

crajiilor, legea asigură în mod explicit dreptul unei persoane acuzate de a-și chema acuzatorul în fața Curții. Acuzațiile anonime nu pot servi drept probă.

Pe scurt, rețelele de calculatoare, asemenea industriei tipografice cu 500 de ani în urmă, permit cetățenilor obișnuiți să-și lanseze opiniile prin mijloace diferite și către audiente diferite față de cele de până acum. Această libertate nou descoperită aduce cu ea probleme nerezolvate de ordin social, politic și moral.

Odată cu binele vine și răul. Viața pare a fi construită astfel. Internetul oferă posibilitatea de a găsi repede informații, dar multe dintre ele sunt greșit informate, tendențioase sau chiar complet eronate. Sfatul medical pe care tocmai l-ați luat de pe Internet poate să vină de la un laureat al premiului Nobel sau de la un repetent din liceu. Rețelele de calculatoare au introdus de asemenea și noi tipuri de comportamente antisociale și infracționale. Transmiterea electronică a fleacurilor și gunoaielor (eng.: junk) a devenit parte din viață pentru că oamenii au colectionat milioane de adrese pe care le vând pe CD-ROM-uri așa-zisilor agenți de marketing. Mesajele care au un conținut activ (de obicei programe sau macrourori care se execută pe mașina receptorului) pot avea efecte distructive.

Furtul de identitate devine o problemă serioasă, pentru că hoții colectează destule informații despre o potențială victimă pentru a putea obține cărți de credit și alte documente în numele acesteia. În fine, posibilitatea de a transmite digital muzică și filme a deschis ușa pentru încălcarea masivă a drepturilor de autor care sunt greu de depistat și pedepsit.

Multe dintre aceste probleme puteau fi rezolvate dacă industria de calculatoare ar fi luat în serios securitatea calculatoarelor. Dacă toate mesajele erau criptate și autentificate, ar fi fost mai greu să se comită nedreptăți sau furturi. Această tehnologie este bine conturată și o vom studia în detaliu în cap. 8. Problema este că vânzătorii de hardware și aplicații software știu că introducerea unor atribuții de securitate costă bani, iar cumpărătorii nu solicită astfel de atribuții. Mai mult, un număr substanțial de probleme este determinat de aplicațiile care funcționează cu erori, ceea ce se întâmplă pentru că producătorii adaugă din ce în ce mai multe facilități programelor lor, ceea ce înseamnă inevitabil mai mult cod și de aceea mai multe erori. O taxă pentru noile facilități ar putea ajuta, dar ar face produsele greu de vândut în anumite segmente de piață. Plata unei despăgubiri pentru programele care funcționează eronat ar fi foarte cinstită, doar că ar duce la faliment întreaga industrie software chiar din primul an.

## 1.2 HARDWARE-UL REȚELEI

A venit acum timpul să ne îndreptăm atenția de la aplicațiile și problemele sociale ale interconectării (partea distractivă) la aspectele tehnice care intervin în proiectarea rețelelor (partea serioasă de lucru). Deși nu există o taxonomie general acceptată în care pot fi încadrate toate rețelele de calculatoare, sunt extrem de importante două criterii: tehnologia de transmisie și scară la care operează rețeaua. Vom examina pe rând fiecare din aceste aspecte.

În principal există două tipuri de tehnologii de transmisie care se folosesc pe scară largă. Acestea sunt:

1. Legături cu difuzare.
2. Legături punct-la-punct.

**Rețelele cu difuzare** au un singur canal de comunicații care este partajat de toate mașinile din rețea. Orice mașină poate trimite mesaje scurte, numite în anumite contexte **pachete**, care sunt primite de toate celelalte mașini. Un câmp de adresă din pachet specifică mașina căreia îi este adresat pachetul. La recepționarea unui pachet, o mașină controlează câmpul de adresă. Dacă pachetul îi este adresat, mașina îl prelucreză; dacă este trimis pentru o altă mașină, pachetul este ignorat.

Să considerăm, ca analogie, că cineva se află la capătul unui corridor cu multe încăperi și strigă „Watson, vino aici: Am nevoie de tine.” Deși pachetul poate fi primit (auzit) de multă lume, numai Watson va răspunde. Ceilalți pur și simplu îl ignoră. Un alt exemplu ar fi un aeroport unde se anunță că toți pasagerii zborului 644 sunt rugați să se prezinte la poarta 12.

Sistemele cu difuzare permit în general și adresarea unui pachet către *toate* destinațiile, prin folosirea unui cod special în câmpul de adresă. Un pachet transmis cu acest cod este primit și prelucrat de toate mașinile din rețea. Acest mod de operare se numește **difuzare**. Unele sisteme cu difuzare suportă de asemenea transmisia la un subset de mașini, operație cunoscută sub numele de **trimitere multiplă**. Una din schemele posibile este să se rezerve un bit pentru a indica trimiterea multiplă. Restul de  $n - 1$  biți de adresă pot forma un număr de grup. O mașină se poate „abona” la orice grup sau la toate grupurile. Un pachet trimis unui anumit grup va ajunge la toate mașinile abonate la grupul respectiv.

Prin contrast, **rețelele punct-la-punct** dispun de numeroase conexiuni între perechi de mașini individuale. Pentru a ajunge de la sursă la destinație pe o rețea de acest tip, un pachet s-ar putea să fie nevoie să treacă prin una sau mai multe mașini intermediare. Deseori sunt posibile trasee multiple, de diferite lungimi, și de aceea descoperirea drumurilor celor mai potrivite este foarte importantă. Ca o regulă generală (deși există numeroase excepții), rețelele mai mici, localizate geografic, tend să utilizeze difuzarea, în timp ce rețelele mai mari sunt de obicei punct-la-punct. Transmisiile punct la punct cu un sigur transmițător și un singur receptor sunt numite uneori și **unicasting**.

Distanța între procesoare	Procesoare localizate în același (aceeași)...	Exemplu
1 m	Metru pătrat	Rețea personală
10 m	Cameră	Rețea locală
100 m	Clădire	Rețea metropolitană
1 km	Campus	Rețea larg răspândită geografic
10 km	Oraș	
100 km	Tară	
1000 km	Continent	
10.000 km	Planetă	Internet-ul

**Fig. 1-6.** Clasificarea procesoarelor interconectate în funcție de dimensiune.

Un criteriu alternativ pentru clasificarea rețelelor este mărimea lor. În fig. 1-6 este prezentată o clasificare a sistemelor cu procesoare multiple după mărimea lor fizică. Prima categorie o reprezintă rețelele personale (personal area networks), rețele gândite pentru o singură persoană. De exemplu,

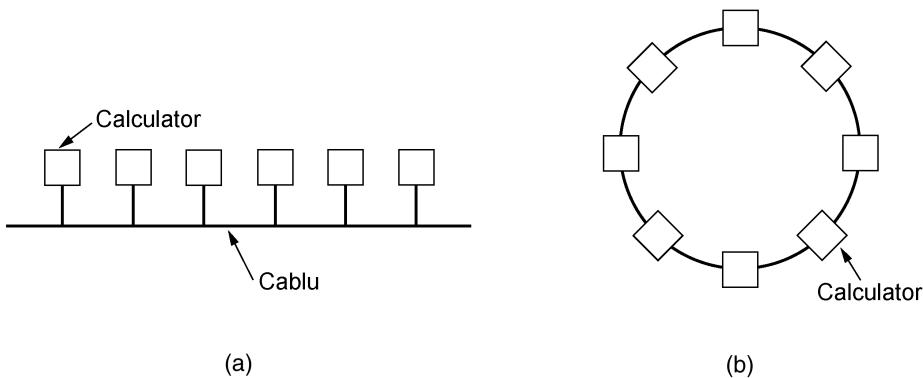
o rețea fără fir care conectează calculatorul cu perifericele sale (tastatură, imprimantă, mouse) este o rețea personală. De asemenea, un PDA care controlează aparatul auditiv al utilizatorului sau regulatorul lui de ritm cardiac se încadrează în aceeași categorie. Mai departe de aceste rețele personale sunt rețele cu domenii mai mari. Acestea pot fi împărțite în rețele locale, rețele metropolitane și rețele larg răspândite geografic. În sfârșit, prin conectarea a două sau mai multe rețele rezultă o inter-rețea. Internet-ul este un exemplu bine cunoscut de inter-rețea. Distanța este un criteriu de clasificare important, pentru că, la scări diferite, sunt folosite tehnici diferite. În această carte ne vom ocupa de rețele din toate aceste categorii. Prezentăm mai jos o scurtă introducere în subiectul echipamentelor de rețea.

### 1.2.1 Rețele locale

**Rețelele locale (Local Area Networks)**, denumite în general LAN-uri, sunt rețele private localizate într-o singură clădire sau într-un campus de cel mult câțiva kilometri. Ele sunt frecvent utilizate pentru a conecta calculatoarele personale și stațiile de lucru din birourile companiilor și fabricilor, în scopul de a partaja resurse (imprimante, de exemplu) și de a schimba informații. LAN-urile se disting de alte tipuri de rețele prin trei caracteristici: (1) mărime, (2) tehnologie de transmisie și (3) topologie.

LAN-urile au dimensiuni restrânse, ceea ce înseamnă că timpul de transmisie în cazul cel mai deficitar este limitat și cunoscut dinainte. Cunoscând această limită, este posibil să utilizăm anumite tehnici de proiectare care altfel nu ar fi fost posibile. Totodată, se simplifică administrarea rețelei.

LAN-urile utilizează frecvent o tehnologie de transmisie care constă dintr-un singur cablu la care sunt atașate toate mașinile, aşa cum erau odată cablurile telefonice comune în zonele rurale. LAN-urile tradiționale funcționează la viteze cuprinse între 10 și 100 Mbps, au întârzieri mici (microsecunde sau nanosecunde) și produc erori foarte puține. LAN-urile mai noi pot opera la viteze mai mari, până la 10 Gbps. În această carte vom păstra tradiția și vom măsura vitezele de transmisie pe linii în megabit/sec (1 Mbps reprezintă 1.000.000 biți), și gigabit/sec (1 Gbps reprezintă 1.000.000.000 biți).



**Fig. 1-7.** Două rețele cu difuzare. (a) Magistrală. (b) Inel.

Pentru LAN-urile cu difuzare sunt posibile diverse topologii. Fig. 1-7 prezintă două dintre ele. Într-o rețea cu magistrală (cu cablu liniar), în fiecare moment cel mult una dintre mașini este master și are dreptul să transmită. Restul mașinilor nu pot transmite. Când două sau mai multe mașini vor

să transmită simultan, este necesar un mecanism de arbitrage. Mecanismul de arbitrage poate fi centralizat sau distribuit. De exemplu, IEEE 802.3, popular numită **Ethernet™**, este o rețea cu difuzare bazată pe magistrală cu control descentralizat, lucrând la viteze între 10 Mbps și 10 Gbps. Calculatoarele dintr-un Ethernet pot transmite oricând doresc; dacă două sau mai multe pachete se ciocnesc, fiecare calculator așteaptă o perioadă de timp aleatorie și apoi încearcă din nou.

Un al doilea tip de rețea cu difuzare este rețeaua în inel. Într-un inel fiecare bit se propagă independent de ceilalți, fără să aștepte restul pachetului să devină înapoi. În mod tipic, fiecare bit navighează pe circumferința întregului inel într-un interval de timp în care se transmit doar câțiva biți, de multe ori înainte chiar ca întregul pachet să fi fost transmis. Ca în orice alt sistem cu difuzare, este nevoie de o regulă pentru a arbitra accesele simultane la inel. Pentru aceasta se utilizează diferite metode, care vor fi discutate în carte mai târziu. IEEE 802.5 (inelul cu jeton de la IBM) este un LAN popular de tip inel, care operează la 4 și la 16 Mbps. Un alt exemplu de rețea de tip inel este **FDDI (Fiber Distributed Data Interface)**, rom: Interfață de date distribuite pe fibră optică).

Rețelele cu difuzare pot fi în continuare împărțite în statice și dinamice, în funcție de modul de alocare al canalului. O metodă tipică de alocare statică ar fi să diviziăm timpul în intervale discrete și să rulăm un algoritm round-robin, lăsând fiecare mașină să emită numai atunci când îi vine rândul. Alocarea statică irosește capacitatea canalului atunci când o mașină nu are nimic de transmis în ceea ce urmărește de timp care i-a fost alocată, astfel că majoritatea sistemelor încearcă să aloce canalul dinamic (la cerere).

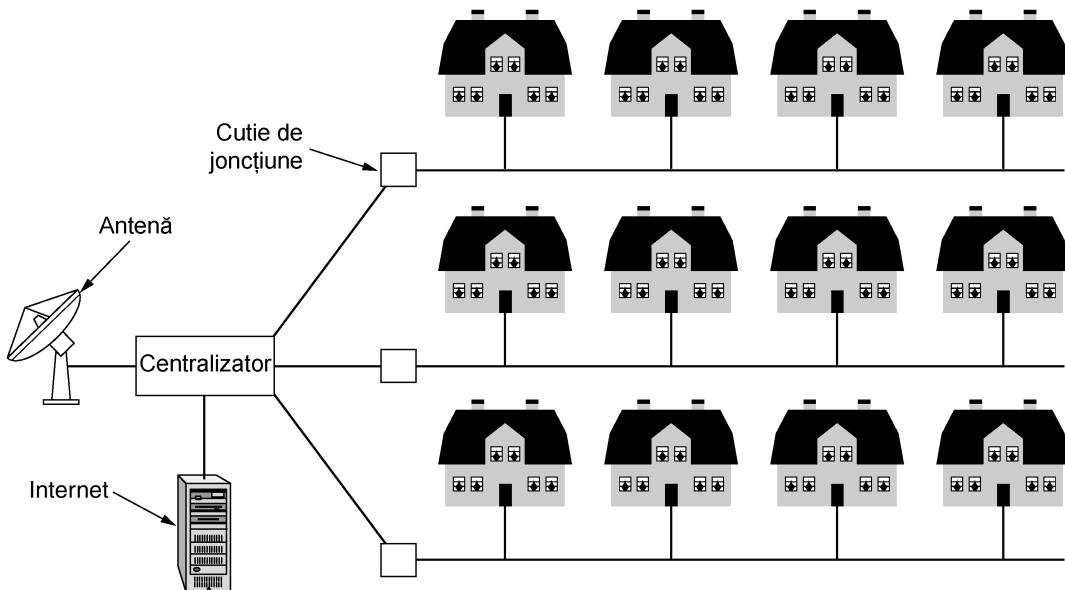
Metodele de alocare dinamică pentru un canal comun sunt fie centralizate, fie descentralizate. În cazul metodei centralizate de alocare a canalului există o singură entitate, de pildă o unitate de arbitrage a magistralei, care determină cine urmează la rând. Poate face acest lucru acceptând cereri și luând o decizie conform unui algoritm intern. În cazul metodei descentralizate de alocare a canalului nu există o entitate centrală; fiecare mașină trebuie să hotărască pentru ea însăși dacă să transmită sau nu. S-ar putea crede că în acest fel se ajunge totdeauna la haos, dar lucrurile nu stau așa. Vom studia mai târziu numerosi algoritmi proiectați să refacă ordinea dintr-un potențial haos.

### 1.2.2 Rețele metropolitane

O rețea metropolitană (**Metropolitan Area Network**), sau **MAN** (plural: **MAN-uri**) deservește un oraș. Cel mai bun exemplu de MAN este rețeaua de televiziune prin cablu disponibilă în cele mai multe orașe. Acest sistem s-a dezvoltat de la primele antene colective folosite în zone în care semnalul recepționat prin aer era foarte slab. În aceste sisteme timpuri, o antenă foarte mare era amplasată pe vârful celui mai apropiat deal și semnalul captat era retransmis către casele abonaților.

La început, acestea erau sisteme proiectate local, ad-hoc. Apoi companiile au început să se implice în această afacere, obținând contracte de la municipalitățile orașelor pentru a cabla chiar și întreg orașul. Următorul pas a fost programarea televiziunii și chiar canale de televiziune produse numai pentru furnizarea prin cablu. De cele mai multe ori aceste canale sunt foarte specializate, pe domenii precum stirile, sporturile, gastronomia, grădinăritul, și altele. Dar încă de la începuturi și până în ultima perioadă a anilor 1990, aceste rețele erau exclusiv dedicate receptiei de televiziune.

Din momentul în care Internet-ul a început să atragă audiенță de masă, operatorii de rețele de cablu TV au realizat că, dacă vor face anumite schimbări în sistem, ar putea să ofere servicii bidirectionale în Internet în părțile nefolosite ale spectrului. La acel moment, sistemul de cablu TV a început să se transforme dintr-o soluție de a distribui semnalul TV în oraș într-o rețea metropolitană. La o primă aproximare, o MAN poate să arate oarecum similar cu sistemul prezentat în fig. 1-8.



**Fig. 1-8.** O rețea metropolitană care se bazează pe cablu TV.

În această figură se văd atât semnalele de televiziune cât și Internet-ul trimise într-un centralizator (head end) pentru a fi apoi redistribuite în casele oamenilor. Vom reveni la acest subiect în detaliu în cap. 2.

Televiziunea prin cablu nu este singurul MAN. Ultimele dezvoltări în domeniul accesului la Internet fără fir, a dus la dezvoltarea unei noi rețele metropolitane care a fost standardizată cu numele de IEEE 802.16. Vom studia acest domeniu în cap. 2.

### 1.2.3 Rețele larg răspândite geografic

O rețea larg răspândită geografic (**Wide Area Network**), sau WAN, acoperă o arie geografică întinsă - deseori o țară sau un continent întreg. Rețeaua conține o colecție de mașini utilizate pentru a executa programele utilizatorilor (adică aplicații). În concordanță cu termenul uzual, vom numi aceste mașini **gazde**. Gazdele sunt conectate printr-o **subrețea de comunicație** sau, pe scurt, **subrețea**. Gazdele aparțin clientilor (de exemplu calculatoarele personale ale oamenilor), deși subrețeaua de comunicație aparține și esteexploatață, de cele mai multe ori, de o companie de telefonie sau de un furnizor de servicii Internet (ISP). Sarcina subrețelei este să transporte mesajele de la gazdă la gazdă, exact așa cum sistemul telefonic transmite cuvintele de la vorbitor la ascultător. Prin separarea aspectelor de pură comunicație ale rețelei (subrețelei) de aspectele referitoare la aplicații (gazde), proiectarea întregii rețele se simplifică mult.

În majoritatea rețelelor larg răspândite geografic, subrețeaua este formată din două componente distincte: liniile de transmisie și elementele de comutare. **Liniile de transmisie** transportă bițiî între mașini. Ele pot fi alcătuite din fire de cupru, fibră optică sau chiar legături radio. **Elementele de comutare** sunt calculatoare specializate, folosite pentru a conecta două sau mai multe linii de transmisie. Când sosesc date pe o anumită linie, elementul de comutare trebuie să aleagă o nouă linie pentru a retrasmite datele mai departe. Din păcate, nu există nici o terminologie standard pentru de-

numirea acestor calculatoare. Aceste elemente de comutare au primit diverse nume în trecut; numele de **ruter** (**router**<sup>1</sup>) este acum cel mai folosit.

În acest model, prezentat în fig. 1-9, fiecare gazdă este de cele mai multe ori conectată la un LAN în care există un ruter, desă în anumite cazuri o gazdă poate fi legată direct cu un ruter. Colecția de linii de comunicație și de rutere (dar nu și gazdele) formează subrețea.

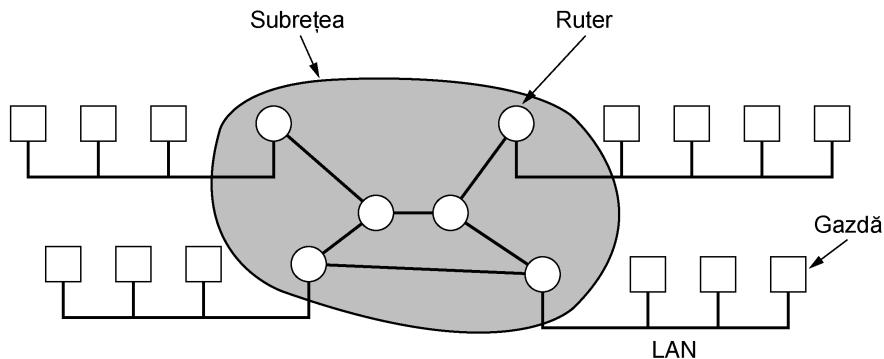


Fig. 1-9. Relația dintre gazde și subrețea.

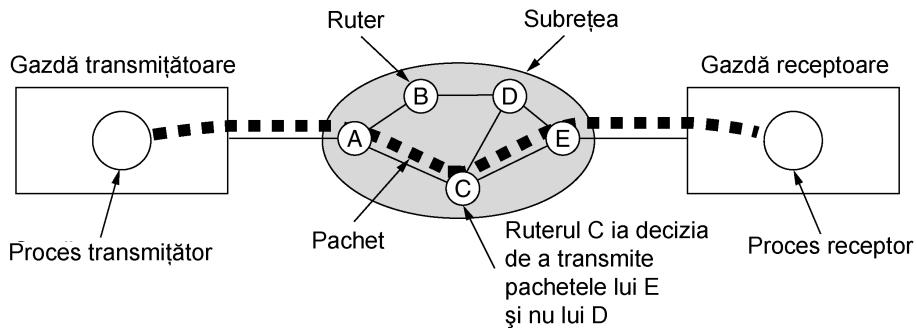
Merită să facem un scurt comentariu în jurul termenului de „subrețea”. Inițial, singura sa accepție se referea la colecția ruterelor și liniilor de comunicație care mutau pachetele de la gazda sursă la gazda destinație. Totuși, câțiva ani mai târziu, cuvântul a mai căpătat un al doilea înțeles, în conjuncție cu adresarea rețelelor (pe care o vom discuta în Cap. 5). Din nefericire, nu există o alternativă larg acceptată pentru înțelesul său inițial, drept care noi vom folosi acest termen, cu unele rezerve, în ambele sensuri. Din context, va fi totdeauna clar care din ele este subînțeles.

În cazul celor mai multe WAN-uri, rețeaua conține numeroase linii de transmisie, fiecare din ele legând o pereche de rutere. Dacă două rutere nu împart un același cablu, dar doresc să comunice, atunci ele trebuie să facă acest lucru indirect, prin intermediul altor rutere. Când un pachet este transmis de la un ruter la altul prin intermediul unuia sau mai multor rutere, pachetul este primit în întregime de fiecare ruter intermediar, este reținut acolo până când linia de ieșire cerută devine liberă și apoi este retransmis. O subrețea care funcționează pe acest principiu se numește subrețea **memorează-și-retransmite** sau subrețea **cu comutare de pachete**. Aproape toate rețelele larg răspândite geografic (exceptie făcând cele care utilizează sateliți) au subrețele memorează-și-retransmite. Când pachetele sunt mici și au aceeași mărime, ele sunt adesea numite **celule**.

Principiul de funcționare a unui WAN cu comutare de pachete este atât de important încât merită să mai adăugăm câteva cuvinte despre el. În general, atunci când un proces al unei gazde are un mesaj de transmis către un proces de pe o altă gazdă, gazda care transmite va sparge mesajul în pachete, fiecare dintre ele reținându-și numărul de ordine din secvență. Aceste pachete sunt apoi transmise în rețea unul către unul într-o succesiune rapidă. Pachetele sunt transportate individual prin rețea și depozitate la gazda receptoare, unde sunt reasamblate în mesajul inițial și furnizate pro-

<sup>1</sup> Din păcate, unii îl pronunță ca englezescul „rooter” și alții preferă să îl asocieze ca pronunție cu „doubter”. Determinarea pronunției corecte în limba engleză va fi lăsată ca exercițiu cititorului. (răspunsul pe care îl veți afla poate depinde de zona în care întrebăți).

cesului receptor. Un flux de pachete rezultat din descompunerea unui mesaj inițial oarecare este prezentat în fig. 1-10.



**Fig. 1-10.** Un flux de pachete de la transmițător la receptor.

În această figură, toate pachetele parcurg ruta A-C-E, în loc de A-B-D-E sau A-C-D-E. În unele rețele, toate pachetele aparținând unui mesaj dat trebuie să urmeze aceeași rută; în altele, fiecare pachet este dirijat separat. Desigur, dacă A-C-E este cea mai bună rută, toate pachetele pot fi transmise pe acolo, chiar dacă fiecare dintre ele este dirijat individual.

Deciziile de dirijare se iau la nivelul local al ruterului. Când un pachet ajunge la ruterul A, este de datoria lui A să decidă dacă acest pachet trebuie trimis pe linia către B sau pe linia către C. Modul în care ruterul A ia această decizie este denumit **algoritm de rutare**. Există mulți astfel de algoritmi. Pe unii dintre ei îi vom studia în detaliu în cap. 5.

Nu toate WAN-urile sunt cu comutare de pachete. O a doua posibilitate pentru un WAN este un sistem de sateliți. Fiecare ruter are o antenă prin care poate trimite și poate primi. Toate ruterele pot asculta ieșirea *de la* satelit, iar în anumite cazuri pot să asculte chiar și transmisia celorlalte rutere *către* satelit. Uneori, ruterele sunt conectate la o rețea punct-la-punct și numai unele dintre ele pot avea antene de satelit. Rețelele satelit sunt în mod implicit rețele cu difuzare și sunt foarte utile când proprietatea de difuzare este importantă.

#### 1.2.4 Rețele fără fir

Comunicațiile digitale fără fir nu reprezintă o idee nouă. Încă din 1901, fizicianul italian Guglielmo Marconi a realizat legătura între un vapor și un punct de pe coastă folosind telegraful fără fir și codul Morse (punctele și liniile sunt, în definitiv, binare). Sistemele radio moderne au performanțe mai bune, dar ideea fundamentală a rămas aceeași.

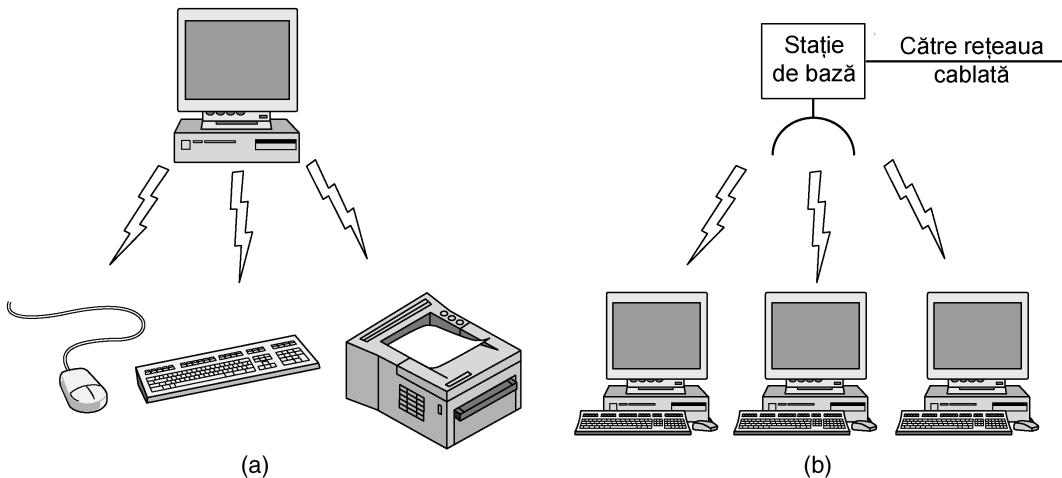
La o primă aproximare, rețelele fără fir pot fi împărțite în 3 mari categorii:

1. Interconectarea componentelor unui sistem
2. LAN-uri fără fir
3. WAN-uri fără fir

Interconectarea componentelor se referă numai la interconectarea componentelor unui calculator folosind unde radio cu rază mică de acțiune. Aproape orice calculator are un monitor, o tastatură, un mouse și o imprimantă legate la unitatea centrală prin cabluri. Multă din noii utilizatori au probleme cu conectarea tuturor cablurilor exact în mufele mici în care trebuie (chiar dacă acestea

sunt de cele mai multe ori codificate pe culori), aşa că producătorii de calculatoare oferă opţiunea de a trimite un tehnician pentru instalare. În consecinţă, câteva companii s-au adunat pentru a proiecta o retea fără fir cu rază mică de acţiune denumită Bluetooth pentru a conecta toate componentele fără cabluri. De asemenea, Bluetooth permite camerelor digitale, căştilor, scannerelor și altor dispozitive să se conecteze la calculator prin simpla poziţionare în zona acoperită de retea. Fără cabluri, fără instalarea de drivere, doar poziţionare, pornire și ... merge. Pentru mulți oameni această ușurință în utilizare este un mare avantaj.

În cea mai simplă formă, retelele de interconectare în sistem folosesc paradigma stăpân-sclav (master-slave) din fig. 1-11(a). Unitatea centrală a sistemului este în mod normal stăpânul, care discută cu perifericele ca sclavi. Stăpânul le comunică sclavilor ce adrese să folosească, când pot să difuzeze mesaje, cât timp pot să transmită, ce frecvențe pot să folosească, și aşa mai departe. Vom discuta despre Bluetooth în detaliu în cap. 4.



**Fig. 1-11.** (a) Configurație Bluetooth. (b) Rețea locală fără fir.

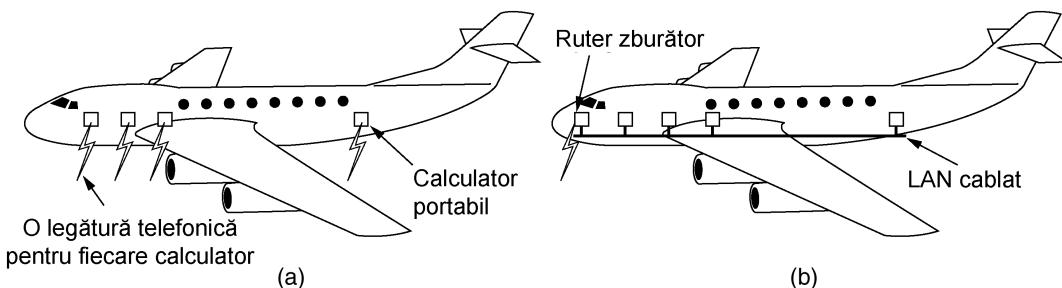
Următoarea treaptă în retelele fără fir o reprezintă retelele locale fără fir. Acestea sunt sisteme în care fiecare calculator are un modem radio și o antenă cu care poate comunica cu alte calculatoare. De multe ori există o antenă în tavan cu care mașinile vorbesc, aşa cum se poate vedea în fig. 1-11(b). Oricum, dacă sistemele sunt destul de apropiate, ele pot comunica direct unul cu altul într-o configurație punct-la-punct. Rețelele locale fără fir devin din ce în ce mai utilizate în birouri mai mici și acasă, unde instalarea unei rețele Ethernet este considerată prea complicată, precum și în clădiri de birouri mai vechi, în cantinele companiilor, în camerele de conferințe, și în alte asemenea locuri. Există un standard pentru rețelele locale fără fir, numit **IEEE 802.11**, pe care îl implementează majoritatea sistemelor și care devine din ce în ce mai răspândit. Îl vom discuta în cap. 4.

Cea de-a treia categorie de rețele fără fir este folosită în sistemele răspândite pe arii geografice largi (Wide Area Networks). Rețeaua radio utilizată de telefonia mobilă este un exemplu de sistem fără fir cu lărgime de bandă redusă. Acest sistem este deja la generația a treia. Prima generație era analogică și numai pentru voce. A doua generație era digitală, dar numai pentru voce. Cea de-a treia generație este digitală și este utilizată atât pentru voce cât și pentru date. Într-un anume sens, rețelele celulare fără fir sunt foarte asemănătoare cu rețelele locale fără fir, cu excepția faptului că distan-

țele implicate sunt mult mai mari, iar ratele de transfer sunt mult mai mici. Rețelele locale fără fir pot opera la rate de până la 50 Mbps pe distanțe de zeci de metri. Sistemele celulare pot opera sub 1 Mbps, dar distanțele dintre stația de bază și calculator sau telefon este măsurată mai degrabă în kilometri decât în metri. Vom avea multe de spus despre aceste rețele în cap. 2.

În plus față de aceste rețele de viteză redusă, sunt dezvoltate și WAN-uri cu lărgime de bandă mare. Important este în primul rând accesul la Internet de acasă sau din cadrul companiei prin conexiune rapidă fără fir, eliminând necesitatea folosirii sistemului de telefonie. Acest serviciu este de multe ori denumit serviciu local de distribuire multipunct. Îl vom studia mai târziu în carte. A fost dezvoltat și un standard al său, numit IEEE 802.16. Îl vom examina în cap. 4.

Aproape toate rețelele ajung mai devreme sau mai târziu să fie parte dintr-o rețea cablată pentru a oferi acces la fișiere, baze de date sau Internet. Sunt multe variante prin care aceste conexiuni pot fi realizate, în funcție de circumstanțe. De exemplu, în fig. 1-12(a) este prezentat un avion în care un număr de persoane folosesc modemuri și telefoane încorporate în spătarul scaunului (eng.: seat-back telephone) pentru a suna la birou. Fiecare apel este independent de toate celelalte. O opțiune mult mai eficientă este LAN-ul zburător (flying LAN) din fig. 1-12(b). Aici, fiecare scaun este echipat cu un conector Ethernet în care pasagerii pot să își conecteze calculatoarele. Un singur ruter al avionului menține o legătură radio cu un ruter de la sol, schimbând acest ruter pe măsură ce își parcurge traseul. Această configurație este o rețea locală tradițională, doar că pentru a se conecta cu restul lumii folosește o legătură radio în loc de o linie cablată.



**Fig. 1-12.** (a) Calculatoare mobile individuale. (b) Un LAN zburător.

Multă lume crede că tehnologiile fără fir reprezintă valul viitorului (de ex. Bi et al., 2001; Leeper, 2001; Varschez și Vetter, 2000), dar există cel puțin o părere contrară cunoscută. Bob Metcalfe, inventatorul Ethernet-ului, a scris următoarele: „Calculatoarele mobile fără fir sunt ca băile mobile fără țevi - niște olițe de noapte portabile. Ele vor fi ceva comun în vehicule, pe șantiere și la concerte rock. Sfatul meu este să vă racordați cabluri în casă și să rămâneți acolo” (Metcalfe, 1995). Istoria ar putea să rețină această afirmație în aceeași categorie cu a lui T.J. Watson, președintele IBM, care explica în 1945 de ce IBM nu se intră în afacerea calculatoarelor: „Patru sau cinci calculatoare ar trebui să fie suficiente pentru întreaga lume până în anul 2000”.

### 1.2.5 Rețelele casnice (Home networks)

Rețelele în mediul casnic sunt la orizont. Ideea fundamentală este că în viitor, cele mai multe locuințe vor fi pregătite pentru instalarea de rețele. Fiecare dispozitiv din casă va fi capabil să comunique cu orice alt dispozitiv și toate vor fi accesibile prin Internet. Aceasta este unul dintre acele concepte

revoluționare pe care nu l-a cerut nimeni (cum sunt telecomenziile TV sau telefoanele mobile), dar de îndată ce au fost implementate nimeni nu și-a mai putut închipui cum au trăit fără ele.

Multe dispozitive sunt capabile să fie legate în rețea. Unele dintre categoriile cele mai simple, însoțite de exemple sunt cele care urmează:

1. Calculatoarele (PC-uri staționare, PC-uri portabile, PDA-uri, periferice partajate)
2. Dispozitivele de divertisment (TV, DVD, VCR, camera video, combina muzicală)
3. Dispozitive pentru telecomunicații (telefonul, telefonul mobil, fax-ul, sistemul de comunicare interioară)
4. Aparatura casnică (cuporul cu microunde, frigiderul, ceasul, cuporul, aparatul de aer condiționat, luminile)
5. Contoarele și alarmele (contoarele pentru utilități, alarmele de fum sau hoți, termostate, sistemele de supraveghere a copilului)

Rețelele casnice sunt deja implementate într-o oarecare măsură. Multe case au deja un dispozitiv pentru conectarea mai multor calculatoare la Internet printr-o conexiune rapidă. Divertismentul prin rețea nu este chiar la îndemână, dar pentru că din ce în ce mai multă muzică și mai multe filme sunt disponibile pentru descărcare din Internet, va exista o cerere de conectare a combinelor muzicale și a televizoarelor în rețea. De asemenea, oamenii vor dori să împartă propriile clipuri video cu prietenii și familia, astfel că această conexiune va trebui să fie bidirectională. Angrenajul telecomunicațiilor este deja conectat la lumea exterioră, dar în curând aceste vor fi digitale și transmise prin Internet. În medie, o casă are cam o duzină de ceasuri (de exemplu, cele de la aparatele electrocasnice), care toate trebuie potrivite cel puțin de două ori pe an, când se trece la ora de vară și apoi la ora de iarnă. Dacă toate aceste ceasuri ar fi conectate la Internet, această potrivire s-ar face automat. În fine, monitorizarea de la distanță a casei și a interiorului său este un posibil domeniu de succes. Probabil că mulți dintre părinți ar fi gata să cheltuiască niște bani pentru a-și supravegheza copiii adormiți, prin intermediul PDA-urilor, în timp ce iau masa în oraș, chiar și dacă au angajat un adolescent pentru a avea grija de ei. În timp ce unii își pot imagina o rețea separată pentru fiecare zonă de aplicații, integrarea tuturor într-o singură rețea mai mare este probabil o idee mult mai bună.

Rețelele casnice au câteva proprietăți fundamentale diferite de alte tipuri de rețele. Mai întâi, atât rețeaua cât și dispozitivele trebuie să fie ușor de instalat. Autorul a instalat multe componente hardware și software pe diverse calculatoare de-a lungul anilor, cu diverse rezultate. O serie de telefoane la biroul de suport tehnic al producătorului au rezultat în răspunsuri de tipul (1) Citiți manualul, (2) Reporniți calculatorul, (3) Scoateți toate componentele hardware și software cu excepția celor furnizate de noi și încercați din nou, (4) Descărcați cea mai nouă versiune a programului de configurare de pe situl nostru Web și dacă toate acestea eșuează, (5) Reformați discul și apoi reinstalați Windows de pe CD-ROM. A spune unui cumpărător de frigider care poate fi conectat la Internet să descarce și să instaleze o nouă versiune a sistemului de operare pentru frigiderul său nu este de natură să facă prea mulți clienți fericiți. Utilizatorii de calculatoare sunt obișnuiați cu instalarea de produse care nu merg din prima; cumpărătorii de mașini, televizoare sau frigidere sunt mai puțin toleranți. Ei se așteaptă ca produsele să răspundă corect la 100% din comenzi.

În al doilea rând, rețelele și dispozitivele trebuie să fie protejate împotriva utilizării neglijente. Primele aparate de aer condiționat aveau un buton cu patru poziții: OPRIT, SCĂZUT, MEDIU, RAPID. Acum au manuale de 30 de pagini. De îndată ce vor fi conectate în rețea, așteptați-vă ca numai capitolul de securizare să aibă 30 de pagini. Ceea ce va depăși capacitatea de înțelegere a majorității utilizatorilor.

În al treilea rând, prețul scăzut este esențial pentru succes. Cumpărătorii nu vor plăti 50 de dolari în plus pentru un termostat numai pentru că unii oameni consideră important să-și supravegheze de la birou temperatura din casă. Pentru numai 5 dolari în plus, s-ar putea să se vândă.

În al patrulea rând, programul principal este foarte probabil să implice facilități multimedia, aşa că rețeaua are nevoie de capacitate suficientă. Nu există piață pentru televizoare conectate la Internet care să prezinte filme de groază în rezoluție de  $320 \times 240$  pixeli și la 10 cadre/s. Ethernet-ul rapid (fast Ethernet), mediul de lucru în majoritatea birourilor, nu este destul de bun pentru facilitățile multimedia. În consecință, rețelele casnice vor avea nevoie de performanțe mai bune decât cele ale rețelelor care există acum în companii și de prețuri mai mici pentru a deveni articole care se vând în masă.

În cel de-al cincilea rând, trebuie să fie posibil să se pornească cu unul sau două dispozitive și extinderea să se poată face gradat. Aceasta înseamnă fără schimbări revoluționare. A spune consumatorilor să își cumpere periferice cu interfețe IEEE 1394 (FireWire) și apoi, după câțiva ani, să retrageți spunând că USB 2.0 este interfața lunii va face consumatorii să devină capricioși. Interfața de rețea va trebui să rămână stabilă pentru mulți ani; cablajul (dacă există) va trebui să rămână același pentru decade întregi.

În cel de-al șaselea rând, securitatea și siguranța vor fi foarte importante. Pierderea câtorva fișiere datorită unui virus de poștă electronică e una, dar dacă un hoț îți dezarmează sistemul de securitate al locuinței de la PDA-ul său și apoi intră în casă este cu totul altă situație.

O întrebare interesantă este dacă rețelele casnice trebuie să fie cablate sau fără fir. Majoritatea locuințelor au deja șase rețele instalate: electrică, telefonică, televiziune prin cablu, apă, gaz și canalizare. Adăugarea unei a șaptea rețele în timpul construcției nu este dificilă, dar reamenajarea caselor deja construite este costisitoare. Costul este un motiv de a alege rețelele fără fir, dar securitatea este un motiv pentru cele cablate. Problema cu rețelele fără fir este aceea că undele radio pe care le folosesc trec foarte ușor prin garduri. Nimeni nu este foarte bucuros dacă vecinii îți pot intercepta conexiunea la Internet și îți pot citi mesajele de poștă electronică în timp ce acestea sunt trimise la proprietar. În cap. 8 vom vedea cum se poate folosi criptarea pentru a oferi securitate, dar în contextul unei rețele casnice, securitatea trebuie să fie și ea protejată împotriva utilizării neglijente, chiar și în cazul utilizatorilor fără experiență. Aceasta este mai ușor de spus decât de făcut, chiar și pentru utilizatori foarte pricepuți. Pe scurt, rețelele casnice oferă multe facilități și provocări. Multe dintre ele sunt legate de necesitatea de a fi ușor de administrat, sigure și securizate, mai ales în mâinile utilizatorilor care nu sunt implicați în domeniul tehnic, concomitent cu necesitatea de a obține performanțe ridicate la prețuri scăzute.

### 1.2.6 Inter-rețelele

În lume există multe rețele, cu echipamente și programe diverse. Persoanele conectate la o anumită rețea doresc adesea să comunice cu persoane racordate la alta. Această cerință impune conectarea unor rețele diferite, de multe ori incompatibile, ceea ce uneori se realizează utilizând mașini numite **porți (gateways)**. Acestea realizează conectarea și asigură conversiile necesare, atât în termeni de hardware cât și de software. O colecție de rețele interconectate este numită **inter-rețea** sau **internet**. Acești termeni vor fi folosiți în sens generic, spre deosebire de Internet-ul mondial (care este un internet special), al cărui nume va fi scris mereu cu majusculă.

O formă comună de inter-rețea este o colecție de LAN-uri conectate printr-un WAN. De fapt, dacă am înlocui eticheta „subrețea” din fig. 1-9 prin „WAN”, în figură nu ar mai trebui să schimbe nimic altceva. În acest caz, singura diferență tehnică reală între o subrețea și un WAN se referă la

prezența gazdelor. Dacă sistemul din interiorul zonei gri conține numai rutere, atunci este o subrețea. Dacă el conține atât rutere, cât și gazde cu utilizatori proprii, atunci este un WAN. Diferențele reale sunt legate de proprietate și utilizare.

Deseori se produc confuzii între subrețele, rețele și inter-rețele. Termenul de subrețea este mai potrivit în contextul unei rețele larg răspândite geografic, unde se referă la colecția de rutere și linii de comunicație aflate în proprietatea operatorului de rețea. Ca o analogie, sistemul telefonic constă din centrale telefonice de comutare, care sunt conectate între ele prin linii de mare viteză și sunt legate la locuințe și birouri prin linii de viteză scăzută. Aceste linii și echipamentele, deținute și întreținute de către compania telefonică, formează subrețea sistemului telefonic. Telefoanele propriu-zise (care corespund în această analogie gazdelor) nu sunt o parte a subrețelei. Combinarea dintre o subrețea și gazdele sale formează o rețea. În cazul unui LAN, rețea este formată din cablu și gaze de. Aici nu există cu adevărat o subrețea.

O inter-rețea se formează atunci când se leagă între ele rețele diferite. Din punctul nostru de vedere, legarea unui LAN și a unui WAN sau legarea a două LAN-uri formează o inter-rețea, dar nu există un consens asupra terminologiei din acest domeniu. O regulă simplă este aceea că dacă diferențe companii sunt plătite să construiască diverse părți ale unei rețele și fiecare trebuie să își întrețină propria parte, avem o inter-rețea mai degrabă decât o singură rețea. De asemenea, dacă tehnologiile diferă în diverse zone ale rețelei (de exemplu: difuzare și punct-la-punct), probabil că discutăm nu despre una ci despre două rețele.

## 1.3 PROGRAMELE DE REȚEA

În proiectarea primelor rețele de calculatoare, s-a acordat atenție în primul rând echipamentelor, iar programele au fost gândite ulterior. Această strategie nu mai este valabilă. Programele de rețea sunt acum foarte structurate. În secțiunile următoare vom examina unele detalii ale tehnicii de structurare a programelor. Metoda descrisă aici formează punctul de sprijin al întregii cărți și ea va apărea mai departe în repetate rânduri.

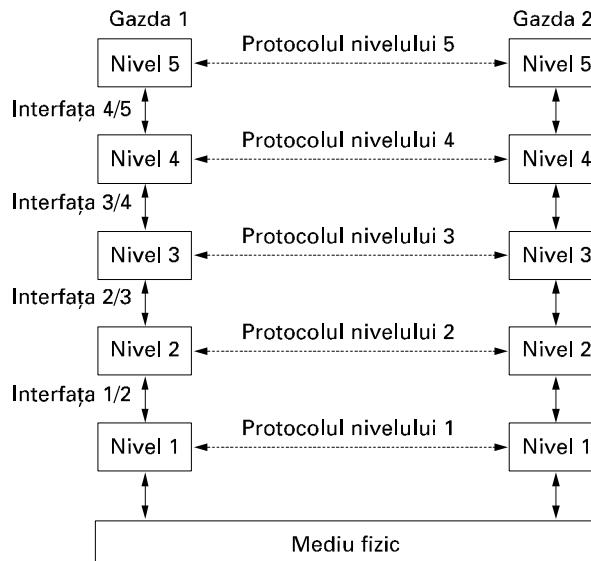
### 1.3.1 Ierarhiile de protocoale

Pentru a reduce din complexitatea proiectării, majoritatea rețelelor sunt organizate sub forma unei serii de **straturi** sau **niveluri**, fiecare din ele construit peste cel de dedesubt. Numărul de niveluri, numele fiecărui nivel, conținutul și funcția sa variază de la rețea la rețea. Oricum, în toate rețelele, scopul fiecărui nivel este să ofere anumite servicii nivelurilor superioare, protejându-le totodată de detaliile privitoare la implementarea efectivă a serviciilor oferte. Într-un anumit sens, fiecare nivel este un fel de mașină virtuală, oferind anumite servicii nivelului de deasupra lui.

Nivelul  $n$  de pe o mașină conversează cu nivelul  $n$  de pe altă mașină. Regulile și convențiile utilizate în conversație sunt cunoscute sub numele de **protocolul** nivelului  $n$ . În principal, un protocol reprezintă o înțelegere între părțile care comunică, asupra modului de realizare a comunicării. Ca o analogie, atunci când o femeie este prezentată unui bărbat, ea poate hotărî să-i intindă bărbatului mâna. La rândul său, bărbatul poate decide fie să-i strângă, fie să-i sărute mâna, decizie care depinde, să spunem, dacă femeia este o avocată americană care a venit la o întâlnire de afaceri sau este o

prințesă europeană prezintă la un bal. Încălcarea protocolului va face comunicarea mai dificilă, dacă nu chiar imposibilă.

În fig. 1-13 este ilustrată o rețea cu cinci niveluri. Entitățile din niveluri corespondente de pe mașini diferite se numesc **egale**. Entitățile egale pot fi procese, dispozitive hardware, sau chiar ființe umane. Cu alte cuvinte, entitățile egale sunt cele care comunică folosind protocolul.



**Fig. 1-13.** Niveluri, protocole și interfețe.

În realitate, nici un fel de date nu sunt transferate direct de pe nivelul  $n$  al unei mașini pe nivelul  $n$  al altrei mașini. Fiecare nivel transferă datele și informațiile de control nivelului imediat inferior, până când se ajunge la nivelul cel mai de jos. Sub nivelul 1 se află **mediul fizic** prin care se produce comunicarea efectivă. În fig. 1-13, comunicarea virtuală este reprezentată prin linii punctate, iar comunicarea fizică prin linii continue. Între două niveluri adiacente există o **interfață**. Interfața definește ce operații și servicii primitive oferă nivelul de jos către nivelul de sus. Când proiectanții de rețea decid câte niveluri să includă într-o rețea și ce are de făcut fiecare din ele, unul din considerențele cele mai importante se referă la definirea de interfețe clare între niveluri.

Aceasta presupune ca, la rândul său, fiecare nivel să execute o colecție specifică de funcții clar definite. Pe lângă minimizarea volumului de informații care trebuie transferate între niveluri, interfețele clare permit totodată o mai simplă înlocuire a implementării unui nivel cu o implementare complet diferită (de exemplu, toate liniile telefonice se înlocuiesc prin canale de satelit). Așa ceva este posibil, pentru că tot ceea ce i se cere noii implementări este să furnizeze nivelului superior exact setul de servicii pe care îl oferea vechea implementare. De altfel, este un fapt obișnuit ca două gazde să folosească implementări diferite.

O mulțime de niveluri și protocole este numită **arhitectură de rețea**. Specificația unei arhitecturi trebuie să conțină destule informații pentru a permite unui proiectant să scrie programele sau să construiască echipamentele necesare fiecarui nivel, astfel încât nivelurile să îndeplinească corect protocolele corespunzătoare. Nici detaliile de implementare și nici specificațiile interfețelor nu fac parte din arhitectură, deoarece acestea sunt ascunse în interiorul mașinilor și nu sunt vizibile din afară. Nu este necesar nici măcar ca interfețele de pe mașinile dintr-o rețea să fie aceleași - cu condi-

ția, însă, ca fiecare mașină să poată utiliza corect toate protocolele. O listă de protocole utilizate de către un anumit sistem, câte un protocol pentru fiecare nivel, se numește **stivă de protocole**. Arhitecturile de retea, stivele de protocole și protocolele propriu-zise constituie principalele subiecte ale acestei cărți.

O analogie poate ajuta la explicarea ideii de comunicare multinivel. Imaginea-vă doi filosofi (procesele egale de la nivelul 3), unul din ei vorbind limbile urdu și engleză, iar celălalt vorbind chineza și franceza. Deoarece filosofii nu cunosc o limbă comună, fiecare din ei angajează câte un translator (procesele egale de la nivelul 2), iar fiecare translator contactează la rândul său o secretară (procesele egale de la nivelul 1). Filosoful 1 dorește să comunice partenerului afecțiunea sa pentru *oryctolagus cuniculus*. Pentru aceasta, el trimite un mesaj (în engleză) prin interfață 2/3 către translatorul său, căruia îi spune următoarele cuvinte: „I like rabbits”<sup>2</sup> (ceea ce este ilustrat în fig. 1-14). Translatorii s-au întăles asupra unei limbi neutre, olandeza, așa că mesajul este convertit în „Ik vind konijnen leuk.” Alegerea limbii reprezintă protocolul nivelului 2 și este la latitudinea proceselor pe-reche de pe acest nivel.

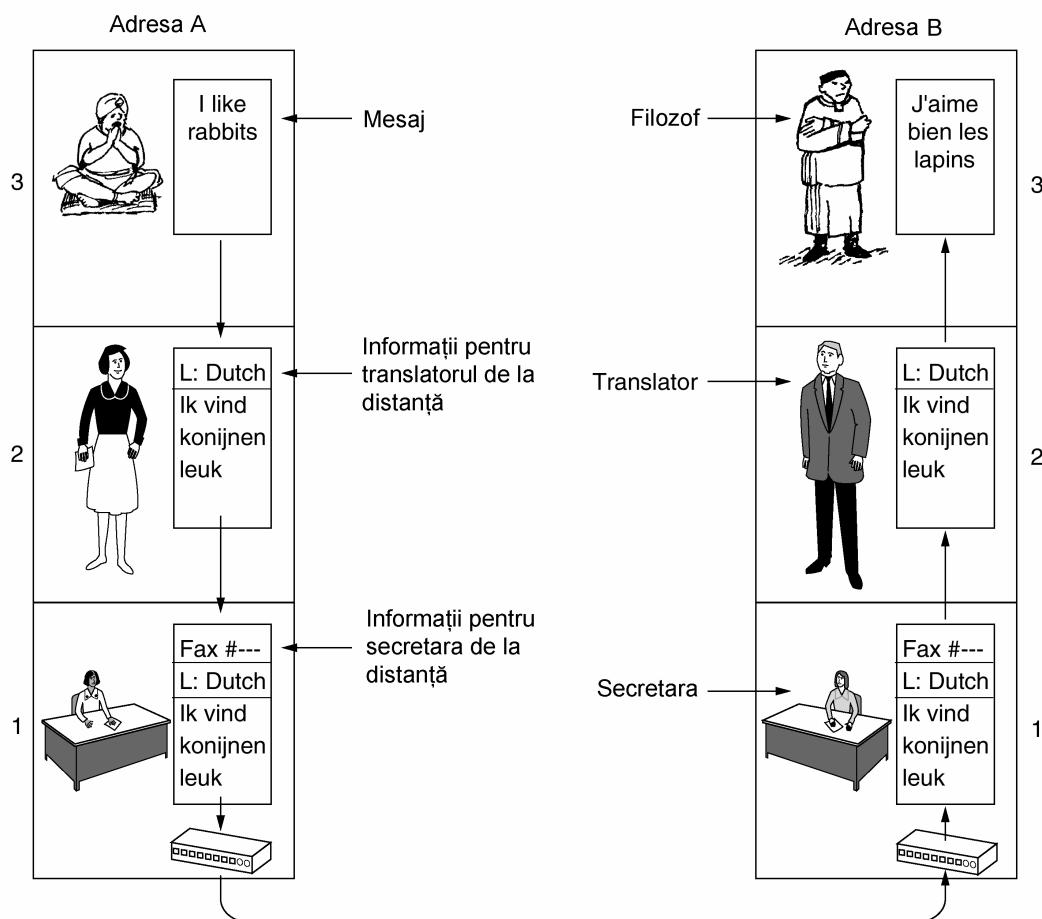
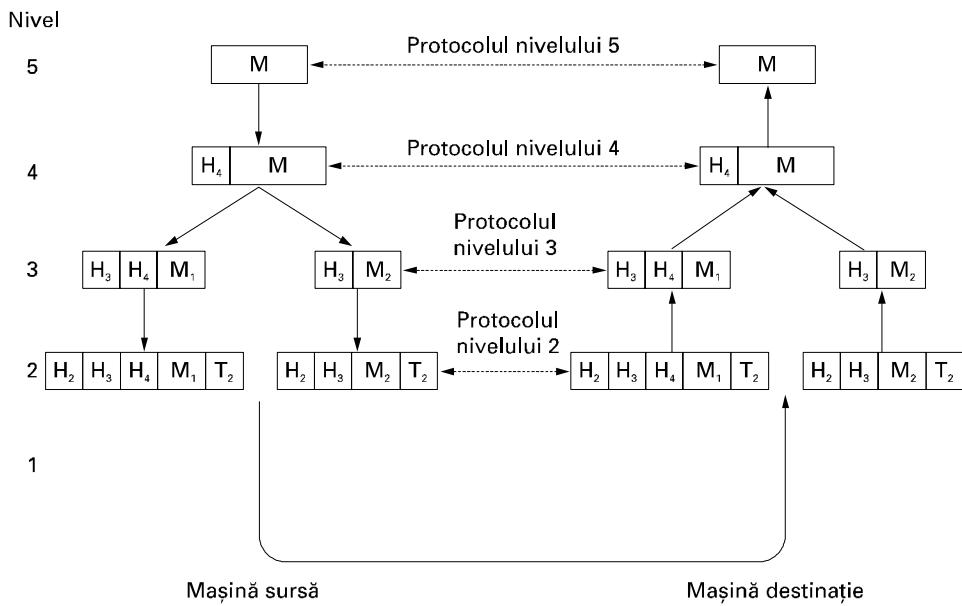


Fig. 1-14. Arhitectura filosof-translator-secretară.

<sup>2</sup>Propoziția înseamnă “Îmi plac iepurii.” (n.t.)



**Fig. 1-15.** Exemplu de flux de informații pentru suportul comunicării virtuale la nivelul 5.

În continuare, translatorul înmânează mesajul secretarei, care îl trimite, de exemplu, prin fax (protocolul nivelului 1). Când mesajul este primit, el este tradus în franceză și trimis prin interfață 2/3 către filosoful 2. Observați că, atâtă timp cât interfețele nu se modifică, fiecare protocol este complet independent de celelalte. Dacă doresc, translatorii pot schimba olandeza cu altă limbă, să spunem finlandeza, cu condiția ca amândoi să se înțeleagă asupra acestui lucru și ca nici unul din ei să nu își modifice interfața cu nivelul 1 sau cu nivelul 3. În mod similar, secretarele pot înlocui faxul cu poșta electronică sau cu telefonul fără a deranja (sau măcar a informa) celealte niveluri. Fiecare proces poate adăuga anumite informații suplimentare destinate numai procesului său pereche. Aceste informații nu sunt transmise în sus, către nivelul superior.

Să considerăm acum un exemplu mai tehnic: cum se realizează comunicarea la ultimul nivel din rețea cu cinci niveluri din fig. 1-15. O aplicație care se execută în nivelul 5 produce un mesaj **M** și îl furnizează nivelului 4 pentru a-l transmită. Nivelul 4 inserează un **antet** în fața mesajului, pentru a identifica respectivul mesaj și pasează rezultatul nivelului 3. Antetul include informații de control, de exemplu numere de ordine care ajută nivelul 4 de pe mașina de destinație să livreze mesajele în ordinea corectă în cazul în care nivelurile inferioare nu păstrează această ordine. Pe unele niveluri, antetele conțin de asemenea câmpuri de control pentru mărime, timp și alte informații.

În numeroase rețele nu există nici o limită cu privire la mărimea mesajelor transmise în protocolul nivelului 4, dar există aproape întotdeauna o limită impusă de protocolul nivelului 3. În consecință, nivelul 3 trebuie să spargă mesajele primite în unități mai mici, pachete, atașând fiecarui pachet un antet specific nivelului 3. În acest exemplu, **M** este descompus în două părți, **M<sub>1</sub>** și **M<sub>2</sub>**.

Nivelul 3 decide ce linie de transmisie să utilizeze și trimite pachetele nivelului 2. Nivelul 2 adăugă nu numai câte un antet pentru fiecare bucătă, ci și o încheiere, după care furnizează unitatea rezultantă nivelului 1 pentru a o transmită fizic. În mașina receptoare mesajul este trimis în sus, din

nivel în nivel, pe parcurs fiind eliminate succesiv toate antetele. Nici un antet corespunzător nivelurilor de sub  $n$  nu este transmis în sus nivelului  $n$ .

Ceea ce este important de înțeles în fig. 1-15 este relația dintre comunicația virtuală și cea efectivă și diferența între protocole și interfețe. De exemplu, procesele egale de la nivelul 4 își imaginează conceptual comunicarea ca realizându-se pe „orizontală”, utilizând protocolul nivelului 4. Deși fiecare din ele are, probabil, o procedură de genul *TrimiteÎnCeaLaltăParte* și o alta *PrimeșteDinCeaLaltăParte*, aceste proceduri nu comunică de fapt cu cealaltă parte, ci cu nivelurile inferioare prin interfața 3/4.

Abstractizarea proceselor pereche este crucială pentru proiectarea întregii rețele. Cu ajutorul ei, această sarcină practic imposibilă poate fi descompusă în probleme de proiectare mai mici, rezolvabile, și anume proiectarea nivelurilor individuale.

Deși Secțiunea 1-3 este intitulată „Programele de rețea”, merită să subliniem că nivelurile inferioare dintr-o ierarhie de protocole sunt implementate frecvent în hardware sau în firmware. Nu e mai puțin adevărat că aici intervin algoritmi complecsi, chiar dacă ei sunt înglobați (partial sau în totalitate) în hardware.

### 1.3.2 Probleme de proiectare a nivelurilor

O parte din problemele cheie care apar la proiectarea rețelelor de calculatoare sunt prezente în mai multe niveluri. Vom menționa pe scurt unele probleme mai importante.

Fiecare nivel are nevoie de un mecanism pentru a identifica emițătorii și receptorii. Dat fiind că o rețea cuprinde în mod normal numeroase calculatoare, iar o parte dintre acestea dețin mai multe procese, este necesară o modalitate prin care un proces de pe o anumită mașină să specifice cu cine dorește să comunice. Ca o consecință a destinațiilor multiple, pentru a specifica una dintre ele, este necesară o formă de adresare.

Un alt set de decizii de proiectare se referă la regulile pentru transferul de date. În unele sisteme datele circulă într-un singur sens; în altele datele pot circula în ambele sensuri. Protocolul trebuie, de asemenea, să determine cător canale logice le corespunde conexiunea și care sunt prioritățile acestora. Multe rețele dispun de cel puțin două canale logice pe conexiune, unul pentru date normale și unul pentru date urgente.

**Controlul erorilor** este o problemă importantă deoarece circuitele fizice de comunicații nu sunt perfecte. Se cunosc multe coduri detectoare și corectoare de erori, dar ambele capete ale conexiunii trebuie să se înțeleagă asupra codului utilizat. În plus, receptorul trebuie să aibă cum să-i spună emițătorului care mesaje au fost primite corect și care nu.

Nu toate canalele de comunicații păstrează ordinea mesajelor trimise. Pentru a putea trata o eventuală pierdere a secvențialității, protocolul trebuie să furnizeze explicit receptorului informația necesară pentru a putea reconstitui mesajul. O soluție evidentă este numerotarea fragmentelor, dar această soluție încă nu rezolvă problema fragmentelor care sosesc la receptorul aparent fără legătură cu restul mesajului.

O problemă ce intervine la fiecare nivel se referă la evitarea situației în care un emițător rapid trimit unui receptor lent date la viteza prea mare. Au fost propuse diverse rezolvări și ele vor fi discutate mai târziu. Unele dintre acestea presupun o anumită reacție, directă sau indirectă, prin care receptorul îl informează pe emițător despre starea sa curentă. Altele limitează viteza de transmisie a emițătorului la o valoare stabilită de comun acord cu receptorul. Acest subiect se numește **controlul fluxului**.

O altă problemă care apare la câteva niveluri privește incapacitatea tuturor proceselor de a accepta mesaje de lungime arbitrară. Acest fapt conduce la mecanisme pentru a dezasambla, a transmite și apoi a reasambla mesajele. O problemă asemănătoare apare atunci când procesele insistă să transmită datele în unități atât de mici, încât transmiterea lor separată este ineficientă. În această situație, soluția este să se asambleze împreună mai multe mesaje mici destinate aceluiași receptor și să sedezasambleze la destinație mesajul mare obținut astfel.

Atunci când este neconvenabil sau prea costisitor să se aloce conexiuni separate pentru fiecare pereche de procese comunicante, nivelul implicat în comunicare poate hotărî să utilizeze aceeași conexiune pentru mai multe conversații independente. Atâtă timp cât această **multiplexare** și **demultiplexare** se realizează transparent, ea poate fi utilizată de către orice nivel. Multiplexarea este necesară, de exemplu, în nivelul fizic, unde traficul pentru toate conexiunile trebuie să fie transmis prin cel mult câteva circuite fizice.

Atunci când există mai multe cai între sursă și destinație, trebuie ales un anumit drum. Uneori această decizie trebuie împărțită pe două sau mai multe niveluri. De exemplu, este posibil ca trimiterea unor date de la Londra la Roma să necesite atât o decizie la nivel înalt pentru alegerea ca țară de tranzit a Franței sau a Germaniei - în funcție de legile lor de protejare a secretului datelor - cât și o decizie de nivel scăzut pentru alegerea uneia din multele trasee posibile, pe baza traficului curent. Acest subiect poartă numele de **dirijare** sau **rutare (routing)**.

### 1.3.3 Servicii orientate pe conexiuni și servicii fără conexiuni

Nivelurile pot oferi nivelurilor de deasupra lor două tipuri de servicii: orientate pe conexiuni și fără conexiuni. În această secțiune vom arunca o privire asupra acestor două tipuri și vom examina diferențele între ele.

**Serviciul orientat pe conexiuni** este modelat pe baza sistemului telefonic. Când vrei să vorbești cu cineva, mai întâi ridici receptorul, apoi formezi numărul, vorbești și închizi. Similar, pentru a utiliza un serviciu orientat pe conexiuni, beneficiarul trebuie mai întâi să stabilească o conexiune, să folosească această conexiune și apoi să o elibereze. În esență conexiunea funcționează ca o țeavă: emițătorul introduce obiectele (biții) la un capăt, iar receptorul le scoate afară, în aceeași ordine, la celălalt capăt. În majoritatea cazurilor ordinea este menținută, astfel încât biții să ajungă în aceeași ordine în care au fost trimiși.

În anumite cazuri când se stabilește o conexiune, transmițătorul, receptorul și subrețea negociază parametrii care vor fi folosiți, cum sunt dimensiunea maximă a mesajului, calitatea impusă a serviciilor, și alte probleme de acest tip. De obicei, una dintre părți face o propunere și cealaltă parte poate să o accepte, să o rețeleze sau să facă o contraproponere.

**Serviciul fără conexiuni** este modelat pe baza sistemului poștal. Toate mesajele (scrisorile) conțin adresele complete de destinație și fiecare mesaj circulă în sistem independent de celelalte. În mod normal, atunci când două mesaje sunt trimise la aceeași destinație, primul expediat este primul care ajunge. Totuși, este posibil ca cel care a fost expediat primul să întârzie și să ajungă mai repede al doilea. În cazul unui serviciu orientat pe conexiuni, aşa ceva este imposibil.

Fiecare serviciu poate fi caracterizat printr-o **calitate a serviciului**. Unele servicii sunt sigure în sensul că nu pierd date niciodată. De obicei, un serviciu sigur se implementează obligând receptorul să confirme primirea fiecărui mesaj, astfel încât expeditorul să fie sigur că mesajul a ajuns la destinație. Procesul de confirmare introduce un timp suplimentar și întârzieri. Aceste dezavantaje sunt adesea acceptate, însă uneori ele trebuie evitate.

Transferul de fișiere este una din situațiile tipice în care este adecvat un serviciu sigur orientat pe conexiuni. Proprietarul fișierului dorește să fie sigur că toți biții ajung corect și în aceeași ordine în care au fost trimiși. Foarte puțini utilizatori ai transferului de fișiere ar prefera un serviciu care uneori amestecă sau pierde câțiva biți, chiar dacă acest serviciu ar fi mult mai rapid.

Serviciul sigur orientat pe conexiuni admite două variante: secvențele de mesaje și fluxurile de octeți. Prima variantă menține delimitarea între mesaje. Când sunt trimise două mesaje de 1024 de octeți, ele vor sosi sub forma a două mesaje distincte de 1024 de octeți, niciodată ca un singur mesaj de 2048 de octeți. În a doua variantă, conexiunea este un simplu flux de octeți și nu există delimitări între mesaje. Când receptorul primește 2048 de octeți, nu există nici o modalitate de a spune dacă ei au fost trimiși sub forma unui mesaj de 2048 octeți, a două mesaje de 1024 de octeți sau a 2048 mesaje de câte 1 octet. Dacă paginile unei cărți sunt expediate unei mașini fotografice de tipărit printr-o rețea, sub formă de mesaje, atunci delimitarea mesajelor poate fi importantă. Pe de altă parte, în cazul unui utilizator care se conectează la un server aflat la distanță, este nevoie numai de un flux de octeți de la calculatorul utilizatorului la server. Delimitarea mesajelor nu mai este relevantă.

Așa cum am menționat mai sus, întârzierile introduse de confirmări sunt inacceptabile pentru unele aplicații. O astfel de aplicație se referă la traficul de voce digitizată. Pentru abonații telefonici este preferabil să existe puțin zgomot pe linie sau să audă ocazional câte un cuvânt distorsionat decât să se producă o întârziere din cauza așteptării confirmării. Similar, atunci când se transmite o videoconferință, câțiva pixeli diferiți nu reprezintă o problemă, în schimb întreburile pentru a corecta erorile ar fi extrem de supărătoare.

Nu orice aplicație necesită conexiuni. De exemplu, în măsura în care poșta electronică devine ceva tot mai ușual, se poate să nu apară foarte curând publicitatea prin poștă electronică? Expeditorul de publicitate prin poștă electronică probabil că nu vrea să se complice stabilind și apoi eliberând o conexiune doar pentru un singur mesaj. Nici furnizarea la destinație cu o rată de corectitudine de 100% nu este esențială, mai ales dacă lucru acesta costă mai mult. Tot ceea ce se cere este un mijloc de a trimite un singur mesaj cu o probabilitate mare de a ajunge la destinație, dar fără o garanție în acest sens. Serviciul nesigur (adică neconfirmat) fără conexiuni este deseori numit **serviciu datagramă**, prin analogie cu serviciul de telegramme - care, la rândul său, nu prevede trimitera unei confirmări către expeditor.

În alte situații, avantajul de a nu fi necesară stabilirea unei conexiuni pentru a trimite un mesaj scurt este de dorit, dar siguranța este de asemenea esențială. Aceste aplicații pot utiliza **serviciul datagramă confirmat**. Este ca și cum ai trimite o scrisoare recomandată și ai solicita o confirmare de primire. În clipa în care sosește confirmarea, expeditorul este absolut sigur că scrisoarea a fost livrată la destinația corectă și nu a fost pierdută pe drum.

Mai există un serviciu, și anume **serviciul cerere-răspuns**. În acest serviciu emițătorul transmite o singură datagramă care conține o cerere; replica primită de la receptor conține răspunsul. În această categorie intră, de exemplu, un mesaj către biblioteca locală în care se întrebă unde este vorbită limba Uighur. Serviciul cerere-răspuns este utilizat în mod frecvent pentru a implementa comunicația în modelul client-server: clientul lansează o cerere și serverul răspunde la ea. În fig. 1-16 sunt rezumate tipurile de servicii discutate mai sus.

Conceptul de a utiliza comunicații nesigure poate părea derulant la început. La urma urmei, de ce ar prefera cineva comunicațiile nesigure în locul comunicațiilor sigure? Mai întâi, comunicațiile sigure (ceea ce înseamnă, pentru noi, confirmate) pot să nu fie disponibile. De exemplu, Ethernet-ul nu oferă comunicații sigure. Pachetele pot fi uneori alterate în timpul tranzitului. Urmează ca protocolele nivelurilor superioare să se ocupe de această problemă.

	<b>Serviciu</b>	<b>Exemplu</b>
Orientate pe conexiuni	Flux de mesaje sigur	Secvență de pagini
Fără conexiuni	Flux de octeți sigur	Conectare la distanță
	Conexiune nesigură	Voce digitizată
	Datagramă nesigură	Publicitate prin e-mail
	Datagramă confirmată	Scrisori cu confirmare
	Cerere-răspuns	Interrogări baze de date

**Fig. 1-16.** Șase tipuri diferite de servicii.

În al doilea rând, întârzierile inerente în cazul în care se oferă servicii sigure ar putea fi inaceptabile, mai ales în cazul aplicațiilor de timp real cum sunt aplicațiile multimedia. Pentru aceste motive, comunicațiile sigure cât și cele nesigure coexistă.

#### 1.3.4 Primitive de serviciu

Un serviciu este specificat formal printr-un set de **primitive** (operații) puse la dispoziția utilizatorului care folosește serviciul. Aceste primitive comandă serviciului să execute anumite acțiuni sau să raporteze despre acțiunile executate de o entitate pereche. Dacă stiva de protocoale este localizată în sistemul de operare, aşa cum se întâmplă de cele mai multe ori, primitivele sunt în mod normal apeluri sistem. Aceste apeluri cauzează o trecere a sistemului de operare în modul nucleu (kernel), care preia controlul mașinii pentru a trimite pachetele necesare.

Setul de primitive disponibile depinde de natura serviciului oferit. Primitivele serviciilor orientate pe conexiuni sunt diferite de cele ale serviciilor fără conexiuni. Ca un exemplu minimal de primitive de serviciu care pot fi oferite pentru a implementa un flux de octeți într-un mediu client-server, putem considera primitivele listate în fig. 1-17.

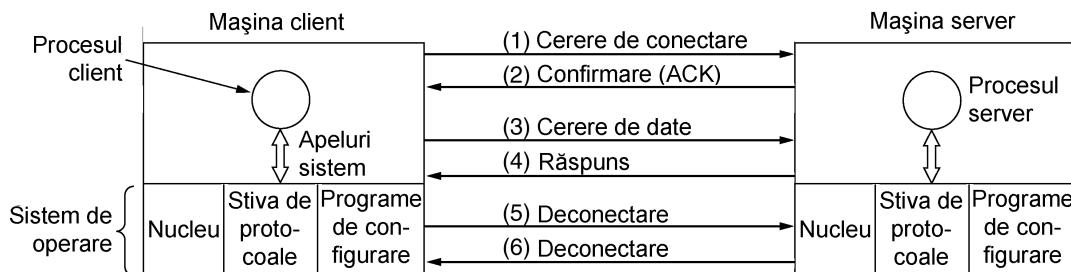
<b>Primitiva</b>	<b>Semnificația</b>
LISTEN (Ascultă)	Blocare în așteptarea unei conexiuni
CONNECT (Conectează)	Stabilirea unei conexiuni cu o entitate pereche aflată în așteptare
RECEIVE (Primește)	Blocare în așteptarea unui mesaj
SEND (Trimite)	Trimite un mesaj entității pereche
DISCONNECT (Deconectează)	Termină o conexiune

**Fig. 1-17.** Cinci primitive de serviciu pentru implementarea unui serviciu simplu orientat pe conexiune.

Aceste primitive pot fi folosite în următorul mod: mai întâi serverul execută LISTEN pentru a indica faptul că este pregătit să accepte conexiuni. Un mod obișnuit de a implementa LISTEN este a

face un apel de sistem blocant. După execuția primitivei, procesul server este blocat până la apariția unei cereri de conectare.

Apoi procesul client execută CONNECT pentru a stabili o conexiune cu serverul. Apelul CONNECT trebuie să specifică cu cine se dorește conectarea, așa că ar putea avea un parametru prin care se transmite adresa serverului. De cele mai multe ori, sistemul de operare va trimite un prim pachet entității pereche cerându-i să se conecteze, după cum este arătat de (1) în fig. 1-18. Procesul client este suspendat până când apare un răspuns. Când pachetul ajunge la server, el este procesat de sistemul de operare al acestuia. Când sistemul de operare observă că pachetul cere o conexiune, verifică dacă există un ascultător. Dacă da, va face două lucruri: va debloca ascultătorul și va trimite înapoi o confirmare (2). Sosirea acestei confirmări elibereză apoi clientul. În acest moment, atât clientul cât și serverul sunt în execuție și au stabilit o conexiune între ei. Este important de observat că secvența de confirmare (2) este generată de codul protocolului însuși, nu ca răspuns al unei primitive de la nivelul utilizatorului. Dacă apare o cerere de conexiune și nu există nici un ascultător, rezultatul este nedefinit. În anumite sisteme, pachetul poate fi păstrat un scurt timp într-o coadă, anticipând o eventuală comandă LISTEN.



**Fig. 1-18.** Pachetele trimise într-o simplă interacțiune client-server pe o rețea orientată pe conexiuni.

Analogia evidentă între acest protocol și viața reală este cazul clientului care sună la directorul departamentului de service al unei companii. Directorul stă lângă telefon pentru a putea răspunde în cazul în care acesta sună. Clientul face un apel. Când directorul ridică receptorul, conexiunea este stabilită.

Pasul următor este ca serverul să execute RECEIVE pentru a se pregăti să accepte prima cerere. În mod normal serverul face această operație de îndată ce a fost eliberat din blocarea impusă de LISTEN, înainte să ajungă confirmarea înapoi la client. Apelul RECEIVE blochează serverul.

Apoi clientul execută SEND pentru a transmite cererea sa (3) urmat de execuția unui RECEIVE pentru a obține răspunsul.

Sosirea pachetului de cerere la mașina server deblochează procesul server astfel încât acesta să poată procesa cererea. După ce a terminat lucrul, folosește SEND pentru a răspunde clientului (4). Sosirea acestui pachet deblochează clientul care poate acum să analizeze răspunsul obținut. Dacă mai există cereri din partea clientului, acesta le poate face acum. Dacă a terminat, poate folosi DISCONNECT pentru a termina conexiunea. De obicei, apelul inițial DISCONNECT este blocant, suspendând clientul și trimînd un pachet către server pentru a-i comunica faptul că respectiva conexiune nu mai este necesară (5). Când serverul primește pachetul, el lansează un DISCONNECT propriu, confirmând cererea clientului și eliberând conexiunea. Când pachetul serverului (6) ajunge

înapoi la mașina clientului, procesul client este eliberat și conexiunea este întreruptă. Foarte pe scurt, așa funcționează comunicațiile orientate pe conexiuni.

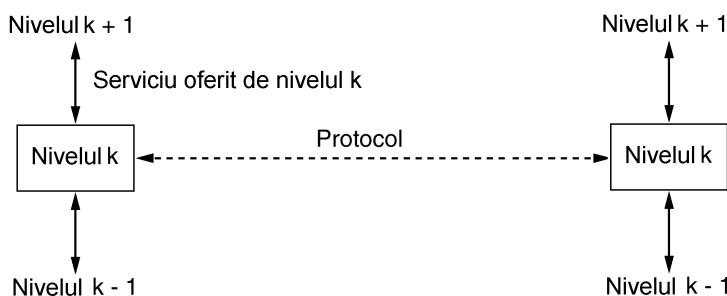
Desigur, viața nu este simplă. Multe dintre lucruri pot să nu funcționeze corect. Sincronizarea poate fi proastă (de exemplu, dacă se încearcă un CONNECT înainte de LISTEN), pachetele se pot pierde și multe altele. Vom studia toate acestea în detaliu ceva mai târziu, dar deocamdată fig. 1-18 rezumă pe scurt modul în care ar putea să funcționeze o comunicație client-server într-o rețea orientată pe conexiuni.

Știind că acele șase pachete sunt necesare pentru a realiza acest protocol, cititorul se poate întreba de ce nu se folosește un protocol fără conexiune în locul său. Răspunsul este că ar fi posibil într-o lume perfectă, și atunci ar fi nevoie de numai două pachete: unul pentru cerere și unul pentru răspuns. Oricum, în cazul real cu mesaje lungi în oricare dintre direcții (de exemplu un fișier de 1 MB), cu erori de transmisie și cu pachete pierdute, situația se modifică. Dacă răspunsul ar avea sute de pachete, dintre care unele s-ar putea pierde în timpul transmisiei, cum ar putea clientul să își dea seama că unele piese lipsesc? Cum ar putea să clientul dacă ultimul pachet recepționat este de fapt ultimul pachet trimis? Să presupunem că de la client se face o cerere pentru un alt doilea fișier. Cum ar putea clientul să diferențieze pachetele din cel de-al doilea fișier de eventualele pachete pierdute din primul fișier? Pe scurt, în lumea reală, un simplu protocol cerere-răspuns implementat într-o rețea nesigură este de cele mai multe ori inadecvat. În cap. 3 vom studia în detaliu o largă varietate de protocoale, care pot rezolva aceste probleme și altele similare. Pentru moment însă este de ajuns să spunem că a avea un flux de octeți sigur și ordonat între procese este de multe ori foarte convenabil.

### 1.3.5 Relația dintre servicii și protocoale

Deși sunt adesea confundate, serviciile și protocoalele reprezintă concepte distincte. Diferența între ele este atât de importantă, încât o subliniem din nou în această secțiune. Un *serviciu* este un set de primitive (operații) pe care un nivel le furnizează nivelului de deasupra să. Serviciul definește ce operații este pregătit nivelul să realizeze pentru utilizatorii săi, dar nu spune nimic despre cum sunt implementate aceste operații. Un serviciu este definit în contextul unei interfețe între două niveluri, nivelul inferior fiind furnizorul serviciului și nivelul superior fiind utilizatorul serviciului.

Prin contrast, un *protocol* este un set de reguli care guvernează formatul și semnificația cadrelor, pachetelor sau mesajelor schimbate între ele de entitățile pereche dintr-un nivel. Entitățile folosesc protocoale pentru a implementa definițiile serviciului lor. Ele sunt libere să își schimbe protocoalele după cum doresc, cu condiția să nu modifice serviciul pe care îl văd utilizatorii. În acest fel, serviciul și protocolul sunt complet decuplate.



**Fig. 1-19.** Relația dintre un server și un protocol.

Cu alte cuvinte, serviciile sunt legate de interfețele dintre niveluri, după cum este ilustrat și în fig. 1-19. Prin contrast, protocolele sunt legate de pachetele trimise între entitățile pereche de pe diferite mașini. Este important să nu existe confuzii între cele două concepte.

Merită să facem o analogie cu limbajele de programare. Un serviciu este ca un tip de date abstracte sau ca un obiect într-un limbaj orientat pe obiecte. Acesta definește operațiile care pot fi aplicate pe un obiect, dar nu specifică modul de implementare a operațiilor. Un protocol se referă la *implementarea* serviciului și nu este vizibil pentru utilizatorul serviciului.

Multe protocole mai vechi nu făceau diferență între serviciu și protocol. Ca urmare, un nivel tipic putea avea o primitivă de serviciu SEND PACKET în care utilizatorul furniza o referință către un pachet complet asamblat. Acest aranjament însemna că toate modificările protocolului erau imediat vizibile pentru utilizatori. Majoritatea proiectanților de rețele privesc acum un astfel de mecanism ca pe o eroare gravă.

## 1.4 MODELE DE REFERINȚĂ

Acum, după ce am discutat la modul abstract structura pe niveluri a rețelelor, a sosit timpul să studiem câteva exemple. În următoarele două secțiuni vom discuta două arhitecturi de rețea importante, modelul de referință OSI și modelul de referință TCP/IP. Deși protocolele asociate cu modelul OSI nu sunt folosite aproape deloc, *modelul* în sine este destul de general și încă valabil, iar caracteristicile puse în discuție la fiecare nivel sunt în continuare foarte importante. Modelul TCP/IP are caracteristici opuse: modelul în sine nu este foarte util, dar protocolele sunt folosite pe scară largă. Din acest motiv, le vom studia pe fiecare în detaliu. În plus, uneori poți învăța mai multe din eșecuri decât din succese.

### 1.4.1 Modelul de referință OSI

Modelul OSI este prezentat în fig. 1-16 (mai puțin mediul fizic). Acest model se bazează pe o propunere dezvoltată de către Organizația Internațională de Standardizare (International Standards Organization - ISO) ca un prim pas către standardizarea internațională a protocolelor folosite pe diferite niveluri (Day și Zimmermann, 1983). A fost revizuit în 1995 (Day, 1995). Modelul se numește **ISO OSI (Open Systems Interconnection**, rom: interconectarea sistemelor deschise), pentru că el se ocupă de conectarea sistemelor deschise - adică de sisteme deschise comunicării cu alte sisteme. În continuare vom folosi mai ales termenul prescurtat de model OSI.

Modelul OSI cuprinde șapte niveluri. Principiile aplicate pentru a se ajunge la cele șapte niveluri sunt următoarele:

1. Un nivel trebuie creat atunci când este nevoie de un nivel de abstractizare diferit.
2. Fiecare nivel trebuie să îndeplinească un rol bine definit.
3. Funcția fiecărui nivel trebuie aleasă acordându-se atenție definirii de protocole standardizate pe plan internațional.
4. Delimitarea nivelurilor trebuie făcută astfel încât să se minimizeze fluxul de informații prin interfețe.

5. Numărul de niveluri trebuie să fie suficient de mare pentru a nu fi nevoie să se introducă în același nivel funcții diferite și suficient de mic pentru ca arhitectura să rămână funcțională.

În continuare vom discuta fiecare nivel al modelului, începând cu nivelul cel mai de jos. Modelul OSI nu reprezintă în sine o arhitectură de rețea, pentru că nu specifică serviciile și protocoalele utilizate la fiecare nivel. Modelul spune numai ceea ce ar trebui să facă fiecare nivel. ISO a produs de asemenea standarde pentru fiecare nivel, însă aceste standarde nu fac parte din modelul de referință propriu-zis. Fiecare din standardele respective a fost publicat ca un standard internațional separat.

