyűjtő távolítja el őket, amikor már nincs rájuk szükség. Ily módon a tárkezelést a Javában a rendszer végzi, így nincs szükség a rettegett *malloc* és *free* eljárásokra, vagy akár explicit mutatókra.

Minden osztály egy másik osztályon alapul. Egy újonnan definiált osztály **alosztálya (subclass)** annak, amelyiken alapul, míg az utóbbi az előbbinek **szuperosztálya (superclass)**. Egy (al)osztály mindig örökli a szuperosztályának eljárásait. A szuperosztály belső változóit vagy eléri, vagy nem, attól függően, hogy a szuperosztály ezt akarja-e. Például ha egy *A* szuperosztálynak *M1*, *M2* és *M3* metódusai vannak, és a *B* alosztály egy új metódust hoz létre, *M4*-et, akkor a *B*-ből készített objektumoknak *M1*, *M2*, *M3* és *M4* metódusaik lesznek. Azt a tulajdonságot, hogy egy osztály automatikusan megkapja a szuperosztályának metódusait, **öröklődésnek (inheritance)** nevezzük, és ez a Java egyik fontos tulajdonsága. További metódusok hozzáadását a szuperosztályhoz a szuperosztály **kiterjesztésének (extending)** nevezzük. Egyébként néhány objektumorientált nyelv megengedi, hogy az osztályok két vagy több szuperosztálytól is örökölhessenek metódusokat, de a Java tervezői ezt a tulajdonságot túl zavarosnak ítélték meg, és szándékosan kihagyták.

Mivel minden osztálynak pontosan egy szuperosztálya van, egy Java program összes osztálya egy fát alkot. A fa tetején levő osztályt **Objektumnak (Object)** hívják.

```
class KomplexSzám {                    // Az objektum egy KomplexSzám-nak nevezett
                                       alosztályának definiálása
  // Elrejtett adat
  protected double re, im;             // valós és képzetes részek

  // Öt metódus, amely az elrejtett adatot manipulálja
  public void Komplex(double x, double y) {re = x; im = y;}
  public double Valós() {return re;}
  public double Képzetes() {return im;}
  public double Abszolútérték() {return Math.sqrt(re*re + im*im);}
  public double Szög() {return Math.atan(im/re);}
}

class teszt                            // Egy második osztály, a KomplexSzám
                                       letesztelésére
  public static void main /String args[]) {
    KomplexSzám c;                     // Egy KomplexSzám osztályú objektum
                                       deklarálása

    c = new KomplexSzám()              // c számára tárterület lefoglalása
    c.Komplex(3.0, 4.0);               // A Komplex metódus meghívása a c
                                       inicializálásához
    System.out.println("c abszolútértéke " + c.Abszolútérték() );
  }
}
```

7.72. ábra. *Egy csomag, amely két osztályt definiál*

Az összes többi osztály örökli ennek a metódusait. Bármely osztály, amelynek a definíciójában nincs kifejezetten megnevezve a szuperosztálya, alapértelmezésként az *Objektum* osztály alosztálya lesz. Például a 7.71. ábra *Faktoriális* osztálya ezért az *Objektum* egy alosztálya.

Most nézzünk egy példát az eddig bemutatott objektumorientált elvekre. A 7.72. ábrán egy két osztályt definiáló csomag látható. A *KomplexSzám* komplex számok (vagyis egy valós és egy képzetes résszel rendelkező számok) definiálására és használatára való, a *teszt* pedig megmutatja, hogyan lehet a *KomplexSzám*-ot használni.

Mint a *Faktoriális*, a *KomplexSzám* osztály is az *Objektum*-on alapul, mivel nincs más szuperosztály megnevezve a definícióban. Minden *KomplexSzám* osztályú objektum egy komplex számot jelképez. Ennek az osztálynak minden objektuma két rejtett változót tartalmaz, a *re*-t és az *im*-et. Mindkettő 64 bites lebegőpontos szám, és a valós és képzetes részeket jelképezi. Az osztály (és alosztályai) definícióján kívül máshonnan nem érhetők el, mert a protected kulcsszóval védettnek nyilvánítottuk azokat. Ha private-nak (magán) nyilvánítottuk volna őket, akkor csak a *KomplexSzám* számára lennének láthatóak, az alosztályok számára nem. Most még rendben volna a private használata, de rövidesen egy alosztályt is definiálni fogunk. Ha public-nak (közös) deklaráltuk volna őket, akkor mindenhonnan látni lehetne őket, ahonnan az osztályt látni lehet, és ezáltal az objektumorientált programozás sok előnye elveszne. Ennek ellenére vannak olyan helyzetek, amikor szükség van arra, hogy egy objektum belső állapotának public-nak kell lennie.

Öt metódust definiáltunk a *KomplexSzám* osztályhoz tartozó objektumok számára, így az osztály felhasználói csak az ezen öt metódus által biztosított műveletekre szorítkozhatnak, és nem juthatnak közvetlenül az állapotváltozókhoz. A *teszt*-ben egy példát adunk arra, hogyan lehet *KomplexSzám* osztályú objektumokat készíteni, inicializálni, és használni.

Amikor ezt a csomagot lefordítják, a Java fordító két bináris (bájtkódú) állományt állít elő, amelyek közül mindegyik egy osztályt tartalmaz, és a nevét is arról kapja. A

```
java teszt
```

parancs begépelésekor meghívjuk a Java értelmezőt a *teszt* osztállyal, mint paraméterrel. Ezután az értelmező egy *main* nevű metódust keres, és amikor megtalálja végrehajtja azt. A végrehajtás eredménye a következő sor kiírása lesz:

```
c abszolútértéke 5
```

Most definiáljuk a *KomplexSzám* egy alosztályát, csak hogy lássuk, hogyan is megy ez. Ez az eredeti osztály beemelésével kezdődik, hogy megtudjuk, mit csinál és mik a metódusai. Ezután definiáljuk a *KomplexSzám* egy kiterjesztését, amelyet *SzőrösSzám*-nak fogunk hívni. Az új osztály automatikusan örökli a szuperosztályban szereplő öt metódust. Hogy érdekesebbé tegyük a dolgot, az alosztályban egy hatodik metódust is definiálunk, az *AddHozzá*-t, amely egy komplex számot ad hozzá az objektumhoz, megnövelve annak valós és képzetes részét.

Az alosztály definíciója a 7.73. ábrán látható, egy másik tesztprogrammal együtt, amely azt mutatja, hogy hogyan lehet egy *SzőrösSzám* osztályhoz tartozó objektumot használni. Amikor az új tesztprogram lefut, a következőt fogja kiírni:

```
h=(–0,5,6)
```

Ne felejtsük el, hogy a hat eljárás az *a* és *h* objektumokon használható, attól függetlenül, hogy melyik metódust hol definiáltuk. Ha most még egy alosztályt definiálunk a *SzőrösSzám*-ra alapozva, és mondjuk három új metódussal látjuk el, akkor az abból képzett osztályoknak kilenc érvényes metódusa lesz.

Azon kívül, hogy új metódusokat adhat hozzá a szuperosztályhoz, egy alosztály felülírhatja (kicserélheti) a már meglevő metódusokat egyszerűen újradefiniálva azokat. Így lehetséges egy alosztálynak az összes, szuperosztálytól örökölt metódusát felülírnia, így a két osztályba tartozó objektumoknak semmi közük nem lesz egymáshoz. Viszont ez rossz ízlésre vall, és legjobb, ha az ilyet elkerüljük.

Végül egy Java osztály több metódust is definiálhat azonos névvel, de különböző paraméterekkel és különböző definíciókkal. Amikor a fordító egy olyan hívást lát, amely ezt a nevet használja, a paramétertípusokat használva kell eldöntenie, melyik metódust kell használnia. Ezt a tulajdonságot **túltöltésnek (overloading)** vagy **polimorfizmusnak (polymorphism)** hívják. A C++-ban a műveleti jeleket is lehet túltölteni, de a Javában csak a metódusokat, hogy a programokat könnyebben érthetővé tegyék.

```
import KomplexSzám;                    // A KomplexSzám csomag beemelése

class SzőrösSzám extends KomplexSzám { // egy új osztály definiálása
  public void AddHozzá(KomplexSzám z) { // egy metódussal
    re = re + z.Valós();
    im = im + z.Képzetes();
  }
}

class teszt2 {                         // A SzőrösSzám tesztprogramja
  public static void main(String args[]) {
    SzőrösSzám a, h;                   // Két SzőrösSzám deklarálása

    a = new SzőrösSzám();              // Tárhely foglalása a-nak
    a = new SzőrösSzám();              // Tárhely foglalása h-nak
    a.Komplex(1.0, 2.0);               // Érték adása a-nak
    h.Komplex(-1.5, 430);              // Érték adása h-nak
    h.AddHozzá(a);                     // h AddHozzá metódusának meghívása
    System.out.println("h = (" + h.Valós() + "," + h.Képzetes() + ")" );
  }
}
```

7.73. ábra. *A KomplexSzám egy alosztálya, amely egy új metódust is definiál*

Az alkalmazásprogramozók interfésze

Az alapnyelvet kiegészítendő, a Java tervezői az eredeti kiadásban még körülbelül 200 osztályt is definiáltak és implementáltak. Az ezekben az osztályokban található metódusok egyfajta szabványos környezetet biztosítanak a Java programok fejlesztőinek. (Application Programmers Interface, API) Az osztályokat mind Javában írták, így minden platformra és operációs rendszerre átvihetők.

Bár mindezen osztályok és metódusok tárgyalása nyilvánvalóan túlnyúlik e könyv keretein, egy rövid leírás azért érdekes lehet. A 200 osztályt hét egyenlőtlen méretű csomagba csoportosították, amelyek közül mindegyik valamely központi téma köré csoportosul. Azok a kisalkalmazások, amelyeknek egy bizonyos csomagra van szükségük, a Java import utasítását használva emelhetik be azt. Ezután az abban levő metódusok tetszés szerint használhatók. Ez a mechanizmus helyettesíti a C-ben szokásos header állományok beemelését, valamint a könyvtárakra sincs szükség, mivel a csomagokat a végrehajtáskor dinamikusan töltik be, amikor azokat meghívják.

A hét csomagot a 7.74. ábra foglalja össze. A *java.lang* csomag olyan osztályokat tartalmaz, amelyekre mint a nyelv részére tekinthetünk, de technikailag mégsem azok. Ezek között vannak az osztályok szervezését, a szálakat, és a kivételkezelést megvalósító osztályok. Szintén itt kaptak helyet a szokásos matematikai és a karakterláncokhoz szükséges könyvtárak.

A C-hez hasonlóan a Java nyelv nem tartalmaz Bevitel/Kivitel primitíveket. A B/K műveleteket a *java.io* csomag betöltésével és használatával lehet elvégezni. Ez hasonlatos a C standard B/K könyvtárához. Különböző metódusok vannak a folyamok és a tetszőleges elérésű állományok írásához és olvasásához, valamint a kiíráshoz szükséges formázáshoz is. A 7.61. ábrán láthattuk ezen metódusok egyikét, a *println*-t, amely formázott kiíratást végez.

A B/K-hez szorosan kapcsolódik a hálózati szállítás. Itt vannak azon metódusok, amelyek az IP címek felkutatását és kezelését végzik. A csomag része a socketekhez való hozzáférés és a datagramok előkészítése is. A tulajdonképpeni átvitelt a *java.io* kezeli.

A következő csomag a *java.util*. Ez a szokásos adatstruktúrákhoz, mint pl. a verem

Csomag	Példa a feladatkörre
Java.lang	Osztályok, szálak, kivételek, matematika, karakterláncok
Java.io	B/K folyamokon és tetszőleges elérésű állományokon, nyomtatás
Java.net	Socketek, IP címek, URL-ek, datagramok
Java.util	Vermek, hash táblák, vektorok, idő, dátum
Java.applet	Háló-oldalak megszerzése és megjelenítése, hang, Object osztály
Java.awt	Események, párbeszédablakok, menük, karakterkészletek, grafika, ablakkezelés
Java.awt.image	Színek, képbegyűjtés, szűrés, és átalakítás
Java.awt.peer	Az alatta elterülő ablakozó rendszer elérése

7.74. ábra. *A szabványos API-ban meglevő csomagok*

és a hash-táblák, tartalmaz osztályokat és metódusokat, így a programozóknak nem kell mindig újra feltalálniuk a kereket. Itt van a dátum és az idő kezelése is.

A *java.applet* csomag a kisalkalmazások néhány alapvető eszközét tartalmazza, többek között azokat a metódusokat, amelyeknek egy URL-t megadva megszereznek egy Háló-oldalt. A Háló-oldalak megjelenítéséhez és a hangklipek (vagyis a háttérzene) lejátszásához is vannak metódusok. A *java.applet* csomag tartalmazza az *Object* osztályt is. Minden objektum örökli ennek eljárásait, hacsak nem írják azokat felül. Az eljárások között van pl. egy objektum klónozása, két objektum egyenlőségének vizsgálata, egy objektum karakterlánccá alakítása, és még számos más eljárás.

Végül eljutottunk a *java.awt*-ig, és annak két alcsomagjáig. Az AWT annyit tesz: **Abstract Window Toolkit (absztrakt ablak eszközkészlet)**, és arra tervezték, hogy az ablakozó rendszerek közt hordozhatóvá tegye a kisalkalmazásokat. Például, hogyan rajzoljon egy kisalkalmazás egy téglalapot a képernyőre úgy, hogy a kisalkalmazásnak ugyanaz a lefordított (bájtkódú) változata fusson UNIX, Windows és Macintosh alatt is, még ha mindnek más is az ablakozó rendszere. A csomag egy része a képernyőre történő rajzolással foglalkozik, így vannak metódusok vonalak, geometriai ábrák, szöveg, menük, gombok, görgetősávok és sok más elem képernyőre történő kihelyezésére is. A Javában programozók ezeket a metódusokat hívják meg ahhoz, hogy a képernyőre írjanak. A *java.awt* csomag feladata, hogy a helyi operációs rendszer megfelelő meghívásával a munkát elvégezze. Ez a stratégia azt eredményezi, hogy a *java.awt*-t minden platformra újra kell írni, de ezután a kisalkalmazások platformfüggetlenek lesznek, és ez sokkal fontosabb.

Ennek a csomagnak egy másik fontos feladata az eseménykezelés. A legtöbb ablakozó rendszer alapvetően eseményvezérelt. Ez azt jelenti, hogy az operációs rendszer érzékeli a billentyűleütéseket, az egérmozgást, a gombok lenyomását és felengedését, valamint más eseményeket, majd ezeket a felhasználói eljárások meghívásába alakítja át. A Java esetében a *java.awt* nagy könyvtárat biztosít ezeknek az eseményeknek a kezelésére. Ezeket használva egyszerűbb olyan programokat írni, amelyek együttműködnek a helyi ablakozó rendszerrel, de mégis 100 százalékig hordozhatók a különféle operációs rendszerek és ablakozó rendszerek közt.

A csomag munkájának egy részét a *java.awt.image* végzi el, mint pl. a képkezelést, egy másik részét pedig a *java.awt.peer*, amely elérhetővé teszi az alatta elterülő ablakozó rendszert.

Biztonság

Ami a Javában a legfontosabb, az a biztonság. Amikor egy kisalkalmazást tartalmazó Háló-oldalt elhoznak, akkor a kisalkalmazás automatikusan végrehajtásra kerül az ügyfél gépén. Elméletileg ennek nem lenne szabad az ügyfél gépét lefagyasztania vagy máshogy elrontania.

Továbbá az sem kíván nagy fantáziát, hogy elképzeljünk egy vállalkozó kedvű egyetemistát, aki előállít egy csillogó új játékot tartalmazó Háló-oldalt, majd széles körben elterjeszti az URL-jét (például keresztbepostázza az összes hírcsoportba). Ám azt a kis részletet elfelejti megemlíteni a levélben, hogy az oldal egy kisalkalmazást is

tartalmaz, amely megérkezésekor azonnal titkosítja az összes állományt a felhasználó merevlemezén. Amikor ezzel végzett, akkor a kisalkalmazás bejelenti, hogy mit tett, és udvariasan megemlíti, hogy a titkosító kulcsot megvásárolni kívánó felhasználók ezt úgy tehetik meg, hogy egy bizonyos panamai postafiókba 1000 dollárt küldenek kis, megjelöletlen bankjegyekben.

A fenti hogyan-gazdagodjunk-meg-gyorsan eljáráson kívül más veszélyeket is rejt az, ha idegen kódot engedünk futni a saját gépünkön. Egy kisalkalmazás körbenézhet érdekes információkat (elmentett elektronikus levelek, a jelszóállomány, a helyi környezeti változók stb.) keresve, majd visszaküldheti vagy postázhatja azokat a hálózaton keresztül. Erőforrásokat is fogyaszthat (pl. megtöltheti a lemezt), kínos képeket vagy politikai szlogeneket jeleníthet meg a képernyőn, vagy fülhasogató lármát kelthet a hangkártyát használva.

A Java tervezői természetesen nagyon is tudatában voltak ezeknek a problémáknak, és egy egész sor korlátot állítottak fel ellenük. A védelem első vonala a típusok szempontjából biztonságos nyelv. A Java erősen típusos, indexhatár-ellenőrzéssel rendelkező valódi tömböket tartalmaz, mutatókat viszont nem. Ezek a korlátozások lehetetlenné teszik azt, hogy egy Java program tetszőleges memóriahely olvasására vagy írására mutatót hozzon létre.

Viszont Trudi, aki azóta felhagyott a kriptográfiai protokollok feltörésére irányuló kísérletekkel, és inkább a rosszindulatú Java kisalkalmazások írásának sokkal érdekesebb üzletébe szállt be, egyszerűen írhat vagy módosíthat egy C fordítót, hogy az Java bájtkódot állítson elő, és ezzel túljuthat a Java nyelv és fordító által felállított minden védelmen.

A védelem második vonala, hogy mielőtt végrehajtanának egy beérkezett kisalkalmazást, végigfuttatják azt egy bájtkód-ellenőrzőn. A bájtkód-ellenőrző figyeli a mutatók előállítására vonatkozó kísérleteket, az érvénytelen paraméterekkel végrehajtott utasításokat vagy metódushívásokat, a változók inicializálás előtti használatát és így tovább. Ezeknek az ellenőrzéseknek garantálniuk kellene azt, hogy csak a szabályos kisalkalmazások kerüljenek végrehajtásra, de Trudi nyilván keményen fog dolgozni azon, hogy olyan trükköket találjon, amelyeknek az ellenőrző nem néz utána.

A védelem harmadik vonala az osztálybetöltő. Mivel az osztályokat röptében lehet betölteni, fennáll az a veszély, hogy egy kisalkalmazás az egyik saját osztályát tölti be, hogy egy kritikus rendszerosztályt helyettesítsen, s ezáltal túllépjen az osztály biztonsági ellenőrzésein. Ezt a trójai faló típusú támadást azzal tették lehetetlenné, hogy minden osztálynak saját névteret (egyfajta absztrakt könyvtárat) adtak, és először a rendszerosztályokat vizsálják át, mielőtt a felhasználó osztályait vennék sorra. Más szavakkal, ha a felhasználó a *println* egy rosszindulatú változatát tölti be, az soha nem kerül felhasználásra, mert először mindig a hivatalos *println* jön elő.

A védelem negyedik vonala, hogy néhány szabványos osztály saját beépített biztonsági intézkedésekkel rendelkezik. Például az állományelérési osztály karbantart egy listát azokról az állományokról, amelyeket a kisalkalmazások elérhetnek, és felrak egy párbeszédablakot, valahányszor egy kisalkalmazás olyat kísérel meg, amely megsérti a védelmi szabályokat.