### Nivelul fizic

Nivelul fizic se ocupă de transmiterea bițiilor printr-un canal de comunicație. Proiectarea trebuie să garanteze că atunci când unul din capete trimite un bit 1, acesta e receptat în cealaltă parte ca un bit 1, nu ca un bit 0. Problemele tipice se referă la câți volți trebuie utilizați pentru a reprezenta un 1 și câți pentru un 0, dacă transmisia poate avea loc simultan în ambele sensuri, cum este stabilită conexiunea inițială și cum este întreruptă când au terminat de comunicat ambele părți, câți pini are conectorul de rețea și la ce folosește fiecare pin. Aceste aspecte de proiectare au o legătură strânsă cu interfețele mecanice, electrice, funcționale și procedurale, ca și cu mediul de transmisie situat sub nivelul fizic.

### Nivelul legătură de date

Sarcina principală a **nivelului legăturii de date** este de a transforma un mijloc oarecare de transmisie într-o linie care să fie disponibilă nivelului rețea fără erori de transmisie nedetectate. Nivelul legătură de date realizează această sarcină obligând emițătorul să descompună datele de intrare în **cadre de date** (în mod tipic, câteva sute sau câteva mii de octeți) și să transmită cadrele secvențial. Dacă serviciul este sigur, receptorul confirmă fiecare cadru trimisând înapoi un cadru de confirmare pozitivă.

O altă problemă care apare la nivelul legătură de date (și, de asemenea, la majoritatea nivelurilor superioare) este evitarea inundării unui receptor lent cu date provenite de la un emițător rapid. În acest scop sunt necesare mecanisme de reglare a traficului care să permită emițătorului să afle cât spațiu tampon deține receptorul la momentul curent. Controlul traficului și tratarea erorilor sunt deseori integrate. Rețelele cu difuzare determină în nivelul legătură de date o problemă suplimentară: cum să fie controlat accesul la canalul partajat. De această problemă se ocupă un subnivel special al nivelului legătură de date și anume subnivelul de control al accesului la mediu.

### Nivelul rețea

**Nivelul rețea** se ocupă de controlul funcționării subrețelei. O problemă cheie în proiectare este determinarea modului în care pachetele sunt dirijate de la sursă la destinație. Dirijarea se poate baza pe tabele statistice care sunt „cablate” intern în rețea și care sunt schimbate rar. Traseele pot fi de asemenea stabilite la începutul fiecărei conversații, de exemplu la începutul unei sesiuni la terminal (de ex. o operație de login pe o mașină la distanță). În sfârșit, dirijarea poate fi foarte dinamică, traseele determinându-se pentru fiecare pachet în concordanță cu traficul curent din rețea.

Dacă în subrețea există prea multe pachete simultan, ele vor intra unul pe traseul celuilalt și astfel se vor produce gătuiri. Controlul unor astfel de congestii îi revine tot nivelului rețea. Mai general, calitatea serviciilor oferite (întârziere, timp de tranzitare, fluctuații, etc.) este tot o responsabilitate a nivelului rețea.

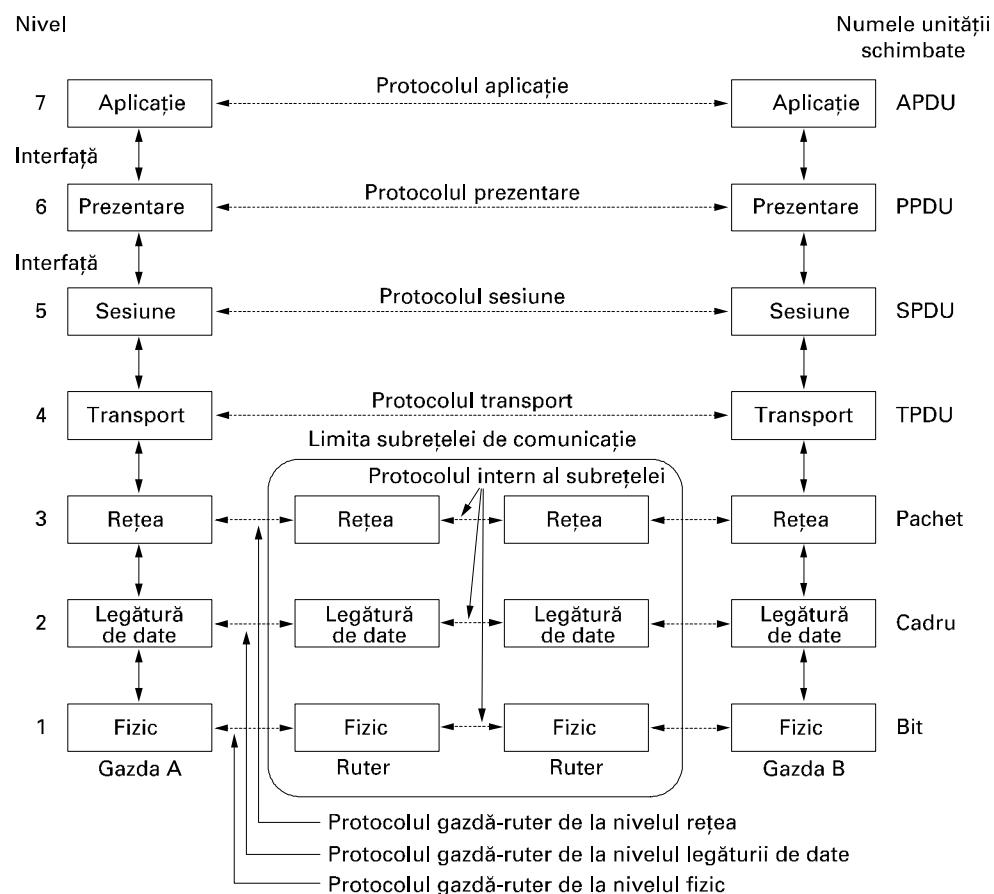
Multe probleme pot apărea când un pachet trebuie să călătorească dintr-o rețea în alta ca să ajungă la destinație. Modul de adresare folosit de a doua rețea poate să difere de cel pentru prima.

A doua rețea poate chiar să nu accepte deloc pachetul pentru că este prea mare. De asemenea, protocolele pot fi diferite și aşa mai departe. Rezolvarea acestor probleme în vederea interconectării retelelor eterogene este sarcina nivelului rețea. În rețelele cu difuzare, problema dirijării este simplă, astfel că nivelul rețea este deseori subțire sau chiar nu există deloc.

### Nivelul transport

Rolul principal al nivelului transport este să accepte date de la nivelul sesiune, să le descompună, dacă este cazul, în unități mai mici, să transfere aceste unități nivelului rețea și să se asigure că toate fragmentele sosesc corect la celălalt capăt. În plus, toate acestea trebuie făcute eficient și într-un mod care izolează nivelurile de mai sus de inevitabilele modificări în tehnologia echipamentelor.

Nivelul transport determină, de asemenea, ce tip de serviciu să furnizeze nivelului sesiune și, în final, utilizatorilor rețelei. Cel mai obișnuit tip de conexiune transport este un canal punct-la-punct fără erori care furnizează mesajele sau octetii în ordinea în care au fost trimisi. Alte tipuri posibile de servicii de transport sunt transportul mesajelor individuale - fără nici o garanție în privința ordinii de livrare - și difuzarea mesajelor către destinații multiple. Tipul serviciului se determină când se stabilește conexiunea. (Ca un comentariu secundar: este imposibil de obținut un canal fără erori; ceea ce oamenii înțeleg prin această expresie este că rata erorilor este destul de mică pentru a fi ignorată în practică).



**Fig. 1-20.** Modelul de referință OSI.

Nivelul transport este un adevărat nivel capăt-la-capăt, de la sursă la destinație. Cu alte cuvinte, un program de pe mașina sursă poartă o conversație cu un program similar de pe mașina destinație, folosind în acest scop antetele mesajelor și mesaje de control. În nivelurile inferioare protocolele au loc între fiecare mașină și vecinii săi imediați (niveluri înlántuite), și nu direct între mașinile sursă și destinație (niveluri capăt-la-capăt), care pot fi separate de numeroase rutere. Diferența între nivelurile de la 1 până la 3, care sunt înlántuite și nivelurile de la 4 la 7, care sunt capăt-la-capăt, este ilustrată în fig. 1-20.

### Nivelul sesiune

Nivelul sesiune permite utilizatorilor de pe mașini diferite să stabilească între ei sesiuni. Sesiunile oferă diverse servicii, inclusiv controlul dialogului (respectarea ordinii în raport cu dreptul de a transmite), **gestionarea jetonului** (prevenirea situației în care două entități încearcă aceeași operație critică în același timp) și **sincronizarea** (introducerea de puncte de control pe parcursul transmisiei lungi, astfel încât, în cazul unui eșec, acestea să poată fi reluate de unde rămăseseră).

### Nivelul prezentare

În particular, spre deosebire de nivelurile inferioare, care se ocupă numai de transferul biților dintr-un loc în altul, nivelul prezentare se ocupă de sintaxa și semantica informațiilor transmise. Pentru a face posibilă comunicarea între calculatoare cu reprezentări diferite ale datelor, structurile de date care se schimbă între ele pot fi definite într-un mod abstract, alături de o codificare standardizată ce va fi utilizată „pe cablu”. Nivelul prezentare gestionează aceste structuri de date abstracte și permite definirea și comunicarea unor structuri de date de nivel mai înalt (de ex. înregistrări bancare).

### Nivelul aplicație

**Nivelul aplicație** conține o varietate de protocole frecvent utilizate. Un exemplu de protocol utilizat pe scară largă este **HTTP (HyperText Transfer Protocol**, rom: protocol de transfer al hiper-textului), care sta la baza **WWW (World Wide Web**, rom: rețea de întindere planetară). Atunci când un program de navigare (browser) accesează o pagină Web, el trimite serverului numele paginii pe care o dorește folosind HTTP. Serverul va trimite ca răspuns pagina. Alte protocole de aplicație sunt folosite pentru transferul fișierelor, poștă electronică, știri în rețea.

#### 1.4.2 Modelul de referință TCP/IP

Să ne îndreptăm acum atenția de la modelul de referință OSI spre modelul de referință utilizat de strămoșul tuturor rețelelor de calculatoare, ARPANET-ul, și de succesorul său, Internet-ul. Deși vom prezenta mai târziu o scurtă istorie a ARPANET-ului, este util să menționăm acum câteva aspecte esențiale. ARPANET a fost o rețea de cercetare sponsorizată de către DoD (U.S. Department of Defense, rom: Departamentul de Apărare al Statelor Unite). În cele din urmă, rețeaua a ajuns să conecteze între ele, utilizând linii telefonice închiriate, sute de rețele universitare și guvernamentale. Atunci când au fost adăugate, mai târziu, rețele prin satelit și radio, interconectarea acestora cu protocolele existente a pus diferențe probleme. Era nevoie de o nouă arhitectură de referință. De aceea, posibilitatea de a interconecta fără probleme mai multe tipuri de rețele a reprezentat de la bun început un obiectiv de proiectare major. Această arhitectură a devenit cunoscută mai târziu sub denumirea de **modelul de referință TCP/IP**, dată după numele celor două protocoale fundamentale utilizate. Arhitectura respectivă a fost definită prima dată în (Cerf și Kahn, 1974). O perspectivă ul-

terioară este prezentată în (Leiner și.a., 1985). Filosofia de proiectare din spatele modelului este discutată în (Clark, 1988).

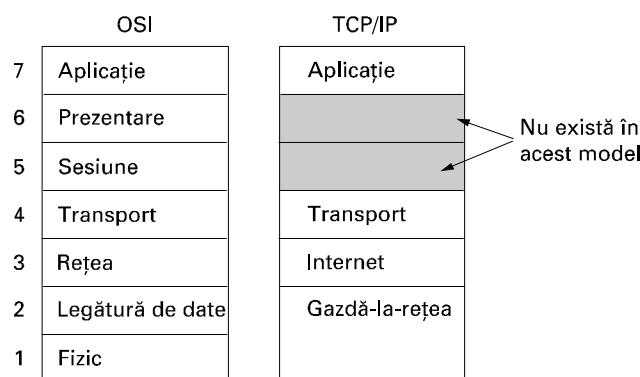
Dată fiind îngrijorarea Departamentului de Aparare că o parte din prețioasele sale gazde, rutere și porți de interconectare ar putea fi distruse dintr-un moment în altul, un alt obiectiv major a fost ca rețea să poată supraviețui pierderii echipamentelor din subrețea fără a fi întrerupte conversațiile existente. Cu alte cuvinte, DoD dorea ca, atât timp cât funcționau mașina sursă și mașina destinație, conexiunile să rămână intacte, chiar dacă o parte din mașini sau din liniile de transmisie erau brusc scoase din funcțiune. Mai mult, era nevoie de o arhitectură flexibilă, deoarece se aveau în vedere aplicații cu cerințe divergente, mergând de la transferul de fișiere până la transmiterea vorbirii în timp real.

### Nivelul internet

Toate aceste cerințe au condus la alegerea unei rețele cu comutare de pachete bazată pe un nivel inter-rețea fără conexiuni. Acest nivel, numit **nivelul internet**, este axul pe care se centrează întreaga arhitectură. Rolul său este de a permite gazdelor să emită pachete în orice rețea și a face ca pachetele să circule independent până la destinație (fiind posibil ca aceasta să se găsească pe o altă rețea). Pachetele pot chiar să sosească într-o ordine diferită față de cea în care au fost trimise, caz în care – dacă se dorește livrarea lor ordonată - rearanjarea cade în sarcina nivelurilor superioare. De observat că „internet” este folosit aici într-un sens generic, chiar dacă acest nivel este prezent și în Internet.

Aici, analogia este cu sistemul de poștă (clasică). O persoană dintr-o anumită țară poate depune într-o cutie poștală mai multe scrisori internaționale și, cu puțin noroc, majoritatea scrisorilor vor ajunge la adresa corectă din țara de destinație. Probabil că scrisorile vor trece pe drum prin mai multe oficii de cartare, dar acest lucru se face transparent pentru utilizatori. Mai mult, faptul că fiecare țară (adică fiecare rețea) are propriile timbre, propriile mărimi favorite de plicuri și propriile reguli de livrare este ascuns beneficiarilor.

Nivelul internet definește oficial un format de pachet și un protocol numit **IP (Internet Protocol)**, rom: protocol Internet). Sarcina nivelului internet este să livreze pachete IP către destinație. Problemele majore se referă la dirijarea pachetelor și evitarea congestiei. În consecință, este rezonabil să spunem că nivelul internet din TCP/IP funcționează asemănător cu nivelul rețea din OSI. Fig. 1-21 arată această corespondență.

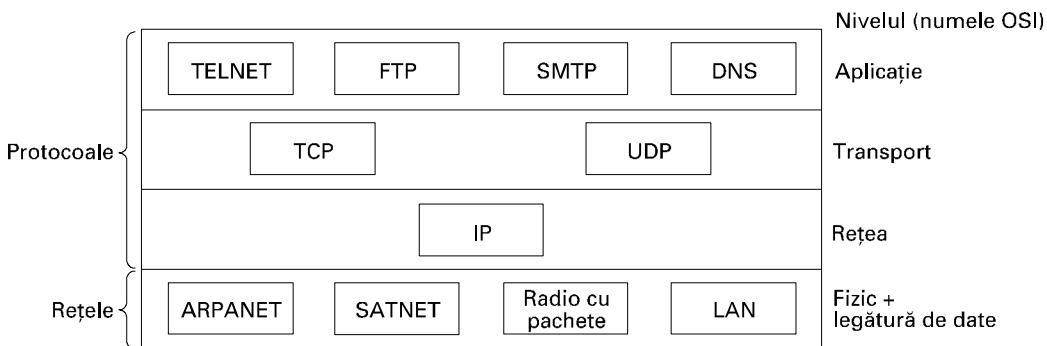


**Fig. 1-21.** Modelul de referință TCP/IP.

### Nivelul transport

Nivelul situat deasupra nivelului internet din modelul TCP/IP este frecvent numit **nivelul transport**. Acesta este proiectat astfel, încât să permită conversații între entitățile pereche din gazdele sursă și, respectiv, destinație, la fel ca în nivelul transport OSI. În acest sens au fost definite două protocoale capăt-la-capăt. Primul din ele, **TCP (Transmission Control Protocol**, rom: protocolul de control al transmisiei), este un protocol sigur orientat pe conexiuni care permite ca un flux de octeți trimiși de pe o mașină să ajungă fără erori pe orice altă mașină din inter-rețea. Acest protocol fragmentează fluxul de octeți în mesaje discrete și pasează fiecare mesaj nivelului internet. La destinație, procesul TCP receptor reasamblează mesajele primite într-un flux de ieșire. TCP tratează totodată controlul fluxului pentru a se asigura că un emițător rapid nu inundă un receptor lent cu mai multe mesaje decât poate acesta să prelucreze.

Al doilea protocol din acest nivel, **UDP (User Datagram Protocol**, rom: protocolul datagramelor utilizator), este un protocol nesigur, fără conexiuni, destinat aplicațiilor care doresc să utilizeze propria lor securitate și control al fluxului, și nu pe cele asigurate de TCP. Protocolul UDP este de asemenea mult folosit pentru interogări rapide întrebare-răspuns, client-server și pentru aplicații în care comunicarea promptă este mai importantă decât comunicarea cu acuratețe, așa cum sunt aplicațiile de transmisie a vorbirii și a imaginilor video. Relația dintre IP, TCP și UDP este prezentată în fig. 1-22. De când a fost dezvoltat acest model, IP a fost implementat pe multe alte rețele.



**Fig. 1-22.** Protocole și rețele din modelul TCP/IP inițial.

### Nivelul aplicație

Modelul TCP/IP nu conține niveluri sesiune sau prezentare. Acestea nu au fost incluse pentru că nu s-a simțit nevoie lor. Experiența modelului OSI a dovedit că această viziune a fost corectă: în majoritatea aplicațiilor, nivelurile respective nu sunt de mare folos.

Deasupra nivelului transport se află **nivelul aplicație**. Acesta conține toate protocolele de nivel mai înalt. Așa cum se vede din fig. 1-22, primele protocole de acest gen includeau terminalul virtual (TELNET), transferul de fișiere (FTP) și poșta electronică (SMTP). Protocolul de terminal virtual permite unui utilizator de pe o mașină să se conecteze și să lucreze pe o mașină aflată la distanță. Protocolul de transfer de fișiere pune la dispoziție o modalitate de a muta eficient date de pe o mașină pe alta. Poșta electronică a fost la origine doar un tip de transfer de fișiere, dar ulterior a fost dezvoltat un protocol specializat (SMTP – Simple Mail Transfer Protocol, rom: Protocol simplu de transfer al poștei) pentru acest serviciu. Pe parcursul anilor, la aceste protocole s-au adăugat multe altele, așa cum sunt Serviciul Numelor de Domenii (Domain Name Service - DNS) pentru stabilirea corespondenței dintre numele gazdelor și adresele rețelelor, NNTP, protocolul

utilizat pentru a transfera articole de știri USENET, HTTP, folosit pentru aducerea paginilor de pe Web și multe altele.

### Nivelul gazdă-rețea

Sub nivelul internet se află necunoscutul. Modelul de referință TCP/IP nu spune mare lucru despre ce se întâmplă acolo, însă menționează că gazda trebuie să se lege la rețea, pentru a putea trimite pachete IP, folosind un anumit protocol. Acest protocol nu este definit și variază de la gazdă la gazdă și de la rețea la rețea. Cărțile și articolele despre TCP/IP rareori discută despre acest protocol.

#### 1.4.3 O comparație între modelele de referință OSI și TCP

Modelele de referință OSI și TCP/IP au multe lucruri în comun. Amândouă se bazează pe conceptul unei stive de protocoale independente. De asemenea, funcționalitatea nivelurilor este în linii mari similară. De exemplu, în ambele modele, nivelurile până la nivelul transport inclusiv sunt necesare pentru a pune la dispoziția proceselor care doresc să comunice un serviciu de transport capăt-la-capăt independent de rețea. Nivelurile respective formează furnizorul de transport. Din nou, în ambele modele, nivelurile de deasupra transportului sunt beneficiari orientați pe aplicații ai serviciului de transport.

În pofida acestor similitudini fundamentale, între cele două modele există și multe deosebiri. În această secțiune ne vom concentra asupra diferențelor cheie dintre cele două modele de referință. Este important de subliniat că vom compara aici *modelele de referință*, nu *stivele de protocoale* corespunzătoare. Protocoalele propriu-zise vor fi discutate mai târziu. Pentru o întreagă carte consacrată comparației și diferențelor dintre TCP/IP și OSI, a se vedea (Piscitello și Chapin, 1993).

Trei concepte sunt esențiale pentru modelul OSI:

1. Servicii
2. Interfețe
3. Protocoale

Probabil că cea mai mare contribuție a modelului OSI este că a făcut explicită diferența între aceste trei concepte. Fiecare nivel realizează niște servicii pentru nivelul situat deasupra sa. Definiția *serviciului* spune ce face nivelul, nu cum îl folosesc entitățile de deasupra sa sau cum funcționează nivelul. El definește semantica nivelului.

*Interfața* unui nivel spune proceselor aflate deasupra sa cum să facă accesul. Interfața precizează ce reprezintă parametrii și ce rezultat se obține. Nici interfața nu spune nimic despre funcționarea internă a nivelului.

În sfârșit, *protocolele* pereche folosite într-un nivel reprezintă treaba personală a nivelului. Nivelul poate folosi orice protocol dorește, cu condiția ca acesta să funcționeze (adică să îndeplinească serviciul oferit). Nivelul poate de asemenea să schimbe protocoalele după cum vrea, fără ca acest lucru să afecteze programele din nivelurile superioare.

Aceste idei se potrivesc foarte bine cu ideile moderne referitoare la programarea orientată pe obiect. Un obiect, ca și un nivel, posedă un set de metode (operații) care pot fi invocate de către procese din afara obiectului. Semanticele acestor metode definesc multimea de servicii pe care le oferă obiectul. Parametrii și rezultatele metodelor formează interfața obiectului. Codul intern al obiectului reprezintă protocolul său și nu este vizibil și nici important în afara obiectului.

Deși lumea a încercat ulterior să îl readapteze pentru a fi mai asemănător modelului OSI, modelul TCP/IP nu a făcut inițial o distincție clară între serviciu, interfață și protocol. De exemplu, singurele servicii veritabile oferite de nivelul internet sunt SEND IP PACKET și RECEIVE IP PACKET.

În consecință, protocoalele din modelul OSI sunt mai bine ascunse decât în modelul TCP/IP și pot fi înlocuite relativ ușor pe măsură ce se schimbă tehnologia. Capacitatea de a face asemenea modificări reprezintă unul din scopurile principale ale organizării protocoalelor pe niveluri în modelul OSI.

Modelul de referință OSI a fost conceput *înainte* să fie inventate protocoalele corespunzătoare. Ordinea respectivă semnifică faptul că modelul nu a fost orientat către un set specific de protocoale, fiind prin urmare destul de general. Reversul este că proiectanții nu au avut multă experiență în ceea ce privește acest subiect și nu au avut o idee coerentă despre împărțirea funcțiilor pe niveluri.

De exemplu, nivelul legătură de date se ocupa inițial numai cu rețelele punct-la-punct. Atunci când au apărut rețelele cu difuzare, a trebuit să fie introdus în model un subnivel nou. Când lumea a început să construiască rețele reale utilizând modelul OSI și protocoalele existente, s-a descoperit că acestea nu se potriveau cu specificațiile serviciului cerut (minunea minunilor), astfel că a trebuit introdusă în model convergența subnivelurilor, ca să existe un loc pentru a glosa pe marginea diferențelor. În sfârșit, comitetul se aștepta inițial ca fiecare țară să aibă câte o rețea care să fie în custodia guvernului și să folosească protocoalele OSI, să că nu s-a dat nici o atenție interconectării. Pentru a nu mai lungi povestea, să spunem doar că lucrurile s-au petrecut altfel.

În ceea ce privește TCP/IP, lucrurile stau exact pe dos: mai întâi au apărut protocoalele, iar modelul a fost de fapt doar o descriere a protocoalelor existente. Cu protocoalele respective nu era nici o problemă: ele se potriveau perfect cu modelul. Singurul necaz era că *modelul* nu se potrivea cu nici o altă stivă de protocoale. Prin urmare, modelul nu a fost prea util pentru a descrie alte rețele non-TCP/IP.

Pentru a ne întoarce de la subiectele filosofice la subiecte mai specifice, o diferență evidentă între cele două modele se referă la numărul de niveluri: modelul OSI are șapte niveluri, iar TCP/IP are patru. Ambele modele au niveluri (inter-)rețea, transport și aplicație, dar restul nivelurilor sunt diferite.

O altă deosebire privește subiectul comunicării fără conexiuni față de cel al comunicării orientată pe conexiuni. Modelul OSI suportă ambele tipuri de comunicații la nivelul rețea, dar numai comunicații orientate pe conexiuni în nivelul transport, unde acest fapt are importanță (pentru că serviciul de transport este vizibil utilizatorilor). Modelul TCP/IP are numai un mod (fără conexiuni) la nivelul rețea, dar suportă ambele moduri la nivelul transport, ceea ce lasă utilizatorilor posibilitatea alegерii. Această alegere este importantă în mod special pentru protocolele întrebare-răspuns simple.

#### 1.4.4 O critică a modelului și protocoalelor OSI

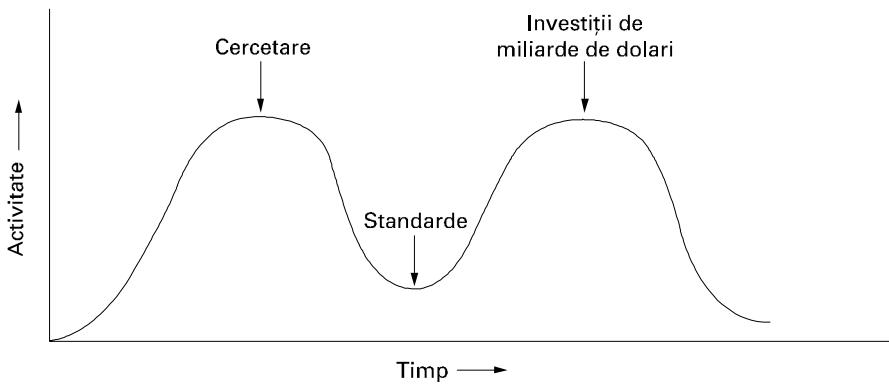
Nici modelul și protocoalele OSI și nici modelul și protocoalele TCP/IP nu sunt perfecte. Asupra lor se pot formula, și s-au formulat, câteva critici. În prezentă și în următoarea secțiune vom vedea unele dintre aceste critici. Vom începe cu OSI, după care vom examina TCP/IP.

La momentul când a fost publicată a doua ediție a acestei cărți (1989), majoritatea expertilor în domeniu credeau că modelul și protocoalele OSI se vor impune peste tot și vor elibera orice concurent. Acest lucru nu s-a întâmplat. De ce? O privire spre lecțiile trecutului poate fi utilă. Aceste lecții pot fi rezumate astfel:

1. Ratarea momentului.
2. Tehnologii proaste.
3. Implementări proaste.
4. Politici proaste.

### Ratarea momentului

Să vedem mai întâi prima problemă: ratarea momentului. Momentul la care se stabilește un standard este absolut critic pentru succesul acestuia. David Clark de la M.I.T. are o teorie asupra standardelor pe care o numește *Apocalipsa celor doi elefanți* și care este ilustrată în fig. 1-23.



**Fig. 1-23.** Apocalipsa celor doi elefanți.

Această figură arată volumul de activitate desfășurată în jurul unui subiect nou. Când subiectul este lansat, are loc o explozie a activității de cercetare sub formă de discuții, articole și întâlniri. După un timp, cercetarea se reduce foarte mult, subiectul este descoperit de companii și piața cunoaște un val de investiții de miliarde de dolari.

Este esențial ca standardele să fie definite în intervalul dintre cei doi „elefanți”. Dacă ele sunt definite prea devreme, înainte să se încheie cercetarea, atunci subiectul poate să nu fie încă destul de bine înțeles, ceea ce conduce la standarde proaste. Dacă ele sunt definite prea târziu, atunci probabil că atât de multe firme au făcut deja investiții majore realizând lucrurile altfel, încât standardele sunt efectiv ignorate. Dacă intervalul dintre cei doi elefanți este foarte scurt (pentru că toată lumea arde de nerăbdare să treacă la lucru), atunci cei care dezvoltă standardele pot fi prinși la mijloc și striviti.

Acum se vede că protocolele OSI standard au fost strivite. La momentul apariției lor, protocolele concurente TCP/IP erau deja folosite pe scară largă în universități, în cercetare. Înainte să vină valul investițiilor de miliarde de dolari, piața din domeniul academic era destul de dezvoltată pentru ca multe firme să înceapă, prudent, să ofere produse TCP/IP. Când a apărut OSI, firmele nu au mai vrut, decât forțate, să sprijine o sau două stivă de protocoale, și, prin urmare, n-au apărut nici un fel de oferte inițiale din partea lor. Fiecare firmă aștepta să înceapă celelalte firme, aşa că până la urmă n-a mai început nici o firmă și fenomenul OSI nu s-a mai produs niciodată.

### Tehnologii proaste

Al doilea motiv pentru care OSI n-a prins niciodată este că atât modelul cât și protocolele au defecte. Opțiunea pentru șapte niveluri a fost mai mult politică decât tehnică, și două dintre niveluri (sesiune și prezentare) sunt aproape goale, în timp ce alte două (legătura de date și rețea) sunt prea aglomerate.

Modelul OSI, alături de protocolele și definițiile de servicii asociate, este extraordinar de complex. Atunci când sunt puse unul peste altul, standardele tipărite au o grosime de câțiva zeci de centimetri. Standardele sunt, de asemenea, dificil de implementat și ineficiente în funcționare. În acest context îmi vine în minte o ghicitoare formulată de Paul Mockapetris și citată în (Rose, 1993):

Î: Ce obții când aplici un standard internațional unui gangster?

R: O persoană care îți face o ofertă pe care n-o poți înțelege.

Pe lângă faptul că este incomprehensibil, o altă problemă cu OSI este că unele funcții, cum sunt adresarea, controlul fluxului și controlul erorilor apar repetat în fiecare nivel. Saltzer și alții (1994), de exemplu, au arătat că, pentru a fi eficient, controlul erorilor trebuie făcut la nivelul cel mai înalt și că repetarea sa de atâtea ori în nivelurile de mai jos este adesea inutilă și ineficientă.

### Implementări proaste

Dată fiind enorma complexitate a modelului și a protocolelor, nu este de mirare în faptul că implementările inițiale erau uriașe, greoale și ineficiente. Oricine le încerca se simtea ca opărit. Nu a trecut mult și lumea a asociat „OSI” cu „calitate slabă.” Deși odată cu trecerea timpului produsele au devenit mai bune, imaginea s-a deteriorat.

Din contrar, una din primele implementări de TCP/IP făcea parte din Berkeley UNIX și era desigur de bună (ca să nu mai spunem că era și gratuită). Lumea a început să o folosească repede, ceea ce a determinat apariția unei comunități largi de utilizatori, ceea ce a dus mai departe la îmbunătățiri, iar aceasta a dus la o comunitate și mai numeroasă. În acest caz spirala nu cobora, ci urca.

### Politici proaste

Din cauza implementării inițiale, multă lume, în special din mediul academic, a considerat TCP/IP ca o parte din Unix; iar în anii '80 Unix-ul era pentru oamenii din lumea academică cam la fel de popular ca paternitatea (numita apoi incorrect maternitate) sau ca plăcinta cu mere.

OSI, pe de altă parte, a fost gândit ca o creație a ministerelor de telecomunicații europene, apoi a Comunității Europene și, mai târziu, a guvernului Statelor Unite. Această vizionare s-a dovedit adevărată numai în parte; dar chiar ideea în sine - un grup de birocați guvernamentalni încercând să bage un standard inferior tehnic pe gâtul bieților cercetători și programatorii care stau în tranșee și dezvoltă efectiv rețelele de calculatoare - nu a ajutat prea mult. Unii oameni au văzut această abordare în aceeași lumină în care a fost văzut IBM când a anunțat în anii '60 că PL/I era limbajul viitorului, sau DoD care a corectat IBM-ul anunțând că limbajul respectiv era de fapt Ada.

#### 1.4.5 O critică a modelului de referință TCP/IP

Modelul și protocolele TCP/IP au și ele problemele lor. Mai întâi, modelul nu face o distincție clară între conceptele de serviciu, interfață și protocol. O practică recomandabilă în ingineria programării este să se facă diferență între specificație și implementare, ceea ce OSI face cu multă atenție, pe când TCP/IP nu face. De aceea, modelul TCP/IP nu este un ghid prea bun de proiectare a rețelelor noi folosind tehnologii noi.

În al doilea rând, modelul TCP/IP nu este deloc general și nu este aproape deloc potrivit pentru descrierea altor stive de protocole în afara celei TCP/IP. De exemplu, descrierea Bluetooth folosind modelul TCP/IP ar fi aproape imposibilă.

În al treilea rând, nivelul gazdă-rețea nu este deloc un nivel - în sensul normal în care este folosit termenul în contextul protocolelor organizate pe niveluri - ci este o interfață (între nivelurile rețea și legătură de date). Distincția între o interfață și un nivel este crucială și de aceea trebuie să i se acorde atenția cuvenită.

În al patrulea rând, modelul TCP/IP nu distinge (și nici măcar nu menționează) nivelurile fizice și legătură de date. Acestea sunt complet diferite. Nivelul fizic are de-a face cu caracteristicile

transmisiei prin cablu de cupru, fibre optice sau radio. Rolul nivelului legătură de date este să delimitize începutul și sfârșitul cadrelor și să le transporte dintr-o parte în alta cu gradul de siguranță dorit. Un model corect ar trebui să includă ambele niveluri ca niveluri separate. Modelul TCP/IP nu face acest lucru.

În sfârșit, deși protocolele IP și TCP au fost atent gândite și bine implementate, multe din celelalte protocole au fost construite ad-hoc, fiind în general opera cătorva absolvenți care tot „măstereau” la ele până oboseau. Implementările protocalelor erau apoi distribuite gratuit; ca urmare, ele erau larg utilizate, fără să li se asigure suportul necesar, fiind de aceea greu de înlocuit. Unele protocole au ajuns acum să fie mai mult o pacoste. Protocolul de terminal virtual, TELNET, de exemplu, a fost proiectat pentru un terminal teletype mecanic de zece caractere pe secundă. Cu toate acestea, 25 de ani mai târziu, protocolul este încă foarte utilizat.

Pentru a rezuma, în pofida acestor probleme, *modelul OSI* (mai puțin nivelurile sesiune și prezentare) s-a dovedit a fi excepțional de util pentru a discuta rețelele de calculatoare. Din contră, *protocolele OSI* nu au devenit populare. Pentru TCP/IP este adevărată afirmația inversă: *modelul* este practic inexistent, dar *protocalele* sunt larg utilizate. Dat fiind faptul că informaticienilor le place să prepare - și apoi să și măñânce - propria lor prăjitură, în această carte vom folosi un model OSI modificat, dar ne vom concentra în primul rând pe TCP/IP și alte protocole înrudite cu el; de asemenea, vom folosi și protocole mai noi, precum 802, SONET și Bluetooth. Modelul de lucru folosit în carte este modelul hibrid prezentat în fig. 1-24.

Nivelul aplicație
Nivelul transport
Nivelul rețea
Nivelul legătură de date
Nivelul fizic

**Fig. 1-24.** Modelul hibrid de referință care va fi utilizat în această carte.

## 1.5 EXEMPLE DE REȚELE

Subiectul rețelelor de calculatoare acoperă diferite tipuri de rețele, mari și mici, arhicunoscute sau mai puțin cunoscute. Ele au scopuri, dimensiuni și tehnologii diverse. În următoarele secțiuni, vom studia câteva exemple, pentru a avea o idee despre varietatea pe care o poate regăsi oricine în domeniul rețelelor de calculatoare.

Vom porni cu Internet-ul, probabil cea mai cunoscută rețea, și vom studia istoria, evoluția și tehnologiile sale. Apoi vom discuta ATM, care este de multe ori utilizată în nucleul rețelelor (telefonice) mari. Din punct de vedere tehnic, este destul de diferită de Internet, ceea ce evidențiază un contrast interesant. Apoi vom introduce Ethernet, dominantă în cazul rețelelor locale. În final, vom studia IEEE 802.11, standardul pentru rețele fără cablu.

### 1.5.1 Internet

Internet-ul nu este deloc o rețea, ci o colecție vastă de rețele diverse, care utilizează anumite protocole comune și oferă anumite servicii comune. Este un sistem neobișnuit prin aceea că nu a

Cap. 4 se referă la subnivelul de acces la mediu, care face parte din nivelul legătură de date. Problema fundamentală cu care se ocupă este cum să determine cine poate folosi rețeaua - atunci când rețeaua constă dintr-un singur canal partajat, aşa cum se întâmplă în majoritatea LAN-urilor și în unele rețele de sateliți. Sunt date multe exemple din domeniul LAN-urilor cu cablu sau fără (în special Ethernet), din cel al MAN-urilor fără fir, din cadrul rețelelor bazate pe Bluetooth și al rețelelor de sateliți. Tot aici sunt discutate și punctile, care se folosesc pentru a interconecta LAN-urile.

Cap. 5 se ocupă de nivelul rețea, în special de dirijare, cu prezentarea mai multor algoritmi de dirijare, atât statici cât și dinamici. Chiar dacă se folosesc algoritmi de rutare foarte buni, dacă traficul cerut este mai mare decât cel pe care îl poate dirija rețeaua, se ajunge la congestia rețelei, aşa că se va discuta despre congestie și despre cum poate fi ea evitată. O variantă încă și mai bună decât evitarea congestiei este oferirea unei garanții de calitate a serviciilor. Își acordă subiect va fi abordat aici. Interconectarea rețelelor eterogene în inter-rețele conduce la numeroase probleme care sunt discutate aici. Se acordă mare atenție nivelurilor din Internet.

Cap. 6 se ocupă de nivelul transport. Se discută pe larg protocolele orientate pe conexiuni, deoarece ele sunt necesare în numeroase aplicații. Se discută în detaliu un exemplu de serviciu de transport și implementarea sa. Este prezentat chiar și codul sursă pentru acest exemplu simplu, pentru a se putea demonstra modul în care poate fi implementat. Ambele protocole din Internet – UDP și TCP – sunt discutate în detaliu și este abordată problema performanțelor lor. În plus, se discută despre problemele impuse de rețelele fără fir.

Cap. 7 se ocupă de nivelul aplicație, de protocolele și aplicațiile sale. Primul subiect este DNS, care este cartea de telefoane a Internet-ului. Apoi urmează poșta electronică, inclusiv o discuție despre protocolele sale. Apoi ne vom muta atenția asupra Web-ului, cu discuții detaliate despre conținut static, conținut dinamic, ce se întâmplă la client, ce se întâmplă pe server, protocole, performanță, Web fără fir. În cele din urmă vom examina informația multimedia care este transmisă prin rețea, inclusiv fluxuri audio, radio prin Internet și video la cerere.

Cap. 8 se referă la securitatea rețelelor. Acest subiect include aspecte legate de fiecare dintre niveluri, aşa că este mai ușor de tratat către final, când toate nivelurile au fost deja explicate pe larg. Capitolul începe cu o introducere în criptografie. În continuare, este prezentat modul în care criptografia poate fi utilizată pentru a securiza comunicațiilor, poșta electronică și Web-ul. Cartea se încheie cu o discuție despre anumite domenii în care securitatea interferează cu intimitatea, libertatea de exprimare, cenzura, precum și alte probleme sociale care decurg de aici.

Cap. 9 conține o listă adnotată de lecturi sugerate, aranjate în ordinea capitolelor. Lista este gândită ca un ajutor pentru cititorii care doresc să continue studiul rețelelor. Capitolul are de asemenea o bibliografie alfabetică a tuturor referințelor citate în această carte.

Situsul Web al autorului de la Prentice Hall: <http://www.prenhall.com/tanenbaum> are o pagină cu legături la mai multe sinteze, liste de întrebări frecvente (FAQs), companii, consorții industriale, organizații profesionale, organizații de standardizare, tehnologii, lucrări științifice și altele.

## 1.9 REZUMAT

Rețelele de calculatoare pot fi utilizate pentru numeroase servicii, atât pentru firme cât și pentru persoane particulare. Pentru companii, rețelele de calculatoare personale care folosesc servere par-

tajate asigură accesul la informațiile corporației. De obicei, acestea urmează modelul client-server, cu stațiile de lucru clienți pe mesele de lucru ale angajaților accesând serverele puternice din camera mașinilor. Pentru persoane particulare, rețelele oferă acces la o mulțime de informații și de resurse de divertisment. De cele mai multe ori persoanele particulare accesează Internet-ul folosind un modem pentru a apela un ISP, deși din ce în ce mai mulți utilizatori au chiar și acasă o conexiune Internet fixă, permanentă. Un domeniu care se dezvoltă rapid este acela al rețelelor fără fir, care conduc la dezvoltarea de noi aplicații, cum ar fi mobilitatea accesului la poșta electronică și comerțul mobil.

În mare, rețelele pot fi împărțite în LAN-uri, MAN-uri, WAN-uri și inter-rețele, fiecare cu caracteristicile, tehnologiile, vitezele și rolurile sale proprii. LAN-urile acoperă suprafața unei clădiri și lăcuză la viteze mari, MAN-urile acoperă suprafața unui oraș – de exemplu rețeaua de televiziune prin cablu, care este actualmente folosită de mulți dintre utilizatori și pentru conectarea la Internet. WAN-urile se întind pe suprafața unei țări sau a unui continent. LAN-urile și MAN-urile sunt necomutate (adică nu au rutere); WAN-urile sunt comutate. Rețelele fără fir devin din ce în ce mai populare, în special la nivelul rețelelor locale. Rețelele pot fi interconectate pentru a forma inter-rețele.

Programele de rețea constau din protocole, adică reguli prin care procesele pot să comunice. Protocolele pot fi fie fără conexiuni, fie orientate pe conexiuni. Majoritatea rețelelor asigură suport pentru ierarhiile de protocole, fiecare nivel asigurând servicii pentru nivelurile de deasupra sa și izolându-le de detaliile protocolelor folosite în nivelurile de mai jos. Stivele de protocole se bazează în mod tipic fie pe modelul OSI, fie pe modelul TCP/IP. Ambele modele posedă niveluri rețea, transport și aplicație, dar ele diferă în ceea ce privește celelalte niveluri. Problemele care apar în procesul de proiectare a acestor protocole includ multiplexarea, controlul traficului, controlul erorilor și încă altele. O mare parte a acestei cărți este dedicată protocolelor și proiectării lor.

Rețelele oferă servicii utilizatorilor lor. Aceste servicii pot fi orientate pe conexiune sau fără conexiune. În anumite rețele, serviciile fără conectare sunt oferite la un anumit nivel și pot fi completeate cu serviciile orientate pe conexiune oferite de un alt nivel.

Ca rețele bine-cunoscute sunt menționate Internet-ul, rețelele ATM, Ethernet-ul și LAN-ul fără fir, standard denumit IEEE 802.11. Internet-ul a evoluat din ARPANET, prin adăugarea de noi rețele pentru a se forma o inter-rețea. În prezent, Internet-ul este în fapt o colecție de multe mii de rețele și nu o singură rețea. Ceea ce caracterizează această colecție este folosirea stivei TCP/IP peste tot. Rețelele ATM sunt răspândite mai ales în sistemele de telefonie pentru trafic de date intensiv. Ethernet-ul este cea mai populară rețea locală și este implementată în majoritatea companiilor mari și în universități. În fine, rețelele locale fără fir, cu viteze de transfer surprinzător de mari (până la 54 Mbps) încep să fie folosite pe scară largă.

Pentru a putea determina mai multe calculatoare să comunice între ele este nevoie de o importantă muncă de standardizare, atât pentru partea de echipamente (hardware), cât și pentru partea de programe (software). Organizațiile ca ITU-T, ISO, IEEE și IAB administrează diverse părți din procesul de standardizare.

## 1.10 PROBLEME

1. Imaginea-vă că v-ați dresat câinele St. Bernard, pe nume Bernie, ca, în locul clasicei sticle cu rom, să poarte o cutie cu trei benzi de 8 mm. (Când îți se umple discul, respectiva cutie reprezintă o ur-

gență.) Aceste benzi conțin fiecare câte 7 gigabytes. Câinele poate călători până la dvs., oriunde v-ați afla, cu 18 km/h. Pentru ce ordin de distanță are Bernie o viteză mai mare de transmisie a datelor decât o linie a cărei viteză de transfer (fără supraîncărcare) este de 150 Mbps?

2. O alternativă la un LAN este pur și simplu un mare sistem, cu divizarea timpului cu terminale pentru toți utilizatorii. Prezentați două avantaje ale unui sistem client-server care folosește un LAN.
3. Performanța unui sistem client-server este influențată de doi factori ai rețelei: lărgimea de bandă (câtă biți poate transporta într-o secundă) și latență (câte secunde durează transferul primului bit de la client la server). Dați un exemplu de rețea care are și lărgime de bandă ridicată și latență mare. Apoi dați un exemplu de rețea cu lărgime de bandă scăzută și latență mică.
4. Pe lângă lărgime de bandă și latență, ce alt parametru este necesar pentru a caracteriza calitatea serviciilor oferite de o rețea folosită pentru trafic de voce digitizată?
5. Un factor de întârziere al unui sistem memorează-și-retransmite cu comutare de pachete este cât de mult timp ia operația de stocare și retrimitere a unui mesaj printr-un comutator. Dacă timpul de comutare este de  $10 \mu s$ , este acesta un factor important în răspunsul unui sistem client-server în care clientul este în New York și serverul în California? Presupuneți că viteza de propagare a semnalului printr-un fir de cupru sau prin fibra optică ar fi de  $2/3$  din viteza luminii în vid.
6. Un sistem client-server folosește o rețea-satelit, cu satelitul amplasat la o înălțime de 40.000 km. În cazul optim, care este întârzierea cu care vine răspunsul la o cerere?
7. În viitor, când toată lumea va avea acasă un terminal conectat la o rețea de calculatoare, vor deveni posibile referendumuri publice imediate pe subiecte de legislație importante. În ultimă instanță ar putea fi chiar eliminate parlamentele, pentru a lăsa voința poporului să se exprime direct. Aspectele pozitive ale unei astfel de democrații directe sunt destul de evidente; discutați unele din aspectele negative.
8. O colecție de cinci rutere trebuie să fie conectată într-o subrețea punct-la-punct. Între două rutere proiectanții pot instala o linie de mare viteză, o linie de viteză medie, o linie de viteză scăzută sau nici o linie. Dacă generația și examinarea fiecărei topologii pe calculator durează 100 ms, cât timp va dura examinarea tuturor topologiilor pentru a o găsi pe cea care se potrivește cel mai bine cu încărcarea prevăzută?
9. Un grup de  $2^n - 1$  rutere sunt interconectate într-un arbore binar centralizat, cu un ruter în fiecare nod al arborelui. Ruterul  $i$  comunică cu ruterul  $j$  trimițând un mesaj rădăcinii arborelui. Rădăcina trimite apoi mesajul înapoi în jos până la  $j$ . Deducreți o expresie aproximativă pentru numărul mediu de salturi pe mesaj în cazul unui număr  $n$  mare, presupunând că toate perechile de rutere sunt la fel de probabile.
10. Un dezavantaj al unei subrețele cu difuzare este risipa de capacitate datorată multiplelor gazde care încearcă să acceseze canalul în același timp. Ca un exemplu simplist, să presupunem că timpul este împărțit în intervale discrete și fiecare din cele  $n$  gazde încearcă să utilizeze canalul cu probabilitatea  $p$  în timpul fiecărui interval. Ce fracțiune din intervale se pierde datorită coliziunilor?
11. Care sunt două din motivele utilizării protocolelor organizate pe niveluri?

12. Președintelui Companiei de Vopsele Speciale îi vine ideea să lucreze împreună cu un producător local de bere în scopul de a produce o cutie de bere invizibilă (ca o măsură anti-gunoii). Președintele comandă departamentului său juridic să analizeze ideea, iar acesta cere ajutorul, la rândul său, departamentului de ingineri. Ca rezultat, inginerul șef îl cheamă pe inginerul-șef de la cealaltă firmă pentru a discuta aspectele tehnice ale proiectului. Apoi, inginerii prezintă un raport către departamentele juridice respective, iar acestea aranjează prin telefon aspectele legale. În final, cei doi președinți de firme discută partea financiară a afacerii. Este acesta un exemplu de protocol multilinier în sensul modelului OSI?6. Care sunt adresele SAP în cazul difuzării radio FM ?
13. Care este principala diferență între comunicarea fără conexiuni și comunicarea orientată pe conexiuni?
14. Două rețele furnizează, fiecare, servicii orientate pe conexiuni sigure. Una din ele oferă un flux sigur de octeți, iar cealaltă oferă un flux sigur de mesaje. Sunt acestea identice? Dacă da, de ce se face această distincție? Dacă nu, exemplificați prin ce diferă.
15. Ce înseamnă „negociere” atunci când se discută protocolele de rețea? Dați un exemplu.
16. În fig. 1-19 este prezentat un serviciu. Există și servicii implicate în această figură? Dacă da, unde? Dacă nu, de ce nu?
17. În unele rețele, nivelul legătură de date tratează erorile de transmisie, solicitând retransmiterea cadrelor deteriorate. Dacă probabilitatea de a se strica un cadru este  $p$ , care este numărul mediu de transmisii necesare pentru a trimite un cadru, în cazul în care confirmările nu se pierd niciodată?
18. Care dintre nivelurile OSI se ocupă de fiecare din următoarele sarcini:  
a) Descompunerea fluxului de biți transmiși în cadre.  
b) Determinarea traseului care trebuie folosit în subrețea.  
c) TDPU-urile încapsulează pachete sau invers? Discuție.
19. Dacă unitățile de date schimbă la nivelul legătură de date se numesc cadre și unitățile de date schimbă la nivelul rețea se numesc pachete, pachetele încapsulează cadre sau cadrele încapsulează pachete? Explicați răspunsul dat.
20. Un sistem are o ierarhie de protocole organizată pe  $n$  niveluri. Aplicațiile generează mesaje de lungime  $M$  octeți. La fiecare nivel este adăugat un antet de  $h$  octeți. Ce fracțiune din lățimea benzii rețelei este ocupată de antete?
21. Prezentați două aspecte comune modelului de referință OSI și modelului de referință TCP/IP. Prezentați apoi două aspecte prin care modelele diferă.
22. Care este principala deosebire între TCP și UDP?
23. Subrețeaua din fig. 1-25(b) a fost proiectată pentru a putea rezista unui război nuclear. Câte bombe ar fi necesare pentru a parta nodurile sale în două seturi complet deconectate? Presupuneți că orice bombă distrugă un nod și toate legăturile conectate cu el.

24. Internet-ul își dublează dimensiunea o dată la aproximativ 18 luni. Deși nimeni nu știe cu siguranță, se estimează numărul gazdelor la 100 de milioane în 2001. Folosiți aceste date pentru a calcula numărul de gazde Internet prevăzut pentru anul 2010. Puteți crede acest scenariu? Explicați de ce da sau de ce nu.
25. La transferul unui fișier între două calculatoare există (cel puțin) două strategii de confirmare. Conform primei strategii, fișierul este descompus în pachete care sunt confirmate individual de către server, dar transferul de fișiere pe ansamblu nu este confirmat. În a doua strategie, pachetele nu sunt confirmate individual, dar la sfârșit este confirmat întregul fișier. Discutați aceste două abordări.
26. De ce folosește ATM-ul celule mici, de lungime fixă?
27. Cât de lung era un bit în standardul original 802.3 măsurat în metri? Folosiți viteza de transmisie de 10 Mbps și presupuneți că viteza de transmisie prin cablu coaxial este de 2/3 din viteza de propagare a luminii în vid.
28. O imagine are 1024 x 768 pixeli și reține câte 3 octeți pentru fiecare pixel. Presupuneți că imaginea este necomprimată. Cât durează transmisia ei pe un canal de modem de 56 Kbps? Dar printr-un modem de cablu de 1 Mbps? Dar prin Ethernet la 10 Mbps? Dar prin Ethernet la 100 Mbps?
29. Ethernet-ul și rețelele fără fir au unele asemănări și deosebiri. O proprietate a Ethernet-ului este aceea că un singur cadru poate fi transmis la un moment dat pe mediu. Are și 802.11 această proprietate? Discutați răspunsul dat.
30. Rețelele fără fir sunt ușor de instalat, ceea ce le face mai ieftine, deoarece de cele mai mult ori operația de instalare depășește semnificativ costul echipamentelor. Totuși, aceste rețele au și unele dezavantaje. Numeți două dintre ele.
31. Prezentați două avantaje și două dezavantaje ale existenței standardelor internaționale pentru protocoalele de rețea.
32. Atunci când un sistem dispune de o parte permanentă și de o parte detașabilă, de exemplu un cititor de CD-uri și un CD-ROM, este important ca sistemul să fie standardizat, astfel ca diferite firme să poată realiza atât părțile permanente cât și cele mobile și ca ele să se potrivească fără probleme. Dați trei exemple din afara industriei de calculatoare unde există astfel de standarde internaționale. Indicați apoi trei domenii din afara industriei de calculatoare unde nu există astfel de standarde.
33. Alcătuiți o listă de activități pe care le faceți zilnic și în care sunt implicate rețele de calculatoare. Cum ar fi viața voastră alterată dacă aceste rețele ar fi deconectate la un moment dat?
34. Descoperiți ce rețele sunt utilizate în școală sau la locul de muncă. Descrieți tipurile de rețele, topologii și metodele de comutare folosite acolo.
35. Programul *ping* vă permite să trimiteți un pachet de test la o locație dată pentru a vedea cât de mult durează până când acesta ajunge acolo și înapoi. Încercați să folosiți *ping* pentru a vedea cât de mult durează transferul pachetului între locul în care vă găsiți și alte câteva locuri cunoscute.

cute. Din aceste date, calculați timpul de tranzit într-o sigură direcție în funcție de distanță. Este bine să folosiți universitățile deoarece locațiile serverelor lor sunt cunoscute foarte bine. De exemplu, *berkley.edu* este în Berkley, California, *mit.edu* este în Cambridge, Massachusetts, *vu.nl* este în Amsterdam, Olanda, *www.usyd.edu.au* este în Sydney, Australia și *www.uct.ac.za* este în Cape Town, Africa de Sud.

36. Vizitați situl Web al IETF, [www.ietf.org](http://www.ietf.org) pentru a vedea ce mai fac. Alegeti un proiect care vă place și scrieți un raport de jumătate de pagină despre problemă și despre o soluție propusă.
37. Standardizarea este foarte importantă în lumea rețelelor. ITU și ISO sunt principalele organizații oficiale de standardizare. Vizitați siturile lor Web, [www.itu.org](http://www.itu.org) și [www.iso.org](http://www.iso.org), respectiv, și aflați despre munca lor de standardizare. Scrieți un scurt raport despre tipurile de lucruri pe care le-au standardizat.
38. Internet-ul este alcătuit dintr-un mare număr de rețele. Aranjarea lor determină topologia Internet-ului. O importantă cantitate de informații despre topologia Internet-ului este disponibilă online. Folosiți un motor de căutare pentru a afla mai multe despre acest subiect și scrieți un scurt raport care să rezume informațiile pe care le-ați găsit.

# 5

## NIVELUL REȚEA

Nivelul rețea are ca sarcină preluarea pachetelor de la sursă și transferul lor către destinație. Ajungerea la destinație poate necesita mai multe salturi prin rutere intermediare de-a lungul drumului. Această funcție contrastează clar cu cea a nivelului legătură de date, care avea scopul mult mai modest de a transfera cadre de la un capăt al unui fir la celălalt. Astfel nivelul rețea este cel mai scăzut nivel care se ocupă de transmisii capăt la capăt.

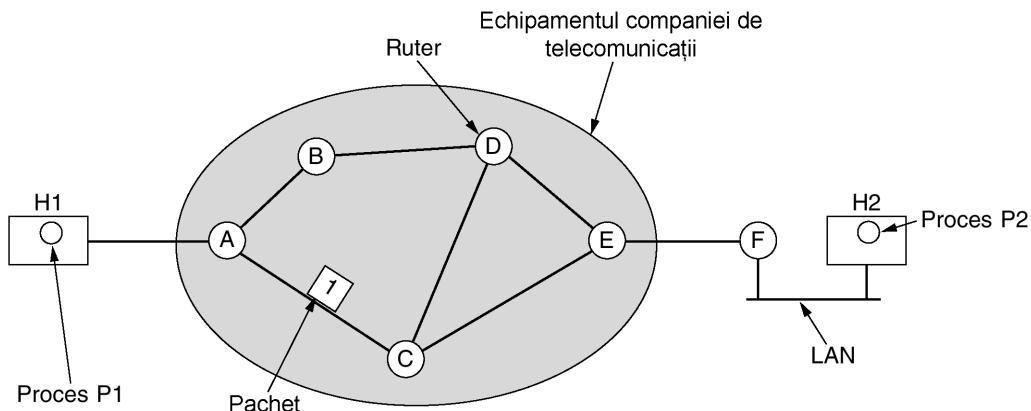
Pentru realizarea scopurilor propuse, nivelul rețea trebuie să cunoască topologia subrețelei de comunicație (de exemplu mulțimea tuturor ruterelor) și să aleagă calea cea mai potrivită prin aceasta. De asemenea trebuie să aleagă căile de urmat astfel, încât să nu încarce excesiv unele legături de comunicație sau rutere în timp ce altele sunt inactive. În fine, când sursa și destinația fac parte din rețele diferite, apar probleme noi. Este sarcina nivelului rețea să se ocupe de ele. În acest capitol vom studia toate aceste aspecte și le vom exemplifica, în primul rând folosind Internetul și protocolul lui la nivelul rețea, IP, cu toate că vom vorbi și despre rețele fără fir.

### 5.1 CERINȚELE DE PROIECTARE ALE NIVELULUI REȚEA

Vom prezenta, în continuare, o introducere a cerințelor pe care proiectantul nivelului rețea trebuie să le rezolve. Acestea includ serviciile furnizate nivelului transport și proiectarea internă a subrețelei.

### 5.1.1 Comutare de pachete de tip Memorează-și-Retrasmite (Store-and-Forward)

Dar înainte de a începe explicarea detaliilor nivelului rețea, merită probabil să reinitializăm contextul în care operează protoalele de la nivelul rețea. Acest context este prezentat în fig. 5-1. Componentele majore ale sistemului sunt echipamentul companiei de telecomunicații (rutere conectate prin linii de transmisie), prezentat în interiorul ovalului umbrit, și echipamentul clientului, prezentat în afara ovalului. Gazda  $H1$  este conectată direct la unul dintre ruterele companiei de telecomunicații,  $A$ , printr-o linie închiriată. În contrast,  $H2$  este într-o rețea LAN cu un ruter,  $F$ , deținut și operat de către client. Acest ruter are, deasemeni, și o linie închiriată către echipamentul companiei de telecomunicații. Am prezentat  $F$  ca fiind în afara ovalului, deoarece nu aparține companiei de telecomunicații, dar în termeni de construcție, software și protoale, probabil că nu diferă față de ruterele acesteia. Este discutabil dacă aparține subretelei, dar în contextul acestui capitol ruterele din localul clientului sunt considerate parte a subretelei deoarece rulează aceeași algoritmi ca și ruterele companiei de telecomunicații (și aici principala noastră preocupare sunt algoritmii).



**Fig. 5-1.** Cadrul protoalelor nivelului rețea.

Acest echipament este folosit după cum urmează. O gazdă care are de transmis un pachet îl transmite celui mai apropiat ruter, fie în aceeași rețea LAN, fie printr-o legătură punct la punct cu compania de telecomunicații. Pachetul este memorat acolo până ajunge integral, astfel încât să poată fi verificată suma de control. Apoi este trimis mai departe către următorul ruter de pe traseu, până ajunge la gazda destinație, unde este livrat. Acest mecanism reprezintă comutarea de pachete de tip memorează-și-retransmite, aşa cum am văzut în capitolele anterioare.

### 5.1.2 Servicii furnizate nivelului transport

Nivelul rețea furnizează servicii nivelului transport la interfața dintre cele două niveluri. O întrebare importantă este ce fel de servicii furnizează nivelul rețea nivelului transport. Serviciile nivelului rețea au fost proiectate având în vedere următoarele scopuri:

1. Serviciile trebuie să fie independente de tehnologia ruterului.
2. Nivelul transport trebuie să fie independent de numărul, tipul și topologia ruterelor existente.

3. Adresele de rețea disponibile la nivelul transport trebuie să folosească o schemă de nume-rotare uniformă, chiar în cadrul rețelelor LAN și WAN.

Obiectivele fiind stabilite, proiectantul nivelului rețea are o mare libertate în a scrie specificațiile detaliate ale serviciilor oferite nivelului transport. Această libertate degeneră adesea într-o aprigă bătălie între două tabere opuse. Problema centrală a discuției este dacă nivelul rețea trebuie să furnizeze servicii orientate pe conexiune sau servicii neorientate pe conexiune.

O tabără (reprezentată de comunitatea Internet) afirmă că scopul ruterului este de a transfera pachete și nimic mai mult. În viziunea lor (bazată pe experiența a aproape 30 de ani de exploatare a unei rețele de calculatoare în funcțiu), subrețea este inherentă și nesigură, indiferent cum ar fi proiectată. De aceea calculatoarele gazdă trebuie să accepte faptul că rețea este nesigură și să facă controlul erorilor (i.e., detectia și corecția erorii) și controlul fluxului ele însese.

Acest punct de vedere duce rapid la concluzia că serviciul rețea trebuie să fie neorientat pe conexiune, cu două primitive SEND PACKET și RECEIVE PACKET și cu foarte puțin în plus. În particular, nu trebuie făcută nici o operație pentru controlul ordinii sau fluxului pachetelor pentru că oricum calculatorul gazdă va face acest lucru, și, de obicei, dublarea acestor operațiuni aduce un câștig nesemnificativ. În continuare, fiecare pachet va trebui să poarte întreaga adresă de destinație, pentru că fiecare pachet este independent de pachetele predecesoare, dacă acestea există.

Cealaltă tabără (reprezentată de companiile de telefoane) afirmă că subrețea trebuie să asigure un serviciu orientat pe conexiune sigur. Ei susțin că 100 de ani de experiență cu sistemul telefonic mondial reprezintă un ghid excelent. În această perspectivă, calitatea serviciului este elementul dominant, și într-o subrețea fără conexiuni, calitatea serviciului este dificil de obținut, în special pentru trafic în timp real cum ar fi voce și imagine.

ACESTE DOUĂ TABERE sunt cel mai bine exemplificate de Internet și rețele ATM. Rețea Internet oferă un serviciu la nivelul rețea neorientat pe conexiune; rețelele ATM oferă un serviciu la nivelul rețea orientat pe conexiune. Totuși, este interesant de notat că cu cât garantarea calității serviciului devine din ce în ce mai importantă, Internetul evoluează. În particular, începe să dobândească proprietăți asociate normal cu serviciile orientate conexiune, aşa cum vom vedea mai târziu. De fapt, ne-am făcut o părere despre această evoluție în timpul studiului despre rețele VLAN în Cap. 4.

### 5.1.3 Implementarea serviciului neorientat pe conexiune

După ce am văzut cele două clase de servicii pe care nivelul rețea le furnizează utilizatorilor săi, este momentul să vedem funcționarea internă a acestui nivel. Sunt posibile două organizări diferite, în funcție de tipul serviciului oferit. Dacă este oferit un serviciu neorientat pe conexiune, atunci pachetele sunt trimise în subrețea individual și dirijate independent de celelalte. Nu este necesară nici o inițializare prealabilă. În acest context, pachetele sunt numite frecvent **datagrame** (datagrams) (prin analogie cu telegramme), iar subrețea este numită **subrețea datagramă** (datagram subnet). Dacă este folosit serviciul orientat conexiune, atunci, înainte de a trimite pachete de date, trebuie stabilită o cale de la ruterul sursă la ruterul destinație. Această conexiune este numită **VC (virtual circuit, circuit virtual)**, prin analogie cu circuitele fizice care se stabilesc în sistemul telefonic, iar subrețea este numită **subrețea cu circuite virtuale (virtual-circuit subnet)**. În această secțiune vom studia subrețele datagramă; în următoarea secțiune vom studia subrețelele cu circuite virtuale.

Să vedem cum funcționează o subrețea datagramă. Să presupunem că procesul *P1* din fig. 5-2 are un mesaj lung pentru procesul *P2*. El transmite mesajul nivelului transport, cu instrucțiunile de livrare către procesul *P2* aflat pe calculatorul gazdă *H2*. Codul nivelului transport rulează pe calculatorul

gazdă  $H1$ , de obicei în cadrul sistemului de operare. Acesta inserează la începutul mesajului un antet corespunzător nivelului transport și transferă rezultatul nivelului rețea, probabil o altă procedură din cadrul sistemului de operare.

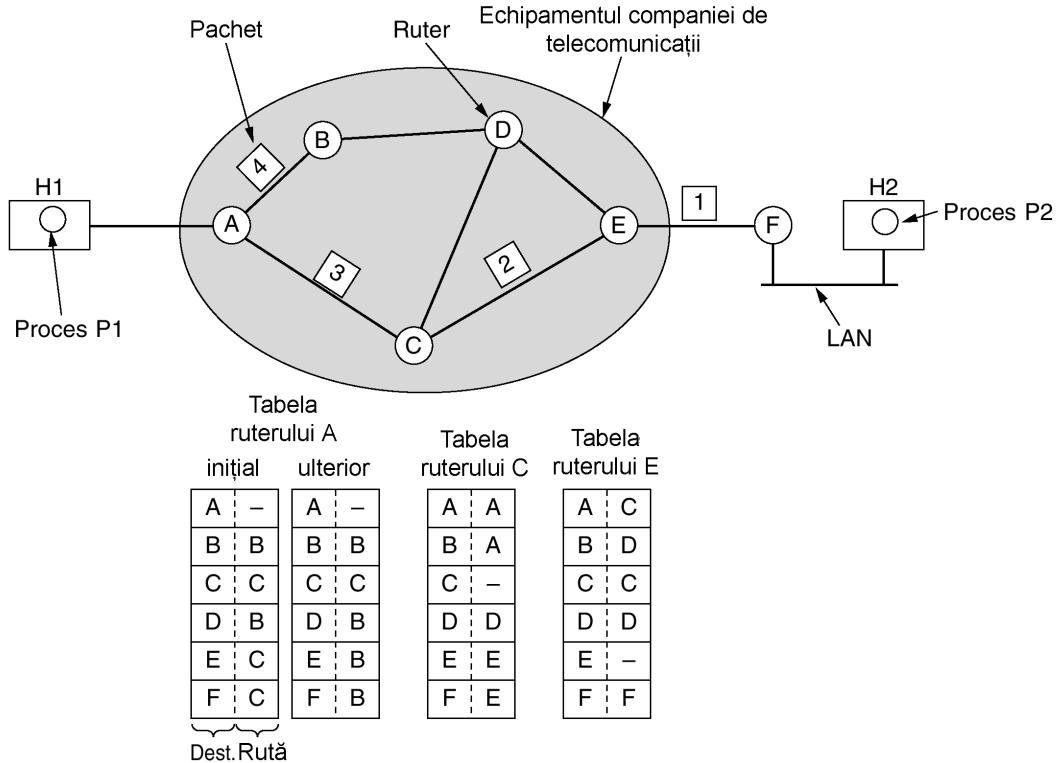


Fig. 5-2. Dirijarea într-o subrețea datagramă.

Să presupunem că mesajul este de patru ori mai lung decât dimensiunea maximă a unui pachet, aşa că nivelul rețea trebuie să îl spargă în patru pachete, 1, 2, 3, și 4 și să le trimită pe fiecare în parte ruterului  $A$ , folosind un protocol punct-la-punct, de exemplu, PPP. Din acest punct controlul este preluat de compania de telecomunicații. Fiecare ruter are o tabelă internă care îi spune unde să trimite pachete pentru fiecare destinație posibilă. Fiecare intrare în tabelă este o pereche compusă din destinație și linia de ieșire folosită pentru acea destinație. Pot fi folosite doar linii conectate direct. De exemplu, în fig. 5-2,  $A$  are doar două linii de ieșire – către  $B$  și  $C$  – astfel că fiecare pachet ce vine trebuie trimis către unul dintre aceste rutere, chiar dacă ultima destinație este alt ruter. Tabela de rutare inițială a lui  $A$  este prezentată în figură sub eticheta „inițial”.

Cum au ajuns la  $A$ , pachetele 1, 2 și 3 au fost memorate pentru scurt timp (pentru verificarea sumei de control). Apoi fiecare a fost trimis mai departe către  $C$  conform tabelei lui  $A$ . Pachetul 1 a fost apoi trimis mai departe către  $E$  și apoi către  $F$ . Când a ajuns la  $F$ , a fost încapsulat într-un cadru al nivelului legătură de date și trimis către calculatorul gazdă  $H2$  prin rețeaua LAN.

Totuși, ceva diferit s-a întâmplat cu pachetul 4. Când a ajuns la  $A$  a fost trimis către ruterul  $B$ , chiar dacă și el este destinat tot lui  $F$ . Dintr-un motiv oarecare,  $A$  a decis să trimită pachetul 4 pe o rută diferită de cea urmată de primele trei. Poate că aflat despre o congestie undeva pe calea ACE

și și-a actualizat tabela de rutare, aşa cum apare sub eticheta „mai târziu”. Algoritmul ce administrează tabelele și ia deciziile de rutare se numește **algoritm de rutare** (routing algorithm). Algoritmii de rutare sunt unele dintre principalele elemente pe care le vom studia în acest capitol.

#### 5.1.4 Implementarea serviciilor orientate pe conexiune

Pentru serviciile orientate conexiune, avem nevoie de o subrețea cu circuite virtuale. Să vedem cum funcționează aceasta. Ideea care se stă la baza circuitelor virtuale este evitarea alegerii unei noi căi (rute) pentru fiecare pachet trimis, ca în fig. 5-2. În schimb, atunci când se stabilește o conexiune, se alege o cale între mașina sursă și mașina destinație, ca parte componentă a inițializării conexiunii și aceasta este memorată în tabelele ruterelor. Acea cale este folosită pentru tot traficul de pe conexiune, exact în același mod în care funcționează sistemul telefonic. Atunci când conexiunea este eliberată, este închis și circuitul virtual. În cazul serviciilor orientate conexiune, fiecare pachet poartă un identificator care spune cărui circuit virtual îi aparține.

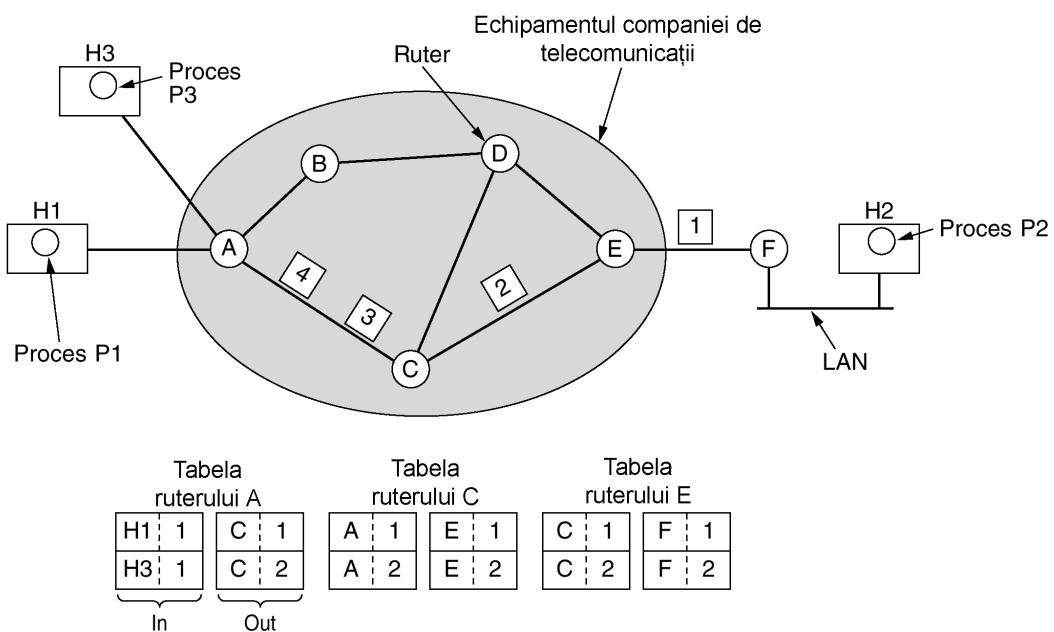


Fig. 5-3. Dirijare în cadrul unei subrețele cu circuite virtuale.

De exemplu, să considerăm situația din fig. 5-3. Aici calculatorul gazdă *H1* a stabilit conexiunea 1 cu calculatorul gazdă *H2*. Aceasta este memorată ca prima intrare în fiecare tabelă de rutare. Prima linie a tabelei lui *A* spune că dacă un pachet purtând identificatorul de conexiune 1 vine de la *H1*, atunci trebuie trimis către ruterul *C*, dându-i-se identificatorul de conexiune 1. Similar, prima intrare a lui *C* dirijează pachetul către *E*, tot cu identificatorul de conexiune 1.

Acum să vedem ce se întâmplă dacă *H3* vrea, de asemenea, să stabilească o conexiune cu *H2*. Alege identificatorul de conexiune 1 (deoarece inițializează conexiunea și aceasta este singura conexiune) și indică subrețelei să stabilească circuitul virtual. Aceasta conduce la a doua linie din tabele. Observați că apare un conflict deoarece deși *A* poate distinge ușor pachetele conexiunii 1 de la *H1*

de pachetele conexiunii 1 de la  $H_3$ ,  $C$  nu poate face asta. Din acest motiv,  $A$  asociază un identificator de conexiune diferit pentru traficul de ieșire al celei de a doua conexiuni. Pentru evitarea conflictelor de acest gen ruterele trebuie să poată înlocui identificatorii de conexiune în pachetele care pleacă. În unele contexte, aceasta se numește comutarea etichetelor (label switching).

### 5.1.5 Comparație între subrețele cu circuite virtuale și subrețele datagramă

Atât circuitele virtuale cât și datagramele au suporterii și oponenți. Vom încerca acum să rezumăm argumentele ambelor tabere. Principalele aspecte sunt prezentate în fig. 5-4, deși cei extrem de riguroși ar putea probabil găsi un contraexemplu pentru toate cele descrise în această figură.

Problema	Subrețea datagramă	Subrețea cu circuite virtuale (CV)
Stabilirea circuitului	Nu este necesară	Obligatorie
Adresare	Fiecare pachet conține adresa completă pentru sursă și destinație	Fiecare pachet conține un număr mic de CV
Informații de stare	Ruterele nu păstrează informații despre conexiuni	Fiecare CV necesită spațiu pentru tabela ruterului per conexiune
Dirijare	Fiecare pachet este dirijat independent	Calea este stabilită la inițierea CV; toate pachetele o urmează
Efectul defectării ruterului	Nici unul, cu excepția pachetelor pierdute în timpul defectării	Toate circuitele virtuale care trec prin ruterul defect sunt terminate
Calitatea serviciului	Dificil	Simplu, dacă pentru fiecare CV pot fi alocate în avans suficiente resurse
Controlul congestiei	Dificil	Simplu, dacă pentru fiecare CV pot fi alocate în avans suficiente resurse

Fig. 5-4. Comparație între subrețele datagramă și subrețele cu circuite virtuale.

În interiorul subrețelei există situații în care trebuie să se aleagă între facilități antagoniste specifice fie circuitelor virtuale, fie datagramelor. Un astfel de compromis este acela între spațiul de memorie al ruterului și lățimea de bandă. Circuitele virtuale permit pachetelor să conțină numere de circuite în locul unor adrese complete. Dacă pachetul tinde să fie foarte mic, atunci existența unei adrese complete în fiecare pachet poate reprezenta o supraîncărcare (overhead) importantă și deci o irosire a lățimii de bandă. Prețul plătit pentru folosirea internă a circuitelor virtuale este spațiul necesar păstrării tabelei în ruter. Soluția mai ieftină este determinată de raportul între costul circuitelor de comunicație și cel al memoriei ruterului.

Alt compromis este cel între timpul necesar stabilirii circuitului și timpul de analiză a adresei. Folosirea circuitelor virtuale presupune existența unei faze initiale de stabilire a căii, care cere timp și consumă resurse. Oricum, este ușor să ne imaginăm ce se întâmplă cu un pachet de date într-o subrețea bazată pe circuite virtuale: ruterul folosește numărul circuitului ca un index într-o tabelă pentru a afla unde merge pachetul. Într-o rețea bazată pe datagrame, pentru a găsi intrarea corespunzătoare destinației se folosește o procedură de căutare mult mai complicată.

O altă problemă este cea a dimensiunii spațiului necesar pentru tabela din memoria ruterului. O subrețea datagramă necesită o intrare pentru fiecare destinație posibilă, în timp ce o rețea cu circuite virtuale necesită o intrare pentru fiecare circuit virtual. Totuși, acest avantaj este relativ iluzoriu deoarece și pachetele de initializare a conexiunii trebuie rutate, iar ele folosesc adresele destinație, la fel ca și datagramele.

Circuitele virtuale au unele avantaje în garantarea calității serviciului și evitarea congestiunii subretelei, deoarece resursele (de exemplu zone tampon, lărgime de bandă și cicluri CPU) pot fi rezervate în avans, atunci când se stabilește conexiunea. La sosirea pachetelor, lățimea de bandă necesară și capacitatea ruterului vor fi deja pregătite. Pentru o subretea bazată pe datagrame, evitarea congestiunii este mult mai dificilă.

Pentru sistemele de prelucrare a tranzacțiilor (de exemplu apelurile magazinelor pentru a verifica cumpărături realizate cu cărți de credit) overhead-ul implicat de stabilirea și eliberarea unui circuit virtual poate reduce cu ușurință utilitatea circuitului. Dacă majoritatea traficului este de acest tip, folosirea internă a circuitelor virtuale în cadrul subretelei nu prea are sens. Pe de altă parte, ar putea fi de folos circuite virtuale permanente, stabilite manual și care să dureze luni sau chiar ani.

Circuitele virtuale au o problemă de vulnerabilitate. Dacă un ruter se defectează și își pierde conținutul memoriei, atunci toate circuitele virtuale care treceau prin el sunt suprimate, chiar dacă aceasta își revine după o secundă. Prin contrast, dacă se defectează un ruter bazat pe datagrame vor fi afectați doar acei utilizatori care aveau pachete memorate temporar în cozile de așteptare ale ruterului și este posibil ca numărul lor să fie și mai mic, în funcție de câte pachete au fost deja confirmate. Pierderea liniei de comunicație este fatală pentru circuitele virtuale care o folosesc, însă poate fi ușor compensată dacă se folosesc datagrame. De asemenea, datagramele permit ruterului să echilibreze traficul prin subretea, deoarece căile pot fi modificate parțial în cursul unei secvențe lungi de pachete transmise.

## 5.2 ALGORITMI DE DIRIJARE

Principala funcție a nivelului rețea este dirijarea pachetelor de la mașina sursă către mașina destinație. În majoritatea subretelelor pachetele vor face salturi multiple pentru a ajunge la destinație. Singura excepție remarcabilă o reprezintă rețelele cu difuzare, dar chiar și aici dirijarea este importantă, atunci când sursa și destinația nu sunt în același rețea. Algoritmii care aleg calea și structurile de date folosite de acestia reprezintă un domeniu important al proiectării nivelului rețea.

**Algoritmul de dirijare (routing algorithm)** este acea parte a software-ului nivelului rețea care răspunde de alegerea liniei de ieșire pe care trebuie trimis un pachet recepționat. Dacă subretea folosește intern datagrame, această decizie trebuie luată din nou pentru fiecare pachet recepționat, deoarece este posibil ca cea mai bună rută să se fi modificat între timp. Dacă subretea folosește circuite virtuale, deciziile de dirijare sunt luate doar la inițializarea unui nou circuit virtual. După aceea pachetele de date vor urma doar calea stabilită anterior. Acest ultim caz este numit uneori **dirijare de sesiune (session routing)**, deoarece calea rămâne în funcțiune pentru o întreagă sesiune utilizator (de exemplu o sesiune de conectare de la un terminal -login- sau un transfer de fișiere).

Uneori este util să se facă distincția între dirijare, care înseamnă alegerea căii care va fi folosită, și retransmitere, care se referă la ceea ce se întâmplă atunci când sosesc un pachet. Se poate spune despre un ruter că rulează intern două procese. Unul dintre ele preia fiecare pachet care sosesc, căutând în tabela de dirijare linia de ieșire folosită pentru el. Acesta este procesul de **retransmitere (forwarding)**. Celălalt proces se ocupă de completarea și actualizarea talelei de rutare. Aici algoritmul intervine de dirijare.

Unele protocoale inter-rețea extind această metodă și consideră întreaga transmisie pe un circuit virtual ca fiind un pachet gigant, așa încât fiecare fragment conține numărul absolut de octet al primului octet din fragment.

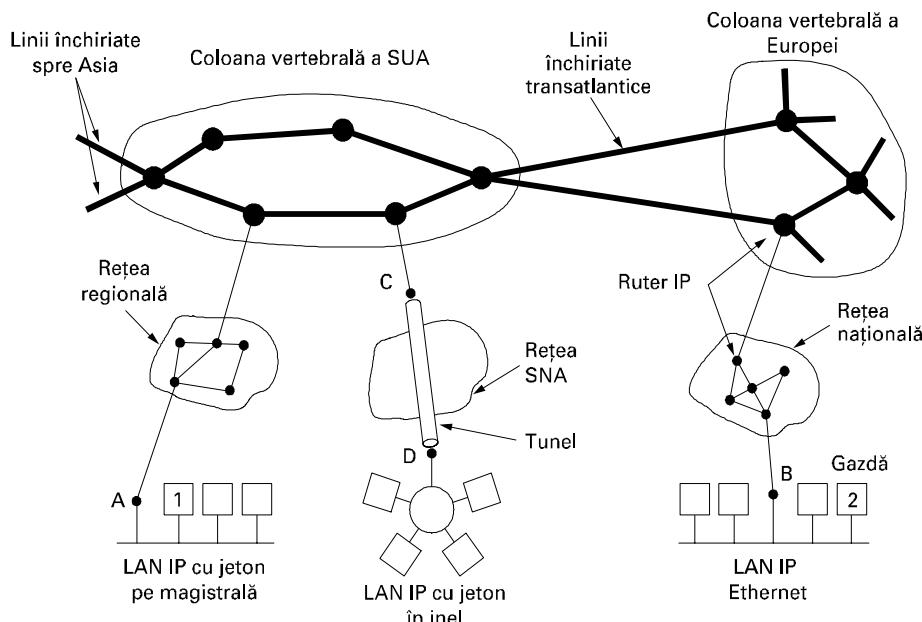
## 5.6 NIVELUL REȚEA ÎN INTERNET

Înainte de a intra în detaliile nivelului rețea din Internet merită studiate principiile care au ghidat proiectarea lui în trecut și l-au făcut succesul care este astăzi. În ziua de azi, în prea multe cazuri oamenii par să le fi uitat. Aceste principii sunt enumerate și discutate în RFC 1958, care merită citit (și ar trebui să fie obligatoriu pentru toți proiectanții de protocoale – cu un examen final la sfârșit). Acest RFC se bazează în mare parte pe idei care se găsesc în (Clark, 19 și Saltzer, 1984). Vom rezuma ceea ce considerăm a fi cele 10 principii (de la cel mai important la cel mai puțin important).

1. **Fiți siguri că funcționează.** Nu finalizați proiectarea sau standardul până când mai multe prototipuri nu au comunicat cu succes unele cu altele. De prea multe ori proiectanții întâi scriu un standard de 1000 de pagini, primesc aprobarea pentru el și apoi descoperă că acest standard este eronat și nu funcționează. Apoi scriu versiunea 1.1 a standardului. Nu aşa se face.
2. **Menține-l simplu.** Când există îndoieri, folosiți soluția mai simplă. William de Occam a enunțat acest principiu (briciul lui Occam) în secolul al 14-lea. Reformulat în termeni moderni: împotriviți-vă caracteristicilor suplimentare. Dacă o caracteristică nu este absolut necesară, nu o includeți, mai ales dacă același efect poate fi obținut prin combinarea altor caracteristici.
3. **Faceți alegeri clare.** Dacă există mai multe moduri de a realiza același lucru, alegeti unul. A avea două sau mai multe moduri de a rezolva un lucru înseamnă să se caute neacuzul cu lumânarea. Standardele conțin de obicei multe opțiuni sau moduri sau parametri pentru că anumite grupări importante susțin că modul lor este cel mai bun. Proiectanții ar trebui să reziste la această tendință. Spuneți nu.
4. **Exploatați modularitatea.** Acest principiu conduce direct la ideea de a avea stive de protocoale, fiecare nivel fiind independent de toate celelalte. În acest mod, dacă circumstanțele cer ca un modul sau nivel să fie schimbat, celelalte nu vor fi afectate.
5. **Așteptați-vă la medii eterogene.** În orice rețea mare vor apărea diferite tipuri de hardware, posibilități de transmisie și aplicații. Pentru a le trata pe toate, proiectarea rețelei trebuie să fie simplă, generală și flexibilă.
6. **Evitați opțiuni sau parametri statici.** Dacă un parametru nu poate fi evitat (de exemplu dimensiunea maximă a unui pachet), este mai bine ca transmițătorul și receptorul să negocieze o valoare decât să se definească valori fixe.
7. **Căutați o proiectare bună; nu este necesar să fie perfectă.** Deseori proiectanții au un proiect bun care nu poate trata un caz mai ciudat. Decât să strice tot proiectul, proiectanții ar trebui să-l păstreze și să transfere povara rezolvării aceluia caz celor cu cerințe ciudate.

8. **Fiți stricți atunci când trimiteți și toleranți atunci când recepționați.** Cu alte cuvinte, trimiteți numai pachete care sunt în deplină conformitate cu standardul, dar așteptați-vă ca pachetele recepționate să nu fie în deplină conformitate cu standardul și încercați să le folosiți.
9. **Gândiți-vă la scalabilitate.** Dacă sistemul trebuie să suporte milioane de gazde și efectiv miiliarde de utilizatori, nu se poate accepta nici un fel de bază de date centralizată, iar încărcarea trebuie distribuită cât mai uniform posibil folosind resursele disponibile.
10. **Luați în considerare performanțele și costurile.** Dacă o rețea are performanțe slabe sau prețuri exorbitante, nu o va folosi nimeni.

Să lăsăm acum principiile generale și să începem să studiem detaliile nivelului rețea din Internet. La nivelul rețea, Internet-ul poate fi văzut ca o colecție de subrețele sau **sisteme autonome** (eng.: **Autonomous Systems - AS**) care sunt interconectate. Nu există o structură reală, dar există câteva coloane vertebrale majore. Acestea sunt construite din linii de înaltă capacitate și rutere rapide. La coloanele vertebrale sunt atașate rețelele regionale (de nivel mediu), iar la aceste rețele regionale sunt atașate LAN-urile din multe universități, companii și furnizori de servicii Internet. O schiță a acestei organizări evazi-ierarhice este dată în fig. 5-52.



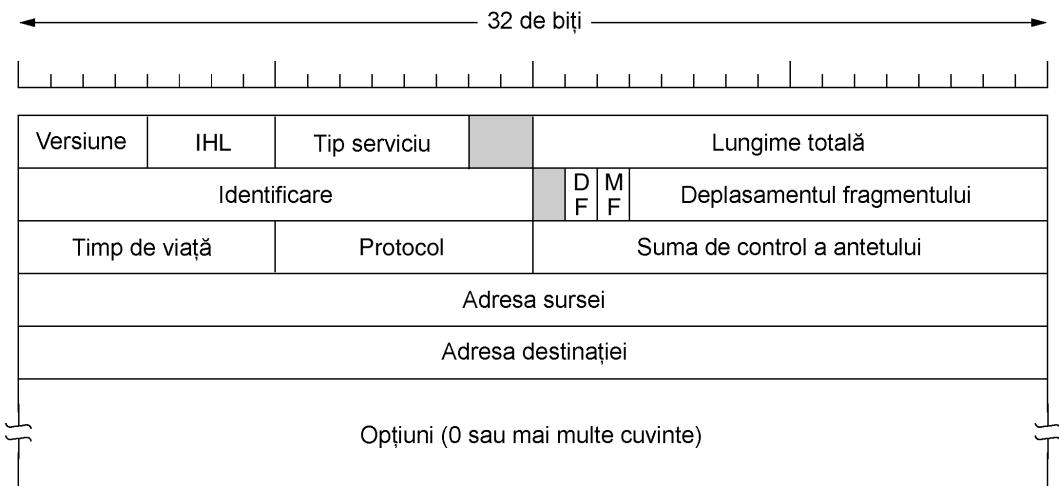
**Fig. 5-52.** Internet-ul este o colecție de multe rețele interconectate.

Lantul care ține Internet-ul la un loc este protocolul de nivel rețea, numit **IP (Internet Protocol - protocolul Internet)**. Spre deosebire de protocolele mai vechi de nivel rețea, acesta a fost proiectat de la început având în vedere interconectarea rețelelor. O metodă bună de a gândi nivelul rețea este aceasta. Sarcina lui este de a oferi cel mai bun mod posibil (adică negarantat) de a transporta datagramme de la sursă la destinație, fără a ține seama dacă aceste mașini sunt sau nu în aceeași rețea sau dacă între ele există sau nu alte rețele.

Comunicația în Internet funcționează după cum urmează. Nivelul transport preia șiruri de date și le sparge în datagrame. În teorie, datagramele pot avea fiecare până la 64 KB, dar în practică, ele nu depășesc 1500 octeți (pentru a intra într-un cadru Ethernet). Fiecare datagramă este transmisă prin Internet, fiind eventual fragmentată în unități mai mici pe drum. Când toate bucățile ajung în sfârșit la mașina destinație, ele sunt reasamblate de nivelul rețea în datagrama originală. Datagrama este apoi pasată nivelului transport, care o inserează în sirul de intrare al procesului receptor. Dupa cum se poate vedea în fig. 5-52 un pachet transmis de gazda 1 trebuie să traverseze șase rețele pentru a ajunge la gazda 2. În practică se trece de obicei prin mult mai mult decât șase rețele.

### 5.6.1 Protocolul IP

Un loc potrivit pentru a porni studiul nostru despre nivelul rețea în Internet este însuși formatul datagramelor IP. O datagramă IP constă dintr-o parte de antet și o parte de text. Antetul are o parte fixă de 20 de octeți și o parte optională cu lungime variabilă. Formatul antetului este prezentat în fig. 5-53. El este transmis în ordinea *big endian* (cel mai semnificativ primul): de la stânga la dreapta, începând cu bitul cel mai semnificativ al câmpului *Versiune*. (Procesorul SPARC este de tip *big endian*; Pentium este de tip *little endian* - cel mai puțin semnificativ primul). Pe mașinile de tip *little endian*, este necesară o conversie prin program atât la transmisie cât și la recepție.



**Fig. 5-53.** Antetul IPv4 (protocolul Internet).

Câmpul *Versiune* memorează cărei versiuni de protocol îi aparține datagramă. Prin includerea unui câmp versiune în fiecare datagramă, devine posibil ca tranzitia între versiuni să dureze ani de zile, cu unele mașini rulând vechea versiune, iar altele rulând-o pe cea nouă. La ora actuală are loc o tranzitie de la IPv4 la IPv6, care deja durează de câțiva ani și în nici un caz nu s-a apropiat de final (Durand, 2001; Wilijakka, 2002; and Waddington and Chang 2002). Anumite persoane consideră chiar că nu se va întâmpla niciodată (Weiser, 2001). Referitor la numerotare, IPv5 a fost un protocol experimental de flux în timp real care nu a fost folosit pe scară largă.

Din moment ce lungimea antetului nu este constantă, un câmp din antet, *IHL*, este pus la dispoziție pentru a spune cât de lung este antetul, în cuvinte de 32 de octeți. Valoarea minimă este 5, care se aplică atunci când nu sunt prezente opțiuni. Valoarea maximă a acestui câmp de 4 biți este 15,

cea ce limitează antetul la 60 de octeți și, astfel, câmpul de opțiuni la 40 de octeți. Pentru unele opțiuni, cum ar fi cea care înregistrează calea pe care a mers un pachet, 40 de octeți sunt mult prea puțini, făcând această opțiune nefolositoare.

Câmpul *Tip serviciu* este unul dintre puținele câmpuri care și-a schimbat sensul (oarecum) de-a lungul anilor. A fost și este în continuare menit să diferențieze diferitele clase de servicii. Sunt posibile diferite combinații de fiabilitate și viteză. Pentru vocea digitizată, livrarea rapidă are prioritate față de transmisia corectă. Pentru transferul de fișiere, transmisia fără erori este mai importantă decât transmisia rapidă.

La început, câmpul de 6 biți conținea (de la stânga la dreapta), un câmp *Precedență* de trei biți și trei indicatori, *D*, *T* și *R*. Câmpul *Precedență* reprezintă prioritatea, de la 0 (normal) la 7 (pachet de control al rețelei). Cei trei biți indicatori permit calculatorului gazdă să specifice ce îl afectează cel mai mult din mulțimea {Delay (Întârziere), Throughput (Productivitate), Reliability (Fiabilitate)}. În teorie, aceste câmpuri permit ruterelor să aleagă între, de exemplu, o legătură prin satelit cu o productivitate mare și o întârziere mare sau o linie dedicată cu o productivitate scăzută și o întârziere mică. În practică, rutele curente ignoră adesea întregul câmp *Tip serviciu*.

Până la urmă, IETF a cedat și a modificat puțin câmpul pentru a-l adapta la servicii diferențiate. Șase dintre biți sunt folosiți pentru a indica căreia dintre clasele de servicii descrise mai devreme îi aparține pachetul. Aceste clase includ patru priorități de introducere în coadă, trei probabilități de respingere și clasele istorice.

*Lungimea totală* include totul din datagramă - atât antet cât și date. Lungimea maximă este de 65535 octeți. În prezent, această limită superioară este tolerabilă, dar în viitoarele rețele cu capacitați de gigaocete vor fi necesare datagrame mai mari. Câmpul *Identificare* este necesar pentru a permite gazdei destinație să determine cărei datagrame îi aparține un nou pachet primit. Toate fragmentele unei datagrame conțin aceeași valoare de *Identificare*.

Urmează un bit nefolosit și apoi două câmpuri de 1 bit. *DF* însemnă Don't Fragment (A nu se fragmenta). Aceasta este un ordin dat ruterelor ca să nu fragmenteze datagrama pentru că destinația nu este capabilă săreasambleze piesele la loc. De exemplu, când un calculator pornește, memoria sa ROM poate cere să își se trimită o imagine de memorie ca o singură datagramă. Prin marcarea datagramei cu bitul *DF*, emițătorul știe că aceasta va ajunge într-o singură bucată, chiar dacă aceasta înseamnă că datagrama trebuie să evite o rețea cu pachete mai mici pe calea cea mai bună și să aleagă o rută suboptimală. Toate mașinile trebuie să accepte fragmente de 576 octeți sau mai mici.

*MF* însemnă More Fragments (mai urmează fragmente). Toate fragmentele, cu excepția ultimului, au acest bit activat. El este necesar pentru a ști când au ajuns toate fragmentele unei datagrame.

*Deplasamentul fragmentului* spune unde este locul fragmentului curent în cadrul datagramei. Toate fragmentele dintr-o datagramă, cu excepția ultimului, trebuie să fie un multiplu de 8 octeți - unitatea de fragmentare elementară. Din moment ce sunt prevăzuți 13 biți, există un maxim de 8192 de fragmente pe datagramă, obținându-se o lungime maximă a datagramei de 65536 octeți, cu unul mai mult decât câmpul *Lungime totală*.

Câmpul *Timp de viață* este un contor folosit pentru a limita durata de viață a pachetelor. Este prevăzut să contorizeze timpul în secunde, permitând un timp maxim de viață de 255 secunde. El trebuie să fie decrementat la fiecare salt (hop - trecere dintr-o rețea în alta) și se presupune că este decrementat de mai multe ori când stă la coadă un timp îndelungat într-un ruter. În practică, el contorizează doar salturile. Când ajunge la valoarea zero, pachetul este eliminat și se trimite înapoi la gazda sursă un pachet de avertisment. Această facilitate previne hoinărea la infinit a datagramelor, ceea ce se poate întâmpla dacă tabelele de dirijare devin incoerente.

Când nivelul rețea a asamblat o datagramă completă, trebuie să știe ce să facă cu ea. Câmpul *Protocol* spune căruui proces de transport trebuie să o predea. TCP este o posibilitate, dar tot așa sunt și UDP și alte câteva. Numerotarea protocolelor este globală la nivelul întregului Internet. Protocolele și alte numere alocate erau anterior definite în RFC 1700, dar astăzi ele sunt conținute într-o bază de date on-line, care se găsește la adresa [www.iana.org](http://www.iana.org).

*Suma de control a antetului* verifică numai antetul. O astfel de sumă de control este utilă pentru detectarea erorilor generate de locații de memorie proaste din interiorul unui ruter. Algoritmul este de a aduna toate jumătățile de cuvinte, de 16 biți, atunci când acestea sosesc, folosind aritmetică în complement față de unu și păstrarea complementului față de unu al rezultatului. Pentru scopul acestui algoritm, se presupune că la sosirea *sumă de control a antetului* este zero. Acest algoritm este mai robust decât folosirea unei adunări normale. Observați că *suma de control a antetului* trebuie recalculată la fiecare salt, pentru că întotdeauna se schimbă cel puțin un câmp (câmpul *timp de viață*), dar se pot folosi trucuri pentru a accelera calculul.

*Adresa sursei* și *Adresa destinației* indică numărul de rețea și numărul de gazdă. Vom discuta adresele Internet în secțiunea următoare. Câmpul *Optiuni* a fost proiectat pentru a oferi un subterfugiu care să permită versiunilor viitoare ale protocolului să includă informații care nu sunt prezente în proiectul original, pentru a permite cercetătorilor să încerce noi idei și pentru a evita alocarea unor biți din antet pentru informații folosite rar. Optiunile sunt de lungime variabilă. Fiecare începe cu un cod de un octet care identifică opțiunea. Unele opțiuni sunt urmate de un câmp de un octet reprezentând lungimea opțiunii, urmat de unul sau mai mulți octeți de date. Câmpul *Optiuni* este completat până la un multiplu de 4 octeți. Inițial erau definite cinci opțiuni, aşa cum sunt listate în fig. 5-54, dar de atunci au mai fost adăugate și altele. Lista completă se găsește la adresa [www.iana.org/assignments/ip-parameters](http://www.iana.org/assignments/ip-parameters).

Opțiune	Descriere
Securitate	Menționează cât de secretă este datagrama
Dirijare strictă de la sursă	Indică calea completă de parcurs
Dirijare aproximativă de la sursă	Indică o listă a rutierelor care nu trebuie să ruteze
Înregistrează calea	Determină fiecare ruter să-și adauge adresa IP
Amprință de timp	Determină fiecare ruter să-și adauge adresa și o amprință de timp.

Fig. 5-54. Unele dintre opțiunile IP.

Opțiunea *Securitate* menționează cât de secretă este informația. În teorie, un ruter militar poate folosi acest câmp pentru a menționa că nu se dorește o dirijare prin anumite țări pe care militarii le consideră a fi „băieții răi”. În practică, toate ruterele îl ignoră, deci singura sa funcție practică este să ajute spionii să găsească mai ușor lucrurile de calitate.

Opțiunea *Dirijare strictă de la sursă* dă calea completă de la sursă la destinație ca o secvență de adrese IP. Datagrama este obligată să urmărească această cale precisă. Ea este deosebit de utilă pentru administratorii de sistem pentru a trimite pachete de urgență atunci când tabelele de dirijare sunt distruse sau pentru a realiza măsurători de timp.

Opțiunea *Dirijare aproximativă de la sursă* cere ca pachetul să traverseze o listă specificată de rutere și în ordinea specificată, dar este permisă trecerea prin alte rutere pe drum. În mod normal, această opțiune ar putea oferi doar câteva rutere, pentru a forma o anumită cale. De exemplu, pentru a forma un pachet de la Londra la Sydney să meargă spre vest în loc de est, această opțiune poate specifica rutere în New York, Los Angeles și Honolulu. Această opțiune este foarte utilă atunci când motive politice sau economice dictează trecerea prin anumite țări sau evitarea lor.

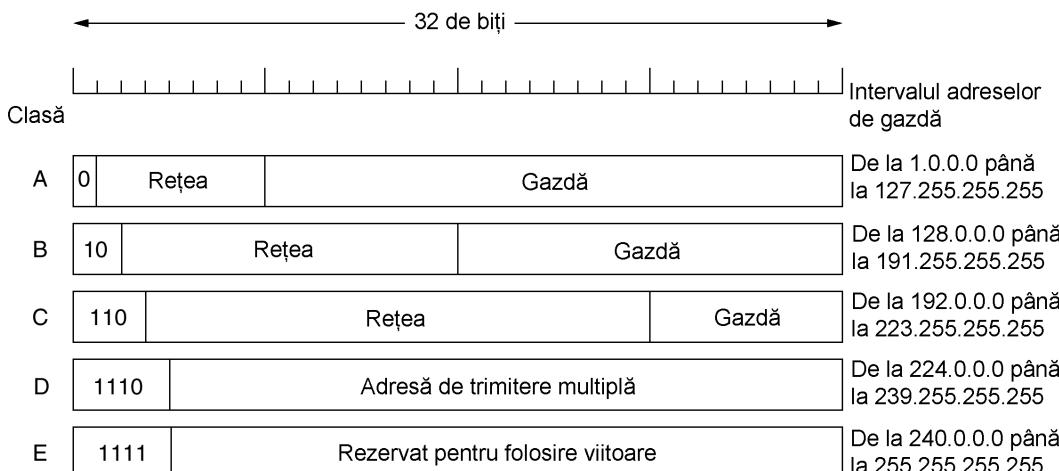
Opțiunea *Înregistrează calea* indică ruterelor de pe cale să-și adauge adresele IP la câmpul opțiunii. Aceasta permite administratorilor de sistem să localizeze pene în algoritmii de dirijare („De ce pachetele de la Houston la Dallas trec mai întâi prin Tokio?”). Când rețeaua ARPANET a fost înființată, nici un pachet nu trecea vreodată prin mai mult de nouă rutere, deci 40 de octeți pentru opțiuni au fost destui. Așa cum s-a menționat anterior, acum dimensiunea este prea mică.

În sfârșit, opțiunea *Amprentă de timp* este similară opțiunii *Înregistrează ruta*, cu excepția faptului că, în plus față de înregistrarea adresei sale de 32 de biți, fiecare ruter înregistrează și o amprentă de timp de 32 de biți. Si această opțiune este folosită în special pentru depanarea algoritmilor de dirijare.

### 5.6.2 Adrese IP

Fiecare gazdă și ruter din Internet are o adresă IP, care codifică adresa sa de rețea și de gazdă. Combinăția este unică: în principiu nu există două mașini cu aceeași adresă IP. Toate adresele IP sunt de 32 de biți lungime și sunt folosite în câmpurile *Adresă sursă* și *Adresă destinație* ale pachetelor IP. Este important de observat că o adresă IP nu se referă de fapt la o gazdă. Se referă de fapt la o interfață de rețea, deci dacă o gazdă este în două rețele, trebuie să folosească două adrese IP. Totuși în practică, cele mai multe gazde sunt conectate la o singură rețea și deci au o adresă IP.

Timp de mai multe decenii, adresele IP erau împărțite în cinci categorii ilustrate în fig. 5-55. Acest model de alocare a fost denumit **clase de adrese**. Nu mai este folosit, dar referințele la acest model sunt în continuare des întâlnite în literatură. Vom discuta puțin mai târziu ce a înlocuit modelul claselor de adrese.



**Fig. 5-55.** Formatul adreselor IP.

Formatele de clasă A, B, C și D permit până la 128 rețele cu 16 milioane de gazde fiecare, 16.384 rețele cu până la 64K gazde, 2 milioane de rețele (de exemplu, LAN-uri) cu până la 256 gazde fiecare (desi unele dintre acestea sunt speciale). De asemenea este suportată și trimitera multiplă (multicast), în care fiecare datagramă este direcționată mai multor gazde. Adresele care încep cu 1111 sunt rezervate pentru o folosire ulterioară. Peste 500 000 de rețele sunt conectate acum la Internet și numărul acestora crește în fiecare an. Pentru a evita conflictele numerele de rețea sunt atribuite de **ICANN** (**Internet Corporation for Assigned NAMES and Numbers** – Corporația Internet

pentru numere și nume atribuite). La rândul său, ICANN a împărtășit diverse autorități regionale să administreze părți din spațiul de adrese și acestea, la rândul lor, au împărtit adrese ISP-urilor și altor companii.

Adresele de rețea, care sunt numere de 32 de biți, sunt scrise în mod ușual în **notația zecimală cu punct**. În acest format, fiecare din cei 4 octeți este scris în zecimal, de la 0 la 255. De exemplu, adresa hexazecimală C0290614 este scrisă ca 192.41.6.20. Cea mai mică adresă IP este 0.0.0.0 și cea mai mare este 255.255.255.255.

Valorile 0 și -1 au semnificații speciale, așa cum se arată în fig. 5-56. Valoarea 0 înseamnă rețea-ua curentă sau gazda curentă. Valoarea -1 este folosită ca o adresă de difuzare pentru a desemna toate gazdele din rețea-ua indicată.

0 0				Stația gazdă
0 0		...	0 0	Gazdă
1 1				Difuzare în rețea-ua locală
Rețea		1 1 1 1	...	1 1 1 1
127				Bucătă locală
(Orice)				

Fig. 5-56. Adrese IP speciale.

Adresa IP 0.0.0.0 este folosită de gazde atunci când sunt pornite. Adresele IP cu 0 ca număr de rețea se referă la rețea-ua curentă. Aceste adrese permit ca mașinile să refere propria rețea fără a cunoaște numărul de rețea (dar ele trebuie să cunoască clasa adresei pentru a ști câte zerouri să includă). Adresele care constau numai din 1-uri permit difuzarea în rețea-ua curentă, în mod ușual un LAN. Adresele cu un număr exact de rețea și numai 1-uri în câmpul gazdă permit mașinilor să trimitem pachete de difuzare în LAN-uri la distanță, aflate oriunde în Internet (deși mulți administratori de sistem dezactivează această opțiune). În final, toate adresele de forma 127.xx.yz sunt rezervate pentru testări în buclă locală (loopback). Pachetele trimise către această adresă nu sunt trimise prin cablu; ele sunt prelucrate local și tratate ca pachete sosite. Aceasta permite trimiterea pachetelor către rețea-ua locală fără ca emițătorul să-i cunoască numărul.

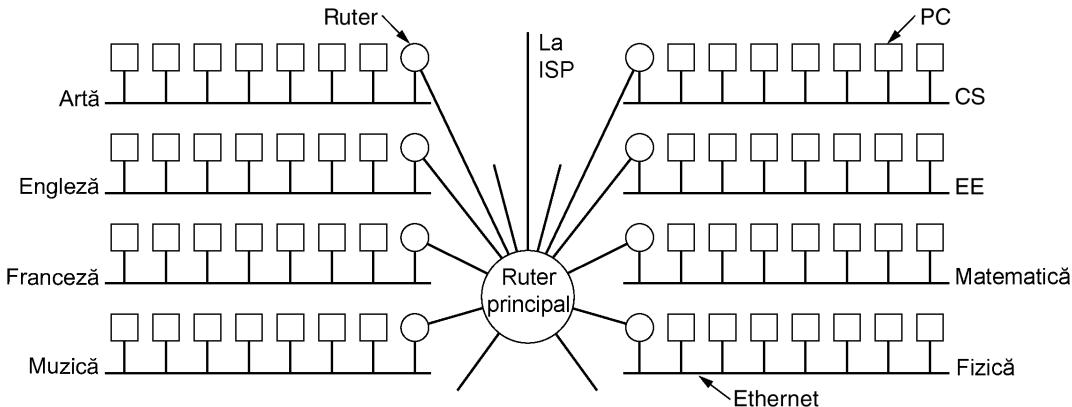
### Subiecte

Așa cum am văzut, toate gazdele dintr-o rețea trebuie să aibă același număr de rețea. Această proprietate a adresării IP poate crea probleme când rețea-ua crește. De exemplu, să considerăm o universitate care a folosit la început o rețea de clasă B pentru calculatoarele din rețea-ua Ethernet a Departamentului de Calculatoare. Un an mai târziu, departamentul de Inginerie Electrică a vrut acces la Internet, deci a cumpărat un repetor pentru a extinde Ethernetul Departamentului de Calculatoare în clădirea lor. Cu timpul, multe alte departamente au achiziționat calculatoare și limita de patru repezoare per Ethernet a fost rapid atinsă. Era nevoie de o altă organizare.

Achiziționarea altrei adrese de rețea ar fi fost dificilă deoarece adresele sunt insuficiente și universitatea avea deja adrese suficiente pentru peste 60,000 de stații. Problema provine de la regula care afirmă că o singură adresă de clasă A, B sau C se referă la o singură rețea, nu la o mul-

țime de rețele. Cum tot mai multe organizații s-au lovit de această problemă, sistemul de adresare a fost puțin modificat pentru a o rezolva

Soluția este să se permită ca o rețea să fie divizată în mai multe părți pentru uz intern, dar pentru lumea exterioară să se comporte tot ca o singură rețea. În ziua de azi o rețea de campus tipică poate arăta ca în fig. 5-57, cu un ruter principal conectat la un ISP sau la rețea regională și numeroase rețele Ethernet împărtăsite prin campus în diferite departamente. Fiecare Ethernet are propriul ruter conectat la ruterul principal (posibil printr-un LAN coloană vertebrală, dar natura conexiunii interruter nu este relevantă aici)



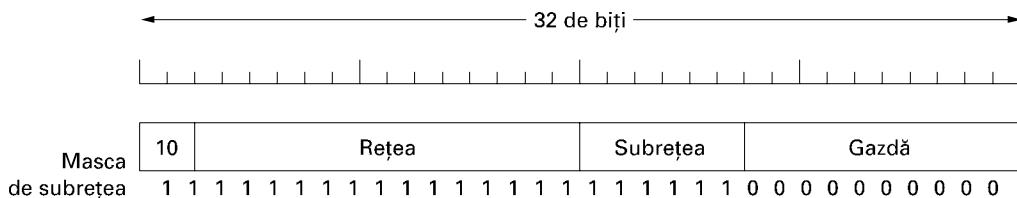
**Fig. 5-57.** O rețea de campus compusă din mai multe LAN-uri ale diferitelor departamente

În literatura Internet, aceste părți sunt numite **subrețele**. Așa cum am menționat în Cap. 1, această utilizare intră în conflict cu termenul „subrețea”, care înseamnă mulțimea tuturor ruterelor și liniei de comunicație dintr-o rețea. Din fericire, va fi clar din context care semnificație este atribuită. În această secțiune, și în următoarea definiția folosită va fi cea nouă.

Atunci când un pachet ajunge la ruterul principal, de unde știe acesta către ce subrețea (Ethernet) să-l trimită? O soluție ar fi să existe o tabelă cu 65,536 înregistrări în ruterul principal care să-i spună acestuiu ce ruter să folosească pentru fiecare stație din campus. Această idee ar funcționa, dar ar fi nevoie de o tabelă foarte mare în ruterul principal și de multă muncă manuală de întreținere pe măsură ce stațiile ar fi adăugate, mutate sau scoase din uz.

În locul ei a fost inventată o altă soluție. În esență, în loc să existe o singură adresă de clasă B, cu 14 biți pentru numărul rețelei și 16 biți pentru gazdă, un număr de biți din numărul gazdei sunt folosiți pentru a crea un număr de subrețea. De exemplu, dacă o universitate are 35 de departamente, ar putea folosi un număr de subrețea de 6 biți și un număr de 10 biți pentru gazde, ceea ce ar permite un număr de 64 de rețele Ethernet, fiecare cu un maxim de 1022 de gazde (așa cum am menționat mai devreme, 0 și -1 nu sunt disponibile). Această împărțire ar putea fi modificată ulterior în caz că a fost o greșeală.

Pentru a se putea folosi subrețele, ruterul principal trebuie să aibă nevoie de o **mască de subrețea**, care indică separarea dintre numărul rețea + subrețea și gazdă, așa cum este ilustrat în fig. 5-58. Măștile de subrețea sunt scrise de asemenea în notație zecimală cu punct, cu adăugarea unui caracter / (slash) urmat de numărul de biți din partea cu rețea + subrețea. Pentru exemplul din fig. 5-58, masca de subrețea poate fi scrisă ca 255.255.255.0 . O notație alternativă este /22 pentru a indica că masca subrețelei are lungimea de 22 de biți.



**Fig. 5-58.** O rețea de clasă B împărțită în 64 de subrețele

În afara rețelei, împărțirea în subrețele nu este vizibilă, astfel încât alocarea unei noi subrețele nu necesită contactarea ICANN sau schimbarea unor baze de date externe. În acest exemplu, prima subretea poate folosi adrese IP începând de la 130.50.4.1, cea de-a doua poate începe de la 130.50.8.1, cea de-a treia poate începe de la 130.50.12.1 și aşa mai departe. Pentru a se înțelege de ce numărul subretelei crește cu patru, să observăm că adresele binare corespunzătoare sunt următoarele:

Subrețea 1 :	10000010	00110010	000001 00	00000001
Subrețea 2 :	10000010	00110010	000010 00	00000001
Subrețea 3 :	10000010	00110010	000011 00	00000001

Aici, bara verticală (|) arată granița dintre numărul subretelei și numărul gazdei. La stânga se găsește numărul de 6 biți al subretelei, la dreapta numărul de 10 biți al gazdei.

Pentru a vedea cum funcționează subrețelele, este necesar să explicăm cum sunt prelucrate pachetele IP într-un ruter. Fiecare ruter are o tabelă ce memorează un număr de adrese IP de forma (rețea, 0) și un număr de adrese IP de forma (această-rețea, gazdă). Primul tip indică cum se ajunge la rețelele aflate la distanță. Al doilea tip spune cum se ajunge la gazdele locale. Fiecărei tabele îi este asociată interfață de rețea care se folosește pentru a ajunge la destinație și alte câteva informații.

Când sosesc un pachet IP, adresa destinație este căutată în tabela de dirijare. Dacă pachetul este pentru o rețea aflată la distanță, el este trimis ruterului următor prin interfață specificată în tabelă. Dacă este o gazdă locală (de exemplu în LAN-ul ruterului), pachetul este trimis direct către destinație. Dacă rețeaua nu este prezentă, pachetul este trimis unui ruter implicit care are tabele mai extinse. Acest algoritm înseamnă că fiecare ruter trebuie să memoreze numai rețele și gazde, nu perechi (rețea, gazdă), reducând considerabil dimensiunea tabelelor de dirijare.

Când este introdusă împărțirea în subrețele, tabelele de dirijare sunt schimbată, adăugând intrări de formă (această-rețea, subrețea, 0) și (această-rețea, această-subrețea, gazdă). Astfel un ruter din subrețeaua  $k$  știe cum să ajungă la toate celelalte subrețele și, de asemenea, cum să ajungă la toate gazdele din subrețeaua  $k$ . El nu trebuie să cunoască detalii referitoare la gazde din alte subrețele. De fapt, tot ceea ce trebuie schimbat este de a impune fiecarui ruter să facă un SI logic cu **masca de subrețea** a rețelei pentru a scăpa de numărul de gazdă și a căuta adresa rezultată în tabelele sale (după ce determină cărei clase de rețele aparține). De exemplu, asupra unui pachet adresat către 130.50.15.6 care ajunge la ruterul principal se face un SI logic cu masca de subrețea 255.255.252.0/22 pentru a obține adresa 130.50.12.0. Această adresă este căutată în tabelele de dirijare pentru a se găsi cum se ajunge la gazdele din subrețeaua 3. Ruterul din subrețeaua 5 este astfel ușurat de munca de a memora toate adresele de nivel legătură de date ale altor gazde decât cele din subrețeaua 5. Împărțirea în subrețele reduce astfel spațiul tabelelor de dirijare prin crearea unei ierarhii pe trei niveluri, alcătuită din rețea, subrețea și gazdă.

## CIDR - Dirijarea fără clase între domenii

IP este folosit intens de câteva decenii. A funcționat extrem de bine, aşa cum a fost demonstrat de creșterea exponențială a Internet-ului. Din nefericire, IP devine rapid o victimă a propriei popularități: își epuizează adresele. Acest dezastru fantomatic a generat foarte multe discuții și controverse în cadrul comunității Internet referitor la cum să fie tratat. În această secțiune vom descrie atât problema cât și câteva soluții propuse.

Prin 1987, câțiva vizionari au prezis că într-o zi Internet-ul poate crește până la 100.000 de rețele. Cei mai mulți experti s-au exprimat cu dispreu că aceasta va fi peste decenii, dacă va fi vreodată. Cea de-a 100.000-a rețea a fost conectată în 1996. Problema, expusă anterior, este că Internet-ul își epuizează rapid adresele IP. În principiu, există peste 2 miliarde de adrese, dar practica organizării spațiului de adrese în clase (vezi fig. 5-55) irosește milioane din acestea. În particular, adevărații vinovați sunt de rețelele de clasă B. Pentru cele mai multe organizații, o adresă de clasă A, cu 16 milioane de adrese este prea mare, iar o rețea de clasă C, cu 256 de adrese, este prea mică. O rețea de clasă B, cu 65.536 adrese, este numai bună. În folclorul Internet, această situație este cunoscută ca **problema celor trei urși** (ca din *Goldilocks and the Three Bears*).

În realitate, o adresă de clasă B este mult prea mare pentru cele mai multe organizații. Studiile au arătat că mai mult de jumătate din toate rețelele de clasă B au mai puțin de 50 de gazde. O rețea de clasă C ar fi fost suficientă, dar fără îndoială că fiecare organizație care a cerut o adresă de clasă B a crezut că într-o zi ar putea depăși câmpul gazdă de 8 biți. Privind retrospectiv, ar fi fost mai bine să fi avut rețele de clasă C care să folosească 10 biți în loc de opt pentru numărul de gazdă, permitând 1022 gazde pentru o rețea. În acest caz, cele mai multe organizații s-ar fi decis, probabil, pentru o rețea de clasă C și ar fi existat jumătate de milion dintre acestea (comparativ cu doar 16.384 rețele de clasă B).

Este greu să fie învinuiri proiectanții Internetului pentru că nu au pus la dispoziție mai multe adrese de clasă B (și mai mici). În momentul în care s-a luat decizia să fie create cele trei clase, Internetul era o rețea de cercetare care conecta marile universități din SUA (plus un număr mic de companii și sit-uri militare care făceau cercetări în domeniul rețelelor). Pe atunci nimeni nu a percepuit Internetul ca fiind un sistem de comunicație în masă care va rivaliza cu rețeaua telefonică. Fără îndoială că în acel moment cineva a spus: „SUA are aproximativ 2000 de universități și colegii. Chiar dacă toate se conectează la Internet și se mai conectează și multe universități din alte țări, nu o să ajungem niciodată la 16.000 pentru că nu există atâta universitate în întreaga lume. În plus faptul că numărul gazdei este un număr întreg de octeți, mărește viteza de prelucrare a pachetelor.”

Totuși, dacă s-ar fi alocat 20 de biți numărului de rețea pentru rețele de clasă B, ar fi apărut o altă problemă: explozia tabelelor de dirijare. Din punctul de vedere al ruterelor, spațiul de adrese IP este o ierarhie pe două niveluri, cu numere de rețea și numere de gazde. Ruterul nu trebuie să știe despre toate gazdale, dar el trebuie să știe despre toate rețelele. Dacă ar fi fost în folosință jumătate de milion de rețele de clasă C, fiecare ruter din întregul Internet ar fi necesitat o tabelă cu o jumătate de milion de intrări, una pentru fiecare rețea, spunând care linie se folosește pentru a ajunge la respectiva rețea, împreună cu alte informații.

Memorarea fizică efectivă a tabelelor cu jumătate de milion de intrări este probabil realizabilă, deși costisoare pentru ruterelor critice care trebuie să mențină tabelele în RAM static pe plăcile I/O. O problemă mai serioasă este reprezentată de creșterea complexității diferenților algoritmi referitori la gestiunea tabelelor, care este mai rapidă decât liniară. Și mai rău, o mare parte din programele și firmware-ul ruterelor existente a fost proiectat pe vremea când Internet-ul avea 1000 de rețele co-

nectate și 10.000 de rețele păreau la depărtare de decenii. Deciziile de proiectare făcute atunci sunt de departe de a fi optime acum.

În plus, diferenți algoritmi de dirijare necesită ca fiecare ruter să-și transmită tabelele periodic (de exemplu protocolul vectorilor distanță). Cu cât tabelele sunt mai mari, cu atât este mai probabil ca anumite părți să se piardă pe drum, ducând la date incomplete la celălalt capăt și, posibil, la instabilități de dirijare.

Problema tabelelor de dirijare ar fi putut fi rezolvată prin adoptarea unei ierarhii mai adânci. De exemplu, ar fi mers dacă s-ar fi impus ca fiecare adresă să conțină un câmp țară, județ/provincie, oraș, rețea și gazdă. Atunci fiecare ruter ar fi trebuit să știe doar cum să ajungă la fiecare țară, la județele sau provinciile din țara sa, orașele din propriul județ sau provincie și rețelele din orașul său. Din nefericire, această soluție ar necesita mai mult de 32 de biți pentru adrese IP și ar folosi adresele neficiente (Liechtenstein ar fi avut tot atâția biți ca Statele Unite).

Pe scurt, cele mai multe soluții rezolvă o problemă, dar creează o altă. Soluția care a fost implementată acum și care a dat Internet-ului un pic de spațiu de manevră este **CIDR** (**C**lassless **I**nter**D**omain **R**outing - Dirijarea fără clase între domenii). Ideea de la baza CIDR, descrisă în RFC 1519, este de a aloca adresele IP rămase, în blocuri de dimensiune variabilă, fără a se ține cont de clase. Dacă un sit are nevoie, să zicem, de 2000 de adrese, îl este dat un bloc de 2048 adrese la o granită de 2048 de octeți.

Renunțarea la clase face rutarea mai complicată. În vechiul sistem cu clase, rutarea se efectua în felul următor. Atunci când un pachet ajungea la un ruter, o copie a adresei IP era deplasată la dreapta cu 28 de biți pentru a obține un număr de clasă de 4 biți. O ramificație cu 16 cai sortă pachetele în A, B, C și D (dacă era suportat), cu 8 cazuri pentru clasa A, patru cazuri pentru clasa B, două cazuri pentru clasa C și câte unul pentru D și E. Programul pentru fiecare clasă masca numărul de rețea de 8, 16 sau 24 de biți și îl alinia la dreapta într-un cuvânt de 32 de biți. Numărul de rețea era apoi căutat în tabela pentru A, B sau C, de obicei indexat pentru rețelele A și B și folosind dispersia (hashing) pentru rețelele C. Odată intrarea găsită, se putea găsi linia de ieșire și pachetul era retransmis.

Cu CIDR, acest algoritm simplu nu mai funcționează. În loc de aceasta, fiecare intrare din tabela de rutare este extinsă cu o mască de 32 de biți. Din acest motiv, acum există o singură tabelă de rutare pentru toate rețelele, constituită dintr-un vector de triplete (adresă IP, mască subrețea, linie de ieșire). Când sosește un pachet IP, mai întâi se extrage adresa IP a destinației. Apoi (conceptual) tabela de rutare este scanată intrare cu intrare, mascând adresa destinație și comparând cu intrarea din tabelă, în căutarea unei potriviri. Este posibil ca mai multe intrări (cu măști de subrețea de lungimi diferite) să se potrivească, caz în care este folosită cea mai lungă mască. Astfel, dacă există potrivire pentru o mască /20 și pentru o mască /24, este folosită intrarea cu /24.

Pentru a accelera procesul de găsire a adreselor au fost imaginati algoritmi complecsi (Ruiz-Sanchez et al., 2001). Ruterele comerciale folosesc chip-uri VLSI proprietare cu acești algoritmi implementați în hardware.

Pentru a face algoritmul de retransmitere mai ușor de înțeles, să considerăm un exemplu în care sunt disponibile milioane de adrese, începând de la 194.24.0.0. Să presupunem că Universitatea Cambridge are nevoie de 2048 de adrese și are atribuite adresele de la 194.24.0.0 până la 194.24.7.255, împreună cu masca 255.255.248.0. Apoi, Universitatea Oxford cere 4096 de adrese. Deoarece un bloc de 4096 de adrese trebuie să fie aliniat la o frontieră de 4096 octeți, acestea nu pot fi numerotate începând de la 194.8.0.0. În loc, se primesc adrese de la 194.24.16.0 până la 194.24.31.255, împreună cu o mască de 255.255.240.0. Acum Universitatea din Edinburgh cere

1024 de adrese și îi sunt atribuite adresele de la 194.24.8.0 până la 194.24.11.255 și masca 255.255.252.0. Alocările sunt rezumate în fig. 5-59.

Universitatea	Prima adresă	Ultima adresă	Număr adrese	Notăție
Cambridge	194.24.0.0	194.24.7.255	2048	194.24.0.0/21
Edinburgh	194.24.8.0	194.24.11.255	1024	194.24.8.0/22
(disponibil)	194.24.12.0	194.24.15.255	1024	194.24.12/22
Oxford	194.24.16.0	194.24.31.255	4096	194.24.16.0/20

**Fig. 5-59.** Un set de atribuiri de adrese IP.

Tabelele de dirijare din toată lumea sunt acum actualizate cu cele trei intrări atribuite. Fiecare intrare conține o adresă de bază și o mască de subrețea. Aceste intrări (în binar) sunt:

Adresă	Mască
C 11000010 00011000 00000000 00000000	11111111 11111111 11111000 00000000
E 11000010 00011000 00010000 00000000	11111111 11111111 11111100 00000000
C 11000010 00011000 00010000 00000000	11111111 11111111 11110000 00000000

Să vedem acum ce se întâmplă când sosește un pachet adresat pentru 194.24.17.4, care este reprezentat ca următorul sir de 32 de biți :

11000010 00011000 00010001 00000100

Mai întâi, se face un řI logic cu masca de la Cambridge pentru a obține:

11000010 00011000 00010000 00000000

Această valoare nu se potrivește cu adresa de bază de la Cambridge, așa că adresei originale i se aplică un řI logic cu masca de la Edinburgh pentru a se obține:

11000010 00011000 00010000 00000000

Această valoare nu se potrivește cu adresa de bază de la Edinburgh, așa că se încearcă în continuare Oxford, obținându-se:

11000010 00011000 00010000 00000000

Această valoare se potrivește cu adresa de bază de la Oxford. Dacă în restul tabelei nu sunt găsite potriviri mai lungi, atunci este folosită intrarea pentru Oxford și pachetul va fi trimis pe linia corespunzătoare.

Acum să privim cele trei universități din punctul de vedere al unui ruter din Omaha, Nebraska, care are doar patru linii de ieșire: Minneapolis, New York, Dallas și Denver. Atunci când software-ul ruterului primește cele trei noi intrări, realizează că le poate combina pe toate trei într-o singură **intrare agregată** (eng.: **aggregate entry**) 194.24.0.0/19 cu adresa binară și o masca de subrețea următoare :

11000010 00000000 00000000 00000000	11111111 11111111 11100000 00000000
-------------------------------------	-------------------------------------

Această intrare trimite toate pachetele destinate uneia dintre cele trei universități la New York. Reunind cele trei intrări ruterul din Omaha și-a redus mărimea tabelei de rutare cu doi.

Dacă New York-ul are o singură conexiune cu Londra pentru tot traficul spre Marea Britanie, poate folosi de asemenea o intrare agregată. Totuși, dacă are două conexiuni separate pentru Londra și Edinburgh, atunci trebuie să aibă trei intrări.

Ca o notă finală despre acest exemplu, intrarea agregată a ruterului din Omaha va trimite și pacetele către adresele nealocate tot spre New York. Cât timp adresele sunt cu adevărat nealocate, aceasta nu contează pentru că aşa ceva nu ar trebui să se întâpte. Totuși, dacă mai târziu sunt alocate unei companii din California, pentru a le trata va fi nevoie de o înregistrare separată, 194.24.12.0/22.

### NAT – Translatarea adreselor de rețea

Adresele IP sunt insuficiente. Un ISP ar putea avea o adresă /16 (anterior de clasă B) oferindu-i 65.534 numere de stații. Dacă are mai mulți clienți, are o problemă. Pentru utilizatorii casnici cu conexiuni pe linie telefonică (eng.: dial-up), o soluție ar fi să se aloce dinamic o adresă IP fiecărui calculator în momentul în care sună și se conectează și să o dealoce în momentul în care se termină sesiunea. În acest fel o singură adresă /16 poate fi folosită pentru 65.534 utilizatori activi, ceea ce este probabil suficient pentru un ISP cu mai multe sute de mii de utilizatori. În momentul în care sesiunea se termină, adresa IP este alocată altui calculator care sună. Deși această strategie funcționează bine pentru un ISP cu un număr moderat de utilizatori casnici, eșuează pentru ISP-uri care deservesc preponderent companii.

Problema este că o companie se așteaptă să fie on-line în mod continuu în timpul orelor de program. Atât companiile mici, cum ar fi o companie de turism compusă din trei persoane, cât și mari corporații au mai multe calculatoare conectate de un LAN. Unele sunt calculatoarele personale ale angajaților; altele pot fi servere Web. În general în LAN există un ruter care este conectat cu ISP-ul printr-o linie închiriată pentru a avea conexiune continuă. Această situație impune ca fiecărui calculator să îi fie asociat un IP propriu pe parcursul întregii zile. De fapt, numărul total al calculatoarelor companiilor care sunt clienți ai ISP-ului nu poate depăși numărul de adrese IP pe care le posedă acesta. Pentru o adresă /16, aceasta limitează numărul total de calculatoare la 65.534. Pentru un ISP cu zeci de mii de companii cliente, această limită va fi repede depășită.

Pentru a agrava situația, din ce în ce mai mulți utilizatori se abonează de acasă la ADSL sau Internet prin cablu. Două dintre facilitățile oferite de aceste servicii sunt (1) utilizatorul primește o adresă IP permanentă și (2) nu există taxă de conectare (doar o taxă lunară), aşa că mulți utilizatori ai ADSL-ului și ai Internetului prin cablu rămân conectați permanent. Aceasta contribuie la insuficiența adreselor IP. Alocarea dinamică a adreselor IP aşa cum se face cu utilizatorii dial-up nu este utilă pentru că numărul adreselor IP folosite la un moment dat poate fi de multe ori mai mare decât numărul adreselor deținute de ISP.

Și pentru a complica și mai mult lucrurile, mulți utilizatori de ADSL și Internet prin cablu au doar sau mai multe calculatoare acasă, deseori câte unul pentru fiecare membru al familiei, și toti vor să fie online tot timpul folosind singura adresă IP care le-a fost alocată de către ISP. Soluția este să se conecteze toate PC-urile printr-un LAN și să se pună un ruter. Din punctul de vedere al ISP-ului, familia este acum ca o companie cu câteva calculatoare. Bine-ați venit la Jones, Inc.

Problema epuizării adreselor IP nu este o problemă teoretică care ar putea apărea cândva în viitorul îndepărtat. A apărut aici și acum. Soluția pe termen lung este ca tot Internetul să migreze la IPv6, care are adrese de 128 de biți. Tranzitia se desfășoară încet, dar vor trece ani până la finalizarea procesului. În consecință, anumite persoane au considerat că este nevoie de o rezolvare rapidă pe termen scurt. Rezolvarea a venit în forma NAT (Network Address Translation –

Translatarea adreselor de rețea), care este descrisă în RFC 3022 și pe care o vom rezuma mai jos. Pentru informații suplimentare consultați (Dutcher, 2001).

Ideea de bază a NAT-ului este de a aloca fiecărei companii o singură adresă IP (sau cel mult un număr mic de adrese) pentru traficul Internet. În interiorul companiei, fiecare calculator primește o adresă IP unică, care este folosită pentru traficul intern. Totuși, atunci când un pachet părăsește compania și se duce la ISP, are loc o translatare de adresă. Pentru a face posibil acest lucru, au fost declarate ca fiind private trei intervale de adrese IP. Companiile le pot folosi intern aşa cum doresc. Singura regulă este ca nici un pachet conținând aceste adrese să nu apară pe Internet. Cele trei intervale rezervate sunt :

10.0.0.0	- 10.255.255.255/8	(16.777.216 gazde)
172.16.0.0	- 172.31.255.255/12	(1.048.576 gazde)
192.168.0.0	- 192.168.255.255/16	(65.536 gazde)

Primul interval pune la dispoziție 16.777.216 adrese (cu excepția adreselor 0 și -1, ca de obicei) și este alegerea obișnuită a majorității companiilor, chiar dacă nu au nevoie de aşa de multe adrese.

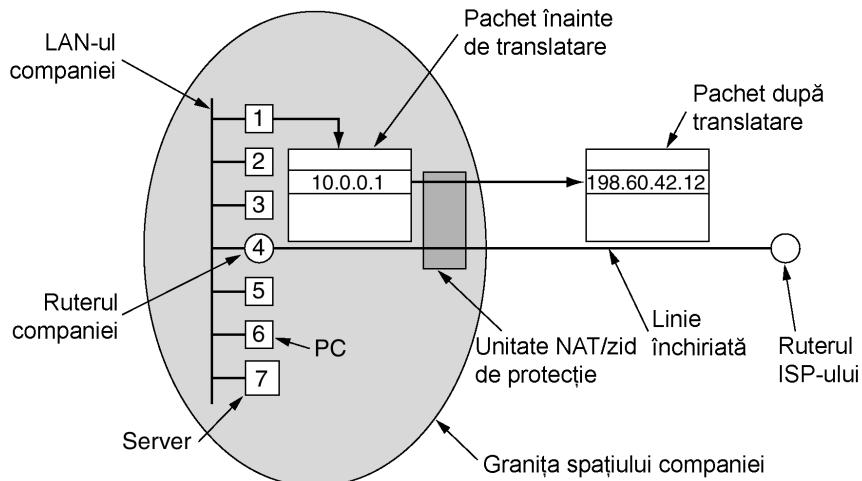


Fig. 5-60. Amplasarea și funcționarea unei unități NAT.

Funcționarea NAT este ilustrată în fig. 5-60. În interiorul companiei fiecare mașină are o adresă unică de forma 10.x.y.z. Totuși, când un pachet părăsește compania, trece printr-o **unitate NAT** (eng.: **NAT box**) care convertește adresa IP internă, 10.0.0.1 în figură, la adresa IP reală a companiei, 198.60.42.12, în acest exemplu. Unitatea NAT este deseori combinată într-un singur dispozitiv cu un zid de protecție (eng.: **firewall**), care asigură securitatea controlând ce intră și ieșe din companie. Vom studia zidurile de protecție în Cap. 8. De asemenea este posibilă integrarea unității NAT în ruterul companiei.

Până acum am trecut cu vederea un detaliu: atunci când vine răspunsul (de exemplu de la un server Web), este adresat 198.60.42.12, deci de unde știe unitatea NAT cu care adresă să o înlocuiească pe aceasta? Aici este problema NAT-ului. Dacă ar fi existat un câmp liber în antetul IP, acel câmp ar fi putut fi folosit pentru a memora cine a fost adevăratul emițător, dar numai 1 bit este încă nefolositor. În principiu, o nouă opțiune ar putea fi creată pentru a reține adevărată adresa a

sursei, dar acest lucru ar necesita schimbarea codului din protocolul IP de pe toate stațiile din întregul Internet pentru a putea prelucra noua opțiune. Aceasta nu este o alternativă promițătoare pentru o rezolvare rapidă.

În cele ce urmează se prezintă ceea ce s-a întâmplat cu adevărat. Proiectanții NAT au observat că marea majoritate a pachetelor IP aveau conținut TCP sau UDP. Atunci când vom studia proto-coalele TCP și UDP în Cap. 6, vom vedea că ambele au antete care conțin un port sursă și un port destinație. În continuare vom discuta doar despre porturi TCP, dar exact aceleași lucruri se aplică și la UDP. Porturile sunt numere întregi pe 16 biți care indică unde începe și unde se termină o conexiune TCP. Aceste câmpuri pun la dispoziție câmpul necesar funcționării NAT.

Atunci când un proces dorește să stabilească o conexiune TCP cu un proces de la distanță, se atașează unui port TCP nefolosit de pe mașina sa. Acesta se numește **portul sursă** și spune codului TCP unde să trimită pachetele care vin și aparțin acestei conexiuni. Procesul furnizează și un **port destinație** pentru a spune cui să trimită pachetele în cealaltă parte. Porturile 0-1023 sunt rezervate pentru servicii bine cunoscute. De exemplu portul 80 este folosit de serverele Web, astfel încât clientii să le poată localiza. Fiecare mesaj TCP care pleacă conține atât un port sursă cât și un port destinație. Cele două porturi permit identificarea proceselor care folosesc conexiunea la cele două capete.

S-ar putea ca o analogie să clarifice mai mult utilizarea porturilor. Imagineați-vă o companie cu un singur număr de telefon principal. Atunci când cineva sună la acest număr, vorbește cu un operator care întreabă ce interior dorește și apoi face legătura la acel interior. Numărul principal este analog cu adresa IP a companiei iar interioarele de la ambele capete sunt analoage cu porturile. Porturile reprezintă 16 biți suplimentari de adresare identificând procesul care primește un pachet sosit.

Folosind câmpul *Port sursă* rezolvăm problema de corespondență. De fiecare dată când un pachet pleacă, el intră în unitatea NAT și adresa sursă 10.x.y.z este înlocuită cu adresa reală a companiei. În plus, câmpul TCP *Port sursă* este înlocuit cu un index în tabela de translatăre a unității NAT, care are 65.536 intrări. Această tabelă conține adresa IP inițială și portul inițial. În final, sunt recalculate și inserate în pachet sumele de control ale antetelor IP și TCP. Câmpul *Port sursă* trebuie înlocuit pentru că s-ar putea întâmpla, de exemplu, ca stațiile 10.0.0.1 și 10.0.0.2 să aibă amândouă conexiuni care să folosească portul 5000, deci câmpul *Port sursă* nu este suficient pentru a identifica procesul emițător.

Atunci când la unitatea NAT sosesc un pachet de la ISP, *Portul sursă* din antetul TCP este extras și folosit ca index în tabela de corespondență a unității NAT. Din intrarea localizată sunt extrase și inserate în pachet adresa IP internă și *Portul sursă* TCP inițial. Apoi sunt recalculate sumele de control TCP și IP și inserate în pachet. După aceea pachetul este transferat ruterului companiei pentru transmitere normală folosind adresa 10.x.y.z.

NAT poate fi folosit pentru rezolvă insuficiența adreselor IP pentru utilizatori de ADSL și Internet prin cablu. Atunci când un ISP aloca fiecărui utilizator o adresă, folosește adrese 10.x.y.z. Atunci când pachete de la stațiile utilizatorilor ies din ISP și intră în Internet trec printr-o cutie NAT care le translatează în adresele Internet adevărate ale ISP-ului. La întoarcere pachetele trec prin maparea inversă. Din această perspectivă, pentru restul Internetului, ISP-ul și utilizatorii săi ADSL/cablu arată la fel ca o mare companie.

Deși această schemă rezolvă într-un fel problema, multe persoane din comunitatea IP o consideră un monstru pe fața pământului. Rezumate succint, iată câteva dintre obiecții. În primul rând, NAT violează modelul arhitectural al IP-ului, care afirmă că fiecare adresă IP identifică unic o sin-

gură mașină din lume. Întreaga structură software a Internetului este construită pornind de la acest fapt. Cu NAT, mii de mașini pot folosi (și folosesc) adresa 10.0.0.1.

În al doilea rând, NAT schimbă natura Internetului de la o rețea fără conexiuni la un fel de rețea cu conexiuni. Problema este că o unitate NAT trebuie să mențină informații (corespondențe) pentru fiecare conexiune care trece prin ea. Faptul că rețea păstrează starea conexiunilor este o proprietate a rețelelor orientate pe conexiune, nu a celor neorientate pe conexiune. Dacă o unitate NAT se defectează și tabela de corespondență este pierdută, toate conexiunile TCP sunt distruse. În absența NAT, căderea rutelor nu are nici un efect asupra TCP. Procesul care transmite ajunge la limita de timp în câteva secunde și retrasmite toate pachetele neconfirmate. Cu NAT, Internetul devine la fel de vulnerabil ca o rețea cu comutare de circuite.

În al treilea rând, NAT încalcă cea mai fundamentală regulă a protocolelor pe niveluri: nivelul  $k$  nu poate face nici un fel de presupuneri referitor la ceea ce a pus nivelul  $k+1$  în câmpul de informație utilă. Prințipiu de bază este să se păstreze niveluri independente. Dacă mai târziu TCP este înlocuit de TCP-2, cu un antet diferit (de exemplu porturi pe 32 de biți), NAT va eșua. Ideea protocolelor pe niveluri este de a asigura că modificările la un nivel nu necesită modificări la celelalte niveluri. NAT distrug această independentă.

În al patrulea rând, procesele din Internet nu sunt obligate să folosească TCP sau UDP. Dacă un utilizator de pe mașina  $A$  se decide să folosesc un nou protocol de transport pentru a comunica cu un utilizator de pe mașina  $B$  (de exemplu, pentru o aplicație multimedia), introducerea unei unități NAT va determina eșecul aplicației, deoarece cutia NAT nu va fi în stare să localizeze corect câmpul TCP *Port sursă*.

În al cincilea rând, anumite aplicații introduc adrese IP în corpul mesajului. Receptorul extrage aceste adrese și le folosește. De vreme ce NAT nu știe nimic despre aceste adrese, nu le poate înlocui, deci orice încercare de a le folosi de către cealaltă parte va eșua. **FTP**, standardul **File Transfer Protocol** (rom.: protocol de transfer de fișiere) funcționează în acest mod și poate eșua în prezența NAT dacă nu se iau măsuri speciale. În mod similar protocolul pentru telefonie Internet H.323 (care va fi studiat în Cap. 8) are această proprietate și nu va funcționa în prezența NAT. Ar fi posibil să se modifice NAT-ul pentru a funcționa cu H.323, dar modificarea codului unității NAT de fiecare dată când apare o aplicație nouă nu este o idee bună.

În al șaselea rând, deoarece câmpul TCP *Port sursă* are 16 biți, o adresă IP poate fi pusă în corespondență cu cel mult 65.536 mașini. De fapt acest număr este puțin mai mic, deoarece primele 4096 porturi sunt rezervate pentru utilizări speciale. Totuși dacă sunt disponibile mai multe adrese IP fiecare poate trata 61.440 mașini.

Acestea și alte probleme ale NAT sunt discutate în RFC 2993. În general, opozanții NAT spun că rezolvând problema insuficienței adreselor IP cu o soluție temporară și urâtă, presiunea pentru a implementa soluția reală, adică tranzitia la IPv6, este redusă și acesta este un lucru rău.

#### 5.5.4 Protocole de control în Internet

Pe lângă IP, care este folosit pentru transferul de date, Internet-ul are câteva protocole de control la nivelul rețea, inclusiv ICMP, ARP, RARP, BOOTP și DHCP. În această secțiune vom arunca o privire asupra fiecărui dintre ele.

## Protocolul mesajelor de control din Internet

Operarea Internet-ului este strâns monitorizată de către rutere. Atunci când se întâmplă ceva neobișnuit, evenimentul este raportat prin **ICMP (Internet Control Message Protocol)** - protocolul mesajelor de control din Internet), care este folosit și pentru testarea Internet-ului. Sunt definite aproape o duzină de tipuri de mesaje ICMP. Cele mai importante sunt enumerate în fig. 5-61. Fiecare tip de mesaj ICMP este încapsulat într-un pachet IP.

Tipul mesajului	Descriere
Destinație inaccesibilă	Pachetul nu poate fi livrat
Timp depășit	Câmpul timp de viață a ajuns la 0
Problema de parametru	Câmp invalid în antet
Oprire sursă	Pachet de soc
Redirectare	Învăță un ruter despre geografie
Cerere de ecou	Întreabă o mașină dacă este activă
Răspuns ecou	
Cerere de amprentă de timp	La fel ca cererea de ecou, dar cu amprentă de timp
Răspuns cu amprentă de timp	La fel ca răspunsul ecou, dar cu amprentă de timp

Fig. 5-61. Tipurile principale de mesaje ICMP.

Mesajul **DESTINAȚIE INACCESIBILĂ** (DESTINATION UNREACHABLE) este folosit atunci când subrețeaua sau un ruter nu pot localiza destinația, sau un pachet cu bitul DF nu poate fi livrat deoarece o rețea cu „pachete mici” îi stă în cale.

Mesajul **TIMP DEPĂȘIT** (TIME EXCEEDED) este trimis când un pachet este eliminat datorită ajungerii contorului său la zero. Acest mesaj este un simptom al buclării pachetelor, al unei enorme congestii sau al stabilirii unor valori prea mici pentru ceas.

Mesajul **PROBLEMĂ DE PARAMETRU** (PARAMETER PROBLEM) indică detectarea unei valori nepermise într-un câmp din antet. Această problemă indică o eroare în programele IP ale gazdei emițătoare sau eventual în programele unui ruter tranzitat.

Mesajul **OPRIRE SURSĂ** (SOURCE QUENCH) a fost folosit pe vremuri pentru a limita traficul gazdelor care trimiteau prea multe pachete. Când o gazdă primea acest mesaj, era de așteptat să incetinească ritmul de transmisie. Este folosit arareori, deoarece când apare congestie, aceste pachete au tendința de a turna mai mult gaz pe foc. Controlul congestiei în Internet este făcut acum pe larg la nivelul transport și va fi studiat în detaliu în Cap. 6.

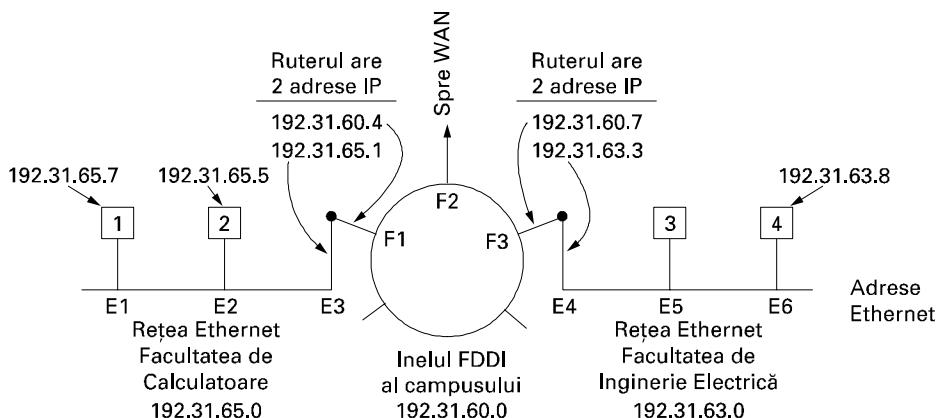
Mesajul **REDIRECTARE** (REDIRECT) este folosit atunci când un ruter observă că un pachet pare a fi dirijat greșit. Este folosit de ruter pentru a spune gazdei emițătoare despre eroarea probabilă.

Mesajele **CERERE ECOU** (ECHO REQUEST) și **RĂSPUNS ECOU** (ECHO REPLY) sunt folosite pentru a vedea dacă o anumită destinație este accesibilă și activă. Este de așteptat ca la recepția mesajului ECOU, destinația să răspundă printr-un mesaj RĂSPUNS ECOU. Mesajele **CERERE AMPRENTĂ DE TIMP** (TIMESTAMP REQUEST) și **RĂSPUNS AMPRENTĂ DE TIMP** (TIMESTAMP REPLY) sunt similare, cu excepția faptului că în răspuns sunt înregistrate timpul de sosire a mesajului și de plecare a răspunsului. Această facilitate este folosită pentru a măsura performanțele rețelei.

În plus față de aceste mesaje, au fost definite și altele. Lista se află on-line la adresa [www.iana.org/assignments/icmp-parameters](http://www.iana.org/assignments/icmp-parameters).

### Protocolul de rezoluție a adresei: ARP

Deși fiecare mașină din Internet are una sau mai multe adrese IP, acestea nu pot fi folosite de fapt pentru trimitera pachetelor deoarece hardware-ul nivelului legăturii de date nu înțelege adresele Internet. Actualmente, cele mai multe gazde sunt atașate la un LAN printr-o placă de interfață care înțelege numai adresele LAN. De exemplu, fiecare placă Ethernet fabricată până acum vine cu o adresă Ethernet de 48 biți. Fabricanții plăcilor Ethernet cer un spațiu de adrese de la o autoritate centrală pentru a se asigura că nu există două plăci cu aceeași adresă (pentru a evita conflictele care ar apărea dacă cele două plăci ar fi în același LAN). Plăcile trimit și primesc cadre pe baza adresei Ethernet de 48 biți. Ele nu știu absolut nimic despre adresele IP pe 32 de biți.



**Fig. 5-62.** Trei rețele /24 interconectate: două rețele Ethernet și un inel FDDI.

Se pune atunci întrebarea: Cum sunt transformate adresele IP în adrese la nivelul legăturii de date, ca de exemplu Ethernet? Pentru a explica care este funcționarea, vom folosi exemplul din fig. 5-62, în care este ilustrată o universitate mică ce are câteva rețele de clasă C (denumită acum /24). Avem două rețele Ethernet, una în facultatea de Calculatoare, cu adresa IP 192.31.65.0 și una în facultatea de Inginerie Electrică, cu adresa IP 192.31.63.0. Acestea sunt conectate printr-un inel FDDI la nivelul campusului, care are adresa IP 192.31.60.0. Fiecare mașină dintr-o rețea Ethernet are o adresă Ethernet unică, etichetată de la E1 la E6, iar fiecare mașină de pe inelul FDDI are o adresă FDDI, etichetată de la F1 la F3.

Să începem prin a vedea cum trimit un utilizator de pe gazda 1 un pachet unui utilizator de pe gazda 2. Presupunem că expeditorul știe numele destinatarului, ceva de genul *ary@eagle.cs.uni.edu*. Primul pas este aflarea adresei IP a gazdei 2, cunoscută ca *eagle.cs.uni.edu*. Această căutare este făcută de sistemul numelor de domenii (DNS), pe care îl vom studia în Cap. 7. Pentru moment, vom presupune că DNS-ul întoarce adresa IP a gazdei 2 (192.31.65.5).

Programele de la nivelurile superioare ale gazdei 1 construiesc un pachet cu 192.31.65.5 în câmpul *adresa destinatarului*, pachet care este trimis programelor IP pentru a-l transmite. Programele IP se uită la adresa și văd că destinatarul se află în propria rețea, dar au nevoie de un mijloc prin care să determine adresa Ethernet a destinatarului. O soluție este să avem undeva în sistem un fișier de configurare care transformă adresele IP în adrese Ethernet. Această soluție este posibilă, desigur, dar pentru organizații cu mii de mașini, menținerea fișierelor actualizate este o acțiune consumatoare de timp și care poate genera erori.

O soluție mai bună este ca gazda 1 să trimită un pachet de difuzare în rețeaua Ethernet întreband: „Cine este proprietarul adresei IP 192.31.65.5?”. Pachetul de difuzare va ajunge la toate mașinile din rețeaua Ethernet 192.31.65.0 și fiecare își va verifica adresa IP. Numai gazda 2 va răspunde cu adresa sa Ethernet (E2). În acest mod gazda 1 află că adresa IP 192.31.65.5 este pe gazda cu adresa Ethernet E2. Protocolul folosit pentru a pune astfel de întrebări și a primi răspunsul se numește **ARP (Address Resolution Protocol - Protocolul de rezoluție a adresei)**. Aproape toate mașinile din Internet îl folosesc. El este definit în RFC 826.

Avantajul folosirii ARP față de fișierele de configurare îl reprezintă simplitatea. Administratorul de sistem nu trebuie să facă prea multe, decât să atribuie fiecărei mașini o adresă IP și să hotărască măștile subretelelor. ARP-ul face restul.

În acest punct, programele IP de pe gazda 1 construiesc un cadru Ethernet adresat către E2, pun pachetul IP (adresat către 193.31.65.5) în câmpul informație utilă și îl lansează pe rețeaua Ethernet. Placa Ethernet a gazdei 2 detectează acest cadru, recunoaște că este un cadru pentru ea, îl ia repede și generează o intrerupere. Driverul Ethernet extrage pachetul IP din informația utilă și îl trimită programelor IP, care văd că este corect adresat și îl prelucrăză.

Pentru a face ARP-ul mai eficient sunt posibile mai multe optimizări. Pentru început, la fiecare execuție a ARP, mașina păstrează rezultatul pentru cazul în care are nevoie să contacteze din nou aceeași mașină în scurt timp. Dacă viitoare va găsi local corespondentul adresei, evitându-se astfel necesitatea unei a două difuzări. În multe cazuri, gazda 2 trebuie să trimită înapoi un răspuns, ceea ce îl forțează să execute ARP, pentru a determina adresa Ethernet a expeditorului. Această difuzare ARP poate fi evitată obligând gazda 1 să includă în pachetul ARP corespondența dintre adresa sa IP și adresa Ethernet. Când pachetul ARP ajunge la gazda 2, perechea (192.31.65.7, E1) este memorată local de ARP pentru o folosire ulterioară. De fapt, toate mașinile din rețeaua Ethernet pot memoria această relație în memoria ARP locală.

Altă optimizare este ca fiecare mașină să difuzeze corespondența sa de adrese la pornirea mașinii. Această difuzare este realizată în general printr-un pachet ARP de căutare a propriei adrese IP. Nu ar trebui să existe un răspuns, dar un efect lateral al difuzării este introducerea unei înregistrări în memoria ascunsă ARP a tuturor. Dacă totuși sosesc un răspuns (neșteptat), înseamnă că două mașini au aceeași adresă IP. Noua mașină ar trebui să-l informeze pe administratorul de sistem și să nu pornească.

Pentru a permite schimbarea relației, de exemplu, când o placă Ethernet se strică și este înlocuită cu una nouă (și astfel apare o nouă adresă Ethernet), înregistrările din memoria ascunsă ARP ar trebui să expire după câteva minute.

Să privim din nou fig. 5-62, numai că de această dată gazda 1 vrea să trimită un pachet către gazda 4 (192.31.63.8). Folosirea ARP va eşua pentru că gazda 4 nu va vedea difuzarea (ruterele nu trimit mai departe difuzările de nivel Ethernet). Există două soluții. Prima: ruterul facultății de Calculatoare poate fi configurat să răspundă la cererile ARP pentru rețeaua 193.31.63.0 (și posibil și pentru alte rețele locale). În acest caz, gazda 1 va memoria local perechea (193.31.63.8, E3) și va trimite tot traficul pentru gazda 4 către ruterul local. Această soluție se numește **ARP cu intermediar (proxy ARP)**. A doua soluție este ca gazda 1 să-și dea seama imediat că destinația se află pe o rețea aflată la distanță și să trimită tot traficul către o adresă Ethernet implicită care tratează tot traficul la distanță, în acest caz E3. Această soluție nu necesită ca ruterul facultății de Calculatoare să știe ce rețele la distanță deservește.

În ambele cazuri, ceea ce se întâmplă este că gazda 1 împachetează pachetul IP în câmpul informație utilă dintr-un cadru Ethernet adresat către E3. Când ruterul facultății de Calculatoare primeș-

te cadrul Ethernet, extrage pachetul IP din câmpul informație utilă și cauță adresa IP din tabelele sale de dirijare. Descoperă că pachetele pentru rețeaua 193.31.63.0 trebuie să meargă către ruterul 192.31.60.7. Dacă nu cunoaște încă adresa FDDI a lui 193.31.60.7, difuzează un pachet ARP pe inel și află că adresa din inel este *F3*. Apoi inserează pachetul în câmpul informație utilă al unui cadru FDDI adresat către *F3* și îl transmite pe inel.

Driverul FDDI al ruterului facultății de Inginerie Electrică scoate pachetul din câmpul informație utilă și îl trimită programelor IP care văd că trebuie să trimită pachetul către 192.31.63.8. Dacă această adresă IP nu este în memoria ascunsă ARP, difuzează o cerere ARP pe rețeaua Ethernet a facultății de Inginerie Electrică și află că adresa destinație este *E6*, astfel încât construiește un cadru Ethernet adresat către *E6*, pune pachetul în câmpul informație utilă și îl trimită în rețeaua Ethernet. Când cadrul Ethernet ajunge la gazda 4, pachetul este extras din cadrul și trimis programelor IP pentru procesare.

Transferul între gazda 1 și o rețea la distanță peste un WAN funcționează în esență asemănător, cu excepția că de data aceasta tabelele ruterului facultății de Calculatoare îi vor indica folosirea ruterului WAN, a cărui adresă FDDI este *F2*.

### RARP, BOOTP și DHCP

ARP-ul rezolvă problema aflării adresei Ethernet corespunzătoare unei adrese IP date. Câteodată trebuie rezolvată problema inversă: dându-se o adresă Ethernet, care este adresa IP corespunzătoare? În particular, această problemă apare când se pornește o stație de lucru fără disc. O astfel de mașină va primi, în mod normal, imaginea binară a sistemului său de operare de la un server de fișiere la distanță. Dar cum își află adresa IP?

Prima soluție proiectată a fost **RARP (Reverse Address Resolution Protocol** - Protocol de rezoluție inversă a adresei) (definit în RFC 903). Acest protocol permite unei stații de lucru de-a băia pornită să difuzeze adresa sa Ethernet și să spună: „Adresa mea Ethernet de 48 de biți este 14.04.05.18.01.25. Știe cineva adresa mea IP?” Serverul RARP vede această cerere, cauță adresa Ethernet în fișierele sale de configurare și trimită înapoi adresa IP corespunzătoare.

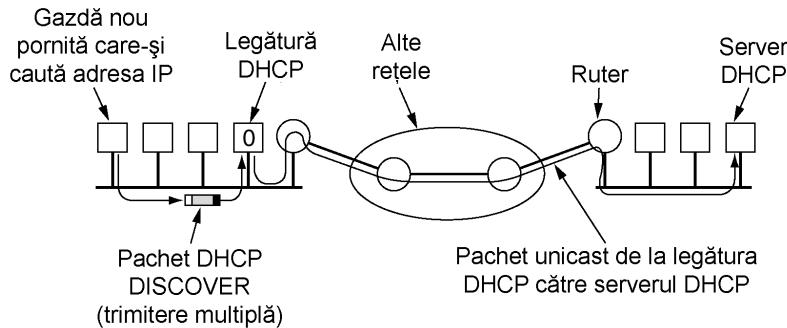
Folosirea RARP este mai bună decât introducerea unei adrese IP în imaginea de memorie, pentru că permite ca aceeași imagine să fie folosită pe toate mașinile. Dacă adresa IP ar fi fixată înăuntru imaginii, atunci fiecare stație de lucru ar necesita imaginea sa proprie.

Un dezavantaj al RARP este că, pentru a ajunge la serverul RARP, folosește o adresă destinație numai din 1-uri (difuzare limitată). Cu toate acestea, asemenea difuzări nu sunt propagate de rutere, așa încât este necesar un server RARP în fiecare rețea. Pentru a rezolva această problemă, a fost inventat un protocol alternativ de pornire, numit BOOTP (vezi RFC-urile 951, 1048 și 1084). Spre deosebire de RARP, acesta folosește mesaje UDP, care sunt propagate prin rutere. De asemenea furnizează unei stații de lucru fără disc informații suplimentare, care includ adresa IP a serverului de fișiere care deține imaginea de memorie, adresa IP a ruterului implicit și masca de subrețea care se folosește. BOOTP-ul este descris în RFC 951, 1048 și 1084.

O problemă serioasă cu BOOTP este că necesită configurarea manuală a corespondențelor între adresele IP și adresele Ethernet. Atunci când o nouă gazdă este adăugată la LAN, nu poate folosi BOOTP decât atunci când un administrator îi aloca o adresă IP și introduce manual (adresă Ethernet, adresă IP) în tabelele de configurare BOOTP. Pentru a elimina acest pas predispus la erori, BOOTP a fost extins și i-a fost dat un nou nume: **DHCP (Dynamic Host Configuration Protocol** – Protocol dinamic de configurare a gazdei). DHCP permite atât atribuirea manuală

de adrese IP, cât și atribuirea automată. Este descris în RFC-urile 2131 și 2132. În majoritatea sistemelor, a înlocuit în mare parte RARP și BOOTP.

Ca și RARP și BOOTP, DHCP este bazat pe ideea unui server special care atribuie adrese IP gazdelor care cer una. Acest server nu trebuie să se afle în același LAN cu gazda care face cererea. Deoarece serverul DHCP s-ar putea să nu fie accesibil prin difuzare, este nevoie ca în fiecare LAN să existe un **agent de legătură DHCP (DHCP relay agent)**, aşa cum se vede în fi. fig. 5-63.



**Fig. 5-63.** Funcționarea DHCP.

Pentru a-și afla adresa IP, o mașină tocmai pornită difuzează un pachet DHCP DISCOVER. Agentul de legătură DHCP din LAN interceptează toate difuzările DHCP. Atunci când găsește un pachet DHCP DISCOVER, îl trimită ca pachet unicast serverului DHCP, posibil într-o rețea depărtată. Singura informație de care are nevoie agentul este adresa IP a serverului DHCP.

O problemă care apare cu atribuirea automată a adreselor IP dintr-o rezervă comună este cât de mult ar trebui alocată o adresă IP. Dacă o gazdă părăsește rețeaua și nu returnează adresa sa IP serverului DHCP, acea adresă va fi pierdută permanent. După o perioadă de timp vor fi pierdute multe adrese. Pentru a preveni aceasta, atribuirea adresei IP va fi pentru o perioadă fixă de timp, o tehnică numită închiriere. Chiar înainte ca perioada de închiriere să expire, gazda trebuie să îi ceară DHCP-ului o reînnoire. Dacă nu reușește să facă cererea sau dacă cererea este respinsă, gazda nu va mai putea folosi adresa IP care îi fusesese dată mai devreme.

### 5.5.5 Protocolul de dirijare folosit de portile interioare: OSPF

Am terminat acum studiul protocolelor de control ale Internetului. Este timpul să trecem la următorul subiect : rutarea în Internet. Așa cum am menționat anterior, Internet-ul este construit dintr-un număr mare de sisteme autonome. Fiecare AS este administrat de o organizație diferită și poate folosi propriul algoritm de dirijare în interior. De exemplu, rețelele interne ale companiilor X, Y și Z ar fi văzute ca trei AS-uri dacă toate ar fi în Internet. Toate trei pot folosi intern algoritmi de dirijare diferenți. Cu toate acestea, existența standardelor, chiar și pentru dirijarea internă, simplifică implementarea la granițele dintre AS-uri și permite reutilizarea codului. În această secțiune vom studia dirijarea în cadrul unui AS. În următoarea, vom examina dirijarea între AS-uri. Un algoritm de dirijare în interiorul unui AS este numit **protocol de porți interioare**; un algoritm de dirijare utilizat între AS-uri este numit **protocol de porți exterioare**.

Protocolul de porți interioare inițial în Internet a fost un protocol de dirijare pe baza vectorilor distanță (RIP) bazat pe algoritmul Bellman-Ford moștenit de la ARPANET. El funcționa bine în sisteme mici, dar mai puțin bine pe măsură ce AS-urile s-au extins. El suferă de asemenea de pro-

blema numărării la infinit și de o convergență slabă în general, aşa că în mai 1979 a fost înlocuit de un protocol bazat pe starea legăturilor. În 1988, Internet Engineering Task Force a început lucrul la un succesor. Acel succesor, numit **OSPF (Open Shortest Path First)** - protocol public (deschis) bazat pe calea cea mai scurtă) a devenit un standard în 1990. Mulți producători de rutere oferă suport pentru el și acesta a devenit principalul protocol de porți interioare. În cele ce urmează vom face o schiță a funcționării OSPF. Pentru informații complete, vedeți RFC 2328.

Dată fiind experiența îndelungată cu alte protocole de dirijare, grupul care a proiectat noul protocol a avut o listă lungă de cerințe care trebuiau satisfăcute. Mai întâi, algoritmul trebuia publicat în literatura publică (open), de unde provine „O” din OSPF. O soluție în proprietatea unei companii nu era un candidat. În al doilea rând, noul protocol trebuia să suporte o varietate de metrii de distanță, incluzând distanța fizică, întârzierea și.a.m.d. În al treilea rând, el trebuia să fie un algoritm dinamic, care să se adapteze automat și repede la schimbările în topologie.

În al patrulea rând și nou pentru OSPF, trebuia să suporte dirijarea bazată pe tipul de serviciu. Noul protocol trebuia să fie capabil să dirijeze traficul de timp real într-un mod, iar alt tip de trafic în alt mod. Protocolul IP are câmpul *Tip serviciu*, dar nici un protocol de dirijare nu-l folosea. Acest câmp a fost introdus în OSPF dar tot nu îl folosea nimici și în cele din urmă a fost eliminat.

În al cincilea rând și în legătură cu cele de mai sus, noul protocol trebuia să facă echilibrarea încarcării, divizând încarcarea pe mai multe linii. Majoritatea protocolelor anterioare trimit toate pachetele pe cea mai bună cale. Calea de pe locul doi nu era folosită de loc. În multe cazuri, divizarea încarcării pe mai multe linii duce la performanțe mai bune.

În al șaselea rând, era necesar suportul pentru sisteme ierarhice. Până în 1988, Internet a crescut atât de mult, încât nu se poate aștepta ca un ruter să cunoască întreaga topologie. Noul protocol de dirijare trebuia să fie proiectat astfel, încât nici un ruter să nu fie nevoie să cunoască toată topologia.

În al șaptelea rând, se cerea un minim de măsuri de securitate, pentru a evita ca studenții iubitori de distracții să păcălească ruterele trimițându-le informații de dirijare false. În fine, a fost necesară luarea de măsuri pentru a trata ruterele care au fost conectate la Internet printr-un tunel. Protocolele precedente nu trauau bine acest caz.

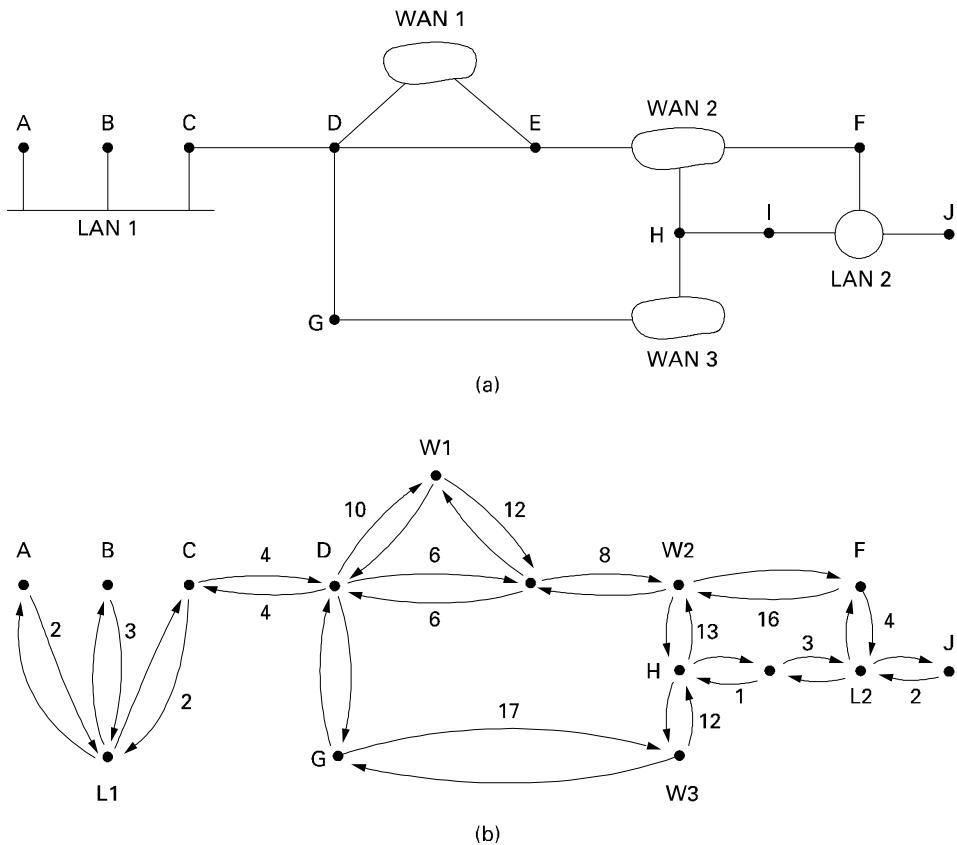
OSPF suportă trei tipuri de conexiuni și rețele:

1. Linii punct-la-punct între exact două rutere.
2. Rețele multiacces cu difuzare (de exemplu, cele mai multe LAN-uri).
3. Rețele multiacces fără difuzare (de exemplu, cele mai multe WAN-uri cu comutare de pachete).

O rețea **multiacces** este o rețea care poate să conțină mai multe rutere, fiecare dintre ele putând comunica direct cu toate celelalte. Toate LAN-urile și WAN-urile au această proprietate. Fig. 5-64(a) prezintă un AS care conține toate cele trei tipuri de rețele. Observați că gazdele, în general, nu joacă un rol în OSPF.