Mindezen intézkedések ellenére várhatóak biztonsági problémák. Először is, lehetnek hibák a Java szoftverben, amelyeket a ravasz programozók a védelem áthágására

használhatnak fel. Az 1988-as elhíresült Internet féreg a UNIX Finger daemonjában használt fel egy hibát arra, hogy az egész Interneten több ezer gépet kényszerítsen leállásra (Hafner és Markoff, 1991, Spafford, 1989).

Másodszor, bár lehetséges egy kisalkalmazást meggátolni abban, hogy a képernyőre íráson kívül bármi mást is csináljon, sok kisalkalmazásnak ennél több hatalomra van szüksége, így amikor további jogosultságokat kérnek, akkor a felhasználók morogva (vagy jóhiszeműen) bár, de megadják azokat. Például a kisalkalmazásoknak esetleg átmeneti állományokat kell írniuk, így a felhasználók elérést adnak nekik a */tmp* könyvtárhoz, azt gondolva, hogy nincs ott semmi fontos. Sajnos a legtöbb szövegszerkesztő az éppen szerkesztés alatt álló dokumentumok és elektronikus levelek átmeneti változatait ott tartja, így a rosszindulatú alkalmazások lemásolhatják azokat, és megpróbálhatják átküldeni a hálózaton. Természetesen lehetséges letiltani a kisalkalmazások hozzáférését a hálózathoz, de közülük sok nem fog így működni, így meg kell nekik adni ezt a hatalmat is.

De még abban a valószínűtlen esetben is, amikor a kisalkalmazásoknak nincs engedélyezve a hozzáférés a hálózathoz, még akkor is képesek lehetnek információt átvinni **rejtett csatornákat (covert channels)** (Lampson, 1973) használva. Például valamely információ megszerzése után egy kisalkalmazás egy bitfolyamot alkothat a rendszer valós idejű óráját használva. Logikai 1 küldéséhez ΔT ideig erősen számol, 0 küldéséhez pedig ΔT ideig várakozik.

Hogy ezt az információt begyűjtse, a kisalkalmazás tulajdonosa egy összeköttetést hozhat létre az ügyfél gépéhez, hogy néhány nyilvános Háló-oldalát olvassa, vagy FTP-vel letöltve néhány nyilvános állományát. Gondosan megfigyelve a bejövő adatsebességet, a kisalkalmazás tulajdonosa láthatja, hogy a kisalkalmazás számol-e (és ezáltal lelassítja a megfigyelt kimeneti folyamot) vagy pihen. Természetesen ez a csatorna zajos, de ezt a szokásos technikákkal kezelni lehet. A folyamot jelzőbájtokkal elválasztva keretekre lehet tagolni, az egyes keretek egy erős hibajavító kódot használhatnak, és minden keretet kétszer vagy háromszor el lehet küldeni. Sok más rejtett csatorna létezik, és különösen nehéz az összeset felderíteni és letiltani. A Java biztonsági problémáiról lásd (Dean és Wallach, 1995) műveit.

Röviden, a Java sok új lehetőséget és esélyt ad a Világhálónak. Lehetővé teszi, hogy a Háló-oldalak interaktívak legyenek, animációt és hangot tartalmazzanak. Azt is lehetővé teszi, hogy a böngészők végtelenül bővíthetők legyenek. Viszont a Java modellje a letölthető kisalkalmazásokról bevezet néhány súlyos új biztonsági problémát is, amelyeket még nem teljesen oldottak meg.

7.6.5. Információ megtalálása a Hálón

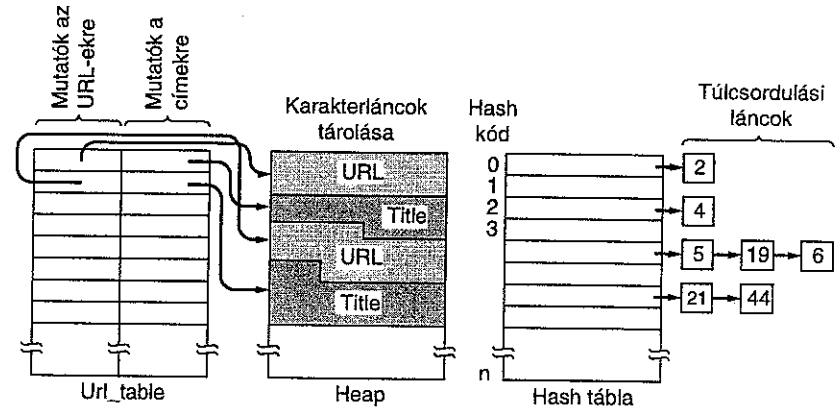
Bár a Háló hatalmas mennyiségű információt tartalmaz, nem mindig könnyű a megfelelőt megtalálni. Hogy az emberek számára megkönnyítsék a számukra hasznos oldalakat, számos kutató írt a Háló különféle szempontok szerinti indexelésére programot. Ezek közül néhány olyan sikeres lett, hogy üzleti alapra helyezték át a működésüket. Azokat a programokat, amelyek végigkeresik a Hálót, néha **keresőgépeknek (search engines), pókoknak (spiders), csúszómászóknak (crawlers), férgeknek (worms)** vagy **okos robotoknak (knowbots, knowledge robots)** nevezik. Ebben a szakaszban egy rövid bevezetőt adunk ehhez a témához. A további tudnivalókat lásd (Pinkerton, 1994; McBryan, 1994) műveiben.

Bár a Háló hatalmas, mégis a legalapvetőbb elemeire szűkítve a Háló egy nagy gráf, ahol az oldalak a csomópontok és az élőkapcsok az élek. Egy gráf minden csomópontjának meglátogatására jól ismert algoritmusok léteznek. Ami a Háló indexelését bonyolulttá teszi, az a hatalmas adatmennyiség, amelyet kezelni kell, és az a tény, hogy ez állandóan változik.

Kezdjük a tárgyalásunkat egy egyszerű céllal: a Háló-oldalak címében levő összes kulcsszó indexelésével. Az algoritmusunkhoz három adatstruktúrára lesz szükségünk. Először is kell egy nagy lineáris tömb, az *url_table*, amely több millió bejegyzést tartalmaz, végső soron minden Háló-oldalhoz egyet. Ezt a virtuális memóriában kell tartani, hogy a nem sűrűn használt oldalakat automatikusan ki lehessen lapozni a lemezre. Minden bejegyzés két mutatót tartalmaz, az egyiket az oldal URL-jére, a másikat a címére. Mind az URL, mind a cím egy változó hosszúságú karakterlánc, és a heapben tárolható. (A heap a virtuális memória egy nagy, szervezetlen darabja, amelyhez karakterláncokat lehet hozzáfűzni.) A heap a második adatstruktúránk.

A harmadik adatstruktúra egy n bejegyzésből álló hash tábla. Ezt a következőképpen használják: Bármely URL-re le lehet futtatni a hash függvényt, és az egy n-nél kisebb nem negatív egész számot fog eredményül adni. Mindegyik URL, amelyre a hash függvény k-t ad eredményül, egy láncolt listába van összekapcsolva, amely a hash tábla k. bejegyzésénél kezdődik. Ahányszor egy URL bekerül az *url_table*-be, bekerül a hash táblába is. A hash tábla fő célja, hogy egy URL-ből kiindulva gyorsan meg tudjuk állapítani, hogy az szerepel-e már az *url_table*-ben. Ezt a három adatstruktúrát a 7.75. ábra mutatja be.

Az index felépítése két lépésből áll: a keresésből és az indexelésből. Kezdjük egy egyszerű géppel, amely a keresést végzi. A keresőgép szíve egy rekurzív eljárás, a *process_url*, amely egy URL karakterláncot vár a bemenetén. A következőképpen



7.75. ábra. *Egy egyszerű keresőgépben használt adatstruktúrák*

működik: Először végrehajtja a hash függvényt az URL-en, hogy lássa, jelen van-e már az *url_table*-ben., Ha igen, akkor azonnal visszatér. Minden URL-t csak egyszer dolgoz fel.

Ha az URL még nem ismert, akkor elhozza az oldalt. Ezután az URL-t és a címet bemásolja a heapbe, és az erre a két karakterláncra mutató mutatókat elhelyezi az *url_table*-ben. Az URL-t a hash táblába is beírja.

Végül a *process_url* kigyűjti az összes élőkapcsot az oldalról, és meghívja a *process_url*-t minden élőkapocsra, az élőkapocs URL-jét átadva a bemeneti paraméternek.

Hogy a keresőgépet lefuttassák, a *process_url*-t valamely kezdeti URL-lel hívják meg. Amikor visszatér, az összes, arról az URL-ról elérhető oldal bekerült az *url_table*-be és a keresési lépés véget ért.

Bár ez a terv egyszerű és elméletileg helyes, van egy súlyos problémája a Hálóhoz hasonlóan nagyméretű rendszerek esetén. A gond az, hogy ez az algoritmus mélységi bejárást végez, és annyiszor fogja meghívni önmagát, amilyen hosszú a leghosszabb, önmagába vissza nem térő útja. Senki nem tudja, hogy milyen hosszú ez az út, de valószínűleg több ezer élőkapocsra tehető. Ennek következményeként bármely keresőgép, amely ezt a mélységi bejárást használja, valószínűleg veremtúlcsordulással le fog állni, mielőtt végezne.

A gyakorlatban a keresőgépek először minden beolvasott oldalon összegyűjtik az élőkapcsokat, kiveszik azokat, amelyeket már feldolgoztak, és a maradékot elmentik. Ezután a Hálót szélességében járják be, vagyis egy oldalon minden élőkapcsot követnek, és a hivatkozott oldalakról minden élőkapcsot begyűjtenek, de nem a megszerzés sorrendjében járják be azokat.

A második lépésben történik a kulcsszavak indexelése. Az indexelési eljárás lineárisan halad végig az *url_table*-n, sorban minden bejegyzést feldolgozva. Minden bejegyzésnél megvizsgálja a címet és kiválasztja a tiltólistán nem szereplő szavakat. (A tiltólista megakadályozza a névelők, a kötőszavak, az igekötők és más, sok találatot produkáló, de keveset érő szavak indexelését.) Minden kiválasztott szóhoz egy, a szóból és az *url_table* aktuális bejegyzésének számából álló sort ír egy állományba. Amikor az egész táblázatot végignézte, akkor az állományt névsorba rendezik.

Az indexet lemezen kell tárolni, és a következőképpen használható. A felhasználó kitölt egy űrlapot, és felsorol egy vagy több kulcsszavat, majd a SUBMIT gombra kattint. Ezen cselekvés hatására egy *POST* kérést intéznek egy CGI scripthez, amely azon a gépen fut, ahol az index helyet kapott. Ez a script (vagy inkább program) ezután kikeresi az indexben szereplő kulcsszavakat, hogy mindegyikhez megtudja az ehhez tartozó *url_table* indexek halmazát. Ha a felhasználó a kulcsszavak logikai ÉS kapcsolatára kíváncsi, akkor a halmazok metszetét számítja ki. Ha a logikai VAGY kapcsolatot óhajtja, akkor a halmazok unióját számítja ki.

A script ezután megcímzi az *url_table*-t, így megtalálja az összes címet és URL-t. Ezekből azután előállít egy Háló-oldalt, és a *POST*-ra válaszként visszaküldi a felhasználónak. A böngésző ezután megjeleníti az oldalt, és a felhasználónak bármelyik érdekesnek tűnő tételre lehetősége van rákattintani.

Könnyűnek hangzik? Pedig nem az. Bármely, a gyakorlatban is működő rendszerben a következő problémákat kell megoldani:

1. Egyes URL-ek elavultak (azaz olyan oldalakra mutat, amelyek már nem léteznek).

2. Néhány gép ideiglenesen elérhetetlen lesz.

3. A kezdő URL-ről nem minden oldal érhető el.

4. Néhány oldalt csak aktív térképeken keresztül lehet elérni.

6. Néhány dokumentumot nem lehet indexelni (mint pl. a hangállományokat).

7. Nem minden dokumentumnak van (hasznos) címe.

8. A keresőgép kifogyhat a memóriából vagy a lemezterületből.

9. Az egész folyamat túl sok időt vehet igénybe.

Az elavult URL-ek időveszteséget okoznak, de leginkább csak kellemetlenséget jelentenek, mert az a kiszolgáló, amelyiken lenniük kellene, azonnal válaszol egy hibakóddal. Ezzel ellentétben, amikor a kiszolgáló tönkremegy, akkor a keresőgép csak egy hosszú késleltetést érzékel a TCP összeköttetés felépülésekor. Hogy meggátoljuk, hogy végtelen ideig függőben maradjon, időzítéssel kell rendelkeznie. Ha az időzítés túl rövid, akkor érvényes URL-ek fognak kimaradni. Ha túl hosszú, akkor a keresés észrevehetően le fog lassulni.

A kezdő URL kiválasztása nyilvánvalóan fontos kérdés. Ha a keresőgép egy asztrofizikus honlapján kezd, akkor bizonyára mindent meg fog találni a csillagászatról, a fizikáról, a kémiáról és az űrkutatásról, de teljesen kihagyhatja az állatorvoslásról, a középkori angol nyelvről, vagy a rock and rollról szóló oldalakat. Ezeknek a halmazoknak esetleg egyszerűen nem lesz közös elemük. Egy megoldás az, hogy olyan sok URL-t gyűjtsünk össze, amennyit csak lehet, és mindet használjuk kezdőoldalnak. A kezdő URL-eket a USENET hírek cikkeiből és az *url_table* múlt heti változatából lehet összegyűjteni, mivel ezek az oldalak a közelmúltban változhattak. (Pl. egy asztrofizikus és egy állatorvos összeházasodott, és ünnepélyesen felfrissítették a honlapjaikat az egymásra hivatkozásokkal.)

A szöveg indexelése jól működik, de a gond az, hogy egyre nagyobb mértékben tartalmaznak az oldalak szövegen kívül mást is, képeket, hangokat, mozgóképeket. Erre egy megközelítés az, hogy minden újonnan talált URL-t ki kell próbálni a *HEAD* metódussal annak érdekében, hogy a MIME fejrészt kapjuk vissza. Semmit nem vizsgálunk meg, ami nem *text* típusú.

Az összes Háló-oldalak mintegy 20 százalékának nincs címe, és még ha van is, soknál ez kvázi használhatatlan („Joe oldala"). Az alapvető indexhez képest nagy előrelépés, hogy ne csak a címeket vegyük be, hanem az egész hiperszöveget is. Ezzel a megközelítéssel, amikor egy oldalt végignézünk, az összes élőkapcsot is feljegyezzük, azzal együtt, hogy hol voltak és mire hivatkoztak. Miután a keresési lépés lezárult, a hiperszavakat is indexelni lehet.

Még ennél is ambiciózusabb az, hogy minden oldal összes fontos szavát indexel-

jük. Hogy eldöntsük, melyek a fontos szavak, a tiltólistán nem szereplő szavak gyakoriságát Háló-oldalanként meg lehet számolni. A 10 vagy 20 leggyakoribb szót várhatóan érdemes indexelni. Végül is, ha a „máj" szó a leggyakoribb az oldalon, van esély arra, hogy az oldal érdekelni fogja az epével foglalkozó sebészeket (vagy a szakácsokat). Néhány keresőgép (mint a Lycos) ezt a stratégiát használja.

Végül, a keresőgép kifuthat a memóriából vagy az időből. Egy megoldási kísérlet az, hogy óvatosabban tervezzük újra az algoritmusokat. Egy teljesen más megközelítés az, amit a Harvest használ, amely másnak adja át a munkát (Bowman és mások, 1994, 1996). Pontosabban, a Harvest egy programot biztosít, amelyet a résztvevő gépeken futtatni kell. Ez a program helyileg végzi el a keresést, és a végleges helyi indexet visszaküldi. A központi állomásnál az összes helyi indexet egy főindexbe fésülik össze. Ez a megközelítés nagyságrendekkel csökkenti a szükséges memória, CPU idő és hálózati sávszélesség mennyiségét, de megvan az a nagy hátránya, hogy minden Háló-kiszolgálónak idegen szoftvert futtatva kell együttműködnie. A vírusok és férgek potenciális problémáját figyelembe véve nem meglepő, hogy amikor egy rendszergazdához a következő kérést intézik: „Futtatná kérem ezt a programot nekem a saját gépén?", sok nem fogja ezt teljesíteni.

Egyetlen kis kérés következik. Bár egy keresőgép megírása egyszerűnek tűnik, egy hibás keresőgép pusztulást hozhat a hálózatra, ha nagyszámú hamis kérést generál, és ezáltal nemcsak a sávszélesség vész kárba, hanem sok kiszolgálót is térdre kényszeríthet a terhelés. Ha nem tud ellenállni annak a kísértésnek, hogy saját keresőgépet írjon, akkor – ahogy azt a hálózati etikett megköveteli – a keresőgép teljes hibamentességéig annak működését csak a saját DNS tartományára korlátozza.

7.7. Multimédia

A mutimédia a számítógépes hálózatok Szent Grálja. Amikor ez szóba kerül, mind a kockafejűeknek, mind az öltönyös alakoknak mintegy varázsütésre csorogni kezd a nyáluk. Az előzőek hatalmas műszaki kihívásokat látnak abban, hogy minden otthonban lehetővé tegyék az (interaktív) hálózati videózást. Az utóbbiak ugyanilyen hatalmas profitot. Egy hálózatokról szóló könyv nem lehet teljes anélkül, hogy legalább egy bevezetőt ne tartalmazna ehhez a témához. Mivel a mi könyvünk már enélkül is hosszú, ezért ez a bevezető szükségképpen rövid lesz. Erről az izgalmas és esetleg jövedelmező témáról a további tudnivalókat lásd: (Buford, 1994; Deloddere és mások, 1994; Dixit és Skelly, 1995; Fluckiger, 1995; Minoli, 1995; Steinmetz és Nahrstedt, 1995).

Szó szerint értelmezve, a multimédia két vagy több médiát jelent. Ha ennek a könyvnek a kiadója csatlakozni akarna a multimédia körüli jelenlegi felhajtáshoz, úgy hirdethetné ezt a könyvet, mint amely multimédia technológiát használ. Végül is két médiát tartalmaz: szöveget és grafikát (az ábrákat). Ennek ellenére, amikor a legtöbb ember multimédiáról beszél, általában két vagy több **folytonos média (continuous media)** kombinációját érti alatta, vagyis olyanokét, amelyeket valamely jól meghatározott időintervallum alatt kell lejátszani, rendszerint némi felhasználói beavatkozással. A gyakorlatban a két média rendszerint az audio és a video, vagyis a hang és a mozgókép. Ezért a tárgyalást egy bevezetéssel kezdjük a hang- és képtechnikába. Ezután ezeket kombinálni fogjuk, és áttérünk a valódi multimédia rendszerekre, köztük a hálózati videózásra és az Internet multimédia rendszerére, az MBone-ra.

7.7.1. A hang (audio)

Egy hanghullám egy egydimenziós akusztikai (nyomás)hullám. Amikor egy akusztikai hullám belép a fülbe, a dobhártya elkezd rezegni, azt okozva, hogy a belsőfül kicsiny csontjai is vele rezegnek, és ingerlöketeket küldenek az agyba. Ezeket a löketeket a hallgató mint hangot érzékeli. Hasonló módon, amikor egy akusztikai hullám elér egy mikrofont, a mikrofon egy villamos jelet állít elő, amely a hang amplitúdóját mint az idő függvényét jellemzi. Az ilyen hangjelek reprezentációja, feldolgozása, tárolása és továbbítása nagy részét képezi a multimédia rendszerek tanulmányozásának.

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 hallása logaritmikus, így két A és B amplitúdójú hang arányát szokásosan **dB-ben (decibelben)** fejezik ki, a következő képlet szerint:

$$\mathrm{dB} = 20 \log_{10}(A/B)$$

Ha egy 1 kHz-es szinuszhullám hallhatóságának alsó határát (amely kb. 20 μPa nyomásnak felel meg) 0 dB-nek definiáljuk, akkor egy szokásos beszélgetés 50 dB körül van, a fájdalomküszöb pedig 120 dB, amely kb. egymilliószoros dinamikatartományt jelent. Hogy minden félreértést elkerüljünk, a fenti összefüggésben A és B *amplitúdók*. Ha a teljesítményszintet használtuk volna, amely az amplitúdó négyzetével arányos, a logaritmus együtthatója 20 helyett 10 lett volna.