OSPF funcționează prin abstractizarea colecției de rețele, rutere și linii reale într-un graf orientat în care fiecărui arc îi este atribuit un cost (distanță, întârziere etc.). Apoi calculează cea mai scurtă cale bazându-se pe ponderile arcelor. O conexiune serială între două rutere este reprezentată de o pereche de arce, câte unul în fiecare direcție. Ponderile lor pot fi diferite. O rețea multiacces este reprezentată de un nod pentru rețeaua însăși plus câte un nod pentru fiecare ruter. Arcele de la nodul rețea la rutere au pondere 0 și sunt omise din graf.

Fig. 5-64(b) prezintă graful pentru rețeaua din fig. 5-64(a). Ponderile sunt simetrice, dacă nu se specifică altfel. Fundamental, ceea ce face OSPF este să reprezinte rețeaua reală printr-un graf ca acesta și apoi să calculeze cea mai scurtă cale de la fiecare ruter la fiecare alt ruter.



**Fig. 5-64.** (a) Un sistem autonom. (b) O reprezentare de tip graf a lui (a).

Multe din AS-urile din Internet sunt foarte mari și nu sunt simplu de administrat. OSPF le permite să fie divizate în **zone** numerotate, unde o zonă este o rețea sau o mulțime de rețele învecinate. Zonele nu se suprapun și nu este necesar să fie exhaustive, în sensul că unele rutere pot să nu aparțină nici unei zone. O zonă este o generalizare a unei subrețele. În afara zonei, topologia și detaliile sale nu sunt vizibile.

Orice AS are o zonă **de coloană vertebrală**, numită zona 0. Toate zonele sunt conectate la coloana vertebrală, eventual prin tuneluri, astfel încât este posibil să se ajungă din orice zonă din AS în orice altă zonă din AS prin intermediul coloanei vertebrale. Un tunel este reprezentat în graf ca un arc și are un cost. Fiecare ruter care este conectat la două sau mai multe zone aparține coloanei vertebrale. Analog cu celelalte zone, topologia coloanei vertebrale nu este vizibilă din afara coloanei vertebrale.

În interiorul unei zone, fiecare ruter are aceeași bază de date pentru starea legăturilor și folosește același algoritm de cea mai scurtă cale. Principala sa sarcină este să calculeze cea mai scurtă cale de la sine la fiecare alt ruter din zonă, incluzând ruterul care este conectat la coloana vertebrală, din care trebuie să existe cel puțin unul. Un ruter care conectează două zone are nevoie de baze de date pentru ambele zone și trebuie să folosească algoritmul de cale cât mai scurtă separat pentru fiecare zonă.

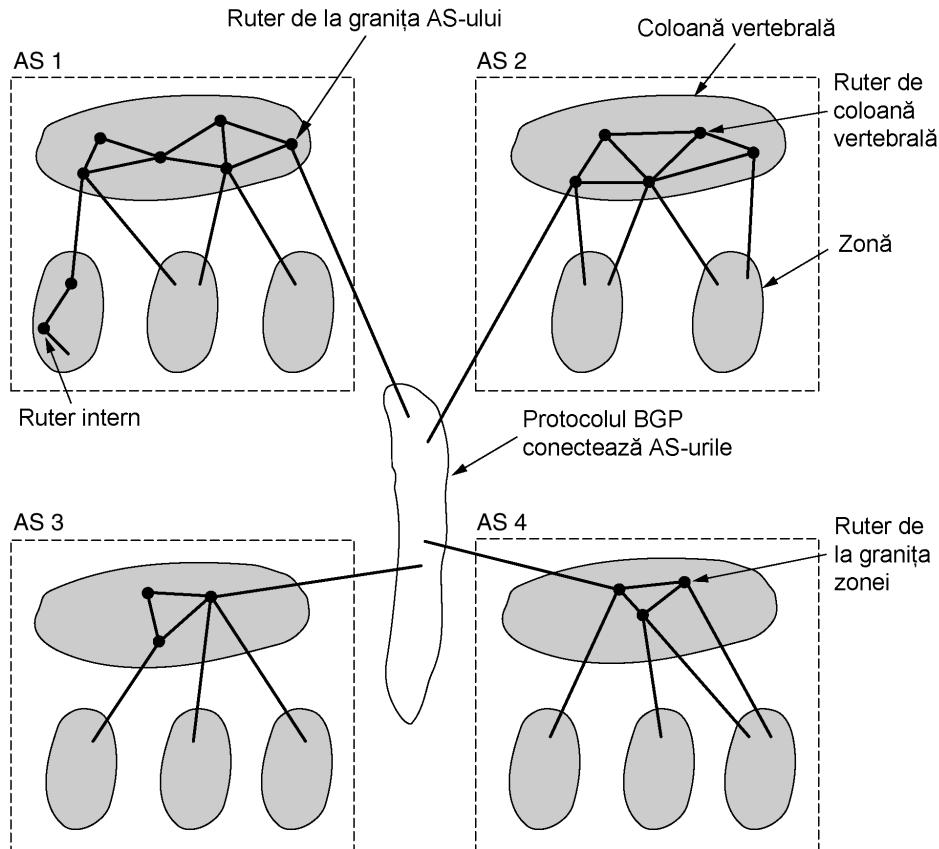
În timpul operării normale pot fi necesare trei tipuri de căi: intrazonale, interzonale și interAS-uri. Rutele intrazonale sunt cele mai ușoare, din moment ce ruterul sursă știe întotdeauna calea cea

mai scurtă spre ruterul destinație. Dirijarea interzonală se desfășoară întotdeauna în trei pași: drum de la sursă la coloana vertebrală; drum de-a lungul coloanei vertebrale până la zona destinație; drum la destinație. Acest algoritm forcează o configurație de tip stea pentru OSPF, coloana vertebrală fiind concentratorul (hub), iar celelalte zone fiind spițele. Pachetele sunt dirijate de la sursă la destinație „ca atare”. Ele nu sunt încapsulate sau trecute prin tunel, cu excepția cazului în care merg spre o zonă a cărei unică conexiune la coloana vertebrală este un tunel. Fig. 5-65 arată o parte a Internetului cu AS-uri și zone.

OSPF distinge patru clase de rutere:

1. Ruterele interne sunt integral în interiorul unei zone.
2. Ruterele de la granița zonei conectează două sau mai multe zone.
3. Ruterele coloanei vertebrale sunt pe coloana vertebrală.
4. Ruterele de la granița AS-urilor discută cu ruterele din alte AS-uri.

ACESTE CLASE POT SĂ SE SUPRAPUNĂ. De exemplu, toate ruterele de graniță fac parte în mod automat din coloana vertebrală. În plus, un ruter care este în coloana vertebrală, dar nu face parte din nici o altă zonă este de asemenea un ruter intern. Exemple din toate cele patru clase de rutere sunt ilustrate în fig. 5-65.



**Fig. 5-65.** Relația dintre AS-uri, coloane vertebrale și zone în OSPF.

Când un ruter pornește, trimite mesaje HELLO pe toate liniile sale punct-la-punct și trimite multiplu (multicast) în LAN-urile grupului compus din toate celelalte rutere. În WAN-uri, are nevoie de anumite informații de configurație, pentru a ști pe cine să contacteze. Din răspunsuri, fiecare ruter află care sunt vecinii săi. Ruterele din același LAN sunt toate vecine.

OSPF funcționează prin schimb de informații între rutere **adiacente**, care nu este același lucru cu schimbul de informații între ruterele vecine. În particular, este ineficient ca fiecare ruter dintr-un LAN să discute cu fiecare alt ruter din LAN. Pentru a evita această situație, un ruter este ales ca **ruter desemnat**. Se spune că el este adiacent cu toate celelalte rutere din LAN-ul său și schimbă informații cu ele. Ruterele vecine care nu sunt adiacente nu schimbă informații între ele. De asemenea, este actualizat în permanență și un ruter desemnat de rezervă pentru a ușura tranzitia dacă ruterul desemnat primar se defectează și trebuie să fie înlocuit imediat.

În timpul funcționării normale, fiecare ruter inundă periodic cu mesaje ACTUALIZARE STARE LEGĂTURĂ (Link State Update) fiecare ruter adiacent. Acest mesaj indică starea sa și furnizează costurile folosite în baza de date topologică. Mesajele de inundare sunt confirmate pentru a le face sigure. Fiecare mesaj are un număr de secvență, astfel încât un ruter poate vedea dacă un mesaj ACTUALIZARE STARE LEGĂTURĂ este mai vechi sau mai nou decât ceea ce are deja. De asemenea, ruterele trimit aceste mesaje când o linie cade sau își revine sau când costul acesteia se modifică.

Mesajele DESCRIERE BAZA DE DATE (Database Description) dau numerele de secvență pentru toate intrările de stare a liniei deținute actual de emițător. Prin compararea valorilor proprii cu acele ale emițătorului, receptorul poate determina cine are cea mai recentă valoare. Aceste mesaje sunt folosite când o linie este refăcută.

Fiecare partener poate cere informații de stare a legăturii de la celălalt folosind mesaje CERERE STARE LEGĂTURĂ (Link State Request). Rezultatul concret al acestui algoritm este că fiecare pereche de rutere adiacente verifică să vadă cine are cele mai recente date și astfel, noua informație este răspândită în zonă. Toate aceste mesaje sunt trimise ca simple pachete IP. Cele cinci tipuri de mesaje sunt rezumate în fig. 5-66.

Tip mesaj	Descriere
Hello	Folosit pentru descoperirea vecinilor
Actualizare Stare Legătură	Emitătorul furnizează vecinilor săi costurile sale
Confirmare Stare Legătură	Confirmă actualizarea stării legăturii
Descriere Bază de Date	Anunță ce actualizări are emițătorul
Cerere Stare Legătură	Cere informații de la partener

Fig. 5-66. Cele cinci tipuri de mesaje OSPF.

În final, putem să asamblăm toate piesele. Folosind inundarea, fiecare ruter informează toate celelalte rutere din zona sa despre vecinii și costurile sale. Această informație permite fiecărui ruter să construiască graful zonei (zonelor) sale și să calculeze cea mai scurtă cale. Zona de coloană vertebrală face și ea același lucru. În plus, ruterele de coloană vertebrală acceptă informația de la ruterele zonei de graniță cu scopul de a calcula cea mai bună cale de la fiecare ruter de coloană vertebrală către fiecare alt ruter. Această informație este propagată înapoi către ruterele zonei de graniță, care o fac publică în zonele lor. Folosind această informație, un ruter gata să transmită un pachet interzonal poate selecta cel mai bun ruter de ieșire către coloana vertebrală.

### 5.6.5 Protocolul de dirijare pentru porti externe: BGP

În cadrul unui singur AS, protocolul de dirijare recomandat este OSPF (deși, desigur, nu este singurul folosit). Între AS-uri se folosește un protocol diferit, **BGP** (**Border Gateway Protocol** - Protocolul portilor de graniță). Între AS-uri este necesar un protocol diferit pentru că scopurile unui protocol pentru porti interioare și ale unui protocol pentru porti exterioare sunt diferite. Tot ce trebuie să facă un protocol pentru porti interioare este să mute pachetele cât mai eficient de la sursă la destinație. El nu trebuie să-și facă probleme cu politica.

Ruterele ce folosesc protocolul de porti exterioare trebuie să țină cont într-o mare măsură de politică (Metz, 2001). De exemplu, un AS al unei corporații poate dori facilitatea de a trimite pachete oricărui sit Internet și să receptioneze pachete de la orice sit Internet. Cu toate acestea, poate să nu dorească să asigure tranzitarea pentru pachetele originare dintr-un AS străin destinate unui AS străin în diferit, chiar dacă prin AS-ul propriu trece cea mai scurtă cale dintre cele două AS-uri străine („Asta este problema lor, nu a noastră.”). Pe de altă parte, poate fi dispus să asigure tranzitarea pentru vecinii săi, sau chiar pentru anumite AS-uri care au plătit pentru acest serviciu. Companiile telefonice, de exemplu, pot fi fericite să acționeze ca un purtător pentru clienții lor, dar nu și pentru alții. Protocolele pentru porti externe, în general și BGP în particular, au fost proiectate pentru a permite forțarea multor tipuri de politici de dirijare pentru traficul între AS-uri.

Politicele tipice implică considerații politice, de securitate sau economice. Câteva exemple de constrângeri de dirijare sunt:

1. Nu se tranzitează traficul prin anumite AS-uri.
2. Nu se plasează Irak-ul pe o rută care pornește din Pentagon.
3. Nu se folosesc Statele Unite pentru a ajunge din Columbia Britanică în Ontario.
4. Albania este tranzitată numai dacă nu există altă alternativă către destinație.
5. Traficul care pleacă sau ajunge la IBM nu trebuie să tranziteze Microsoft.

În mod obișnuit politicele sunt configurate manual în fiecare ruter BGP (sau sunt incluse folosind un anumit tip de script). Ele nu sunt parte a protocolului însuși.

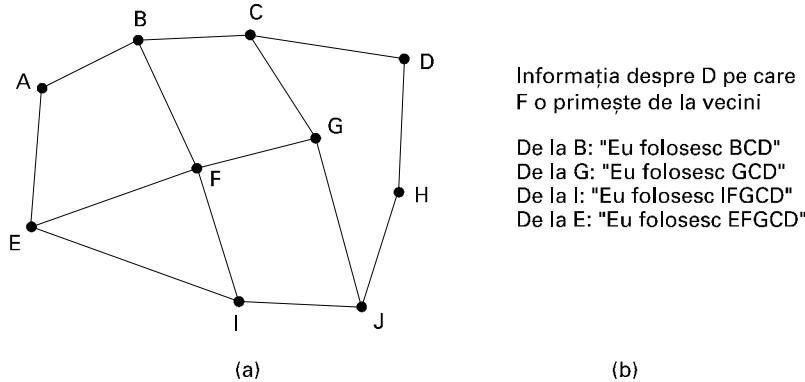
Din punctul de vedere al unui ruter BGP, lumea constă din AS-uri și liniile care le conectează. Două AS-uri sunt considerate conectate dacă există o linie între două rutere de graniță din fiecare. Dat fiind interesul special al BGP-ului pentru traficul în tranzit, rețelele sunt grupate în trei categorii. Prima categorie este cea a **rețelelor ciot (stub networks)**, care au doar o conexiune la graful BGP. Acestea nu pot fi folosite pentru traficul în tranzit pentru că nu este nimeni la capătul celălalt. Apoi vin **rețelele multiconectate**. Acestea pot fi folosite pentru traficul în tranzit, cu excepția a ceea ce ele refuză. În final, sunt **rețele de tranzit**, cum ar fi coloanele vertebrale, care sunt doritoare să manevreze pachetele altora, eventual cu unele restricții, și de obicei pentru o plată.

Perechile de rutere BGP comunică între ele stabilind conexiuni TCP. Operarea în acest mod oferă comunicație sigură și ascunde toate detaliile rețelelor traversate.

BGP este la bază un protocol bazat pe vectori distanță, dar destul de diferit de majoritatea celor-lalte cum ar fi RIP. În loc să mențină doar costul până la fiecare destinație, fiecare ruter BGP memoră calea exactă folosită. Similar, în loc să transmită periodic fiecărui vecin costul său estimat către fiecare destinație posibilă, fiecare ruter BGP comunică vecinilor calea exactă pe care o folosește.

Ca exemplu, să considerăm ruterele BGP din fig. 5-67(a). În particular, să considerăm tabela de dirijare a lui *F*. Să presupunem că el folosește calea *FGCD* pentru a ajunge la *D*. Când vecinii îi dau

informațiile de dirijare, ei oferă căile lor complete, ca în fig. 5-67(b) (pentru simplitate, este arătată doar destinația *D*).



**Fig. 5-67.** (a) O mulțime de rutere BGP. (b) Informația trimisă lui *F*.

După ce sosesc toate căile de la vecini, *F* le examinează pentru a vedea care este cea mai bună. El elimină imediat căile de la *I* și *E*, din moment ce aceste căi trec chiar prin *F*. Alegerea este apoi între *B* și *G*. Fiecare ruter BGP conține un modul care examinează căile către o destinație dată și le atribuie scoruri, întorcând, pentru fiecare cale, o valoare pentru „distanță” către acea destinație. Orice cale care violează o constrângere politică primește automat un scor infinit. Apoi, ruterul adoptă calea cu cea mai scurtă distanță. Funcția de acordare a scorurilor nu este parte a protocolului BGP și poate fi orice funcție doresc administratorii de sistem.

BGP rezolvă ușor problema numără-la-infinit care chinuie alți algoritmi bazati pe vectori distanță. De exemplu, să presupunem că *G* se defectează sau că linia *FG* cade. Atunci *F* primește căi de la ceilalți trei vecini rămași. Aceste rute sunt *BCD*, *IFGCD* și *EFGCD*. El poate observa imediat că ultimele două căi sunt inutile, din moment ce trec chiar prin *F*, așa că alege *FBCD* ca nouă sa cale. Alți algoritmi bazati pe vectori distanță fac de multe ori alegerea greșită, pentru că ei nu pot spune care din vecinii lor au căi independente către destinație și care nu. Definirea BGP-ului se găsește în RFC 1771 și 1774.

### 5.6.6 Trimiterea multiplă în Internet

Comunicația IP normală este între un emițător și un receptor. Cu toate acestea, pentru unele aplicații, este util ca un proces să fie capabil să trimită simultan unui număr mare de receptori. Exemple sunt actualizarea bazelor de date distribuite multiple, transmiterea cotărilor de la bursă mai multor agenți de bursă, tratarea convorbirilor telefonice de tipul conferințelor digitale (așadar cu mai mulți participanți).

IP-ul suportă trimiterea multiplă (multicast), folosind adrese de clasă D. Fiecare adresă de clasă D identifică un grup de gazde. Pentru identificarea grupurilor sunt disponibili douăzeci și opt de biți, așa încât pot exista în același moment peste 250 milioane de grupuri. Când un proces trimit un pachet unei adrese de clasă D, se face cea mai bună încercare pentru a-l livra tuturor membrilor grupului adresat, dar nu sunt date garanții. Unii membri pot să nu primească pachetul.

Sunt suportate două tipuri de adrese de grup: adrese permanente și adrese temporare. Un grup permanent există întotdeauna și nu trebuie configurat. Fiecare grup permanent are o adresă de grup permanentă. Câteva exemple de adrese de grup permanente sunt:

- 224.0.0.1 Toate sistemele dintr-un LAN.
- 224.0.0.2 Toate ruterele dintr-un LAN.
- 224.0.0.5 Toate ruterele OSPF dintr-un LAN.
- 224.0.0.6 Toate ruterele desemnate dintr-un LAN.

Grupurile temporare trebuie create înainte de a fi utilizate. Un proces poate cere gazdei sale să intre într-un anume grup. De asemenea, el poate cere gazdei sale să părăsească grupul. Când ultimul proces părăsește un grup, acel grup nu mai este prezent pe calculatorul său gazdă. Fiecare gazdă memorează căror grupuri aparțin la un moment dat procesele sale.

Trimiterea multiplă este implementată de rutere speciale de trimitere multiplă, care pot sau nu coabita cu ruterele standard. Cam o dată pe minut, fiecare ruter de trimitere multiplă face o trimitere multiplă hardware (la nivel legătură de date) pentru gazdele din LAN-ul său (adresa 224.0.0.1), cerându-le să raporteze căror grupuri aparțin proceselor lor la momentul respectiv. Fiecare gazdă trimite înapoi răspunsuri pentru toate adresele de clasă D de care este interesată.

Aceste pachete de întrebare și răspuns folosesc un protocol numit **IGMP (Internet Group Management Protocol)** - Protocol de gestiune a grupurilor Internet), care se asemănă întrucâtva cu ICMP. El are numai două tipuri de pachete: întrebare și răspuns, fiecare cu un format fix, simplu, care conține unele informații de control în primul cuvânt al câmpului informație utilă și o adresă de clasă D în al doilea cuvânt. El este descris în RFC 1112.

Dirijarea cu trimitere multiplă este realizată folosind arbori de acoperire. Fiecare ruter de dirijare multiplă schimbă informații cu vecinii săi, folosind un protocol modificat bazat pe vectori distanță cu scopul ca fiecare să construiască pentru fiecare grup un arbore de acoperire care să acopere toți membrii grupului. Pentru a elimina ruterele și rețelele neinteresante de anumite grupuri, se utilizează diverse optimizări de reducere a arborelui. Pentru evitarea deranjării nodurilor care nu fac parte din arborele de acoperire, protocolul folosește intensiv trecerea prin tunel.

### 5.6.7 IP mobil

Mulți utilizatori ai Internet-ului au calculatoare portabile și vor să rămână conectați la Internet atunci când vizitează un sit Internet aflat la distanță, și chiar și pe drumul dintre cele două. Din nevoie, sistemul de adresare IP face lucrul la depărtare de casă mai ușor de zis decât de făcut. În această secțiune vom examina problema și soluția. O descriere mai detaliată este dată în (Perkins, 1998a).

Problema apare chiar în schema de adresare. Fiecare adresă IP conține un număr de rețea și un număr de gazdă. De exemplu, să considerăm mașina cu adresa IP 160.80.40.20/16. Partea 160.80 indică numărul de rețea (8272 în notație zecimală). Ruterele din toată lumea au tabele de dirijare care spun ce linie se folosește pentru a ajunge la rețeaua 160.80. Oricând vine un pachet cu adresa IP destinație de forma 160.80.xxx.yyy, pachetul pleacă pe respectiva linie.

Dacă, dintr-o dată, mașina cu adresa respectivă este transferată într-un alt loc din Internet, pachetele vor continua să fie dirijate către LAN-ul (sau ruterul) de acasă. Proprietarul nu va mai primi poștă electronică și aşa mai departe. Acordarea unei noi adrese IP mașinii, adresă care să corespundă cu noua sa locație, nu este atractivă pentru că ar trebui să fie informate despre schimbare un mare număr de persoane, programe și baze de date.

O altă abordare este ca ruterele să facă dirijarea folosind adresa IP completă, în loc să folosească numai adresa rețelei. Cu toate acestea, această strategie ar necesita ca fiecare ruter să aibă milioane de intrări în tabele, la un cost astronomic pentru Internet.

Când oamenii au început să ceară posibilitatea de a-și conecta calculatoarele portabile oriunde să ar duce, IETF a constituit un Grup de Lucru pentru a găsi o soluție. Grupul de Lucru a formulat rapid un număr de obiective considerate necesare în orice soluție. Cele majore au fost:

1. Fiecare gazdă mobilă trebuie să fie capabilă sa folosească adresa sa IP de baza oriunde.
2. Nu au fost permise schimbări de programe pentru gazdele fixe.
3. Nu au fost permise schimbări pentru programele sau tabelele ruterelor.
4. Cele mai multe pachete pentru gazdele mobile nu ar trebui să facă ocoluri pe drum.
5. Nu trebuie să apară nici o suprasolicitare când o gazdă mobilă este acasă.

Soluția aleasă este cea descrisă în secțiunea 5.2.8. Pentru a o recapitula pe scurt, fiecare sit care dorește să permită utilizatorilor săi să se deplaseze trebuie să asigure un agent local. Fiecare sit care dorește să permită accesul vizitatorilor trebuie să creeze un agent pentru străini. Când o gazdă mobilă apare într-un sit străin, ea contactează gazda străină de acolo și se înregistrează. Gazda străină contactează apoi agentul local al utilizatorului și îi dă o **adresă a intermediarului**, în mod normal adresa IP proprie a agentului pentru străini.

Când un pachet ajunge în LAN-ul de domiciliu al utilizatorului, el vine la un ruter atașat la LAN. Apoi ruterul încearcă să localizeze gazda în mod ușor, prin difuzarea unui pachet ARP întrebând, de exemplu: „Care este adresa Ethernet a lui 160.80.40.20?” Agentul local răspunde la această întrebare dând propria adresă Ethernet. Apoi ruterul trimite pachetele pentru 160.80.40.20 la agentul local. El, în schimb, le trimite prin tunel la adresa intermediarului prin încapsularea lor în câmpul informație utilă al unui pachet IP adresat agentului pentru străini. După aceasta, agentul pentru străini le desface și le livrează la adresa de nivel legătură de date a gazdei mobile. În plus, agentul local dă emițătorului adresa intermediarului, așa încât viitoarele pachete pot fi trimise prin tunel direct la agentul pentru străini. Această soluție răspunde tuturor cerințelor expuse mai sus.

Probabil că merită menționat un mic amănunt. În momentul în care gazda mobilă se mută, probabil că ruterul are adresa ei Ethernet (care în curând va fi invalidă) memorată în memoria ascunsă. Pentru a înlătura această adresă Ethernet cu cea a agentului local, se folosește un truc numit **ARP gratuit**. Acesta este un mesaj special, nesolicitat, către ruter pe care îl determină să schimbe o anumită intrare din memoria ascunsă, în acest caz cea a gazdei mobile care urmează să plece. Când, mai târziu, gazda mobilă se întoarce, este folosit același truc pentru a actualiza din nou memoria ascunsă a ruterului.

Nu există nimic în proiect care să împiedice o gazdă mobilă să fie propriul său agent pentru străini, dar această abordare funcționează numai dacă gazda mobilă (în postura sa de agent pentru străini) este conectată logic în Internet la situl său curent. De asemenea, ea trebuie să fie capabilă să obțină pentru folosire o adresă (temporară) de intermediar. Acea adresă IP trebuie să aparțină LAN-ului în care este atașată în mod curent.

Soluția IETF pentru gazde mobile rezolvă un număr de alte probleme care nu au fost menționate până acum. De exemplu, cum sunt localizați agentii? Soluția este ca fiecare agent să-și difuzeze periodic adresa și tipul de serviciu pe care dorește să-l ofere (de exemplu, agent local, pentru străini sau amândouă). Când o gazdă mobilă ajunge undeva, ea poate asculta așteptând aceste difuzări, numite **anunțuri**. Ca o alternativă, ea poate difuza un pachet prin care își anunță sosirea și să spere că agentul pentru străini local îi va răspunde.

O altă problemă care a trebuit rezolvată este cum să se trateze gazdele mobile nepoliticoase, care pleacă fără să spună la revedere. Soluția este ca înregistrarea să fie valabilă doar pentru un interval de timp fixat. Dacă nu este reîmprospătată periodic, ea expiră și ca urmare gazda străină poate să-și curețe tabelele.

O altă problemă este securitatea. Când un agent local primește un mesaj care-i cere să fie amabil să retrimită toate pachetele Robertei la o anume adresă IP, ar fi mai bine să nu se supună decât dacă este convins că Roberta este sursa acestei cereri și nu altcineva încercând să se dea drept ea. În acest scop sunt folosite protocole criptografice de autentificare. Vom studia asemenea protocole în cap. 8.

Un punct final adresat de Grupul de Lucru se referă la nivelurile de mobilitate. Să ne imaginăm un avion cu o rețea Ethernet la bord folosită de către calculatoarele de navigare și de bord. În această rețea Ethernet există un ruter standard, care discută cu Internet-ul cablat de la sol printr-o legătură radio. Într-o bună zi, unui director de marketing ișteț îi vine ideea să instaleze conexiunea Ethernet în toate brațele fotoliilor, astfel încât și pasagerii cu gazde mobile să se poată conecta.

Acum avem două niveluri de mobilitate: calculatoarele proprii ale aeronavei, care sunt staționare în raport cu rețeaua Ethernet și calculatoarele pasagerilor, care sunt mobile în raport cu ea. În plus, ruterul de la bord este mobil în raport cu ruterele de la sol. Mobilitatea în raport cu un sistem care este la rândul său mobil este tratată folosind recursiv tunele.

### 5.6.8 IPv6

În timp ce CIDR și NAT îi mai pot acorda câțiva ani, toată lumea își dă seama că zilele IP-ului în forma curentă (IPv4) sunt numărate. În plus față de aceste probleme tehnice, există un alt aspect întrețărit în fundal. La începuturile sale, Internet-ul a fost folosit în mare măsură de universități, industria de vârf și de guvernul Statelor Unite (în mod special de Departamentul Apărării). O dată cu explozia interesului față de Internet începând de la mijlocul anilor 1990, a început să fie utilizat de un grup diferit de persoane, în special persoane cu cerințe diferite. Pe de o parte, numeroase persoane cu calculatoare portabile fără fir îl folosesc pentru a ține legătura cu baza de acasă. Pe de altă parte, o dată cu iminenta convergență a industriilor calculatoarelor, comunicațiilor și a distracțiilor, s-ar putea să nu mai fie mult până când fiecare telefon sau televizor din lume va fi un nod Internet, producând un miliard de mașini folosite pentru audio și video la cerere. În aceste condiții, a devenit clar că IP-ul trebuie să evolueze și să devină mai flexibil.

Observând aceste probleme la orizont, IETF a început să lucreze în 1990 la o nouă versiune de IP, una care să nu își epuizeze niciodată adresele, să rezolve o gamă largă de alte probleme și să fie totodată mai flexibilă și mai eficientă. Obiectivele majore au fost:

1. Să suporte miliarde de gazde, chiar cu alocare ineficientă a spațiului de adrese.
2. Să reducă dimensiunea tabelelor de dirijare.
3. Să simplifice protocolul, pentru a permite ruterelor să proceseze pachetele mai rapid.
4. Să asigure o securitate mai bună (autentificare și confidențialitate) față de IP-ul curent.
5. Să acorde o mai mare atenție tipului de serviciu, în special pentru datele de timp real.
6. Să ajute trimiterea multiplă, permitând specificarea de domenii.
7. Să creeze condițiile pentru ca o gazdă să poată migra fără schimbarea adresei sale.
8. Să permită evoluția protocolului în viitor.
9. Să permită coexistența noului și vechiului protocol pentru câțiva ani.

Pentru a găsi un protocol care să îndeplinească toate aceste cerințe, IETF a emis o cerere de propuneri și discuții în RFC 1550. Au fost primite douăzeci și unu de răspunsuri, nu toate din ele propuneri complete. Până în decembrie 1992, au ajuns pe masa discuțiilor șapte propuneri serioase. Ele variau de la efectuarea de mici cârpeți la IP până la renunțarea completă la el și înlocuirea cu un protocol complet nou.

O propunere a fost folosirea TCP peste CLNP, care cu cei 160 de biți de adresă ai săi ar fi oferit spațiu de adrese suficient pentru totdeauna și ar fi unificat două protocole majore de nivel rețea. Cu toate acestea, multe persoane au simțit că aceasta ar fi fost o acceptare a faptului că, de fapt, a fost făcut ceva chiar bine în lumea OSI, o afirmație considerată incorectă din punct de vedere politic în cercurile Internet. CLNP a fost modelat apropiat de IP, așa încât cele două nu sunt chiar atât de diferite. De fapt, protocolul care a fost ales în final diferă de IP cu mult mai mult decât diferă CLNP. O altă lovitură împotriva CLNP a fost slabul suport pentru tipuri de servicii, ceva necesar pentru transmiterea eficientă de multimedia.

Trei din cele mai bune propuneri au fost publicate în *IEEE Network* (Deering, 1993; Francis, 1993 și Katz și Ford, 1993). După multe discuții, revizii și manevre de culise, a fost selectată o versiune combinată modificată a propunerilor lui Deering și Francis, până atunci numită **SIPP (Simple Internet Protocol Plus - Protocol simplu, îmbunătățit, pentru Internet)** și i s-a dat numele de **IPv6**.

IPv6 îndeplinește obiectivele destul de bine. El menține caracteristicile bune ale IP-ului, le elimină sau atenuază pe cele rele și adaugă unele noi acolo unde este nevoie. În general, IPv6 nu este compatibil cu IPv4, dar el este compatibil cu celealte protocole Internet auxiliare, incluzând TCP, UDP, ICMP, IGMP, OSPF, BGP și DNS, cîteodată fiind necesare mici modificări (majoritatea pentru a putea lucra cu adrese mai lungi). Principalele trăsături ale IPv6 sunt discutate mai jos. Mai multe informații despre el pot fi găsite în RFC 2460 până la RFC 2466.

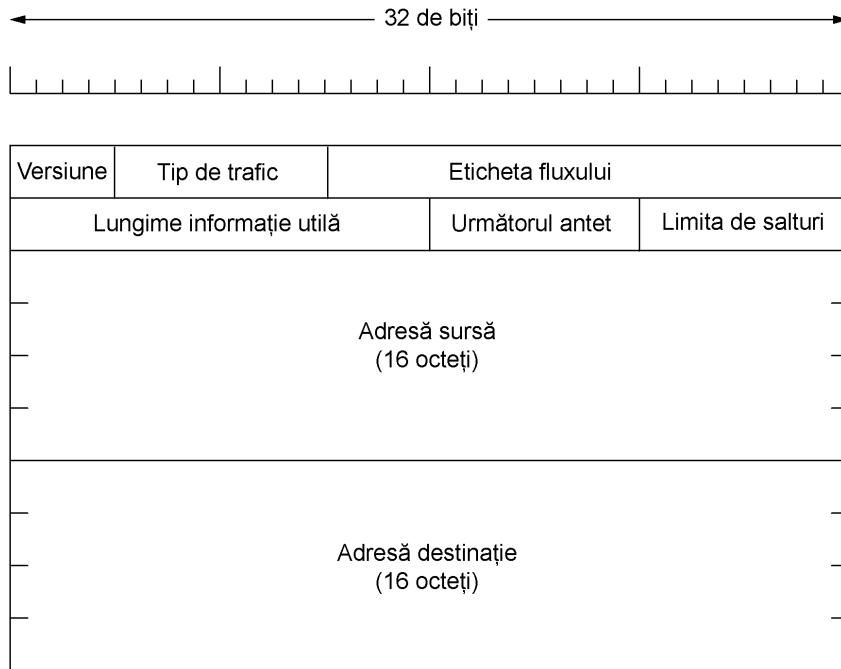
În primul rând și cel mai important, IPv6 are adrese mai lungi decât IPv4. Ele au o lungime de 16 octeți, ceea ce rezolvă problema pentru a cărei soluționare a fost creat IPv6: să furnizeze o sursă efectiv nelimitată de adrese Internet. În curînd vom spune mai multe despre adrese.

A doua mare îmbunătățire a lui IPv6 este simplificarea antetului. El conține numai 7 câmpuri (față de 13 în IPv4). Această schimbare permite ruterelor să prelucreze pachetele mai rapid, îmbunătățind astfel productivitatea și întărzierea. De asemenea, vom discuta în curînd și antetul.

A treia mare îmbunătățire a fost suportul mai bun pentru opțiuni. Această schimbare a fost esențială în noul antet, deoarece câmpurile care erau necesare anterior sunt acum opționale. În plus, modul în care sunt reprezentate opțiunile este diferit, ușurînd ruterelor saltul peste opțiunile care nu le sunt destinate. Această caracteristică accelerează timpul de prelucrare a pachetelor.

Un al patrulea domeniu în care IPv6 reprezintă un mare progres este în securitate. IETF a avut porția sa de povești de ziar despre copii precoce de 12 ani care își folosesc calculatoarele personale pentru a sparge bănci sau baze militare în tot Internet-ul. A existat un sentiment puternic că ar trebui făcut ceva pentru a îmbunătăți securitatea. Autentificarea și confidențialitatea sunt trăsături cheie ale noului IP. Ulterior ele au fost adaptate și în IPv4, astfel că în domeniul securității diferențele nu mai sunt așa de mari.

În final, a fost acordată o mai mare atenție calității serviciilor. În trecut s-au făcut eforturi, fără prea mare tragere de inimă, dar acum, o dată cu creșterea utilizării multimedia în Internet, presiunea este și mai mare.



**Fig. 5-68.** Antetul fix IPv6 (obligatoriu).

### Antetul principal IPv6

Antetul IPv6 este prezentat în fig. 5-68. Câmpul *Versiune* este întotdeauna 6 pentru IPv6 (și 4 pentru IPv4). În timpul perioadei de tranziție de la IPv4, care va lua probabil un deceniu, ruterele vor fi capabile să examineze acest câmp pentru a spune ce tip de pachet analizează. Ca un efect lateral, acest test irosește câteva instrucțiuni pe drumul critic, așa încât multe implementări vor încerca să-l evite prin folosirea unui câmp din antetul legăturii de date ca să diferențieze pachetele IPv4 de pachetele IPv6. În acest mod, pachetele pot fi transmise direct rutinei de tratare de nivel rețea corecte. Cu toate acestea, necesitatea ca nivelul legătură de date să cunoască tipurile pachetelor nivelului rețea contravine complet principiul de proiectare care spune că fiecare nivel nu trebuie să cunoască semnificația biților care îi sunt dați de către nivelul de deasupra. Discuțiile dintre taberele „Fă-o corect” și „Fă-o repede” vor fi, fără îndoială, lungi și virulente.

Câmpul *Tip de trafic* (*Traffic class*) este folosit pentru a distinge între pachetele care au diverse cerințe de livrare în timp real. Un câmp cu acest scop a existat în IP de la început, dar a fost implementat sporadic de către rutere. În acest moment se desfășoară experimente pentru a determina cum poate fi utilizat cel mai bine pentru transmisii multimedia.

Câmpul *Eticheta fluxului* este încă experimental, dar va fi folosit pentru a permite unei surse și unei destinații să stabilească o pseudo-conexiune cu proprietăți și cerințe particulare. De exemplu, un șir de pachete de la un proces de pe o anumită gazdă sursă către un anumit proces pe o anumită gazdă destinație poate avea cerințe de întârziere stricte și din acest motiv necesită capacitate de transmisie rezervată. Fluxul poate fi stabilit în avans și poate primi un identificator. Când apare un pachet cu o *Etichetă a fluxului* diferită de zero, toate ruterele pot să o caute în tabelele interne pentru

a vedea ce tip de tratament special necesită. Ca efect, fluxurile sunt o încercare de a combina două moduri: flexibilitatea unei subretele cu datagrame și garanțile unei subretele cu circuite virtuale.

Fiecare flux este desemnat de adresa sursă, adresa destinație și numărul de flux, aşa încât, între o pereche de adrese IP pot exista mai multe fluxuri active în același timp. De asemenea, în acest mod, chiar dacă două fluxuri venind de la gazde diferite, dar cu același număr de flux trec prin același ruter, ruterul va fi capabil să le separe folosind adresele sursă și destinație. Se așteaptă ca numerele de flux să fie alese aleator, în loc de a fi atribuite secvențial începând cu 1, pentru că se așteaptă ca ruterele să le folosească în tabele de dispersie.

Câmpul *Lungimea informației utile* spune câți octeți urmează după antetul de 40 de octeți din fig. 5-68. Numele a fost schimbat față de câmpul *Lungime totală* din IPv4 deoarece semnificația este ușor modificată: cei 40 de octeți nu mai sunt parte a lungimii (aşa cum erau înainte).

Câmpul *Antetul următor* dă de gol proiectanții. Motivul pentru care antetul a putut fi simplificat este că există antete de extensie suplimentare (optionale). Acest câmp spune care din cele șase antete (actuale) de extensie, dacă există vreunul, urmează după cel curent. Dacă acest antet este ultimul antet IP, câmpul *Antetul următor* spune căruia tip de protocol (de exemplu TCP, UDP) i se va transmite pachetul.

Câmpul *Limita salturilor* este folosit pentru a împiedica pachetele să trăiască veșnic. El este, în practică, identic cu câmpul *Timp de viață* din IPv4, și anume un câmp care este decrementat la fiecare salt dintr-o rețea în alta. În teorie, în IPv4 era un timp în secunde, dar nici un ruter nu-l folosea în acest mod, aşa încât numele a fost modificat pentru a reflecta modul în care este de fapt folosit.

Apoi urmează câmpurile *Adresă sursă* și *Adresă destinație*. Propunerea originală a lui Deering, SIP, folosea adrese de 8 octeți, dar în timpul procesului de evaluare, multe persoane au simțit că adresele de 8 octeți s-ar putea epuiza în câteva decenii, în timp ce adresele de 16 octeți nu s-ar epuiza niciodată. Alte persoane au argumentat că 16 octeți ar fi un exces, în timp ce alții încurajau folosirea adreselor de 20 de octeți pentru a fi compatibile cu protocolul datagramă OSI. O altă grupare dorea adresa de dimensiune variabilă. După multe discuții, s-a decis că adresele cu lungime fixă de 16 octeți sunt cel mai bun compromis.

Pentru scrierea adreselor de 16 octeți a fost inventată o nouă notație. Ele sunt scrise ca opt grupuri de câte patru cifre hexazecimale cu semnul : (două puncte) între grupuri, astfel:

8000:0000:0000:0000:0123:4567:89AB:CDEF

Din moment ce multe adrese vor avea multe zerouri în interiorul lor, au fost autorizate trei optimizări. Mai întâi, zerourile de la începutul unui grup pot fi omise, astfel încât 0123 poate fi scris ca 123. În al doilea rând, unul sau mai multe grupuri de 16 zerouri pot fi înlocuite de o pereche de semne două puncte (:). Astfel, adresa de mai sus devine acum

8000::123:4567:89AB:CDEF

În final, adresele IPv4 pot fi scrise ca o pereche de semne două puncte și un număr zecimal în vechea formă cu punct, de exemplu

::192.31.20.46

Probabil că nu este necesar să fim atât de expliciti asupra acestui lucru, dar există o mulțime de adrese de 16 octeți. Mai exact, sunt  $2^{128}$  adrese, care reprezintă aproximativ  $3 \times 10^{38}$ . Dacă întreaga planetă, pământ și apă, ar fi acoperite cu calculatoare, IPv6 ar permite  $7 \times 10^{23}$  adrese IP pe metru pătrat. Studenții de la chimie vor observa că acest număr este mai mare decât numărul lui Avogadro. Deși nu a

există intenția de a da fiecărei molecule de pe suprafața planetei adresa ei IP, nu suntem chiar așa de departe de aceasta.

În practică, spațiul de adrese nu va fi folosit eficient, aşa cum nu este folosit spațiul de adrese al numerelor de telefon (prefixul pentru Manhattan, 212, este aproape plin, dar cel pentru Wyoming, 307, este aproape gol). În RFC 3194, Durand și Huitema a calculat că, folosind ca referință alocarea numerelor de telefon, chiar și în cel mai pesimist scenariu, vor fi totuși mult peste 1000 de adrese IP pe metru pătrat de suprafață planetară (pământ sau apă). În orice scenariu credibil, vor fi trilioane de adrese pe metru pătrat. Pe scurt, pare improbabil că vom epuiza adresele în viitorul previzibil.

Este instructiv să comparăm antetul IPv4 (fig. 5-53) cu antetul IPv6 (fig. 5-68) pentru a vedea ce a fost eliminat în IPv6. Câmpul *IHL* a dispărut pentru că antetul IPv6 are o lungime fixă. Câmpul *Protocol* a fost scos pentru că în câmpul *Antetul următor* se indică ce urmează după ultimul antet IP (de exemplu, un segment TCP sau UDP).

Toate câmpurile referitoare la fragmentare au fost eliminate, deoarece IPv6 are o abordare diferită a fragmentării. Pentru început, toate gazdele și ruterelor care sunt conforme cu IPv6 trebuie să determine dinamic mărimea datagramei care va fi folosită. Această regulă face ca, de la început, fragmentarea să fie mai puțin probabilă. De asemenea minimul a fost mărit de la 576 la 1280 pentru a permite date de 1024 de octeți și mai multe antete. În plus, când o gazdă trimite un pachet IPv6 care este prea mare, ruterul care este incapabil să îl retransmită trimite înapoi un mesaj de eroare în loc să fragmenteze pachetul. Acest mesaj de eroare îi spune gazdei să spargă toate pachetele viitoare către acea destinație. Este mult mai eficient să obligi gazdele să transmită de la bun început pachete corecte dimensionale, decât să obligi ruterelor să le fragmenteze din mers.

În sfârșit, câmpul *Sumă de control* este eliminat deoarece calculul acestuia reduce mult performanțele. Datorită rețelelor fiabile folosite acum, combinate cu faptul că nivelurile de legătură de date și de transport au în mod normal propriile sume de control, valoarea a încă unei sume de control nu merita prețul de performanță cerut. Eliminarea tuturor acestor caracteristici a avut ca rezultat un protocol de nivel rețea simplu și eficient. Astfel, obiectivul lui IPv6 – un protocol rapid, dar flexibil, cu o bogăție de spațiu de adrese – a fost atins prin acest proiect.

### Antete de extensie

Câteva din câmpurile care lipsesc sunt încă necesare ocazional, astfel încât IPv6 a introdus conceptul de **antet de extensie** (optional). Aceste antete pot fi furnizate pentru a oferi informații suplimentare, dar codificate într-un mod eficient. În prezent sunt definite șase tipuri de antete de extensie, prezентate în fig. 5-69. Fiecare este optional, dar dacă sunt prezente mai multe, ele trebuie să apară imediat după antetul fix și, preferabil, în ordinea prezentată.

Antet de extensie	Descriere
Optiuni salt-după-salt	Diverse informații pentru rutere
Optiuni pentru destinație	Informații suplimentare pentru destinație
Dirijare	Calea, parțială sau totală, de urmat
Fragmentare	Gestiunea fragmentelor datagramelor
Autentificare	Verificarea identității emițătorului
Informație de siguranță criptată	Informații despre conținutul criptat

Fig. 5-69. Antetele de extensie IPv6.

Unele dintre antete au un format fix; altele conțin un număr variabil de câmpuri de lungime variabilă. Pentru acestea, fiecare element este codificat ca un tuplu (Tip, Lungime, Valoare). *Tipul* este

un câmp de 1 octet care spune ce opțiune este aceasta. Valorile *Tipului* au fost alese astfel, încât primii 2 biți spun ruterelor care nu știu cum să proceseze opțiunea ce anume să facă. Variantele sunt: sărirea opțiunii; eliminarea pachetului; eliminarea pachetului și trimiterea înapoi a unui pachet ICMP; la fel ca mai înainte, doar că nu se trimit pachete ICMP pentru adrese de trimitere multiplă (pentru a preveni ca un pachet eronat de multicast să genereze milioane de răspunsuri ICMP).

Câmpul *Lungime* este de asemenea un câmp de lungime 1 octet. El precizează cât de lungă este valoarea (de la 0 la 255). *Valoarea* este orice informație necesară, până la 255 octeți.

Antetul salt-după-salt este folosit pentru informații ce trebuie examineate de toate ruterele de pe cale. Până acum a fost definită o opțiune: suportul pentru datagrame ce depășesc de 64K. Formatul acestui antet este prezentat în fig. 5-70. Atunci când este folosit, câmpul *Lungimea informației utile* din antetul fix este zero.

Antetul următor	0	194	4
Lungimea informației utile foarte mari			

Fig. 5-70. Antetul de extensie salt-după-salt pentru datagrame mari (jumbograme).

Ca toate antetele de extensie, acesta începe cu un octet care spune ce tip de antet este următorul. Acest octet este urmat de unul care spune cât de lung este antetul salt-după-salt în octeți, excludând primii 8 octeți, care sunt obligatorii. Toate extensiile încep la fel.

Următorii 2 octeți arată că această opțiune definește dimensiunea datagramei (codul 194) ca un număr de 4 octeți. Ultimii 4 octeți dau dimensiunea datagramei. Dimensiunile mai mici de 65.536 nu sunt permise și au ca rezultat eliminarea pachetului de către primul ruter, care va trimite înapoi un mesaj ICMP de eroare. Datagramele care folosesc acest antet de extensie sunt numite **jumbograme**. Folosirea jumbogramelor este importantă pentru aplicațiile supercalculatoarelor care trebuie să transfere gigaocteți de date eficient prin intermediul Internet-ului.

Antetul opțiunilor pentru destinație este prevăzut pentru câmpuri care trebuie interpretate numai de către gazda destinație. În versiunea inițială a IPv6 singura opțiune definită este cea de completare a acestui antet până la un multiplu de 8 octeți, aşa că pentru început nu va fi folosit. El a fost inclus pentru a asigura posibilitatea ca un nou software al ruterului sau al gazdei să îl poate trata, în cazul în care cineva, cândva, se gândește la o opțiune pentru destinație.

Antetul de dirijare enumera unul sau mai multe rutere care trebuie să fie vizitate pe calea spre destinație. Este foarte asemănător cu dirijarea aproximativă din IPv4 prin aceea că toate adresele listate trebuie vizitate în ordine, dar între acestea pot fi vizitate alte rutere nelistate. Formatul antetului de dirijare este prezentat în fig. 5-71.

Antetul următor	Lungimea antetului de extensie	Tipul de dirijare	Segmente rămase
Date specifice tipului			

Fig. 5-71. Antetul de extensie pentru dirijare.

Primii 4 octeți ai antetului de extensie de dirijare conțin patru întregi de 1 octet. Câmpurile *Antet următor* și *Lungimea antetului* extensie au fost descrise anterior. Câmpul *Tipul dirijării* precizează formatul părții rămase din header. Tipul 0 arată că după primul cuvânt urmează un cuvânt rezervat pe 32 de biți, urmat de un număr de adrese IPv6. În viitor, în funcție de necesități, ar putea fi inventate alte tipuri. În final, câmpul *Segmente rămase* reține câte adrese din listă nu au fost vizitate și este decrementat de fiecare dată când este vizitată o adresă. Atunci când se ajunge la 0 pachetul este pe cont propriu, fără nici o indicație referitoare la ruta de urmat. De obicei, în acest punct este atât de aproape de destinație încât calea cea mai bună este evidentă.

Antetul fragment tratează fragmentarea într-un mod similar cu cel al IPv4. Antetul menține identificatorul datagramei, numărul de fragment și un bit care spune dacă mai urmează fragmente. În IPv6, spre deosebire de IPv4, numai gazda sursă poate fragmenta un pachet. Ruterele de pe cale nu pot face acest lucru. Deși această schimbare este o rupere filozofică majoră cu trecutul, ea simplifică munca ruterelor și permite ca dirijarea să se facă mai rapid. Așa cum s-a menționat mai sus, dacă un ruter este confruntat cu un pachet care este prea mare, el elimină pachetul și trimite un pachet ICMP înapoi la sursă. Această informație îi permite gazdei sursă să fragmenteze pachetul în bucăți mai mici folosind acest antet și apoi să reîncerce.

Antetul de autentificare oferă un mecanism prin care receptorul unui mesaj poate fi sigur de cel care l-a trimis. Informația utilă de siguranță criptată face posibilă criptarea conținutului unui pachet, astfel încât doar receptorul căruia îi este destinat poate să-l citească. Pentru a-și realiza misiunea acestei antete folosesc tehnici criptografice.

### Controverse

Dat fiind procesul de proiectare deschisă și opiniile ferm susținute ale multora dintre persoanele implicate, nu ar trebui să fie o surpriză că multe din deciziile luate pentru IPv6 au fost foarte controversate. În cele ce urmează vom rezuma câteva dintre acestea. Pentru toate amănuntele tăioase, vezi RFC-urile.

Am menționat deja disputa legată de lungimea adresei. Rezultatul a fost un compromis: adrese de lungime fixă de 16 octeți.

O altă dispută s-a dus în jurul lungimii câmpului *Limita salturilor*. O tabără a simțit că limitarea numărului maxim de salturi la 255 (implicită în cazul folosirii unui câmp de 8 biți) ar fi o greșeală grosolană. Până la urmă, căi de 32 de salturi sunt obișnuite în zilele noastre, iar peste 10 ani pot fi obișnuite căi mult mai lungi. Aceste persoane au argumentat că folosirea unui spațiu de adrese enorm a fost clarvizibile, dar folosirea unui contor minuscul de salturi a fost miopie. După părerea lor, cel mai mare păcat pe care îl poate comite un informatician este să ofere prea puțini biți într-un loc.

Răspunsul a fost că pot fi aduse argumente pentru lărgirea fiecărui câmp, conducând la un antet umflat. De asemenea, funcția câmpului *Limita salturilor* este de a împiedica hoinăreală pachetelor pentru un timp îndelungat și 65.536 de salturi este mult prea mult. În cele din urmă, pe măsură ce Internet-ul crește, se vor construi din ce în ce mai multe legături de mare distanță, făcând posibilă ajungerea dintr-o țară în alta în cel mult o jumătate de duzină de salturi. Dacă este nevoie de mai mult de 125 de salturi pentru a ajunge de la sursă sau destinație la porțile lor internaționale, ceva este în neregulă cu coloanele vertebrale naționale. Adeptii celor 8 biți au câștigat această luptă.

O altă problemă spinoasă a fost dimensiunea maximă a pachetului. Comunitatea supercalculatoarelor a dorit pachete mai mari de 64 KB. Când un supercalculator începe să transfere, aceasta înseamnă într-adevăr lucru serios și nu dorește să fie întrerupt după fiecare 64KB. Argumentul împotriva

va pachetelor mari este că dacă un pachet de 1 MB ajunge la o linie T1 de 1.5 Mbps, acel pachet va monopoliza linia pentru mai mult de 5 secunde, producând o întârziere semnificativă pentru utilizatorii interactivi care partajează linia. Aici s-a ajuns la un compromis: pachetele normale au fost limitate la 64 KB, dar antetul de extensie salt-după-salt poate fi folosit pentru a permite jumbograme.

Un al treilea punct fierbinte a fost eliminarea sumei de control IPv4. Unele persoane au asimilat această mutare cu eliminarea frânelor de la mașină. Făcând acest lucru, mașina devine mai ușoară, astfel încât poate merge mai repede, dar dacă intervine ceva neașteptat, apar probleme.

Argumentul împotriva sumei de control a fost că orice aplicație care are într-adevăr grija de integritatea datelor trebuie oricum să aibă o sumă de control la nivelul transport, aşa încât menținerea a încă o sumă în IP (în plus față de suma de control a nivelului legătură de date) este un exces. Mai mult, experiența a arătat că în IPv4 calculul sumei de control IP era foarte costisitoare. Tabăra împotriva sumei de control a învins de această dată, deci IPv6 nu are o sumă de control.

Gazdele mobile au fost de asemenea un punct de conflict. Dacă un calculator portabil face jumătate din ocolul lumii, poate continua el să opereze la destinație cu aceeași adresă IPv6 sau trebuie să folosească o schemă cu agenți locali și agenți pentru străini? De asemenea, gazdele mobile introduc asimetrii în sistemul de dirijare. Se poate foarte bine întâmpla ca un mic calculator mobil să audă semnalul puternic trimis de un ruter staționar, dar ruterul staționar nu poate auzi semnalul slab trimis de gazda mobilă. În consecință, unele persoane au dorit să includă în IPv6 suport explicit pentru gazde mobile. Acest efort a eşuat pentru că nu s-a putut ajunge la un consens pentru o propunere concretă.

Probabil că cea mai mare bătălie a fost pentru securitate. Toată lumea a fost de acord că este necesară. Războiul a fost pentru unde și cum. Mai întâi unde. Argumentul pentru plasarea la nivelul rețea este că devine un serviciu standard pe care toate aplicațiile îl pot folosi fără o planificare prealabilă. Argumentul contra este că, în general, aplicațiile cu adevărat sigure doresc cel puțin criptare capăt-la-capăt, în care aplicația sursă cripteză și aplicația destinație decripteză. Altfel, utilizatorul este la mila unor implementări potențial pline de pene a nivelurilor rețea, implementări asupra cărora nu are nici un control. Răspunsul la acest argument este că aceste aplicații pot să se abțină de la folosirea facilităților de securitate IP și să-și facă treaba ele însese. Replica la aceasta este că persoanele care nu au încredere că rețeaua face treaba cum trebuie, nu doresc să plătească prețul unor implementări de IP lente, greoaie, care au această facilitate, chiar dacă este dezactivată.

Un alt aspect al disputei unde să fie pusă securitatea este legat de faptul că multe țări (dar nu toate) au legi de export severe referitoare la criptografie. Unele, notabil în Franța și Irak, reduc în mare măsură folosirea internă a criptografiei, aşa încât oamenii nu pot avea secrete față de poliție. Ca rezultat, orice implementare de IP care utilizează un sistem criptografic suficient de puternic pentru a fi de mare valoare nu poate fi exportat din Statele Unite (și multe alte țări) clientilor din lumea întreagă. Necesitatea menținerii a două seturi de programe, unul pentru uz intern și altul pentru export, este un fapt ce întâmpină o opozitie viguroasă din partea firmelor de calculatoare.

Un punct asupra căruia nu au existat controverse este că nimeni nu se așteaptă ca Internet-ul bazat pe IPv4 să fie închis într-o duminică dimineață și să repornească ca un Internet bazat pe IPv6 luni dimineață. În schimb, vor fi convertite „insule” izolate de IPv6, initial comunicând prin tunele. Pe măsură ce insulele IPv6 cresc, ele vor fuziona în insule mai mari. Până la urmă, toate insulele vor fuziona și Internet-ul va fi convertit complet. Date fiind investiția masivă în rutere IPv4 în folosință curentă, procesul de conversie va dura probabil un deceniu. Din acest motiv s-a depus o enormă cantitate de efort pentru a asigura că această tranziție va fi cât mai puțin dureroasă posibil. Pentru mai multe informații despre IPv6, vezi (Loshin, 1999).

## 5.7 REZUMAT

Nivelul rețea furnizează servicii nivelului transport. El se poate baza fie pe circuite virtuale, fie pe datagrame. În ambele cazuri, principala sarcină este dirijarea pachetelor de la sursă la destinație. În subretelele bazate pe circuite virtuale, decizia de dirijare se ia atunci când este stabilit circuitul. În subretelele bazate pe datagrame decizia este luată pentru fiecare pachet.

În rețelele de calculatoare sunt folosiți mulți algoritmi de dirijare. Algoritmii statici includ dirijarea pe calea cea mai scurtă și inundarea. Algoritmii dinamici include dirijarea după vectorul distanțelor și dirijarea după starea legăturii. Majoritatea rețelelor actuale folosesc unul dintre acești algoritmi. Alte teme importante referitoare la dirijare sunt dirijarea ierarhică, dirijarea pentru gazde mobile, dirijarea pentru difuzare, dirijarea multidezinație și cea în rețele punct-la-punct.

Subretelele pot deveni cu ușurință congestionate, măringînd întârzierea și micșorând productivitatea pentru pachete. Proiectanții rețelelor încearcă să evite congestia printr-o proiectare adecvată. Tehnicile includ politica de retransmitere, folosirea memoriei ascunse, controlul fluxului și altele. Dacă apare congestia, ea trebuie să fie tratată. Pot fi trimise înapoi pachete de soc, încărcarea poate fi eliminată sau se pot aplica alte metode.

Următorul pas dincolo de simpla tratare a congestiei este încercarea de a se ajunge la calitatea promisă a serviciului. Metodele care pot fi folosite pentru aceasta includ folosirea zonelor tampon la client, modelarea traficului, rezervarea resurselor și controlul accesului. Abordările care au fost proiectate pentru o bună calitate a serviciului includ servicii integrate (ca RSVP), servicii diferențiate și MPLS.

Rețelele diferă prin multe caracteristici, așa că atunci când se conectează mai multe rețele, pot să apară probleme. Uneori problemele pot fi evitate prin trecerea prin tunel a unui pachet ce traversează o rețea ostilă, dar dacă rețeaua sursă și cea destinație diferă, această abordare eşuează. Atunci când rețele diferite au dimensiunile maxime ale pachetelor diferite, se poate produce fragmentarea.

Internet-ul posedă o mare varietate de protocoale legate de nivelul rețea. Acestea includ protocolul de transport al datelor, IP, dar și protocoalele de control ICMP, ARP și RARP și protocoalele de dirijare OSPF și BGP. Internet-ul va rămâne foarte repede fără adrese IP, așa că s-a dezvoltat o nouă versiune de IP, IPv6.

## 5.8 PROBLEME

1. Dați două exemple de aplicații pentru care este adecvat un serviciu orientat pe conexiune. Apoi dați două exemple pentru care un serviciu fără conexiuni este cel mai potrivit.
2. Există vreo situație în care un serviciu cu circuit virtual va (sau cel puțin ar putea) livra pachetele în altă ordine? Explicați.
3. Subretelele bazate pe datagrame dirijează fiecare pachet ca pe o unitate separată, independentă de toate celelalte. Subretelele bazate pe circuite virtuale nu trebuie să facă acest lucru, pentru că fiecare pachet de date urmează o cale predeterminată. Oare această observație înseamnă că

Aceste două programe (la fel ca și orice material referitor la această carte) pot fi luate de la adresa de Web a cărții

<http://www.prenhall.com/tanenbaum>

dând clic pe link-ul către situl Web de lângă fotografia copertăii. Ele pot fi descărcate și compilate pe orice sisteme UNIX (de exemplu Solaris, BSD, Linux) cu comenziile:

```
cc -o client client.c -lsocket -lnsl  
cc -o server sever.c -lsocket -lnsl
```

Serverul este pornit tastând doar

```
server
```

Clientul are nevoie de două argumente, așa cum s-a discutat mai sus. O versiune de Windows este de asemenea disponibilă pe situl Web.

Ca o observație, acest server nu este ultimul cuvânt în domeniul programelor server. Verificarea erorilor este ineficientă și raportarea erorilor este mediocru. În mod clar serverul nu a auzit niciodată de securitate, și folosirea doar a apelurilor de sistem UNIX nu este ultimul cuvânt în independență de platformă. De asemenea face unele presupuneri care sunt tehnice ilegale, cum ar fi presupunerea că numele fișierului începe în zona de memorie tampon și este transmis automat. Din moment ce tratează toate cererile strict secvențial (deoarece are doar un singur fir de execuție) performanța este slabă. În ciuda acestor neajunsuri, este un server de fișiere Internet complet și funcțional. În exercițiul, cititorul este invitat să le îmbunătățească. Pentru mai multe informații despre programare cu socluri, a se vedea (Stevens, 1997).

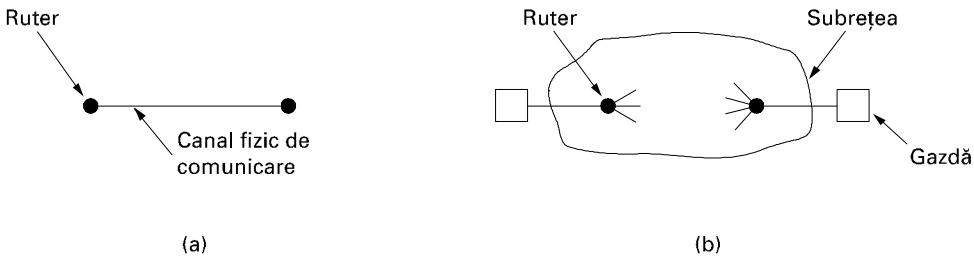
## 6.2 NOTIUNI DE BAZĂ DESPRE PROTOCOALELE DE TRANSPORT

Serviciul transport este implementat prin intermediul unui **protocol de transport** folosit de cele două entități de transport. Câteva caracteristici sunt asemănătoare pentru protocoalele de transport și pentru cele de legătură de date studiate în detaliu în cap. 3. Amândouă trebuie să se ocupe, printre altele, de controlul erorilor, de secvențiere și de controlul fluxului.

Totuși, există diferențe semnificative între cele două protocoale. Aceste diferențe sunt datorate deosebirilor majore dintre mediile în care operează protocoalele, așa cum rezultă din fig. 6-7. La nivelul legăturii de date, cele două rutere comunică direct printr-un canal fizic, în timp ce la nivelul transport acest canal fizic este înlocuit de întreaga subrețea. Această deosebire are mai multe implicații importante pentru protocoale, așa cum vom vedea în acest capitol.

În cazul legăturii de date, pentru un ruter nu trebuie specificat cu care alt ruter vrea să comunice, deoarece fiecare linie specifică în mod unic o destinație. În schimb, în cazul nivelului transport este necesară adresarea explicită.

În plus, procesul stabilirii unei conexiuni prin cablul din fig. 6-7(a) este simplu: celălalt capăt este întotdeauna acolo (în afară de cazul în care nu a ‘căzut’) și în nici unul din cazuri nu sunt prea multe de făcut. Pentru nivelul transport însă, stabilirea inițială a conexiunii este mult mai complicată, așa cum vom vedea.



**Fig. 6-7.** (a) Mediul pentru nivelul legătură de date. (b) Mediul pentru nivelul transport.

O altă diferență între nivelurile legătură de date și transport, care generează multe probleme, este existența potențială a unei capacitați de memorare a subrețelei. Atunci când un ruter trimit un cadru (nivel legătură de date), acesta poate să ajungă sau poate să se piardă, dar nu poate să se plimbe un timp ajungând până la capătul lumii și să apară 30 de secunde mai târziu, într-un moment nepotrivit. Dacă subrețeaua folosește datagrame și dirijare adaptivă, există o posibilitate - care nu poate fi neglijată - ca un pachet să fie păstrat pentru un număr oarecare de secunde și livrat mai târziu. Consecințele capacității de memorare a subrețelei pot fi uneori dezastruoase și necesită folosirea unor protocoale speciale.

O ultimă diferență între nivelurile legătură de date și transport este una de dimensionare și nu de proiectare. Folosirea tampoanelor și controlul fluxului sunt necesare la amândouă nivelurile, dar prezența unui număr mare de conexiuni în cazul nivelului transport necesită o abordare diferită de cea de la nivelul legătură de date. În cap. 3, unele protocoale alocau un număr fix de tampoane pentru fiecare linie, astfel încât atunci când sosea un cadru, exista întotdeauna un tampon disponibil. La nivel transport, numărul mare de conexiuni care trebuie să fie gestionate face ca ideea de a aloca tampoane dedicate să fie mai puțin atractivă. În următoarele secțiuni, vom examina atât aceste probleme importante cât și altele.

### 6.2.1 Adresarea

Atunci când un proces aplicație (de exemplu, un proces utilizator) dorește să stabilească o conexiune cu un proces aflat la distanță, el trebuie să specifică cu care proces dorește să se conecteze. (La protocoalele de transport neorientate pe conexiune apare aceeași problemă: cui trebuie trimis mesajul?). Metoda folosită în mod normal este de a defini adresa de transport la care procesele pot să aștepte cereri de conexiune. În Internet acestea se numesc porturi. La rețelele ATM perechile se numesc AAL - SAP-uri. În continuare vom folosi pentru acestea termenul generic **TSAP** (**T**ransport **S**ervice **A**ccess **P**oint - punct de acces la serviciul de transport). Punctele similare în cazul nivelului rețea (adică adresele la nivel rețea) sunt numite **NSAP** (**N**etwork **S**ervice **A**ccess **P**oint). Adresele IP sunt exemple de NSAP-uri.

Fig. 6-8 ilustrează relația între TSAP, NSAP și conexiunile transport. Procesele aplicație, atât clienții cât și serverele, se pot ataşa la TSAP pentru a stabili o conexiune la un TSAP la distanță. Aceste conexiuni rulează prin TSAP-uri pe fiecare gazdă așa cum se arată. Necesitatea de a avea mai multe TSAP-uri este dată de faptul că în unele rețele, fiecare calculator are un singur NSAP, deci cumva este nevoie să se distingă mai multe puncte de sfârșit de transport care partajează acel NSAP.

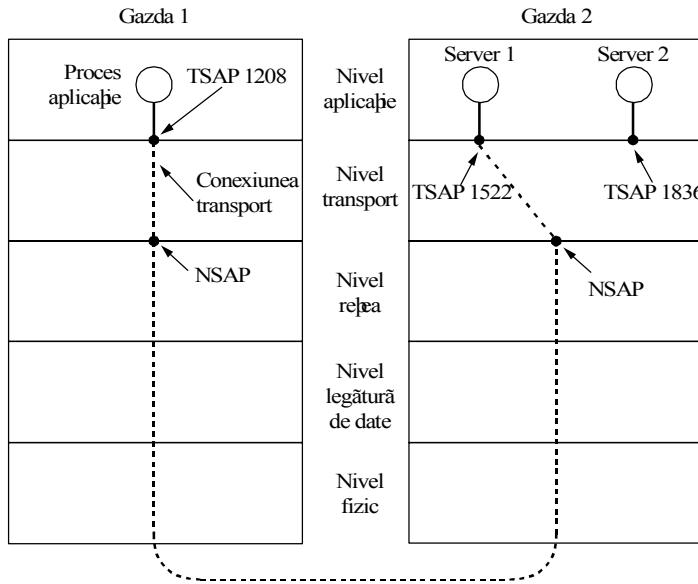


Fig. 6-8. TSAP, NSAP și conexiunile la nivel transport.

Un scenariu posibil pentru stabilirea unei conexiuni la nivel transport este următorul.

1. Un proces server care furnizează ora exactă și care rulează pe gazda 2 se atașează la TSAP 122 așteptând un apel. Felul în care un proces se atașează la un TSAP nu face parte din modelul de rețea și depinde numai de sistemul de operare local. Poate fi utilizat un apel de tip LISTEN din capitolul precedent.
2. Un proces aplicatie de pe gazda 1 dorește să afle ora exactă; atunci el generează un apel CONNECT specificând TSAP 1208 ca sursă și TSAP 1522 ca destinație. Această acțiune are ca rezultat în cele din urmă stabilirea unei conexiuni la nivel transport între procesele aplicatie de pe gazda 1 și serverul 1 de pe gazda 2.
3. Procesul aplicatie trimite o cerere o cerere pentru timp.
4. Procesul server de timp răspunde cu timpul curent.
5. Conexiunea transport este apoi eliberată.

Trebuie reținut că foarte bine pot exista alte servere pe gazda 2 care să fie atașate la alte TSAP-uri și care să aștepte conexiuni care ajung pe același NSAP.

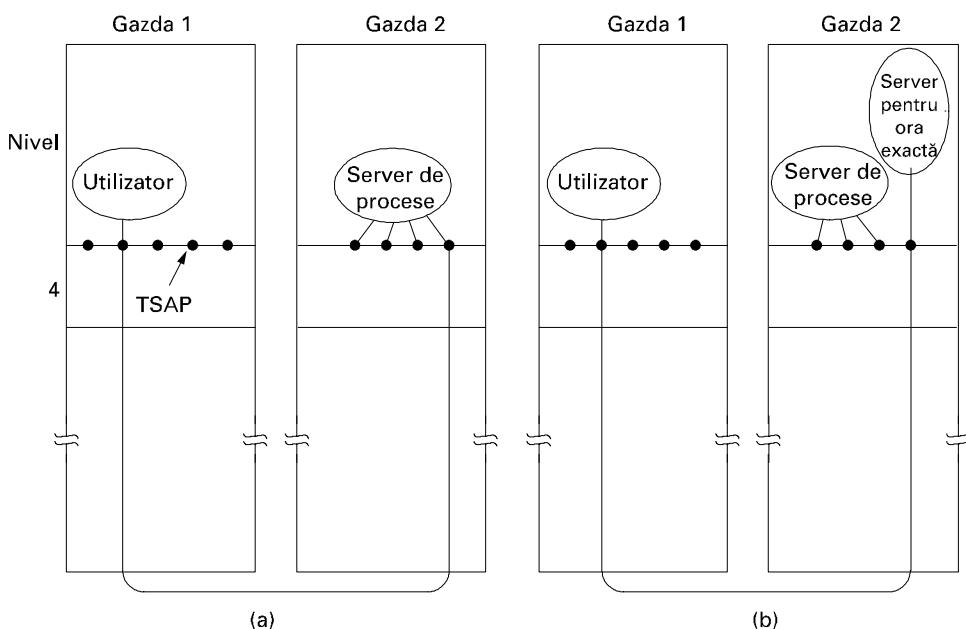
Fig. 6-8 explică aproape tot, cu excepția unei mici probleme: cum știe procesul utilizator de pe mașina 1 că serverul de oră exactă este atașat la TSAP 1522? O posibilitate este ca acest server de oră exactă să se atașeze la TSAP 1522 de ani de zile și, cu timpul, toți utilizatorii au aflat acest lucru. În acest model serviciile au adrese TSAP fixe, care pot fi afișate în fișiere în locuri bine cunoscute, cum este fișierul *etc/services* pe sistemele UNIX care afișează ce servere sunt atașate permanent și la ce porturi.

Dar schema cu adrese de servicii fixe funcționează doar pentru un număr mic de servicii cheie, a căror adresă nu se schimbă niciodată (de exemplu server de Web). Însă, în general, procesele utilizator vor să comunice cu alte procese care există numai pentru scurt timp și nu au o adresă TSAP dinainte cunoscută. Pe de altă parte, pot exista mai multe procese server, majoritatea utilizate foarte

rar, și ar fi neeconomic ca fiecare să fie activ și să asculte la o adresă TSAP fixă tot timpul. Pe scurt, este necesară o soluție mai bună.

O astfel de soluție este prezentată în fig. 6-9, într-o formă simplificată. Ea este cunoscută ca **protocolul de conectare inițială**. În loc ca orice server să asculte la un TSAP fixat, fiecare mașină care dorește să ofere servicii utilizatorilor aflați la distanță are un **server de procese (process server)** special care acționează ca un intermediar pentru toate serverele mai puțin utilizate. El ascultă în același timp la un număr de porturi, așteptând o cerere de conexiune. Utilizatorii potențiali ai serviciului încep prin a face o cerere de conexiune, specificând adresa TSAP a serviciului pe care îl doresc. Dacă nu există un server care să aștepte conexiuni la acel port, ele obțin o conexiune la serverul de procese, ca în fig. 6-9 (a).

După ce primește cererea, serverul de procese dă naștere serverului cerut, permittându-i să moștenească conexiunea cu procesul utilizator. Noul server execută prelucrarea cerută, în timp ce serverul de procese continuă să aștepte noi cereri, ca în fig. 6-9 (b).



**Fig. 6-9.** Stabilirea unei conexiuni între calculatorul gazdă 1 și serverul pentru ora exactă.

În timp ce acest protocol funcționează bine pentru serverele care pot fi create ori de câte ori este nevoie de ele, există mai multe situații în care serviciile există independent de serverul de procese. De exemplu, un server de fișiere va rula folosind un hardware specializat (o mașină cu disc) și nu poate fi creat din mers.

Pentru a trata această situație, este des utilizată o soluție alternativă. În acest model există un proces special numit **server de nume (name server sau, uneori, directory server)**. Pentru a găsi adresa TSAP corespunzătoare unui serviciu dat prin nume, așa cum este „ora exactă”, utilizatorul stabilește o conexiune cu serverul de nume (care așteaptă mesaje la un TSAP cunoscut). Apoi utilizatorul trimite un mesaj specificând numele serviciului, iar serverul de nume îi trimită înapoi adresa TSAP a

acestuia. După aceasta, utilizatorul eliberează conexiunea cu serverul de nume și stabilește o nouă conexiune cu serviciul dorit.

În acest model, atunci când este creat un nou serviciu, el trebuie să se înregistreze singur la serverul de nume, furnizând atât numele serviciului oferit (în general un șir ASCII) cât și adresa TSAP. Serverul de nume înregistreză această informație într-o bază de date internă, astfel încât el va ști răspunsul atunci când vor sosi noi cereri.

Funcționarea serverului de nume este asemănătoare cu serviciul de informații de la un sistem telefonic: este furnizată corespondența dintre nume și numere de telefon. Ca și în cazul telefoanelor, este esențial ca adresa bine cunoscută a serverului de nume (sau a serverului de procese, în protocolul de conectare inițială) să fie într-adevăr bine cunoscută. Dacă nu știi numărul de la informații, nu poți afla nici un alt număr de telefon. Dacă crezi că numărul de la informații este evident pentru toți, încearcă să-l folosești și în altă țară!

### 6.2.2 Stabilirea conexiunii

Stabilirea unei conexiuni poate să pară ușoară dar, în realitate, este surprinzător de complicată. La prima vedere, ar părea suficient ca o entitate de transport să trimită numai un TPDU CONNECTION REQUEST și să aștepte replica CONNECTION ACCEPTED. Problema apare deoarece rețeaua poate pierde, memoră sau duplica pachete. Acest comportament duce la complicații serioase.

Putem imagina o subrețea care este atât de congestionată încât confirmările ajung greu înapoi, și, din această cauză, fiecare pachet ajunge să fie retransmis de câteva ori. Putem presupune că subrețea folosește datagrame și fiecare pachet urmează un traseu diferit. Unele pachete pot să întâlnească o congestie locală de trafic și să întârzie foarte mult, ca și cum ar fi fost memorate de subrețea un timp și eliberate mai târziu.

Cel mai neplăcut scenariu ar fi: un utilizator stabileste o conexiune cu o bancă și trimite un mesaj cerând transferul unei sume de bani în contul unei alte persoane în care nu poate avea încredere în totalitate, și apoi eliberează conexiunea. Din nefericire, fiecare pachet din acest scenariu este duplicat și memorat în subrețea. După ce conexiunea a fost eliberată, pachetele memorate ies din subrețea și ajung la destinatar, cerând băncii să stabilizească o nouă conexiune, să facă transferul (încă o dată) și să elibereze conexiunea. Banca nu poate să știe că acestea sunt duplicate, ea trebuie să presupună că este o tranzacție independentă și va transfera banii încă o dată. În continuarea acestei secțiuni, vom studia problema dupliacelor întârziate, punând accentul în mod special pe algoritmii pentru stabilirea sigură a conexiunilor, astfel încât scenarii ca cel de mai sus să nu poată să apară.

După cum am mai spus, punctul crucial al problemei este existența dupliacelor întârziante. El poate fi tratat în mai multe feluri, dar nici unul nu este într-adevăr satisfăcător. O posibilitate este de a utiliza adrese de transport valabile doar pentru o singură utilizare. În această abordare, ori de câte ori este necesară o adresă la nivel transport, va fi generată una nouă. După ce conexiunea este eliberată, adresa nu mai este folosită. Acest mecanism face însă imposibil modelul cu server de procese din fig. 6-9.

O altă posibilitate este de a atribui fiecărei conexiuni un identificator (adică, un număr de secvență incrementat pentru fiecare conexiune stabilită), ales de cel care inițiază conexiunea, și pus în fiecare TPDU, inclusiv în cel care inițiază conexiunea. După ce o conexiune este eliberată, fiecare entitate de transport va completa o tabelă cu conexiunile care nu mai sunt valide, reprezentate ca perechi (entitate de transport, identificator conexiune). Ori de câte ori apare o cerere de conexiune se va verifica în tabelă că ea nu aparține unei conexiuni care a fost eliberată anterior.

Din nefericire, această schemă are un defect important: ea necesită ca fiecare entitate de transport să mențină informația despre conexiunile precedente un timp nedefinit. Dacă o mașină cade și își pierde datele din memorie, ea nu va mai ști care identificatori de conexiune au fost deja utilizati.

Putem încerca și o altă soluție. În loc să permitem pachetelor să trăiască la nesfârșit în subrețea, putem inventa un mecanism care să eliminate pachetele îmbătrânite. Dacă suntem siguri că nici un pachet nu poate să supraviețuiască mai mult de un anume interval de timp cunoscut, problema devine ceva mai ușor de rezolvat.

Durata de viață a pachetelor poate fi limitată la un maxim cunoscut, folosind una (sau mai multe) din următoarele tehnici:

1. Restricții în proiectarea subrețelei
2. Adăugarea unui contor al nodurilor parcuse în fiecare pachet
3. Adăugarea unei amprente de timp la fiecare pachet

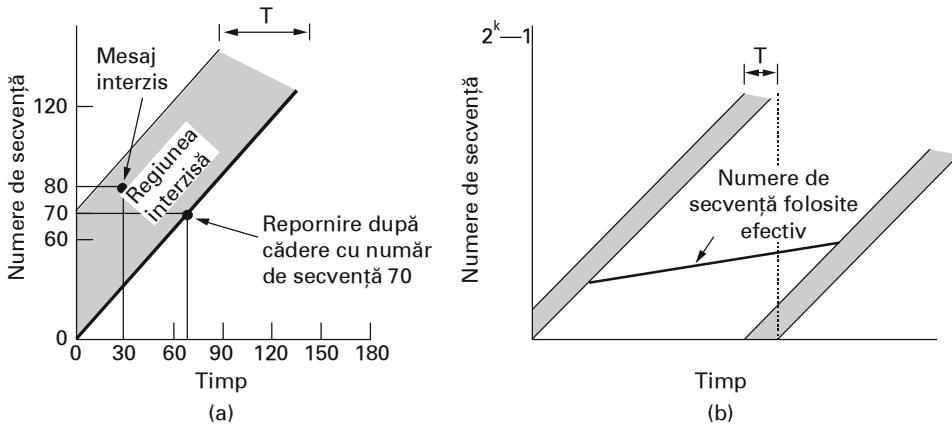
Prima metodă include soluțiile care împiedică pachetele să stea în buclă, combinate cu modalități de a limita întârzierile datorate congestiilor, pe orice cale din rețea (indiferent de lungime). A doua metodă constă în a inițializa contorul cu o valoare adecvată și în a-l decrementa la trecerea prin orice nod. Protocolul de nivel rețea pur și simplu elimină pachetele al căror contor a devenit zero. A treia metodă presupune ca fiecare pachet să conțină timpul creării sale, ruterele acceptând să eliminate pachetele mai vechi de un anumit moment de timp, asupra căruia au căzut de acord. Această metodă necesită ca ceasurile de la fiecare ruter să fie sincronizate, și această cerință în sine este destul de greu de îndeplinit (mai ușor este dacă sincronizarea ceasurilor se obține din exteriorul rețelei, de exemplu folosind GPS sau stații radio care transmit periodic ora exactă).

În practică, nu este suficient doar să garantăm că pachetul este eliminat, ci trebuie garantat și că toate confirmările sale au fost eliminate, astfel încât vom introduce  $T$ , care va fi un multiplu (mic) al duratei maxime de viață a unui pachet. Depinde de protocol de câte ori  $T$  este mai mare decât durata de viață a unui pachet. Dacă așteptăm un timp  $T$  după trimiterea unui pachet putem fi siguri că toate urmele sale au dispărut și nici el, nici vreo confirmare de-a sa nu vor apărea din senin, doar ca să complice lucrurile.

Folosind durata de viață limitată a pachetelor, există metode de a obține conexiuni sigure a căror corectitudine a fost demonstrată. Metoda descrisă în cele ce urmează este datorată lui Tomlinson (1975). Ea rezolvă problema, dar introduce câteva particularități proprii. Metoda a fost îmbunătățită de Sunshine și Dalal (1978). Variante ale sale sunt larg folosite în practică, inclusiv în TCP.

Pentru a ocoli problemele generate de pierderea tuturor datelor din memoria unei mașini după o cădere, Tomlinson propune echiparea fiecărei mașini cu un ceas. Nu este nevoie ca ceasurile de pe mașini diferite să fie sincronizate. Fiecare ceas va fi de fapt un contor binar care se autoincrementează după un anumit interval de timp. În plus, numărul de biți ai contorului trebuie să fie cel puțin egal cu numărul de biți al numerelor de secvență. În cele din urmă, și cel mai important, ceasul trebuie să continue să funcționeze chiar în cazul în care calculatorul gazdă cade.

Ideeoa de bază este de a fi siguri că două TPDU numerotate identic nu pot fi generate în același timp. Atunci când conexiunea este inițiată, k biți mai puțin semnificativi ai ceasului sunt folosiți ca număr inițial de secvență (tot k biți). Astfel, fiecare conexiune începe să-și numeroteze TPDU-urile sale cu un număr de secvență diferit. Spațiul numerelor de secvență ar trebui să fie suficient de mare pentru ca, în timpul scurs până când contorul ajunge din nou la același număr, toate TPDU-urile vechi cu acel număr să fi dispărut deja. Această relație liniară între timp și numărul de secvență inițial este prezentată în fig. 6-10.



**Fig. 6-10.** (a) TPDU-urile nu pot să intre în regiunea interzisă.  
(b) Problema resincronizării.

Odată ce ambele entități de transport au căzut de acord asupra numărului de secvență inițial, pentru controlul fluxului poate fi folosit orice protocol cu fereastră glisantă. În realitate curba ce reprezintă numărul inițial de secvență (desenată cu linie îngroșată) nu este chiar liniară, ci în trepte, căci ceasul avansează în trepte. Pentru simplitate, vom ignora acest detaliu.

O problemă apare atunci când cade un calculator gazdă. Când el își revine, entitatea sa de transport nu știe unde a rămas în spațiul numerelor de secvență. O soluție este de a cere entității de transport să stea neocupată  $T$  secunde după revenire pentru ca în acest timp toate vechile TPDU să dispară. Totuși, într-o rețea complexă  $T$  poate fi destul de mare, astfel că această strategie nu este prea atrăgătoare.

Pentru a evita cele  $T$  secunde de timp nefolosit după o cădere, este necesar să introducem o nouă restricție în utilizarea numerelor de secvență. Necesitatea introducerii acestei restricții este evidentă în următorul exemplu. Fie  $T$ , timpul maxim de viață al unui pachet, egal cu 60 de secunde și să presupunem că ceasul este incrementat la fiecare secundă. După cum arată linia îngroșată din fig. 6-10(a), numărul inițial de secvență pentru o conexiune inițiată la momentul  $x$  este  $x$ . Să ne imaginăm că la  $t=30$  sec, unui TPDU trimis pe conexiunea cu numărul 5 (deschisă anterior) i se dă numărul de secvență 80. Să numim acest TPDU  $X$ . Imediat după ce  $X$  este trimis, calculatorul gazdă cade și revine imediat. La  $t=60$  el redeschide conexiunile de la 0 la 4. La  $t=70$ , el deschide conexiunea 5, folosind un număr de secvență inițial 70, așa cum am stabilit. În următoarele 15 secunde el va transmite TPDU-uri cu date numerotate de la 70 la 80. Astfel că la  $t=85$ , în subrețea este generat un nou TPDU cu numărul de secvență 80 și conexiunea 5. Din nefericire, TPDU  $X$  încă mai există. Dacă el ajunge înaintea noului TPDU 80, atunci TPDU  $X$  va fi acceptat și TPDU-ul corect va fi respins ca fiind duplicat.

Pentru a preveni o astfel de problemă trebuie să luăm măsuri ca numerele de secvență să nu fie utilizate (adică atribuite unor noi TPDU-uri) un timp  $T$  înaintea utilizării lor ca noi numere de secvență. Combinățiile imposibile - timp, număr de secvență - sunt prezentate în fig. 6-10(a) ca **regiunea interzisă**. Înainte de trimitera oricărui TPDU pe orice conexiune, entitatea de transport trebuie să citească ceasul și să verifice dacă nu cumva se află în regiunea interzisă.

Pot să apară probleme în două cazuri: dacă un calculator gazdă trimite prea multe date și prea repede pe o conexiune nou deschisă, curba numărului de secvență în funcție de timp poate să fie mult

mai abruptă decât cea inițială. Aceasta înseamnă că rata de transmisie pentru orice conexiune este de cel mult un TPDU pe unitatea de timp a ceasului. De asemenea, este necesar ca entitatea de transport să aștepte până când ceasul avansează o dată, înainte să deschidă o nouă conexiune pentru ca, la revenirea după o cădere, același număr de secvență să nu fie utilizat de două ori. Cele două observații de mai sus sunt argumente pentru ca perioada ceasului să fie cât mai mică (câteva μs sau mai mică).

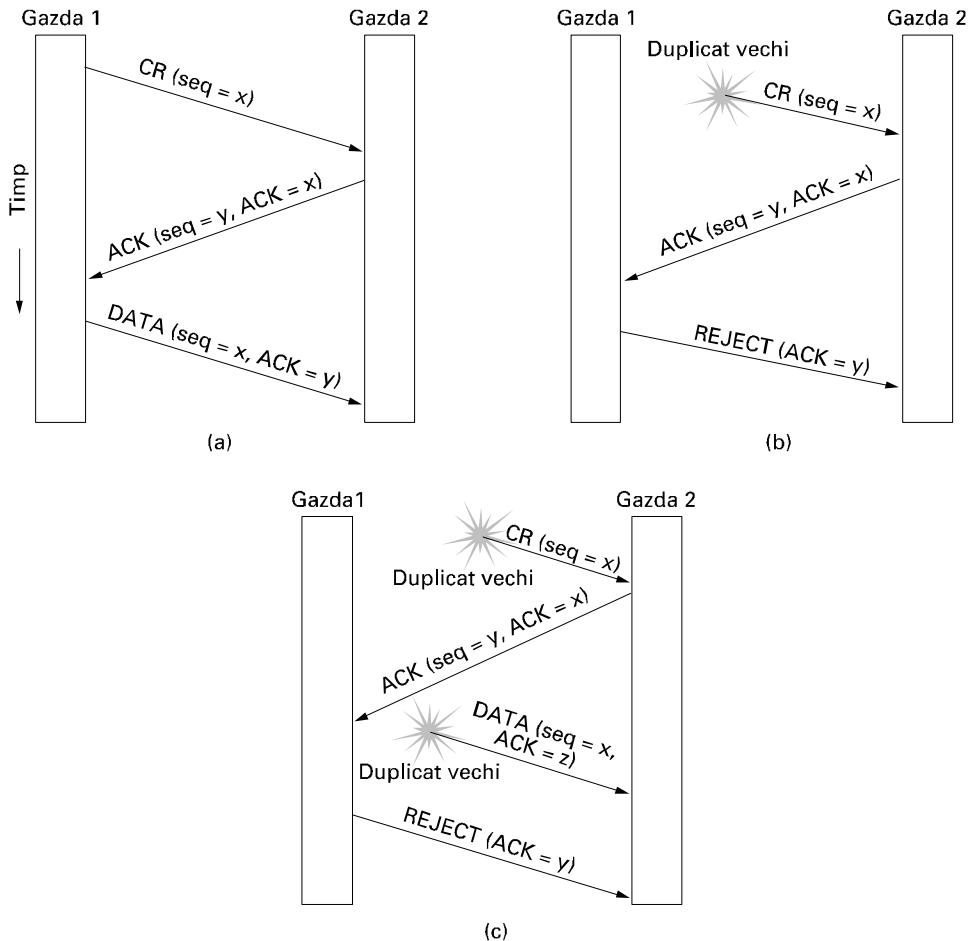
Din nefericire, intrarea în regiunea interzisă prin trimitere prea rapidă nu este singura situație care creează probleme. Fig. 6-10(b) arată că la orice rată de transfer mai mică decât frecvența ceasului curba numerelor de secvență utilizate raportată la timp va ajunge până la urmă în regiunea interzisă din stânga. Cu cât curba numerelor de secvență utilizate va fi mai înclinată, cu atât mai târziu se ajunge în regiunea interzisă. Așa cum am afirmat anterior, imediat înaintea trimiterii unui TPDU, entitatea de transport trebuie să verifice dacă nu se află cumva în regiunea interzisă, și, dacă se află, să întârzie transmisia cu T secunde sau să resincronizeze numerele de secvență.

Metoda bazată pe ceasuri rezolvă problema duplicatelor întârziate pentru TPDU-urile de date, dar pentru ca această metodă să poată fi folosită, trebuie mai întâi să stabilim conexiunea. Deoarece TPDU-urile de control pot și ele să fie întârziate, pot apărea probleme atunci când entitățile de transport încercă să cadă de acord asupra numărului inițial de secvență. Să presupunem, de exemplu, că, pentru a stabili o conexiune, gazda 1 trimit un mesaj CONNECTION REQUEST conținând numărul de secvență inițial propus și portul destinație gazdei 2. Acesta va confirma mesajul trimițând înapoi un TPDU CONNECTION ACCEPTED. Dacă TPDU-ul CONNECTION REQUEST este pierdut, dar un duplicat întârziat al unui alt CONNECTION REQUEST va ajunge la gazda 2, atunci conexiunea nu va fi stabilită corect.

Pentru a rezolva aceasta problemă, Tomlinson (1975) a introdus stabilirea conexiunii cu înțelegere în trei pași (three-way handshake). Acest protocol nu necesită ca ambele părți să înceapă să trimită același număr de secvență, deci poate fi utilizat și împreună cu alte metode de sincronizare decât ceasul global. Procedura normală de inițiere a conexiunii este exemplificată în fig. 6-11(a). Gazda 1 alege un număr de secvență  $x$  și trimit un TPDU CONNECTION REQUEST care conține  $x$  gazdei 2. Gazda 2 răspunde cu CONNECTION ACK, confirmând  $x$  și anunțând numărul său inițial de secvență,  $y$ . În cele din urmă gazda 1 confirmă alegerea lui  $y$  gazdei 2 în primul mesaj de date pe care îl trimit.

Vom arunca acum o privire asupra stabilirii conexiunii cu înțelegere în trei pași în prezența TPDU-urilor de control duplicate întârziate. În fig. 6-11(b) primul TPDU sosit este o copie întârziată a unui CONNECTION REQUEST de la o conexiune mai veche. Acest TDU ajunge la gazda 2 fără ca gazda 1 să știe. Gazda 2 răspunde acestui TPDU trimițând gazdei 1 un TPDU ACK, verificând de fapt că gazda 1 a încercat într-adevăr să stabilească o conexiune. Atunci când gazda 1 refuză cererea gazdei 2 de a stabili conexiunea, gazda 2 își dă seama că a fost păcălită de o copie întârziată și abandonează conexiunea. În acest fel o copie întârziată nu poate să strice nimic.

În cel mai rău caz, atât CONNECTION REQUEST cât și ACK sunt copii întârziate în subrețea. Acest caz este prezentat în 6-11(c). Ca și în exemplul precedent, gazda 2 primește o comandă CONNECTION REQUEST întârziată și răspunde la ea. În acest moment este extrem de important să ne aducem aminte că gazda 2 a propus  $y$  ca număr inițial de secvență pentru traficul de la 2 la 1, fiind sigur că nu mai există în rețea nici un TPDU (sau confirmare) cu același număr de secvență. Atunci când al doilea TPDU întârziat ajunge la gazda 2, aceasta deduce, din faptul că a fost confirmat  $z$  și nu  $y$ , că are de-a face cu o copie mai veche. Important este că nu există nici o combinație posibilă ale unor copii vechi ale TPDU-urilor întârziate care să reușească să inițieze o conexiune atunci când nimeni nu a cerut asta.



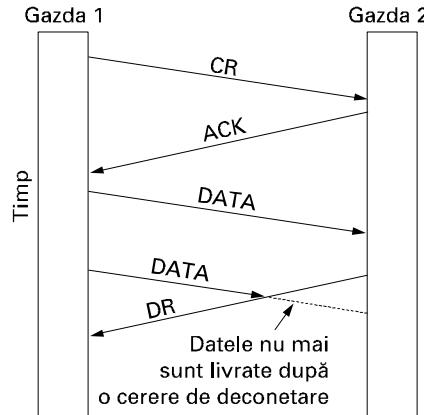
**Fig. 6-11.** Trei scenarii posibile de stabilire a conexiunii pentru un protocol cu înțelegere în trei pași. CR reprezintă CONNECTION REQUEST. (a) Cazul normal, (b) Un duplicat vechi al unui mesaj CONNECTION REQUEST apare când nu trebuie, (c) Sunt duplicate atât CONNECTION REQUEST cât și CONNECTION ACCEPTED.

### 6.2.3 Eliberarea conexiunii

Eliberarea unei conexiuni este mai ușoară decât stabilirea ei. Totuși, există mai multe dificultăți decât ne-am așteptă. Așa cum am mai amintit, există două moduri de a termina o conexiune: eliberare simetrică și eliberare asimetrică. Sistemul telefonic folosește eliberarea asimetrică: atunci când unul din interlocutori închide, conexiunea este întreruptă. Eliberarea simetrică privește conexiunea ca pe două conexiuni separate unidirectionale și cere ca fiecare să fie eliberată separat.

Eliberarea asimetrică este bruscă și poate genera pierderi de date. Să considerăm scenariul din fig. 6-12. După stabilirea conexiunii, gazda 1 trimite un TPDU care ajunge corect la gazda 2. Gazda

1 mai trimite un TPDU dar, înainte ca acesta să ajungă la destinație, gazda 2 trimite DISCONNECT REQUEST. În acest caz, conexiunea va fi eliberată și vor fi pierdute date.



**Fig. 6-12.** Deconectare bruscă cu pierdere de date. CR= CONNECTION REQUEST, ACK=CONNECTION ACCEPTED , DR=DISCONNECT REQUEST.

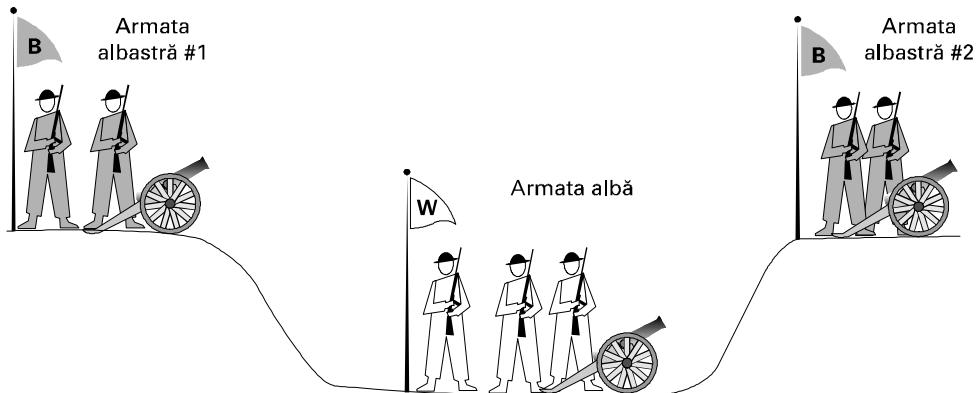
Evident, pentru a evita pierderea de date, este necesar un protocol de eliberare a conexiunii mai sofisticat. O posibilitate este utilizarea eliberării simetrice: fiecare direcție este eliberată independent de celalătă; un calculator gazdă poate să continue să primească date chiar și după ce a trimis un TPDU de eliberare a conexiunii.

Eliberarea simetrică este utilă atunci când fiecare proces are o cantitate fixă de date de trimis și știe bine când trebuie să transmită și când a terminat. În alte situații însă, nu este deloc ușor de determinat când trebuie eliberată conexiunea și când a fost trimis tot ce era de transmis. S-ar putea avea în vedere un protocol de tipul următor: atunci când 1 termină, trimit ceva de tipul: Am terminat. Ai terminat și tu? Dacă gazda 2 răspunde: Da, am terminat. Închidem! conexiunea poate fi eliberată în condiții bune.

Din nefericire, acest protocol nu merge întotdeauna. Binecunoscuta **problemă a celor două armate** este similară acestei situații: să ne imaginăm că armată albă și-a pus tabăra într-o vale (ca în fig. 6-13) Pe amândouă cretele care mărginesc valea sunt armatele albastre. Armata albă este mai mare decât fiecare din cele două armate albastre, dar împreună armatele albastre sunt mai puternice. Dacă oricare din armatele albastre atacă singură, ea va fi înfrântă, dar dacă ele atacă simultan, atunci vor fi victorioase.

Armatele albastre vor să-și sincronizeze atacul. Totuși singura lor posibilitate de comunicație este să trimită un mesaj care să străbată valea. Mesajul poate fi capturat de armata albă și mesajul poate fi pierdut (adică vor trebui să utilizeze un canal de comunicație nesigur). Problema este următoarea: există vreun protocol care să permită armatelor albastre să învingă?

Să presupunem că comandantul primei armate albastre trimite un mesaj: „Propun să atacăm pe 29 martie”, mesajul ajunge la armata 2 al cărei comandant răspunde: „De acord” iar răspunsul ajunge înapoi la armata 1. Va avea loc atacul în acest caz? Probabil că nu, deoarece comandantul armatei 2 nu știe dacă răspunsul său a ajuns sau nu la destinație. Dacă nu a ajuns, armata 1 nu va ataca, deci ar fi o prostie din partea lui să intre în luptă.



**Fig. 6-13.** Problema celor două armate.

Să încercăm să îmbunătățim protocolul, transformându-l într-unul cu înțelegere în trei pași. Inițiatorul propunerii de atac trebuie să confirme răspunsul. Presupunând că nici un mesaj nu este pierdut, armata 2 va avea confirmarea, dar comandantul armatei 1 va ezita acum. Până la urmă, el nu știe dacă confirmarea sa a ajuns la destinație și este sigur că dacă aceasta nu a ajuns, armata 2 nu va ataca. Am putea să încercăm un protocol cu confirmare în patru timpi, dar ne-am lovit de aceleasi probleme.

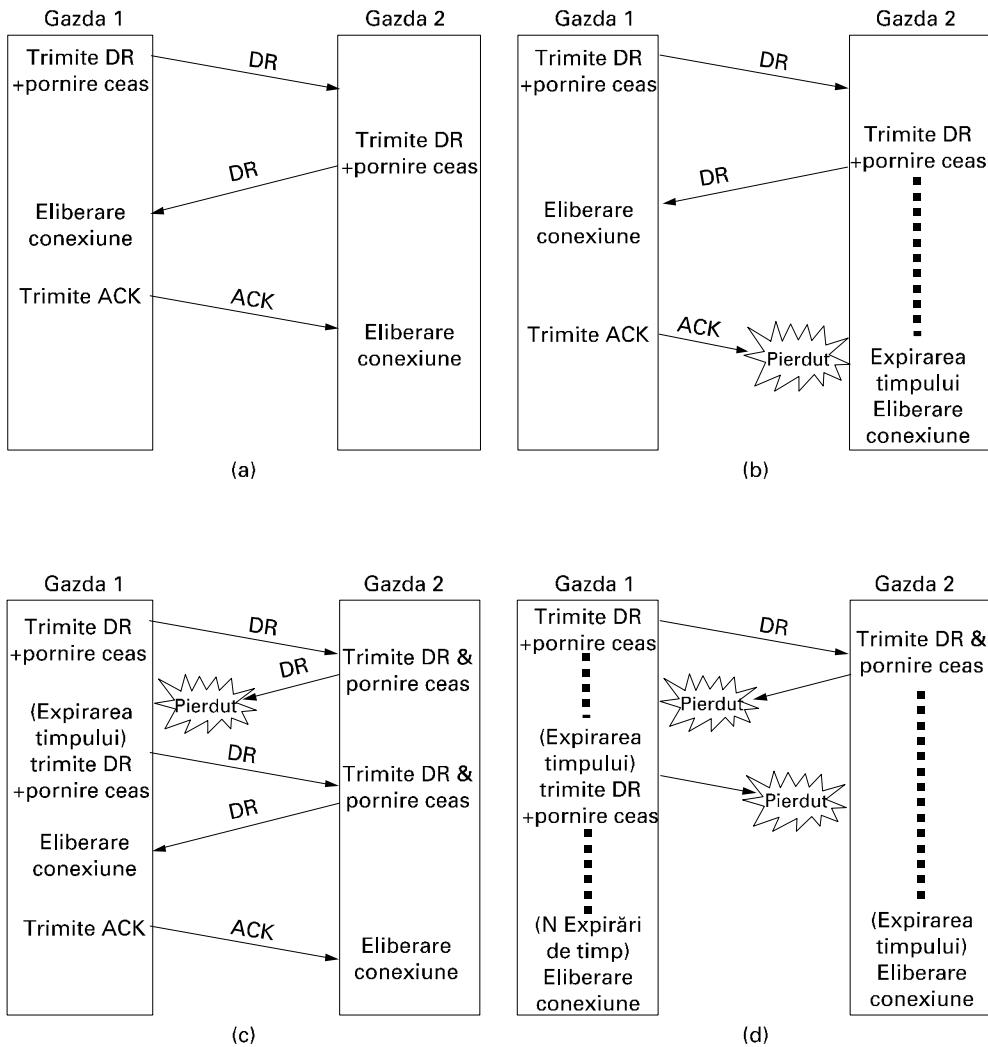
De fapt, poate fi demonstrat că nu există un protocol care să funcționeze. Să presupunem că ar exista un asemenea protocol: decizia finală poate să depindă sau nu de ultimul mesaj al unui asemenea protocol. Dacă nu depinde, putem elimina acest mesaj (și oricare altul la fel) până ajungem la un protocol în care orice mesaj este vital. Ce se va întâmpla dacă ultimul mesaj este interceptat? Tocmai am hotărât că acest mesaj era unul vital, deci dacă este pierdut, atacul nu va avea loc. Deoarece cel care trimite ultimul mesaj nu poate fi niciodată sigur că mesajul a ajuns, el nu va risca atât. Mai rău chiar, cealaltă armată albastră știe și ea acest lucru, deci nu va ataca nici ea.

Pentru a vedea legătura problemei celor două armate cu problema eliberării conexiunii este suficient să înlocuim ‘atac’ cu ‘deconectare’. Dacă niciuna din părți nu se deconectează până nu este sigură că cealaltă parte este gata să se deconecteze la rândul ei, atunci deconectarea nu va mai avea loc niciodată.

În practică suntem dispuși să ne asumăm mai multe riscuri atunci când este vorba de eliberarea conexiunii decât atunci când este vorba de atacarea armatei albe, aşa încât situația nu este întru totul fără speranță. Fig. 6-14 prezintă patru scenarii de eliberare a conexiunii folosind un protocol cu confirmare în trei timpi. Deși acest protocol nu este infailibil, el este în general adecvat.

În fig. 6-14(a) apare cazul normal în care unul dintre utilizatori trimite un TPDU de tip DR (DISCONNECT REQUEST) pentru a iniția eliberarea conexiunii. Atunci când acesta sosesc, receptorul trimite înapoi tot un TPDU DR și pornește un ceas pentru a trata cazul în care mesajul său este pierdut. Când primește mesajul înapoi, inițiatorul trimite o confirmare și eliberează conexiunea. În sfârșit, la primirea confirmării, receptorul eliberează și el conexiunea. Eliberarea conexiunii înseamnă de fapt că entitatea de transport șterge din tabelele sale informația despre conexiunea respectivă din tabela de conexiuni deschise în momentul curent și semnalează acest lucru utilizatorului nivelului transport. Această acțiune nu este același lucru cu apelul unei primitive DISCONNECT de către un utilizator al nivelului transport.

Dacă ultima confirmare este pierdută, ca în fig. 6-14(b), putem salva situația cu ajutorul ceasului: după scurgerea unui anumit interval de timp conexiunea este eliberată oricum.



**Fig. 6-14.** Patru cazuri posibile la eliberarea conexiunii: (a) Cazul normal cu confirmare în trei timpi.  
 (b) Ultima confirmare este pierdută. (c) Răspunsul este pierdut.  
 (d) Răspunsul și următoarele cereri de deconectare sunt pierdute.  
 (DR=DISCONNECT REQUEST).

Să considerăm acum cazul în care cel de-al doilea DR este pierdut: utilizatorul care a inițiat deconectarea nu va primi răspunsul așteptat, va aștepta un anumit timp și va trimite din nou un DR. În fig. 6-14(c), putem vedea cum se petrec lucrurile în acest caz, presupunând că la a doua încercare toate TPDU-urile ajung corect și la timp.

Ultima posibilitate pe care o studiem, prezentată în fig. 6-14(d), este similară cu cea din 6-14(c), cu următoarea diferență: de aceasta dată niciuna din încercările următoare de a retransmite DR nu reușește. După  $N$  încercări, emițătorul va elibera pur și simplu conexiunea. În același timp, și receptorul va elibera conexiunea după expirarea timpului.

Deși acest protocol este, în general, destul de bun, în teorie el poate să dea greș dacă atât mesajul DR inițial cât și  $N$  retransmisii ale sale se pierd. Emitterul va renunța și va elibera conexiunea, în timp ce la celălalt capăt nu se știe nimic despre încercările de deconectare și aceasta va rămâne în continuare activă. În această situație rezultă o conexiune deschisă pe jumătate.

Am putea evita această problemă nepermittând emittorului să cedeze după  $N$  reîncercări nereușite, ci cerându-i să continue până primește un răspuns. Totuși, dacă celelalte părți i se permite să elibereze conexiunea după un interval de timp, este posibil ca inițiatorul să ajungă să aștepte la infinit. Dacă însă nu s-ar permite eliberarea conexiunii după expirarea unui interval de timp, atunci în cazul din fig. 6-14 (b) protocolul s-ar bloca.

O altă posibilitate de a scăpa de conexiunile pe jumătate deschise este de a aplica o regulă de tipul: dacă nici un TPDU nu sosesc într-un anumit interval de timp, atunci conexiunea este eliberată automat. În acest fel, dacă una din părți se deconectează, cealaltă parte va detecta lipsa de activitate și se va deconecta și ea. Desigur, pentru a implementa aceasta regulă este nevoie ca fiecare entitate de transport să aibă un ceas care va fi repornit la trimiterea oricărui TPDU. La expirarea timpului, se transmite un TPDU vid, doar pentru a menține conexiunea deschisă. Pe de altă parte, dacă este aleasă această soluție, și câteva TPDU-uri vide sunt pierdute la rând pe o conexiune altfel liberă, este posibil ca, mai întâi una din părți, apoi cealaltă să se deconecteze automat.

Nu vom mai continua să detaliem acest subiect, dar probabil că acum este clar că eliberarea unei conexiuni fără pierderi de date nu este atât de simplă cum parea la început.

#### 6.2.4 Controlul fluxului și memorarea temporară (buffering)

După ce am studiat în detaliu stabilirea și eliberarea conexiunii, vom arunca o privire asupra modului în care sunt tratate conexiunile cât timp sunt utilizate. Una din problemele cheie a apărut și până acum: controlul fluxului. La nivel transport există asemănări cu problema controlului fluxului la nivel legătură de date, dar există și deosebiri. Principala asemănare: la ambele niveluri este necesar un mecanism (fereastră glisantă sau altceva) pentru a împiedica un emittor prea rapid să depășească capacitatea de recepție a unui receptor prea lent. Principala deosebire: un ruter are în general puține linii, dar poate să aibă numeroase conexiuni. Această diferență face nepractică implementarea la nivel transport a strategiei de memorare temporară a mesajelor folosită la nivel legătură de date.

În protocolele pentru legătura de date prezentate în cap. 3, cadrele sunt memorate temporar atât de ruterul care emite cât și de cel care receptionează. În protocolul 6, de exemplu, atât emittorul cât și receptorul au alocate un număr de  $MAXSEQ+1$  tampoane pentru fiecare linie, jumătate pentru intrări și jumătate pentru ieșiri. Pentru un calculator gazdă cu, să spunem, 64 de conexiuni și numere de secvență de 4 biți, acest protocol ar necesita 1024 tampoane.

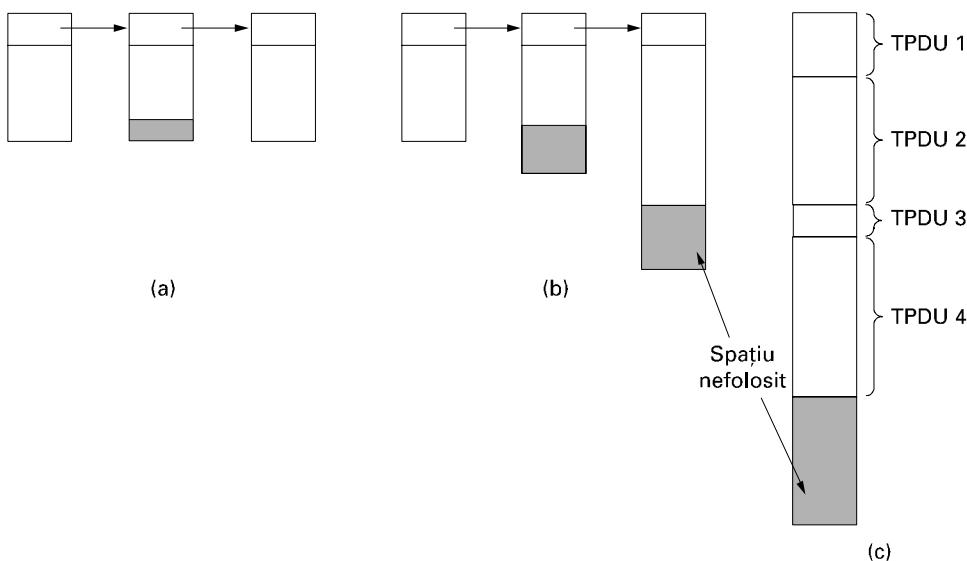
La nivel legătură de date, emittorul trebuie să memoreze cadrele transmise, pentru că poate fi necesară retransmiterea acestora. Dacă subrețea oferă un serviciu datagramă, atunci entitatea de transport emittatoare va trebui să memoreze pachetele trimise din aceeași motive. Dacă receptorul știe că emittorul stochează toate TPDU-urile până când acestea sunt confirmate, el poate să aloce sau nu tampoane specifice fiecărei conexiuni, după cum i se pare mai bine.

Receptorul poate, de exemplu, să rezerve un singur grup de tampoane pentru toate conexiunile. La sosirea unui TPDU se face o încercare de a obține dinamic un nou tampon. Dacă un tampon este liber, atunci TPDU-ul este acceptat, altfel, este refuzat. Cum emittorul este gata să retransmitem TPDU-urile pierdute de subrețea, faptul că unele TPDU-uri sunt refuzate nu produce nici o daună, deși în acest fel sunt risipite resurse. Emittorul va retransmite până când va primi confirmarea.

Pe scurt, dacă serviciul rețea nu este sigur, emițătorul va trebui să memoreze toate TPDU-urile trimise, la fel ca la nivel legătură de date. Totuși, folosind un serviciu la nivel rețea sigur sunt posibile unele compromisuri. În particular, dacă emițătorul știe că receptorul are întotdeauna tampoane disponibile, atunci nu trebuie să păstreze copiile TPDU-urilor trimise. Totuși, dacă receptorul nu poate garanta că orice TPDU primit va fi acceptat, emițătorul va trebui să păstreze copii. În ultimul caz, emițătorul nu poate avea încredere în confirmarea primită la nivel rețea, deoarece aceasta confirmă sosirea TPDU-ului la destinație, dar nu și acceptarea lui. Vom reveni asupra acestui punct important mai târziu.

Chiar dacă receptorul va realiza memorarea temporară a mesajelor primite, mai rămâne problema dimensiunii tamponului. Dacă cea mai mare parte a TPDU-urilor au aceeași dimensiune, este naturală organizarea tampoanelor într-o resursă comună care conține tampoane de aceeași dimensiune, cu un TPDU per tampon, ca în fig. 6-15(a). Dacă însă dimensiunea TPDU-urilor variază de la câteva caractere tipărite la un terminal, la mii de caractere pentru un transfer de fișiere, organizarea ca o resursă comună cu tampoane de aceeași dimensiune va pune probleme. Dacă dimensiunea tampoanelor ar fi constantă, egală cu cel mai mare TPDU posibil, atunci va apărea o risipă de spațiu ori de câte ori este primit un TPDU mai scurt. Dacă dimensiunea tampoanelor este aleasă mai mică decât cel mai mare TPDU posibil, atunci pentru memorarea unui TPDU mai lung vor fi necesare mai multe tampoane, iar complexitatea operației va crește.

O altă soluție este utilizarea unor tampoane de dimensiune variabilă, ca în fig. 6-15(b). Avantajul este o mai bună utilizare a memoriei, cu prețul unei gestiuni a tampoanelor mai complicată. O a treia posibilitate este alocarea unui singur tampon circular pentru fiecare conexiune, ca în fig. 6-15(c). Această soluție are de asemenea avantajul unei utilizări eficiente a memoriei, dar numai în situația în care conexiunile sunt relativ încărcate.



**Fig. 6-15.** (a) Tampoane de dimensiune fixă înlănțuite. (b) Tampoane de dimensiune variabilă înlănțuite. (c) Un tampon circular pentru fiecare conexiune.

Compromisul optim între memorarea temporară la sursă sau la destinație depinde de tipul traficului prin conexiune. Pentru un trafic în rafală cu o bandă de transfer îngustă, ca traficul produs de

un terminal interactiv, este mai bine ca tampoanele să nu fie prealocate, ci mai curând, alocate dinamic. Întrucât emițătorul nu poate să fie sigur că receptorul va reuși să aloce un tampon la sosirea unui pachet, emițătorul va fi nevoie să rețină copia unui TPDU transmis până când acesta va fi confirmat. Pe de altă parte, pentru un transfer de fișiere sau pentru orice alt trafic care necesită o bandă de transfer largă este mai bine dacă receptorul alocă un set întreg de tampoane, pentru a permite un flux de date la viteza maximă. Cu alte cuvinte, pentru un trafic în rafală cu o bandă de transfer îngustă este mai bine să fie folosite tampoane la emițător, în timp ce pentru un trafic continuu cu o bandă de transfer largă, este mai eficientă folosirea tampoanelor la receptor.

Pe măsură ce conexiunile sunt create și eliberate de trafic, iar şablonul se schimbă, emițătorul și receptorul trebuie să își ajusteze dinamic politica de alocare a tampoanelor. În consecință, protocolul de transport trebuie să permită emițătorului să ceară spațiu pentru tampoane la capătul celălalt al conexiunii. Tampoanele pot fi alocate pentru o anumită conexiune sau pot fi comune pentru toate conexiunile între două calculatoare gazdă. O alternativă este ca receptorul, cunoscând situația tampoanelor sale (dar necunoscând şablonul traficului) să poată spune emițătorului: „Am rezervat  $X$  tampoane pentru tine”. Dacă numărul conexiunilor deschise trebuie să crească, poate fi necesar ca spațiul alocat unei singure conexiuni să scadă, deci protocolul trebuie să furnizeze și această facilitate.

O modalitate înțeleaptă de a trata alocarea dinamică a tampoanelor este separarea stocării în tampoane de confirmarea mesajelor, spre deosebire de protocolul cu fereastră glisantă din cap. 3. Alocarea dinamică a tampoanelor înseamnă, de fapt, o fereastră cu dimensiune variabilă. La început, emițătorul trimite cereri pentru un anumit număr de tampoane bazându-se pe o estimare a necesităților. Receptorul îi alocă atâtea tampoane cât își poate permite. De fiecare dată când emițătorul trimite un TPDU, el decrementează numărul de tampoane pe care le are alocate la receptor, oprindu-se când acest număr devine zero. Receptorul trimite înapoi confirmări și situația tampoanelor alocate, împreună cu traficul în sens invers.

Fig. 6-16 este un exemplu pentru modul în care administrarea dinamică a ferestrelor poate fi folosită într-o rețea cu datagrame, cu numere de secvență pe 4 biți. Să presupunem că informația despre alocarea tampoanelor este împachetată în TPDU-uri distințe și că este separată de traficul în sens invers. La început A dorește opt tampoane, dar nu i se acordă decât patru. După aceea trimite trei TPDU-uri, iar al treilea este pierdut. TPDU-ul 6 confirmă receptia tuturor TPDU-urilor cu numere de secvență mai mici sau egale cu 1, permitând lui A să elibereze acele tampoane și, mai mult, îl informează pe A că B poate să mai recepționeze trei TPDU-uri (adică TPDU-urile cu numere de secvență 2, 3, 4). A știe că a trimis deja numărul 2, deci se gândește că ar putea trimite 3 și 4, ceea ce și încearcă să facă. În acest moment, el este blocat și trebuie să aștepte alocarea unui tampon. Expirarea timpului pentru primirea confirmării determină retrasmisia mesajului 2 (linia 9), care poate avea loc, deși emițătorul este blocat, deoarece se vor utiliza tampoane deja alocate. În linia 10, B confirmă primirea tuturor TPDU-urilor până la 4, dar îl ține pe A încă blocat. O astfel de situație ar fi fost imposibilă cu protocolul cu fereastră glisantă prezentat în cap. 3. Următorul TPDU de la B la A alocă încă un tampon și îi permite lui A să continue.

În cazul strategiilor pentru alocarea tampoanelor de tipul celei de mai sus, eventuale probleme pot să apară în rețele neorientate pe conexiune dacă sunt pierdute TPDU-uri de control. Să privim linia 16 din fig. 6-16: B a mai alocat tampoane pentru A, dar TPDU-ul care transmitea această informație a fost pierdut. Deoarece TPDU-urile de control nu sunt numerotate sau retransmise, A va fi blocat. Pentru a preveni această situație, fiecare calculator gazdă trebuie să trimită periodic pe fiecare conexiune TPDU-uri de control ce pot conține confirmări și informații despre starea tampoanelor. În acest fel, A va fi deblocat mai devreme sau mai târziu.

A	Mesajul	B	Comentarii
1	→ <cere 8 tampoane>	→	A cere 8 tampoane
2	← <ack=15, buf=4>	←	B îi acordă tampoane numai de la 0 la 3
3	→ <seq = 0, data = m0>	→	A mai are 3 tampoane libere
4	→ <seq = 1, data = m1>	→	A mai are 2 tampoane libere
5	→ <seq = 2, data = m2>	...	Mesaj pierdut, dar A crede că mai are un singur tampon liber
6	← <ack = 1, buf = 3>	←	B confirmă 0 și 1 și permite 2-4
7	→ <seq = 3, data = m3>	→	A mai are tampoane
8	→ <seq = 4, data = m4>	→	A nu mai are tampoane libere și trebuie să se opreasă
9	→ <seq = 2, data = m2>	→	A retransmite la expirarea intervalului de timp
10	← <ack = 4, buf = 0>	←	Toate mesajele sunt confirmate, dar A este în continuare blocat
11	← <ack = 4, buf = 1>	←	A poate să îl trimită acum pe 5
12	← <ack = 4, buf = 2>	←	B a mai găsit un tampon
13	→ <seq = 5, data = m5>	→	A mai are un tampon liber
14	→ <seq = 6, data = m6>	→	A este blocat din nou
15	← <ack = 6, buf = 0>	←	A este blocat în continuare
16	... <ack = 6, buf = 4>	←	Posibilă interblocare

**Fig. 6-16.** Alocarea dinamică a tampoanelor. Săgețile indică direcția transmisiei.

Punctele de suspensie (...) indică pierderea unui TPDU.

Până acum am presupus tacit că singura limită impusă ratei de transfer a emițătorului este legată de dimensiunea spațiului alocat la receptor pentru tampoane. Deoarece prețul memoriei continuă să scadă vertiginos, ar putea deveni posibilă echiparea unei gazde cu suficient de multă memorie, astfel încât lipsa tampoanelor să pună rar probleme, dacă le va pune vreodată.

Atunci când spațiul de memorie alocat pentru tampoane nu limitează fluxul maxim, va apărea o altă limitare: capacitatea de transport a subretelei. Dacă două rutere adiacente pot să schimbe cel mult  $x$  pachete pe secundă și există  $k$  căi distincte între două calculatoare găzădă, atunci este imposibil transferul la o rată mai mare de de  $k * x$  TPDU-uri pe secundă, oricăr de multe tampoane ar fi alocate la cele două capete ale conexiunii. Dacă emițătorul se grăbește prea tare (adică trimite mai mult de  $k * x$  TPDU-uri pe secundă), rețeaua se va congestionă, deoarece nu va putea să livreze datele le fel de repede cum le primește.

Este necesar un mecanism bazat pe capacitatea de transport a subretelei și nu pe capacitatea de memorare în tampoane a receptorului. Evident, mecanismul de control al fluxului trebuie aplicat la emițător pentru a preveni existența prea multor TPDU-uri neconfirmate la acesta. Belsens (1975) a propus folosirea unei scheme cu fereastră glisantă în care emițătorul modifică dinamic dimensiunea ferestrei pentru a o potrivi la capacitatea de transport a rețelei. Dacă rețeaua poate să transporte  $c$  TPDU-uri pe secundă și durata unui ciclu (inclusiv transmisia, propagarea, timpul petrecut în cozi, prelucrarea la destinație și revenirea confirmării) este  $r$ , atunci dimensiunea ferestrei la emițător trebuie să fie  $c * r$ . Folosind o fereastră cu această dimensiune, emițătorul va putea lucra la capacitate maximă. Orice mică scădere a performanțelor rețelei va genera blocări ale emițătorului.

Pentru a ajusta periodic dimensiunea ferestrei, emițătorul poate urmări cei doi parametri, după care poate calcula dimensiunea dorită a ferestrei. Capacitatea de transport a subretelei poate fi de-

terminată pur și simplu numărând TPDU-urile confirmate într-o anumită perioadă de timp și raportând la acea perioadă. În timpul măsurătorii, emițătorul trebuie să transmită cât mai repede pentru a fi sigur că factorul care limitează numărul confirmărilor este capacitatea de transport a subrețelei și nu rata mică de emisie. Timpul necesar pentru ca un TPDU transmis să fie confirmat poate fi măsurat cu exactitate și media poate fi calculată continuu. Deoarece capacitatea de transport a rețelei disponibilă pentru orice flux dat variază în timp, dimensiunea ferestrei trebuie ajustată frecvent pentru a urmări schimbările în capacitatea de transport. Așa cum vom vedea mai departe, în Internet se folosește un mecanism similar.

### 6.2.5 Multiplexarea

Multiplexarea mai multor conversații pe conexiuni, circuite virtuale și legături fizice joacă un rol important în mai multe niveluri ale arhitecturii rețelei. În cazul nivelului transport, multiplexarea poate fi necesară din mai multe motive. De exemplu, dacă doar o singură adresă de rețea este disponibilă pe o gazdă, toate conexiunile transport de pe acea mașină trebuie să o folosească. Când un TDPU sosește este necesar un mod de a spune cărui proces trebuie dat. Această situație numită **multiplexare în sus**, este prezentată în fig. 6-17(a). În această figură, patru conexiuni transport diferite folosesc în comun aceeași conexiune rețea (de exemplu, adresa IP) către calculatorul gazdă de la distanță.

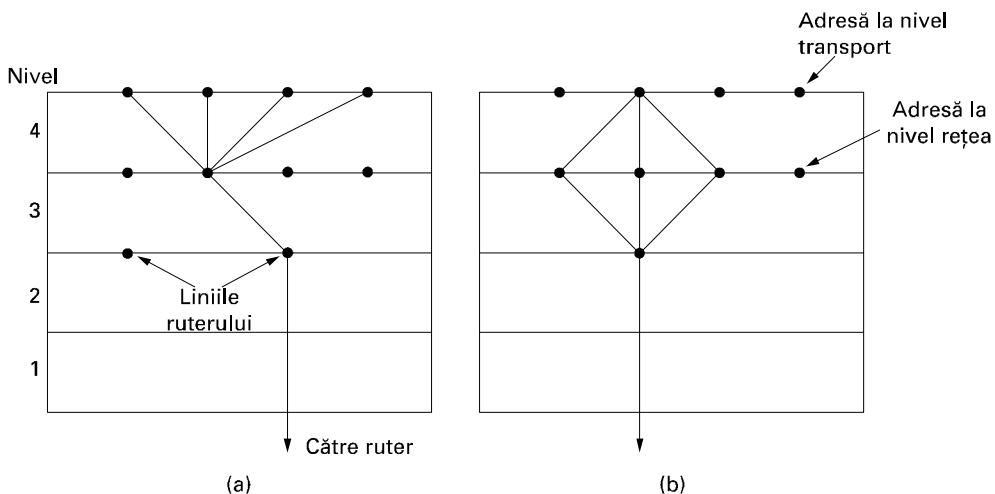


Fig. 6-17. (a) Multiplexare în sus. (b) Multiplexare în jos.

Multiplexarea poate să fie utilă nivelului transport și din alt motiv, legat de deciziile tehnice și nu de politica de prețuri ca până acum. Să presupunem, de exemplu, că o subrețea folosește intern circuite virtuale și impune rată de date maximă p fiecare dintre ele. Dacă un utilizator are nevoie de mai multă lățime de bandă decât poate oferi un circuit virtual, o soluție este ca nivelul transport să deschidă mai multe conexiuni rețea și să distribue traficul prin acestea (într-un sistem round-robin), la fel ca în fig. 6-17(b). Acest mod de operare se numește **multiplexare în jos**. În cazul a  $k$  conexiuni rețea deschise lățimea de bandă reală este multiplicată cu un factor  $k$ . Un exemplu obișnuit de multiplexare în jos apare la utilizatorii care au acasă o linie ISDN. Această linie pune la dispoziție două conexiuni separate de 64 Kbps. Folosirea ambelor pentru apelul unui distribuitor de Internet și împărțirea traficului pe ambele linii, face posibilă atingerea unei lățimi de bandă efectivă de 128 Kbps.

## 6.2.6 Refacerea după cădere

În cazul în care calculatoarele gazdă sau ruterele se întâmplă să cadă, recuperarea după o astfel de cădere devine o problemă. Dacă entitatea de transport este în întregime conținută de calculatorul gazdă, atunci revenirea după o cădere a rețelei sau a unui ruter este simplă. Dacă nivelul rețea furnizează un serviciu datagramă, atunci entitatea de transport știe să rezolve problema TPDU-urilor pierdute. Dacă nivelul rețea furnizează un serviciu orientat pe conexiune, atunci pierderea unui circuit virtual este tratată stabilind un circuit virtual nou și apoi întrebând entitatea de transport aflată la distanță care TPDU-uri a primit deja și ce TPDU-uri nu. Acestea din urmă pot fi retransmise.

Căderea unui calculator gazdă pune o problemă mult mai supărătoare. În particular, clienții pot dori să continue să lucreze imediat după ce serverul cade și repornește. Pentru a ilustra această dificultate, să presupunem că o gazdă (clientul) trimite un fișier lung unei alte gazde (serverul) folosind un protocol simplu de tip pas-cu-pas (stop-and-wait). Nivelul transport de pe server nu face decât să paseze utilizatorului TPDU-urile primite, unul câte unul. Dacă în timpul transmisiei serverul cade, la revenirea acestuia tabelele sunt reinitializate și serverul nu va mai ști precis unde a rămas.

Într-o încercare de a reveni în starea sa inițială, serverul ar putea să trimită cereri tuturor celorlalte gazde anunțând că tocmai s-a refăcut după o cădere și cerând clientilor să-l informeze despre situația tuturor conexiunilor deschise. Fiecare client poate fi în una din următoarele stări: un TPDU neconfirmat (starea S1) sau nici un TPDU neconfirmat (starea S0). Bazându-se numai pe această informație, clientul trebuie să decidă dacă să retransmită sau nu cel mai recent TPDU.

La prima vedere pare evident: atunci când află de cădere și revenirea serverului, clientul trebuie să retransmită doar dacă are TPDU-uri neconfirmate (adică este în starea S1). Totuși, o privire mai atentă descoperă dificultățile care apar în această abordare naivă. Să considerăm, ca exemplu, situația în care entitatea de transport de pe server trimite mai întâi confirmarea și apoi, după ce confirmarea a fost trimisă, pasează datele procesului aplicație. Trimiterea confirmării și transferul datelor procesului aplicație sunt două evenimente distincte care nu pot avea loc simultan. Dacă o cădere a serverului are loc după trimiterea confirmării, dar înainte de transferul datelor, clientul va primi confirmarea și se va afla în starea S0, atunci când anunțul despre cădere ajunge la el. În acest caz, clientul nu va mai retransmite (ceea ce este incorrect), crezând că TPDU-ul respectiv a fost recepționat. Această decizie a clientului va conduce la lipsa unui TPDU.

În acest moment s-ar putea spune: „Nimic mai simplu! Tot ceea ce trebuie făcut este să reprogramăm entitatea de transport astfel, încât să transfere datele mai întâi și să trimită confirmarea după aceea!”. Să vedem: dacă transferul datelor a avut loc, dar serverul cade înainte să trimită confirmarea, clientul va fi în starea S1, va retransmite și astfel vom avea un TPDU duplicat în fluxul de date către procesul aplicație.

Oricum ar fi proiectați clientul și serverul, întotdeauna vor exista situații în care revenirea după o cădere nu se va face corect. Serverul poate fi programat în două feluri: să facă mai întâi confirmarea sau să transfere mai întâi datele. Clientul poate fi programat în patru feluri: să retransmită întotdeauna ultimul TPDU, să nu retransmită niciodată, să retransmită numai în starea S0, să retransmită numai în starea S1. Rezultă astfel opt combinații, dar, aşa cum vom vedea, pentru fiecare combinație există o anumită succesiune de evenimente pentru care protocolul nu funcționează corect.

La server sunt posibile trei evenimente: trimiterea unei confirmări (A), transferul datelor la procesul aplicație (T) și cădere (C). Aceste trei evenimente pot să fie ordonate în 6 feluri: AC(T), ATC, C(AT), C(TA), TAC și TC(A). Parantezele sunt folosite pentru a indica faptul că nici A, nici

T nu pot urma după C (odata ce serverul a căzut, e bun căzut). Fig. 6-18 prezintă toate cele opt combinații ale strategiilor clientului și serverului și prezintă sevențele valide pentru fiecare din ele. Se observă că pentru orice strategie există o sevență de evenimente pentru care protocolul nu funcționează corect. De exemplu, dacă clientul retransmite întotdeauna, sevența ATC va genera un duplicat care nu poate fi detectat, în timp ce pentru celelalte două sevențe totul este în regulă.

		Strategia folosită de receptor					
		Mai întâi confirmă, apoi transferă			Mai întâi transferă, apoi confirmă		
Strategia folosită de emițător		AC(T)	ATC	C(AT)	C(TA)	T AC	TC(A)
		OK	DUP	OK	OK	DUP	DUP
Retransmite întotdeauna		LOST	OK	LOST	LOST	OK	OK
Nu retransmite niciodată		OK	DUP	LOST	LOST	DUP	OK
Retransmite în S0		LOST	OK	OK	OK	OK	DUP
Retransmite în S1							

OK = Protocolul funcționează corect  
DUP = Protocolul generează un mesaj duplicat  
LOST = Protocol pierde un mesaj

**Fig. 6-18.** Combinăriile diferitelor strategii posibile pentru server și client.

Încercarea de a reproiecta mai minuțios protocolul nu ajută. Chiar dacă clientul și serverul schimbă mai multe TPDU-uri înapoi ca serverul să transfere datele, astfel încât clientul știe exact ceea ce se petrece pe server, nu poate afla dacă serverul a căzut imediat înapoi sau imediat după transferul datelor. Concluzia se impune de la sine: data fiind condiția de bază ca două evenimente să nu fie simultane, revenirea după o cădere nu poate fi făcută transparent pentru nivelurile superioare.

În termeni mai generali, acest rezultat poate fi reformulat astfel: restartarea după o cădere a nivelului N nu poate fi făcută decât de către nivelul N+1, și aceasta doar dacă nivelul superior reține suficientă informație de stare. Așa cum am arătat mai sus, nivelul transport poate să revină după erorile nivelului rețea numai dacă fiecare capăt al conexiunii ține minte unde a rămas.

Am ajuns astfel la problema definirii precise a ceea ce înseamnă o confirmare capăt-la-capăt. În principiu, protocolul de transport este unul capăt-la-capăt și nu înlănțuit ca la nivelurile de mai jos. Să considerăm cazul unui utilizator care generează cereri de tranzacții pentru o bază de date de la distanță. Să presupunem că entitatea de transport aflată la distanță este programată să transmită mai întâi TPDU-ul nivelului superior și apoi să confirme. Chiar și în acest caz, primirea unei confirmări de către mașina utilizator nu înseamnă neapărat că mașina gazdă de la distanță a avut timp să actualizeze baza de date. De fapt, o adevărată confirmare capăt-la-capăt, a cărei primire arată că s-au efectuat toate prelucrările de către mașina de la distanță, este probabil imposibil de obținut. Acest subiect este discutat în detaliu de Saltser ș.a. (1984).

## 6.3 UN PROTOCOL SIMPLU DE TRANSPORT

Pentru a concretiza ideile discutate, în această secțiune vom studia în detaliu un exemplu de nivel transport. Vom utiliza setul abstract de primitive orientate pe conexiune prezentat în fig. 6-2. Alegerea acestor primitive orientate pe conexiune face exemplul ales similar cu (dar mai simplu decât) popularul protocol TCP.

### 6.3.1 Primitivele serviciului ales ca exemplu

Prima problemă constă în definirea exactă a primitivelor de transport. Pentru CONNECT este ușor: vom avea o rutină de bibliotecă *connect* care poate fi apelată cu parametri potriviti pentru stabilirea unei conexiuni. Parametrii sunt TSAP-ul local și cel aflat la distanță. În timpul apelului, apelantul este blocat în timp ce entitatea de transport încearcă să stabilească conexiunea. Dacă conexiunea reușește, apelantul este deblocat și poate să înceapă să transmită date.

Atunci când un proces dorește să accepte conexiuni, el face un apel *listen* specificând un anumit TSAP pe care îl ascultă. După aceasta, procesul este blocat până când un proces aflat la distanță încearcă să stabilească o conexiune cu TSAP-ul la care așteaptă.

De observat că acest model este asimetric. O parte este pasivă, executând *listen* și așteptând ca ceva să se întâpte. Cealaltă parte este activă și inițiază conexiunea. O întrebare interesantă este: ce trebuie făcut dacă partea activă începe prima? O posibilitate este ca încercarea de conectare să nu reușească dacă nu există nici un proces care să ascute la TSAP-ul aflat la distanță. O altă posibilitate este ca inițiatorul să se blocheze (eventual pentru totdeauna) până când apare un proces care să ascute la TSAP-ul aflat la distanță.

Un compromis folosit în exemplul nostru este păstrarea cererii de conexiune la receptor pentru un anumit interval de timp. Dacă un proces de pe acest calculator gazdă apelează *listen* îmânte ca timpul să expire, atunci conexiunea este stabilită, altfel conexiunea este refuzată și apelantul este deblocat întorcând un cod de eroare.

Pentru a elibera o conexiune vom folosi un apel *disconnect*. Atunci când ambele părți s-au deconectat, conexiunea este eliberată. Cu alte cuvinte, folosim un model de deconectare simetric.

Transmisia de date are exact aceeași problemă ca și stabilirea conexiunii: emițătorul este activ, dar receptorul este pasiv. Vom folosi aceeași soluție pentru transmisia de date ca și pentru stabilirea conexiunii, adică un apel activ *send* care trimită datele și un apel pasiv *receive* care blochează până când sosesc un TPDU.

Definiția concretă a serviciului constă astfel din cinci primitive: CONNECT, LISTEN, DISCONNECT, SEND și RECEIVE. Fiecare primitive corespunde unei funcții de bibliotecă care execută acea primitive. Parametrii pentru primitive și funcțiile de bibliotecă sunt:

```
connum = LISTEN(local)
connum = CONNECT (local, remote)
status = SEND(connum, buffer, bytes)
status = RECEIVE(connum, buffer, bytes)
status = DISCONNECT(connum)
```

Primitiva LISTEN anunță disponibilitatea serverului de a accepta cereri de conexiune la TSAP-ul indicat. Utilizatorul primitivei este blocat până când se face o încercare de conectare la TSAP-ul specificat. Blocarea poate să fie definitivă.

Primitiva CONNECT are doi parametri, un TSAP local (adică o adresă la nivel transport) și un TSAP aflat la distanță și încearcă să stabilească o conexiune între acestea două. Dacă reușește, ea întoarce *connum*, un număr nenegativ utilizat pentru a identifica conexiunea. Dacă nu reușește, motivul este pus în *connum* ca un număr negativ. În modelul nostru (destul de simplu), fiecare TSAP poate să participe la cel mult o singură conexiune de transport, deci un motiv pentru refuzul unei cereri de conectare poate fi că adresa la nivel transport este deja folosită. Alte câteva motive pot fi: gazda de la distanță căzută, adresă locală incorectă sau adresă de la distanță incorectă.

Primitiva SEND trimită conținutul unui tampon ca un mesaj pe conexiunea indicată, eventual divizându-l în mai multe unități, dacă este nevoie. Erorile posibile, întoarse în *status*, pot fi: nu există conexiune, adresă de tampon invalidă sau număr de biți negativ.

Primitiva RECEIVE indică faptul că apelantul așteaptă să recepționeze date. Dimensiunea mesajului este plasată în câmpul *bytes*. Dacă procesul aflat la distanță a eliberat conexiunea sau dacă adresa pentru tampon este incorectă (de exemplu, în afara spațiului de adrese al programului utilizator), funcția va întoarce un cod de eroare indicând natura problemei.

Primitiva DISCONNECT pune capăt unei conexiunii transport indicate prin parametrul *connum*. Erori posibile sunt: *connum* aparține de fapt altui proces sau *connum* nu este un identificator valid de conexiune. Codul de eroare sau 0 pentru succes sunt returnate în *status*.

### 6.3.2 Entitatea de transport aleasă ca exemplu

Înainte de a studia programul aferent entității de transport, trebuie spus că acesta este un exemplu similar celor din cap. 3: este prezentat mai mult în scop pedagogic decât pentru a fi utilizat. Mai multe detalii tehnice (precum detectarea extensivă a erorilor) care ar fi necesare unui produs real au fost lăsate deoparte pentru simplitate.

Nivelul transport utilizează primitivele serviciului rețea pentru a trimite și receptiona TPDU-uri. Trebuie să alegem primitivele serviciului rețea pe care le vom utiliza pentru acest exemplu. O posibilitate ar fi fost: un serviciu datagramă nesigur. Pentru a păstra simplitatea exemplului, nu am făcut această alegere. Folosind un serviciu datagramă nesigur, codul pentru entitatea de transport ar fi devenit foarte mare și complicat, în cea mai mare parte legat de pachete pierdute sau întârziate. Si în plus, cea mai mare parte a acestor idei au fost deja discutate în cap. 3.

Am ales în schimb un serviciu rețea sigur, orientat pe conexiune. În acest fel ne putem concentra asupra problemelor puse de nivelul transport care nu apar în nivelurile inferioare. Acestea includ, între altele, stabilirea conexiunii, eliberarea conexiunii și gestiunea tampoanelor. Un serviciu de transport simplu, construit peste un nivel rețea ATM, ar putea să semene cu acesta.

În general, entitatea de transport poate să fie parte a sistemului de operare al calculatorului gazdă sau poate să fie un pachet de funcții de bibliotecă în spațiul de adrese utilizator. Pentru simplitate, exemplul nostru a fost programat ca și cum entitatea de transport ar fi un pachet de funcții de bibliotecă, dar schimbările necesare pentru a o face parte a sistemului de operare sunt minime (fiind în principal legate de modul cum sunt adresate tampoanele).

Totuși merită remarcat faptul că, în acest exemplu, „entitatea de transport” nu este o entitate separată, ci este o parte a procesului utilizator. Mai precis, atunci când utilizatorul folosește o primitivă blocantă, de exemplu LISTEN, se blochează toată entitatea de transport. În timp ce această arhitectură este potrivită pentru un calculator gazdă cu un singur proces utilizator, pentru un calculator gazdă cu mai mulți utilizatori este normal ca entitatea de transport să fie un proces separat, distinct de toate celelalte procese.

Interfața cu nivelul rețea se face prin intermediul procedurilor *to\_net* și *from\_net*. Fiecare are săse parametri. Primul este identificatorul conexiunii, care este în corespondență bijectivă cu circuitul virtual la nivel rețea. Apoi urmează biții *Q* și *M* care, atunci când au valoarea 1, indică mesaje de control și, respectiv, faptul că mesajul continuă și în următorul pachet. După acestea urmează tipul pachetului, ales dintr-un set de șase tipuri de pachete prezentate în fig. 6-19. La sfârșit se găsește un indicator către zona de date și un întreg care indică numărul de octeți de date.

Tip pachet	Explicații
CALL_REQUEST	Trimis pentru a stabili conexiunea
CALL_ACCEPTED	Răspuns la CALL_REQUEST
CLEAR_REQUEST	Trimis pentru a elibera conexiunea
CLEAR_CONFIRMATION	Răspuns la CLEAR_REQUEST
DATA	Pentru transport de date
CREDIT	Pachet de control pentru gestionarea ferestrei

Fig. 6-19. Tipurile de pachete folosite la nivel rețea.

La apelurile *to\_net*, entitatea de transport completează toți parametrii pentru ca nivelul rețea să-i poată citi; la apelurile *from\_net* nivelul rețea dezasamblează un pachet sosit pentru entitatea de transport. Transferul informației sub forma unor parametri ai unei proceduri și nu a pachetului de trimis sau de recepționat protejează nivelul transport de toate detaliile protocolului nivelului rețea. Dacă entitatea de transport încearcă să trimită un pachet atunci când fereastra glisantă corespunzătoare a circuitului virtual este plină, ea va fi blocată într-un apel *to\_net* până când va fi spațiu în fereastră. Mecanismul este transparent pentru entitatea de transport și este controlat de nivelul rețea prin comenzi ca *enable\_transport\_layer* și *disable\_transport\_layer*, analoage celor descrise în protocoalele din cap. 3. Gestionarea ferestrei de pachete este de asemenea făcută de către nivelul rețea.

Pe lângă acest mecanism transparent de suspendare mai există două proceduri (care nu sunt prezentate aici), *sleep* și *wakeup*, apelate de entitatea de transport. Procedura *sleep* este apelată atunci când entitatea de transport este blocată logic în așteptarea unui eveniment extern, în general sosirea unui pachet. După ce a fost apelat *sleep*, entitatea de transport (și procesul utilizator, bineînțeles) sunt blocate.

Codul entității de transport este prezentat în fig. 6-20. Orice conexiune este într-una din următoarele șapte stări:

1. *IDLE* - conexiunea nu a fost încă stabilită
2. *WAITING* - a fost executat CONNECT și trimis: CALL\_REQUEST
3. *QUEUED* - a sosit un CALL\_REQUEST, nu a fost executat încă nici un apel LISTEN
4. *ESTABLISHED* - conexiunea a fost stabilită
5. *SENDING* - utilizatorul așteaptă permișionarea de a trimite un pachet
6. *RECEIVING* - a fost apelat RECEIVE
7. *DISCONNECTING* - un apel DISCONNECT a fost făcut local

Pot să apară tranziții între stări ori de câte ori are loc unul din următoarele evenimente: este executată o primitivă, sosește un pachet sau expiră un interval de timp stabilit.

Procedurile prezentate în fig. 6-20 sunt de două tipuri. Majoritatea sunt apelate direct de către programele utilizator, totuși *packet\_arrival* și *clock* sunt diferite. Execuția lor este declanșată de evenimente externe: sosirea unui pachet și, respectiv, avansul ceasului. Ele sunt de fapt rutine de întreprere. Vom presupune că acestea nu sunt niciodată apelate atât timp cât rulează o altă procedură a entității de transport. Acestea pot fi executate numai atunci când procesul utilizator este în așteptare sau atât timp cât el rulează în afara entității de transport. Această proprietate este crucială pentru funcționarea corectă a entității de transport.

```

#define MAX_CONN 32          /* numărul maxim de conexiuni deschise simultan */
#define MAX_MSG_SIZE 8192    /* dimensiunea maximă a unui mesaj în octeți */
#define MAX_PKT_SIZE 512     /* dimensiunea maximă a unui pachet în octeți */
#define TIMEOUT 20
#define CRED 1
#define OK 0

#define ERR_FULL -1
#define ERR_REJECT -2
#define ERR_CLOSED -3
#define LOW_ERR -3

typedef int transport_address;
typedef enum {CALL_REQ,CALL_ACC, CLEAR_REQ, CLEAR_CONF, DATA_PKT,CREDIT}
pkt_type;
typedef enum {IDLE, WAITING, QUEUED, ESTABLISHED, SENDING, RECEIVING, DISCONN}
cstate;
/* Variabile globale */
transport_address listen_address;           /* adresa locală care este ascultată */
int listen_conn;                           /* identificatorul de conexiune pentru listen */
unsigned char data[MAX_PKT_SIZE]           /* zona pentru pachetele de date */

struct conn {
    transport_address local_address, remote_address;      /* starea conexiunii */
    cstate state;                                         /* pointer la tamponul de recepție */
    unsigned char *user_buf_addr;                         /* contor de emisie/recepție*/
    int byte_count;                                      /* setat atunci când este primit un pachet CLEAR_REQ*/
    int clr_req_received;                                /* folosit pentru a evita așteptările infinite */
    int timer;                                           /* numărul de mesaje care poate fi trimis */
    int credits;                                         /* poziția 0 nu e folosită */
}; conn[MAX_CONN + 1]

/* prototipuri */
void sleep(void);
void wakeup(void);
void to_net(int cid, int q, int *m, pkt_type pt, unsigned char *p, int bytes);
void from_net(int *cid, int *q, int *m, pkt_type *pt,unsigned char *p, int *bytes);

int listen (transport_address t)
{
/* Utilizatorul vrea să stabilească o conexiune. Trebuie să vadă dacă CALL_REQ a venit deja */
    int i, found=0;

    for (i=1; i<= MAX_CONN; i++)                  /* căută în tabela de conexiuni CALL_REQ*/
        if (conn[i].state == QUEUED && conn[i].local_address == t) {
            found = i;
            break;
        }
    if (found == 0) {                            /* nu a găsit nici un CALL_REQ. Așteaptă până când sosesc
                                                unul sau până când expira intervalul de timp de așteptare */
        listen_address = t; sleep(); i = listen_conn;
    }
    conn[i].state = ESTABLISHED;                /* conexiunea este stabilită */
}

```

```

conn[i].timer = 0                                /* ceasul nu este folosit */
listen_conn = 0;                                 /* 0 este presupus a fi o adresă invalidă */
to_net(i, 0, 0, CALL_ACC, 0);                  /* spune niv. rețea să accepte cererea de conexiune */
return(i);                                       /* întoarce identificatorul conexiunii */
}

int connect (transport_address l, transport_address r)
{                                                 /* Utilizatorul dorește să stabilească o conexiune cu un proces aflat
   int i;
   struct conn *cptr;

   data[0] = r;                                    /* pachetul CALL_REQ are nevoie de acestea */
   data[1] = l;                                     /* caută în tabelă înapoi */
   i = MAX_CONN;
   while (conn[i].state != IDLE && i>1) i = i-1;
   if (conn[i].state == IDLE) {                     /* Face o intrare în tabelă */
      cptr = &conn[i];
      cptr->local_address = l;
      cptr->remote_address = r;
      cptr->state = WAITING;
      cptr->clr_req_received = 0;
      cptr->credits = 0;
      cptr->timer = 0;
      to_net(i, 0, 0, CALL_REQ, data, 2);
      sleep();                                     /* așteaptă CALL_ACC sau CLEAR_REQ */
      if (cptr->state == ESTABLISHED) return (i);
      if (cptr->clr_req_received ) {               /* cererea de conexiune este refuzată */
         cptr->state = IDLE                      /* înapoi în starea IDLE */
         to_net(i, 0, 0, CLEAR_CONF, data, 0);
         return(ERR_REJECT);
      }
   }
   else return (ERR_FULL)                         /* conexiunea este refuzată: insuficient spațiu în tabele */
}

int send (int cid, unsigned char bufptr[], int bytes)
{                                                 /* Utilizatorul dorește să trimită un mesaj */
   int i, count, m;
   struct conn *cptr = &conn[cid];

   /* Intră în starea SENDING */
   cptr->state = SENDING;
   cptr->byte_count = 0;
   if (cptr->clr_req_received == 0 && cptr->credits == 0) sleep();
   if (cptr->clr_req_received == 0) {             /* Există credite; împarte mesajul în pachete dacă este cazul */
      do {
         if (bytes - cptr->byte_count > MAX_PKT_SIZE) {
            /* mesaj format din mai multe pachete */
            count = MAX_PKT_SIZE;
            m = 1;
            /* mai urmează pachete */
         }
      }
   }
}

```

```

    } else { /* un mesaj format dintr-un pachet */
        count = bytes — cptr->byte->count;
        m=0;                                /* ultimul pachet al acestui mesaj */
    }
    for (i=0; i<count; i++) data[i]= bufptr[cptr->byte_count+1];
    to_net(cid, 0, m,DATA_PKT, data, count);           /* trimite un pachet */
    cptr->byte_count=cptr->byte_count + count; /* incrementează nr. octetilor trimisi */
} while (cptr->byte_count < bytes); /* ciclează până când întregul mesaj este trimis */
cptr->credits--;
cptr->state = ESTABLISHED;
return(OK);
} else { cptr->state = ESTABLISHED;
return(ERR_CLOSED); /* întoarce insucces: celălalt capăt vrea să se deconecteze */
}
}

int receive (int cid, unsigned char bufptr[], int *bytes)
{
/* Utilizatorul este gata să primească un mesaj */
struct conn *cptr = &conn[cid];

if (cptr->clr_req_received == 0) { /* Conexiunea încă stabilită; încearcă să recepționeze */
    cptr->state = RECEIVING;
    cptr->user_buf_addr = bufptr;
    cptr->byte_count = 0;
    data[0] = CRED;
    data[1] = 1;
    to_net(cid, 1, 0, CREDIT, data, 2);           /* trimite CREDIT */
    sleep();                                     /* se blochează în aşteptarea datelor */
    *bytes = cptr->byte_count;
}
cptr->state = ESTABLISHED;
return(cptr->clr_req_received ? ERR_CLOSED : OK);
}

int disconnect (int cid)
{
/* Utilizatorul vrea să se deconecteze */
struct conn *cptr = &conn[cid];

if (cptr -> clr_req_received) {                  /* cealaltă parte a inițiat deconectarea */
    cptr->state = IDLE;                         /* conexiunea este eliberată */
    to_net(cid, 0, 0, CLEAR_CONF, data, 0);
} else {
    cptr->state = DISCONNECT;                   /* se inițiază terminarea */
    to_net(cid, 0, 0, CLEAR_REQ, data, 0);       /* conexiunea nu este eliberată până când
                                                cealaltă parte nu-și dă acordul */
}
return (OK);
}

void packet_arrival (void)
{
/* A sosit un pachet; urmează prelucrarea lui */
int cid;                                         /* conexiunea pe care a sosit pachetul */

```

```

int count, i q, m;
pkt_type ptype;                                /* CALL_REQ, CALL_ACC, CLEAR_REQ,
unsigned char data [MAX_MKT_SIZE];           /* CLEAR_CONF, DATA_PKT, CREDIT */
                                                 /* date din pachetul care sosește */
struct conn *cptr;
from_net(&cid, &q, &ptype, data, &count);          /* preia pachetul */
cptr = &conn[cid];
switch (ptype) {
    case CALL_REQ:        /* utilizatorul de la distanță vrea să stabilească o conexiune */
        cptr->local_address = data[0];
        cptr->remote_address = data[1];
        if (cptr->local_address == listen_address) {
            listen_conn = cid;
            cptr->state = ESTABLISHED;
            wakeup();
        } else {
            cptr->state = QUEUED;
            cptr->timer = TIMEOUT;
        }
        cptr->clr_req_received = 0;
        cptr->credits = 0;
        break;
    case CALL_ACC:         /* utilizatorul de la distanță a acceptat CALL_REQ trimis */
        cptr->state = ESTABLISHED;
        wakeup();
        break;
    case CLEAR_REQ:        /* utilizatorul de la distanță vrea să se deconecteze sau să refuze un apel */
        cptr->clr_req_received = 1;
        if (cptr->state == DISCONN) cptr->state = IDLE;
        if (cptr->state == WAITING || cptr->state == RECEIVING
            || cptr->state == SENDING) wakeup();
        break;
    case CLEAR_CONF:       /* utilizatorul de la distanță acceptă deconectarea */
        cptr->state = IDLE;
        break;
    case CREDIT:           /* utilizatorul de la distanță așteaptă date */
        cptr->credits += data[1];
        if (cptr->state == SENDING) wakeup();
        break;
    case DATA_PKT:         /* au fost trimise date */
        for (i=0; i<count; i++)
            cptr->user_buff_addr[cptr->byte_count + i] = data[i];
        cptr->byte_count += count;
        if (m == 0) wakeup();
    }
}
}

```

```

void clock( void)
{
    /* la fiecare interval de timp al ceasului se verifică eventualele
       depășiri ale timpilor de așteptare pentru cererile aflate în starea QUEUED */
    int i;
    struct conn *cptr;

    for (i=1; i<=MAX_CONN, i++) {
        cptr = &conn[i];
        if (cptr->timer > 0) {                                /* ceasul funcționa */
            cptr->timer--;
            if (cptr->timer == 0) {                            /* timpul a expirat */
                cptr->state = IDLE;
                to_net(i, 0, 0, CLEAR_REQ, data, 0);
            }
        }
    }
}

```

**Fig. 6-20.** Entitatea de transport aleasă ca exemplu.

Existența bitului  $Q$  în antetul pachetului ne permite să evităm timpul suplimentar introdus de antetul protocolului de transport. Mesajele de date obișnuite sunt trimise ca pachete de date cu  $Q=0$ . Mesajele legate de protocolul de transport, dintre care există numai unul (CREDIT) în exemplul nostru, sunt trimise ca pachete de date cu  $Q=1$ . Aceste mesaje de control sunt detectate și prelucrate de entitatea de transport receptoare.

Structura de date de bază folosită de entitatea de transport este vectorul *conn*, care are câte o înregistrare pentru fiecare conexiune posibilă. În înregistrare sunt menținute starea conexiunii, incluzând adresa de nivel transport la fiecare capăt, numărul de mesaje trimise și recepționate pe conexiune, starea curentă, un indicator (pointer) la tamponul utilizator, numărul de octeți ai mesajului trimis sau recepționat, un bit indicând dacă utilizatorul aflat la distanță a apelat DISCONNECT, un ceas, și un contor folosit pentru a permite transmiterea mesajelor. Nu toate aceste câmpuri sunt folosite în exemplul nostru simplu, dar o entitate de transport completă ar avea nevoie de toate, poate chiar de mai multe. Fiecare element din *conn* este inițializat cu starea *IDLE*.

Atunci când un utilizator apelează CONNECT, nivelului rețea i se va cere să trimită un pachet CALL\_REQUEST către mașina de la distanță și utilizatorul este blocat. Atunci când pachetul CALL\_REQUEST ajunge de partea cealaltă, entitatea de transport este întreruptă pentru a executa *packet\_arrival*, care verifică dacă utilizatorul local ascultă la adresa specificată. Dacă da, atunci este trimis înapoi un pachet CALL\_ACCEPTED și utilizatorul de la distanță este trezit; dacă nu, atunci cererea CALL\_REQUEST este pusă într-o coadă pentru un interval de timp TIMEOUT. Dacă în acest interval este făcut un apel LISTEN, atunci conexiunea este stabilită; dacă nu, conexiunea este refuzată la sfârșitul intervalului de timp cu un pachet CLEAR\_REQUEST ca să nu fie blocată pe timp nedefinit.

Desi am eliminat antetul pentru protocolul de transport, tot mai trebuie găsită o modalitate pentru a determina cărei conexiuni transport îi aparține un anumit pachet, deoarece pot exista simultan mai multe conexiuni. Cea mai simplă soluție este folosirea numărului de circuit virtual de la nivel rețea și ca număr de conexiune transport. În plus, numărul de circuit virtual poate fi folosit și ca index în vectorul *conn*. Atunci când un pachet sosește pe circuitul virtual  $k$  la nivel rețea, el îi va apartine conexiunii  $k$  a cărei stare este păstrată în *conn[k]*. La inițierea unei conexiuni, numărul de cone-

xiune este ales de entitatea de transport care a inițiat-o. Pentru cererile de conexiune, nivelul rețea alege ca număr de conexiune orice număr de circuit virtual care nu a fost încă folosit.

Pentru a evita gestionarea tampoanelor în interiorul entității de transport, este folosit un mecanism de gestiune a fluxului diferit de fereastra glisantă tradițională. Când un utilizator apelează RECEIVE, este trimis un **mesaj de credit** entității de transport de pe mașina care va transmite și este înregistrat în vectorul *conn*. Atunci când este apelat SEND, entitatea de transport verifică dacă a sosit vreun credit pe conexiunea respectivă. Dacă da, mesajul este trimis (chiar și în mai multe pachete, dacă este cazul) și credit este decrementat; dacă nu, atunci entitatea de transport se blochează până la sosirea unui credit. Mecanismul garantează că nici un mesaj nu este trimis decât dacă cealaltă parte a apelat deja RECEIVE. Ca rezultat, ori de câte ori sosește un mesaj, există cu siguranță un tampon disponibil pentru el. Această schemă poate fi ușor generalizată pentru a permite receptorilor să ofere tampoane multiple și să ceară mai multe mesaje.

Trebue reținută simplitatea codului din fig. 6-20. O entitate de transport reală ar verifica în mod normal validitatea tuturor parametrilor furnizați de utilizator, ar putea să revină după căderea rețelei, ar putea rezolva coliziunile la apel și ar susține un serviciu transport mult mai general, care ar include facilități cum sunt întreruperile, datagramele și versiunile nonblocante ale primitivelor SEND și RECEIVE.

### 6.3.3 Exemplul văzut ca un automat finit

Scrierea unei entități de transport este o muncă dificilă și riguroasă, în special pentru protocolele reale. Pentru a reduce probabilitatea de apariție a unei erori, adeseori este util să reprezentăm protocolul ca un automat finit.

Am văzut deja că protocolul nostru folosit ca exemplu are șapte stări pentru fiecare conexiune. Este de asemenea posibil să izolăm cele douăsprezece evenimente care pot să apară pentru a schimba starea unei conexiuni. Cinci dintre aceste evenimente sunt cele cinci primitive ale serviciului de transport. Alte șase sunt reprezentate de sosirile celor șase tipuri distințe de pachete. Ultimul este expirarea intervalului de timp stabilit. Fig. 6-21 arată acțiunile principale ale protocolului sub forma unei matrice. Pe coloane sunt prezentate stările, iar pe linii cele 12 evenimente.

Fiecare intrare în matrice (adică în automatul finit) din fig. 6-21 are până la trei câmpuri: un predicat, o acțiune și o nouă stare. Predicatul indică condițiile în care acțiunea este executată. De exemplu, pentru intrarea din colțul stânga-sus, dacă este executat un LISTEN și nu mai există spațiu în tabele (predicatul P1), atunci LISTEN eșuează și starea rămâne aceeași. Pe de altă parte, dacă un pachet CALL\_REQUEST a sosit deja la adresa de nivel transport la care se face LISTEN (predicatul P2), atunci conexiunea este stabilită imediat. O altă posibilitate este ca P2 să fie fals, adică nici un CALL\_REQUEST nu a fost primit, caz în care conexiunea rămâne în starea IDLE în așteptarea unui pachet CALL\_REQUEST.

Merită să subliniem că alegerea stărilor folosite în matrice nu este în întregime determinată de protocolul însuși. În acest exemplu, nu există nici o stare *LISTENING*, care ar fi putut urma în mod normal apelului LISTEN. Nu a fost introdusă o stare *LISTENING* deoarece o stare este asociată cu o intrare în tabela de conexiuni și la un apel LISTEN nu se creează nici o intrare în tabelă. De ce? Pentru că am decis să folosim identificatorii circuitelor virtuale de la nivel rețea ca identificatori pentru conexiune și, pentru un apel LISTEN, numărul circuitului virtual este în cele din urmă ales de nivelul rețea atunci când sosește un CALL\_REQUEST.

Aceștia de la A1 la A12 sunt acțiuni importante, precum trimitera pachetelor sau rearmarea ceasurilor. Nu sunt menționate toate acțiunile minore, cum ar fi inițializarea unor câmpuri ale înregistrării atașate conexiunii. Dacă o acțiune implică trezirea unui proces în aşteptare, atunci acțiunile care urmează trezirii procesului sunt considerate și ele. De exemplu, dacă un pachet CALL\_REQUEST sosește și există un proces adormit care îl așteaptă, atunci transmiterea pachetului CALL\_ACCEPT este considerată ca făcând parte din acțiunea care urmează recepției lui CALL\_REQUEST. După ce este efectuată fiecare acțiune, conexiunea ajunge într-o nouă stare, așa cum apare și în fig. 6-21.

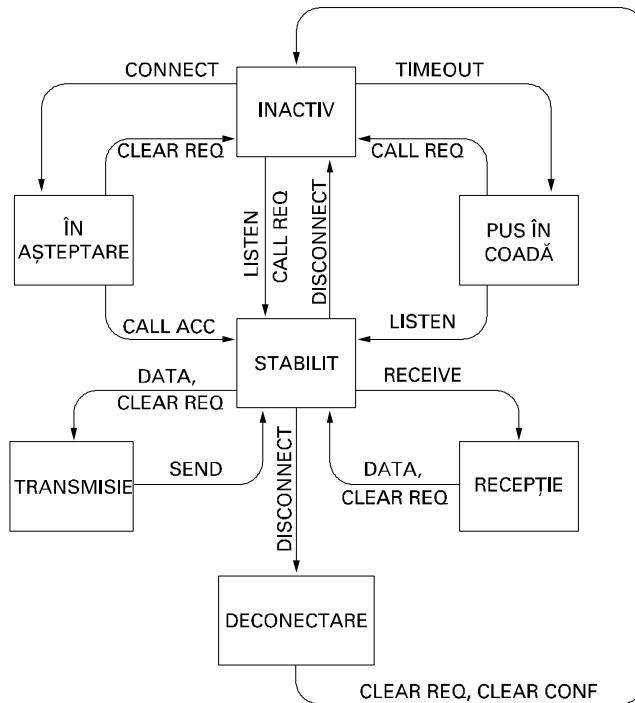
		Stare						
		Idle	Waiting	Queued	Established	Sending	Receiving	Disconnecting
Primitive	LISTEN	P1: ~1/Idle P2: A1/Estab P2: A2/Idle		~/Estab				
	CONNECT	P1: ~/Idle P1: A3/Wait						
	DISCONNECT				P4: A5/Idle P4: A6/Disc			
	SEND				P5: A7/Estab P5: A8/Send			
	RECEIVE				A9/Receiving			
	Call_req	P3: A1/Estab P3: A4/Que'd						
	Call_acc		~/Estab					
	Clear_req		~/Idle		A10/Estab	A10/Estab	A10/Estab	~/Idle
	Clear_conf							~/Idle
	DataPkt						A12/Estab	
Pachete sosite	Credit				A11/Estab	A7/Estab		
	Timeout			~/Idle				
		Predicte		Acțiuni				
		P1: Tabela de conexiuni plină	P2: Cerere de conexiune în aşteptare	P3: Apel LISTEN în aşteptare	P4: Pachet Clear_req în aşteptare	P5: Credit disponibil	A1: Trimite Call_acc	A7: Trimite mesaj
							A2: Așteaptă Call_req	A8: Așteaptă credit
							A3: Trimite Call_req	A9: Trimite credit
							A4: Poarte ceasul	A10: Setează indicator
							A5: Trimite Clear_conf	Clr_req_received
							A6: Trimite Clear_req	A11: Înregistrează credit
								A12: Acceptă mesajul

**Fig. 6-21.** Protocolul ales ca exemplu, reprezentat ca un automat finit. Fiecare intrare are un predicat optional, o acțiune optională și o nouă stare. Caracterul tilda indică faptul că nici o acțiune importantă nu este efectuată. Bara deasupra predicatorului este reprezentarea pentru predicatorul negat. Intrările vide corespund unor sevențe de evenimente imposibile sau eronate.

Avantajul reprezentării protocolului ca o matrice este întreit. În primul rând, în această formă este mult mai simplu pentru programator să verifice sistematic fiecare combinație stare/ eveniment/ acțiune. În implementările reale, unele combinații vor fi folosite pentru tratarea erorilor. În fig. 6-21 nu s-a făcut nici o distincție între situațiile imposibile și cele care sunt numai ilegale. De exemplu, dacă o conexiune este în starea *WAITING*, evenimentul *DISCONNECT* este imposibil deoarece utilizatorul este blocat și nu poate să facă nici un apel. Pe de altă parte, în starea *SENDING* nu pot sosi date deoarece nu a fost generat nici un credit. Sosirea unui pachet de date va fi o eroare a protocolului.

Al doilea avantaj al reprezentării protocolului ca o matrice este legat de implementarea acestuia. Într-un vector bidimensional elementul  $act[i][j]$  ar putea fi văzut ca un indicator la procedura care tratează evenimentul i atunci când starea este j. O implementare posibilă este codificarea entității de transport ca o buclă în care se așteaptă la început producerea unui eveniment. După producerea evenimentului, este identificată conexiunea implicată și este extrasă starea ei. Cunoscând acum starea și evenimentul, entitatea de transport nu face decât să accesze vectorul *act* și să apeleze procedura adecvată. Această abordare ar conduce la o arhitectură mult mai regulată și mai simetrică decât cea a entității de transport implementate de noi.

Al treilea avantaj al abordării folosind un automat finit este legat de descrierea protocolului. În unele standarde, protocolele sunt specificate ca un automat finit de tipul celui din fig. 6-21. Trecea-re de la acest tip de descriere la o entitate de transport funcțională este mult mai ușoară dacă entitatea de transport este și ea modelată cu ajutorul unui automat finit.



**Fig. 6-22.** Protocolul dat ca exemplu, în reprezentare grafică. Pentru simplificare, tranzițiile care lasă starea conexiunii nemodificată au fost omise.

Cel mai important dezavantaj este că această abordare poate să fie mai dificil de înțeles decât metoda directă pe care am utilizat-o la început. Totuși, această problemă poate fi rezolvată, fie și parțial, desenând automatul finit ca un graf, aşa cum am făcut-o în fig. 6-22.

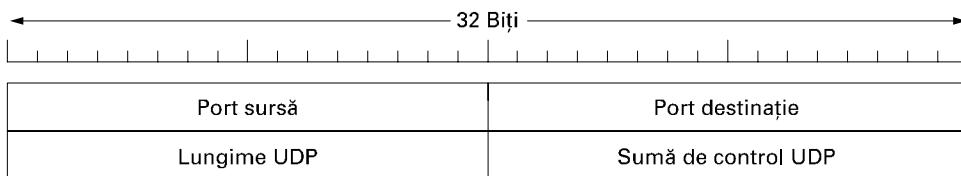
## 6.4 PROTOCOALE DE TRANSPORT PRIN INTERNET: UDP

Internet-ul are două protocole principale în nivelul de transport: unul neorientat pe conexiune și unul orientat pe conexiune. În următoarele secțiuni o să le studiem pe ambele. Protocolul neorientat pe conexiune se numește UDP. Protocolul orientat pe conexiune se numește TCP. O să începem cu UDP-ul deoarece în esență este la fel ca IP-ul cu un mic antet adăugat. De asemenei, o să studiem și două aplicații ale UDP-ului.

### 6.4.1 Introducere în UDP

Setul de protocole Internet suportă un protocol de transport fără conexiune, **UDP** (User Protocol – Protocol cu Datagrame Utilizator). UDP oferă aplicațiilor o modalitate de a trimite datagrame IP încapsulate și de a le transmite fără a fi nevoie să stabilească o conexiune. UDP este descris în RFC 768.