A fül meglepően érzékeny az akár csak pár ezredmásodpercig tartó hangváltozásokra is. A szem ezzel ellentétben nem vesz észre a fényszintben olyan változásokat, amelyek csak pár ezredmásodpercig tartanak. Ennek a megfigyelésnek az eredménye, hogy egy multimédia átvitelnél bekövetkező pár ezredmásodperces dzsitter az észlelt hangminőséget jobban befolyásolja, mint az észlelt képminőséget.

A hanghullámokat egy **ADC (Analog Digital Converter – analóg-digitális átalakító)** segítségével lehet digitális formára alakítani. Egy ADC villamos feszültséget kap a bemenetén, és a kimenetén bináris számokat állít elő. A 7.76.(a) ábrán látható egy szinuszhullám példája. Hogy digitálisan jelképezhessük ezt a jelet, minden ΔT időközönként mintát vehetünk belőle, ahogy a 7.76.(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 (l. 2. fejezet) elegendő $2f$ frekvenciával mintákat venni. Gyakrabban mintavételezni nincs értelme, mivel azok a magasabb frekvenciák, amelyeket egy ilyen mintavételezés észlelni tudna, nincsenek jelen.

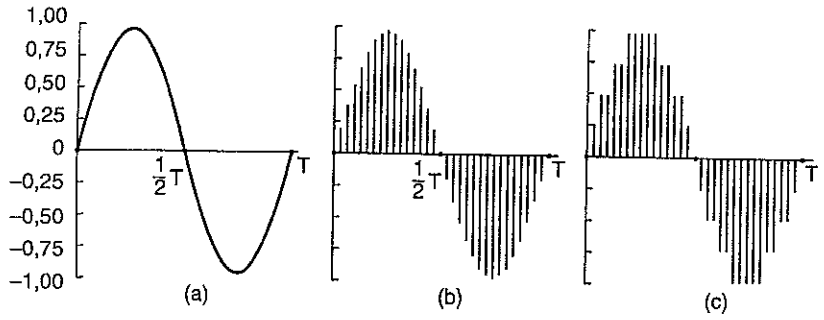
A digitális minták soha nem teljesen pontosak. A 7.76.(c) ábra hárombites mintái csak nyolc értéket tesznek lehetővé –1,00 és +1,00 közt, 0,25-ös lépésekben. Egy 8 bites minta 256 különböző értéket tenne lehetővé. Egy 16 bites minta 65 536 különböző értéket tenne lehetővé. A mintánkénti véges számú bit által behozott hibát **kvantálási zajnak (quantization noise)** nevezik. Ha ez túl nagy, a fül ezt észleli.

A mintavételezett hangok két jól ismert példája a telefon és a hang-CD-k. A pulzuskód-moduláció, amelyet a telefonrendszerben használnak, 7 (Észak-Amerika és Japán) vagy 8 (Európa) bites mintákkal dolgozik, és másodpercenként 8000 mintát vesz. Ez a rendszer 56 000 vagy 64 000 b/s adatsebességet eredményez. Másodpercenként mindössze 8000 minta esetén a 4 kHz fölötti hangok elvesznek.

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 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 a legkisebb hallható hang egységével mérve 1 millió körül van. Ezért mintánként csak 16 bit használata okoz némi kvantálási zajt (bár nincs lefedve a teljes dinamikatartomány, a CD-knek nem szabad fájdalmat okozniuk). Másodpercenként 44 100 16 bites minta esetén egy hang-CD-nek 705,6 Kb/s (mono) vagy 1,411 Mb/s (sztereo) sávszélességre van szüksége. Bár ez kisebb, mint amire a mozgóképnek szüksége van, a tömörítetlen CD minőségű sztereohang átvitele majdnem egy teljes T1 csatornát igényel.

A digitalizált hangot a számítógépek szoftverrel könnyen feldolgozhatják. Több tucat olyan program létezik személyi számítógépekre, amely lehetővé teszi, hogy a felhasználók különböző forrásokból származó hanghullámokat felvegyenek, megjelenítsenek, szerkesszenek, keverjenek és eltároljanak. Tulajdonképpen manapság minden professzionális hangfelvétel és szerkesztés digitálisan történik.

Sok hangszernek már digitális interfésze is van. Amikor az első digitális hangszerek megjelentek, mindegyiknek saját interfésze volt, de egy idő után kifejlesztettek egy szabványt, a **MIDI-t (Musical Instruments Digital Interface – hangszerek digitális interfésze),** amelyet gyakorlatilag az egész zeneipar magáévá tett. Ez a szab



7.76. ábra. *(a) Egy szinuszhullám. (b) A szinuszhullám mintavételezettje. (c) A minták 3 bitre történt kvantáltja*

vány a csatlakozót, a kábelt és az üzenetformátumot határozza meg. Minden MIDI üzenet egy státusbájtból áll, amelyet nulla vagy több adatbájt követ. Egy MIDI üzenet egy zeneileg jelentéssel bíró eseményt fed le. Ilyen események egy billentyű leütése, egy csúszka odébbtolása, vagy egy lábpedál felengedése. A státusbájt jelzi az eseményt, és az adatbájtok adják meg a paramétereket, hogy például melyik billentyűt és milyen sebességgel nyomták le.

Minden hangszerhez hozzárendeltek egy MIDI kódot. Például a versenyzongoráé a 0-s, a marimbáé a 12-es, és a hegedűé a 40-es. Ez azért kell, hogy egy fuvolára írt concertót ne tubán játsszanak vissza. A definiált „hangszerek" száma 127, bár ezek közül nem mind hangszer, hanem különleges effekt, mint a madárcsicsergés, a helikopter zaja, és az a konzervtaps, amely sok tévéműsort kísér.

Minden MIDI rendszer szíve egy szintetizátor (gyakran egy számítógép), amely üzeneteket fogad, és azokból zenét állít elő. A szintetizátor ismeri mind a 127 hangszert, így egy középső C hang esetén más teljesítményspektrumot állít elő, ha az trombitán szólal meg, mint ha xilofonon. A zene MIDI használatával történő átvitelének előnye a digitalizált hullámforma elküldésével szemben az, hogy a szükséges sávszélesség nagyban lecsökken, gyakran akár az ezredrészére. A MIDI hátránya, hogy a vevőnek rendelkeznie kell egy MIDI szintetizátorral, hogy újra előállítsa a zenét, és a különböző szintetizátorok némileg eltérő módon adhatják azt vissza.

A zene természetesen az általános hang egy speciális, ám fontos esete. Egy másik fontos speciális eset a beszéd. Az emberi beszéd általában 600 és 6000 Hz közé esik. A beszéd magánhangzókból és mássalhangzókból áll, amelyeknek mások a tulajdonságaik. A magánhangzók akkor képződnek, amikor a hang útjában nem áll semmi, és az rezonanciákat állít elő, amelyek alapfrekvenciája a beszélő hangképző rendszerének méretétől és alakjától, valamint a beszélő nyelvének és állkapcsának helyzetétől függ. Ezek a hangok majdhogynem periodikusak úgy 30 ezredmásodperces intervallumokra nézve. A mássalhangzók akkor képződnek, amikor a hang útja részlegesen el van torlaszolva. Ezek a hangok kevésbé szabályosak, mint a magánhangzók.

Néhány beszédgeneráló és -átvivő rendszer a hangképző rendszer modelljét használja, hogy a beszédet mindössze pár paraméterre sűrítse össze (a különféle üregek méretére és alakjára), a beszéd hullámformájának egyszerű mintavételezése helyett.

7.7.2. A mozgókép (video)

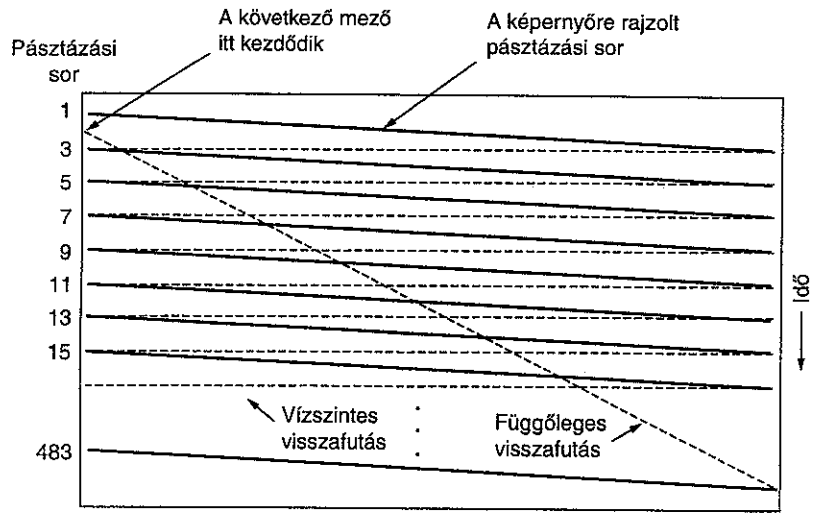
Az emberi szemnek megvan az a tulajdonsága, hogy amikor egy képet villantanak a retinára, az néhány ezredmásodpercig ottmarad, mielőtt elenyészne. Ha képek sorozatát 50 kép/s sebességnél gyorsabban villantjuk fel, a szem nem fogja észrevenni, hogy valójában különálló képeket néz. Minden videós (azaz televíziós) rendszer ezt az elvet használja ki mozgóképek előállításához.

Analóg rendszerek

Hogy megértsük a videórendszereket, legjobb, ha az egyszerű régimódi fekete-fehér televízióval kezdjük. A kamera, hogy az előtte levő kétdimenziós képet az idő függvényében mint egydimenziós feszültséget tudja ábrázolni, egy elektronsugarat futtat végig gyorsan keresztbe és fokozatosan lefele haladva a képen, mérve a fényintenzitás értékeit. Egy letapogatás (vagy más néven **keret, frame**) végén a sugár visszafut a kezdéshez. Ez az intenzitás, mint az idő függvénye, műsorszórásra kerül, és a vevők megismétlik a letapogatási folyamatot, hogy helyreállítsák a képet. A mind a kamera, mind a vevő által használt pásztázási minta a 7.77. ábrán látható. (Egyébként a CCD kamerák a letapogatás helyett inkább integrálnak, de néhány kamera és az összes monitor a letapogatást használja.)

A pontos letapogatási paraméterek országról országra változnak. Az Észak- és Dél-Amerikában, valamint Japánban használt rendszer 525 sorral, 4:3 vízszintes:függőleges aránnyal, és 30 kép/s sebességgel dolgozik. Az európai rendszer 625 sorral, ugyanazzal a 4:3 torzítási hányadossal, és 25 kép/s sebességgel dolgozik. Mindkét rendszerben a legalsó és legfelső pár sor nem jelenik meg (hogy az eredeti kerek katódsugárcsöveken is közelítsék a téglalap alakú képeket). Az NTSC 525 pásztázási sorából csak 483 (és a PAL/SECAM 625 pásztázási sorából csak 576) jelenik meg. A sugarat a függőleges visszafutás alatt kikapcsolják, így sok (leginkább európai) állomás ezekben az időközökben szórja a Teletext adását, amely egy szövegoldalakból álló, híreket, időjárást, sporthíreket, részvényárfolyamokat stb. tartalmazó rendszer.

Bár a 25 kép/s elegendő ahhoz, hogy folytonos mozgásérzetet adjon vissza, ennél a képsebességnél sok ember, különösen az idősebbek, villogónak fogja látni a képet,



7.77. ábra. *Az NTSC videóhoz és televízióhoz használt pásztázási minta*

mivel a régi kép még azelőtt eltűnt a retináról, mielőtt az új kép megérkezett volna. A képsebesség növelése helyett, amely még többet használna el az amúgy is szűkös sávszélességből, más megközelítést alkalmaznak. A pásztázási sorok sorrendben történő megjelenítése helyett először csak a páratlan sorokat, majd csak a páros sorokat jelenítik meg. Mindegyik félképet egy **mezőnek (field)** nevezik. A kísérletek azt mutatták, hogy bár az emberek észlelik a villogást 25 kép/s esetén, viszont 50 félkép/s esetén nem. Ezt a technikát **sorugrásos megjelenítésnek (interlacing)** nevezik. A nem sorugrásos televíziót vagy videót **progresszívnek (progressive)** nevezik.

A színes video ugyanazt a letapogatási mintát használja, mint a monokróm (fekete-fehér), csak egy helyett három, együtt mozgó sugárral jelenítik meg a képet. Mindhárom elsődleges additív színhez (piros, zöld és kék, RGB) egy sugarat használnak. Ez a technika működőképes, mert bármely szín előállítható a piros, zöld és kék megfelelő intenzitású lineáris szuperpozíciójából. *(A ford. megj: ez nincs így, bizonyos színekhez negatív piros érték tartozna.)* Ám az egyetlen csatornán való átvitelhez a három színjelet egyetlen **kompozit (composite)** jelbe kell összevonni.

Amikor a színes televíziót feltalálták, a színmegjelenítésnek technikailag számos megoldása volt lehetséges, és a különböző országok más-más döntéseket hoztak, ami aztán olyan rendszerekhez vezetett, amelyek még ma is inkompatíbilisak egymással. (Ezeknek a döntéseknek semmi közük a VHS-Betamax-P2000 rögzítési eljárások közti inkompatibilitáshoz.) Minden országban politikai követelmény volt, hogy a színesben átvitt programoknak foghatónak kellett lenniük a meglevő fekete-fehér televíziókészülékeken is. Következésképpen a legegyszerűbb megoldás, az RGB jelek külön való kódolása nem volt elfogadható, továbbá az RGB nem is a leghatékonyabb módszer.

Az első színes rendszert az Egyesült Államokban a National Television Standards Committee (Nemzeti Televíziós Szabványügyi Bizottság) szabványosította, ahonnan a szabvány rövidítése, az **NTSC** is származik. A színes televíziózást Európában néhány évvel később vezették be, amikorra a technológia már felfejlődött, és ez kisebb zajérzékenységű és jobb színeket adó rendszereket eredményezett. Ezek a Franciaországban és Kelet-Európában használt **SECAM (SEquentiel Couleur Avec Memoire)** és az Európa többi részén használt **PAL (Phase Alternating Line)** rendszerek lettek. A PAL/SECAM és az NTSC színminősége közti különbség eredményezte azt a szakmában elterjedt viccet, miszerint az NTSC valójában annyit tesz: Never Twice the Same Color (soha nem lesz kétszer ugyanaz a szín).

Hogy a színes adásokat a fekete-fehér vevőkön is nézhetővé tegyék, mindhárom rendszer az RGB jeleket lineárisan egy **luminancia** (fényesség) és két **krominancia** (szín) jelbe kombinálja, bár ezen jelek előállításához más együtthatókat használnak. Érdekes, hogy a szem sokkal érzékenyebb a luminanciajelre, mint a krominanciajelekre, így az utóbbiakat nem kell olyan pontosan átvinni. Következésképpen a luminanciajelet ugyanazon a frekvencián lehet szórni, mint a régi fekete-fehér jelet, így azt egy régi fekete-fehér televíziókészülékkel venni lehet. A két krominanciajelet nagyobb frekvenciákon, és szűkebb sávban szórják. Néhány televíziókészüléknek van fényesség, színárnyalat és telítettség (vagy fényesség, árnyalat és szín) feliratú kezelőszerve ezen három jel külön-külön állításához. A luminancia és krominancia elvének megértése szükséges a képtömörítés megértéséhez.

Az elmúlt pár évben nagy érdeklődés mutatkozott a **HDTV (High Definition TeleVision, nagyfelbontású TV)** iránt, amely a pásztázási sorok kb. megduplázásával élesebb képet nyújt. Az Egyesült Államok, Európa és Japán mind kifejlesztettek egymástól eltérő, és kölcsönösen inkompatíbilis HDTV rendszereket. A HDTV pásztázást, luminanciát, krominanciát stb. érintő alapelvei hasonlatosak a meglevő rendszerekéhez, viszont a torzítási hányadosa mindháromnak 4:3 helyett 16:9, hogy jobban illeszkedjenek a mozifilmekhez használt formátumhoz, amelyeket 35 mm-es filmre rögzítenek.

A televíziós technológiába bevezetőt ad (Buford, 1994).

Digitális rendszerek

Egy digitális mozgókép legegyszerűbb reprezentációja képek sorozata, amelyek mindegyike képelemek (pixelek, képpontok) téglalap alakú rácsából áll. Minden képpont egy bit lehet, amely vagy fehéret, vagy feketét jelképez. Egy ilyen rendszer minősége hasonló ahhoz, amit színes fénykép faxolásakor kapunk – szörnyű. (Próbálja ki, ha teheti; egyébként fénymásoljon le egy színes fényképet egy olyan másológépen, amely nem raszterizál.)

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 a jó rendszerek az RGB színek mindegyikéhez 8 bitet használnak, bár átvitelhez ezeket majdnem minden rendszer a kompozit videojelbe keveri össze. Bár képpontonként 24 bit használata a színek számát körülbelül 16 millióra korlátozza, az emberi szem már ennyi színt sem tud megkülönböztetni, nemhogy többet. A digitális színes képeket három letapogató sugárral állítják elő, színenként eggyel. A geometria ugyanaz, mint a 7.77. ábra analóg rendszerénél, kivéve, hogy a folytonos pásztázási vonalak helyére most diszkrét képelemekből álló takaros sorok kerültek.

Hogy sima mozgást adjon, a digitális videónak az analóg videóhoz hasonlóan legalább 25 kép/s sebességgel kell a képeket megjelenítenie. Ám mivel a jó minőségű számítógépes monitorok a memóriában tárolt kép alapján másodpercenként 75-ször vagy többször újrapásztázzák a képernyőt, a sorugrásos megjelenítésre nincs szükség, és azt rendszerint nem is használják. Ugyanannak a képnek háromszor egymás után történő újrarajzolása elég ahhoz, hogy kiküszöbölje a villódzást.

Más szavakkal, a mozgás simasága a *különböző* képek másodpercenkénti számától függ, a villódzás pedig a képernyő másodpercenkénti újrarajzolásainak számától függ. Ez a két paraméter nem ugyanaz. Egy másodpercenként 20-szor újrarajzolt állókép nem fog darabos mozgásúnak tűnni, de villogni fog, mert az egyik kép gyorsabban enyészik el a retináról, mint ahogy a következő megjelenne. Egy mozifilm, amely másodpercenként 20 különböző képet mutat, amelyek mindegyikét négyszer újrarajzolják, nem fog villogni, de a mozgás darabosnak fog tűnni.

Ennek a két paraméternek a jelentősége akkor válik világossá, amikor számba vesszük a digitális mozgókép hálózaton keresztül való továbbításának sávszélességigényét. A mostani számítógép-monitorok mind 4:3-mas torzítási hányadost használnak,

hogy a fogyasztói televíziós piac számára tömeggyártásban olcsón előállított képcsöveket használhassanak. A szokásos beállítások a 640×480 (VGA), a 800×600 (SVGA) és az 1024×768 (XGA). Egy XGA megjelenítőt, amely képpontonként 24 bites és másodpercenként 25 képet jelenít meg, 472 Mb/s-mal kell táplálni. Erre még az OC–9 sem egészen megfelelő, és nincs igazán napirenden az, hogy minden otthonba eljuttassanak egy OC–9-es SONET hordozót. Ennek a sebességnek a megduplázása, a villódzás elkerülése céljából, még kevésbé vonzó. Egy jobb megoldás az, hogy 25 képet vigyünk át másodpercenként, a számítógép tárolja el mindegyiket, és kétszer rajzolja ki azokat. A televíziós műsorszórás nem használja ezt a stratégiát, mivel a televíziókészülékekben nincs memória, és egyébként is, analóg jeleket csak úgy lehet RAM-ban tárolni, ha előtte azokat digitális formára alakítjuk, amely külön hardvert igényel. Következésképpen a sorugrásos megjelenítésre a televíziós műsorszórásban szükség van, de a digitális videónál nincs.

7.7.3. Adattömörítés

Mostanra már nyilvánvaló kell legyen, hogy a multimédia anyag tömörítetlen formában történő továbbítása szóba sem jöhet. Az egyetlen remény az, hogy nagyfokú tömörítésre van lehetőség. Szerencsére az utóbbi pár évtizedben végzett kiterjedt kutatómunka sok olyan tömörítési technikához és algoritmushoz vezetett, amelyek megvalósíthatóvá teszik a multimédia átvitelt. Ebben a szakaszban néhány, a multimédia adatok tömörítésére használt eljárást fogunk megvizsgálni, különösen azokat, amelyek képek tömörítésére szolgálnak. A részleteket lásd (Fluckiger, 1995; Steinmetz és Nahrstedt, 1995) műveiben.

Minden tömörítő rendszernek két algoritmusra van szüksége: az egyik a forrásnál tömöríti össze az adatokat, a másik pedig a célban. Az irodalomban ezekre az algoritmusokra mint **kódoló (encoding)** és **dekódoló (decoding)** algoritmusokra hivatkoznak. Mi is ezt a terminológiát fogjuk használni.

Ezek az algoritmusok bizonyos aszimmetriával rendelkeznek, amelyet fontos megérteni. Először is, sok alkalmazásnál egy multimédia dokumentumot, mondjuk egy filmet, egyszer fognak kódolni (amikor a multimédia kiszolgálón tárolásra kerül), de sok ezerszer fognak dekódolni (amikor az ügyfelek megnézik azt). 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. Végül is egy multimédia kiszolgáló üzemeltetője igenis hajlandó lehet pár hétre egy párhuzamos szuperszámítógépet bérelni, hogy az egész videokönyvtárát lekódolja, de az, hogy a felhasználóknak két órára szuperszámítógépeket kell bérelniük egy film megtekintéséhez, nem számíthat nagy sikerre. Sok gyakorlati tömörítő rendszer sokat megtesz azért, hogy a dekódolást gyorssá és egyszerűvé tegye, még annak az árán is, hogy a kódolás lassú és komplikált lesz.

Másfelől a valós idejű multimédiához, mint amilyen a videokonferencia, 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 elfogadhatóan kisebb tömörítéssel.

Egy másik aszimmetria, hogy a kódolás/dekódolás folyamatnak nem kell visszafordíthatónak lennie. Egy állomány 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. Rendszerint elfogadható az, hogy a videojel a kódolás és dekódolás után némileg eltérjen az eredetitől. 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 hatalmas nyereségre tehetünk szert az elérhető tömörítési arányban.

Entrópiakódolás

A tömörítési eljárásoknak két általános csoportja van: az entrópiakódolás és a forráskódolás. Most mindkettőt sorra vesszük.

Az **entrópiakódolás (entropy encoding)** egyszerűen bitfolyamokat manipulál, tekintet nélkül arra, hogy a bitek mit jelentenek. Ez egy általános, veszteségmentes, teljesen visszafordítható technika, amely mindenfajta adatra alkalmazható. Három példán keresztül fogjuk bemutatni.

Az entrópiakódolás első példája a **futamhosszkódolás (run-length encoding)**. Sokfajta adatban szokásosak az ismétlődő szimbólumokból (bitekből, számokból stb.) álló sorozatok. Ezeket egy, az adatban nem megengedett speciális jelzővel helyettesíthetjük, amelyet a futam szimbóluma, majd az előfordulások száma követ. Ha a speciális jelző előfordul az adatban, akkor azt megduplázzák (mint a karakterbeszúrásnál). Példának vegyük a következő, decimális számjegyekből álló sorozatot:

315000000000000845871111111111111635467400000000000000000000065

Ha most bevezetjük az A-t jelzőnek, és az ismétlődési számlálónak kétjegyű számokat használunk, a fenti számjegysorozatot a következőképpen kódolhatjuk:

315A01284587A1136354674A02265

Itt a futamhosszkódolás felére nyomta össze az adatsorozatot.

A futamok elterjedtek a multimédiában. A hangoknál a csendet gyakran nullák futamai jelképezik. A képeknél az ugyanabból a színből álló futamok az égről, falakról és sok sík felületről készült felvételekben fordulnak elő. Mindezek a futamok nagymértékben tömöríthetőek.

Az entrópiakódolás második példája a **statisztikai kódolás (statistical encoding)**. Ezen azt értjük, hogy rövid kódot használunk a gyakori szimbólumok, és hosszú kódokat a ritka szimbólumok reprezentálására. A Morse-kód ezt az elvet használja, mivel az E a • (pont), a Q a - - • -, és így tovább. A Huffman-kódolás és a UNIX Compress programja által használt Lempel–Ziv-algoritmus ugyancsak statisztikai kódolást használ.

Az entrópiakódolás harmadik példája a **CLUT (Color Look Up Table – színkódtáblázat)** kódolás. Vegyünk egy képpontonként 3 bájtot használó RGB kódolással el-

tárolt képet. Elméletben a kép akár 2^{24} különböző színértéket is tartalmazhat. A gyakorlatban ennél rendszerint sokkal kevesebb értéket fog tartalmazni, különösen, ha a kép egy képregényből való, vagy egy számítógép által rajzolt ábra, és nem egy fénykép. Tegyük fel, hogy valójában csak 256 színértéket használ. Egy majdnem háromszoros tömörítés érhető el, ha egy 768 bájtos táblázatot építünk fel a valóban használt 256 szín RGB értékeiből, majd minden képpontot az RGB értékei táblázatból kikeresett címével írunk le. Ez egy nyilvánvaló példa arra is, amikor a kódolás lassabb, mint a dekódolás, mivel a kódolás táblázatban való keresést igényel, míg a dekódolás egy egyszerű címzési művelettel elvégezhető.

Forráskódolás

Következőnek a **forráskódolás (source encoding)** jön, amely az adat tulajdonságait kihasználva ér el nagyobb (rendszerint veszteséges) tömörítést. Az ötletet itt is három példával fogjuk szemléltetni. Első példánk a **különbségi kódolás (differential encoding)**, amelyben értékek egy sorozatát (pl. hangmintákat) úgy kódolunk, hogy mindegyiket az előző értékhez viszonyított eltérésével írjuk le. A 2. fejezetben látott különbségi pulzuskód-moduláció ennek a technikának egy példája. Ez veszteséges, mivel a jel két egymást követő érték közt annyit ugorhat, hogy a különbség már nem fér bele az eltérés leírására használt mezőbe, így legalább egy helytelen érték rögzítésre kerül, és némi információ el fog veszni.

A különbségi kódolás a forráskódolás egy fajtája, mivel azt a tulajdonságot használja ki, hogy az egymást követő adatpontok közt valószínűtlenek a nagy ugrások. Nem minden számsorozat rendekezik ezzel a tulajdonsággal. Egy ilyen tulajdonsággal nem rendelkező példa lehet egy véletlen telefonszámokból álló, számítógép által előállított lista, amelyet a telefonos ügynökök használnak arra, hogy az embereket vacsorájuk közben zargassák. A listában egymást követő telefonszámok különbségének leírásához annyi bit kell, mint maguknak a számoknak a leírásához.

A második forráskódolási példánk **transzformációkból** áll. Ha a jeleket az egyik tartományból a másikba transzformáljuk, a tömörítés sokkal könnyebbé válhat. Vegyük például a 2.1.(a) ábra Fourier-transzformációját. Itt egy idő függvényt amplitúdók egy listájával írunk le. Ha minden amplitúdó pontos értéke adott, az eredeti függvényt tökéletesen helyre lehet állítani. Ám ha csak mondjuk az első nyolc amplitúdó értékét adják meg két tizedesjegy pontossággal, akkor is előfordulhat, hogy a jel olyan jól visszaállítható, hogy a hallgató nem ismeri fel, hogy információ veszett el. A nyereség itt az, hogy nyolc amplitúdó átvitele sokkal kevesebb bitet igényel, mint a mintavételezett hullámforma átvitele.

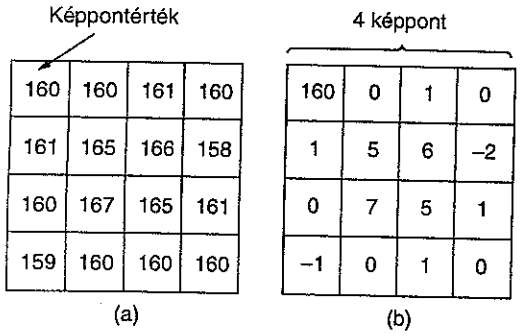
Transzformációkat alkalmazhatunk kétdimenziós képadatokra is. Tegyük fel, hogy a 7.78.(a) ábra 4×4-es mátrixa egy monokróm kép szürkeskálás értékeit írja le. Ezt az adatot úgy transzformálhatjuk, hogy a bal felső sarokban levő értéket kivonjuk az összes elemből, kivéve saját magát, ahogy a 7.78.(b) ábrán látható. Ez a transzformáció hasznos lehet, ha változó szóhosszúságú kódolást alkalmazunk. Például a +7 és –7 közé eső értékeket 4 bites számokként, kódolhatjuk, a 0 és 255 közé esőket pedig mint egy speciális 4 bites kódot (a –8-at) követő 8 bites számot.

Bár ez az egyszerű transzformáció veszteségmentes, más hasznosabb transzformációk nem azok. Egy különösen fontos kétdimenziós térbeli transzformáció a **DCT (Discrete Cosine Transformation – diszkrét koszinusz-transzformáció)** (Feig és Winograd, 1992). Ez a transzformáció olyan tulajdonsággal bír, hogy a meredek egyenetlenségek nélküli képekre a spektrális teljesítmény az első néhány együtthatóba kerül, így a későbbiek különösebb információvesztés nélkül elhagyhatók. Rövidesen visszatérünk a DCT-re.

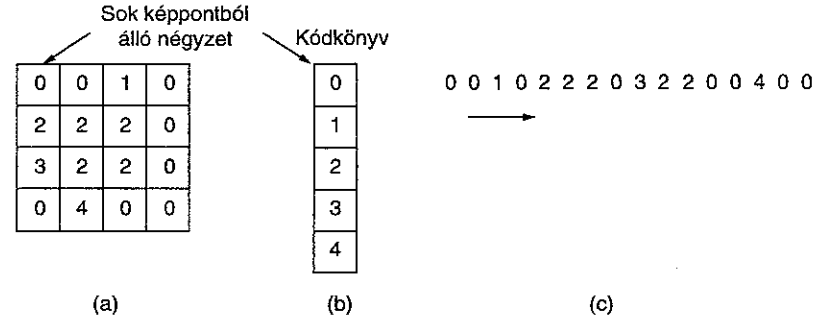
A forráskódolás harmadik példája a **vektorkvantálás (vector quantization)**, amely szintén közvetlenül alkalmazható képadatokra. Itt a képet fix méretű téglalapokra osztjuk fel. A képen kívül szükségünk lesz még egy (esetleg a képből előállított) téglalapokból álló táblázatra, amely elemeinek mérete megegyezik a kép téglalapjainak méretével. Ezt a táblázatot **kódkönyvnek (code book)** nevezzük. Minden téglalapot úgy viszünk át, hogy kikeressük azt a kódkönyvből, és a téglalap helyett csak annak címét visszük át. Ha a kódkönyvet dinamikusan (vagyis képenként) hozzuk létre, akkor azt is át kell vinni. Nyilvánvaló, hogy ha a kép túlnyomórészt csak pár téglalapból áll, akkor itt nagy sávszélesség-megtakarítás lehetséges.

A vektorkvantálásra egy példa látható a 7.79. ábrán. A 7.79.(a) ábrán egy közelebbről meg nem határozott méretű téglalapokból álló rácsot láthatunk. A 7.79.(b) ábrán láthatjuk a kódkönyvet. A kimeneti folyam a 7.79.(c) ábrán látható, egész számokból álló lista: 00102220302200400. Mindegyik szám a kódkönyv egy bejegyzését jelöli.

Bizonyos értelemben a vektorkvantálás egyszerűen a CLUT kétdimenziós általánosítása. A valódi különbség ott van, hogy mi történik akkor, ha nincs illeszkedés. Három stratégia lehetséges. Az első az, hogy egyszerűen a legjobban illeszkedő bejegyzést használjuk. A második az, hogy használjuk a legjobban illeszkedő bejegyzést, és fűzzünk hozzá valami információt arról, hogy hogyan lehet a becslést javítani (pl. a valódi átlagértéket). A harmadik az, hogy használjuk a legjobban illeszkedő bejegyzést, és fűzzünk még hozzá valamit, amiből a dekódoló pontosan helyre tudja állítani az adatot. Az első két stratégia veszteséges, de nagy tömörítés érhető el velük. A harmadik veszteségmentes, de kevésbé hatékony tömörítő algoritmus. Itt is látható, hogy a kódolás (mintaillesztés) sokkal időigényesebb, mint a dekódolás (egy tábla megcímzése).



7.78. ábra. *(a) Képpontértékek egy képrészletben. (b) Egy transzformáció, amelyben a bal felső elemet saját magán kívül mindegyikből kivonjuk*



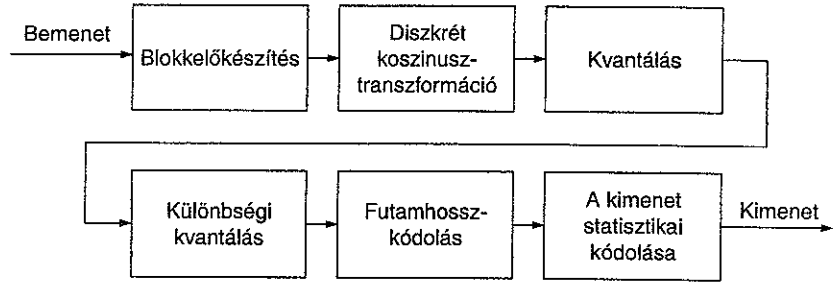
7.79. ábra. *A vektorkvantálás egy példája. (a) Egy négyzetekre felosztott kép. (b) Egy kódkönyv a képhez. (c) A kódolt kép*

A JPEG szabvány

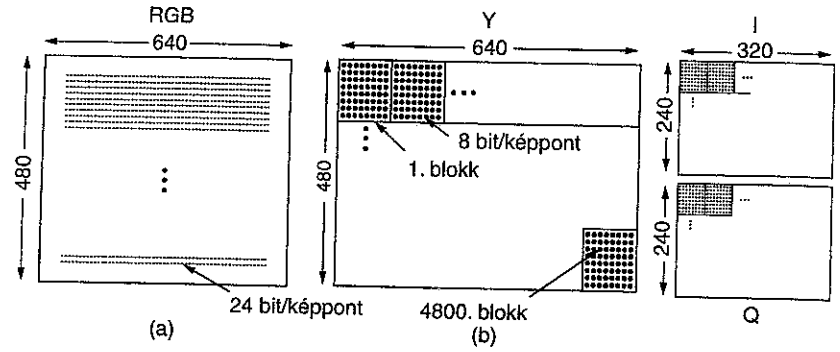
A **JPEG (Joint Photographic Experts Group)** szabványt, amely folytonos tónusú állóképek (vagyis fényképek) tömörítésére szolgál, az ITU, ISOés az IEC felügyelete alatt dolgozó fényképszakértők fejlesztették ki. A multimédia számára azért fontos, mert első közelítésben a mozgóképek multimédia szabványa, az MPEG, nem más, mint minden kép külön-külön való JPEG kódolása, kiegészítve néhány pluszszolgáltatással a képek közti tömörítéshez és a mozgásérzékeléshez. A JPEG-et az ISO 10918 Nemzetközi Szabvány definiálja.

A JPEG-nek négy működési módja és számos opciója van. Jobban hasonlít egy bevásárlólistára, mint egy algoritmusra. Mindazonáltal a mi céljainkhoz csak a veszteséges soros mód (lossy sequential mode) bír jelentőséggel, amelyet a 7.80. ábra mutat be. Ezen kívül a JPEG normális használati módjára összpontosítunk, amely a 24 bites RGB képek tömörítése, és némely apróbb részletet az egyszerűség kedvéért ki fogunk hagyni.

Egy kép JPEG kódolásának első része a blokkok előkészítése. Az egyszerűség kedvéért tegyük fel, hogy a JPEG bemenetén egy 640×480-as, képpontonként 24 bites



7.80. ábra. *A JPEG működése veszteséges soros módban*



7.81. ábra. *(a) Az RGB bemeneti adat. (b) A blokkok előkészítése után*

RGB kép van, amint a 7.81. ábrán látszik. Mivel a luminancia és krominancia használata jobb tömörítést eredményez, ezért először kiszámítjuk az *Y* luminanciát és az (NTSC esetében) *I* és *Q* krominanciát, a következő képletek szerint:

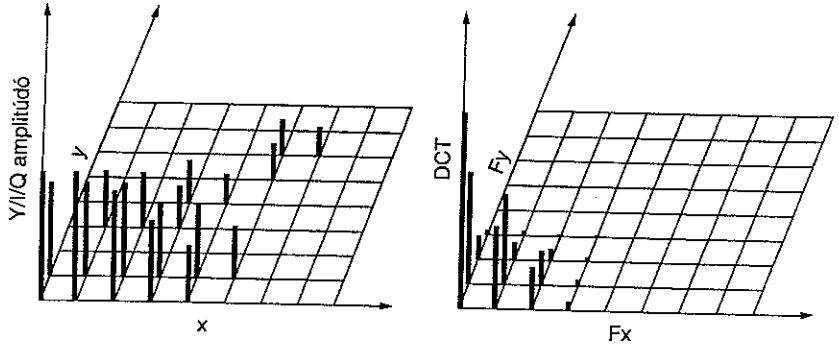
$$Y = 0{,}30R + 0{,}59G + 0{,}11B$$

$$I = 0{,}6R - 0{,}28G - 0{,}32B$$

$$Q = 0{,}21R - 0{,}52G + 0{,}31B$$

A PAL esetén a krominanciákat *U*-nak és *V*-nek hívják, és az együtthatók mások, de az ötlet ugyanaz. A SECAM mind az NTSC-től, mind a PAL-tól eltér.

Az *Y*, *I* és *Q* számára külön mátrixokat készítünk, amelyek közül mindegyik 0 és 255 közé eső elemeket tartalmaz. Következőnek az *I* és *Q* 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 az adatot a felére tömöríti össze. Most



7.82. ábra. *(a) Az Y mátrix egy blokkja. (b) A DCT együtthatók*

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 7.81.(b) ábrán látszik.

A JPEG 2. lépésében mind a 7200 blokkra külön alkalmazunk egy diszkrét koszinusz-transzformációt. 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. Elméletben a DCT veszteségmentes, de a gyakorlatban a lebegőpontos számok és transzcendens függvények használata mindig okoz valamilyen kerekítési hibát, amely némi információveszteségben jelentkezik. Rendszerint az elemek az origótól (0, 0) való távolsággal arányosan rohamosan csökkennek, ahogy a 7.82. ábra is jelzi.

Ha a DCT kész van, akkor a JPEG továbblép a 3. lépésre, amelyet **kvantálásnak (quantization)** hívnak. Itt a kevésbé fontos DCT együtthatókat kitöröljük. Ezt a (veszteséges) transzformációt úgy végezzük, hogy a 8×8-as DCT mátrixban minden együtthatót leosztunk egy táblázatból vett súllyal. Ha mindegyik súly értéke 1, akkor a transzformáció semmit sem csinál. Viszont, ha a súlyok az origótól távolodva gyorsan csökkennek, akkor a nagyobb térbeli frekvenciák gyorsan kiesnek.