UDP transmite **segmente** constând într-un antet de 8 octeți urmat de informația utilă. Antetul este prezentat în fig. 6-23. Cele două porturi servesc la identificarea punctelor terminale ale mașinilor sursă și destinație. Când ajunge un pachet UDP, conținutul său este predat procesului atașat portului destinație. Această atașare are loc atunci când se folosește o simplă procedură de nume sau ceva asemănător, aşa cum am văzut în fig. 6-6 pentru TCP (procesul de legătură este același pentru UDP). De fapt, valoarea cea mai importantă dată de existența UDP-ului față de folosirea doar a IP-ului simplu, este aceea a adăugării porturilor sursă și destinație. Fără câmpurile portului, nivelul de transport nu ar ști ce să facă cu pachetul. Cu ajutorul lor, segmentele se livrează corect.



**Fig. 6-23.** Antetul UDP.

Portul sursă este în primul rând necesar atunci când un răspuns trebuie transmis înapoi la sursă. Prin copierea câmpului *port sursă* din segmentul care sosește în câmpul *port destinație* al segmentului care pleacă, procesul ce trimite răspunsul specifică ce proces de pe mașina de trimitere urmează să-l primească.

Câmpul *lungime UDP* include antetul de 8 octeți și datele. Câmpul *sumă de control UDP* este optional și stocat ca 0 (zero) dacă nu este calculat (o valoare de adevăr 0 rezultată în urma calculelor

este memorată ca un șir de biți 1). Dezactivarea acestuia este o prostie, excepție făcând cazul în care calitatea informației chiar nu contează (de exemplu, transmisia vocală digitalizată).

Merită probabil menționate, în mod explicit, unele dintre lucrurile pe care UDP-ul *nu le face*. Nu realizează controlul fluxului, controlul erorii, sau retransmiterea unui segment incorrect primit. Toate acestea depind de procesele utilizatorului. Ceea ce face este să ofere protocolului IP o interfață cu facilități adăugate de demultiplexare a mai multor procese, folosind porturi. Aceasta este tot ceea ce face UDP-ul. Pentru aplicațiile care trebuie să aibă un control precis asupra fluxului de pachete, controlului erorii sau cronometrarea, UDP-ul furnizează doar ceea ce “a ordonat doctorul”.

Un domeniu unde UDP-ul este în mod special util este acela al situațiilor client-server. Deseori, clientul trimite o cerință scurtă server-ului și așteaptă înapoi un răspuns scurt. Dacă se pierde ori cererea ori răspunsul, clientul poate pur și simplu să încerce din nou după ce a expirat timpul. Nu numai că va fi mai simplu codul, dar sunt necesare și mai puține mesaje (câte unul în fiecare direcție) decât la un protocol care solicită o inițializare inițială.

O aplicație care folosește UDP-ul în acest fel este DNS (Domain Name System, rom: Sistem de rezolvare de nume), pe care îl vom studia în cap. 7. Pe scurt, un program care trebuie să caute adresele de IP ale unor nume gazdă, de exemplu *www.cs.berkley.edu*, poate trimite un pachet UDP, conținând numele gazdă, către un server DNS. Serverul răspunde cu un pachet UDP conținând adresa de IP a gazdei. Nu este necesară nici o inițializare în avans și nici o închidere de sesiune. Doar două mesaje traversează rețea.

#### 6.4.2 Apel de procedură la distanță (Remote Procedure Call)

Într-un anume sens, trimitera unui mesaj către o stație la distanță și primirea înapoi a unui răspuns seamănă mult cu realizarea unei funcții de apel într-un limbaj de programare. În ambele cazuri se începe cu unul sau mai mulți parametri și se primește înapoi un rezultat. Această observație le-a făcut pe unele persoane să încerce să organizeze interacțiunile cerere-răspuns în rețele pentru a fi puse împreună sub forma apelurilor procedurale. Un astfel de aranjament face aplicațiile de rețea mai ușor programabile și mai abordabile. De exemplu, imaginați-vă doar procedura numită *get\_IP\_address(host\_name)* care funcționează prin trimitera unui pachet UDP către un server DNS și așteptarea răspunsului, cronometrând și încercând încă o dată, dacă răspunsul nu apare suficient de rapid. În acest fel, toate detaliile de rețea pot fi ascunse programatorului.

Efortul cel mai important în acest domeniu a fost depus de către Birell și Nelson (1984). Rezumând, ce au sugerat Birell și Nelson a fost să permită programelor să apeleze proceduri localizate pe stații aflate la distanță. Când procesul de pe mașina 1 invocă o procedură de pe mașina 2, procesul apelant de pe prima mașină este suspendat și execuția procedurii invocate are loc pe cea de-a doua. Informația poate fi transportată de la cel care apelează la cel care este apelat în parametri și se poate întoarce în rezultatul procedurii. Nici un transfer de mesaje nu este vizibil pentru programator. Tehnica este cunoscută sub numele de **RPC (Remote Procedure Call**, rom: Apel de procedură la distanță), și a devenit baza pentru multe aplicații de rețea. În mod tradițional, procedura care apelează este cunoscută ca fiind clientul și procedura apelată ca fiind serverul și vom folosi denumiri și aici.

Ideeoa din spatele RPC-ului este aceea de a face un apel de procedură la distanță să arate pe cât posibil ca unul local. În forma cea mai simplă, pentru apelarea unei proceduri la distanță, programul client trebuie să fie legat cu o mică procedură de bibliotecă, numită **client stub** (rom: ciot), care reprezintă procedura server-ului în spațiul de adresă al clientului. În mod similar, serverul este legat cu

o procedură numită **server stub**. Aceste proceduri ascund faptul că apelul de procedură de la client la server nu este local.

Pașii efectivi ai realizării unui RPC sunt prezențați în fig. 6-24. Pasul 1 este cel în care clientul apelează stub-ul client. Acest apel este un apel de procedură locală, cu parametrii introdusi în stivă în modul obișnuit. Pasul 2 constă în împachetarea parametrilor de către stub-ul client într-un mesaj și realizarea unui apel de sistem pentru a trimite mesajul. Împachetarea parametrilor este denumită **marshaling** (rom: împachetare). Pasul 3 constă în faptul că nucleul sistemului de operare trimite un mesaj de la mașina client la mașina server. Pasul 4 constă în trimiterea de către nucleu a pachetelor care sosesc la stub-ul server. În sfârșit, pasul 5 constă în faptul că stub-ul server apelează procedura server cu parametrii despachetați. Răspunsul urmează aceeași cale și în cealaltă direcție.

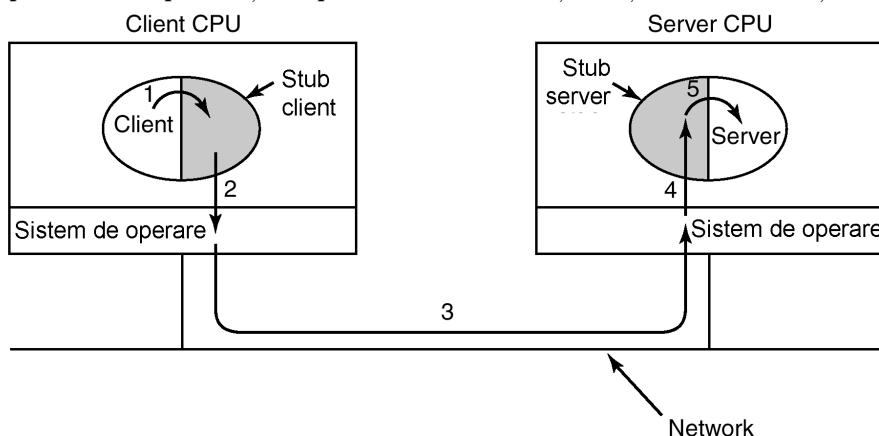


Fig. 6-24. Pașii pentru a crea un apel de procedură la distanță. Stub-urile sunt hașurate.

Elementul cheie de reținut aici este acela că procedura client, scrisă de către utilizator, doar face un apel normal de procedură (adică local) către stub-ul client, care are același nume cu procedura server. Cum procedura client și stub-ul client se găsesc în același spațiu de adresă, parametrii sunt transferați în modul obișnuit. În mod asemănător, procedura server este apelată de o procedură din spațiul său de adresă cu parametrii pe care îi așteaptă. Pentru procedura server nimic nu este neobișnuit. În felul acesta, în loc ca intrarea/ieșirea să se facă pe socluri, comunicația de rețea este realizată imitând o procedură de apelare normală.

În ciuda eleganței conceptului de RPC, „sunt câțiva șerpi care se ascund prin iarbă”. Unul mare este utilizarea parametrilor pointer. În mod normal, transmiterea unui pointer către procedură nu este o problemă. Procedura apelată poate folosi pointer-ul în același mod în care poate să o facă și cel care o apelează, deoarece ambele proceduri se găsesc în același spațiu de adrese virtuale. Cu RPC-ul, transmiterea pointer-ilor este imposibilă deoarece clientul și server-ul se găsesc în spații de adresă diferite.

În anumite cazuri, se pot folosi unele trucuri pentru a face posibilă transmiterea pointer-ilor. Să presupunem că primul parametru este un pointer către un întreg,  $k$ . Stub-ul client poate împacheta variabila  $k$  și să o trimită server-ului. Atunci, server stub creează un pointer către  $k$  și-l transmite procedurii server, exact așa cum aceasta se așteaptă. Când procedura server cedează controlul server stub, acesta din urmă trimite variabila  $k$  înapoi clientului, unde noul  $k$  este copiat peste cel vechi, în caz că serverul l-a schimbat. În fapt, secvența standard de apelare apel-prin-referință a fost înlocuită

de copiază-restaurează (eng.: copy-restore). Din păcate, acest truc nu funcționează întotdeauna, de exemplu dacă un pointer este către un grafic sau altă structură complexă de date. Din acest motiv, trebuie puse anumite restricții asupra parametrilor procedurilor apelate la distanță.

O a doua problemă este aceea că în limbajelor mai puțin bazate pe tipuri, cum ar fi C-ul, este perfect legal să scrii o procedură care calculează produsul scalar a doi vectori, fără a specifica dimensiunea vreunui dintre ei. Fiecare poate fi terminat printr-o valoare specială cunoscută doar de către procedura apelată și de cea apelantă. În aceste condiții, în mod cert este imposibil pentru stub-ul client să împacheteze parametrii: nu are nici o modalitate de a determina cât de mult spațiu ocupă aceștia.

O a treia problemă este aceea că nu întotdeauna este posibilă deducerea tipurilor parametrilor, nici măcar dintr-o specificație formală sau din cod în sine. Un exemplu este *printf*, care poate avea orice număr de parametri (cel puțin unul), iar parametrii pot fi o combinație arbitrară a tipurilor întregi, short, long, caractere, siruri, numere în virgulă mobilă de diferite lungimi și alte tipuri. A încerca să invoci *printf* ca procedură cu apel la distanță ar fi practic imposibil, deoarece C-ul este prea permisiv. Totuși, o regulă care să spună că RPC-ul poate fi folosit cu condiția să nu programezi în C (sau C++) nu ar fi prea populară.

O a patra problemă este legată de utilizarea variabilelor globale. În mod normal, procedura de apelare și cea care este apelată pot comunica folosind variabilele globale, în plus față de comunicarea prin parametri. Dacă procedura apelată este mutată acum pe o mașină la distanță, codul va da erori deoarece variabilele globale nu mai sunt partajate.

Aceste probleme nu sunt menite să sugereze că RPC-ul este lipsit de șanse. De fapt, este larg folosit, dar sunt necesare anumite restricții pentru a-l face să funcționeze bine în practică.

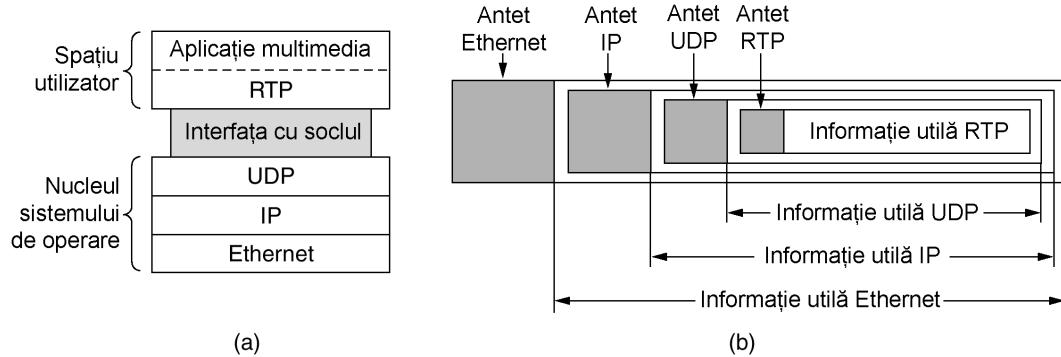
Desigur, RPC-ul nu are nevoie să folosească pachete UDP, dar RPC și UDP se potrivesc bine, și UDP este ușual folosit pentru RPC. Totuși, când parametrii sau rezultatele pot să fie mai mari decât pachetul maxim UDP sau atunci când operația cerută nu este idempotentă (adică nu poate fi repetată în siguranță, ca de exemplu atunci când se incrementează un contor), poate fi necesară stabilirea unei conexiuni TCP și trimiterea cererii prin aceasta, în loc să se folosească UDP-ul.

#### 6.4.3 Protocolul de transport în timp real – Real-Time Transport Protocol

RPC-ul client-server este un domeniu în care UDP este mult folosit. Un alt domeniu este acela al aplicațiilor multimedia în timp real. În particular, având în vedere că radioul pe internet, telefonia pe Internet, muzica la cerere, video-conferințele, video la cerere și alte aplicații multimedia au devenit mai răspândite, oamenii au descoperit că fiecare aplicație a folosit, mai mult sau mai puțin, același protocol de transport în timp real. Treptat a devenit clar faptul că un protocol generic de transport în timp real, pentru aplicații multimedia, ar fi o idee bună. Așa a luat naștere RTP-ul (**Real-time Transport Protocol**, rom: Protocol de transport în timp real). Este descris în RFC 1889 și acum este folosit pe scară largă.

Pozitia RTP-ului în stiva de protocoale este oarecum ciudată. S-a hotărât să se pună RTP-ul în spațiul utilizator și să se ruleze (în mod normal) peste UDP. El funcționează după cum urmează. Aplicațiile multimedia constau în aplicații audio, video, text și posibil alte fluxuri. Acestea sunt trimise bibliotecii RTP, care se află în spațiul utilizator împreună cu aplicația. Apoi, această bibliotecă multiplează fluxurile și le codează în pachete RTP, pe care apoi le trimite printr-un soclu. La celălalt capăt al soclului (în nucleul sistemului de operare), pachete UDP sunt generate și încapsulate în pachete IP. Dacă computer-ul se găsește într-o rețea Ethernet, pachetele IP sunt puse apoi în cadre

Ethernet, pentru transmisie. Stiva de protocole pentru această situație este prezentată în fig. 6-25 (a). Încapsularea pachetului este prezentată în fig. 6-25 (b).



**Fig. 6-25.** (a) Poziționarea Protocolului RTP în stiva de protocole. (b) Încapsularea pachetului.

Ca o consecință a acestei proiectări, este cam dificil de spus în ce nivel este RTP-ul. Cum rulează în spațiul utilizator și este legat la programul aplicație, în mod cert arată ca un protocol de aplicație. Pe de altă parte, este un protocol generic independent de aplicație, care doar furnizează facilități de transport, astfel încât arată totodată ca un protocol de transport. Probabil că cea mai potrivită descriere este aceea că este un protocol de transport care este implementat la nivelul aplicație.

Funcția de bază a RTP-ului este multiplexarea mai multor fluxuri de date în timp real într-un singur flux de pachete UDP. Fluxul UDP poate fi transmis către o singură destinație (unicasting) sau către destinații multiple (multicasting). Deoarece RTP-ul folosește numai UDP normal, pachetele sale nu sunt tratate în mod special de către rutere, decât dacă sunt activate anumite facilități de calitate a serviciilor din IP. În particular, nu există garanții speciale referitoare la livrare, bruijaj etc.

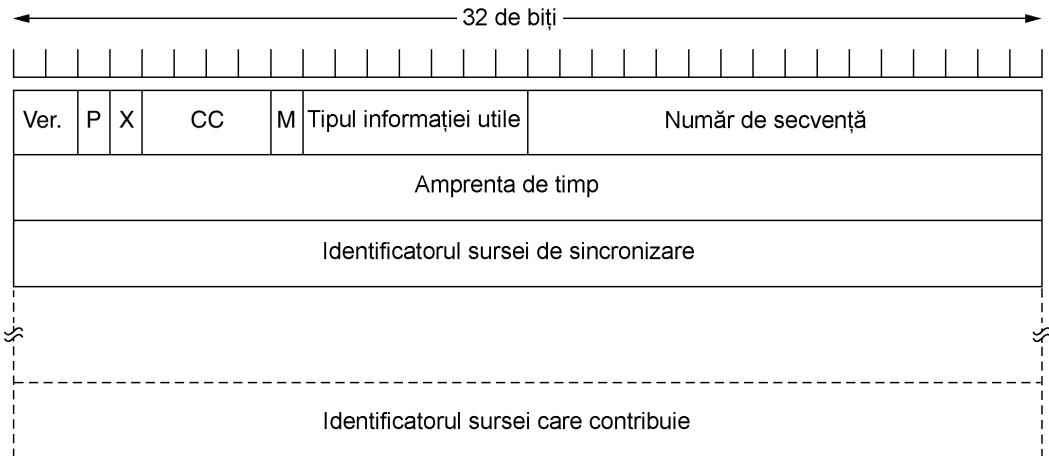
Fiecarui pachet trimis în fluxul RTP i se dă un număr cu unu mai mare decât al predecesorului său. Această numerotare permite destinației să stabilească dacă lipsesc unele pachete. Dacă un pachet lipsește, cea mai bună decizie ce poate fi luată de către destinație este de a approxima valoarea lipsă prin interpolare. Retransmiterea nu este o opțiune practică având în vedere că pachetul retransmis va ajunge probabil prea târziu pentru a fi util. Ca o consecință, RTP-ul nu are control al fluxului, control al erorii, nu are confirmări și nu are mecanism pentru a cere retransmiterea.

Fiecare informație utilă din RTP poate să conțină mostre multiple și ele pot fi codate în orice mod dorește aplicația. Pentru a permite compatibilitatea, RTP-ul definește mai multe profiluri (de exemplu un singur flux audio) și pentru fiecare profil pot fi permise multiple formate de codare. De exemplu, un singur flux audio poate fi codat ca mostre de 8 biți PCM la 8 KHz, codare delta, codare previzibilă, codare GSM, MP3 și așa mai departe. RTP-ul furnizează un câmp antet în care sursa poate specifica codarea, dar altfel nu este implicat în modul în care este făcută codarea.

O altă facilitate de care au nevoie multe aplicații multimedia este stabilirea amprentei de timp. Aici ideea este de a permite sursei să asocieze o amprentă de timp cu prima moștră din fiecare pachet. Amprente de timp sunt relative la începutul fluxului, așa că numai diferențele dintre acestea sunt semnificative. Valorile absolute nu au nici o semnificație. Acest mecanism permite destinației să folosească zone tampon de dimensiuni mici și să reproducă fiecare eșantion la numărul corect de milisecunde după începutul fluxului, independent de momentul în care ajunge pachetul ce conține eșantionul. Stabilirea amprentelor de timp nu numai că reduce efectele bruijajului, ci permite de asemenea mai multor fluxuri să se sincronizeze între ele. De exemplu, un program de televiziune

digital poate avea un flux video și două fluxuri audio. Cele două fluxuri audio pot fi pentru emisiuni stereo sau pentru a permite filmelor să fie manipulate cu o coloană sonoră în limba originală și o coloană sonoră dublată în limba locală, oferind o alegere celui care vede. Fiecare flux provine dintr-un dispozitiv fizic diferit, dar dacă sunt stabilite amprente de timp de către un singur contor, ele pot fi redate sincronizat, chiar dacă fluxurile sunt transmise într-un mod dezordonat.

Antetul RTP este ilustrat în fig. 6-26. Acesta constă din trei cuvinte de 32 biți și eventual unele extensii. Primul cuvânt conține câmpul *Versiune*, care este deja la 2. Să sperăm că această versiune este foarte asemănătoare cu ultima versiune deoarece a mai rămas aici doar un punct de cod (deși 3 ar putea fi definit ca semnificând faptul că versiunea reală se găsește într-un cuvânt extins).



**Fig. 6-26.** Antetul RTP-ului.

Bitul *P* indică faptul că pachetul a fost extins la un multiplu de 4 octeți. Ultimul octet extins ne spune câți octeți au fost adăugați. Bitul *X* indică prezența unui antet extins. Formatul și semnificația antetului extins nu sunt definite. Singurul lucru care este definit este acela că primul cuvânt al extensiei dă lungimea. Aceasta este o cale de scăpare pentru orice cerințe neprevăzute.

Câmpul *CC* arată câte surse contribuibile sunt prezente, de la 0 la 15 (vezi mai jos). Bitul *M* este un bit de marcă specific aplicației. Poate fi folosit pentru a marca începutul unui cadru video, începutul unui cuvânt într-un canal audio sau altceva ce aplicația înțelege. Câmpul *Tip informație utilă* indică ce algoritm de codare a fost folosit (de exemplu 8 biți audio necompresati, MP3, etc). Din moment ce fiecare pachet transportă acest câmp, codarea se poate schimba în timpul transmisiei. *Numărul de secvență* este doar un contor care este incrementat pe fiecare pachet RTP trimis. Este folosit pentru a detecta pachetele pierdute.

Amprenta de timp este stabilită de către sursa fluxului pentru a se ști când este făcut primul eșantion din pachet. Această valoare poate să ajute la reducerea bruiajului la receptor prin separarea redării de momentul ajungerii pachetului. *Identificatorul sursei de sincronizare* spune căruia flux îi aparține pachetul. Este metoda utilizată pentru a multiplexa și demultiplexa mai multe fluxuri de date într-un singur flux de pachete UDP. În sfârșit, dacă există, *identificatorii sursei care contribuie* sunt folosiți când în studio există mixere. În acest caz, mixerul este sursa de sincronizare și fluxurile, fiind amestecate, apar aici.

RTP are un protocol înrudit numit **RTCP (Real Time Transport Control Protocol)**, rom: Protocol de control al transportului în timp real). Acesta se ocupă de răspuns, sincronizare și de interfață cu utilizatorul, dar nu transportă date. Prima funcție poate fi folosită pentru a oferi surselor reacție (eng.: feedback) la întârzieri, bruijaj, lățime de bandă, congestie și alte proprietăți ale rețelei. Această informație poate fi folosită de către procesul de codare pentru a crește rata de transfer a datelor (și să ofere o calitate mai bună) când rețeaua merge bine și să reducă rata de transfer când apar probleme pe rețea. Prin furnizarea continuă de răspunsuri, algoritmii de codare pot fi în continuu adaptăți pentru a oferi cea mai bună calitate posibilă în circumstanțele curente. De exemplu, dacă lățimea de bandă crește sau scade în timpul transmisiei, codarea poate să se schimbe de la MP3 la PCM pe 8 biți la codare delta, aşa cum se cere. Câmpul *Tip informație utilă* este folosit pentru a spune destinației ce algoritm de codare este folosit pentru pachetul curent, făcând posibilă schimbarea acestuia la cerere.

De asemenea, RTCP-ul se ocupă de sincronizarea între fluxuri. Problema este că fluxuri diferite pot folosi ceasuri diferite, cu granularități diferite și devieri de flux diferite. RTCP poate fi folosit pentru a le menține sincronizate.

În sfârșit, RTCP oferă un mod pentru a numi diversele surse (de exemplu în text ASCII). Această informație poate fi afișată pe ecranul receptorului pentru a indica cine vorbește în acel moment.

Mai multe informații despre RTP pot fi găsite în (Perkins, 2002).

## 6.5. PROTOCOALE DE TRANSPORT PRIN INTERNET: TCP

UDP-ul este un protocol simplu și are anumite nișe de utilizare, cum ar fi interacțiunile client-server și cele multimedia, dar pentru cele mai multe aplicații de Internet este necesar un transport de încredere, secvențial al informației. UDP-ul nu poate oferi acest lucru, deci este nevoie de un alt protocol. Acesta este TCP și este pionul principal de lucru al Internet-ului. Să-l studiem acum în amănunt.

### 6.5.1 Introducere în TCP

**TCP (Transport Communication Protocol** - protocol de comunicatie de nivel transport) a fost proiectat explicit pentru a asigura un flux sigur de octeți de la un capăt la celălalt al conexiunii într-o inter-rețea nesigură. O inter-rețea diferă de o rețea propriu-zisă prin faptul că diferite părți ale sale pot dифeири substantiјal în topologie, lărgime de bandă, întârzieri, dimensiunea pachetelor și alți parametri. TCP a fost proiectat să se adapteze în mod dinamic la proprietățile inter-rețelei și să fie robust în ceea ce privește mai multe tipuri de defecte.

TCP a fost definit în mod oficial în RFC 793. O dată cu trecerea timpului, au fost detectate diverse erori și inconsistențe și au fost modificate cerințele în anumite subdomenii. Aceste clarificări, precum și corectarea câtorva erori sunt detaliate în RFC 1122. Extensiile sunt furnizate în RFC 1323.

Fiecare mașină care suportă TCP dispune de o entitate de transport TCP, fie ca proces utilizator, fie ca procedură de bibliotecă, fie ca parte a nucleului. În toate aceste cazuri, ea care gestionează fluxurile TCP și interfețele către nivelul IP. O entitate TCP acceptă fluxuri de date utilizator de la procesele locale, le împarte în fragmente care nu depășesc 64K octeți (de regulă în fragmente de aproximativ 1460 de octeți, pentru a încăpea într-un singur cadru Ethernet împreună cu antetele

TCP și IP) și expediază fiecare fragment ca o datagramă IP separată. Atunci când datagramele IP conținând informație TCP sosesc la o mașină, ele sunt furnizate entității TCP, care reconstruiește fluxul original de octetii. Pentru simplificare, vom folosi cîteodată doar TCP, subînțelegând prin aceasta sau entitatea TCP de transport (o porțiune de program) sau protocolul TCP (un set de reguli). Din context va fi clar care din cele două notiuni este referită. De exemplu, în „Utilizatorul furnizează date TCP-ului” este clară referirea la entitatea TCP de transport.

Nivelul IP nu oferă nici o garanție că datagramele vor fi livrate corect, astfel că este sarcina TCP-ului să detecteze eroarea și să efectueze o retransmisie atunci când situația o impune. Datagramele care ajung (totuși) la destinație pot sosi într-o ordine eronată; este, de asemenea, sarcina TCP-ului să le reassembleze în mesaje respectând ordinea corectă (de secvență). Pe scurt, TCP-ul trebuie să ofere fiabilitatea pe care cei mai mulți utilizatori o doresc și pe care IP-ul nu o oferă.

### 6.5.2 Modelul serviciului TCP

Serviciul TCP este obținut prin crearea atât de către emițător, cât și de către receptor, a unor puncte finale, numite socluri (sockets), așa cum s-a discutat în Sec. 6.1.3. Fiecare soclu are un număr de soclu (adresă) format din adresa IP a mașinii gazdă și un număr de 16 biți, local gazdei respective, numit **port**. Port este numele TCP pentru un TSAP. Pentru a obține o conexiune TCP, trebuie stabilită explicit o conexiune între un soclu de pe mașina emițătoare și un soclu de pe mașina receptoare. Apelurile de soclu sunt prezentate în fig. 6-5.

Un soclu poate fi folosit la un moment dat pentru mai multe conexiuni. Altfel spus, două sau mai multe conexiuni se pot termina la același soclu. Conexiunile sunt identificate prin identificatorii soclurilor de la ambele capete, adică (*soclu 1, soclu 2*). Nu este folosit nici un alt număr sau identificator de circuit virtual.

Numerele de port mai mici decât 256 se numesc **porturi general cunoscute** și sunt rezervate serviciilor standard. De exemplu, orice proces care dorește să stabilească o conexiune cu o mașină gazdă pentru a transfera un fișier utilizând FTP, se poate conecta la portul 21 al mașinii destinație pentru a contacta demonul său FTP. Similar, portul 23 este folosit pentru a stabili o sesiune de lucru la distanță utilizând TELNET. Lista porturilor general cunoscute se găsește la [www.iana.org](http://www.iana.org). Cîteva dintre cele foarte cunoscute sunt prezentate în fig. 6-27.

Port	Protocol	Utilitate
21	FTP	Transfer de fișiere
23	Telnet	Login la distanță
25	SMTP	E-mail
69	TFTP	Protocol de transfer de fișiere trivial
79	Finger	Căutare de informații despre un utilizator
80	HTTP	World Wide Web
110	POP-3	Acces prin e-mail la distanță
119	NNTP	Știri USENET

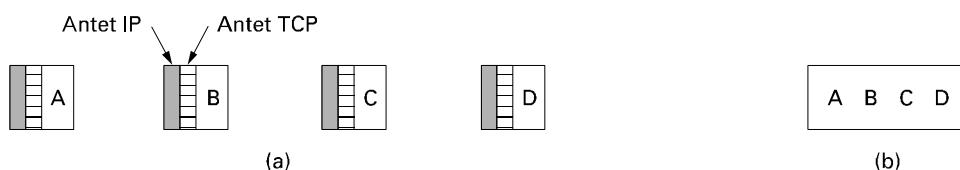
Fig. 6-27. Cîteva porturi asignate.

Cu siguranță ar fi posibil ca, în momentul încărcării, demonul de FTP să se autoatașeze la portul 21, demonul telnet la portul 23 și tot așa. Totuși, dacă s-ar proceda astfel s-ar umple memoria cu demoni inactivi în majoritatea timpului. În schimb, în general se folosește un singur demon, numit **inetd** (**Internet daemon**, rom: demon de Internet) în UNIX, care să se autoatașeze la mai multe porturi și să aștepte prima conexiune care vine. Când acest lucru se întâmplă, inetd creează un nou pro-

ces și execută în el demonul adecvat, lăsând acel demon să se ocupe de cerere. Astfel, demonii, în afară de inetd, sunt activi doar când au de lucru. Inetd află ce porturi să folosească dintr-un fișier de configurare. În consecință, administratorul de sistem poate seta sistemul să aibă demoni permanenti pe cele mai ocupate porturi (de exemplu portul 80) și inetd pe restul.

Toate conexiunile TCP sunt duplex integral și punct-la-punct. Duplex integral înseamnă că traficul se poate desfășura în ambele sensuri în același timp. Punct-la-punct indică faptul că fiecare conexiune are exact două puncte finale. TCP nu suportă difuzarea parțială sau totală.

O conexiune TCP este un flux de octeți și nu un flux de mesaje. Dimensiunile mesajelor nu se conservă de la un capăt la celălalt. De exemplu, dacă procesul emițător execută patru scrieri de câte 512 octeți pe un canal TCP, aceste date pot fi livrate procesului receptor ca patru fragmente (chunks) de 512 octeți, două fragmente de 1024 octeți, un singur fragment de 2048 octeți (vezi fig. 6-28) sau în orice alt mod. Nu există posibilitatea ca receptorul să determine numărul de unități în care a fost scrisă informația.



**Fig. 6-28.** (a) Patru segmente de 512 octeți au fost trimise ca datagrame IP separate.

(b) Livrarea celor 2048 octeți către aplicație, printr-un singur apel *read*.

În UNIX, aceeași proprietate o au și fișierele. Cititorul unui fișier nu poate spune dacă fișierul a fost scris bloc cu bloc, octet cu octet sau tot dintr-o dată. Ca și un fișier UNIX, programele TCP nu au nici cea mai vagă idee despre semnificația octetilor și nici cel mai mic interes pentru a afla acest lucru. Un octet este pur și simplu un octet.

Atunci când o aplicație trimită date către TCP, TCP-ul le poate expedia imediat sau le poate reține într-un tampon (în scopul colectării unei cantități mai mari de informație pe care să o expedieze toată odată), după bunul său plac. Cu toate acestea, câteodată, aplicația dorește ca informația să fie expediată imediat. De exemplu, să presupunem că un utilizator este conectat la o mașină de la distanță. După ce a fost terminată o linie de comandă și s-a tastat Return, este esențial ca linia să fie imediat expediată către mașina de la distanță și să nu fie memorată până la terminarea următoarei linii. Pentru a forța expedierea, aplicația poate folosi indicatorul PUSH, care îi semnalează TCP-ului să nu întârzie procesul de transmisie.

Unele din primele aplicații foloseau indicatorul PUSH ca un fel de marcat pentru a delimita marginile mesajelor. Deși acest truc funcționează câteodată, uneori el eşuează datorită faptului că, la recepție, nu toate implementările TCP-ului transmit aplicației indicatorul PUSH. Mai mult decât atât, dacă mai multe indicațioare PUSH apar înainte ca primul să fi fost transmis (de exemplu, pentru că linia de legătură este ocupată), TCP-ul este liber să colecteze toată informația referită de către aceste indicațioare într-o singură datagramă IP, fără să includă nici un separator între diferitele sale părți.

O ultimă caracteristică a serviciului TCP care merită menționată aici constă în **informația urgență**. Atunci când un utilizator apasă tasta DEL sau CTRL-C pentru a întrerupe o prelucrare la distanță, aflată deja în execuție, aplicația emițător plasează o informație de control în fluxul de date și o furnizează TCP-ului împreună cu indicatorul URGENT. Acest eveniment impune TCP-ului întreprerea acumulării de informație și transmisia imediată a întregii informații disponibile deja pentru conexiunea respectivă.

Atunci când informația urgentă este recepționată la destinație, aplicația receptoare este întreruptă (de ex. prin emisia unui semnal, în terminologie UNIX), astfel încât, eliberată de orice altă activitate, aplicația să poată citi fluxul de date și să poată regăsi informația urgentă. Sfârșitul informației urgente este marcat, astfel încât aplicația să știe când se termină informația. Începutul informației urgente nu este marcat. Este sarcina aplicației să determine acest început. Această schemă furnizează de fapt un rudiment de mecanism de semnalizare, orice alte detalii fiind lăsate la latitudinea aplicației.

### 6.5.3 Protocolul TCP

În această secțiune vom prezenta un punct de vedere general asupra protocolului TCP, pentru a ne concentra apoi, în secțiunea care îi urmează, asupra antetului protocolului, câmp cu câmp.

O caracteristică importantă a TCP, care domină structura protocolului, este aceea că fiecare octet al unei conexiuni TCP are propriul său număr de secvență, reprezentat pe 32 biți. Când a luat ființă Internetul, liniile dintre rutere erau în cel mai bun caz linii încăricate de 56 Kbps, deci unei gazde funcționând la viteza maximă îi lua mai mult de o săptămână să utilizeze toate numerele de secvență. La vitezele rețelelor moderne, numerele de secvență pot fi consumate într-un ritm alarmant, după cum vom vedea mai târziu. Numerele de secvență sunt utilizate atât pentru confirmări cât și pentru mecanismul de secvențiere, acesta din urmă utilizând câmpuri separate de 32 de biți din antet.

Entitățile TCP de transmisie și de receptie interschimbă informație sub formă de segmente. Un **segment TCP** constă dintr-un antet de exact 20 de octeți (plus o parte optională) urmat de zero sau mai mulți octeți de date. Programul TCP este cel care decide cât de mari trebuie să fie aceste segmente. El poate acumula informație provenită din mai multe scrieri într-un singur segment sau poate fragmenta informația provenind dintr-o singură scriere în mai multe segmente. Există două limite care restricționează dimensiunea unui segment. În primul rând, fiecare segment, inclusiv antetul TCP, trebuie să încapă în cei 65.535 de octeți de informație utilă IP. În al doilea rând, fiecare rețea are o **unitate maximă de transfer** sau **MTU (Maximum Transfer Unit)**, deci fiecare segment trebuie să încapă în acest MTU. În realitate, MTU este în general de 1500 octeți (dimensiunea informației utile din Ethernet), definind astfel o limită superioară a dimensiunii unui segment.

Protocolul de bază utilizat de către entitățile TCP este protocolul cu fereastră glisantă. Atunci când un emițător transmite un segment, el pornește un cronometru. Atunci când un segment ajunge la destinație, entitatea TCP receptoare trimite înapoi un segment (cu informație utilă, dacă aceasta există sau fără, în caz contrar) care conține totodată și numărul de secvență următor pe care aceasta se așteaptă să-l recepționeze. Dacă cronometrul emițătorului depășește o anumită valoare înaintea primirii confirmării, emițătorul retransmite segmentul neconfirmat.

Deși acest protocol pare simplu, pot apărea multe situații particulare pe care le vom prezenta mai jos. Segmentele pot ajunge într-o ordine arbitrară, deci octeții 3072-4095 pot fi receptionați, dar nu pot fi confirmăți datorită absenței octetilor 2048-3071. Segmentele pot de asemenea întârzia pe drum un interval de timp suficient de mare pentru ca emițătorul să detecteze o depășire a cronometrului și să le retransmită. Retransmisiiile pot include porțiuni de mesaj fragmentate altfel decât în transmisia inițială, ceea ce impune o tratare atentă, astfel încât să se țină evidența octetilor primiți corect. Totuși, deoarece fiecare octet din flux are un deplasament unic față de începutul mesajului, acest lucru se poate realiza.

TCP trebuie să fie pregătit să facă față unor astfel de situații și să le rezolve într-o manieră eficientă. Un efort considerabil a fost dedicat optimizării performanțelor fluxurilor TCP, ținându-se cont inclusiv de probleme legate de rețea. În continuare vor fi prezentate un număr de algoritmi utilizati de numeroase implementări TCP.

### 6.5.4 Antetul segmentului TCP

În fig. 6-29 este prezentată structura unui segment TCP. Fiecare segment începe cu un antet format dintr-o structură fixă de 20 de octeți. Antetul fix poate fi urmat de un set de opțiuni asociate antetului. În continuarea opțiunilor, dacă ele există, pot urma până la  $65.535 - 20 - 20 = 65.495$  de octeți de date, unde primul 20 reprezintă antetul IP, iar al doilea antetul TCP. Segmente care nu conțin octeți de date sunt nu numai permise, dar și utilizate în mod frecvent pentru confirmări și mesaje de control.

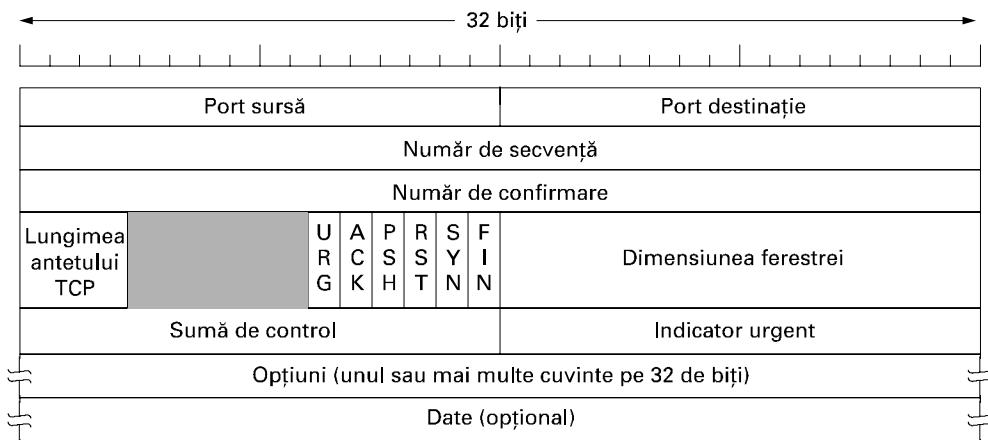


Fig. 6-29. Antetul TCP.

Să disecăm acum structura antetului TCP, câmp cu câmp. Câmpurile *Port sursă* și *Port destinație* identifică punctele finale ale conexiunii. Porturile general cunoscute sunt definite la [www.iana.org](http://www.iana.org), dar fiecare gazdă le poate aloca pe celelalte după cum dorește. Un port formează împreună cu adresa IP a mașinii sale un unic punct de capăt (eng.: end point) de 48 de biți. Conexiunea este identificată de punctele de capăt ale sursei și destinației.

Câmpurile *Număr de secvență* și *Număr de confirmare* au semnificația funcțiilor lor uzuale. Trebuie observat că cel din urmă indică octetul următor așteptat și nu ultimul octet recepționat în mod corect. Ambele câmpuri au lungimea de 32 de biți, deoarece într-un flux TCP fiecare bit de informație este numerotat.

*Lungimea antetului TCP* indică numărul de cuvinte de 32 de biți care sunt conținute în antetul TCP. Această informație este utilă, deoarece câmpul *Optiuni* este de lungime variabilă, proprietate pe care o transmite astfel și antetului. Tehnic vorbind, acest câmp indică în realitate începutul datelor din segment, măsurat în cuvinte de 32 de biți, dar cum acest număr este identic cu lungimea antetului în cuvinte, efectul este același.

Urmează un câmp de șase biți care este neutilizat. Faptul că acest câmp a supraviețuit intact mai mult de un sfert de secol este o mărturie despre cât de bine a fost proiectat TCP-ul. Protocole mai prost concepute ar fi avut nevoie de el pentru a corecta erori ale proiectării inițiale.

Urmează acum șase indicatori de câte un bit. URG este poziționat pe 1 dacă Indicatorul Urgent este valid. Indicatorul Urgent este folosit pentru a indica deplasamentul în octeți față de numărul curent de secvență la care se găsește informația urgentă. O astfel de facilitate ține locul mesajelor de

întrerupere. Așa cum am menționat deja anterior, această facilitate reprezintă esența modului în care emițătorul poate transmite un semnal receptorului fără ca TCP-ul în sine să fie cauza îintruperii.

Bitul *ACK* este poziționat pe 1 pentru a indica faptul că *Numărul de confirmare* este valid. În cazul în care *ACK* este poziționat pe 0, segmentul în discuție nu conține o confirmare și câmpul *Număr de confirmare* este ignorat.

Bitul *PSH* indică informația FORTATĂ. Receptorul este rugat respectuos să livreze aplicației informația respectivă imediat ce este recepționată și să nu o memoreze în așteptarea umplerii tam-poanelor de comunicație (lucru care, altminteri, ar fi făcut din rațiuni de eficiență).

Bitul *RST* este folosit pentru a desființa o conexiune care a devenit inutilizabilă datorită defectiunii unei mașini sau oricărui alt motiv. El este de asemenea utilizat pentru a refuza un segment invalid sau o încercare de deschidere a unei conexiuni. În general, recepționarea unui segment având acest bit poziționat indică o problemă care trebuie tratată în funcție de context.

Bitul *SYN* este utilizat pentru stabilirea unei conexiuni. Cererea de conexiune conține *SYN* = 1 și *ACK* = 0 pentru a indica faptul că acel câmp suplimentar de confirmare nu este utilizat. Răspunsul la o astfel de cerere conține o confirmare, având deci *SYN* = 1 și *ACK* = 1. În esență, bitul *SYN* este utilizat pentru a indica o CERERE DE CONEXIUNE și o CONEXIUNE ACCEPTATĂ, bitul *ACK* făcând distincția între cele două posibilități.

Bitul *FIN* este folosit pentru a încheia o conexiune. El indică faptul că emițătorul nu mai are nici o informație de transmis. Cu toate acestea, după închiderea conexiunii, un proces poate receptiona în continuare date pe o durată nedefinită. Ambele segmente, *SYN* și *FIN*, conțin numere de secvență și astfel este garantat faptul că ele vor fi prelucrate în ordinea corectă.

În TCP, fluxul de control este tratat prin ferestre glisante de dimensiune variabilă. Câmpul *Fereastră* indică numărul de octeți care pot fi trimiși, începând de la octetul confirmat. Un câmp *Fereastră* de valoare 0 este perfect legal și spune că octetii până la *Număr de confirmare* - 1 inclusiv au fost receptionați, dar receptorul dorește cu ardoare o pauză, așa că mulțumește frumos, dar pentru moment nu dorește continuarea transferului. Permisunea de expediere poate fi acordată ulterior de către receptor prin trimiterea unui segment având același *Număr de confirmare*, dar un câmp *Fereastră* cu o valoare nenulă.

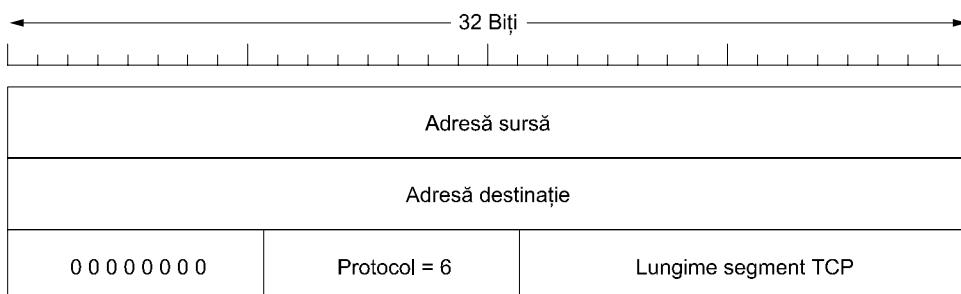
În protocolele din cap. 3, confirmările pentru cadrele primite și permisiunea de a trimite noi cadre erau legate una de alta. Aceasta era o consecință a dimensiunii fixe a ferestrei pentru fiecare protocol. În TCP, confirmările și permisiunea de a trimite noi date sunt total decuplate. De fapt, receptorul poate spune: Am primit octetii până la al *k*-lea, dar în acest moment nu mai doresc să primesc alții. Această decuplare (care de fapt reprezintă o fereastră de dimensiune variabilă) oferă mai multă flexibilitate. O vom studia detaliat mai jos.

Este de asemenea prevăzută o *Sumă de control*, în scopul obținerii unei fiabilități extreme. Această sumă de control este calculată pentru antet, informație și pseudo-antetul conceptual prezentat în fig. 6-30. În momentul calculului, *Suma de control* TCP este poziționată pe zero, iar câmpul de date este completat cu un octet suplimentar nul, dacă lungimea sa este un număr impar. Algoritmul de calcul al sumei de control este simplu, el adunând toate cuvintele de 16 biți în complement față de 1 și aplicând apoi încă o dată complementul față de 1 asupra sumei. În acest mod, atunci când receptorul aplică același calcul asupra întregului segment, inclusiv asupra *Sumei de control*, rezultatul ar trebui să fie 0.

Pseudo-antetul conține adresele IP ale mașinii sursă și destinație, de 32 de biți fiecare, numărul de protocol pentru TCP (6) și numărul de octeți al segmentului TCP (inclusiv și antetul). Prin includerea pseudo-antetului în calculul sumei de control TCP se pot detecta pachetele care au fost

preluate eronat, dar procedând astfel, este negată însăși ierarhia protocolului, deoarece adresa IP aparține nivelului IP și nu nivelului TCP.

Câmpul *Optiuni* a fost proiectat pentru a permite adăugarea unor facilități suplimentare neacooperite de antetul obișnuit. Cea mai importantă opțiune este aceea care permite fiecărei mașini să specifică încărcarea maximă de informație utilă TCP pe care este dispusă să o accepte. Utilizarea segmentelor de dimensiune mare este mai eficientă decât utilizarea segmentelor de dimensiune mică datorită amortizării antetului de 20 de octeți prin cantitatea mai mare de informație utilă. Cu toate acestea, este posibil ca mașini mai puțin performante să nu fie capabile să manevreze segmente foarte mari. În timpul inițializării conexiunii, fiecare parte anunță dimensiunea maximă acceptată și așteaptă de la partener aceeași informație. Câștigă cel mai mic dintre cele două numere. Dacă o mașină nu folosește această opțiune, cantitatea implicită de informație utilă este de 536 octeți. Toate mașinile din Internet trebuie să accepte segmente de dimensiune  $536 + 20 = 556$  octeți. Dimensiunea maximă a segmentului nu trebuie să fie aceeași în cele două direcții.



**Fig. 6-30.** Pseudo-antetul inclus în suma de control TCP.

O fereastră de 64 K octeți reprezintă adesea o problemă pentru liniile cu o lărgime de bandă mare și/sau cu întârzieri mari. Pe o linie T3 (44.736 Mbps) trimitera unei ferestre întregi de 64 K octeți durează doar 12 ms. Dacă întârzierea propagării dus-întors este de 50 ms (care este valoarea tipică pentru o linie trans-continențală), emițătorul va aștepta confirmări - fiind deci inactiv  $\frac{3}{4}$  din timp. Pe o conexiune prin satelit, situația este chiar mai rea. O fereastră de dimensiune mare ar permite emițătorului să continue trimiterea informației, însă o astfel de dimensiune nu poate fi reprezentată în cei 16 biți ai câmpului Fereastră. În RFC 1323 se propune o opțiune Scală a ferestrei, permitând emițătorului și receptorului să negocieze un factor de scalare a ferestrei. Acest număr permite ambelor părți să deplaseze câmpul Fereastră cu până la 14 biți spre stânga, permitând astfel ferestre de până la 230 octeți. Această opțiune este suportată în prezent de cele mai multe implementări ale TCP-ului.

O altă opțiune propusă de RFC 1106, și care este în prezent implementată pe scară largă, constă în utilizarea unei repetări selective în locul unui protocol cu întoarcere de n pași (eng.: go back n protocol). Dacă receptorul primește un segment eronat urmat de un număr mare de segmente corecte, protocolul TCP clasic va constata într-un final o depășire de timp și va retrimitre toate segmentele neconfirmate, deci și pe acelea care au fost recepționate corect (adică se face o întoarcere de n pași). RFC 1106 introduce NAK-urile pentru a permite receptorului să ceară un anumit segment (sau segmente). După obținerea acestora, el poate confirma toată informația memorată reducând astfel cantitatea de informație retransmisă.

### 6.5.5 Stabilirea conexiunii TCP

În TCP conexiunile sunt stabilite utilizând „întelegerea în trei pași”, discutată în Sec. 6.2.2. Pentru a stabili o conexiune, una din părți - să spunem serverul - așteaptă în mod pasiv o cerere de conexiune prin execuția primitivelor LISTEN și ACCEPT, putând specifica o sursă anume sau nici o sursă în mod particular.

Cealaltă parte - să spunem clientul - execută o primitivă CONNECT, indicând adresa IP și numărul de port la care dorește să se conecteze, dimensiunea maximă a segmentului TCP pe care este dispusă să o accepte și, optional, o informație utilizator (de exemplu o parolă). Primitiva CONNECT trimează un segment TCP având bitul SYN poziționat și bitul ACK nepozitionat, după care așteaptă un răspuns.

Atunci când sosesc la destinație un segment, entitatea TCP receptoare verifică dacă nu cumva există un proces care a executat LISTEN pe numărul de port specificat în câmpul *Port destinație*. În caz contrar, trimite un răspuns cu bitul RST poziționat, pentru a refuza conexiunea.

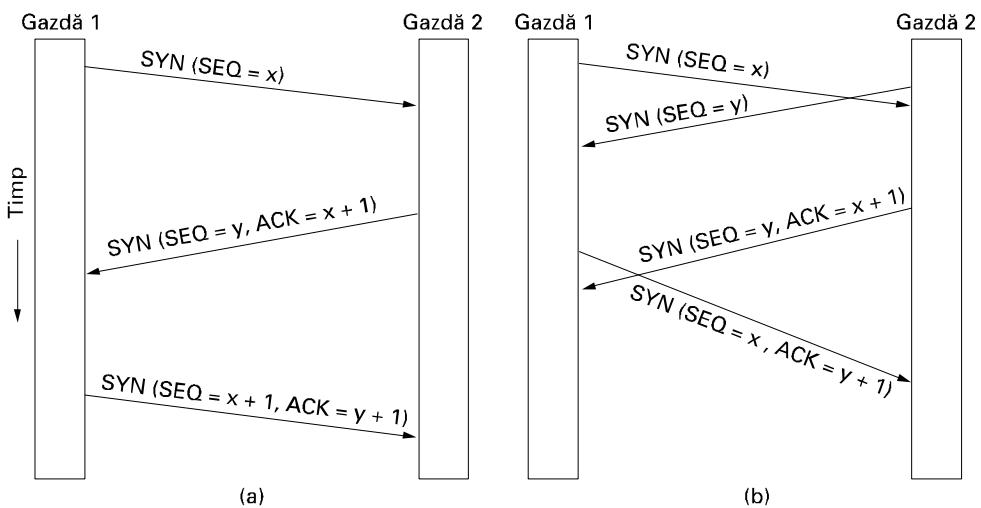


Fig. 6-31. (a) Stabilirea unei conexiuni TCP în cazul normal. (b) Coliziunea apelurilor.

Dacă există vreun proces care ascultă la acel port, segmentul TCP recepționat va fi dirijat către procesul respectiv. Acesta poate accepta sau refuza conexiunea. Dacă o acceptă, trimite înapoi expeditorului un segment de confirmare. În fig. 6-31(a) este reprezentată secvența de segmente TCP transferate în caz de funcționare normală. De notat că un segment SYN consumă un octet din spațiul numerelor de secvență, astfel încât confirmarea să poată fi făcută fără ambiguități.

Secvența de evenimente ilustrată în fig. 6-31(b) reprezintă cazul în care două mașini încearcă simultan să stabilească o conexiune între aceleași două porturi. Rezultă că este suficient să fie stabilită o singură conexiune și nu două, deoarece conexiunile sunt identificate prin punctele lor terminale. Dacă prima inițializare conduce la crearea unei conexiuni identificată prin  $(x, y)$  și același lucru îl face și cea de-a doua inițializare, atunci este construită o singură intrare de tabel, în spățiu pentru  $(x, y)$ .

Numărul inițial de secvență asociat unei conexiuni nu este 0, din motivele discutate anterior. Se utilizează o schemă bazată pe un ceas cu o bătaie la fiecare 4 μs. Pentru mai multă siguranță, atunci când

o mașină se defectează, este posibil ca ea să nu fie reinicializată în timpul de viață maxim al unui pachet, garantându-se astfel că pachetele unei conexiuni anterioare nu se plimbă încă pe undeva prin Internet.

### 6.5.6 Eliberarea conexiunii TCP

Desi conexiunile TCP sunt bidirectionale, pentru a înțelege cum sunt desființate conexiunile, cel mai bine este să ni le imaginăm sub forma unei perechi de legături unidirectionale. Fiecare legătură unidirectională este eliberată independent de perechea sa. Pentru eliberarea unei conexiuni, orice partener poate expedia un segment TCP având bitul *FIN* setat, lucru care indică faptul că nici o informație nu mai urmează să fie transmisă. Atunci când *FIN*-ul este confirmat, sensul respectiv de comunicare este efectiv oprit pentru noi date. Cu toate acestea, informația poate fi transferată în continuare, pentru un timp nedefinit, în celălalt sens. Conexiunea este desființată atunci când ambele direcții au fost opriți. În mod normal, pentru a elibera o conexiune sunt necesare patru segmente TCP: câte un *FIN* și un *ACK* pentru fiecare sens. Cu toate acestea, este posibil ca primul *ACK* și cel de-al doilea *FIN* să fie cuprinse în același segment reducând astfel numărul total la trei.

La fel ca în conversațiile telefonice, în care ambele persoane pot spune „la revedere” și pot închide telefonul simultan, ambele capete ale unei conexiuni TCP pot expedia segmente *FIN* în același timp. Acestea sunt confirmate ca de obicei, conexiunea fiind astfel eliberată. Nu există de fapt nici o diferență esențială între cazurile în care mașinile eliberează conexiunea secvențial respectiv simultan.

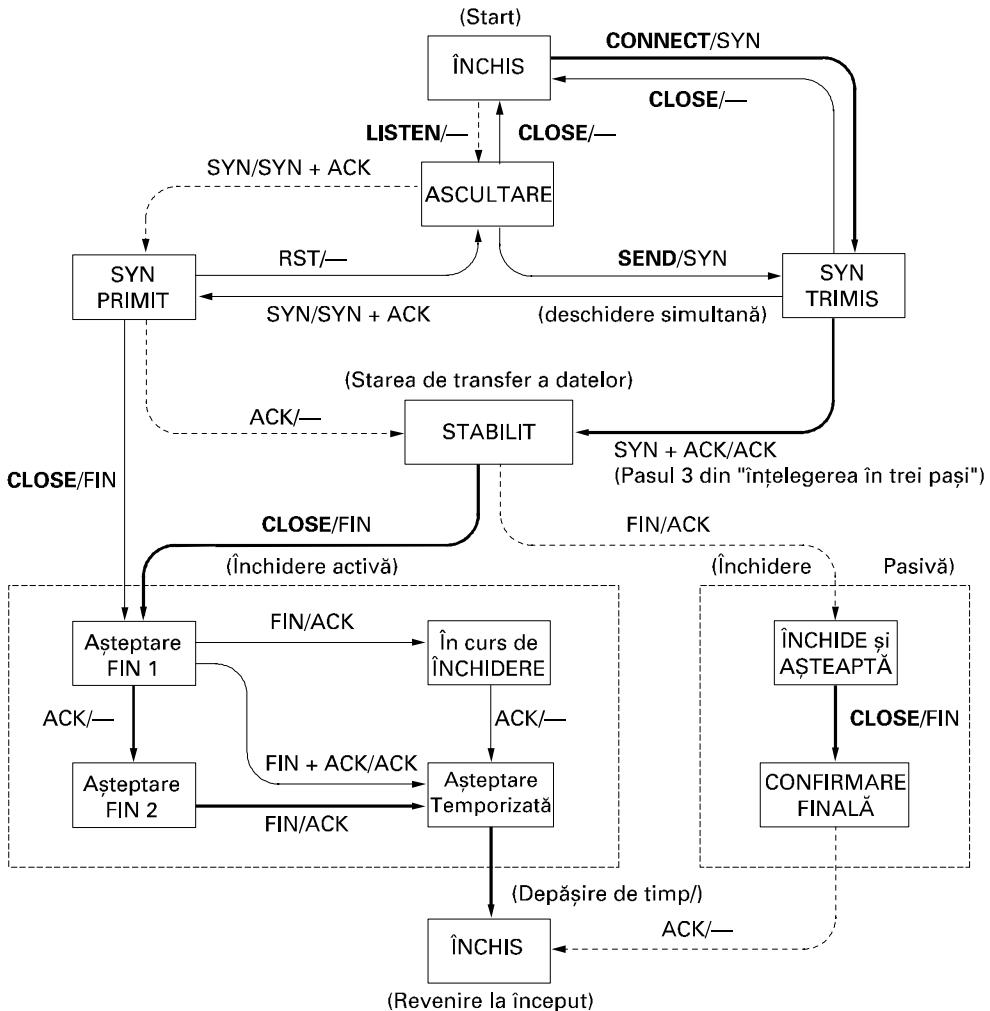
Pentru a evita problema celor două armate, sunt utilizate cronometre. Dacă un răspuns la un *FIN* nu este recepționat pe durata a cel mult două cicluri de maxime de viață ale unui pachet, emițătorul *FIN*-ului eliberează conexiunea. Cealaltă parte va observa în final că nimeni nu mai pare să asculte la celălalt capăt al conexiunii, și va elibera conexiunea în urma expirării unui interval de timp. Această soluție nu este perfectă, dar având în vedere faptul că o soluție perfectă este teoretic imposibilă, va trebui să ne mulțumim cu ce avem. În realitate astfel de probleme apar foarte rar.

### 6.5.7 Modelarea administrării conexiunii TCP

Pașii necesari stabilirii unei conexiuni pot fi reprezentați printr-un automat cu stări finite, cele 11 stări ale acestuia fiind prezentate în fig. 6-32. În fiecare stare pot apărea doar anumite evenimente. Atunci când are loc un astfel de eveniment, este îndeplinită o acțiune specifică. Atunci când se produce un eveniment a cărui apariție nu este legală în starea curentă, este semnalată o eroare.

Stare	Descriere
CLOSED (ÎNCHIS)	Nici o conexiune nu este activă sau în aşteptare
LISTEN (ASCULTARE)	Serverul aşteaptă recepționarea unui apel
SYN RCV (Recepție SYN)	S-a recepționat o cerere de conexiune; aştept ACK
SYN SENT (Transmisie SYN)	Aplicația a început deschiderea unei conexiuni
ESTABLISHED (STABILIT)	Starea normală de transfer a datelor
FIN WAIT 1 (Așteptare FIN 1)	Aplicația a anunțat că termină
FIN WAIT 2 (Așteptare FIN 2)	Partenerul este de acord cu eliberarea conexiunii
TIMED WAIT (Așteptare Temporizată)	Se aşteaptă „moartea” tuturor pachetelor
CLOSING (În curs de ÎNCHIDERE)	Ambele părți încearcă simultan închiderea
CLOSE WAIT (ÎNCHIDE și AŞTEAPTA)	Partenerul a inițiat eliberarea conexiunii
LAST ACK (CONFIRMARE FINALĂ)	Se aşteaptă „moartea” tuturor pachetelor

Fig. 6-32. Stările utilizate în automatul cu stări finite pentru controlul conexiunii TCP.



**Fig. 6-33.** Automatul cu stări finite pentru controlul conexiunii TCP. Linia groasă continuă este calea normală pentru client. Linia groasă întreruptă este calea normală pentru server. Liniile subțiri sunt evenimente neuzuale. Fiecare tranziție este etichetată de evenimentul care a creat-o și acțiunea care rezultă din el, separate de slash.

Fiecare conexiune începe în starea **ÎNCHIS**. Această stare este părăsită dacă urmează să se stabilească o conexiune pasivă (LISTEN) sau activă (CONNECT). Dacă partenerul stabilește o conexiune de tipul opus, starea devine **STABILIT**. Desființarea conexiunii poate fi inițiată de oricare din partener, o dată cu eliberarea conexiunii revenindu-se în starea **ÎNCHIS**.

Automatul cu stări finite este reprezentat în fig. 6-33. Cazul cel mai comun, al unui client conectându-se activ la un server pasiv, este reprezentat prin linii groase - continue pentru client și întrerupe pentru server. Liniile subțiri reprezintă secvențe de evenimente mai puțin obișnuite, dar posibile. Fiecare linie din fig. 6-33 este etichetată cu o pereche *eveniment/acțiune*. Evenimentul poate fi unul inițiat de către utilizator printr-un apel sistem (CONNECT, LISTEN, SEND sau CLOSE), recepțio-

narea unui segment (*SYN*, *FIN*, *ACK* sau *RST*) sau, într-un singur caz, expirarea unui interval de timp egal cu dublul ciclului de viață a unui pachet. Acțiunea constă în expedierea unui segment de control (*SYN*, *FIN* sau *RST*) sau „nici o acțiune”, lucru reprezentat prin —. Comentariile sunt incluse între paranteze.

Diagrama poate fi înțeleasă cel mai bine urmărind de la bun început calea urmată de un client (linia groasă continuă) și apoi calea urmată de un server (linia groasă întreruptă). Atunci când un program aplicație de pe mașina client generează o cerere CONNECT, entitatea TCP locală creează o înregistrare de conexiune, o marchează ca fiind în starea *SYN SENT* și trimit un segment *SYN*. De observat că mai multe conexiuni pot fi deschise (sau în curs de a fi deschise) în același timp spre folosul mai multor aplicații, astfel încât o stare este asociată unei conexiuni și este înregistrată în înregistrarea asociată acesteia. La receptia unui *SYN + ACK*, TCP expediază ultima confirmare (*ACK*) din „înțelegerea în trei pași” și comută în starea *STABILIT*. Din acest moment, informația poate fi atât expediată cât și receptuată.

Atunci când se termină o aplicație, se apelează primitiva CLOSE care impune entității TCP locale expedierea unui segment *FIN* și așteptarea *ACK*-ului corespunzător (dreptunghiul figurat cu linie întreruptă și etichetat „închidere activă”). Atunci când *ACK*-ul este receptuat, se trece în starea *AȘTEPTARE FIN 2*, unul din sensuri fiind în acest moment închis. Atunci când celălalt sens este la rândul său închis de partenerul de conexiune, se receptionează un *FIN* care este totodată și confirmat. În acest moment, ambele sensuri sunt închise, dar TCP-ul așteaptă un interval de timp egal cu dublul duratei de viață a unui pachet, garantând astfel că toate pachetele acestei conexiuni au murit și că nici o confirmare nu a fost pierdută. Odată ce acest interval de timp expiră, TCP-ul șterge înregistrarea asociată conexiunii.

Să examinăm acum gestiunea conexiunii din punctul de vedere al server-ului. Acesta execută LISTEN și se „așează” fiind totodată atent pentru a vedea cine „se ridică în picioare”. La receptuarea unui *SYN*, acesta este confirmat și serverul comută în starea *SYN RCV*D. Atunci când *SYN*-ul server-ului este la rândul său confirmat, „înțelegerea în trei pași” este completă, serverul comutând în starea *STABILIT*. De acum, transferul informației poate începe.

Atunci când clientul a terminat, execută CLOSE, ceea ce conduce la atenționarea server-ului prin receptuarea unui *FIN* (dreptunghiul figurat cu linie întreruptă și etichetat „închidere pasivă”). Atunci când și acesta execută un CLOSE, se trimit un *FIN* către client. O dată cu primirea confirmării clientului, serverul desființează conexiunea și șterge înregistrarea asociată.

### 6.5.8 Politica TCP de transmisie a datelor

Cum s-a menționat anterior, administrarea ferestrei în TCP nu este direct legată de confirmări, așa cum se întâmplă la cele mai multe protocoale de nivel legătură de date. De exemplu, să presupunem că receptorul are un tampon de 4096 octeți, așa cum se vede în fig. 6-34. Dacă emițătorul transmite un segment de 2048 de octeți care este receptuat corect, receptorul va confirma segmentul. Deoarece acum tamponul acestuia din urmă mai are liberi doar 2048 octeți (până când aplicația șterge niște date din acest tampon), receptorul va anunța o fereastră de 2048 octeți începând de la următorul octet așteptat.

Acum, emițătorul transmite alți 2048 octeți, care sunt confirmati, dar fereastra oferită este 0. Emițătorul trebuie să se opreasă până când procesul aplicație de pe mașina receptoare a șters niște date din tampon, moment în care TCP poate oferi o fereastră mai mare.

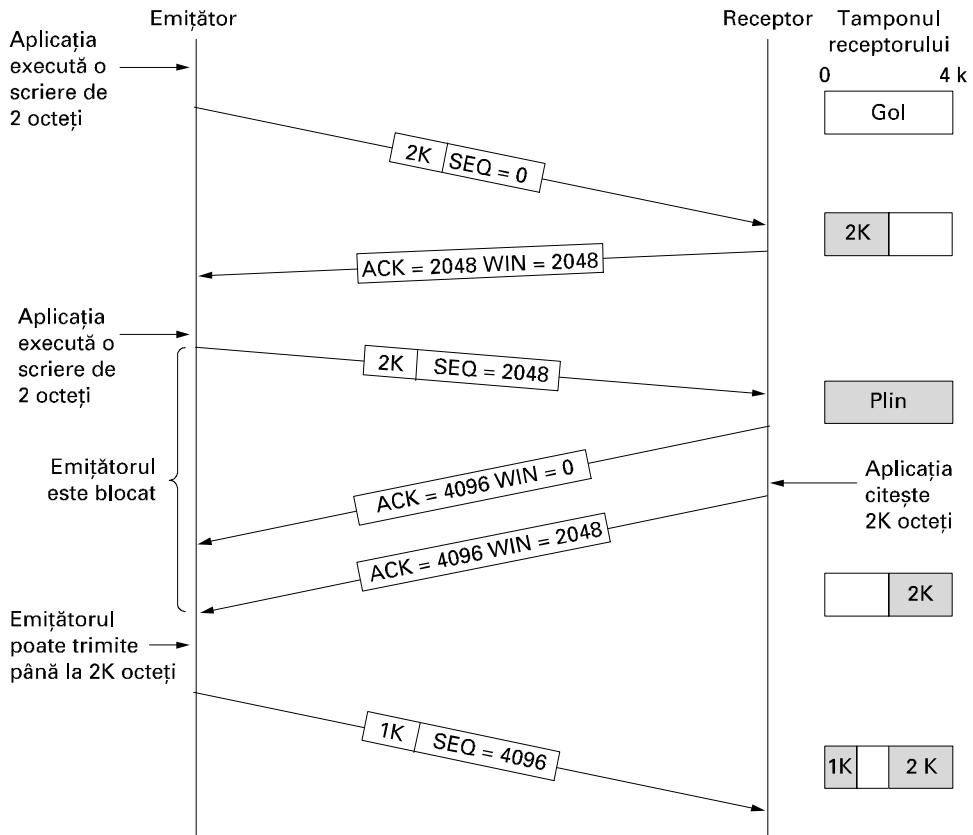


Fig. 6-34. Controlul ferestrei în TCP.

Atunci când fereastra este 0, în mod normal emițătorul nu poate să transmită segmente, cu două excepții. În primul rând, informația urgentă poate fi trimisă, de exemplu pentru a permite utilizatorului să oprească procesele rulând pe mașina de la distanță. În al doilea rând, emițătorul poate trimite un segment de un octet pentru a determina receptorul să renunțe următorul octet așteptat și dimensiunea ferestrei. Standardul TCP prevede în mod explicit această opțiune pentru a preveni interblocarea în cazul în care se întâmplă ca anunțarea unei ferestre să fie vreodată pierdută.

Emițătorii nu transmit în mod obligatoriu date de îndată ce acest lucru este cerut de către aplicație. Nici receptorii nu trimit în mod obligatoriu confirmările de îndată ce acest lucru este posibil. De exemplu, în fig. 6-34, atunci când sunt disponibili primii 2K octeți, TCP, știind că dispune de o fereastră de 4K octeți, va memora informația în tampon până când alți 2K octeți devin disponibili și astfel se va putea transmite un segment cu o încărcare utilă de 4K octeți. Această facilitate poate fi folosită pentru îmbunătățirea performanțelor.

Să considerăm o conexiune TELNET cu un editor interactiv care reacționează la fiecare apăsare a tastelor. În cel mai rău caz, atunci când un caracter sosește la entitatea TCP emițătoare, TCP creează un segment TCP de 21 octeți, pe care îl furnizează IP-ului pentru a fi transmis ca o datagramă IP de 41 octeți. De partea receptorului, TCP transmite imediat o confirmare de 40 octeți (20 octeți antet TCP și 20 octeți antet IP). Mai târziu, când editorul a citit caracterul, TCP transmite o

actualizare a ferestrei, deplasând fereastra cu un octet la dreapta. Acest pachet este de asemenea de 40 octeți. În final, când editorul a prelucrat caracterul, transmite ecoul sub forma unui pachet de 41 octeți. Cu totul, sunt folosiți 162 octeți din lărgimea de bandă și sunt trimise patru segmente pentru orice caracter tipărit. Atunci când lărgimea de bandă este redusă, această metodă de lucru nu este recomandată.

O abordare folosită de multe implementări TCP pentru optimizarea acestei situații constă în întârzierea confirmărilor și actualizările de fereastră timp de 500 ms, în speranța apariției unor informații la care să se atașeze pentru o călătorie pe gratis. Presupunând că editorul are un ecou de 50 ms, este necesar acum un singur pachet de 41 octeți pentru a fi trimis utilizatorului de la distanță, reducând numărul pachetelor și utilizarea lărgimii de bandă la jumătate.

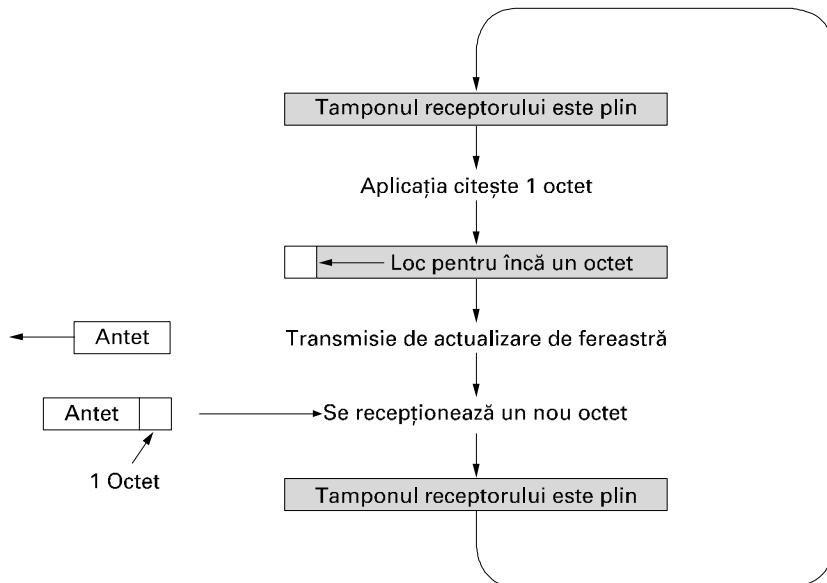
Deși această regulă reduce încărcarea rețelei de către receptor, emițătorul operează încă ineficient trimițând pachete de 41 octeți care conțin un singur octet de date. O modalitate de a reduce această deficiență este cunoscută ca **algoritmul lui Nagle** (Nagle, 1984). Sugestia lui Nagle este simplă: atunci când emițătorul dispune de date, în secvență, de căte un octet, el va trimite doar primul octet, restul octetilor fiind memorati până la confirmarea primului octet. Apoi vor fi trimise toate caracterele memorate într-un segment TCP și va continua memorarea până la confirmarea tuturor octetilor. Dacă utilizatorul tastează repede și rețeaua este lentă, un număr substanțial de caractere poate fi plasat în fiecare segment, reducând cu mult lărgimea de bandă folosită. În plus, algoritmul permite transmisia unui nou pachet, dacă s-a dispus de suficientă informație pentru a umple jumătate dintr-o fereastră sau pentru a completa un segment.

Implementările TCP folosesc pe scară largă algoritmul lui Nagle, dar există situații când este mai bine ca el să fie dezactivat. În particular, când o aplicație X-Windows rulează prin Internet, deplasările mausului trebuie transmise mașinii de la distanță. (Sistemul X Window este sistemul de ferestre utilizat pe majoritatea sistemelor UNIX.) Gruparea lor pentru a fi transmise în rafală provoacă o mișcare imprevizibilă a cursorului, lucru care nemulțumește profund utilizatorii.

O altă problemă care poate degrada performanța TCP este **sindromul ferestrei stupide**. (Clark, 1982). Această problemă apare atunci când informația este furnizată entității TCP emițătoare în blocuri mari, dar la partea receptoare o aplicație interactivă citește datele octet cu octet. Pentru a înțelege problema, să analizăm fig. 6-35. Inițial, tamponul TCP al receptorului este plin și emițătorul știe acest fapt (adică are o fereastră de dimensiune 0). Apoi, aplicația interactivă citește un caracter de pe canalul TCP. Această acțiune face fericită entitatea TCP receptoare, deci ea va trimite o actualizare de fereastră către emițător dându-i astfel dreptul de a mai trimite un octet. Îndatorat, emițătorul trimite un octet. Cu acesta, tamponul este plin și receptorul confirmă segmentul de 1 octet, dar reposiționează dimensiunea ferestrei la 0. Acest comportament poate continua la nesfârșit.

Soluția lui Clark este de a nu permite receptorului să trimită o actualizare de fereastră la fiecare octet. În schimb, el este forțat să aștepte până când spațiul disponibil are o dimensiune decentă, urmând să-l ofere pe acesta din urmă. Mai precis, receptorul nu ar trebui să trimită o actualizare de fereastră până când nu va putea gestiona minimul dintre dimensiunea maximă oferită atunci când conexiunea a fost stabilită și jumătate din dimensiunea tamponului său, dacă este liberă.

Mai mult decât atât, emițătorul poate îmbunătăți situația netrimițând segmente de dimensiune mică. În schimb, el ar trebui să încearcă să aștepte până când acumulează suficient spațiu în fereastră pentru a trimite un segment întreg sau măcar unul conținând cel puțin jumătate din dimensiunea tamponului receptorului (pe care trebuie să o estimateze din secvența actualizărilor de fereastră receptionate până acum).



**Fig. 6-35.** Sindromul ferestrei stupide.

Algoritmul lui Nagle și soluția lui Clark pentru sindromul ferestrei stupide sunt complementare. Nagle a încercat să rezolve problema furnizării datelor către TCP octet cu octet, cauzată de aplicația emițătoare. Clark a încercat să rezolve problema extragerii datelor de la TCP octet cu octet, cauzată de către aplicația receptoare. Ambele soluții sunt valide și pot funcționa împreună. Scopul este ca emițătorul să nu trimită segmente mici, iar receptorul să nu ceară astfel de segmente.

Receptorul TCP poate face, pentru îmbunătățirea performanțelor, mai mult decât simpla actualizare a ferestrei în unități mari. Ca și emițătorul TCP, el are posibilitatea să memoreze date, astfel încât să poată bloca o cerere de READ a aplicației până când îl poate furniza o cantitate semnificativă de informație. Astfel se reduce numărul de apeluri TCP, deci și suprăîncărcarea. Bineînteles, în acest mod va crește și timpul de răspuns, dar pentru aplicații care nu sunt interactive, așa cum este transferul de fișiere, eficiența poate fi mai importantă decât timpul de răspuns la cereri individuale.

O altă problemă de discutat despre receptor se referă la ce trebuie să facă acesta cu segmentele care nu sosesc în ordine. Ele pot fi reținute sau eliminate, după cum dorește receptorul. Bineînteles, confirmările pot fi trimise numai atunci când toată informația până la octetul confirmat a fost recepționată. Dacă receptorul primește segmentele 0, 1, 2, 4, 5, 6 și 7, el poate confirma totul până la ultimul octet din segmentul 2 inclusiv. Atunci când emițătorul constată o depășire de timp, el va retrasmite segmentul 3. Dacă receptorul a memorat în tampon segmentele 4 până la 7, odată cu receptia segmentului 3 el poate confirma toți octetii până la sfârșitul segmentului 7.

### 6.5.9 Controlul congestiei în TCP

Atunci când încărcarea la care este supusă o rețea este mai mare decât poate aceasta să suporte, apare congestia. Internet-ul nu face excepție. În această secțiune, vom discuta algoritmi care se ocupă cu astfel de congestii și care au fost dezvoltăți pe parcursul ultimului sfert de secol. Deși nivelul rețea

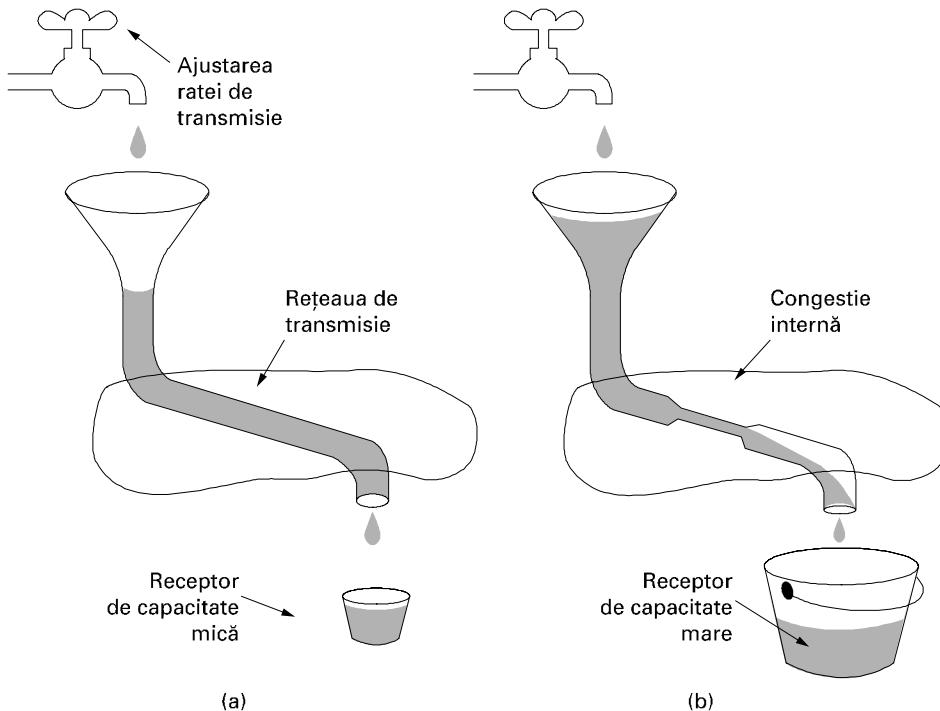
încearcă de asemenea să controleze congestia, cea mai mare parte a muncii este făcută de TCP, și aceasta deoarece adevărată soluție a congestiei constă în micșorarea ratei de transfer a informației.

Theoretic, congestia poate fi controlată pe baza unui principiu împrumutat din fizică: legea conservării pachetelor. Ideea de bază este de a nu injecta un nou pachet în rețea până când un pachet mai vechi nu o părăsește (de exemplu este furnizat receptorului). TCP încearcă să atingă acest scop prin manipularea dinamică a dimensiunii ferestrei.

Primul pas în controlul congestiei este detectia ei. Mai demult, detectia congestiei era dificilă. O depășire de timp datorată pierderii unui pachet putea fi cauzată fie de (1) zgomotul de pe linia de transmisie, fie de (2) eliminarea pachetului de către un ruter congestionat. Diferențierea celor două cazuri era dificilă.

În zilele noastre, pierderea pachetului din pricina erorilor de transmisie este destul de rară, deoarece cele mai multe din trunchiurile principale de comunicație sunt din fibră (deși rețelele fără fir sunt un subiect separat). În consecință, cele mai multe depășiri ale timpilor de transmisie pe Internet se datorează congestiilor. Toți algoritmii TCP din Internet presupun că depășirile de timp sunt cauzate de congestii și monitorizează aceste depășiri pentru a detecta problemele.

Înainte de a discuta despre modalitatea în care TCP reacționează la congestii, să descriem în primul rând modul în care se încearcă prevenirea apariției lor. Atunci când se stabilește o conexiune, trebuie să se aleagă o fereastră de o dimensiune potrivită. Receptorul poate specifica o fereastră bazându-se pe dimensiunea tamponului propriu. Dacă emițătorul acceptă această dimensiune a ferestrei, nu mai pot apărea probleme datorită depășirii tamponului la recepție, dar pot apărea în schimb datorită congestiei interne în rețea.



**Fig. 6-36.** (a) O rețea rapidă alimentând un rezervor de capacitate mică.  
(b) O rețea lentă alimentând un receptor de mare capacitate.

În fig. 6-36, putem vedea interpretarea hidraulică a acestei probleme. În fig. 6-36(a), observăm o conductă groasă care duce la un receptor de dimensiune mică. Atâtă timp cât emițătorul nu trimite mai multă apă decât poate conține găleata, apa nu se va pierde. În fig. 6-36(b), factorul de limitare nu mai este capacitatea găleții, ci capacitatea de transport a rețelei. Dacă vine prea repede prea multă apă, ea se va revârsa și o anumită cantitate se va pierde (în acest caz prin umplerea pâlniei).

Soluția din Internet este de a realiza posibilitatea existenței a două probleme - capacitatea rețelei și capacitatea receptorului - și de a le trata pe fiecare separat. Pentru a face acest lucru, fiecare emițător menține două ferestre: fereastra acceptată de către receptor și o a doua fereastră, **fereastra de congestie**. Fiecare reflectă numărul de octeți care pot fi trimiși de către emițător. Numărul octetilor care pot fi trimiși este dat de minimul dintre cele două ferestre. Astfel, fereastra efectivă este minimul dintre ceea ce emițătorul crede că este „în regulă” și ceea ce receptorul crede că este „în regulă”. Dacă receptorul spune: „Trimite 8K octeți”, dar emițătorul știe că o rafală de mai mult de 4K octeți poate aglomera excesiv rețeaua, el va trimite 4K octeți. Din alt punct de vedere, dacă receptorul spune: „Trimite 8K octeți” și emițătorul știe că o rafală de 32K octeți poate străbate fără efort rețeaua, el va trimite toți cei 8K octeți ceruți.

La stabilirea conexiunii, emițătorul inițializează fereastra de congestie la dimensiunea celui mai mare segment utilizat de acea conexiune. El trimite apoi un segment de dimensiune maximă. Dacă acest segment este confirmat înaintea expirării timpului, mai adaugă un segment la fereastra de congestie, făcând-o astfel de dimensiunea a două segmente de dimensiune maximă, și trimite două segmente. O dată cu confirmarea fiecărui din aceste segmente, fereastra de congestie este redimensionată cu încă un segment de dimensiune maximă. Atunci când fereastra de congestie este de n segmente, dacă toate cele n segmente sunt confirmate în timp util, ea este crescută cu numărul de octeți corespunzător celor n segmente. De fapt, fiecare rafală confirmată cu succes dublează fereastra de congestie.

Fereastra de congestie crește în continuare exponential până când sau se produce o depășire de timp, sau se atinge dimensiunea ferestrei receptorului. Ideea este ca dacă rafale de dimensiune, să spunem, 1024, 2048 și 4096 de octeți funcționează fără probleme, dar o rafală de 8192 octeți duce la o depășire de timp, fereastra de congestie va fi stabilită la 4096 de octeți pentru a evita congestia. Atâtă timp cât fereastra de congestie rămâne la 4096, nu va fi transmisă nici o rafală mai mare de această valoare, indiferent cât de mult spațiu de fereastră este oferit de către receptor. Acest algoritm este numit **algoritmul startului lent**, fără a fi însă cătuși de puțin lent (Jacobson, 1988). Este exponential. Toate implementările TCP trebuie să îl suporte.

Să privim acum algoritmul de control al congestiei în cazul Internetului. El utilizează în plus față de ferestrele de recepție și de congestie un al treilea parametru, numit **prag**, inițial de 64K. Atunci când apare o depășire de timp, pragul este pozitionat la jumătate din fereastra curentă de congestie și fereastra de congestie este repositionată la dimensiunea unui segment maxim. Startul lent este utilizat apoi pentru a determina cât poate rețeaua să ducă, atâtă doar că acea creștere exponentială se oprește odată cu atingerea pragului. De aici înainte transmisiile reușite măresc în mod liniar dimensiunea ferestrei de congestie (cu câte un segment maxim pentru fiecare rafală), în locul unei creșteri pentru fiecare segment. De fapt, algoritmul presupune că este acceptabilă înjumătătirea ferestrei de congestie, din acel punct continuându-și gradual calea spre dimensiuni mai mari.

Funcționarea algoritmului de congestie se poate vedea în fig. 6-37. Dimensiunea unui segment maxim este, în acest caz, de 1024 de octeți. Inițial fereastra de congestie are 64K octeți, dar apare o depășire de timp și deci pragul este stabilit la 32K octeți iar fereastra de congestie la 1K octeți acesta fiind punctul 0 al transmisiei din figură. Fereastra de congestie crește apoi exponential până atinge pragul (32K octeți). Începând de aici, creșterea este liniară.

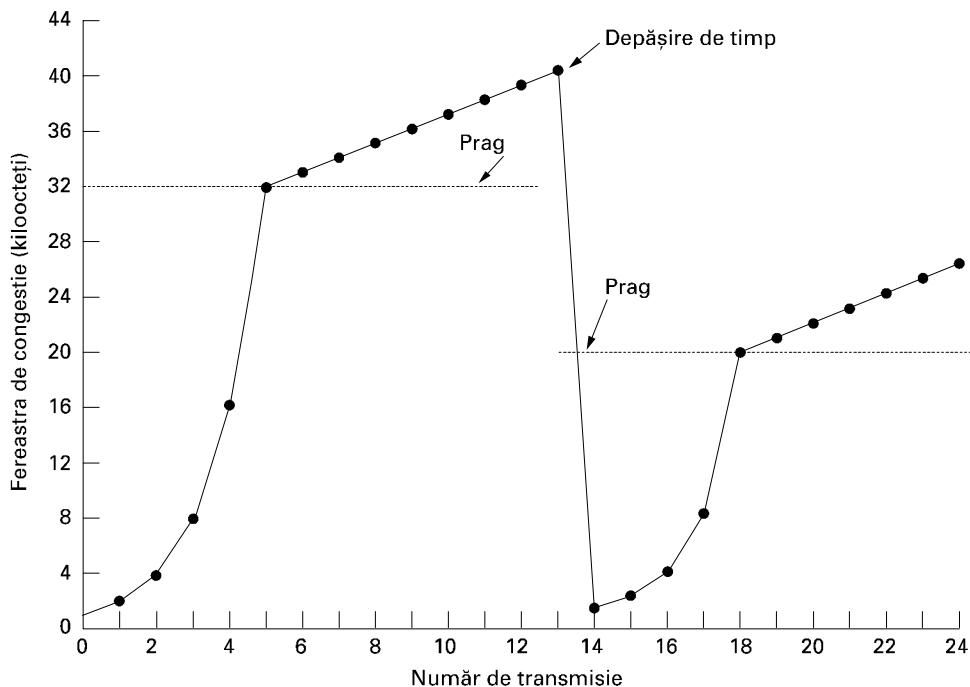


Fig. 6-37. Un exemplu al algoritmului de congestie din Internet.

Transmisia 13 nu are noroc (este de la sine înțeles) și apare o depășire de timp. Pragul este stabilit la jumătate din fereastra curentă (acum 40K octeți, deci jumătate este 20K octeți) și startul lent este inițiat din nou. Atunci când confirmările pentru transmisia 14 încep să sosească, primele patru dublează fiecare fereastra de congestie, dar după aceea creșterea redrevine liniară.

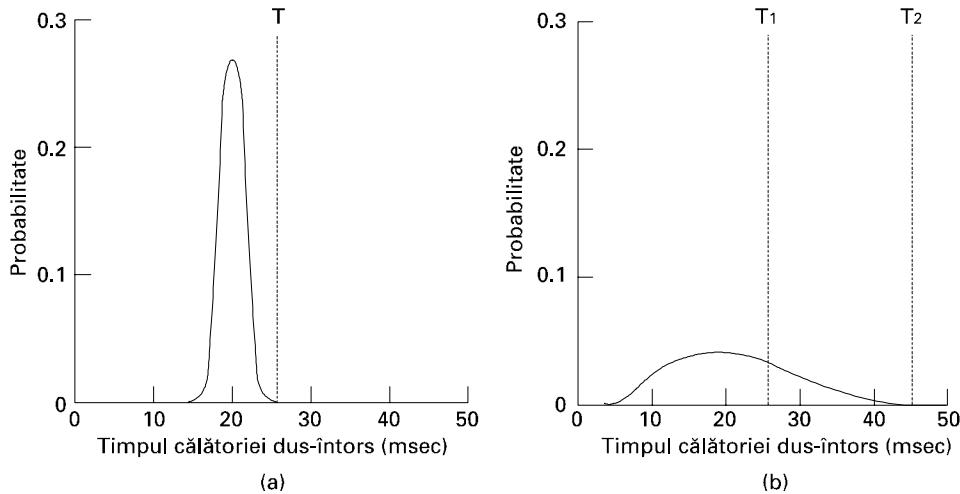
Dacă nu mai apar depășiri de timp, fereastra de congestie va continua să crească până la dimensiunea ferestrei receptorului. În acest punct, creșterea ei va fi oprită și va rămâne constantă atât timp cât nu mai apar depășiri și fereastra receptorului nu își modifică dimensiunea. Ca un alt aspect, dacă un pachet ICMP SOURCE QUENCH sosește și este furnizat TCP-ului, acest eveniment este tratat la fel ca o depășire de timp. O alternativă (și o abordare mai recentă) este descrisă în RFC 3168.

### 6.5.10 Administrarea contorului de timp în TCP

TCP utilizează (cel puțin conceptual) mai multe conțoare pentru a face ceea ce are de făcut. Cel mai important dintre acestea este **contorul de retransmisie**. Atunci când este trimis un segment, se pornește un contor de retransmisie. Dacă segmentul este confirmat înainte de expirarea timpului, contorul este oprit. Pe de altă parte, dacă timpul expiră înaintea primirii confirmării, segmentul este retransmis (și contorul este pornit din nou). Întrebarea care se pune este următoarea: Cât de mare trebuie să fie intervalul de timp până la expirare?

Această problemă este mult mai dificilă la nivelul transport din Internet decât la nivelul protoocoalelor generice de legătură de date prezentate în Cap. 3. În cazul din urmă, întârzierea așteptată este ușor predictibilă (de exemplu, are o varianță scăzută), deci contorul poate fi setat să expire chiar

imediat după ce era așteptată confirmarea, aşa cum se arată în fig. 6-38(a). Cum confirmările sunt rareori întârziate în nivelul legătură de date, absența unei confirmări în momentul în care aceasta era așteptată înseamnă de obicei că s-a pierdut fie cadrul, fie confirmarea.



**Fig. 6-38.** (a) Densitatea de probabilitate a sosirilor în timp a confirmărilor în nivelul legătură de date. (b) Densitatea de probabilitate a sosirii confirmărilor pentru TCP.

TCP trebuie să facă față unui mediu radical diferit. Funcția de densitate a probabilității pentru timpul necesar întoarcerii unei confirmări TCP arată mai degrabă ca în fig. 6-38(b) decât ca în fig. 6-38(a). Este dificilă determinarea timpului în care se realizează circuitul dus-întors până la destinație. Chiar și când acesta este cunoscut, stabilirea intervalului de depășire este de asemenea dificilă. Dacă intervalul este prea scurt, să spunem  $T_1$  în fig. 6-38(b), vor apărea retransmisii inutile, aglomerând Internet-ul cu pachete fără rost. Dacă este prea lung, ( $T_2$ ), performanțele vor avea de suferit datorită întârzierii retransmisiei de fiecare dată când se pierde un pachet. Mai mult decât atât, media și varianța distribuției sosirii confirmărilor se pot schimba cu rapiditate pe parcursul a câtorva secunde atunci când apare sau se rezolvă o congestie.

Soluția este să se utilizeze un algoritm profund dinamic, care ajustează în mod constant intervalul de depășire bazându-se pe măsurători continue ale performanței rețelei. Algoritmul utilizat în general de către TCP este datorat lui Jacobson (1988) și este descris mai jos. Pentru fiecare conexiune, TCP păstrează o variabilă,  $RTT$ , care este cea mai bună estimare a timpului în care se parcurge circuitul dus-întors către destinație în discuție. Atunci când este trimis un segment, se pornește un contor de timp, atât pentru a vedea cât de mult durează până la primirea confirmării cât și pentru a iniția o retransmisie în cazul scurgerii unui interval prea lung. Dacă se primește confirmarea înaintea expirării contorului, TCP măsoară cât de mult i-a trebuit confirmării să sosească, fie acest timp  $M$ . În continuare el actualizează  $RTT$ , după formula:

$$RTT = \alpha RTT + (1 - \alpha)M$$

unde  $\alpha$  este un factor de netezire care determină ponderea dată vechii valori. Uzual,  $\alpha=7/8$ .

Chiar presupunând o valoare bună a lui  $RTT$ , alegerea unui interval potrivit de retransmisie nu este o sarcină ușoară. În mod normal, TCP utilizează  $\beta RTT$ , dar problema constă în alegerea lui  $\beta$ .

În implementările inițiale,  $\beta$  era întotdeauna 2, dar experiența a arătat că o valoare constantă este inflexibilă deoarece nu corespunde în cazul creșterii varianței.

În 1988, Jacobson a propus ca  $\beta$  să fie aproximativ proporțional cu deviația standard a funcției de densitate a probabilității timpului de primire a confirmărilor, astfel încât o varianță mare implică un  $\beta$  mare și viceversa. În particular, el a sugerat utilizarea *deviației medii* ca o estimare puțin costisitoare a *deviației standard*. Algoritmul său presupune urmărirea unei alte variabile de netezire,  $D$ , deviația. La fiecare sosire a unei confirmări, este calculată diferența dintre valorile așteptate și observate  $|RTT - M|$ . O valoare netezită a acesteia este păstrată în  $D$ , prin formula

$$D = \alpha D + (1 - \alpha) |RTT - M|$$

unde  $\alpha$  poate sau nu să aibă aceeași valoare ca cea utilizată pentru netezirea lui  $RTT$ . Deși  $D$  nu este chiar deviația standard, ea este suficient de bună și Jacobson a arătat cum poate fi calculată utilizând doar adunări de întregi, scăderi și deplasări, ceea ce este un mare punct câștigat. Cele mai multe implementări TCP utilizează acum acest algoritm și stabilesc valoarea intervalului de depășire la:

$$\text{Depășire} = RTT + 4 * D$$

Alegerea factorului 4 este într-un fel arbitrară, dar are două avantaje. În primul rând, multiplicarea prin 4 poate fi făcută printr-o singură deplasare. În al doilea rând, minimizează depășirile de timp și retransmisiile inutile, datorită faptului că mai puțin de un procent din totalul pachetelor sosesc cu întârzieri mai mari de patru ori deviația standard. (De fapt, Jacobson a propus inițial să se folosească 2, dar experiența ulterioară a demonstrat că 4 conduce la performanțe mai bune).

O problemă legată de estimarea dinamică a  $RTT$  se referă la ce trebuie făcut în situația în care un segment cauzează o depășire și trebuie retransmis. Atunci când confirmarea este primită, nu este clar dacă aceasta se referă la prima transmisie sau la o transmisie următoare. O decizie eronată poate contamina serios estimarea lui  $RTT$ . Phil Karn a descoperit această problemă cu multă greutate. El este un radio amator entuziasmat, interesat în transmiterea pachetelor TCP/IP prin operare radio, un mediu recunoscut pentru lipsa de fiabilitate (pe o zi senină, la destinație ajung jumătate din pachete). El a făcut o propunere simplă: să nu se actualizeze  $RTT$  pentru nici un segment care a fost retransmis. În loc de aceasta, timpul de expirare este dublat la fiecare eșec, până când segmentele ajung prima oară la destinație. Această corecție este numită **algoritmul lui Karn**. Cele mai multe din implementările TCP utilizează acest algoritm.

Contorul de retransmisie nu este singurul utilizat de către TCP. Un al doilea contor este **contorul de persistență**. El este proiectat să prevină următoarea situație de interblocare. Receptorul trimite o confirmare cu o dimensiune a ferestrei 0, spunându-i emițătorului să aștepte. Mai târziu, receptorul actualizează fereastra, dar pachetul cu această actualizare este pierdut. Acum, atât emițătorul cât și receptorul așteaptă o reacție din partea celuilalt. Atunci când contorul de persistență expiră, emițătorul transmite o sondă către receptor. Răspunsul la această investigare furnizează dimensiunea ferestrei. Dacă această dimensiune continuă să fie zero, contorul de persistență este reposiționat și ciclul se repetă. Dacă este nul, informația poate fi acum transmisă.

Un al treilea contor utilizat de unele implementări este **contorul de menținere în viață**. Când o conexiune a fost inactivă o lungă perioadă de timp, contorul de menținere în viață poate expira pentru a forța una din părți să verifice dacă cealaltă parte există încă. Dacă aceasta nu răspunde, conexiunea este închisă. Această facilitate este controversată, deoarece supraîncarcă rețeaua și poate termina o conexiune altfel sănătoasă datorită unei partiționări tranzitorii a rețelei.

Ultimul contor folosit la fiecare conexiune TCP este acela utilizat în stările de *AȘTEPTARE CONTORIZATĂ* pe parcursul închiderii conexiunilor. El funcționează pe durata a două vieți maximale ale unui pachet, pentru a se asigura că, atunci când o conexiune este închisă, toate pachetele create de aceasta au murit.

### 6.5.11 TCP și UDP în conexiune fără fir

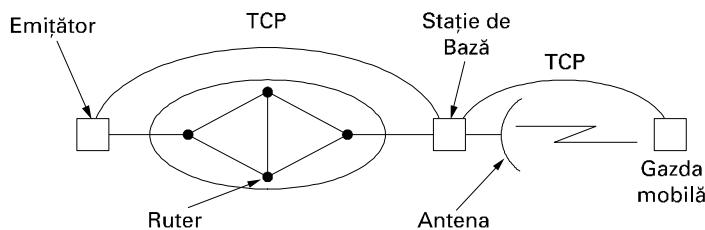
Teoretic, protocoalele de transport ar trebui să fie independente de tehnologia nivelului rețea. În particular, TCP-ului ar trebui să nu-i pese dacă IP rulează peste o rețea cablu sau radio. În practică, acest lucru contează, deoarece cele mai multe implementări TCP au fost atent optimizate pe baza unor presupuneri care sunt adevărate pentru rețele cu cabluri, dar care nu mai sunt valabile în cazul rețelelor fără fir. Ignorarea proprietăților de transmisie fără fir poate conduce la o implementare TCP corectă din punct de vedere logic, dar cu performanțe incredibil de proaste.

Principala problemă este algoritmul de control al congestiei. Aproape toate implementările TCP din zilele noastre pleacă de la premisa că depășirile de timp sunt cauzate de congestie și nu de pierderea pachetelor. În consecință, atunci când expiră un contor, TCP încetinește ritmul și trimite pachete cu mai puțină vigoare (ex. algoritmul startului lent al lui Jacobson). Ideea din spatele acestei abordări constă în reducerea încărcării rețelei, astfel eliminându-se neplăcerile cauzate de congestie.

Din nefericire, legăturile bazate pe transmisia fără fir nu sunt deloc fiabile. Ele pierd tot timpul pachete. Pentru a controla această pierdere a pachetelor, abordarea corectă este să se retrimită cât mai repede posibil. Încetinirea ritmului nu face decât să înrăutățească lucrurile. Dacă presupunem că, atunci când emițătorul transmite 100 de pachete pe secundă, 20% din totalul pachetelor se pierde, productivitatea este de 80 pachete/sec. Dacă emițătorul încetinește ritmul la 50 pachete/sec, productivitatea scade la 40 pachete/sec.

Atunci când se pierde un pachet pe o rețea cu cabluri, emițătorul ar trebui să încetinească ritmul. Atunci când se pierde un pachet pe o rețea fără fir, emițătorul ar trebui să îl măreasca. Dacă emițătorul nu știe despre ce tip de rețea este vorba, luarea unei decizii este dificilă.

În mod frecvent, calea de la emițător la receptor este eterogenă. Primii 1000 km pot să fie într-o rețea cu cabluri, dar ultimul kilometru poate să fie fără fir. Acum, luarea unei decizii în situația unei depășiri de timp este și mai dificilă, dat fiind că intervine și locul în care apare problema. O soluție propusă de Bakne și Badrinath (1995), **TCP indirect**, constă în spargerea conexiunii TCP în două conexiuni separate, ca în fig. 6-39. Prima conexiune pleacă de la emițător la stația de bază. Cea de-a doua leagă stația de bază de receptor. Această stație de bază nu face decât să copieze pachetele din cele două conexiuni în ambele direcții.



**Fig. 6-39.** Spargerea conexiunii TCP în două conexiuni.

Avantajul acestei scheme este acela că ambele conexiuni sunt acum omogene. Depășirile de timp din prima conexiune pot încetini emițatorul, în timp ce depășirile de timp din cea de-a doua îl pot accelera. Alți parametri pot fi de asemenea reglați separat în fiecare din cele două conexiuni. Dezavantajul este acela că este negată însăși semantica TCP. Atâtă timp cât fiecare parte a conexiunii este o conexiune TCP în sine, stația de bază confirmă fiecare segment TCP în mod obișnuit. Doar că acum, receptia unei confirmări de către emițător nu mai înseamnă că receptorul a primit segmentul, ci doar că el a fost primit de către stația de bază.

O soluție diferită, datorată lui Balakrishnan et. al (1995), nu încalcă semantica TCP. Ea se bazează pe mici modificări făcute în codul nivelului rețea din stația de bază. Una din modificări constă în adăugarea unui agent de supraveghere care observă și interceptează pachetele TCP care pleacă spre gazda mobilă precum și confirmările care se întorc de la acesta. Atunci când observă un segment TCP care pleacă spre gazda mobilă, dar nu observă confirmarea receptiōnării acestuia într-un interval de timp dat (relativ scurt), agentul ascuns pur și simplu retransmite acel segment, fără a mai spune sursei acest lucru. De asemenea, el retransmite și atunci când observă confirmări duplicate din partea gazdei mobile, lucru care indică invariabil faptul că aceasta a pierdut ceva. Confirmările duplicate sunt eliminate pe loc, pentru a evita ca sursa să le interpreteze ca un semn de congestie.

Cu toate acestea, un dezavantaj al acestei transparente este acela că, dacă legătura fără fir pierde multe pachete, sursa poate depăși limita de timp în așteptarea unei confirmări și poate invoca în consecință algoritmul de control al congestiei. În cazul TCP-ului indirect, algoritmul de control al congestiei nu va fi niciodată inițiat dacă nu apare într-adevăr o situație de congestie în partea „cablată” a rețelei.

Algoritmul Balakrishnan oferă de asemenea o soluție problemei pierderii segmentelor generate de către gazda mobilă. Atunci când stația de bază constată o pauză în interiorul domeniului numerelor de secvență, aceasta generează o cerere pentru o repetare selectivă a octetului lipsă, utilizând o opțiune TCP.

Utilizând aceste corecturi, legătura fără fir devine mai fiabilă în ambele direcții fără ca sursa să știe acest lucru și fără modificarea semantică TCP.

Desi UDP-ul nu suferă de aceleași probleme ca și TCP-ul, comunicația fără fir induce și pentru el anumite dificultăți. Principala problemă este aceea că programele utilizează UDP se așteaptă ca acesta să fie foarte fiabil. Ele știu că nu este furnizată nici o garanție, dar cu toate acestea se așteaptă ca el să fie aproape perfect. Într-un mediu fără fir, el va fi însă departe de perfecțune. Pentru programele care sunt capabile să se refacă după pierderea mesajelor UDP, dar numai cu un cost considerabil, trecerea bruscă de la un mediu în care mesajele puteau fi pierdute mai mult teoretic decât practic la un mediu în care ele sunt pierdute sistematic poate conduce la un dezastru în ceea ce privește performanțele.

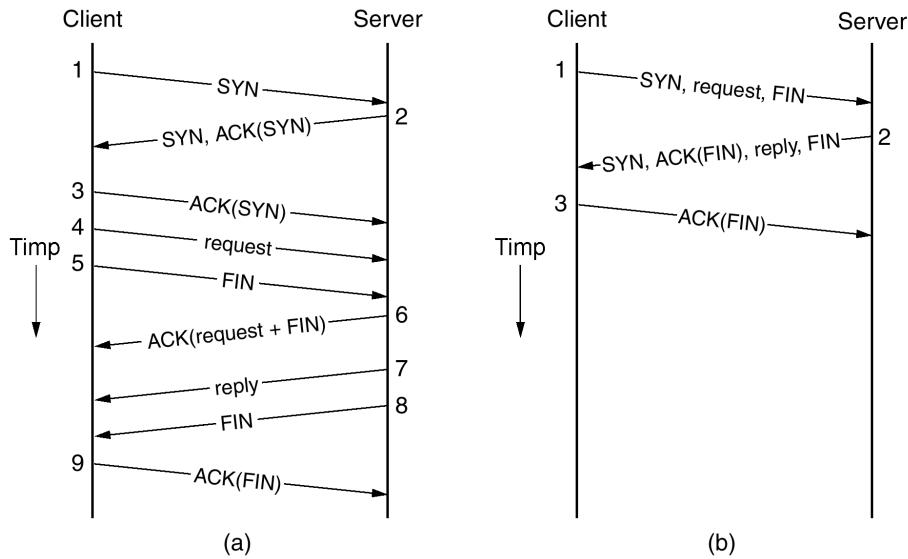
Comunicația fără fir afectează și alte domenii decât cel al performanțelor. De exemplu, cum poate o gazdă mobilă să găsească o imprimantă locală la care să se conecteze, în loc să utilizeze propria imprimantă? Oarecum legată de aceasta este și problema obținerii paginii WWW pentru celula locală, chiar dacă numele ei nu este cunoscut. De asemenea, proiectanții paginilor WWW au tendința să presupună disponibilă o mare largime de bandă. Punerea unei embleme mari pe fiecare pagină poate să devină contraproductivă dacă transmisia paginii printr-o legătură fără fir lentă va dura 10 secunde, și acest lucru ajunge până la urmă să irite utilizatorii.

Cum rețelele cu comunicații fără fir devin tot mai comune, problema rulării TCP-ului pe ele a devenit tot mai acută. Documentații suplimentare în acest domeniu se găsesc în (Barakat ș.a., 2000; Ghani și Dixit, 1999; Huston, 2001; și Xylomenos ș.a., 2001).

### 6.5.12 TCP Tranzacțional

Mai devreme în acest capitol am analizat apelul de proceduri la distanță ca modalitate de a implementa sistemele client-server. Dacă atât cererea, cât și răspunsul sunt suficient de mici încât să se potrivească în pachete simple și operația este idempotentă, UDP-ul poate fi ușor utilizat. Totuși, dacă aceste condiții nu sunt îndeplinite, utilizarea UDP-ului este mai puțin atractivă. De exemplu, dacă răspunsul este unul lung, atunci datagramele trebuie să fie secvențiate și trebuie inițiat un mecanism pentru a retransmite datagramele pierdute. De fapt, aplicației îi este cerut să reinventeze TCP-ul.

În mod cert, acest lucru nu este atractiv, dar nici utilizarea TCP-ului în sine nu este atractivă. Problema este eficiența. Secvența normală a pachetelor pentru a face un RPC peste TCP este prezentată în fig. 6-40(a). În cel mai bun caz sunt necesare nouă pachete.



**Fig. 6-40.** (a) RPC folosind TCP clasic ; (b) RPC folosind T/TCP

Cele nouă pachete sunt după cum urmează:

1. Clientul trimite un pachet SYN pentru a stabili o conexiune.
2. Serverul trimite un pachet ACK pentru a recunoaște pachetul SYN.
3. Clientul finalizează înțelegerea în trei pași.
4. Clientul trimite cererea reală.
5. Clientul trimite un pachet FIN pentru a indica dacă s-a terminat trimiterea.
6. Serverul confirmă cererea și FIN-ul.
7. Serverul trimite răspunsul înapoi clientului.
8. Serverul trimite un pachet FIN pentru a indica că și acest lucru s-a încheiat.
9. Clientul confirmă FIN-ul server-ului.

A se reține că acesta este cazul ideal. În cazul cel mai rău, cererea clientului și FIN-ul sunt confirmate separat, precum sunt răspunsul server-ului și FIN-ul.

Întrebarea care apare imediat este dacă există vreo posibilitate de a combina eficiența RPC-ului folosind UDP (doar 2 mesaje) cu fiabilitatea TCP-ului. Răspunsul este: aproape că da. Se poate

realiza cu o variantă experimentală de TCP numită **T/TCP** (**Transactional TCP**, rom: TCP tranzacțional), care este descrisă în RFC 1379 și 1644.

Ideea principală este aceea de a modifica secvența standard de inițializare a conexiunii astfel încât să permită, la nivel înalt, transferul de date în timpul inițializării. Protocolul T/TCP este prezentat în fig. 6-40(b). Primul pachet al clientului conține bitul SYN, cererea în sine și FIN-ul. De fapt acesta spune: vreau să stabilesc o conexiune, aici sunt datele și am terminat

Când serverul primește cererea, caută sau calculează răspunsul și alege modul în care să răspundă. Dacă răspunsul începe într-un pachet, dă răspunsul din fig. 6-40(b), care spune: confirm FIN-ul tău, iată răspunsul, iar eu am terminat. Clientul confirmă apoi FIN-ul server-ului și protocolul ia sfârșit în (după) trei mesaje.

În orice caz, dacă rezultatul este mai mare de 1 pachet, serverul are de asemenea și opțiunea de a nu seta bitul FIN, caz în care poate trimite pachete multiple înainte de a-i închide direcția.

Merită probabil menționat faptul că T/TCP nu este singura îmbunătățire propusă pentru TCP. O altă propunere este **SCTP** (**Stream Control Transmission Protocol**, rom: Protocolul de control al transmisiei fluxului). Caracteristicile sale includ păstrarea legăturilor dintre mesaje, modalități multiple de livrare (de ex: livrarea neordonată), găzduirea multiplă (destinații de rezervă), și confirmări selective (Stewart and Metz, 2001). În orice caz, oricând cineva propune să schimbe ceva care a funcționat atât de bine de atâta timp, se duce o luptă aprigă între tabăra “utilizatorii doresc mai multe facilități” și cea “dacă nu e stricat, nu-l repară!”.

## 6.6 ELEMENTE DE PERFORMANȚĂ

În rețelele de calculatoare sunt foarte importante elementele de performanță. Atunci când sunt interconectate sute sau mii de calculatoare, au loc adesea interacțiuni complexe cu consecințe nebănuite. Această complexitate conduce în mod frecvent la performanțe slabe fără ca cineva să știe de ce. În secțiunile următoare vom examina mai multe elemente legate de performanța rețelei pentru a identifica tipurile de probleme și ce poate fi făcut pentru rezolvarea lor.

Din nefericire, înțelegerea performanței rețelei este mai degrabă o artă decât o știință. Este prea puțină teorie care stă la bază și de fapt aceasta nu folosește în situații practice. Cel mai bun lucru pe care îl putem face este să indicăm reguli rezultate dintr-o experiență îndelungată și să prezintăm exemple luate din lumea reală. Am amânat în mod intenționat această discuție după studiul nivelului transport din rețelele TCP, pentru a fi capabili să folosim TCP ca exemplu în diverse locuri.

Nivelul transport nu este singurul loc unde apar elemente legate de performanță. Am văzut unele elemente la nivelul rețea, în capitolul precedent. Cu toate acestea, nivelul rețea trebuie să fie preocupat în mare măsură de rutare și controlul congestiei. Problemele mai generale, orientate spre sistem, trebuie să fie legate de nivelul transport, așa că acest capitol este locul potrivit pentru a le examina.

În următoarele cinci secțiuni vom examina cinci aspecte de performanță ale rețelei:

1. Probleme de performanță.
2. Măsurarea performanței rețelei.
3. Proiectarea de sistem pentru performanțe mai bune.
4. Prelucrarea rapidă TPDU.
5. Protocole pentru rețele viitoare de mare performanță.

În sfârșit, sunt utile câteva cuvinte despre programul de protocol. Cea mai mare atenție trebuie acordată cazului de succes. Multe din protocolele vechi aveau tendința de a evidenția ce este de făcut atunci când ceva nu mergea cum trebuie (de exemplu pierderea unui pachet). Pentru a face protocolele să meargă mai repede, proiectanții ar trebui să aibă ca scop minimizarea timpului de prelucrare atunci când totul funcționează corect. Minimizarea timpului de prelucrare în caz de eroare trebuie să treacă pe planul doi.

Un al doilea aspect legat de programe este minimizarea timpului de copiere. Așa cum am văzut mai devreme, copierea datelor introduce în general un cost suplimentar. Ideal ar fi ca echipamentul fizic să depoziteze în memorie fiecare pachet recepționat ca un bloc contiguu de date. Programul ar trebui apoi să copieze acest pachet în tamponul utilizator printre-o singură copiere de bloc. În funcție de modul de lucru al memoriei tampon, ar fi de dorit chiar să se evite copierea în buclă. Cu alte cuvinte, cel mai rapid mod de a copia 1024 de cuvinte este de a avea 1024 de instrucțiuni MOVE una după cealaltă (sau 1024 de perechi încărcare-memorare). Rutina de copiere este critică într-o asemenea măsură, încât, dacă nu există altă modalitate de a păcăli compilatorul ca să producă cu precizie codul optimal, ea ar trebui scrisă de mâna, cu multă atenție, direct în cod de asamblare.

## 6.7 REZUMAT

Nivelul transport reprezintă cheia pentru înțelegerea protocolelor organizate pe niveluri. El furnizează diferite servicii, cel mai important dintre acestea fiind fluxul de octeți capăt-la-capăt de la emițător la receptor, fiabil și orientat pe conexiuni. El este accesat prin primitive de serviciu care permit stabilirea, utilizarea și eliberarea conexiunilor. O interfață de nivel de transport obișnuită este cea oferită de soclurile Berkeley.

Protocolele de transport trebuie să fie capabile să controleze conexiunea în rețele nefiabile. Stabilirea conexiunii este complicată de existența pachetelor duplicate întârziate, care pot apărea la momente inopertune. Pentru a le face față, stabilirea conexiunii trebuie făcută prin intermediul protocolelor cu înțelegere în trei pași. Eliberarea unei conexiuni este mai simplă decât stabilirea sa, dar este încă departe de a fi banală datorită problemei celor două armate.

Chiar și în cazul unui nivel rețea complet fiabil, nivelul transport are suficient de mult de lucru. El trebuie să controleze toate primitivele de serviciu, toate conexiunile și contoarele de timp și trebuie să aloce și să utilizeze credite.

Internetul are două protocoale de transport principale: UDP și TCP. UDP este un protocol neorientat pe conexiune care este în principal un ambalaj pentru pachetele IP, cu caracteristicile suplimentare de multiplexare și demultiplexare a proceselor multiple folosind o singura adresa de IP. UDP poate fi folosit pentru interacțiunile client-server, de exemplu, utilizând RPC. De asemenea poate fi folosit pentru construirea protocolelor în timp real cum ar fi RTP.

Principalul protocol de transport în Internet este TCP. El oferă un flux sigur, bidirectional de octeți. El utilizează un antet de 20 de octeți pentru toate segmentele. Segmentele pot fi fragmentate de rutere în cadrul Internet-ului, deci calculatoarele găzdui trebuie să fie pregătite să lereasambleze. S-a depus un mare efort pentru optimizarea performanțelor TCP-ului, utilizând algoritmii propuși de Nagle, Clark, Jacobson, Karn și alții. Legăturile fără fir adaugă o varietate de complicații TCP-ului.

TCP Tranzacțional este o extensie a TCP-ului care se ocupă de interacțiunile client-server cu un număr redus de pachete.

Performanțele rețelei sunt dominate în mod tipic de protocol și de costul suplimentar datorat tratării TPDU-urilor, situație care se înrăutățește la viteze mari. Protocoalele ar trebui proiectate astfel, încât să minimizeze numărul de TPDU-uri, copierile lor repetitive și comutările de context. Pentru rețelele gigabit sunt de dorit protocole simple.

## 6.8 PROBLEME

1. În exemplele noastre de primitive de transport din fig. 6-2, LISTEN este un apel blocant. Este acest lucru strict necesar? Dacă nu, explicați cum ar putea fi utilizată o primitivă neblocantă. Ce avantaje ar avea aceasta pentru schema descrisă în text?
2. În modelul pe care se bazează fig. 6-4 se presupune că pachetele pot fi pierdute de către nivelul rețea și trebuie deci să fie confirmate individual. Să presupunem că nivelul rețea este 100% fiabil și nu pierde pachete niciodată. Ce modificări sunt necesare (dacă sunt necesare) în fig. 6-4?
3. În ambele părți ale fig. 6-6 este un comentariu conform căruia valoarea SERVER\_PORT trebuie să fie aceeași și în client și în server. De ce este acest lucru atât de important?
4. Să presupunem că pentru generarea numerelor de secvență inițiale se utilizează o schemă dirijată de ceas cu un contor de timp de 15 biți. Ceasul generează un impuls la fiecare 100 ms și durata de viață maximă a unui pachet este de 60 sec. Cât de des este necesar să aibă loc o resynchronizare
  - a) în cel mai rău caz?
  - b) atunci când se consumă 240 de numere de secvență pe secundă?
5. De ce este necesar ca timpul maxim de viață al unui pachet,  $T$ , să fie suficient de mare pentru a acoperi nu numai dispariția pachetului, dar și a confirmării?
6. Să ne imaginăm că pentru stabilirea unei conexiuni se utilizează un protocol cu înțelegere în doi pași și nu unul cu înțelegere în trei pași. Cu alte cuvinte, al treilea mesaj nu mai este necesar. Sunt posibile interblocări în această situație? Dați un exemplu sau arătați că nu există nici o interblocare.
7. Imagineați o problemă generalizată a celor  $n$  armate, în care acordul dintre oricare două armate este suficient pentru victorie. Există un protocol care îi permite albastrului să câștige?
8. Să considerăm problema recuperării după defectarea unei mașini gazdă (adică fig. 6-18). Dacă intervalul dintre scrierea și trimiterea unei confirmări, sau vice-versa, poate fi făcut relativ scurt, care sunt cele mai bune strategii emițător-receptor pentru minimizarea şansei de defectare a protocolului?
9. Sunt posibile interblocările pentru entitățile transport descrise în text (fig. 6-20)?

# 7

## NIVELUL APLICAȚIE

După ce am epuizat toate preliminariile putem aborda nivelul unde pot fi găsite toate aplicațiile. Nivelurile de sub nivelul aplicație servesc la asigurarea unui transport sigur, dar nu îndeplinesc nici o funcție concretă pentru utilizatori. În acest capitol vom studia câteva aplicații reale.

Totuși, chiar și la nivelul aplicație, apare necesitatea unor protocoale-suport care să permită funcționarea aplicațiilor reale. Înainte de a începe studiul aplicațiilor, ne vom ocupa de unul dintre acestea. Subiectul în discuție este DNS, care se ocupă de convențiile de nume în Internet. După aceea vom examina trei aplicații reale: poșta electronică, World Wide Web (rom.: rețea de întindere planetară) și, în final, multimedia.

### 7.1 DNS - SISTEMUL NUMELOR DE DOMENII

Cu toate că teoretic programele ar putea să se refere la sistemele gazdă, la cutiile poștale și la alte resurse prin adresa lor de rețea (de exemplu prin adresa IP), aceste adrese sunt greu de memorat de către oameni. De asemenea, în trimitera de poștă electronică la *tana@128.111.24.41* ar însemna că dacă furnizorul de servicii Internet sau organizația Tanei mută serverul de poștă pe o mașină diferită, cu o adresă IP diferită, adresa ei de e-mail se va schimba. De aceea au fost introduse nume ASCII pentru a separa numele mașinilor de adresele mașinilor. În acest fel, adresa Tanei ar putea fi ceva de genul *tana@art.ucsb.edu*. Cu toate acestea, rețeaua înțelege numai adrese numerice, deci este necesar un mecanism care să convertească șirurile ASCII în adrese de rețea. În secțiunile următoare se va studia cum este realizată această conversie în Internet.

Încă de la ARPANET exista un fișier *host.txt* care cuprindea toate sistemele gazdă și adresele lor IP. În fiecare noapte, toate gazdele îl preluau de la situl unde era păstrat. Pentru o rețea formată din câteva sute de mașini mari, cu divizarea timpului, această abordare era destul de rezonabilă.

Totuși, atunci când la rețea au fost conectate mii de stații de lucru, toți și-au dat seama că această abordare nu putea să funcționeze la nesfârșit. În primul rând dimensiunea fișierului ar devine prea mare. Cu toate acestea și chiar mai important, conflictele de nume de sisteme gazdă ar apărea în permanență dacă nu ar fi administrate centralizat, ceva de negădit într-o rețea internațională de dimensiuni uriașe din cauza încărcării și a latenței. Pentru a rezolva aceste probleme, a fost inventat **DNS (Domain Name System - Sistemul numelor de domenii)**.

Esența DNS-ului constă într-o schemă ierarhică de nume de domenii și a unui sistem de baze de date distribuite pentru implementarea acestei scheme de nume. În principal este utilizat pentru a pune în corespondență numele sistemelor gazdă și adresele destinațiilor de e-mail cu adresele IP, dar poate fi utilizat și pentru alte scopuri. DNS este definit în RFC-urile 1034 și 1035.

Foarte pe scurt, DNS este utilizat după cum urmează. Pentru a stabili corespondența dintre un nume și o adresă IP, programul de aplicatie apelează o procedură de bibliotecă numită **resolver**, transferându-i numele ca parametru. Putem vedea un exemplu de resolver, *gethostbyname*, în fig. 6-6. Resolver-ul trimite un pachet UDP la serverul DNS local, care caută numele și returnează adresa IP către resolver, care o returnează apelantului. Înarmat cu adresa IP, programul poate stabili o conexiune TCP cu destinația sau îi poate trimite pachete UDP.

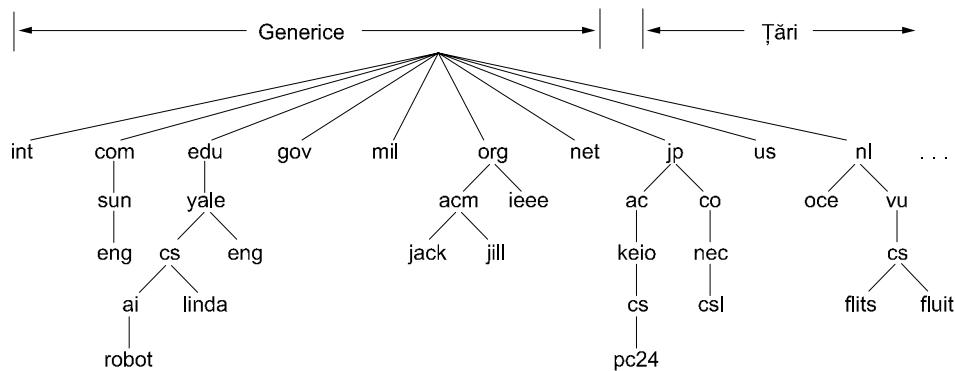
### 7.1.1 Spațiul de nume DNS

Administrarea unui volum mare de nume în permanentă schimbare nu este o problemă prea ușoară. În sistemul poștal, administrarea numelor este realizată impunând ca pe o scrisoare să se specifice (implicit sau explicit) țara, statul sau provincia, orașul, strada și restul adresei destinatarului. Utilizând o astfel de adresare ierarhică, nu există nici o confuzie între Marvin Anderson de pe Main St. din White Plains, N.Y. și Marvin Anderson de pe Main St. din Austin, Texas. DNS lucrează în același mod.

Conceptual, Internetul este divizat în peste 200 **domenii** de nivel superior, fiecare domeniu cuprinzând mai multe sisteme gazdă. Fiecare domeniu este partionat în subdomenii și acestea sunt, la rândul lor, partitionate și.a.m.d. Toate aceste domenii pot fi reprezentate ca un arbore, aşa cum se arată în fig. 7-1. Frunzele arborelui reprezintă domenii care nu au subdomenii (dar, bineînțeles, conțin sisteme). Un domeniu frunză poate conține un singur sistem gazdă sau poate reprezenta o firmă, deci să conțină mii de sisteme gazdă.

Domeniile de pe primul nivel se împart în două categorii: generice și de țări. Domeniile generice sunt *com* (comercial), *edu* (instituții educaționale), *gov* (guvernul federal al SUA), *int* (organizații internaționale), *mil* (forțele armate ale SUA), *net* (furnizori Internet) și *org* (organizații nonprofit). Domeniile de țări includ o intrare pentru fiecare țară, cum se definește în ISO 3166.

În noiembrie 2000, ICANN a aprobat patru domenii de nivel superior noi, de interes general, și anume, *biz* (afaceri), *info* (informații), *name* (nume de persoane), și *pro* (profesii, ca de exemplu doctori sau avocați). În plus, au fost introduse trei domenii de nivel superior cu caracter specializat, curate de către anumite industrii. Acestea sunt *aero* (industria aerospațială), *coop* (cooperative), și *museum* (muzee). În viitor vor fi adăugate alte domenii superioare.



**Fig. 7-1.** O porțiune a spațiului numelor de domenii din Internet.

Pe de altă parte, pe măsură ce Internetul devine mai comercial, el devine și mai discutabil. Să luăm domeniul *pro*, de exemplu, care a fost proiectat pentru profesioniștii atestați. Dar cine este un profesionist? Si de cine este atestat? Dar cum rămâne cu fotografii, profesorii de pian, magicienii, instalatorii, frizerii, exterminatorii, artiștii de tatuaje, mercenarii și prostitutele? Sunt acestea meserii și sunt acceptabile pentru domeniile *pro*? Si dacă da, cine atestă diversi practicieni?

În general, obținerea unui domeniu de nivel secundar, ca de exemplu *nume-al-companiei.com*, este usoară. Pur și simplu este necesară doar consultarea serviciului de înregistrare al nivelului superior corespunzător (*com* în acest caz) pentru a vedea dacă numele dorit este disponibil și nu aparține altcuiva. Dacă nu sunt probleme, solicitantul plătește o mică taxă anuală și primește numele. Până acum, cam toate cuvintele comune (din engleză) au fost luate în domeniul *com*. Încercați nume de articole casnice, animale, plante, părți ale corpului etc. Aproape toate sunt luate.

Fiecare domeniu este identificat prin calea în arbore de la el la domeniul (fără nume) rădăcină. Componentele sunt separate prin puncte (pronunțat „dot”). Astfel, departamentul tehnic de la Sun Microsystems ar putea fi *eng.sun.com*, în loc de numele în stil UNIX */com/sun/eng*. De notat că această numire ierarhică face ca *eng.sun.com* să nu intre în conflict cu posibila utilizare a lui *eng* din *eng.yale.edu*, care ar putea fi folosit pentru departamentul de limba engleză de la Yale.

Numele de domenii pot fi absolute sau relative. Un nume absolut de domeniu se termină cu un punct (de exemplu, *eng.sun.com*), în timp ce unul relativ nu. Numele relative trebuie interpretate în context pentru a le determina înțelesul adeverat. În ambele cazuri, un nume de domeniu se referă la un anumit nod din arbore și la toate nodurile de sub el.

Numele de domenii nu fac distincție între litere mici și litere mari, astfel *edu*, *Edu*, sau *EDU* înseamnă același lucru. Componentele numelor pot avea o lungime de cel mult 63 caractere, iar întreaga cale de nume nu trebuie să depășească 255 de caractere.

În principiu, domeniile pot fi inserate în arbore în două moduri diferite. De exemplu, *cs.yale.edu* ar putea la fel de bine să fie inclus în domeniul țării *us* ca *cs.yale.ct.us*. În practică, totuși, aproape toate organizațiile din Statele Unite sunt repartizate după criteriul generic, iar aproape toate din afara Statelor Unite fac parte din domeniul țării lor. Nu există nici o regulă împotriva înregistrării sub două domenii de nivel superior, însă puține organizații în afară de cele multinaționale o fac (de exemplu, *sony.com* și *sony.nl*).

Fiecare domeniu controlează cum sunt alocate domeniile de sub el. De exemplu, Japonia are domeniile *ac.jp* și *co.jp* echivalente cu *edu* și *com*. Olanda nu face nici o distincție și pune toate orga-

nizațiile direct sub *nl*. Astfel următoarele trei nume sunt toate departamente de calculatoare (computer science) din universități:

1. *cs.yale.edu* (Universitatea Yale din Statele Unite).
2. *cs.vn.nl* (Vrije Universiteit în Olanda).
3. *cs.keio.ac.jp* (Universitatea Keio din Japonia).

Pentru a crea un nou domeniu, se cere permisiunea domeniului în care va fi inclus. De exemplu, dacă un grup VLSI de la Yale dorește să fie cunoscut ca *vlsi.cs.yale.edu*, acesta are nevoie de permisiunea celui care administrează *cs.yale.edu*. Similar, dacă este acreditată o nouă universitate, să zicem Universitatea din Northern South Dakota, ea trebuie să ceară administratorului domeniului *edu* să-i atribuie *unsd.edu*. În acest mod sunt evitate conflictele de nume și fiecare domeniu poate ține evidență tuturor subdomeniilor sale. Odată ce un nou domeniu a fost creat și înregistrat, el poate crea subdomenii, cum ar fi *cs.unsd.edu*, fără a cere permisiunea de la cineva din partea superioară a arborelui.

Atribuirea de nume respectă granițele organizaționale, nu pe cele ale rețelelor fizice. De exemplu, dacă departamentele de știință calculatoarelor și de inginerie electrică sunt localizate în aceeași clădire și folosesc aceeași rețea locală, ele pot avea totuși domenii distincte. Similar, dacă departamentul de știință calculatoarelor este împărțit în două clădiri (Babbage Hall și Turing Hall), toate sistemele gazdă din ambele clădiri aparțin aceluiași domeniu.

### 7.1.2 Înregistrări de resurse

Fiecarui domeniu, fie că este un singur calculator gazdă, fie un domeniu de nivel superior, îi poate fi asociată o mulțime de înregistrări de resurse (**resource records**). Pentru un singur sistem gazdă, cea mai obișnuită înregistrare de resursă este chiar adresa IP, dar există multe alte tipuri de înregistrări de resurse. Atunci când un resolver trimite un nume de domeniu către un DNS, ceea ce va primi ca răspuns sunt înregistrările de resurse asociate aceluia nume. Astfel, adevărată funcție a DNS este să realizeze corespondența dintre numele de domenii și înregistrările de resurse.

O înregistrare de resursă este un 5-tuplu. Cu toate că, din rațiuni de eficiență, înregistrările de resurse sunt codificate binar, în majoritatea expunerilor ele sunt prezentate ca text ASCII, câte o înregistrare de resursă pe linie. Formatul pe care îl vom utiliza este următorul:

Nume\_domeniu Timp\_de\_viață Clasă Tip Valoare

*Nume\_domeniu* (*domain\_name*) precizează domeniul căruia i se aplică această înregistrare. În mod normal există mai multe înregistrări pentru fiecare domeniu și fiecare copie a bazei de date păstrează informații despre mai multe domenii. Acest câmp este utilizat ca cheie de căutare primară pentru a satisface cererile. Ordinea înregistrărilor în baza de date nu este semnificativă.

Câmpul *Timp\_de\_viață* (*time\_to\_live*) dă o indicație despre cât de stabilă este înregistrarea. Informația care este foarte stabilă are asigurată o valoare mare, cum ar fi 86400 (numărul de secunde dintr-o zi). Informației instabile îi este atribuită o valoare mică, cum ar fi 60 (1 minut). Vom reveni la acest punct mai târziu, când vom discuta despre utilizarea memoriei ascunse.

Al treilea câmp dintr-o înregistrare de resursă este *Clasa* (*class*). Pentru informațiile legate de Internet este tot timpul *IN*. Pentru alte informații pot fi folosite alte coduri, însă în practică acestea se întâlnesc rar.

Câmpul *Tip* (*type*) precizează tipul înregistrării. Cele mai importante tipuri sunt prezentate în fig. 7-2.

Tip	Semnificație	Valoare
SOA	Start autoritate	Parametrii pentru această zonă
A	Adresa IP a unui sistem gazdă	Întreg pe 32 de biți
MX	Schimb de poștă	Prioritate, domeniu dispus să accepte poștă electronică
NS	Server de Nume	Numele serverului pentru acest domeniu
CNAME	Nume canonic	Numele domeniului
PTR	Pointer	Pseudonim pentru adresa IP
HINFO	Descriere sistem gazdă	Unitate centrală și sistem de operare în ASCII
TXT	Text	Text ASCII neinterpretat

Fig. 7-2. Principalele tipuri de înregistrări de resurse DNS.

O înregistrare *SOA* furnizează numele sursei primare de informații despre zona serverului de nume (descrișă mai jos), adresa de e-mail a administratorului, un identificator unic și diversi indicaitori și conțoare de timp.

Cel mai important tip de înregistrare este înregistrarea *A* (adresă). Ea păstrează adresa IP de 32 de biți a unui sistem gazdă. Fiecare sistem gazdă Internet trebuie să aibă cel puțin o adresă IP, astfel încât alte mașini să poată comunica cu el. Unele sisteme gazdă au două sau mai multe conexiuni în rețea, caz în care vor avea câte o înregistrare de tip *A* pentru fiecare conexiune (și astfel pentru fiecare adresă IP).

Următoarea ca importanță este înregistrarea *MX*. Aceasta precizează numele sistemului gazdă pregătit să accepte poșta electronică pentru domeniul specificat. El este folosit deoarece nu toate mașinile sunt pregătite să accepte poșta electronică pentru domeniul specificat. Dacă cineva vrea să-i trimită un e-mail, de exemplu, lui *bill@microsoft.com*, sistemul care trimite trebuie să găsească un server la *microsoft.com* dispus să accepte e-mail. Înregistrarea *MX* poate să furnizeze această informație.

Înregistrările *NS* specifică serverele de nume. De exemplu, fiecare bază de date DNS are în mod normal o înregistrare *NS* pentru fiecare domeniu de pe primul nivel, astfel încât, de exemplu, poșta electronică să poată fi trimisă în zone îndepărtate ale arborelui de nume. Vom reveni la acest aspect mai târziu.

Înregistrările *CNAME* permit crearea pseudonimelor. De exemplu, o persoană familiarizată cu atribuirea numelor în Internet, care dorește să trimită un mesaj unei persoane al cărei nume de conectare la un sistem de calcul din departamentul de calculatoare de la M.I.T. este *paul* poate presupune că adresa *paul@cs.mit.edu* este corectă. De fapt această adresă nu este corectă, deoarece domeniul departamentului de calculatoare de la M.I.T. este *lcs.mit.edu*. Totuși, ca un serviciu pentru cei care nu știu acest lucru, M.I.T. poate crea o intrare *CNAME*, pentru a dirija persoanele și programele în direcția corectă. O astfel de intrare poate fi:

```
cs.mit.edu 86400 IN CNAME lcs.mit.edu
```

Ca și *CNAME*, *PTR* se referă la un alt nume. Totuși, spre deosebire de *CNAME*, care este în realitate numai o macro-definiție, *PTR* este un tip de date DNS a cărui interpretare depinde de contextul în care este utilizat. În practică este aproape întotdeauna utilizat pentru asocierea unui nume cu o adresă IP, pentru a permite căutarea adresei IP și obținerea numelui mașinii corespunzătoare. Acestea se numesc **căutări inverse (reverse lookups)**.

Înregistrările *HINFO* permit aflarea tipului de mașină și de sistem de operare cărora le corespunde domeniul. În sfârșit, înregistrările *TXT* permit domeniilor să se autoidentifice într-un mod arbitrar. Aceste două tipuri de înregistrări sunt introduse pentru ușurința utilizatorului. Nici una

dintre ele nu este necesară, astfel încât programele nu pot conta pe obținerea lor (și probabil că dacă le obțin nu le pot trata).

În final ajungem la câmpul *Valoare*. Acest câmp poate fi un număr, un nume de domeniu sau un sir ASCII. Semantica depinde de tipul de înregistrare. O scurtă descriere a câmpurilor *Valoare* pentru fiecare dintre principalele tipuri de înregistrări este dată în fig. 7-2.

Un exemplu de informație ce se poate găsi în baza de date DNS a unui domeniu este prezentat în fig. 7-3. Această figură prezintă o parte (semi-ipotetică) a bazei de date pentru domeniul *cs.vu.nl* prezentat în fig. 7-1. Baza de date conține șapte tipuri de înregistrări de resurse.

;Baza de date pentru cs.vu.nl			
cs.vu.nl.	86400	IN SOA	star boss (9527, 7200, 7200, 241920, 86400)
cs.vu.nl.	86400	IN TXT	„Divisie Wiskunde en Informatica.”
cs.vu.nl.	86400	IN TXT	„Vrije Universiteit Amsterdam.”
cs.vu.nl.	86400	IN MX	1 zephyr.cs.vu.nl.
cs.vu.nl.	86400	IN MX	2 top.cs.vu.nl.
flits.cs.vu.nl.	86400	IN HINFO	Sun Unix
flits.cs.vu.nl.	86400	IN A	130.37.16.112
flits.cs.vu.nl.	86400	IN A	192.31.231.165
flits.cs.vu.nl.	86400	IN MX	1 flits.cs.vu.nl.
flits.cs.vu.nl.	86400	IN MX	2 zephyr.cs.vu.nl.
flits.cs.vu.nl.	86400	IN MX	3 top.cs.vu.nl.
www.cs.vu.nl.	86400	IN CNAME	star.cs.vu.nl.
ftp.cs.vu.nl.	86400	IN CNAME	zephyr.cs.vu.nl.
rowboat		IN A	130.37.56.201
		IN MX	1 rowboat
		IN MX	2 zephyr
		IN HINFO	Sun Unix
little-sister		IN A	130.37.62.23
		IN HINFO	Mac MacOS
laserjet		IN A	192.31.231.216
		IN HINFO	„HP Laserjet IISi” Proprietary

**Fig. 7-3.** O parte dintr-o posibilă bază de date DNS pentru *cs.vu.nl*.

Prima linie necomentată din fig. 7-3 dă câteva informații de bază despre domeniu, de care nu ne vom ocupa. Următoarele două linii furnizează informații textuale despre amplasarea domeniului. Urmează două intrări care specifică primul și al doilea loc unde se încearcă să se livreze poșta electronică trimisă pentru *persoana@cs.vu.nl*. Mai întâi se încearcă trimiterea la mașina *zephyr*. Dacă aceasta eşuează, atunci trebuie să se încerce la *top*.

După o linie liberă, adăugată numai pentru claritate, urmează linii care spun că *flits* este o stație de lucru Sun care lucrează sub UNIX și se specifică ambele sale adrese IP. Urmează trei variante de tratare a poștei electronice trimise la *flits.cs.vu.nl*. Prima alegere este, în mod natural, chiar *flits*, iar dacă nu se reușește, se încearcă la *zephyr* și apoi la *top*. Urmează un pseudonim, *www.cs.vu.nl*, astfel ca această adresă să poată fi utilizată fără a specifica o anumită mașină. Crearea acestui pseudonim permite ca *cs.vu.nl* să schimbe serverul *www* fără invalidarea adresei folosite în mod curent pentru adresarea lui. Un argument similar este valabil pentru *ftp.cs.vu.nl*.

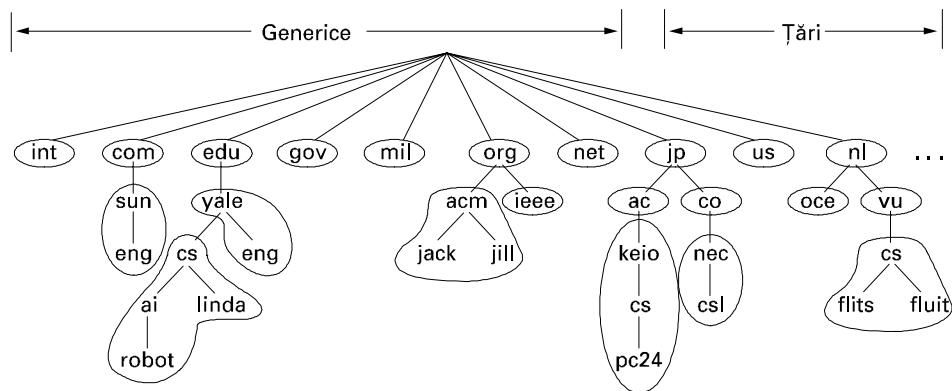
Următoarele patru linii conțin o înregistrare tipică pentru o stație de lucru, în acest caz *rowboat*. *cs.vu.nl*. Informația furnizează adresa IP, destinația primară și secundară pentru poșta electronică și

informații despre mașină. Urmează o intrare pentru un sistem non-UNIX care nu este capabil să primească poșta el însuși, urmat de o intrare pentru o imprimantă laser conectată la Internet.

Ceea ce nu este arătat (și nu există în acest fișier) sunt adresele IP utilizate pentru a căuta adresele domeniilor de pe primul nivel. Acestea sunt necesare pentru a căuta sistemele gazdă aflate la distanță, dar, deoarece ele nu fac parte din domeniul *cs.vu.nl*, nu se găsesc în acest fișier. Ele sunt furnizate de serverele rădăcină ale căror adrese IP sunt prezentate în fișierul de configurare a sistemului și sunt încărcate în memoria ascunsă DNS atunci când este pornit serverul DNS. Există cam o duzină de servere rădăcină în lume și fiecare știe adresele IP ale tuturor celorlalte servere de domenii de nivel superior. Astfel, dacă o mașină știe adresa IP a cel puțin unuia din serverele rădăcină, el poate căuta orice nume DNS.

### 7.1.3 Servere de nume

Teoretic, un singur server de nume poate conține întreaga bază de date DNS și poate să răspundă tuturor cererilor. În practică, acest server poate fi atât de încărcat, încât să devină de neutilizat. În afară de aceasta, dacă se defectează, va fi afectat întregul Internet.



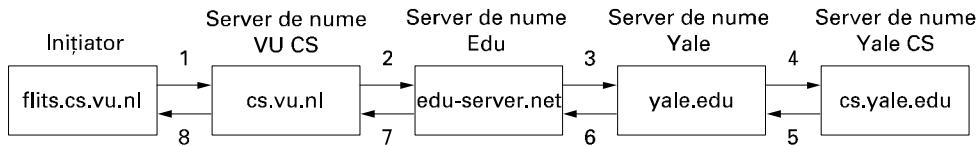
**Fig. 7-4.** O parte din spațiul numelor DNS prezentând împărțirea în zone.

Pentru a evita problemele asociate cu existența unei singure surse de informație, spațiul de nume DNS este împărțit în **zone** care nu se suprapun. O posibilă cale de împărțire a spațiului de nume din fig. 7-1 este arătată în fig. 7-4. Fiecare zonă conține câte o parte a arborelui precum și numele serverelor care păstrează informația autorizată despre acea zonă. În mod normal, o zonă va avea un server de nume primar, care preia informația dintr-un fișier de pe discul propriu și unul sau mai multe servere de nume secundare, care iau informațiile de pe serverul primar. Pentru a îmbunătăți fiabilitatea, unele servere pentru o zonă pot fi plasate în afara zonei.

Plasarea limitelor unei zone este la latitudinea administratorului ei. Această decizie este luată în mare parte bazându-se pe câte servere de nume sunt dorite și unde să fie plasate. De exemplu, în fig. 7-4, Yale are un server pentru *yale.edu* care administrează *eng.yale.edu*, dar nu și *cs.yale.edu*, care este o zonă separată cu propriile servere de nume. O astfel de decizie poate fi luată atunci când un departament ca cel de engleză nu dorește să aibă propriul server de nume, în schimb departamentul de calculatoare dorește. În consecință *cs.yale.edu* este o zonă separată, în timp ce zona *eng.yale.edu* nu este separată.

Atunci când un resolver are o cerere referitoare la un nume de domeniu, el transferă cererea unuia din serverele locale de nume. Dacă domeniul căutat este sub jurisdicția serverului de nume, cum ar fi *ai.cs.yale.edu*, care este sub *cs.yale.edu*, el reîntoarce înregistrări de resurse autorizate. O **înregistrare autorizată (authoritative record)** este cea care vine de la autoritatea care administrează înregistrarea și astfel este întotdeauna corectă. Înregistrările autorizate se deosebesc de înregistrările din memoria ascunsă, care pot fi expirate.

Dacă, totuși, domeniul se află la distanță, iar local nu este disponibilă nici o informație despre domeniul cerut, atunci serverul de nume trimite un mesaj de cerere la serverul de nume de pe primul nivel al domeniului solicitat. Pentru a clarifica acest proces să considerăm exemplul din fig. 7-5. Aici resolverul de pe *flits.cs.vu.nl* dorește să știe adresa IP a sistemului gazdă *linda.cs.yale.edu*. În pasul 1 trimite o cerere la serverul de nume local *cs.vu.nl*. Această cerere conține numele de domeniu căutat, tipul (A) și clasa (IN).



**Fig. 7-5.** Modul în care un resolver caută un nume la distanță, în opt pași.

Să presupunem că serverul local de nume nu a avut niciodată o cerere pentru acest domeniu și nu știe nimic despre el. Poate solicita informații de la câteva servere de nume din apropiere, dar dacă nici unul dintre ele nu știe, va trimite un pachet UDP la serverul pentru *edu* specificat în baza de date (vezi fig. 7-5), *edu-server.net*. Este puțin probabil ca acest server să cunoască adresa *linda.cs.yale.edu* și probabil nu cunoaște nici adresa *cs.yale.edu*, dar trebuie să cunoască adresele fililor din subarbore, astfel că va transmite cererea la serverul de nume *yale.edu* (pas 3). Acesta va transmite cererea mai departe către *cs.yale.edu* (pas 4), care trebuie să aibă înregistrările autorizate de resurse. Deoarece fiecare cerere este de la un client la un server, înregistrarea de resursă parcurge pașii 5 până la 8.

Odată ce aceste înregistrări de resurse ajung înapoi la serverul de nume *cs.vu.nl*, ele vor fi depuse în memoria ascunsă, pentru a fi folosite ulterior. Totuși, această informație nu este autorizată, deoarece orice schimbare făcută la *cs.yale.edu* nu se va propaga spre toate serverele care au folosit-o. Din acest motiv intrările în memoria ascunsă nu ar trebui să aibă viață prea lungă. Aceasta este motivul pentru care câmpul *Timp\_de\_viață* este inclus în fiecare înregistrare de resursă. El informează serverele de nume aflate la distanță cât timp să mențină înregistrările în memoria ascunsă. Dacă o anumită mașină are de ani de zile aceeași adresă IP, această informație ar putea fi păstrată timp de o zi. Pentru informații mai volatile este mai sigur ca înregistrările să fie eliminate după câteva secunde sau un minut.

De menționat că metoda de interogare descrisă aici este cunoscută ca metoda de **interrogare recursivă (recursive query)**, deoarece fiecare server care nu are informația cerută o caută în altă parte și raportează. Este posibil și o altă variantă. În acest caz, atunci când o cerere nu poate fi rezolvată local, cererea eșuează, dar este întors numele următorului server de pe calea ce trebuie încercată. Unele servere nu implementează interogarea recursivă și întorc întotdeauna numele următorului server la care să se încerce.

De asemenea merită menționat faptul că atunci când un client DNS nu reușește să primească un răspuns înainte de expirarea timpului de căutare, data viitoare va încerca un alt server. Se presupune că serverul este probabil nefuncțional, nu că cererea sau răspunsul s-au pierdut.

Deși DNS este foarte important pentru funcționarea corectă a Internetului, el nu face decât să pună în corespondență nume simbolice de mașini cu adresele lor IP. El nu ajută la localizarea oamenilor, resurselor, serviciilor sau obiectelor în general. Pentru localizarea acestora a fost definit un alt serviciu director, numit **LDAP (Lightweight Directory Access Protocol, rom.: Protocol de acces la cataloge simplificate)**. El este o versiune simplificată a serviciului de cataloge OSI X.500, descris în RFC 2251. El organizează informația sub formă de arbore și permite căutări pe diferite componente. Poate fi privit ca o carte de telefon obișnuită (de tipul „pagini albe”). Nu o să intrăm în amănunte referitoare la el în această carte, însă puteți găsi mai multe informații în (Weltman și Dahbura, 2000).

## 7.2 POȘTA ELECTRONICĂ

Poșta electronică, sau **e-mail**, cum este cunoscută de către numeroșii săi admiratori, există de peste două decenii. Înainte de 1990, era folosită în special în mediul academic. În timpul anilor 1990, a devenit cunoscută publicului larg și a crescut exponențial până la punctul în care numărul de mesaje electronice trimise pe zi este acum mult mai mare decât numărul de scrisori tradiționale (adică pe hârtie).

E-mail-ul, ca majoritatea celorlalte forme de comunicare, are convențiile și stilurile sale proprii. În particular, el este foarte neprotocolar și are un prag de folosire foarte scăzut. Oamenii care n-ar visa niciodată să sună la telefon sau chiar să scrie o scrisoare unei Persoane Foarte Importante nu ezită o secundă să trimită un e-mail neglijent.

E-mail-ul este plin de jargoane precum BTW (By The Way - aproape), ROTFL (Rolling On The Floor Laughing – a se tăvăli pe jos de râs) și IMHO (In My Humble Opinion - după umila mea părere). Mulți oameni folosesc în e-mail-urile lor câteva caractere ASCII numite **smileys** sau **emoticons** (față zâmbitoare și față tristă). Câteva din cele mai interesante sunt reproduse în fig. 7-6. Pentru cei mai mulți, rotirea cărtii cu 90 de grade în sensul acelor de ceasornic, le va face mai clare. Pentru o cărticică cu peste 650 smileys, vedeti (Sanderson și Dougherty, 1993).

Smiley	Semnificație	Smiley	Semnificație	Smiley	Semnificație
:)	Sunt fericit	=:-)	Abe Lincoln	:+)	Nas mare
:-(	Sunt trist, supărat	=):-)	Unchiul Sam	:-))	Gușă
:	Sunt apatic	*<:-)	Moș Crăciun	:-{}	Mustață
;)	Fac cu ochiul	<:-(	Dunce	#:-)	Păr încurcat
:(0)	Tip	(:-	Australian	8-)	Poartă ochelari
:(*)	Vomit	:-)X	Om cu papion	C:-)	Cap mare

**Fig. 7-6.** Câteva smileys. Nu vor fi în examenul final :-)

Primele sisteme de poștă electronică constau pur și simplu din protocoale de transfer de fișiere, cu convenția ca prima linie a fiecărui mesaj (adică fișier) să conțină adresa receptorului. Cu timpul, limitările acestei abordări au devenit din ce în ce mai evidente. O parte dintre neajunsuri erau:

1. Trimiterea unui mesaj către un grup de persoane era incomodă. Managerii au nevoie adesea de această facilitate pentru a trimite note și rapoarte tuturor subordonaților.
2. Mesajele nu aveau structură internă, făcând astfel dificilă prelucrarea lor cu ajutorul calculatorului. De exemplu, dacă un mesaj trimis mai departe era inclus în corpul altui mesaj, extagerea părții incluse din mesajul primit era dificilă.
3. Inițiatorul (transmițătorul) nu știa niciodată dacă mesajul a ajuns sau nu.
4. Dacă cineva avea în plan să plece în călătorie de afaceri pentru mai multe săptămâni și doar ca toată poșta primită în acest timp să fie preluată de către secretară, acest lucru nu era ușor de realizat.
5. Interfața cu utilizatorul era slab integrată cu sistemul de transmisie, cerând utilizatorilor ca întâi să editeze un fișier, apoi să părăsească editorul și să apeleze programul de transfer de fișiere.
6. Nu era posibilă transmiterea de mesaje care să conțină o combinație de text, desene, facsimile și voce.

Pe măsură ce s-a câștigat experiență, au fost propuse sisteme de poștă electronică mai complicate. În 1982 au fost publicate propunerile cu privire la e-mail ale ARPANET, sub numele de RFC 821 (protocolul de transmisie) și RFC 822 (formatul mesajelor). Revizii minore, RFC 2821 și RFC 2822, au devenit standarde Internet, totuși toată lumea se referă la e-mail gândindu-se la RFC 822.

În 1984, CCITT a emis recomandarea X.400. După două decenii de competiție, sistemele de poștă electronică bazate pe RFC 822 sunt larg răspândite, în timp ce acele bazate pe X.400 au disparețut. Modul în care un sistem încropit de o mână de absolvenți de știință calculatoarelor a învins un standard internațional oficial, puternic susținut de către toate PTT-urile din lumea întreagă, de multe guverne și de o parte substanțială a industriei calculatoarelor, ne aduce în minte povestea biblică a lui David și Goliat.

Motivul succesului lui RFC822 nu este dat de faptul că ar fi atât de bun, ci acela că X.400 a fost atât de slab proiectat și atât de complex, încât nimeni nu l-ar putea implementa bine. Având de ales între un sistem nesofisticat, dar care funcționează, cum este cel bazat pe RFC822 și sistemul de e-mail X.400, presupus cu adevărat minunat, dar nefuncțional, majoritatea organizațiilor l-au ales pe primul. Poate că este și o lecție în spatele acestei povești. De acea discuția noastră referitoare la e-mail se va concentra asupra sistemului de e-mail din Internet.

### 7.2.1 Arhitectură și servicii

În această secțiune vom furniza o prezentare de ansamblu a ceea ce pot face sistemele de poștă electronică și cum sunt ele organizate. Aceste sisteme constau de obicei din două subsisteme: **agenții-utilizator**, care permit utilizatorilor să citească și să trimită scrisori prin poștă electronică și **agenții de transfer de mesaje**, care transportă mesajele de la sursă la destinație. Agenții-utilizator sunt programe locale, care furnizează o metodă de a interacționa cu sistemul de e-mail bazată pe comenzi, meniu sau grafică. Agenții de transfer de mesaje sunt, de regulă, **demoni** de sistem, adică procese care se execută în fundal. Sarcina lor este să transfere mesajele prin sistem.

În general, sistemele de poștă electronică pun la dispoziție cinci funcții de bază. Să aruncăm o privire asupra lor.

**Componerea** se referă la procesul de creare a mesajelor și a răspunsurilor. Deși pentru corpul mesajului poate fi folosit orice editor de texte, sistemul însuși poate acorda asistență la adresare și la

completarea numeroaselor câmpuri antet atașate fiecărui mesaj. De exemplu, când se răspunde la un mesaj, sistemul poate extrage adresa inițiatorului din mesajul primit și o poate insera automat în locul potrivit din cadrul răspunsului.

**Transferul** se referă la deplasarea mesajului de la autor la receptor. În mare, aceasta necesită stabilirea unei conexiuni la destinație, sau la o mașină intermedieră, emiterea mesajului și eliberarea conexiunii. Sistemul de poștă ar trebui să facă acest lucru singur, fără a deranja utilizatorul.

**Raportarea** se referă la informarea inițiatorului despre ce s-a întâmplat cu mesajul. A fost livrat? A fost respins? A fost pierdut? Există numeroase aplicații în care confirmarea livrării este importantă și poate avea chiar semnificație juridică. („Știți, domnule judecător, sistemul meu de poștă electronică nu e foarte de încredere, aşa că presupun că citația electronică s-a pierdut pe undeva.”)

**Afișarea** mesajelor primite este necesară pentru ca utilizatorii să-și poată citi poșta. Uneori sunt necesare conversii sau trebuie apelat un program de vizualizare special; de exemplu, dacă mesajul este un fișier PostScript, sau voce digitizată. Se mai încercă uneori și conversii simple și formatări.

**Dispozitia** este pasul final și se referă la ceea ce face receptorul cu mesajul, după ce l-a primit. Posibilitățile includ eliminarea sa înainte de a-l citi, aruncarea sa după citire, salvarea sa ș.a.m.d. Ar trebui de asemenea să fie posibilă regăsirea și recitirea de mesaje deja salvate, trimiterea lor mai departe, sau procesarea lor în alte moduri.

În plus față de aceste servicii de bază, unele sisteme de e-mail, în special cele interne companiilor, dispun de o gamă variată de facilități avansate. Să menționăm pe scurt câteva dintre ele. Când utilizatorii se deplasează sau când sunt plecați pentru o perioadă de timp, pot dori ca poșta lor să fie trimisă acolo unde se găsesc, aşa că sistemul ar trebui să fie capabil să facă acest lucru automat.

Majoritatea sistemelor permit utilizatorilor să-și creeze **cutii poștale** (mailboxes) pentru a păstra mesajele sosite. Sunt necesare comenzi de creare și distrugere a cutiilor poștale, de inspectare a conținutului acestora, de inserare și de ștergere de mesaje din cutii poștale ș.a.m.d.

Managerii de companii au adesea nevoie să trimită un același mesaj fiecărui subordonat, client sau furnizor. Acest lucru dă naștere ideii de **listă de poștă** (mailing list), care este o listă de adrese de poștă electronică. Când un mesaj este trimis la lista de poștă, copii identice ale sale sunt expediate fiecăruiu dintre cei de pe listă.

Alte caracteristici evoluate sunt copii la indigo, poștă de prioritate mare, poștă secretă (criptată), receptori alternativi, dacă cel primar nu este disponibil, și posibilitatea de a permite secretarilor să se ocupe de poșta primită de șefii lor.

Poșta electronică este în prezent folosită pe scară largă în industrie, pentru comunicație în cadrul companiilor. Aceasta permite unor angajați, răspândiți la distanțe mari unii de ceilalți, chiar și peste mai multe fusuri orare, să coopereze la proiecte complexe. Eliminând majoritatea indiciilor cu privire la funcție, vârstă și gen, dezbatările prin poștă electronică tind să se concentreze asupra ideilor și nu a statutului din cadrul organizației. Prin poștă electronică, o idee scăpitoare a unui student la cursurile de vară poate avea un impact mai mare decât una stupidă, venită de la un vicepreședinte executiv.

O idee fundamentală în toate sistemele moderne de e-mail este distincția dintre **plic** și conținutul său. Plicul încapsulează mesajul. Conține toată informația necesară pentru transportul mesajului, cum ar fi destinația, adresa, prioritatea, nivelul de securitate, toate acestea fiind distincte de mesajul în sine. Agenții de transfer de mesaje folosesc plicul pentru rutare (dirijare), aşa cum face și oficiul poștal.

Mesajul din interiorul plicului conține două părți: **antetul** și **corpul**. Antetul conține informație de control pentru agenții utilizator. Corpul mesajului se adresează în întregime utilizatorului uman. Plicurile și mesajele sunt ilustrate în fig. 7-7.

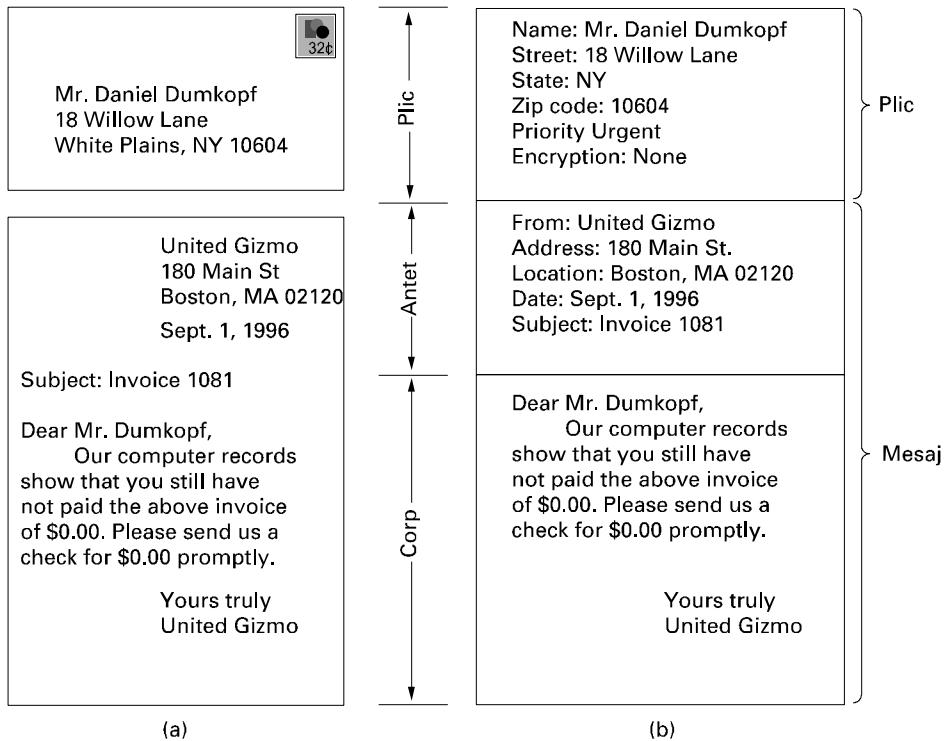


Fig. 7-7. Plicuri și mesaje. (a) poștă clasica (b) poștă electronică.

## 7.2.2 Agentul utilizator

Sistemele de poștă electronică au, aşa cum am văzut, două părți esențiale: agenții-utilizator și agenții de transfer de mesaje. În această secțiune ne vom uita la agenții utilizator. Un agent utilizator este de obicei un program (uneori numit cititor de poștă) care acceptă o varietate de comenzi pentru compunerea, primirea și răspunsul la mesaje, cât și pentru manipularea cutiilor poștale. Unii agenții utilizator au o interfață sofisticată, dirijată prin meniu sau icoane, care necesită un maus, în timp ce altele acceptă comenzi de către un caracter, date de la tastatură. Funcțional însă, toți aceștia sunt identici. Unele sisteme sunt dirigate prin meniu sau icoane sau au și alternative mai „scurte” pe tastatură.

### Trimiterea poștei electronice

Pentru a trimite un mesaj prin poșta electronică, un utilizator trebuie să furnizeze mesajul, adresa destinație, și eventual alți câțiva parametri. Mesajul poate fi produs cu un editor de texte de sine-sătător, cu un program de procesare de text sau, eventual, cu un editor de texte specializat, construit în interiorul agentului utilizator. Adresa de destinație trebuie să fie într-un format cu care agentul utilizator să poată lucra. Mulți agenții-utilizator solicită adrese de forma *utilizator@adresă-dns*. Deoarece aceste lucruri au fost studiate anterior în acest capitol, nu vom relua materialul respectiv aici.

Oricum, merită notat că există și alte forme de adresare. În particular, adresele X.400 arată radical diferit de cele DNS. Ele sunt compuse din perechi de forma *atribut = valoare*, separate de bare oblice. De exemplu:

/C=US/SP=MASSACHUSETTS/L=CAMBRIDGE/PA=360 MEMORIAL DR./CN=KEN SMITH/

Această adresă specifică o țară, un stat, o localitate, o adresă personală și un nume obișnuit (Ken Smith). Sunt posibile multe alte attribute, astfel încât poți trimite mesaje cuiva al cărui nume nu-l știi, atâtă timp cât știi suficiente alte attribute (de exemplu, compania și funcția). Cu toate că adresele X.400 sunt mult mai puțin convenabile decât cele DNS, cele mai multe sisteme de poștă electronică permit folosirea de **pseudonime (aliases**, uneori numite și porecle) pentru obținerea numelor sau adreselor de e-mail corecte ale unei persoane. În consecință, chiar și cu adresele de tip X.400, de obicei nu este necesară scrierea în întregime a celor săriuri ciudate.

Majoritatea sistemelor de e-mail acceptă liste de poștă, astfel că un utilizator poate trimite, cu o singură comandă, un același mesaj tuturor persoanelor dintr-o listă. Dacă lista de poștă este păstrată local, agentul-utilizator poate pur și simplu să trimită căte un mesaj separat fiecăruiu dintre receptoarii doriti. Dacă lista este păstrată la distanță, atunci mesajele vor fi expandate acolo. De exemplu, dacă un grup de admiratori de păsări au o listă de poștă numită *birders*, instalată la *meadowlark.arizona.edu*, atunci orice mesaj trimis la *birders@meadowlark.arizona.edu* va fi dirijat către Universitatea din Arizona și expandat acolo în mesaje individuale pentru toți membrii listei de poștă, oriunde ar fi ei în lume. Utilizatorii acestei liste de poștă nu pot determina că aceasta este o listă de adrese. Ar putea fi la fel de bine cutia poștală personală a Prof. Gabriel O. Birders.

### Citirea poștei electronice

În mod obișnuit, când este lansat un agent-utilizator, înainte de a afișa ceva pe ecran, el se va ui-ta în cutia poștală a utilizatorului după mesajele care sosesc. Apoi poate anunța numărul de mesaje din cutie, sau poate afișa pentru fiecare mesaj căte un rezumat de o linie, pentru ca apoi să aștepte o comandă.

Ca exemplu despre cum lucrează un agent-utilizator, să aruncăm o privire asupra unui scenariu tipic pentru poștă electronică. După lansarea agentului-utilizator, utilizatorul cere un rezumat al mesajelor sale. O imagine ca aceea din fig. 7-8 apare în acest caz pe ecran. Fiecare linie se referă la căte un mesaj. În acest exemplu, cutia poștală conține opt mesaje.

#	Marcaje	Octetii	Transmitător	Subiect
1	K	1030	Asw	Changes to MINIX
2	KA	6348	Trudy	Not all Trudys are nasty
3	K F	4519	Amy N. Wong	Request for information
4		1236	Bal	Bioinformatics
5		103610	Kaashoek	Material on peer-to-peer
6		1223	Frank	Re: Will you review a grant proposal
7		3110	Guido	Our paper has been accepted
8		1204	Dmr	Re: My student's visit

Fig. 7-8. Un exemplu de afișare a conținutului unei cutii poștale.

Fiecare linie de afișaj conține câteva câmpuri extrase de pe plicul sau din antetul mesajului corespunzător. Într-un sistem simplu de poștă electronică, alegerea câmpurilor afișate este făcută în cadrul programului. Într-un sistem mai sofisticat, utilizatorul poate specifica ce câmpuri să fie afișate, furnizând un **profil al utilizatorului**, adică un fișier care descrie formatul de afișare. În exemplul

considerat, primul câmp reprezintă numărul mesajului. Al doilea câmp, *Marcaje*, poate conține un *K*, însemnând că mesajul nu este nou, dar a fost citit anterior și păstrat în cutia poștală; un *A*, însemnând că deja s-a răspuns la acest mesaj; și/sau un *F*, însemnând că mesajul a fost trimis mai departe altcuiva. Sunt de asemenea posibile și alte marcaje.

Al treilea câmp specifică lungimea mesajului și al patrulea spune cine a trimis mesajul. Din moment ce el este pur și simplu extras din mesaj, acest câmp poate conține prenume, nume complete, inițiale, nume de cont, sau orice altceva și-a ales transmițătorul să pună. În sfârșit, câmpul *Subiect* specifică despre ce este mesajul, într-un scurt rezumat. Persoanele care omit să includă un câmp *Subiect* adesea descoperă că răspunsurile la scrisorile lor tind să nu obțină prioritate maximă.

După ce au fost afișate antetele, utilizatorul poate executa oricare dintre comenzi disponibile, ca de exemplu afișarea unui mesaj, ștergerea unui mesaj și asta mai departe. Sistemele mai vechi lucrau în mod text și de obicei foloseau comenzi de un caracter pentru diversele operații, ca de exemplu T (scrie mesaj), A (răspunde la mesaj), D (șterge mesaj) și F (trimită mai departe). Un argument specifică mesajul corespondent. Sistemele mai recente folosesc interfețe grafice. De obicei, utilizatorul selectează un mesaj cu mouse-ul și apoi apasă pe o iconă pentru a scrie, răspunde la mesaj, sau pentru a-l trimite mai departe.

Poșta electronică a parcurs un drum lung de pe vremea când era doar transfer de fișiere. Agentii-utilizator sofisticăți fac posibilă manevrarea unui volum mare de scrisori. Pentru persoane care primesc și trimit mii de mesaje pe an, asemenea instrumente sunt neprețuite.