Erre a lépésre egy példa látható a 7.83. ábrán. Itt a kezdeti DCT táblázat, a kvantálási táblázat, és az eredmény látható, amelyet úgy kapunk, hogy minden DCT elemet

A 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

A 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

A 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

7.83. ábra. *A kvantált DCT együtthatók számítása*

leosztunk 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.

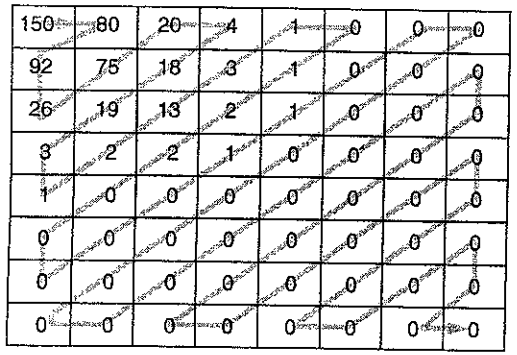
A 4. lépésben lecsökkentjük minden blokk (0, 0) értékét, azáltal, hogy az előző blokk ugyanezen elemével való eltérésével helyettesítik. Mivel ezek az elemek a blokkok átlagai, várhatóan csak lassan fognak változni, ezáltal a különbségi értékek kis számok lesznek. A többi értéknél nem számolunk eltérést. A (0, 0) értékeket mint DC komponenseket szokták emlegetni, a többi érték AC komponens.

Az 5. lépésben sorba rakjuk a 64 elemet és a listára futamhosszkódolást alkalmazunk. A blokk balról jobbra, fentről lefele történő pásztázása nem gyűjtené össze a 0-kat, így egy cikcakkos pásztázási mintát alkalmaznak. Ebben a példában a cikcakk minta 38 egymás utáni 0-t eredményez a mátrix vége felé. Ezt a sorozatot egyetlen számlálóba össze lehet vonni, amely azt mondja, hogy 38 nulla van ott.

Most már van egy számsorozatunk, amely (a transzformált térben) leírja a képet. A 6. lépés Huffman-kódolja a számokat átvitel vagy tárolás céljából.

A JPEG bonyolultnak tűnhet, de ez azért van, mert valóban bonyolult. Mivel azonban gyakran 20:1 vagy még jobb arányú tömörítést ér el, elterjedten használják. Egy JPEG kép dekódolása az algoritmus fordítottjának lefuttatását igényli. Eltérően más, eddig látott tömörítési algoritmusoktól, a JPEG nagyjából szimmetrikus: a dekódolás annyi időbe telik, mint a kódolás.

Érdekes módon a DCT matematikai tulajdonságaiból adódóan lehetséges közvetlenül a transzformált mátrixon elvégezni bizonyos geometriai transzformációkat (mint amilyen a kép elforgatása), az eredeti kép újragenerálása nélkül. Ezeket a transzformációkat tárgyalja (Shen és Sethi, 1995). Az MPEG tömörített hangnak hasonló tulajdonságai vannak.



7.84. ábra. *A sorrend, amelyben a kvantált értékeket átviszik*

Az MPEG szabvány

Végre eljutottunk a dolog lényegéig: az **MPEG (Motion Picture Experts Group)** szabványokig. Ezek a mozgóképek tömörítésére használt fő algoritmusok, és 1993 óta nemzetközi szabványok is. Mivel a filmek képeket és hangokat is tartalmaznak, az MPEG mind hang-, mind képtömörítésre alkalmas. Mivel azonban a kép nagyobb sávszélességet foglal el, és több redundanciát is tartalmaz, mint a hang, ezért a továbbiakban elsődlegesen az MPEG képtömörítéssel fogunk foglalkozni.

Az első végleges szabvány az MPEG–1 volt (ISO 11172 Nemzetközi Szabvány). A cél az volt, hogy 1,2 Mb/s-os bitsebességet használva a videolejátszó minőségét elérjék, amely NTSC esetén a 352x240-es felbontás. Mivel korábban láttuk, hogy a tömörítetlen videoadat sebessége akár 472 Mb/s-ig is felmehet, ezért nem egészen triviális ezt 1,2 Mb/s-ra levinni, még emellett az alacsonyabb felbontás mellett sem. Az MPEG–1 mérsékelt távolságokra alkalmas sodrott érpáron való átvitelre is. Az MPEG–1-et használják filmek CD-ROM-on való tárolásához is CD-I és CD-Video formátumban.

Az MPEG család következő szabványa az MPEG–2 volt (ISO 13818 Nemzetközi Szabvány), amelyet eredetileg műsorszórásra alkalmas minőségű mozgókép 4-6 Mb/s-ba tömörítésére terveztek, hogy azt egy NTSC vagy PAL csatornába be lehessen illeszteni. Később az MPEG–2-t kiterjesztették nagyobb felbontásokra is, a HDTV-t is beleértve. Az MPEG–4 a közepes felbontású videokonferenciákhoz használható alacsony képsebességgel (10 kép/s) és kis sávszélesség esetén (64 kb/s). Ez lehetővé teszi videokonferenciák szervezését egyetlen N-ISDN B vonalon. Valójában az ISO ezeket nem hatványok szerint, hanem sorban számozza. Eredetileg létezett az MPEG–3 is, amelyet a HDTV-hez szántak, de a projektet később leállították, és a HDTV-t az MPEG–2-höz vették hozzá.

Az MPEG–1 és MPEG–2 alapelvei hasonlóak, de a részletekben eltérnek. Első közelítésben az MPEG–1 az MPEG–2 részhalmaza, amelyet további szolgáltatások, keretformátumok és kódolási lehetőségek bővítenek. Valószínűnek tűnik, hogy hosszú távon az MPEG–1 lesz használatos CD-ROM-os filmekhez, és az MPEG–2 a nagy távolságú videoátvitelhez. Először az MPEG–1-et tárgyaljuk, majd az MPEG–2-t is.

Az MPEG–1 három részből áll: audio, video és a rendszer, amely magába foglalja a másik kettőt, ahogy a 7.85. ábrán látszik. A hang- és képkódolók egymástól függetlenül működnek, amely felveti azt a kérdést, hogy hogyan lehet a két folyamot a vevőnél szinkronizálni. Ezt a problémát az oldja meg, hogy van egy 90 kHz-es rendszeróra, amely mindkét kódolónak megadja az aktuális időt. Ezek az értékek 33 bitesek, így egy film 24 óráig futhat körbefordulás nélkül. Ezeket az időbélyegeket tartalmazza a kódolt kimenet, és így eljutnak a vevőig, amely ezeket a hang- és képfolyamok szinkronizálásához használhatja fel.

Az MPEG hangtömörítés a hullámforma 32 kHz-en, 44,1 kHz-en vagy 48 kHz-en történő mintavételezésével kezdődik. Képes mono, külön sztereo (a két csatorna külön kerül tömörítésre) vagy kapcsolt sztereo (kihasználva a csatornák közti redundanciát) csatornák kezelésére. Három rétegbe van szervezve, amelyek fokozatosan további optimalizálásokkal egyre nagyobb tömörítést érnek el (egyre nagyobb áron). Az 1. réteg az alapeljárás, ezt használják például a DCC digitális magnókban. A 2. réteg fejlettebb bitallokációt tesz lehetővé az alapeljáráshoz képest. Ezt használják a CD-ROM audio

és filmhangsávokhoz. A 3. réteg hibrid szűrőket, nem egyenletes kvantálást, Huffman-kódolást, és más fejlett technikákat vesz még ehhez hozzá.

Az MPEG audio képes egy rock and roll CD-t 96 kb/s-ra tömöríteni érzékelhető minőségromlás nélkül, még a nem halláskárosult rock and roll rajongók számára is. Egy zongorahangversenyhez legalább 128 kb/s-ra van szükség. Ennek az az oka, hogy a rock and roll jel/zaj viszonya (legalábbis mérnöki értelemben) jóval nagyobb, mint egy zongorahangversenyé.

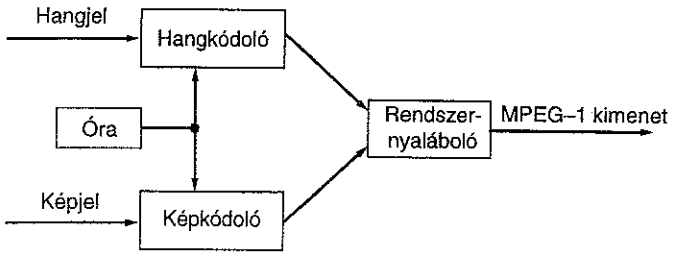
A hangtömörítés egy gyors Fourier-transzformációval kezdődik, hogy az időtartományból áttérjünk a frekvenciatartományba. Az eredményül kapott spektrumot ezután 32 frekvenciasávra osztják fel, és mindegyiket külön dolgozzák fel. Amikor két sztereo csatorna van jelen, akkor a két, egymást nagymértékben átfedő csatornából adódó redundanciát is kihasználják. Az eredményül kapott MPEG–1 hangfolyam sebessége állítható 32 és 448 kb/s között. A folyamatba bevezetőt ad (Pan, 1995).

Most vegyük szemügyre az MPEG–1 képtömörítést. A mozgásokban kétfajta redundancia van: térbeli és időbeli. Az MPEG–1 mindkettőt kihasználja. A térbeli redundancia kihasználásához egyszerűen minden kép külön kódolható a JPEG-gel. Néha ezt a megközelítést használják, különösen amikor a tetszőleges hozzáférés szükséges minden egyes képkockához, mint videók szerkesztése közben. Ebben a módban 8–10 Mb/s sávszélesség érhető el tömörítés után.

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. (Mérje le egy film időtartamát, és számolja a vágásokat!) Ennek ellenére még egy 75 nagymértékben azonos képkockából álló futam is nagy csökkentési lehetőséget kínál a képek külön JPEG kódolásához képest.

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 az, hogy minden képet kivonunk az előzőből, és a különbségre JPEG kódolást alkalmazunk, jól működne. 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–1 kimenet négyfajta képből áll:



7.85. ábra. *A hang- és képfolyamok szinkronizálása az MPEG–1-ben*

1. I (Intracoded) képek: önálló JPEG kódolt állóképek.

2. P (Predictive) képek: blokkonkénti eltérés az előző képtől.

3. B (Bidirectional) képek: eltérés az előző és a következő képtől.

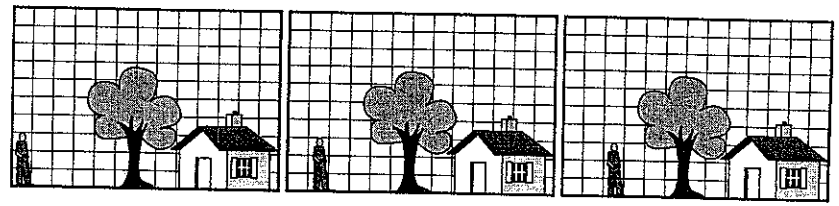
4. D (DC-kódolt) képek: az előretekeréshez használt blokkátlagok.

Az I képek egyszerűen JPEG kódolt állóképek, amelyek mindkét tengely irányába teljes felbontású luminancia- és feleakkora felbontású krominanciamátrixokat használnak. Három ok miatt van szükség arra, hogy a kimeneti folyamban rendszeres időközönként feltűnjenek I képek. Először is, az MPEG–1 használható többesküldésre is, ahol a nézők tetszés szerint kapcsolódhatnak be az adásba. Ha minden kép függ az előzőektől, egészen az első képig visszamenőleg, senki nem tudná a további képeket dekódolni, aki az elsőt elmulasztotta. Másodszor, ha valamely kép hibásan érkezne meg, nem lenne lehetséges a további dekódolás. Harmadszor, I képek nélkül egy előre- vagy visszatekerésnél a dekódernek minden képet ki kellene számolnia, amelyen áthalad, hogy tudja annak a teljes értékét, amelyen megáll. Ezen okok miatt másodpercenként egyszer vagy kétszer I képeket is beszúrnak a kimenetbe.

Ezzel ellentétben a P képek a képek közti különbségeket kódolják. Ezek a **makroblokkokon (macroblocks)** alapulnak, amelyek a luminanciamátrixból egy 16×16-os, a krominanciamátrixokból pedig 8×8-as részt fednek le. Egy makroblokkot úgy kódolnak, hogy az előző képen megkeresik ugyanazt, vagy valami olyat, amely ettől csak kevéssé tér el.

A 7.86. ábrán látható egy olyan példa, ahol a P képek hasznosak lehetnek. Itt három egymás után következő képet látunk, amelyeknek ugyanaz a hátterük, de egy ember helyzete eltér. A hátteret tartalmazó makroblokkok pontosan illeszkedni fognak, de az embert tartalmazó makroblokkok el lesznek valamilyen ismeretlen értékkel tolva, és azokat meg kell keresni.

Az MPEG–1 szabvány nem határozza 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 pontszámmal rendelkező pozíciót kiálthatnák ki nyertesnek, feltéve, hogy az valamilyen küszöbszint felett van. Egyébként a makro-



7.86. ábra. *Három egymást követő kép*

blokkot hiányzónak nevezik. Természetesen sokkal kifinomultabb algoritmusok is lehetségesek.

Ha egy makroblokkot megtalálnak, akkor veszik az előző képpel való különbségét, luminanciában és krominanciákban is. Ezeket a különbségmátrixokat azután diszkrét koszinusz-transzformációnak, kvantálásnak, futamhosszkódolásnak és Huffman-kódolásnak vetik alá, mint a JPEG-nél. A makroblokk ezután a kimeneti folyamban úgy jelenik meg, mint a mozgásvektora (azaz, hogy mennyit mozdult el a makroblokk előző helyzetéhez képest mindkét irányban), amelyet a számok Huffman-kódolt sorozata követ. Ha a makroblokk nem szerepel az előző képen, akkor a jelenlegi értékét JPEG kódolják, úgy, mintha egy I kép lenne.

Ez az algoritmus nyilván nagymértékben aszimmetrikus. Egy megvalósítás akár minden lehetséges pozíciót kipróbálhat az előző képen, ha mindenképpen meg akarja találni az utolsó makroblokkot is. Ez a megközelítés minimalizálni fogja a kódolt MPEG–1 folyamot, azon az áron, hogy nagyon lassú lesz a kódolás. Ez jó lehet egy filmkönyvtár egyszeri lekódolásához, de a valós idejű videokonferenciához nagyon nem jó.

Hasonlóan minden megvalósítás maga döntheti el, hogy mit vesz „megtalált" makroblokknak. Ez a szabadság lehetővé teszi a megvalósítás programozóinak, hogy versenyezzenek a minőségben és a használt keresési algoritmusban, de ugyanakkor az MPEG–1-nek megfelelő kimenetet állítsanak elő. A használt keresési algoritmustól függetlenül a végső kimenet mindig vagy az aktuális makroblokk JPEG kódoltja, vagy az aktuális makroblokk és egy, az előző képen az aktuális makroblokktól meghatározott távolságra levő makroblokk különbségének JPEG kódoltja.

Eddig az MPEG–1 dekódolása magától értetődő. Az I képek dekódolása ugyanúgy megy, mint a JPEG képek dekódolása. A P képek dekódolásához a dekódernek el kell tárolni az előző képet és a következőt egy másik pufferben kell felépítenie a teljesen kódolt és az előző kép óta történt változásokat tartalmazó makroblokkokból. Az új képet makroblokkról makroblokkra rakják össze.

A B képek hasonlóak a P képekhez, csak ezek lehetővé teszik, hogy a hivatkozott makroblokk egy előző vagy egy következő képben legyen. Ez a további szabadság javított mozgáskompenzációt tesz lehetővé, és akkor is hasznos, amikor a tárgyak más tárgyak előtt vagy mögött haladnak el. A B képek kódolásához a kódolónak egyszerre három dekódolt képet kell a memóriában tartania: az előzőt, a jelenlegit és a következőt. Bár a B képek adják a legjobb tömörítést, nem minden megvalósítás támogatja azokat.

A D képek csak arra szolgálnak, hogy egy előre- vagy visszatekerés közben meg lehessen jeleníteni egy alacsony felbontású képet. A rendes valós idejű MPEG–1 dekódolás elég bonyolult feladat. Ha a dekódolótól azt várjuk el, hogy ezt akkor is elvégezze, amikor tízszer olyan gyorsan száguldunk végig a filmen, mint rendesen, akkor túl sokat kívánnánk tőle. Ehelyett a D képeket használják arra, hogy kis felbontású képeket jelenítsenek meg. Minden D kép egyszerűen egy blokk átlagértéke, további kódolás nélkül, amely valós időben könnyen megjeleníthetővé teszi. Ez a szolgáltatás azért fontos, hogy az emberek nagy sebességgel áttekinthessenek egy filmet, egy bizonyos jelenetet keresve.

Most, hogy befejeztük az MPEG–1 tárgyalását, lépjünk tovább az MPEG–2-re. Az MPEG–2 kódolás alapjában véve hasonlít az MPEG–1 kódoláshoz. Megvannak az I, a

P és a B képek, ám a D képeket nem támogatja. A diszkrét koszinusz-transzformáció is 10×10-es (8×8-as helyett), így 50%-kal több együtthatót és ezáltal tisztább képet eredményez. Mivel az MPEG–2 a műsorszóró televíziózást és a CD-ROM alkalmazásokat veszi célba, támogatja a progresszív és a sorugrásos képeket is, míg az MPEG–1 csak a progresszív képeket támogatta. Más kisebb részletekben is van eltérés a két szabvány között.

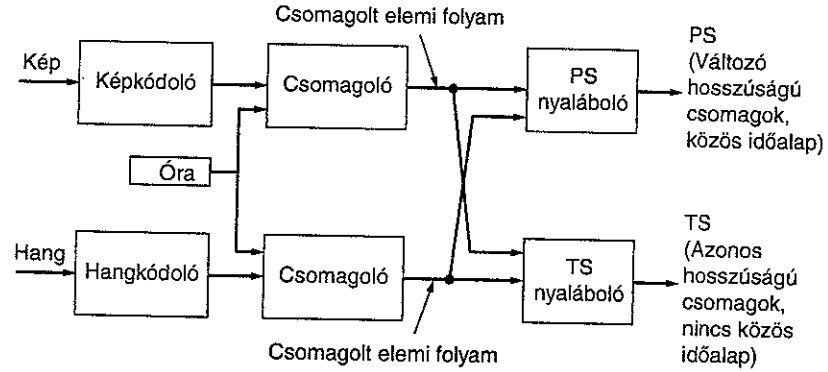
Egyetlen felbontási szint támogatása helyett az MPEG–2 négyet támogat: alacsony (352×240), közepes (720×480), 1440-es nagy (1440×1152) és nagy (1920×1080) felbontási szinteket. Az alacsony felbontás a videokészülékek és az MPEG–1-gyel való visszamenőleges kompatibilitás miatt került bele. A közepes a rendes NTSC műsorszóráshoz való, a másik kettő pedig a HDTV-hez szükséges.

A négy felbontási szint támogatásán kívül az MPEG–2-nek öt **profilja (profile)** is van. Minden profil valamely felhasználási területet céloz meg. A fő (main) profil az általános célú használat, és várhatóan a legtöbb chipet erre a fő profilra és a közepes felbontásra fogják optimalizálni. Az egyszerű (simple) profil hasonlít a fő profilhoz, kivéve, hogy kizárja a B képek használatát, hogy a szoftveres kódolást és dekódolást egyszerűvé tegye. A többi profil a méretezhetőséggel és a HDTV-vel van kapcsolatban. A profilok közt az eltérést a B képek megléte vagy hiánya, a színfelbontás, és a kódolt bitfolyam más formátumokra való méretezhetősége jelenti.

A tömörített adatsebesség minden felbontás-profil kombinációhoz más és más. Ezek úgy 3 Mb/s-tól a HDTV 100 Mb/s-ig terjednek. Normális esetben ez 3 és 4 Mb/s közé esik. Az MPEG néhány teljesítményadatát adja meg (Pancha és El Zarki, 1994).

Az MPEG–2 egy általánosabb módszerrel rendelkezik a hang és kép nyalábolásához, mint a 7.85. ábrán látott MPEG–1 modell. Végtelen számú elemi folyamot definiál, amelyek közt megtalálható a hang és a kép, de az olyan adatfolyamok is, amelyeket a hanggal és a képpel szinkronizálni kell, például a többnyelvű feliratok. Először a folyamok mindegyikét egybecsomagolják az időbélyegekkel. Egy egyszerű kétfolyamos példa látható a 7.87. ábrán.

Minden csomagoló kimenete egy **PES (Packetized Elementary Stream – csoma-**



7.87. ábra. *Két folyam nyalábolása az MPEG–2-ben*

golt elemi folyam). Minden PES csomagnak körülbelül 30 fejrész mezője és jelzője van, többek közt a hossz, a folyam azonosítói, a titkosítás vezérlése, az időbélyegek és egy CRC.

A hangok, képek és esetleg adatok PES folyamait ezután egyetlen kimeneti folyamba nyalábolják az átvitelhez. Kétfajta folyamot definiáltak. Az MPEG–2-es **programfolyam (program stream)** hasonlít az MPEG–1 folyamához, amely a 7.85. ábrán látható. Ez olyan elemi folyamok nyalábolásához használatos, amelyeknek az időalapjuk közös és szinkronizáltan kell azokat megjeleníteni. A programfolyam hosszú és változó hosszúságú csomagokat használ.

A másik MPEG–2 folyam a **szállítási folyam (transport stream).** Ez olyan folyamok (köztük programfolyamok) egybenyalábolására szolgál, amelyeknek nincs közös időalapjuk. A szállítási folyam csomagjai rögzített hosszúságúak (188 bájtosak), hogy könnyebb legyen a szállítás során megsérült vagy elveszett csomagok hatását korlátozni.

Érdemes külön kiemelni, hogy az általunk tárgyalt kódolási módszerek a veszteséges kódolást követő veszteségmentes szállítás modelljén alapulnak. Sem a JPEG, sem az MPEG nem tud elegánsan helyreállni például az elveszett vagy sérült csomagok okozta hibákból. A képtovábbítás egy másik megközelítése az, hogy a képeket oly módon transzformáljuk, amely elkülöníti a fontos információt a kevésbé fontos információtól. (Például ezt teszi a DCT is.) Ezután adjunk hozzá meglehetős redundanciát (akár még kettőzött csomagokat is) a fontos információhoz, de a kevésbé fontos információhoz ne. Ha néhány csomag elveszik vagy összezavarodik, még mindig esély van arra, hogy újraadás nélkül értelmes képeket tudjunk megjeleníteni. Ezeket az ötleteket írják le részletesebben (Danskin és mások, 1995). Különösen jól alkalmazhatók ezek a többesküldéses átvitelhez, ahol egyébként is lehetetlen minden vevőtől visszajelzést várni.

7.7.4. Hálózati videózás

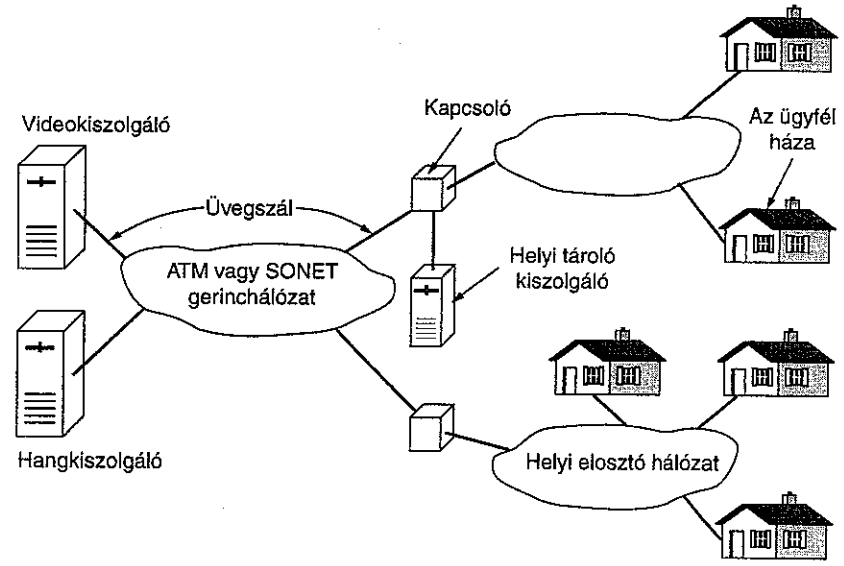
A hálózati videózást néha egy elektronikus videokölcsönzőhöz hasonlítják. A felhasználó (ügyfél) kiválaszt egyet a nagyszámú elérhető videofilm közül, és hazaviszi azt, hogy megnézze. Az eltérés az, hogy a hálózati videózásnál a kiválasztás otthonról történik a televíziókészülék távirányítóját használva, és a videofilm azonnal elindul. Nem kell elmenni a boltba. Felesleges is mondani, hogy a hálózati videózás megvalósítása kissé bonyolultabb, mint a leírása. Ebben a szakaszban egy áttekintést adunk az alapötletekről és azok megvalósításairól. Egy valódi megvalósítás leírása található (Nelson és Linton, 1995) könyvében. Az interaktív televízió egy általánosabb leírását adja (Hodge, 1995). További lényeges hivatkozások: (Chang és mások, 1994; Hodge és mások, 1993; Little és Venkatesh, 1994).

A hálózati videózás valóban olyan, mint egy videofilm kikölcsönzése, vagy inkább olyan, mint egy megnézni kívánt film kiválasztása egy 500 vagy 5000 csatornás kábeltelevíziós rendszerből? A választnak fontos technikai következményei vannak. A videokölcsönzők ügyfelei hozzászoktak ahhoz, hogy képesek egy videofilmet leállítani, gyorsan kimenni a konyhába vagy a fürdőszobába, majd onnan folytatni, ahol a videofilm leállt. A tévénézők nem várják el azt, hogy leállíthassák a műsorokat.

Ha a hálózati videózás sikeresen akar versenyre kelni a videokölcsönzőkkel, szükséges lehet az, hogy a felhasználók tetszésük szerint állíthassák le, indíthassák el és tekerhessék vissza a videofilmeket. Ahhoz, hogy a felhasználók számára ezt a képességet biztosítsa, a videoszolgáltató gyakorlatilag kénytelen mindegyiküknek egy külön másolatot átvinni.

Másfelől viszont, ha a hálózati videózást inkább mint egy továbbfejlesztett televíziót nézzük, akkor elegendő lehet az, ha a videoszolgáltató minden népszerű videofilmet mondjuk 10 percenként elindít, és ezeket megállás nélkül futtatja. Egy népszerű videofilmet nézni akaró felhasználónak esetleg akár 10 percet is kell várnia annak kezdésére. Bár a megállítás/újraindítás itt nem lehetséges, egy néző, aki rövid szünet után visszatér a nappaliba, átkapcsolhat egy másik csatornára, amely ugyanazt a videofilmet mutatja, csak 10 perccel lemaradva. Lesz, amit kétszer fog látni, de semmiről nem fog lemaradni. Ezt a megoldást **közelítő hálózati videózásnak (near video on demand)** nevezik. Ezt a lehetőséget sokkal alacsonyabb költséggel nyújtja, mivel a videokiszolgáló ugyanazon jele egyszerre sok felhasználóhoz is eljuthat. A hálózati videózás és a közelítő hálózati videózás közti különbség hasonlatos ahhoz, amikor a saját autója helyett az ember buszra száll.

A filmek (majdnem) igény szerinti megtekintése csak egy a sok lehetséges szolgáltatás közül, amely a szélessávú hálózatok megjelenésével lehetségessé válik. A sok ember által használt általános modellt a 7.88. ábra mutatja be. Itt egy nagy sávszélességű (nemzeti vagy nemzetközi), nagy területet lefedő gerinchálózatot láthatunk a rendszer középpontjában. Ehhez helyi elosztó hálózatok ezrei kapcsolódnak, például a kábeltelevíziós vagy telefontársaságok elosztó rendszerei. A helyi elosztó rendszerek



7.88. ábra. *Egy hálózati videorendszer áttekintése*

egészen az emberek házáig elérnek, ahol ún. **készülékekben (set-top dobozokban)** végződnek, amelyek tulajdonképpen különleges, nagy teljesítményű személyi számítógépek.

A gerinchálózathoz információszolgáltatók ezrei kapcsolódnak nagy sávszélességű üvegszálakon keresztül. Ezek közül néhány megtekintés szerint fizetendő (pay-per-view) videót vagy meghallgatás szerint fizetendő (pay-per-hear) hang-CD-ket fog kínálni. Mások különleges szolgáltatásokat kínálnak, mint például az otthonról történő vásárlás (azzal a lehetőséggel, hogy egy leveskonzervet megforgathatunk, és ráközelíthetünk az összetevők listájára, vagy egy videoklipet nézhetünk meg arról, hogyan kell egy benzines fűnyírót vezetni). Kétségkívül gyorsan elérhetők lesznek egyéb szolgáltatások is, mint a sportközvetítések, a hírek, a „Dallas" ismétlései, a WWW elérése, és még számos más.

A rendszer helyi tároló kiszolgálókat is tartalmaz, amelyek segítségével a videofilmeket előre közelebb helyezhetjük a felhasználókhoz, hogy sávszélességet takarítsunk meg a csúcsidőszak alatt. Hogy hogyan fognak a részek egymáshoz illeszkedni, és ki mit fog birtokolni, arról heves viták vannak az iparon belül. A következőkben megvizsgáljuk a rendszer fő részeinek tervezését: a videokiszolgálókat, az elosztó hálózatot, és a set-top dobozokat.

Videokiszolgálók

A (közelítő) hálózati videózáshoz szükségünk van egyszerre nagyszámú film tárolására és továbbítására képes **videokiszolgálókra (video server).** A valaha is elkészített filmek számát 65 ezerre becsülik (Minoli, 1995). MPEG–2-vel tömörítve egy rendes film körülbelül 4 GB tárhelyet foglal, így ezekből 65 000-nek durván 260 terabájtra lenne szüksége. Ehhez még hozzávéve a valaha készített összes régi televízióműsort, a sportközvetítéseket, a híradók tekercseit, a hangos áru katalógusokat stb. nyilvánvaló, hogy egy ipari méretű tárolási problémával állunk szemben.

Nagy információtömegek legolcsóbb tárolási módja a mágnesszalag. Ez mindig is így volt, és valószínűleg így is lesz. Egy DAT szalag 8 GB-ot (két filmet) képes tárolni kb. 5 dollár/gigabájt költség mellett. Már ma is elérhetők a kereskedelemben olyan nagy szalagos kiszolgálók, amelyek szalagok ezreit tartalmazzák, és egy robotkarjuk van bármely szalag elhozására és a szalagmeghajtóba helyezésére. Ezekkel a rendszerekkel a gond az elérési idő (különösen a szalagon második film esetén), az átviteli sebesség, és a szalagmeghajtók korlátozott száma. (Egyszerre n film szolgáltatásához az egységnek n meghajtóra lenne szüksége.)

Szerencsére 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 (Chervenak, 1994), ahol C-t úgy számítják, hogy az összeget 1-re normálják, vagyis:

$$C = 1/(1+1/2+1/3+1/4+1/5+...+1/N)$$

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).

Az a tény, hogy egyes filmek sokkal népszerűbbek, mint mások, egy lehetséges megoldást kínál egy tárolóhierarchia formájában, ahogy az a 7.89. ábrán látszik. Itt a teljesítmény a hierarchiában felfele haladva egyre nő.

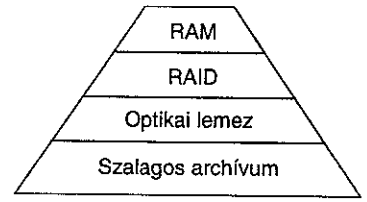
A szalag egy alternatívája az optikai tárolás. A mai CD-ROM-ok csak 650 MB kapacitásúak, de a következő generáció 4 GB körüli kapacitású lesz, ezáltal alkalmas MPEG–2 filmek terítésére. Bár a keresési idő lassú a mágneslemezhez viszonyítva (10 ms helyett 100 ms), az alacsony költségük és a nagy megbízhatóságuk a CD-ROM-ok ezreit tartalmazó optikai juke boxokat a mágnesszalag jó alternatívájává teszik a gyakrabban használt filmek esetére.

Következnek a mágneslemezek. Ezeknek rövid az elérési idejük (10 ms), nagy az átviteli sebességük (10 Mb/s), és megfelelő a kapacitásuk (10 GB), amely kifejezetten alkalmassá tszi őket az éppen adásban levő filmek tárolására (ahelyett, hogy csak tárolnák azokat, arra az esetre, ha valakinek esetleg szüksége lesz azokra). A fő hátrányuk, hogy az áruk magas olyan filmek tárolásához, amelyeket csak ritkán érnek el.

A 7.89. ábra piramisának csúcsán a RAM helyezkedik el. A RAM a leggyorsabb tárolási médium, de egyben a legdrágább is. Legjobban azokhoz a filmekhez illeszkedik, amelyek különböző részeit küldik más-más célokhoz ugyanabban az időben (pl. valódi hálózati videózásnál, ahol 100 felhasználó különböző időben kezdte el azt nézni). Amikor a RAM ára 10 dollárra esik megabájtonként, akkor egy 4 GB-os film 40 000 dollár értékű RAM-ot fog elfoglalni, így 100 film RAM-ban tartása 4 millió dollárba fog kerülni, amennyibe 400 GB memória kerül. Mégis, egy 10 millió dolláros videokiszolgáló esetén ez a költség megtérülhet, ha minden filmnek elegendő fizető ügyfele van egy időben.

Mivel a videokiszolgáló igazából csak egy masszív valós idejű B/K eszköz, más szoftver és hardver architektúrára van szüksége, mint egy PC-nek vagy egy UNIX munkaállomásnak. Egy jellegzetes videokiszolgáló hardverfelépítése látható a 7.90. ábrán. A szerverben egy vagy több nagy teljesítményű RISC CPU van, mind valamennyi helyi memóriával, egy megosztott főmemória, egy masszív RAM gyorstár a népszerű filmek számára, különféle tárolóeszközök, amelyek a filmeket tárolják, és valamilyen hálózati hardver, rendszerint egy optikai interfész egy OC–3 vagy magasabb szintű ATM (vagy SONET) hálózathoz. Ezeket az alrendszereket egy különösen nagy (legalább 1 GB/s) sebességű sín köti össze.

Most tekintsük át röviden a videokiszolgáló szoftverét. A CPU-kat a felhasználói



7.89. ábra. *Egy videokiszolgáló tárolóhierarchiája*

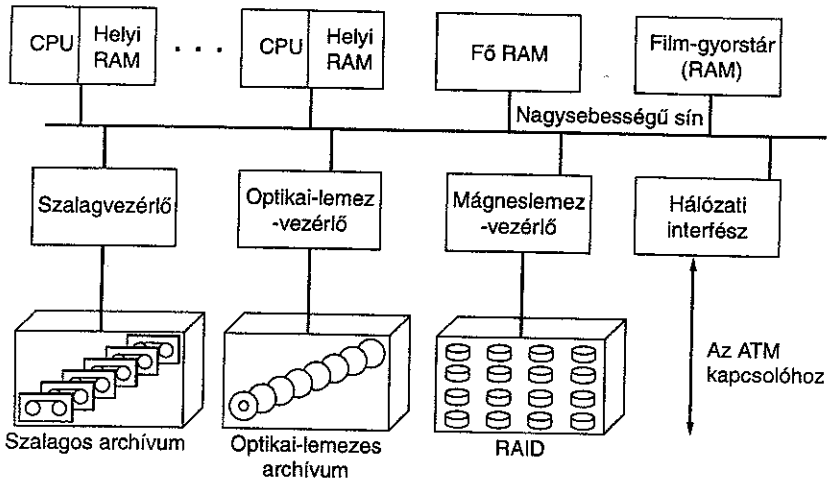
kérések fogadására, a filmek megkeresésére, az adatok eszközök közti mozgatására, az ügyfelek számlázására és még sok más feladatra használják. Ezek közül némelyik nem időkritikus, de némelyik igen, így néhány vagy az összes CPU-nak egy valós idejű operációs rendszert kell futtatnia, például egy valós idejű mikrokernelt. Ezek a rendszerek a feladatokat rendszerint kisebb feladatokba tördelik, amelyeknek mind ismert a határideje. Ezután az ütemező futtathat egy olyan ütemező algoritmust, mint például a legközelebbi határidő következőnek (nearest deadline next) vagy a monoton sebességű (rate monotonic) algoritmust (Liu és Layland, 1973).

A CPU szoftvere a kiszolgáló ügyfelek (tároló kiszolgálók és set-top dobozok) felé mutatott interfészének természetét is meghatározza. Két megoldás terjedt el. Az első egy hagyományos állományrendszer, amelyben az ügyfelek megnyithatnak, olvashatnak, írhatnak és lezárhatnak állományokat. A tárolási hierarchiától és a valós idejű megfontolásoktól eltekintve egy ilyen kiszolgáló állományrendszere az UNIX-éról mintázható.

A második interfész a videofelvevő modelljén alapul. A kiszolgálónak adott parancsok állományok megnyitását, lejátszását, megállítását, előretekerését és visszatekerését kérik. A UNIX modellhez képest az az eltérés, hogy ha egyszer kiadtak egy LEJÁTSZÁS (PLAY) parancsot, akkor a kiszolgáló állandó sebességgel pumpálja ki az adatot, és nincs új parancsokra szükség.

A videokiszolgáló szoftverének szíve a lemezkezelő szoftver. Ennek két fő feladata van: filmek elhelyezése a mágneslemezen, amikor azokat be kell hozni az optikai vagy szalagos tárolóról, és a sok kimeneti folyam lemezkéréseinek kezelése. A filmek elhelyezése azért fontos, mert ez nagyban befolyásolhatja a teljesítményt.

A lemeztár két lehetséges szervezési módja a lemezfarm és a lemeztömb. A **lemezfarm (disc farm)** esetén minden meghajtó néhány teljes filmet tartalmaz. Teljesítmény- és megbízhatósági okok miatt minden filmnek legalább két lemezen kell szere-



7.90. ábra. *Egy jellegzetes videokiszolgáló hardverének felépítése*

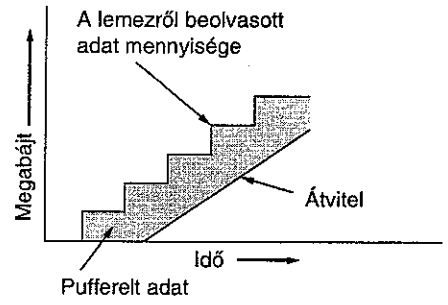
pelnie, esetleg többön is. A másik társszervezés a **lemeztömb (disc array),** vagy **RAID (Redundant Array Of Inexpensive Disks – olcsó lemezek redundáns tömbje),** ahol is minden filmet több meghajtóra terítenek szét, például a 0. blokk a 0. meghajtón, az 1. blokk az 1. meghajtón lesz és így tovább, az n–1. blokk az n–1. meghajtón lesz. Ezután a ciklus elölről kezdődik, az n. blokk a 0. lemezre kerül és így tovább. Ezt az elrendezést **csíkozásnak (striping)** nevezik.

Egy csíkozott lemeztömbnek számos előnye van egy lemezfarmmal szemben. Először is, mind az n meghajtó egyszerre működhet, így a teljesítmény n-szeresére növekedhet. Másodszor, redundánssá tehető minden n-es csoporthoz egy külön meghajtó hozzáadásával, ahol a rendundáns meghajtó a többi meghajtó blokkról blokkra vett KIZÁRÓ VAGY kapcsolatát tartalmazza, így lehetővé teszi a teljes helyreállítást egy meghajtó meghibásodása esetén. Végül, a terheléskiegyenlítés problémája megoldást nyert (nincs szükség kézi elhelyezésre, hogy elkerüljük azt, hogy minden népszerű film ugyanazon a meghajtón legyen). Másfelől viszont a lemeztömb szerveződések bonyolultabbak, mint a lemezfarm, és nagyban érzékenyek több meghibásodásra. Az olyan videofelvevő típusú műveletekhez is rosszul illeszkedik, mint egy film előrevagy visszatekerése. A két szerveződést összehasonlító szimulációs elemzést adnak: (Chervenak és mások, 1995).

A blokkelhelyezéshez szorosan kapcsolódik a blokkok megtalálása. Az a UNIX megoldás, amely szerint az i-node egy lemezblokkokból álló kiegyensúlyozatlan fára mutat, rendszerint elfogadhatatlan, mivel a videoállományok hatalmasak, így a legtöbb blokkot csak három indirekt blokkon keresztül tudjuk megtalálni, amely sok további lemezhozzáférést jelent (Tanenbaum, 1992). Ehelyett szokás a blokkokat egy egyszeresen vagy kétszeresen láncolt listában összekapcsolni. Néha egy UNIX stílusú (i-node) indexet is használnak a tetszőleges elérés lehetővé tételéhez.

A lemezszoftver másik feladata, hogy kiszolgálja az összes valós idejű kimeneti folyamot, és betartsa azok időzítési megkötéseit. Egy 25 kép/s-us MPEG–2 képfolyamnak körülbelül 14 KB-ot kell 40 ezredmásodpercenként elhoznia és átvinnie, de a valódi mennyiség ettől számottevően eltérhet, mert az I, P és B képeknek más a tömörítési arányuk. Ezért egy egyenletes kimeneti sebesség fenntartásához a folyam mindkét végén pufferelésre van szükség.

A 7.91. ábrán egy lépcső látható, amely az egy adott videofolyam számára lemez-



7.91. ábra. *Lemezpufferelés a kiszolgálóban*

ről beolvasott teljes adatmennyiséget mutatja (feltéve, hogy a film lemezen van). Ez diszkrét ugrásokkal megy felfele, minden beolvasott blokk egy ugrás. Ennek ellenére az átvitelnek egyenletesebb sebességgel kell történnie, így a lemezolvasási folyamatnak az átviteli folyamat előtt kell járnia. Az ábra sötétített területe azt az adatot mutatja, amelyet már beolvastak a lemezről, de még nem vittek át.

Rendszerint a lemezeket a felvonó algoritmus alapján ütemezik, amely azzal kezdődik, hogy a kar befelé mozog, és addig halad, amíg el nem éri a legbelső cilindert, az érintett cilinderek sorrendjében feldolgozva a kéréseket. Amikor végigért, akkor a kar megfordul és kifelé kezd mozogni, ismét sorban feldolgozva a függőben levő kéréseket. Bár ez az algoritmus minimalizálja a keresési időt, nem ad garanciát a valós idejű teljesítményre, így egy videokiszolgáló számára nem hasznos.

Egy jobb algoritmus az, hogy tartsunk számon minden videofolyamot, és készítsünk egy listát az általuk következőnek kért blokkokról. Ezeket a blokkszámokat azután rendezzük, és a blokkokat cilinderük szerint olvassuk be. Amikor az utolsó blokkot is beolvastuk, akkor elkezdődik a következő kör az összes folyam elején most levő blokk számának begyűjtésével. Ezeket szintén rendezzük, és cilindereik sorrendjében beolvassuk és így tovább. Ez az algoritmus minden folyamnak valós idejű teljesítményt biztosít, de a keresési időt is minimalizálja egy tiszta először jött-először kap algoritmushoz képest.

Egy másik szoftverkérdés a belépés ellenőrzése. Ha egy új folyamot kérnek, el lehet-e azt fogadni anélkül, hogy ez lerontaná a létező folyamok valós idejű teljesítményét? Egy, a döntés meghozására használható algoritmus a legrosszabb esetet vizsgálja meg, hogy lássa, hogy a k folyamról k+1-re lépés garantáltan lehetséges-e, a CPU, a RAM és a lemez ismert tulajdonságaira alapozva. Egy másik algoritmus csak a statisztikai tulajdonságokat vizsgálja.

Egy másik kérdés a kiszolgálónál, hogy hogyan kezeljék a megjelenítést egy előre- vagy visszatekerés közben (hogy az emberek vizuálisan kereshessenek). A D képek biztosítják az ehhez szükséges információt az MPEG–1-ben, de, hacsak nincsenek különleges módon megjelölve és tárolva, a kiszolgáló nem képes azokat az egész folyam dekódolása nélkül megtalálni, és a kiszolgálók rendszerint nem végeznek MPEG dekódolást átvitel közben. Az MPEG–2 esetén valamely más mechanizmusra lesz szükség, legalább azért, hogy az I képeket könnyen megtalálhatóvá és dekódolhatóvá tegyük.

Végül a titkosítás is egy kérdés. Amikor a filmeket többesküldéssel viszik át (vagyis a helyi elosztó hálózat egy kábel-tévés rendszer), akkor titkosításra van szükség, hogy biztosítsák, hogy csak a fizető ügyfelek nézhessék a filmet. Két megközelítés lehetséges: az előre titkosítás és a röptében titkosítás. Ha a filmeket titkosítva tárolják, akkor bárki, aki egy film kulcsát megtudta, képes lehet azt ingyen megnézni, mivel minden alkalommal ugyanazt a kulcsot használják. Minden folyam külön titkosítása biztonságosabb, de több számítási erőforrást is igényel.

A kulcsgazdálkodás is kérdés. A szokásos megközelítés az, hogy egy egyszerű algoritmussal röptében titkosítsanak, de gyakran cseréljék a kulcsot, így még ha egy támadó feltöri is a kulcsot 10 perc alatt, addigra az már elavul.

Az elosztó hálózat

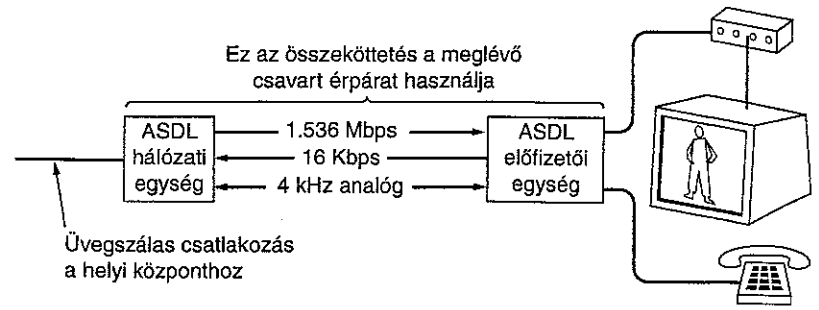
Az elosztó hálózat a forrás és a cél közt levő kapcsolók és vonalak halmaza. Ahogy a 7.88. ábrán láttuk, ez egy SONET vagy ATM hálózatból (vagy SONET hordozójú ATM hálózatból) áll, amely a helyi elosztó hálózathoz kapcsolódik. Rendszerint a gerinchálózat kapcsolt működésű, a helyi elosztó hálózat pedig nem az.

A gerinchálózatra háruló fő követelmények a nagy sávszélesség és az alacsony dzsitter. Egy tiszta SONET gerinchálózattal ezeket triviálisan el lehet érni: a sávszélesség garantált, dzsitter pedig nincs, mivel a hálózat szinkron működésű. Egy ATM vagy SONET hordozójú ATM gerinchálózaton a szolgálat minősége nagyon fontos. Ezt a lyukas vödör algoritmus és az 5. fejezetben nagy részletességgel tanulmányozott többi módszer kezeli, így azt a tárgyalást itt nem ismételjük meg. Az ATM gerinchálózatok feletti valós idejű MPEG-ről a további tudnivalókat lásd: (Dixit és Skelly, 1995; Morales és mások, 1995). A következőkben a helyi elosztó hálózatra fogunk összpontosítani, mivel ezt a témát eddig alig érintettük.

A helyi elosztás nagymértékben kaotikus, mivel a különböző társaságok különböző hálózatokat próbálnak ki különböző körzetekben. A telefontársaságok, a kábel-tévés társaságok, és az új belépők mind meg vannak arról győződve, hogy aki először eléri a célt, az lesz a nagy nyertes, így most a telepítésre kerülő módszerek elburjánzása figyelhető meg. A hálózati videózás négy fő elosztó tervét az ADSL, FTTC, FTTH és HFC betűszavak jelzik. Sorban mindegyiket el fogjuk magyarázni.

Az **ADSL (Asymmetric Digital Subscriber Line – aszimmetrikus digitális előfizetői vonal)** volt a telefonipar első nevezettje a helyi elosztók versenyébe. Az ötlet az, hogy gyakorlatilag az Egyesült Államok, Európa és Japán minden házába vezet egy csavart réz érpár (az analóg telefonszolgálat miatt). Ha ezeket a huzalokat használni lehetne a hálózati videózáshoz, akkor a telefontársaságok besöpörhetnék a nyereséget.

A gond természetesen az, hogy ezek a huzalok nem képesek még az MPEG–1-et sem támogatni az átlagosan 10 km-es hosszukkal, nemhogy az MPEG–2-t. Az ADSL megoldás a digitális jelfeldolgozás eredményeit használja fel arra, hogy a visszhangokat és egyéb vonalzajokat elektronikusan elnyomja. Ahogy a 7.92. ábra mutatja, minden ADSL előfizetőnek adnak egy házon belüli ADSL előfizetői egységet, amely egy digitális jelfeldolgozó chipet tartalmaz. A telefon és a set-top doboz az ADSL egység-



7.92. ábra. *Az ADSL, mint helyi elosztó hálózat*

hez kapcsolódik. Az előfizetői hurok másik végéhez egy másik ADSL egység kapcsolódik. Ez lehet a telefontársaság helyi központjában, vagy ha az előfizetői hurok túl hosszú, akkor a ház szomszédságában egy üvegszál végén.

Az ADSL–1 egy 1,536 Mb/s-os (T1 mínusz a 193. bit) lefelé irányuló csatornát, de csak egy 16 kb/s-os felfele irányuló csatornát kínál. Ezeken kívül a régi 4 kHz-es analóg telefoncsatorna (vagy néhány esetben két N-ISDN digitális csatorna) is ott van. Az ötlet az, hogy a felfelé irányuló összeköttetésnek elegendő a sávszélessége arra, hogy a felhasználó filmeket rendelhessen meg, és a lefele irányuló összeköttetésnek elegendő a sávszélessége arra, hogy azokat MPEG–1 kódolva el lehessen küldeni. Az ADSL inkább egy gyors-de-piszkos trükk, mint egy hosszú távú megoldás, ennek ellenére egyes városokban már telepítik. ADSL–2-nek és ADSL–3-nak nevezett javított változatokon is dolgoznak. Az utóbbi lehetővé teszi a kb. 2 km-nél nem hosszabb előfizetői hurkokon az MPEG–2-t.

A második telefontársasági tervezet az **FTTC (Fiber To The Curb – üvegszál az elosztó dobozig).** Ezt a tervezetet a 2.23.(a) ábrán láthattuk. FTTC esetén a telefontársaság üvegszálat fektet a helyi központból minden lakókörzetig, amely egy **ONU-nak (Optical Network Unit – optikai hálózategység)** nevezett eszközben végződik. Az ONU-t a 2.23.(a) ábrán „csatlakozódoboz" jelöli. Egy ONU-ban 16 előfizetői hurok végződhet. Ezek a hurkok most már olyan rövidek, hogy lehetséges duplex T1 vagy T2 összeköttetéseket futtatni rajtuk, amely lehetővé teszi az MPEG–1-et, illetve az MPEG–2-t. Ezen kívül az otthon dolgozók és kisvállalkozások számára lehetővé válik a videokonferencia, mivel az FTTC szimmetrikus.

A telefontársaságok harmadik megoldása, hogy üvegszálat vigyenek minden házba. Ezt **FTTH-nak (Fiber To The Home – üvegszál otthonra)** nevezik. Ebben a tervezetben mindenkinek lehet egy OC–1, OC–3 vagy akár magasabb hordozója is, ha az szükséges. Az FTTH nagyon drága, és még évekig nem fog bekövetkezni, de nyilvánvalóan új lehetőségek nagy tárházát nyitja meg, amikor végül is igen.

Az ADSL, az FTTC és az FTTH mind kétpont alapú helyi elosztó hálózatok, amely nem meglepő, mivel a jelenlegi telefonrendszer is így szerveződik. Egy teljesen más megközelítés a **HFC (Hybrid Fiber Coax – hibrid üvegszál-koax).** Ez egy előnyben részesített megoldás, amelyet a kábel-tévé szolgáltatók mostanában telepítenek. A történet a következő: a jelenlegi 300–450 MHz-es koax kábeleket 750 MHz-es koax kábelek fogják felváltani, amely a kapacitást 50–75 6 MHz-es csatornáról 125-re bővíti. A 125 csatorna közül hetvenötöt analóg televíziós átvitelre fognak felhasználni.

Az 50 új csatorna mindegyikét QAM–256 modulációs eljárással modulálni fogják, amely csatornánként kb. 40 Mb/s-ot biztosít, összesen 2 Gb/s új sávszélességet adva. A fejállomásokat közelebb viszik a lakókörzethez, így minden kábel csak 500 házat fog átfogni. Egy egyszerű osztás megmutatja, hogy ezután minden házhoz egy dedikált 4 Mb/s-os csatornát lehet hozzárendelni, amelyet valamilyen kombinációban MPEG–1 programok, MPEG–2 programok, analóg és digitális telefónia, felfelé irányuló adatok és így tovább használhatnak.

Bár ez gyönyörűen hangzik, de megköveteli a kábelszolgáltatótól, hogy az összes létező kábelt lecserélje 750 MHz-es koaxra, új fejállomásokat telepítsen, és minden egyirányú erősítőt eltávolítson – röviden, hogy lecserélje az egész kábel-tévés rendszert. Következésképpen itt az új infrastruktúra mennyisége összehasonlítható azzal,

amennyire a telefontársaságoknak van szükségük az FTTC-hez. Mindkét esetben, a helyi hálózati szolgáltatónak üvegszálat kell vinnie a lakossági környékekig. Mindkét esetben az üvegszál egy optoelektromos átalakítóban végződik. Az FTTC-nél a végső szegmens egy kétpontos előfizetői hurok, amely sodrott érpárat használ. A HFC-nél a végső szegmens egy megosztott koaxiális kábel. Műszakilag ezek a rendszerek nem különböznek annyira, amennyire azt az egyes rendszerek védelmezői állítják.

Ennek ellenére van egy valódi különbség, amire érdemes rámutatnunk. A HFC megosztott médiumot használ mindenfajta forgalomirányítás és kapcsolás nélkül. Bármely, a kábelre helyezett információt bármely előfizető minden további nélkül leemelhet. Az FTTC-nek, amely teljesen kapcsolt, nincs meg ez a tulajdonsága. Ennek eredményeképpen a HFC működtetői azt akarják, hogy a videokiszolgálók titkosított folyamokat küldjenek ki, hogy a filmért nem fizető nézők ne láthassák azt. Az FTTC működtetői nem akarnak különösebben titkosítani, mivel ez bonyolítja a rendszert, csökkenti a teljesítményt, és nem ad további biztonságot az ő rendszerükben. Egy videokiszolgálót üzemeltető társaság szempontjából jó ötlet titkosítani vagy sem? Egy telefontársaság vagy fiókvállalatai, vagy partnerei által működtetett kiszolgáló szándékosan úgy dönthet, hogy nem titkosítja a videofilmjeit, a hatékonyságot nevezve meg okként, de valójában azért, hogy veszteséget okozzon HFC-s versenytársainak.

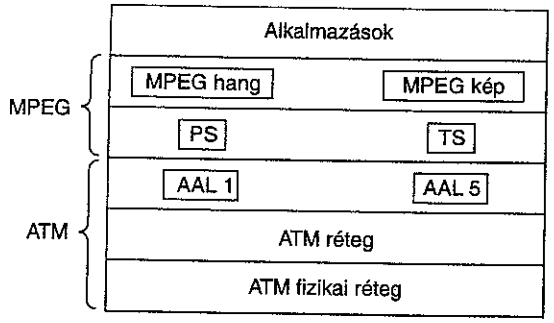
Ezen elosztó hálózatok közül bármelyiknél valószínű, hogy minden lakossági környéken el fognak helyezni egy vagy több tároló kiszolgálót. Ezek tulajdonképpen csak a fentebb tárgyalt videokiszolgálók kisebb változatai. Ezen helyi kiszolgálók nagy előnye, hogy mivel a helyi elosztó hálózatok rendszerint rövidek és általában nem kapcsoltak, nem visznek be dzsittert, mint ahogy tenné azt egy ATM gerinchálózat.

Ezeket dinamikusan vagy foglalás útján lehet előre feltölteni. Például, amikor egy felhasználó kiválaszt egy filmet, annak első percét a helyi kiszolgálóhoz egy OC–3-as vonalon 2 másodpercnél kevesebb idő alatt le lehet tölteni. 55 másodperc után a következő percet lehetne leszállítani a helyi kiszolgálónak 2 másodperc alatt és így tovább. Ily módon az ATM gerinchálózat forgalmának már nem kell dzsittermentesnek lennie, ezáltal lehetséges a drága CBR szolgálat helyett ABR szolgálatot használni.

Ha az emberek jó előre megmondják a szolgáltatónak, hogy mely filmeket akarják, akkor ezek csúcsidőn kívül letölthetők a helyi kiszolgálóra, még nagyobb megtakarításokat eredményezve. Ez a megfigyelés valószínűleg arra fogja késztetni a hálózatüzemeltetőket, hogy légitársaságok igazgatóit csábítsák át a díjszabások kialakításához. El lehet képzelni olyan tarifákat is, amelyben a keddi vagy csütörtöki, du. 6 óra előtti vagy este 11 óra utáni megtekintésre 24–72 órával előbb megrendelt filmekre 27 százalékos kedvezmény jár. Az olyan szerdai napokon, amelynek dátuma prímszám, történő megtekintésre a hónap első vasárnapján reggel 8 előtt megrendelt filmekre 43 százalékos kedvezmény jár és így tovább.

A hálózati videózáshoz használatos protokollkészlet kérdése még mindig a levegőben lóg. Nyilván az ATM a választandó technológia, de mely adaptációs protokollt kellene használni? Az AAL 1-et a videóhoz tervezték, így ez esélyes jelölt, de ez a CBR szolgálati osztályba tartozik. Az esetlegesen igényelt legnagyobb sávszélesség lefoglalása költséges, különösen, mivel az MPEG eredendően VBR forgalom, így a virtuális áramkört túl kell méretezni.

Az AAL 2 nincs befejezve (és valószínűleg soha nem is lesz), és az AAL 3/4 túl



7.93. ábra. *A hálózati videózás egy protokollkészlete*

nehézkes, így az egyetlen megmaradó versenyző az AAL 5. Ez nem kötődik a CBR szolgálathoz, és minden üzenetben egy nagy MPEG blokk elküldése igen hatékony lenne, a felhasználó sávszélességének majdnem 100 százalékát a képfolyam kapná. A hátrányokról is szólva, az AAL 5 végez hibadetektálást. Nem túl vonzó egy egész blokkot eldobni egy 1 bites hiba miatt, különösen mivel a legtöbb hiba az adat közepén elhelyezkedő, egybites hiba. Következésképpen némi mozgolódás van az irányban, hogy az AAL 5-öt úgy változtassák meg, hogy az alkalmazásoknak lehetőségük legyen az összes blokkot lekérni egy bittel együtt, amely megmondja, hogy az ellenőrző összeg helyes volt-e vagy sem.

A hálózati videózás fentebb felvázolt protokollkészletét a 7.93. ábra mutatja be. Az AAL réteg felett láthatjuk az MPEG program- és szállítási folyamait. Ezután jön az MPEG hang- és képkódolása, illetve dekódolása. Végül az alkalmazás helyezkedik el legfelül.

Készülék (set-top) dobozok

A fenti helyi elosztó módszerek közül mindegyik végül is egy vagy több MPEG folyamot juttat el az otthonokba. Hogy ezeket dekódolni lehessen és meg lehessen nézni azokat, szükséges egy hálózati interfész, egy MPEG dekóder és más villamos áramkörök. Itt két megközelítés lehetséges.

Az első megoldásnál az emberek a személyi számítógépeiket használják a filmek dekódolására és megtekintésére. Ez megköveteli egy pár speciális chipet tartalmazó speciális, számítógépbe berakható kártya és a helyi elosztó hálózathoz való csatlakozáshoz egy csatlakozó megvételét. A filmek ezután a számítógép monitorán jelenhetnek meg, esetleg akár egy ablakban is. Ezt akár set-bottom doboznak is nevezhetnénk, mivel a számítógépeknél a doboz rendszerint a monitor alatt van, nem pedig felette. Ez a megközelítés olcsóbb (csak egy berakható kártyára és a szoftverre van szükség), egy nagyfelbontású, nem sorugrásos monitort használ, kifinomult egér alapú felhasználói felülete van, és könnyen integrálható a WWW-vel, és más számítógép-orientált információs és szórakoztatási forrásokkal. Másfelől a PC-knek rendszerint kicsi a képernyőjük, a nappali helyett a dolgozószobában helyezkednek el, és rendszerint egyszerre egy személy használja azokat. Számottevően kevesebb fényt is bocsátanak ki, mint a televíziókészülékek.

A második megközelítésben a helyi hálózat üzemeltetője bérbe ad vagy elad a felhasználónak egy készülék (set-top) dobozt, amely a hálózathoz és a televíziókészülékhez csatlakozik. Ennek a megközelítésnek megvan az az előnye, hogy mindenkinek van televíziója, de nem mindenkinek van PC-je, és az emberek birtokában sok olyan PC van, amelyek régiek, különlegesek, vagy más ok miatt alkalmatlanok az MPEG dekódolásra. Továbbá a televízió rendszerint egy olyan szobában helyezkedik el, amelyet a csoportos nézésre szántak.

A hátránya az, hogy a megjelenítő egy sorugrásos, alacsony felbontású képernyő (amely alkalmatlanná teszi szövegalapú anyagokhoz, mint amilyen a WWW is). Ezenkívül rettenetes a felhasználói felülete (a távvezérlő), amely gyakorlatilag lehetetlenné tesz a felhasználó számára egyszerű menükből történő választáson kívül bármit is. Még egy film nevének a begépelése is fájdalmas, nemhogy párbeszédbe bocsátkozni a kiszolgálóval arról, hogy keressen meg minden filmet, amelyet egy bizonyos színész, rendező vagy produkciós társaság készített egy bizonyos periódusban. Végül, a megkívánt teljesítményű set-top dobozokat nem egyszerű elfogadható árért (amelyet pár száz dollárnak gondolnak) előállítani.

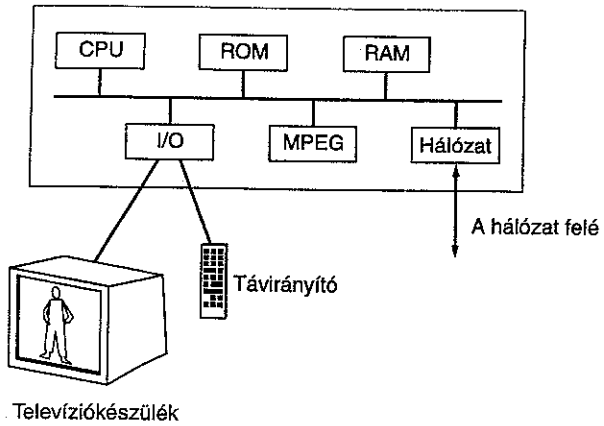
Mindezen tényezőket figyelembe véve a legtöbb hálózati videorendszer a set-top dobozos modellre szavazott, elsődlegesen azért, mert a tömeges piacra gyártók utálnak bármilyen lehetséges ügyfelet (a PC nélküli embereket) kizárni. Továbbá esetleg pénzt lehet keresni set-top dobozok kölcsönzésével vagy eladásával. Ám a PC-be dugható kártyák piaca is elég nagy, így kétségkívül ezeket a kártyákat is elő fogják állítani.

A set-top doboz elsődleges feladatai a helyi hálózathoz való illeszkedés, az MPEG jel dekódolása, a hang- és képfolyamok szinkronizálása, a televíziókészülék számára egy kompozit NTSC, PAL vagy SECAM jel előállítása, figyelni a távirányítóra, és kezelni a felhasználói felületet. A további funkciók közt lehetnek a HIFI-tornyokhoz, telefonokhoz és más eszközökhöz való illeszkedés. Az iparon belül heves harc dúl akörül, hogy mennyi feladatot lásson el a set-top doboz és mennyit a hálózat. Hogy ez merre dől el, az majd elválik.

Egy egyszerű set-top doboz felépítése a 7.94. ábrán látható. Az eszköz egy CPU-ból, ROM-ból, RAM-ból, B/K vezérlőből, MPEG dekódolóból és hálózati interfészből áll. Esetlegesen egy biztonsági chipet hozzá lehet venni a bejövő filmek megfejtésére és a kimenő üzenetek (például az otthoni vásárlás esetén hitelkártyaszámok) titkosítására.

A hálózati videózás egyik fontos kérdése a hang és kép szinkronizációja, és a dzsitter kezelése. További 500 KB RAM hozzáadása egy másodpercnyi MPEG–2 pufferelést tesz lehetővé, de további költséggel jár, és mindez egy olyan eszközben, amelyet a gyártók legfeljebb pár száz dollárért remélnek eladni.

Mivel a set-top doboz is csak egy számítógép, szoftverre lesz szüksége, valószínűleg egy mikrokernel alapú, ROM-ban tárolt valós idejű operációs rendszerre. Hogy a rugalmasságot és az alkalmazkodóképességet biztosítsuk, valószínűleg jó ötlet lehetővé tenni más szoftver letöltését a hálózatról. Ez a lehetőség azután felveti a problémát, hogy mi történik akkor, amikor egy MIPS alapú set-top doboz tulajdonosa egy SPARC



7.94. ábra. *Egy egyszerű set-top doboz hardverének felépítése*

alapú set-top dobozokra írt játékot akar játszani? Egy értelmezett nyelv, mint amilyen a Java, használata megoldja a kompatibilitási gondokat, de súlyosan csökkenti a teljesítményt egy olyan valós idejű környezetben, ahol kritikus a nagy teljesítmény.

Szabványok

A hálózati videózás gazdasági vonzatait nem lehet figyelmen kívül hagyni. Egy nagy videokiszolgáló könnyen többe kerülhet, mint egy mainframe számítógép – 10 millió dollárba biztosan. Tegyük fel, hogy 100 000 otthont lát el, amelyek közül mindegyik kibérelt egy 300 dolláros set-top dobozt. Adjunk még hozzá 10 millió dollár értékű hálózati felszerelést, és 4 éves amortizálódási időt, és a rendszernek otthononként és havonta 10 dollárt kell előállítania. Filmenként 5 dollárral számolva mindenkinek havonta két filmet kellene vennie ahhoz, hogy az üzemeltető nullszaldós legyen (leszámítva a fizetéseket, a marketinget és más költségeket). Hogy ez meg fog-e történni, az közel sem nyilvánvaló.

Milyen technológiát fog a gerinchálózat használni (SONET, ATM, SONET+ATM)?
Milyen sebességgel fog futni a gerinchálózat (OC–3, OC–12)?
Milyen rendszerű lesz a helyi elosztó hálózat (HFC, FTTC)?
Mennyi felfelé irányuló sávszélesség lesz (16 kb/s, 1.5Mb/s)?
Titkosítva lesznek-e a filmek, és ha igen, hogyan?
Lesz-e hibajavítás (kötelező, opcionális, nem)?
Ki lesz a set-top doboz tulajdonosa (felhasználó, a hálózat üzemeltetője)?
Része lesz-e a telefónia a rendszernek (analóg, N-ISDN)?
Támogatják-e a nagyfelbontású, hipertextes alkalmazásokat (WWW)?

7.95. ábra. *Néhány terület, ahol szabványokra van szükség*

A fent megadott számokat sokféleképpen át lehet rendezni (pl. havonként 6 dollárt számítva a set-top doboz bérléséért, és 2 dollárt filmenként), és a költségek állandóan változnak, de nyilvánvalónak kell lennie, hogy tömeges piac nélkül a hálózati videózás nem lehet gazdaságilag értelmes vállakozás. Hogy egy tömeges piac létrejöjjön, ahhoz szükséges, hogy a rendszer minden része szabványosítva legyen. Ha minden videoszolgáltató, hálózati üzemeltető és set-top dobozt gyártó saját interfészt tervez, semmi nem fog a rendszer többi részével együttműködni. Eddig az egyetlen szabvány, amelyben mindenki megegyezett, az az MPEG–2 használata a képkódolásra. Minden másért még szabad a pálya. A 7.95. ábrán felsoroltunk néhány olyan kérdést, amelyet meg kell válaszolni, mielőtt egy országos rendszert meg lehetne építeni.

Ha már minden területet szabványosítottak, akkor könnyen elképzelhetjük, hogy sok gyártó olyan termékeket fog előállítani, amelyek egy dobozból állnak, amelyekben van telefoncsatlakozó, monitor, billentyűzet és egér. Ezeket videózásra, számítógépezésre vagy esetleg egyszerre mindkettőre lehet majd használni. Ekkor a számítástechnika, távközlés és szórakoztatási iparok sokat emlegetett konvergenciája valóság lesz.

7.7.5. MBone – Digitális adatszóró gerinchálózat

Miközben mindezek az iparágak nagyszabású – és nagy nyilvánosságot kapott – terveket készítenek a jövőbeni nemzet(köz)i digitális hálózati videózáshoz, az Internet közösség csendesen megvalósította a saját digitális multimédia rendszerét, az **MBone-t (Multicast Backbone – digitális adatszóró hálózat).** Ebben a szakaszban egy rövid áttekintést adunk arról, hogy mi is ez, és hogyan működik. Egy egész könyv az MBone-ról: (Kuzmar, 1996). Cikkek az MBone-ról: (Eriksson, 1994; Macedonia és Brutzman, 1994).

Az MBone-ra úgy gondolhatunk, mint az Internet rádiójára és televíziójára. A hálózati videózástól eltérően, ahol a hangsúly egy kiszolgálón tárolt, előre tömörített filmek lehívásán és megnézésén volt, az MBone élő hang és kép az Interneten keresztül az egész világra kiterjedő, digitális formában történő műsorszórásához használatos. Már 1992 elején megkezdte működését. Sok tudományos konferenciát, különösen az IETF gyűléseit, közvetítettek, a hírértékű tudományos események mellett, mint amilyen egy űrsikló indítása. Egyszer egy Rolling Stones-koncertet is közvetítettek az MBone-on keresztül. Hogy ez elmegy-e hírértékű tudományos eseménynek, az vitatható. Azon emberek számára, akik digitálisan rögzíteni akarnak egy MBone adást, elérhető ezt megvalósító szoftver (Holfelder, 1995).

Az MBone-t érintő legtöbb kutatás azzal foglalkozott, hogy hogyan lehet hatékony többesküldést megvalósítani a (datagram alapú) Interneten. Keveset tettek a hang- vagy képkódolás terén. Az MBone források szabadon kísérletezhetnek az MPEG-gel vagy bármely más kódolási eljárással. Nincsenek Internet szabványok a tartalomra vagy a kódolásra.

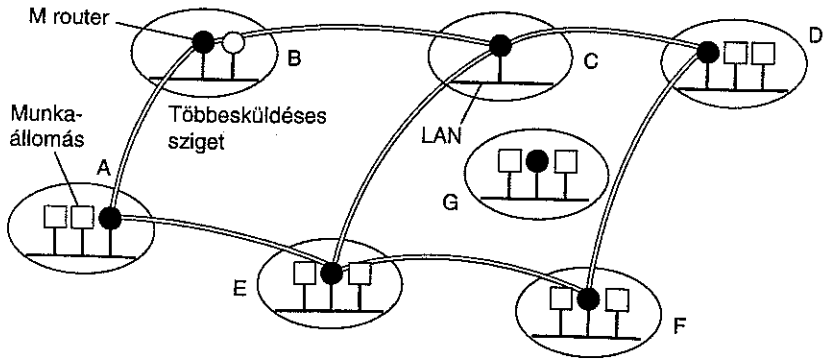
Műszakilag az MBone egy virtuálisan az Internetre terített hálózat. Többesküldésre

képes szigetekből áll, amelyeket alagutak kötnek össze, mint az a 7.96. ábrán látszik. Ezen az ábrán az MBone hat szigetből áll. Minden sziget (rendszerint egy LAN vagy összekapcsolt LAN-ok egy csoportja) hardver szintű adatszórást biztosít a hosztjai számára. Az alagutak terjesztik az MBone csomagokat a szigetek közt. Egy szép napon, amikor az összes forgalomirányító képes lesz többesküldéses forgalmat közvetlenül kezelni, akkor nem lesz szükség erre a felépítményre, de ebben a pillanatban ez is megteszi.

Minden sziget egy vagy több különleges routert tartalmaz, amelyeket **mrouter-eknek (multicast routers – többesküldéses router)** neveznek. Ezek némelyike valódi router, de a legtöbb csak egy UNIX munkaállomás, amelyen különleges felhasználói szintű szoftver fut (amit a rendszergazda indított el). Az mroutereket logikailag csatornák kötik össze. A múltban az MBone csomagokat alagút típusú átvitellel továbbították mroutertől mrouterig (rendszerint egy vagy több olyan routeren keresztül, amely nem tudott az MBone-ról), laza forrás általi forgalomirányítást használva. Manapság az MBone csomagokat IP csomagokba ágyazzák be, és mint rendes egyesküldéses csomagokat küldik el a cél-mrouter IP címére. Viszont ha az összes közbeeső router támogatja a többesküldést, akkor az alagút típusú átvitelre nincs szükség.

Az alagutakat kézzel állítják be. Rendszerint egy alagút egy olyan úton megy, amelyhez van fizikai összeköttetés is, de ez nem követelmény. Ha véletlenül kiesik egy, valamely alagút alapját képező fizikai út, akkor az alagutat használó mrouterek nem is fogják azt észrevenni, mivel az Internet automatikusan átirányítja a köztük zajló IP forgalmat más vonalakra.

Amikor egy új sziget jelenik meg, és az szeretne csatlakozni az MBone-hoz, az adminisztrátora elküld egy, a létezését bejelentő üzenetet az MBone levelezési listára. A közeli állomások adminisztrátorai ezután kapcsolatba lépnek vele, hogy elrendezzék az alagutak felállítását. Néha a meglevő alagutakat újrairányítják, hogy az új sziget kihasználásával optimalizálják a topológiát. Végül is az alagutak fizikailag nem léteznek. Ezeket csak az mrouterek táblázatai definiálják, és ezen táblázatok megváltoztatásával mozgathatók, törölhetők vagy újak is hozzáadhatók. Rendszerint minden, az MBone-on rajta levő országnak van egy gerinchálózata, amelyhez területi szigetek



7.96. ábra. *Az MBone többesküldéses szigetekből áll, amelyeket alagutak kötnek össze*

kapcsolódnak. Az MBone rendszerint úgy van beállítva, hogy egy vagy két alagút szeli keresztül az Atlanti- és a Csendes-óceánt, így lesz az MBone globális méretű.

Ezért az MBone-t bármely pillanatban szigetek és alagutak egy különleges topológiája alkotja, függetlenül az éppen használt többesküldéses címek számától és attól, hogy ki nézi vagy hallgatja azokat. Ez a helyzet nagyon hasonlít egy rendes (fizikai) alhálózathoz, így a rendes forgalomirányító algoritmusok alkalmazhatók rá. Következésképpen az MBone kezdetben egy **DVMRP-nek (Distance Vector Multicast Routing Protocol – távolságvektor alapú többesküldéses forgalomirányító protokoll)** nevezett, a Bellman–Ford távolságvektor algoritmuson alapuló forgalomirányító algoritmust használt. Például a 7.96. ábrán a *C* sziget *A*-ba *B*-n vagy *E*-n (vagy elképzelhető, hogy *D*-n) keresztül irányíthat. A döntését azon értékek alapján hozza meg, amelyeket a csomópontok adnak az ő *A*-tól való távolságukról, majd a saját távolságát is hozzáadja. Ily módon minden sziget meghatározza a legjobb útvonalat minden másik szigethez. Az útvonalakat viszont valójában nem így használják, ahogy azt rövidesen látni fogjuk.

Most vegyük szemügyre, hogyan is történik valójában a többesküldés. Hogy egy hang- vagy képműsort szórni lehessen, a forrásnak először szereznie kell egy D osztályú többesküldéses címet, amely mint egy adó frekvenciája vagy egy csatorna száma működik. A D osztályú címeket egy program foglalja le, amely egy adatbázisban keres szabad többesküldéses címek után. Egyszerre több többesküldés is történhet, és egy hoszt „befoghatja" az őt érdeklőt, ha a megfelelő többesküldéses címre figyel.

Minden mrouter periodikusan kiküld egy IGMP adatszórásos csomagot a szigetére korlátozva, amely megkérdezi, hogy kit melyik csatorna érdekel. Az olyan hosztok, amelyek (továbbra is) fogni akarnak egy vagy több csatornát, egy másik IGMP csomagot küldenek vissza válaszul. Ezeket a válaszokat időben széthúzzák, hogy a helyi LAN-t ne terheljék túl. Minden mrouternek van egy táblázata arról, hogy melyik csatornákat kell ráadnia a LAN-jára, elkerülvén ezzel a senki által nem hallgatott többesküldéses csatornák által okozott sávszélesség-veszteséget.

A többesküldések a következőképpen terjednek az MBone-on: Amikor egy hang- vagy képforrás egy új csomagot állít elő, akkor azt a hardveres többesküldést használva ráadja a helyi szigetre. Ezt a csomagot a helyi mrouter felveszi, majd minden alagútba bemásolja azt, amelyhez csatlakozik.

Minden mrouter, amely ilyen csomagot kap egy alagúton keresztül, leellenőrzi, hogy a csomag a legjobb útvonalon érkezett-e, vagyis azon, amelyet a táblázata szerint használnia kellene a forrás (mint cél) eléréséhez. Ha a csomag a legjobb útvonalon jött, akkor az mrouter bemásolja a csomagot az összes többi alagútjába. Ha a csomag egy szuboptimális útvonalon jött, akkor eldobják. Ezért például a 7.96. ábrán, ha *C*-nek táblázata szerint *B*-t használva kell *A*-hoz eljutnia, akkor ha egy *A* felől érkező többesküldéses csomag *B*-n keresztül éri el *C*-t, a csomagot *D* és *E* felé kimásolja. Viszont, ha *A* egy többesküldéses csomagja *C*-t *E*-n keresztül éri el (amely nem a legjobb út), akkor egyszerűen eldobja. Ez az algoritmus egyszerűen az a visszairányú továbbítási algoritmus, amelyet az 5. fejezetben láttunk. Bár ez nem tökéletes, de elég jó és nagyon egyszerű megvalósítani.

Az Internet elárasztásának megelőzésére a visszairányú továbbításon kívül az IP *Élettartam* mezejét is felhasználják a többesküldés érvényességi körének behatárolására. Minden csomag valamely értékkel indul el (amelyet a forrás határoz meg). Min-