### 7.2.3 Formatele mesajelor

Să ne întoarcem acum de la interfața utilizator la formatul mesajelor de poștă electronică în sine. Mai întâi ne vom uita la e-mailul ASCII de bază, care utilizează RFC 822. După aceea ne vom concentra asupra extensiilor multimedia ale RFC 822.

#### RFC 822

Mesajele constau dintr-un plic simplu (descriși în RFC 821), un număr de câmpuri antet, o linie goală și apoi corpul mesajului. Fiecare câmp antet se compune (din punct de vedere logic) dintr-o singură linie de text ASCII, conținând numele câmpului, două puncte, și, pentru majoritatea câmpurilor, o valoare. RFC 822 a fost creat acum două decenii și nu distinge clar plicul de câmpurile antet, cum ar face un standard nou. Cu toate că a fost corectat în RFC 2822, o refacere completă n-a fost posibilă datorită răspândirii sale largi. La o utilizare normală, agentul-utilizator construiește un mesaj și îl transmite agentului de transfer de mesaje, care apoi folosește unele dintre câmpurile antet pentru a construi plicul efectiv, o combinație oarecum demodată de mesaj și plic.

Principalele câmpuri antet, legate de transportul de mesaje, sunt înfățișate în fig. 7-9. Câmpul *To:* oferă adresa DNS a receptorului primar. Este permisă de asemenea existența de receptori mulți. Câmpul *Cc:* dă adresa oricărui receptor secundar. În termenii livrării, nu este nici o diferență între un receptor primar și unul secundar. Este într-regime o deosebire psihologică, ce poate fi importantă pentru persoanele implicate, dar este neimportantă pentru sistemul de poștă. Termenul *Cc:* (Carbon copy - copie la indigo) este puțin depășit, din moment ce calculatoarele nu folosesc indigo, dar este bine înrădăcinat. Câmpul *Bcc:* (Blind carbon copy - copie confidențială la indigo) este la fel ca *Cc:*, cu excepția că această linie este ștersă din toate copiile trimise la receptorii primari și secundari. Acest element permite utilizatorilor să trimită copii unei a treia categorii de receptori, fără ca cei primari și secundari să știe acest lucru.

Antet	Conținut
To:	Adresa(ele) de e-mail a(le) receptorului(iilor) primar(i)
Cc:	Adresa(ele) de e-mail a(le) receptorului(iilor) secundar(i)
Bcc:	Adresa(ele) de e-mail pentru „blind carbon copy”
From:	Persoana sau persoanele care au creat mesajul
Sender:	Adresa de e-mail a transmițătorului curent
Received:	Linie adăugată de fiecare agent de transfer de-a lungul traseului
Return-Path:	Poate fi folosită pentru a identifica o cale de întoarcere la transmițător

**Fig. 7-9.** Câmpurile antet ale lui RFC 822, legate de transportul de mesaje.

Următoarele două câmpuri, *From:* și *Sender:*, precizează cine a scris și respectiv cine a trimis mesajul. Acestea pot să nu fie identice. De exemplu, se poate ca o directoare executivă să scrie un mesaj, dar ca secretara ei să fie cea care îl trimite efectiv. În acest caz, directoarea executivă va fi afișată în câmpul *From:* și secretara în câmpul *Sender:*. Câmpul *From:* este obligatoriu, dar câmpul *Sender* poate fi omis dacă este identic cu *From:*. Aceste câmpuri sunt necesare în cazul în care mesajul nu poate fi livrat și trebuie returnat transmițătorului.

O linie conținând *Received:* este adăugată de fiecare agent de transfer de mesaje de pe traseu. Linia conține identitatea agentului, data și momentul de timp la care a fost primit mesajul și alte informații care pot fi utilizate pentru găsirea defectiunilor în sistemul de dirijare.

Câmpul *Return-Path:* este adăugat de agentul final de transfer de mesaje și are în intenție să indice cum se ajunge înapoi la transmițător. În teorie, această informație poate fi adunată din toate antetele *Received:* (cu excepția numelui cutiei poștale a transmițătorului), dar rareori este completată așa și de obicei conține chiar adresa transmițătorului.

Antet	Conținut
Date:	Data și momentul de timp la care a fost trimis mesajul
Reply-To:	Adresa de e-mail la care ar trebui trimise răspunsurile
Message-Id:	Număr unic, utilizat ulterior ca referință pentru acest mesaj (identificator)
In-Reply-To:	Identificatorul mesajului al căruia răspuns este mesajul curent
References:	Alți identificatori de mesaje relevanți
Keywords:	Cuvinte cheie alese de utilizator
Subject:	Scurt cuprins al mesajului, afișabil pe o singură linie

**Fig. 7-10.** Câteva câmpuri utilizate în antetul lui RFC 822.

În plus față de câmpurile din fig. 7-9, mesajele RFC 822 pot conține de asemenea o varietate de câmpuri antet, folosite de agenții-utilizator sau de receptorii umani. Cele mai des întâlnite dintre ele sunt prezentate în fig. 7-10. Majoritatea lor se explică de la sine, deci nu vom intra în detaliu la toate.

Câmpul *Reply-To:* este uneori utilizat când nici persoana care a compus mesajul, nici cea care l-a trimis nu vrea să vadă răspunsul. De exemplu, un director de marketing scrie un mesaj prin e-mail pentru a spune clienților despre un nou produs. Mesajul este trimis de o secretară, dar câmpul *Reply-To:* conține șeful departamentului de vânzări, care poate răspunde la întrebări și primi comenzi. Acest câmp este foarte folositor când transmițătorul are două conturi de e-mail și vrea ca răspunsul să ajungă în celălalt.

Documentul RFC 822 afirmă explicit că utilizatorilor le este permis să inventeze noi antete, atât timp cât acestea încep cu sirul de caractere X-. Se garantează că nici o extindere ulterioară nu va folosi nume ce încep cu X-, pentru a evita conflictele (suprapunerile) dintre antetele oficiale și cele

personale. Uneori studenții care fac pe deșteptii includ câmpuri de tipul *X-Fruit-of-the-Day*: sau *X-Disease-of-the-Week*; care sunt legale, deși nu întotdeauna clarificate.

După antete urmează corpul mesajului. Aici utilizatorii pot pune orice vor. Unii oameni își încheie mesajele cu semnături elaborate, incluzând caricaturi ASCII simple, citate din personalități mai mari sau mai mici, declarații politice și declinări de tot felul (de exemplu: Corporația XYZ nu este răspunzătoare pentru părerile mele; de fapt nu poate nici să le înțeleagă).

### **MIME - Multipurpose Internet Mail Extensions (extensii de poștă cu scop multiplu)**

La începuturile ARPANET, poșta electronică consta exclusiv din mesaje de tip text, scrise în engleză și exprimate în ASCII. Pentru acest context, RFC 822 realizează sarcina completă: specifică antetele, dar lăsa conținutul în întregime în seama utilizatorilor. În zilele noastre, această abordare nu mai este adekvată pentru Internetul care se întinde în lumea întreagă. Problemele includ transmisia și receptia de:

1. Mesaje în limbi cu accente (de exemplu franceza și germană).
2. Mesaje în alfabeze ne-latine (de exemplu ebraică și rusă).
3. Mesaje în limbi fără alfabet (de exemplu chineză și japoneză).
4. Mesaje care nu conțin text deloc (de exemplu audio și video).

O soluție posibilă a fost propusă în RFC 1341 și actualizată în RFC-urile 2045-2049. Această soluție, numită **MIME (Multipurpose Internet Mail Extensions)**, este în prezent larg utilizată. O vom descrie în continuare. Pentru informații suplimentare în legătură cu MIME, vedeti RFC-urile.

Idee fundamentală a MIME este să continue să folosească formatul RFC 822, dar să adauge structură corpului mesajului și să definească reguli de codificare pentru mesajele non-ASCII. Deoarece respectă RFC 822, mesajele MIME pot fi trimise utilizând programele și protocoalele de poștă existente. Tot ceea ce trebuie modificat sunt programele de transmitere și receptie, pe care utilizatorii le pot face ei însăși.

Antet	Conținut
MIME-Version:	Identifică versiunea de MIME
Content-Description:	Șir adresat utilizatorului care spune ce este în mesaj
Content-Id:	Identifier unic
Content-Transfer-Encoding:	Cum este împachetat corpul pentru transmisie
Content-Type:	Natura mesajului

**Fig. 7-11.** Antetele RFC 822 adăugate de către MIME.

MIME definește cinci noi antete de mesaje, așa cum se arată în fig. 7-11. Primul dintre acestea specifică pur și simplu agentului-utilizator care primește mesajul că este vorba de un mesaj MIME și ce versiune de MIME utilizează. Orice mesaj care nu conține un antet *MIME-Version*: este presupus ca fiind un mesaj în text pur, în engleză, și este procesat ca atare.

Antetul *Content-Description*: este un șir de caractere ASCII specificând ce este în mesaj. Acest antet este necesar pentru ca receptorul să știe dacă merită să decodifice și să citească mesajul. Dacă șirul de caractere spune: “Fotografia hamsterului Barbarei” și persoana care primește mesajul nu este un mare iubitor de hamsteri, mesajul va fi probabil mai curând aruncat, decât decodificat într-o fotografie color de înaltă rezoluție.

Antetul *Content-Id*: identifică conținutul. Utilizează același format ca antetul standard *Message-Id*.

Antetul *Content-Transfer-Encoding*: arată cum este împachetat pentru transmisie corpul mesajului, într-o rețea care poate ridica obiecții la majoritatea caracterelor diferite de litere, cifre și semne de punctuație. Sunt furnizate cinci scheme (plus o evadare către noi scheme). Cea mai simplă schemă se referă chiar la text ASCII. Caracterele ASCII utilizează 7 biți și pot fi transportate direct prin protocolul de e-mail, atât timp cât nici o linie nu are mai mult de 1000 de caractere.

Următoarea schemă ca simplitate este cam același lucru, dar utilizează caractere de câte 8 biți, reprezentând toate valorile de la 0 la 255 inclusiv. Această schemă de codificare încalcă protocolul (original) de e-mail utilizat în Internet, dar este folosită de unele părți ale Internetului, care implementează niște extensii ale protocolului original. În timp ce declararea codificării nu o face să devină legală, faptul că o avem explicit poate cel puțin să lămurească lucrurile atunci când ceva merge prost. Mesajele utilizând codificarea de 8 biți trebuie încă să respecte lungimea maximă a liniei, care este standard.

Este chiar mai rău în cazul mesajelor care utilizează codificare binară. Aceste mesaje sunt fișiere binare arbitrară, care nu numai că utilizează toți cei 8 biți, dar nu respectă nici limita de linie de 1000 de caractere. Programele executabile intră în această categorie. Nu se acordă nici o garanție că mesajele binare vor ajunge corect, dar mulți le trimit oricum.

Modalitatea corectă de a codifica mesaje binare este de a utiliza **codificarea în bază 64**, numită uneori **armură ASCII**. În această schemă, grupuri de câte 24 de biți sunt împărțite în patru unități de câte 6 biți, fiecare dintre aceste unități fiind transmisă ca un caracter ASCII legal. Codificarea este „A” pentru 0, „B” pentru 1, s.a.m.d., urmate de cele 26 de litere mici, cele 10 cifre, și în cele din urmă + și / pentru 62 și respectiv 63. Secvențele == și = sunt utilizate pentru a arăta că ultimul grup a conținut doar 8 sau respectiv 16 biți. Se ignoră secvențele carriage return și line feed, astfel că ele pot fi inserate după dorință, pentru a păstra liniile suficient de scurte. Utilizând această schemă pot fi trimise sigur texte binare arbitrară.

Pentru mesajele care sunt aproape în întregime ASCII și conțin puține caractere ne-ASCII, codificarea în bază 64 este oarecum ineficientă. În locul acesta se utilizează o codificare numită **quoted-printable-encoding** (codificare afișabilă marcată). Aceasta este o codificare de tip ASCII pe 7 biți, având toate caracterele cu cod mai mare de 127 codificate sub forma unui semn egal urmat de valoarea caracterului reprezentată prin două cifre hexazecimale.

Rezumând, datele binare ar trebui trimise codificate în bază 64 sau sub formă quoted-printable. Când există motive întemeiate pentru a nu utiliza una dintre aceste scheme, este posibil să se specifice în antetul Content-Transfer-Encoding: o codificare definită de către utilizator.

Ultimul antet înfățișat în fig. 7-11 este cu adevărat cel mai interesant. El specifică natura corpului mesajului. În RFC 2045 sunt definite șapte tipuri, fiecare având unul sau mai multe subtipuri. Tipul și subtipul sunt separate printr-o bară oblică (slash), ca în:

Content-Type: video/mpeg

Subtipul trebuie precizat explicit în antet; nu sunt furnizate valori implicate. Lista inițială de tipuri și subtipuri specificate în RFC 2045 este prezentată în fig. 7-12. De atunci au fost adăugate multe altele, introducându-se întrări adiționale de fiecare dată când a devenit necesar.

Să parcurgem acum lista tipurilor. Tipul *text* este utilizat pentru text simplu. Combinarea *text/plain* este folosită pentru mesaje obișnuite care pot fi afișate de îndată ce sunt primite, fără codificare sau procesare ulterioară. Această opțiune permite ca mesajele obișnuite să fie transportate în MIME adăugând doar câteva antete suplimentare.

Tip	Subtip	Descriere
Text	Plain	Text neformatat
	Enriched	Text incluzând comenzi simple de formatare
Image	Gif	Imagini fixe în format GIF
	Jpeg	Imagini fixe în format JPEG
Audio	Basic	Sunet
Video	Mpeg	Film în format MPEG
Application	Octet-stream	Secvență neinterpretată de octeți
	Postscript	Un document afișabil în PostScript
Message	Rfc822	Un mesaj MIME RFC 822
	Partial	Mesajul a fost fragmentat pentru transmisie
	External-body	Mesajul în sine trebuie adus din rețea
Multipart	Mixed	Părți independente în ordine specificată
	Alternative	Același mesaj în formate diferite
	Parallel	Părțile trebuie vizualizate simultan
	Digest	Fiecare parte este un mesaj RFC 822 complet

**Fig. 7-12.** Tipurile și subtipurile aparținând MIME definite în RFC 2045.

Subtipul *text/enriched* permite includerea în text a unui limbaj simplu de marcăre. Acest limbaj furnizează o modalitate independentă de sistem pentru a exprima scrierea cu caractere aldine sau cursive, dimensiunile, alinierea, distanțele dintre rânduri, folosirea de indici superiori sau inferiori și paginarea simplă. Limbajul de marcăre se bazează pe SGML, Standard Generalized Markup Language (limbajul standard generalizat de marcăre), folosit de asemenea ca bază pentru HTML, utilizat în World Wide Web. De exemplu mesajul

The **time** has come the **walrus** said ...

ar fi afișat sub forma:

The **time** has come the **walrus** said...

Depinde de sistemul receptor să aleagă interpretarea potrivită. Dacă sunt disponibile caractere aldine și cursive, acestea vor putea fi folosite; altfel, pentru a scoate în evidență se pot utiliza culori, scriere cu clipire sau video-invers etc. Sisteme diferite pot face alegeri diferite.

Când Web-ul a devenit popular, a fost adăugat un nou subtip, *text/html* (în RFC 2854) pentru a permite paginilor Web să fie trimise într-un e-mail de tip RFC 822. Un subtip pentru sistemul extins de marcăre, *text/xml*, este definit în RFC 3023. Vom studia HTML și XML mai târziu în acest capitol.

Următorul tip MIME este *image*, utilizat pentru trimitera de imagini fixe. În zilele noastre sunt utilizate multe formate, atât cu, cât și fără compresie, pentru a păstra și transmite imagini. Două dintre acestea, GIF și JPEG, sunt recunoscute de aproape toate programele de navigare, dar există și altele care au fost adăugate la lista originală.

Tipurile *video* și *audio* sunt pentru imagini în mișcare și respectiv pentru imagini cărora li se asociază și sunet. Trebuie notat că *video* include doar informația video, nu și coloana sonoră. Dacă trebuie transmis un film cu sunet, s-ar putea ca porțiunile audio și video să trebuiască să fie transmise separat, depinzând de sistemul de codificare utilizat. Primul format video definit a fost cel inventat de cei ce se intitulează modest Moving Picture Experts Group (MPEG - Grupul de experți în imagini în mișcare), dar de atunci au fost adăugate și altele. În plus față de *audio/basic*, un nou tip audio, *audio/mpeg* a fost adăugat în RFC 3003 pentru a permite oamenilor să transmită fisiere MP3 prin e-mail.

Tipul *application* este utilizat ca un colector pentru formatele care necesită prelucrare externă, neidentificate de nici unul dintre celelalte tipuri. Un *octet-stream* este doar o secvență de octeți nein-

terpretați. La primirea unui asemenea flux, un agent-utilizator ar trebui probabil să-l afișeze, sugerându-i utilizatorului să-l copieze într-un fișier și cerându-i un nume pentru acesta. Procesarea ulterioră este apoi la latitudinea utilizatorului.

Celălalt subtip definit este *postscript*, care se referă la limbajul PostScript, produs de Adobe Systems și larg utilizat pentru descrierea paginilor imprimante. Multe imprimante au înglobate interpretoare PostScript. Deși un agent-utilizator poate pur și simplu să apeleze un interpretor PostScript extern pentru a interpreta fișierele PostScript primite, acest lucru nu este lipsit de pericole. PostScript este un întreg limbaj de programare. Dându-i-se destul timp, o persoană suficient de masochistă ar putea scrie în PostScript un compilator de C, sau un sistem de management de baze de date. Afisarea unui mesaj primit în format PostScript se face executând programul PostScript conținut de acesta. Pe lângă afisarea unui text, acest program poate citi, modifica, sau șterge fișierele utilizatorului și poate avea și alte efecte laterale neplăcute.

Tipul *message* permite încapsularea în întregime a unui mesaj în altul. Această schemă este utilă, de exemplu pentru trimiterea mai departe a e-mailului, cu *forward*. Când un mesaj RFC 822 complet este încapsulat într-un mesaj exterior, ar trebui utilizat subtipul *rfc822*.

Subtipul *partial* face posibilă împărțirea unui mesaj încapsulat în bucăți de mesaj și trimiterea separată a acestora (de exemplu, dacă mesajul încapsulat este prea lung). Parametrii fac posibilă reasamblarea în ordinea corectă a tuturor părților, la destinație.

Și în sfârșit, subtipul *external-body* poate fi utilizat pentru mesaje foarte lungi (de exemplu, filme video). În loc de a include fișierul MPEG în mesaj, se dă o adresă FTP și agentul utilizator al receptorului poate aduce din rețea în momentul în care este necesar. Această facilitate este în special utilă când se trimit un film la o întreagă listă de poștă și se presupune că doar câțiva dintre membrii acesteia îl vor vedea (gândiți-vă la e-mailurile inutile, conținând reclame video).

```
From: elinor@abcd.com
To: carolyn@xyz.com
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@abcd.com>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Pământul înconjoară soarele de un număr întreg de ori

Acesta este preambulul. Agentul utilizator îl ignoră. O zi bună.

--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/richtext

Happy birthday to you
Happy birthday to you
Happy birthday dear <bold> Carolyn </bold>
Happy birthday to you

--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
access-type=„anon-ftp”;
site=„bicycle.abcd.com”;
directory=„pub”;
name=„birthday.snd”;
content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm
```

**Fig. 7-13.** Un mesaj multipart conținând alternative de tip text formatat și audio.

Ultimul tip este *multipart*, care permite unui mesaj să conțină mai multe părți, începutul și sfârșitul fiecărei părți fiind clar delimitat. Subtipul *mixed* permite fiecărei părți să fie diferită de celelalte, fără a avea o structură adițională impusă. Multe programe de e-mail permit utilizatorului să aibă una sau mai multe părți atașate la un mesaj text. Acestea sunt trimise folosind tipul *multipart*.

În contrast cu tipul *multipart*, subtipul *alternative* permite ca fiecare parte să conțină același mesaj, dar exprimat într-un alt mediu sau într-o codificare diferită. De exemplu, un mesaj ar putea fi trimis ca ASCII simplu, ca text formatat și ca PostScript. Un agent-utilizator proiectat corespunzător, la primirea unui asemenea mesaj, îl va afișa, dacă va fi posibil, în PostScript. A doua alegere va fi textul formatat. Dacă nici una dintre aceste alternative nu ar fi posibilă, s-ar afișa text ASCII obișnuit. Părțile ar trebui ordonate de la cea mai simplă, la cea mai complexă, pentru a ajuta receptorii care folosesc agenți-utilizator pre-MIME să înțeleagă mesajul (chiar și un utilizator pre-MIME poate citi text ASCII simplu). Subtipul *alternative* poate fi folosit de asemenea pentru limbaje multiple. În acest context, Rosetta Stone poate fi privit ca precursor al mesajului de tip *multipart/alternative*.

Un exemplu multimedia este prezentat în fig. 7-13. Aici, o felicitare este transmisă atât sub formă de text cât și sub formă de cântec. Dacă receptorul are facilități audio, agentul utilizator va aduce fișierul de sunet, *birthday.snd* și îl va interpreta. Dacă nu, versurile vor fi afișate pe ecran într-o liniște de mormânt. Părțile sunt delimitate de două cratime următe de sirul (definit de utilizator) specificat în parametrul *boundary*.

Observați că antetul *Content-Type* apare în trei poziții în acest exemplu. La primul nivel indică faptul că mesajul are mai multe părți. În cadrul fiecărei părți specifică tipul și subtipul acesteia. În sfârșit, în corpul celei de-a doua părți, este necesar pentru a indica agentului utilizator ce fel de fișier extern trebuie să aducă. Pentru a exprima ușoara diferență de utilizare, s-au folosit litere mici, deși toate antetele sunt *case insensitive* (nu fac diferență între literelor mari și cele mici). Antetul *content-transfer-encoding* este în mod similar necesar pentru orice corp extern care nu este codificat ca ASCII pe 7 biți.

Întorcându-ne la subtipurile corespunzătoare mesajelor *multipart*, vom spune că mai există două posibilități. Subtipul *parallel* este utilizat când toate părțile trebuie să fie interpretate simultan. De exemplu, adesea filmele au un canal audio și unul video. Ele sunt mai de efect dacă aceste două canale sunt interpretate în paralel și nu consecutiv.

În sfârșit, subtipul *digest* este utilizat când multe mesaje sunt împachetate împreună, într-unul compus. De exemplu, niște grupuri de dialog de pe Internet pot aduna mesaje de la abonații lor și apoi să le trimită în afară ca un singur mesaj de tip *multipart/digest*.

#### 7.2.4 Transferul mesajelor

Sistemul de transfer de mesaje se ocupă cu transmiterea mesajelor de la expeditor la receptor. Cea mai simplă cale de a realiza acest lucru constă în stabilirea unei conexiuni de transport de la mașina sursă la cea de destinație și apoi, pur și simplu în trimiterea mesajului. După ce examinăm cum se face acest lucru în mod normal, vom studia câteva situații în care metoda nu funcționează și vom vedea ce trebuie făcut în aceste cazuri.

#### SMTP – Simple Mail Transfer Protocol (Protocol simplu de transfer de poștă)

În cadrul Internetului poșta electronică este livrată prin stabilirea de către mașina sursă a unei conexiuni TCP la portul 25 al mașinii de destinație. La acest port se află un demon de e-mail care știe **SMTP (Simple Mail Transfer Protocol)**. Acest demon acceptă conexiunile și copiază mesajele

de la ele în cutiile poștale corespunzătoare. Dacă mesajul nu poate fi livrat, se returnează transmițătorului un raport de eroare conținând prima parte a mesajului nelivrat.

SMTP este un protocol simplu de tip ASCII. După stabilirea conexiunii TCP la portul 25, mașina transmițătoare, operând în calitate de client, așteaptă ca mașina receptoare, operând ca server, să vorbească prima. Serverul începe prin a trimite o linie de text, declarându-și identitatea și spunând dacă este pregătit sau nu să primească mesaje. Dacă nu este, clienții eliberează conexiunea și încearcă din nou mai târziu.

```
S: 220 xyz.com SMTP service ready
C: HELO abcd.com
    S: 250 xyz.com says hello to abcd.com
C: MAIL FROM: <elinor@abcd.com>
    S: 250 sender ok
C: RCPT TO: <carolyn@xyz.com>
    S: 250 recipient ok
C: DATA
    S: 354 Trimite mail; terminat cu "." pe linie nouă
C: From: elinor@abcd.com
C: To: carolyn@xyz.com
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@abcd.com>
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: Pământul înconjoară soarele de un număr întreg de ori
C:
C: Aceasta este preambulul. Agentul utilizator îl ignoră. O zi bună.
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: text/enriched
C:
C: Happy birthday to you
C: Happy birthday to you
C: Happy birthday dear <bold> Carolyn </bold>
C: Happy birthday to you
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
C: access-type="anon-ftp";
C: site="bicycle.abcd.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
```

**Fig. 7-14.** Transferul unui mesaj de la *elinor@abcd.com* la *carolyn@xyz.com*.

Dacă serverul este dispus să primească e-mail, clientul anunță de la cine vine scrisoarea și cui îi este adresată. Dacă un asemenea receptor există la destinație, serverul îi acordă clientului permisiunea să trimită mesajul. Apoi clientul trimite mesajul și serverul îl confirmă. În general nu este necesară atașarea unei sume de control deoarece TCP furnizează un flux sigur de octeți. Dacă mai există și alte mesaje, acestea sunt trimise tot acum. Când schimbul de mesaje, în ambele direcții, s-a încheiat, conexiunea este eliberată. În fig. 7-14 este prezentată o moștră de dialog referitoare la trimiterea mesajului din fig. 7-13, inclusiv codurile numerice utilizate de SMTP. Liniile trimise de client sunt marcate cu C:, iar cele trimise de server cu S:.

Câteva comentarii în legătură cu fig. 7-14 ar putea fi utile. Prima comandă a clientului este într-adevăr *HELO*. Din posibilele abrevieri de patru caractere ale cuvântului *HELLO*, aceasta are numeroase avantaje față de concurența sa cea mai mare. Motivul pentru care toate comenziile trebuiau să aibă patru caractere s-a pierdut în negura vremii.

În Fig.7-14, mesajul este trimis la un singur receptor și de aceea este utilizată o singură comandă *RCPT*. Mai multe asemenea comenzi sunt permise pentru a trimite un singur mesaj mai multor receptori. Fiecare dintre ele este confirmată sau rejetată individual. Chiar dacă unii dintre receptori sunt rejeptați (deoarece ei nu există la destinație), mesajul poate fi trimis celor rămași.

În sfârșit, deși sintaxa comenziilor de patru caractere de la client este rigid specificată, sintaxa replicilor este mai puțin rigidă. Doar codul numeric conținează cu adevărat. Fiecare implementare poate pune după cod ce siruri de caractere vrea.

Pentru a înțelege mai bine cum funcționează SMTP și câteva din celelalte protocoale descrise în acest capitol, încercați-le! În orice caz, mai întâi mergeți la o mașină conectată la Internet. Într-un sistem UNIX introduceți comanda:

```
telnet mail.isp.com 25
```

înlocuind numele DNS cu numele serverului de mail al ISP-ului dvs. Pe un sistem Windows, faceți clic pe Start, apoi Run, apoi tastează comanda în căsuța de dialog. Această comandă va stabili o conexiune Telnet (adică TCP) pe portul 25 pe mașina respectivă. Portul 25 este portul SMTP (vezi Fig. 6-27 pentru câteva porturi uzuale). Probabil o să primiți un răspuns de genul:

```
Trying 192.30.200.66...
Connected to mail.isp.com
Escape character is '^>'.
220 mail.isp.com Smail #74 ready at Thu, 25 Sept 2002 13:26 +0200
```

Primele trei linii spun ce face telnet-ul. Ultima linie este de la serverul SMTP de pe mașina de la distanță, anunțând disponibilitatea acestuia de a vorbi cu dvs. și de a accepta e-mail. Pentru a vedea ce comenzi acceptă, tastează:

```
HELP
```

De aici înainte, este posibilă o secvență de comenzi ca cea din fig. 7-14, începând cu comanda *HELO* dată de client.

E bine de notat faptul că folosirea liniilor de text ASCII pentru comenzi nu e un accident. Cele mai multe protocoale Internet funcționează așa. Folosirea textului ASCII face ca protocoalele foarte ușor de testat și depanat. Ele pot fi testate trimițând manual comenzi, cum am văzut mai sus, pentru care copiile mesajelor (eng.: dumps) sunt ușor de citit.

Chiar dacă protocolul SMTP este bine definit, mai pot apărea câteva probleme. O problemă este legată de lungimea mesajelor. Unele implementări mai vechi nu pot să lucreze cu mesaje mai mari de 64KB. O altă problemă se referă la expirări de timp (timeout). Dacă acestea diferă pentru server și client, unul din ei poate renunța, în timp ce celălalt este încă ocupat, întrerupând conexiunea în mod neașteptat. În sfârșit, în unele situații, pot fi lansate schimburi infinite de mesaje. De exemplu, dacă mașina 1 păstrează lista de poștă A și mașina 2 lista de poștă B și fiecare listă conține o intrare pentru cealaltă, atunci orice mesaj trimis oricăreia dintre cele două liste va genera o cantitate nesfârșită de trafic de e-mail.

Pentru a atinge câteva dintre aceste probleme, în RFC 2821 s-a definit protocolul SMTP extins (**ESMTP**). Clientii care doresc să-l utilizeze trebuie să trimită inițial un mesaj *EHLO* în loc de *HELO*. Dacă acesta este rejetat, atunci serverul este unul standard de tip SMTP și clientul va trebui să se comporte în modul obișnuit. Dacă *EHLO* este acceptat, înseamnă ca sunt permise noile comenzi și noii parametri.

### 7.2.5 Livrarea finală

Până acum, am presupus că toți utilizatorii lucrează pe mașini capabile să trimită și să primească e-mail. După cum am văzut, e-mail-ul este livrat prin stabilirea unei conexiuni TCP între expeditor și destinatar și apoi prin trimiterea e-mail-ului prin ea. Acest model a funcționat bine zeci de ani, atât timp cât toate calculatoarele din ARPANET (și mai târziu din Internet) erau, de fapt, conectate la rețea și gata să accepte conexiuni TCP.

Totuși, odată cu apariția celor care accesează Internet-ul folosind un modem cu care se conecteză la ISP-ul lor, acest lucru nu mai ține. Problema este următoarea: Ce se întâmplă când Elinor vrea să-i trimită Carolynei un e-mail și Carolyn nu este conectată la rețea în acel moment? Elinor nu va putea să stabilească o conexiune TCP cu Carolyn și astfel, nu va putea utiliza protocolul SMTP.

O soluție este ca agentul de transfer de mesaje de pe o mașină ISP să accepte e-mail-ul pentru clientii săi și să-l stocheze în cutiile lor poștale pe o mașină a ISP-ului. Din moment ce acest agent poate fi conectat la rețea tot timpul, se poate trimite e-mail 24 de ore pe zi.

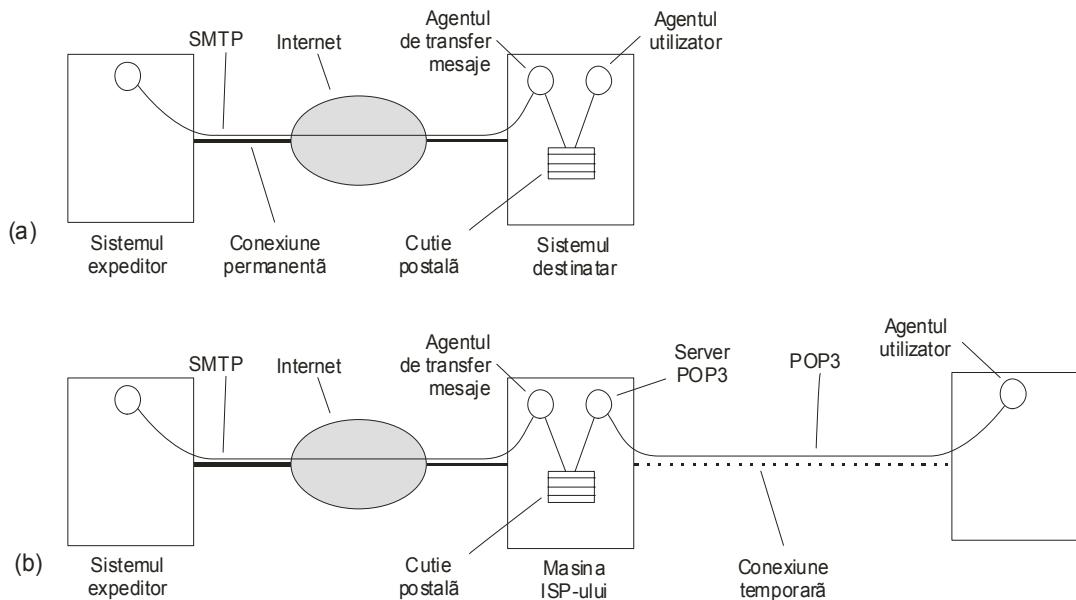
### POP3

Din nefericire, această soluție dă naștere altei probleme: cum își ia utilizatorul e-mail-ul de la agentul de transfer de mesaje al ISP-ului? Soluția acestei probleme este crearea unui alt protocol care să permită agenților de transfer mesaje (aflați pe calculatoarele clientilor) să contacteze agentul de transfer mesaje (de pe o mașină ISP) și să facă posibilă copierea e-mail-ului de la ISP la utilizator. Un astfel de protocol este **POP3** (**Post Office Protocol Version 3**- Protocol de poștă, versiunea 3), definit în RFC 1939.

Situată anteroară (când atât expeditorul cât și destinatarul aveau conexiune permanentă la Internet) este ilustrată în fig. 7-15(a). O situație în care expeditorul este efectiv conectat la rețea (online) dar destinatarul nu, este ilustrată în fig. 7-15(b).

POP3 începe când utilizatorul pornește programul cititor de poștă (mail reader). Acesta sună la ISP (în cază că nu există deja o conexiune) și stabilește o conexiune TCP cu agentul de transfer de mesaje, prin portul 110. Odată ce conexiunea a fost stabilită, protocolul POP3 trece succesiv prin următoarele trei stări:

1. Autorizare.



**Fig. 7-15.** (a) Trimiterea și citirea poștei când destinatarul are o conexiune permanentă la Internet, iar agentul utilizator rulează pe aceeași mașină ca și agentul de transfer de mesaje.  
(b) Citirea e-mail-ului când destinatarul are o conexiune comutată(dial-up) la un ISP.

2. Tranzacționare.
3. Actualizare.

Starea de autorizare se referă la admiterea utilizatorului în sistem (login). Starea de tranzacționare tratează colectarea e-mail-urilor și marcarea lor pentru ștergere din cutia poștală. Starea de actualizare se ocupă cu ștergerea efectivă a mesajelor.

Această comportare poate fi observată tastând ceva de genul:

```
telnet mail.isp.com 110
```

unde *mail.isp.com* reprezintă numele DNS al serverului de mail de la ISP. Telnet stabilește o conexiune TCP prin portul 110, pe care ascultă serverul POP3. După acceptarea conexiunii TCP, serverul trimite un mesaj ASCII anunțându-și prezența. De obicei, el începe cu **+OK** urmat de un comentariu. Un exemplu de scenariu este arătat în fig. 7-16 începând după ce conexiunea TCP a fost stabilită. Ca și mai înainte, liniile marcate cu **C:** sunt ale clientului (utilizatorului) iar cele cu **S:** sunt ale serverului (agentul de transfer de mesaje de la ISP).

În timpul stării de autorizare, clientul trimite numele său de utilizator și parola. După conectarea cu succes, clientul poate să transmită comanda **LIST**, care determină serverul să listeze conținutul cutiei poștale. Lista este terminată cu un punct.

Apoi, clientul poate regăsi mesajele folosind comanda **RETR** și le poate marca pentru ștergere cu **DELE**. Când toate mesajele au fost primele (și eventual marcate pentru ștergere), clientul trimite comanda **QUIT** pentru terminarea stării de tranzacționare și intrarea în stare de actualizare. Când serverul a șters toate mesajele, el trimite un răspuns și desființează conexiunea TCP.

```
S: +OK Serverul POP3 este pregătit
C: USER carolyn
    S: +OK
C: PASS vegetables
    S: +OK autentificare cu succes
C: LIST
    S: 1 2505
    S: 2 14302
    S: 3 8122
    S: .
C: RETR 1
    S: (trimite mesajul 1)
C: DELE 1
C: RETR 2
    S: (trimite mesajul 2)
C: DELE 2
C: RETR 3
    S: (trimite mesajul 3)
C: DELE 3
C: QUIT
    S: +OK Serverul POP3 întrerupe legătura
```

**Fig. 7-16.** Folosirea protocolului POP3 pentru a aduce trei mesaje.

Deși este adeverat că protocolul POP3 are abilitatea de a descărca un anumit mesaj sau un anumit grup de mesaje păstrându-le pe server, cele mai multe programe de e-mail descarcă tot și golesc cutia poștală. Ca urmare, practic singura copie rămâne înregistrată pe discul utilizatorului. Dacă acesta se strică, toate e-mail-urile pot fi pierdute definitiv.

Să recapitulăm pe scurt cum lucrează e-mail-ul pentru clienții unui ISP. Elinor creează un mesaj pentru Carolyn folosind un program de e-mail (adică, agentul utilizator) și face clic pe o icoană pentru a-l trimite. Programul de e-mail trimite mesajul agentului de transfer de mesaje de pe calculatorul Elinorei. Agentul de transfer de mesaje vede că mail-ul este pentru *carolyn@xyz.com* și folosește DNS pentru a căuta înregistrarea MX pentru *xyz.com* (unde *xyz.com* este ISP-ul Carolynei). Această cerere întoarce numele DNS al serverului de mail *xyz.com*. Agentul de transfer de mesaje caută acum adresa IP a acestei mașini folosind din nou DNS, de exemplu *gethostbyname*. Apoi, el stabilește o conexiune TCP cu serverul SMTP pe portul 25 de pe această mașină. Folosind o secvență de comenzi SMTP, asemănătoare celei din fig. 7-14, el transferă mesajul în cutia poștală a Carolynei și întrerupe conexiunea TCP.

După un timp, Carolyn își pornește PC-ul său, se conectează la ISP și pornește programul de e-mail. Programul de e-mail stabilește o conexiune TCP cu serverul POP3 pe portul 110 al serverului de poștă al ISP-ului. Numele DNS sau adresa IP a acestei mașini este configurată în mod normal atunci când programul de e-mail este instalat sau când este făcut contractul cu ISP-ul. După ce conexiunea TCP a fost stabilită, programul de e-mail al Carolynei lansează protocolul POP3 pentru a aduce conținutul cutiei sale poștale pe discul fix folosind comenzi ca cele din fig. 7-16. Odată ce a fost transferat tot e-mail-ul, conexiunea TCP este eliberată. De fapt, conexiunea cu ISP-ul poate fi desființată acum, din moment ce tot e-mailul este pe discul fix al Carolynei. Desigur, pentru a trimite un răspuns, va fi nevoie din nou de conexiunea cu ISP-ul, care, în general, nu este întreruptă imediat după aducerea poștei.

## IMAP

Pentru un utilizator cu un singur cont de e-mail, la un singur ISP, care este tot timpul accesat de la un singur PC, POP3 este bun și larg folosit datorită simplității și robusteții sale. Totuși, există în industria calculatoarelor un adevăr bine înrădăcinat, acela că imediat ce un lucru funcționează bine, cineva va începe să ceară mai multe facilități (și să aibă mai multe probleme). Astă s-a întâmplat și cu e-mail-ul. De exemplu, multă lume are un singur cont de e-mail la serviciu sau la școală și vrea să-l acceseze de pe PC-ul de acasă, de pe calculatorul portabil în călătoriile de afaceri și din Internet căfé-uri în vacanțe. Cu toate că POP3 permite asta, din moment ce în mod normal el descarcă toate mesajele la fiecare conectare, rezultatul constă în răspândirea e-mail-ului utilizatorului pe mai multe mașini, mai mult sau mai puțin la întâmplare, unele dintre ele nefiind ale utilizatorului.

Acest dezavantaj a dat naștere unei alternative a protocolului de livrare finală, **IMAP (Internet Message Access Protocol – Protocol pentru accesul mesajelor în Internet)**, care este definit în RFC 2060. Spre deosebire de POP3, care în mod normal presupune că utilizatorul își va goli căsuța poștală la fiecare conectare și va lucra deconectat de la rețea (off-line) după aceea, IMAP presupune că tot e-mail-ul va rămâne pe server oricât de mult, în mai multe căsuțe poștale. IMAP prevede mecanisme extinse pentru citirea mesajelor sau chiar a părților de mesaje, o facilitate folositoare când se utilizează un modem încet pentru citirea părții textuale a unui mesaj cu mai multe părți audio și video de mari dimensiuni. Întrucât premisa de folosire este că mesajele nu vor fi transferate pe calculatorul utilizatorului în vederea stocării permanente, IMAP asigură mecanisme pentru crearea, distrugerea și manipularea mai multor cutii poștale pe server. Astfel, un utilizator poate păstra o cutie poștală pentru fiecare corespondent și poate muta aici mesajele din inbox după ce acestea au fost citite.

IMAP are multe facilități, ca de exemplu posibilitatea de a se referi la un mesaj nu prin numărul de sosire, ca în fig. 7-8, ci utilizând attribute (de exemplu, dă-mi primul mesaj de la Bobbie). Spre deosebire de POP3, IMAP poate de asemenea să accepte atât expedierea mesajelor spre destinație cât și livrarea mesajelor venite.

Stilul general al protocolului IMAP este similar cu cel al POP3-ului, după cum se arată în fig. 7-16, cu excepția faptului că există zeci de comenzi. Serverul IMAP ascultă pe portul 143. În fig. 7-17 este prezentată o comparație între POP3 și IMAP. E bine de notat, totuși, că nu toate ISP-urile oferă ambele protocoale și că nu toate programele de e-mail le suportă pe amândouă. Așadar, atunci când alegeți un program de e-mail, este important să aflați ce protocoale suportă și să vă asigurați că ISP-ul oferă cel puțin unul din ele.

Caracteristica	POP3	IMAP
Unde este definit protocolul	RFC 1939	RFC 2060
Portul TCP folosit	110	143
Unde este stocat e-mail-ul	PC-ul utilizatorului	Server
Unde este citit e-mail-ul	Off-line	On-line
Timpul necesar conectării	Mic	Mare
Folosirea resurselor serverului	Minimă	Intensă
Mai multe cutii postale	Nu	Da
Cine face copii de siguranță la cutiile poștale	Utilizatorul	ISP-ul
Bun pentru utilizatorii mobili	Nu	Da
Controlul utilizatorului asupra scrisorilor preluate	Mic	Mare
Descărcare parțială a mesajelor	Nu	Da
Volumul discului alocat (disk quota) este o problemă	Nu	Ar putea fi în timp
Simplu de implementat	Da	Nu
Suport răspândit	Da	În creștere

Fig. 7-17. O comparație între POP3 și IMAP.

### Facilități de livrare

Indiferent dacă este folosit POP3 sau IMAP, multe sisteme oferă legături pentru procesarea adițională a mesajelor e-mail sosite. Un instrument deosebit de valoros pentru mulți utilizatori de e-mail este reprezentat de capacitatea de a construi filtre. Acestea sunt reguli care se verifică la sosirea mesajelor sau la pornirea agentului utilizator. Fiecare regulă specifică o condiție și o acțiune. De exemplu, o regulă ar putea spune că orice mesaj venit de la șef trebuie pus în cutia poștală numărul 1, orice mesaj de la un anumit grup de prieteni se duce în cutia poștală numărul 2 și orice alt mesaj conținând anumite cuvinte în Subiect este aruncat fără comentarii.

Unii ISP oferă filtre care clasifică automat mesajele sosite ca fiind importante sau nerelevante (spam) și memorează fiecare mesaj în cutia poștală corespunzătoare. Asemenea filtre funcționează verificând mai întâi dacă sursa este un autor cunoscut de mesaje „spam”. Apoi examinează subiectul. Dacă sute de utilizatori au primit un mesaj cu același subiect, probabil că el este nerelevant. Există și alte tehnici folosite pentru detectarea mesajelor lipsite de importanță.

O altă caracteristică a livrării, pusă la dispoziție adesea, este posibilitatea de a retrimit (temporar) poșta la o adresă diferită. Această adresă poate fi și un calculator utilizat de un serviciu comercial de comunicații, care va contacta utilizatorul prin radio sau satelit, afișând *Subject: linie pe pagerul său*.

O altă trăsătură comună a livrării finale este abilitatea de a instala un **demon de vacanță**. Aceasta este un program care examinează fiecare mesaj SOSIT și trimite o replică insipidă cum ar fi:

**Salut. Sunt în vacanță. Mă întorc pe 24 august. O zi bună.**

Asemenea răspunsuri pot să specifice, de asemenea, cum să fie tratate problemele urgente, alte persoane care pot fi contactate pentru probleme specifice etc. Majoritatea demonilor de vacanță păstrează urma celor cărora le-au trimis replici și se abțin de la a trimite unei aceleiași persoane o a doua asemenea replică. Demonii buni verifică și dacă mesajul SOSIT a fost trimis de la o listă de mail și în acest caz, nu mai răspund deloc. (Cei care trimit mesaje în timpul verii la liste mari de e-mail, probabil că nu doresc să primească sute de replici în care să le fie detaliate planurile de vacanță ale fiecaruia.)

Autorul s-a lovit recent de o formă extremă de prelucrare a livrării când a trimis o scrisoare unei persoane care pretinde că primește 600 de mesaje pe zi. Identitatea sa nu va fi deconspirată aici, ca nu cumva jumătate dintre cititorii acestei cărți să-i trimită și ei scrisori. Să-l numim în continuare John.

John și-a instalat un robot de e-mail care verifică fiecare mesaj SOSIT, ca să vadă dacă este de la un nou corespondent. Dacă este așa, trimite înapoi o replică standard în care explică faptul că nu mai poate să citească personal toate mesajele. În schimb a produs un document FAQ (Frequently Asked Questions) personal, unde răspunde la multe întrebări care i se pun de obicei. În mod normal, grupurile de știri și nu persoanele au documente FAQ.

Documentul FAQ al lui John dă adresa acestuia, numărul de fax și numerele de telefon și spune cum poate fi contactată firma sa. Arată cum poate fi chemat ca vorbitor și explică cum pot fi obținute lucrările sale și alte documente. Furnizează de asemenea referințe la programele scrise de el, o conferință pe care o organizează, un standard al cărui editor este și așa mai departe. E posibil ca această abordare să fie necesară, dar poate că un FAQ personal reprezintă simbolul final al statutului.

### Poșta electronică pe Web (Webmail)

Un subiect care merită menționat este poșta electronică pe Web. Anumite situri de Web, cum ar fi Hotmail sau Yahoo oferă servicii de poștă electronică oricui dorește. Ele funcționează după cum

urmează. Au agenți normali de transfer de mesaje, care așteaptă la portul 25 conexiuni noi de SMTP. Pentru a contacta, să spunem Hotmail, trebuie să obțineți înregistrarea sa DNS *MX*, de exemplu tastând

```
host -a -v hotmail.com
```

pe un sistem UNIX. Să presupunem că serverul de poștă electronică se numește *mx10.hotmail.com*; atunci tastând

```
telnet mx10.hotmail.com 25
```

se poate stabili o conexiune TCP prin care se pot trimite comenzi SMTP în modul obișnuit. Deocamdată, nimic special, cu excepția faptului că aceste servere mari sunt adeseori ocupate, ca atare se poate să dureze ceva mai mult până vă este acceptată o cerere de conexiune TCP.

Partea interesantă este cum se transmite poșta electronică. În principiu, atunci când utilizatorul se duce la pagina de Web a poștei electronice, îi este prezentat un formular în care i se cere un nume de cont și o parolă. Când utilizatorul face clic pe **Sign In**, numele de cont și parola sunt trimise serverului, care le validează. Dacă autentificarea s-a făcut cu succes, serverul găsește cutia poștală a utilizatorului și construiește o listă similară cu cea din fig. 7-8, cu diferența că are formatul unei pagini de Web în HTML. Pagina Web este transmisă apoi programului de navigare pentru a fi afișată. Pe multe din elementele paginii se pot executa clic-uri, astfel că mesajele pot fi citite, șterse, și a.m.d.

## 7.3 WORLD WIDE WEB

Web-ul este un context arhitectural pentru accesul la documente, răspândite pe mii de mașini din Internet, între care există legături. În 10 ani a evoluat de la o aplicație pentru transmiterea de date utile pentru fizica energiilor înalte la o aplicație despre care milioane de oameni cred că este Internetul. Popularitatea sa enormă se datorează faptului că are o interfață grafică plină de culoare, ușor de utilizat de către începători și în același timp oferă o cantitate imensă de informație - de la animale mitologice la tribul Zulu, pe aproape orice subiect posibil.

Web-ul (cunoscut și ca **WWW**) a apărut în 1989 la CERN, Centrul European de Cercetări Nucleare. CERN are câteva acceleratoare utilizate de echipe mari de cercetători din țările europene pentru cercetări în fizica particulelor. Deseori aceste echipe au membri din peste zece țări. Majoritatea experiențelor sunt foarte complicate și necesită ani de pregătire și construire de echipamente. Web-ul a apărut din necesitatea de a permite cercetătorilor răspândiți în lume să colaboreze utilizând colecții de rapoarte, planuri, desene, fotografii și alte tipuri de documente aflate într-o continuă modificare.

Propunerea inițială pentru crearea unei colecții de documente având legături între ele (Web) a fost făcută de fizicianul Tim Berners-Lee, fizician la CERN, în martie 1989. Primul prototip (bazat pe text) era operațional 18 luni mai târziu. În decembrie 1991, s-a făcut o demonstrație publică la conferința Hypertext'91 în San Antonio, Texas.

Aceasta demonstrație și publicitatea aferentă au atras atenția altor cercetători, fapt care l-a determinat pe Marc Andreessen de la University of Illinois să înceapă să dezvolte primul program de navigare grafic, Mosaic. Acesta a fost lansat în februarie 1993. Mosaic a fost atât de popular încât un an mai târziu Marc Andreessen a plecat pentru a forma o nouă companie, Netscape Communications Corp., care se ocupa cu dezvoltarea de software pentru Web. Când Netscape a devenit o com-

panie publică în 1995, investitorii, care probabil că au crezut că este vorba de un fenomen de tip Microsoft, au plătit 1,5 miliarde de dolari pentru acțiunile companiei. Acest record a fost cu atât mai neașteptat cu cât compania avea un singur produs, opera în deficit și anunțase probabilitatea investitorilor că nu se așteaptă la beneficii în viitorul apropiat. În următorii trei ani, Netscape Navigator și produsul Internet Explorer al companiei Microsoft au intrat într-un „război al programelor de navigare”, fiecare din produse încercând cu frenzie adăugarea de noi opțiuni (și astfel a mai multor erori) decât celălalt. În 1998, America Online a cumpărat Netscape Communications Corp. pentru suma de 4.2 miliarde \$, încheind astfel durata scurtă în care Netscape a fost o companie independentă.

În 1994, CERN și M.I.T. au semnat o înțelegere pentru a forma **Consortiul World Wide Web** (câteodată abreviat ca **W3C**), o organizație care are ca obiectiv dezvoltarea Web-ului, standardizarea protocolelor, și încurajarea interoperabilității între situri. Berners-Lee a devenit director. De atunci, sute de universități și companii au intrat în consorțiu. M.I.T. coordonează partea americană a consorțiuului în timp ce centrul de cercetări francez, INRIA, coordonează partea europeană. Deși există foarte multe cărți despre Web, cel mai bun loc pentru găsirea unor informații la zi despre el este (în mod natural) chiar Web-ul. Pagina consorțiuului are adresa [www.w3.org](http://www.w3.org). Cititorii interesați vor găsi acolo legături la pagini care acoperă toate documentele și activitățile consorțiuului.

### 7.3.1 Aspecte arhitecturale

Din punctul de vedere al utilizatorului, Web-ul constă dintr-o colecție imensă de documente sau **pagini de Web (Web pages)**, adesea numite prescurtat **pagini**, răspândite în toată lumea. Fiecare pagină poate să conțină legături (indicatori) la alte pagini, aflate oriunde în lume. Utilizatorii pot să aleagă o legătură prin execuția unui clic care îi va aduce la pagina indicată de legătură. Acest proces se poate repeta la nesfârșit. Ideea că o pagină să conțină legături către altele a fost inventată în 1945, cu mult înainte de a se fi inventat Internet-ul, de către Vannevar Bush, un profesor vizionar de la departamentul de inginerie electrică al M.I.T.

Paginile pot să fie văzute cu ajutorul unui **program de navigare (browser)**. Internet Explorer și Netscape Navigator sunt cele mai cunoscute programe de navigare. Programul de navigare aduce pagina cerută, interpretează textul și comenzi de formatare conținute în text și afișează pagina, formatată corespunzător, pe ecran. Un exemplu este prezentat în fig. 7-18(a). Ca majoritatea paginilor de Web, începe cu un titlu, conține informații și se termină cu adresa de poștă electronică a celui care menține pagina. Sirurile de caractere care reprezintă legături la alte pagini, se numesc **hiper-legături**, sunt afișate în mod diferit, fiind subliniate și/sau colorate cu o culoare specială. Pentru a selecta o legătură, utilizatorul va plasa cursorul pe zona respectivă, ceea ce va determina schimbarea formei cursorului și va executa un clic. Deși există programe de navigare fără interfață grafică, ca de exemplu Lynx, ele nu sunt atât de utilizate ca programele de navigare grafice, astfel încât în continuare ne vom referi numai la ultimele. Au fost dezvoltate și programe de navigare bazate pe voce.

Utilizatorii care sunt interesați să afle mai multe despre „Department of Animal Psychology” vor selecta numele respectiv (apare subliniat). Programul de navigare va aduce pagina la care este legat numele respectiv și o va afișa, așa cum se vede în fig. 7-18(b). Sirurile subliniate aici pot să fie selectate la rândul lor pentru a aduce alte pagini și aşa mai departe. Noua pagină se poate afla pe aceeași mașină ca și prima sau pe o mașină aflată undeva pe glob la polul opus. Utilizatorul nu va ști. Aduceerea paginilor este realizată de către programul de navigare, fără nici un ajutor din partea utilizatorului. Dacă utilizatorul se va întoarce la prima pagină, legăturile care au fost deja utilizate vor fi afișate

**WELCOME TO THE UNIVERSITY OF EAST PODUNK'S WWW HOME PAGE**

- Campus Information
  - [Admissions information](#)
  - [Campus map](#)
  - [Directions to campus](#)
  - [The UEP student body](#)
  
- Academic Departments
  - [Department of Animal Psychology](#)
  - [Department of Alternative Studies](#)
  - [Department Microbiotic Cooking](#)
  - [Department Nontraditional Studies](#)
  - [Department of Traditional Studies](#)

Webmaster @ eastpodunk.edu

(a)

**THE DEPARTMENT OF ANIMAL PSYCHOLOGY**

- [Information for prospective majors](#)
- [Personnel](#)
  - [Faculty members](#)
  - [Graduate students](#)
  - [Nonacademic staff](#)
- [Research Projects](#)
- [Positions available](#)
- Our most popular courses
  - [Dealing with herbivores](#)
  - [Horse management](#)
  - [Organic rat control](#)
  - [Negotiating with your pet](#)
  - [User-friendly dog house construction](#)
- [Full list of courses](#)

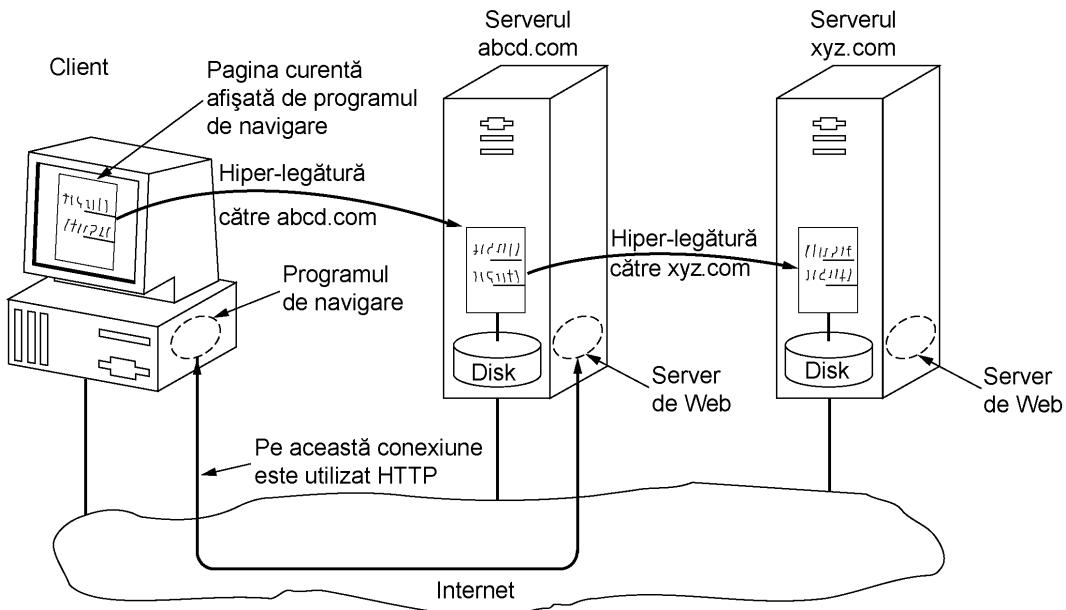
Webmaster @ animalpsyc.eastpodunk.edu

(b)

**Fig. 7-18.** (a) O pagină de Web. (b) pagina la care se ajunge dacă se selectează [Department of Animal Psychology](#)

altfel decât celelalte (subliniate cu linie punctată sau utilizând o altă culoare) pentru a fi deosebite de cele care nu au fost încă selectate. De notat că selecția liniei *Campus Information* din prima pagină nu are nici un efect. Nu este subliniată, ceea ce înseamnă că este pur și simplu un text care nu este legat de o altă pagină.

Modelul de bază al funcționării Web-ului este arătat în fig. 7-19. Aici un program de navigare afișează o pagină de Web pe mașina clientului. Atunci când utilizatorul face clic pe linia de text ce indică spre o pagină de pe serverul *abcd.com*, programul de navigare urmează hiper-legătura trimisă un mesaj serverului *abcd.com* în care se cere pagina respectivă. Atunci când pagina sosește, ea este afișată. Dacă această pagină conține o hiper-legătură către o pagină de pe serverul *xyz.com* pe care utilizatorul face clic, programul de navigare trimită o cerere mașinii respective pentru acea pagină, și aşa mai departe la nesfârșit.



**Fig. 7-19.** Părțile componente ale modelului Web-ului

### Aspecte privind clientul

Să examinăm acum în detaliu aspectele ce privesc clientul din fig. 7-19. În esență, un program de navigare este o aplicație capabilă să afișeze o pagină de Web și să capteze clicurile mouse-ului pe elemente ale paginii afișate. Când un element este selectat, programul de navigare urmează hiper-legătura și obține pagina selectată. Ca atare, hiper-legătura conținută în pagină necesită o modalitate de a adresa prin nume orice altă pagină de pe Web. Paginile sunt adresate prin nume folosind **URL-uri** (Uniform Resource Locators, rom.: Localizatoare Uniforme de Rezurse). Un URL tipic este

<http://www.abcd.com/products.html>

Vom explica ce înseamnă URL mai târziu, în cadrul capitolului curent. Deocamdată, este suficient să știm că un URL are trei părți: numele protocolului (*http*), numele calculatorului pe care se găsește pagina ([www.abcd.com](http://www.abcd.com)) și numele fișierului care conține pagina (*products.html*).

Când un utilizator execută un clic pe o hiper-legătură, programul de navigare urmează o serie de etape pentru a obține pagina indicată de hiper-legătura. Să presupunem ca utilizatorul naveghează pe Web și găsește o legătura despre telefonia pe Internet care indică spre pagina principală a ITU, <http://www.itu.org/home/index.html>. Să urmăm etapele parcuse când această legătură este selectată.

1. Programul de navigare determină URL (pe baza selecției).
2. Programul de navigare întreabă DNS care este adresa IP pentru [www.itu.org](http://www.itu.org).
3. DNS răspunde cu 156.106.192.32.
4. Programul de navigare realizează conexiunea TCP cu portul 80 al 156.106.192.32.
5. Trimit o cerere pentru fișierul */home/index.html*.
6. Serverul [www.itu.org](http://www.itu.org) transmite fișierul */home/index.html*.
7. Conexiunea TCP este eliberată.
8. Programul de navigare afișează textul din */home/index.html*.
9. Programul de navigare aduce și afișează toate imaginile din acest fișier.

Multe programe de navigare informează despre etapa care se execută într-o fereastră de stare, în partea de jos a paginii. În acest mod, dacă performanțele sunt slabe, utilizatorul poate să știe dacă este vorba de faptul că DNS nu răspunde, că serverul nu răspunde, sau pur și simplu de congestia rețelei în timpul transmisiei paginii.

Pentru a afișa noua pagină (sau orice pagină), programul de navigare trebuie să înțeleagă forma în care este scrisă. Pentru a permite tuturor programelor de navigare să înțeleagă orice pagină de Web, paginile de Web sunt scrise într-un limbaj standardizat numit HTML, care descrie paginile de Web. Vom discuta acest limbaj mai târziu în acest capitol.

Deși un program de navigare este în principiu un interpretor de HTML, majoritatea programelor de navigare au numeroase butoane și opțiuni care ajută navigarea prin Web. Multe au un buton pentru revenirea la pagina anterioară, un buton pentru a merge la pagina următoare (acest buton este operațional numai după ce utilizatorul s-a întors înapoi dintr-o pagină) și un buton pentru selecția paginii personale (home page). Majoritatea programelor de navigare au un buton sau un meniu pentru înregistrarea unei adrese de pagină (bookmark) și un altul care permite afișarea unor adrese înregistrate, făcând astfel posibilă revenirea la o pagină cu ajutorul cătorva selectii simple realizate cu mouseul. Paginile pot să fie salvate pe disc sau tipărite. Sunt disponibile numeroase opțiuni pentru controlul ecranului și configurarea programului de navigare conform dorințelor utilizatorului.

În afară de text obișnuit (nesubliniat) și hipertext (subliniat), paginile de Web pot să conțină iconițe, desene, hărți, fotografii. Fiecare dintre acestea poate să fie, în mod optional, legată la altă pagină. Dacă se selectează unul dintre aceste elemente, programul de navigare va aduce pagina respectivă și o va afișa, așa cum se întâmplă în cazul selectării unui text. Pentru imaginile care sunt fotografii sau hărți, alegerea paginii care se aduce poate să depindă de regiunea din imagine pe care se face selecția.

Nu toate paginile conțin HTML. O pagină poate fi formată dintr-un document în format PDF, o iconiță în format GIF, o fotografie în format JPEG, o melodie în format MP3, o înregistrare video în format MPEG sau oricare din cele alte câteva sute de tipuri de fișiere. Deoarece paginile în forma standard HTML pot avea legături către oricare din acestea, programul de navigare are o problemă atunci când întâlnește o pagină pe care nu o poate interpreta.

În loc să facă programele de navigare din ce în ce mai mari, înglobând interpretoare pentru o colecție de tipuri de fișiere în creștere rapidă, majoritatea programelor de navigare au ales o soluție mai generală. Atunci când un server întoarce o pagină, el întoarce de asemenea informații adiționale despre acea pagină. Această informație include tipul MIME al paginii (fig. 7-12). Paginile de tipul *text/html* sunt afișate direct, ca și paginile de alte câteva tipuri interpretate implicit. Dacă tipul MIME nu este unul dintre acestea, programul de navigare își consultă tabela de tipuri MIME pentru a afla cum să afișeze pagina. Această tabelă asociază un tip MIME cu o aplicație de vizualizare.

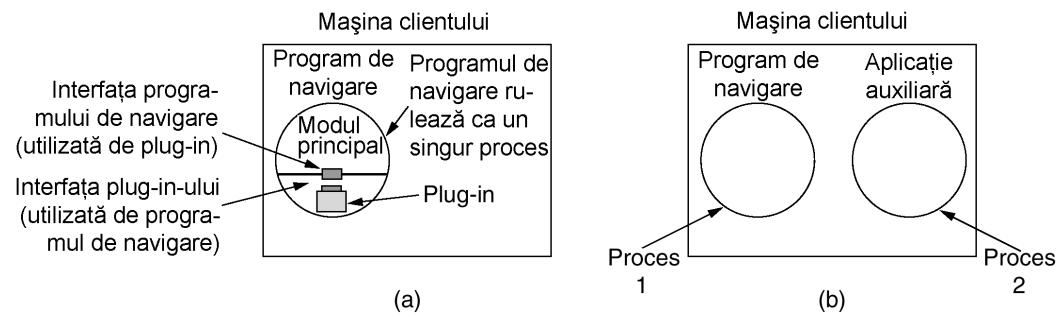


Fig. 7-20. (a) Un plug-in al programului de navigare. (b) O aplicație auxiliară

Există două posibilități: plug-in-uri și programe auxiliare (*helper applications*). Un **plug-in** este un modul pe care programul de navigare îl obține dintr-un director special de pe disc și îl instalează ca o extensie al înșuși programului de navigare, așa cum se arată în fig. 7-20(a). Deoarece plug-in-urile se execută în interiorul programului de navigare, acestea au acces la pagina curentă și pot să modifice modul în care aceasta este afișată. După ce plug-in-ul a terminat ceea ce avea de făcut (de obicei după ce utilizatorul s-a deplasat la altă pagină de Web), acesta este scos din memoria programului de navigare.

Fiecare program de navigare are o colecție de proceduri pe care toate plug-in-urile trebuie să le implementeze pentru ca programul de navigare să poată executa plug-in-ul. De exemplu, există de obicei o procedură pe care modulul principal al programului de navigare o apelează pentru a oferi plug-in-ului date ce trebuie afișate. Această colecție de proceduri constituie interfața unui plug-in și este particulară fiecărui program de navigare.

În plus, programul de navigare pune la dispoziția plug-in-ului propria sa colecție de proceduri. Procedurile tipice din interfața programului de navigare se referă la alocarea și eliberarea memoriei, afișarea de mesaje în fereastra de stare a programului de navigare sau obținerea parametrilor acestuia.

Înainte ca un plug-in să poată fi folosit, acesta trebuie instalat. Procedura uzuială de instalare este ca utilizatorul să navigheze la situl de Web al plug-in-ului și să copieze un fișier de instalare. Pe sistemele Windows, acesta este de obicei o arhivă zip cu decompresie automată cu extensia .exe. Când pe acest fișier se execută un dublu clic, se execută un program de mici dimensiuni atașat părții de început a fișierului. Acest program decomprimă plug-in-ul și îl copiază în directorul de plug-in-uri al programului de navigare. Apoi execută apelurile necesare pentru înregistrarea tipului MIME corespunzător și asocierea acestuia cu plug-in-ul. Pe sistemele UNIX, fișierul de instalare este în mod frecvent un fișier de comenzi care se ocupă de copiere și înregistrare.

Cea de-a doua modalitate de extindere a unui program de navigare este utilizarea **aplicațiilor auxiliare** (*helper applications*). Acestea sunt programe complete ce se execută ca procese separate. Acest fapt este ilustrat în fig. 7-20(b). Deoarece acestea sunt programe separate, nu oferă nici o interfață programului de navigare și nu utilizează serviciile acestuia. De obicei însă acceptă doar numele unui fișier temporar unde a fost stocat conținutul paginii, deschide acest fișier și îi afișează conținutul. De obicei, aplicațiile auxiliare sunt programe de dimensiuni mari care există independent de programul de navigare, cum ar fi Adobe Acrobat Reader pentru afișarea fișierelor PDF, sau Microsoft Word. Unele programe (cum ar fi Acrobat) dispun de un plug-in care execută aplicația auxiliară.

Multe aplicații auxiliare folosesc tipul MIME *application*. A fost definit un număr considerabil de subtipuri, de exemplu *application/pdf* pentru fișiere PDF și *application/msword* pentru fișiere Word. În acest mod, un URL poate să indice direct către un fișier PDF sau Word și atunci când utilizatorul execută un clic asupra sa aplicațiile Acrobat sau Word sunt pornite automat și li se transmite numele fișierului temporar ce conține datele ce trebuie afișate. Ca atare, programele de navigare pot fi configurate să trateze un număr teoretic nelimitat de tipuri de documente, fără schimbări aduse programului de navigare. Serverele de Web moderne sunt adesea configurate cu sute de combinații de tipuri/subtipuri și combinații noi sunt adăugate de fiecare dată când este instalat un program nou.

Aplicațiile auxiliare nu sunt restricționate la utilizarea tipului MIME *application*. Adobe Photoshop folosește *image/x-photoshop* și RealOne Player poate trata de exemplu *audio/mp3*.

Pe sistemele Windows, atunci când un program este instalat pe un calculator, el înregistrează tipurile MIME pe care dorește să le trateze. Acest mecanism conduce la conflicte atunci când mai multe aplicații sunt disponibile pentru vizualizarea unui subtip, cum ar fi *video/mpg*. Ceea ce se în-

tâmplă este că ultimul program ce se înregistrează supraînscrie asociațiile existente (tip MIME, aplicație auxiliară), captând tipul pentru sine. Ca o consecință, instalarea unui nou program poate schimba modul în care un program de navigare tratează tipurile existente.

Pe sistemele UNIX, acest proces de înregistrare nu se face în general automat. Utilizatorul trebuie să schimbe anumite fișiere de configurație. Această abordare conduce la un volum mai mare de muncă dar la surprize mai puține.

Programele de navigare pot de asemenea deschide fișiere locale în loc de a le obține de pe servere de Web de la distanță. Deoarece fișierele locale nu au tipuri MIME, programul de navigare necesită o metodă pentru a determina ce plug-in sau aplicație auxiliară trebuie folosit pentru alte tipuri decât cele tratate implicit ca *text/html* sau *image/jpeg*. Pentru tratarea fișierelor locale, aplicațiile auxiliare pot fi asociate și cu o extensie de fișier, ca și cu un tip MIME. Considerând configurația standard, deschiderea fișierului *foo.pdf* îl va încărca în Acrobat și deschiderea fișierului *bar.doc* îl va încărca în Word. Anumite programe de navigare folosesc tipul MIME, extensia fișierului și chiar informații din interiorul fișierului pentru a ghica tipul MIME. În special Internet Explorer se bazează în primul rând pe extensia fișierului decât pe tipul MIME atunci când acest lucru este posibil.

Și aici pot apărea conflicte deoarece multe programe sunt dispuse, de fapt dormice să trateze *mpg*, de exemplu. În timpul instalării, programele create pentru profesioniști afișează frecvent căsuțe de selecție pentru tipurile MIME și extensiile pe care sunt pregătite să le trateze pentru a permite utilizatorului selecția celor dorite pentru a preveni astfel supraînscrierea accidentală a asociațiilor existente. Programele ce au ca țintă marea masă a consumatorilor presupun că utilizatorul nu știe ce este un tip MIME și pur și simplu acaparează tot ce pot fără să țină seama de ceea ce au făcut programele instalate anterior.

Capacitatea de a extinde programul de navigare cu un număr mare de tipuri noi este utilă, dar poate duce și la probleme. Atunci când Internet Explorer obține un fișier cu extensia *exe* își da seama că acest fișier este un program executabil și ca atare nu are o aplicație adițională asociată. Acțiunea evidentă este execuția fișierului. Însă această acțiune poate fi o problemă de securitate enormă. Tot ceea ce trebuie să facă un site de Web rău intenționat este să ofere o pagină de Web, de exemplu cu fotografii de staruri de cinema sau sportive, toate imaginile indicând către un virus. Un singur clic pe o imagine determină ca un program executabil necunoscut și posibil ostil să fie copiat și executat pe mașina utilizatorului. Pentru a preveni astfel de vizitatori nedoriți, Internet Explorer poate fi configurat să fie selectiv în a executa programe necunoscute în mod automat, dar nu toți utilizatorii înțeleg cum să folosească această configurație.

Pe sistemele UNIX pot exista probleme similare cu fișierele de comenzi pentru consola sistemului, dar aceasta necesită ca utilizatorul să instaleze în mod conștient consola sistemului ca aplicație auxiliară. Din fericire, acest proces este suficient de complicat pentru ca nimeni să nu poată să îl efectueze accidental (și puține persoane pot să îl efectueze chiar intenționat).

### Aspecte privind serverul

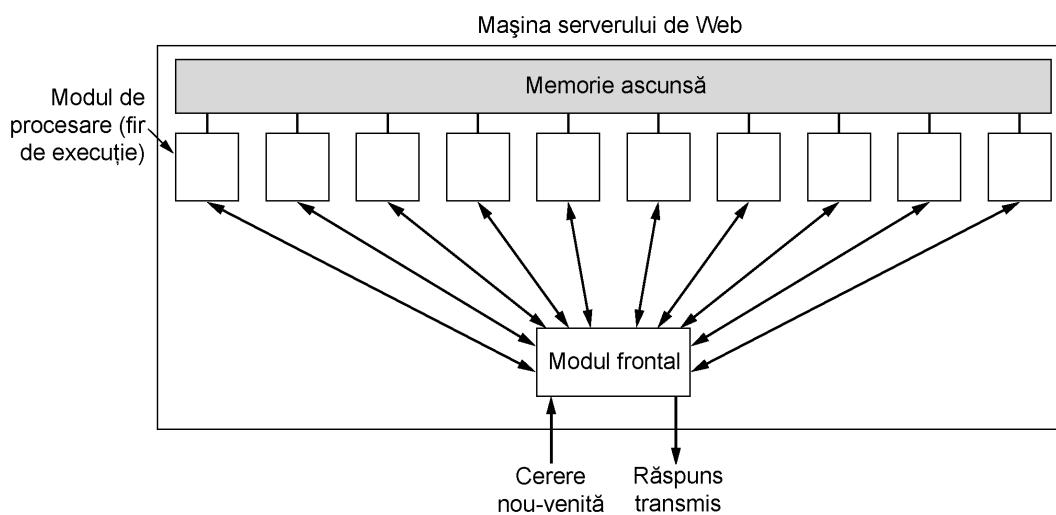
Cum atât despre aspectele privind clientul. Să ne referim acum la aspectele privind serverul. Așa cum am văzut mai sus, atunci când utilizatorul tastează un URL sau execută un clic asupra unei linii de hipertext, programul de navigare analizează URL-ul și interpretează partea între *http://* și următorul caracter / ca un nume DNS ce trebuie căutat. Înarmat cu adresa IP a serverului, programul de navigare stabilește o conexiune TCP la portul 80 de pe acel server. Apoi se transmite o comandă ce conține restul URL-ului, care este de fapt numele fișierului de pe acel server. Serverul întoarce apoi fișierul pentru a fi afișat de către programul de navigare.

Într-o primă aproximare, un server de Web este similar cu serverul din fig. 6-6. Acel server, ca și un server de Web real, primește numele fișierului ce trebuie căutat și transmis programului de navigare. În ambele cazuri, etapele pe care le parcurge serverul în buclă sa principală sunt:

1. Acceptă o conexiune TCP de la un client (program de navigare).
2. Obține numele fișierului cerut.
3. Obține fișierul (de pe disc).
4. Întoarce fișierul clientului.
5. Eliberează conexiunea TCP.

Serverele de Web moderne au mai multe caracteristici, dar în esență acestea sunt funcțiile unui server de Web. O problemă cu această arhitectură este că fiecare cerere necesită acces la disc pentru obținerea fișierului. Rezultatul este că serverul de Web nu poate servi mai multe cereri pe secundă decât numărul de accese la disc ce se pot executa pe secundă. Un disc SCSI are un timp de acces mediu de circa 5 ms, ceea ce limitează serverul la cel mult 200 de cereri/sec, chiar mai puțin dacă trebuie citite des fișiere mari. Pentru un sit de Web de importanță mare, acest număr este prea mic.

O îmbunătățire evidentă (utilizată de toate serverele de Web) este folosirea unui sistem de memorie ascunsă, temporară pentru cele mai recente  $n$  fișiere utilizate. Înainte de obținerea unui fișier de pe disc, serverul verifică memoria ascunsă (cache). Dacă fișierul există acolo, el poate fi servit direct din memorie, eliminând astfel accesul la disc. Deși pentru o memorie ascunsă eficientă sunt necesare o cantitate mare de memorie principală și timp de procesare suplimentară pentru a analiza memoria ascunsă și pentru a-i administra conținutul, economia de timp este aproape întotdeauna superioară timpului suplimentar de procesare și costului memoriei.



**Fig. 7-21.** Un server de Web cu mai multe fire de execuție cu un modul frontal și module de procesare

Următorul pas pentru construcția unui server mai rapid este de a face serverul să admită mai multe fire de execuție (*multithreaded*). Într-o altă arhitectură, serverul este format dintr-un modul frontal (*front-end module*), care acceptă conexiunile noi venite, și  $k$  module de procesare, așa cum arată figura 7-21. Cele  $k + 1$  fire de execuție aparțin toate aceluiași proces, astfel că modulele de proce-

sare au toate acces la memoria ascunsă din interiorul spațiului de adrese al procesului. La sosirea unei cereri, modulul frontal o acceptă și construiește o scurtă înregistrare ce descrie cererea. Aceasta este transmisă apoi unuia dintre modulele de procesare. În altă arhitectură posibilă, modulul frontal este eliminat și fiecare modul de procesare încearcă să își obțină propriile cereri, dar în acest caz este necesar un protocol de sincronizare pentru prevenirea conflictelor.

Modulul de procesare verifică mai întâi memoria ascunsă pentru a determina dacă fișierul necesar se află acolo. Dacă da, modifică înregistrarea pentru a include și un indicator către fișierul din înregistrare. Dacă fișierul nu se află acolo, modulul de procesare începe operațiile cu discul pentru a citi fișierul în memoria ascunsă (renunțând eventual la alte fișiere pentru a face loc acestuia). Când fișierul este citit de pe disc, el este pus în cache și de asemenea transmis clientului.

Avantajul acestei scheme este că în timp ce unul sau mai multe module de procesare sunt blocați așteptând terminarea operațiilor cu discul (și deci nu consumă din timpul procesorului), alte module pot fi active lucrând la satisfacerea altor cereri. Desigur, pentru a obține o îmbunătățire reală asupra modelului cu un singur fir de execuție este necesară existența mai multor unități de disc, astfel încât mai multe discuri să poată fi ocupate în același timp. Cu ajutorul a  $k$  module de procesare și  $k$  unități de disc, eficiența poate crește până la de  $k$  ori față de modelul serverului cu un singur fir de execuție și o singură unitate de disc.

Teoretic, un server cu un singur fir de execuție și  $k$  unități de disc poate de asemenea câștiga un factor  $k$  în ceea ce privește eficiență, dar implementarea și administrarea sunt mult mai complicate deoarece apelele de sistem READ normale, blocante nu pot fi folosite pentru accesul la disc. În cazul unui server cu mai multe fire de execuție, acestea pot fi folosite deoarece o operație READ blochează doar firul de execuție care a executat operația și nu întregul proces.

Servelele de Web moderne efectuează mai multe operații decât acceptarea numelor de fișiere și transmiterea conținutului acestora. De fapt, procesarea fiecărei cereri poate deveni destul de complicată. Din acest motiv, într-un număr mare de servere fiecare modul de procesare efectuează o serie de etape. Modulul frontal transmite fiecare cerere sosită către primul modul de procesare disponibil, care apoi execută cererea, utilizând o submulțime a următorilor pași, în funcție de ce pași sunt necesari pentru respectiva cerere.

1. Rezolvarea numelui paginii de Web cerute.
2. Autentificarea clientului.
3. Verificarea drepturilor de acces ale clientului.
4. Verificarea drepturilor de acces asupra paginii de Web.
5. Verificarea memoriei ascunse.
6. Obținerea paginii cerute, de pe disc.
7. Determinarea tipului MIME ce va fi inclus în răspuns.
8. Rezolvarea altor probleme minore.
9. Transmiterea răspunsului către client.
10. Adăugarea unei înregistrări în jurnalul serverului.

Pasul 1 este necesar deoarece cererea sosită poate să nu conțină numele propriu-zis al fișierului, ca și de caractere. De exemplu, putem considera URL-ul <http://www.cs.vu.nl>, care are un nume de fișier vid. Aceasta trebuie extins la un nume de fișier implicit. De asemenea, programele de navigare moderne pot specifica limba implicită a utilizatorului (de ex.: italiană sau engleză), ceea ce deschide posibilitatea ca serverul să selecteze o pagină de Web în acea limbă, dacă aceasta este disponibilă. În

general, extinderea numelor nu este un proces atât de banal cum ar putea părea la prima vedere, datorită unei varietăți de convenții existente privind numirea fișierelor.

Pasul 2 constă în verificarea identității clientului. Acest pas este necesar pentru paginile care nu sunt disponibile publicului larg. Vom discuta o modalitate de a realiza acest lucru mai târziu, în acest capitol.

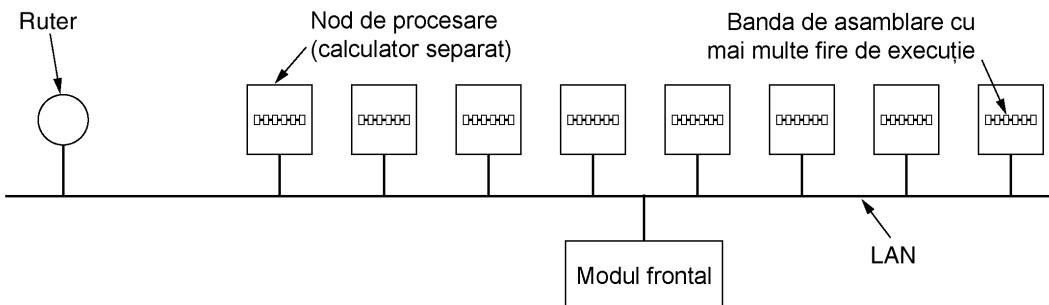
Pasul 3 verifică dacă există restricții referitoare la satisfacerea cererii, având în vedere identitatea și localizarea clientului. Pasul 4 verifică dacă există restricții de acces asociate cu pagina însăși. Dacă un anumit fișier (de ex.: *.htaccess*) este prezent în directorul unde se află și pagina dorită, accesul la acel fișier poate fi restrâns la anumite domenii, de exemplu numai la utilizatorii din interiorul companiei.

Pașii 5 și 6 presupun obținerea paginii. Pasul 6 necesită capacitatea de tratare simultană a mai multor citiri de pe disc.

Pasul 7 se referă la determinarea tipului MIME din extensia fișierului, primele câteva cuvinte din fișier, un fișier de configurare sau alte surse posibile. Pasul 8 este destinat unei diversități de operații, cum ar fi construcția unui profil al utilizatorului sau adunarea unor statistici.

Pasul 9 este cel în care rezultatul este transmis clientului și pasul 10 adaugă o înregistrare în jurnalul sistemului, în scopuri administrative. Asemenea fișiere de jurnalizare pot fi analizate ulterior pentru obținerea de informații importante despre comportamentul utilizatorului, spre exemplu ordinea în care vizitatorii accesează paginile.

Dacă sosesc prea multe cereri în fiecare secundă, procesorul nu va fi capabil să suporte încărcarea, oricără unități de disc ar fi utilizate în paralel. Soluția este adăugarea mai multor noduri (calculatoare), posibil cu unități de disc replicate pentru a evita ca discurile să devină următorul punct de gătuire. Acest fapt conduce la modelul **fermei de servere (server farm)** din fig. 7-22. Un modul frontal acceptă în continuare cererile dar le împarte mai multor procesoare, nu mai multor fire de execuție, pentru a reduce încărcarea pe fiecare calculator. Mașinile individuale pot fi cu mai multe fire de execuție și în bandă de asamblare ca mai sus.

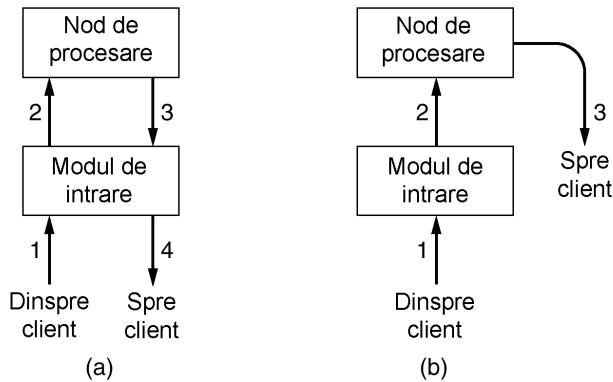


**Fig. 7-22.** O fermă de servere

O problemă cu fermele de servere este că nu mai există o memorie ascunsă partajată deoarece fiecare nod are propria sa memorie – decât dacă se folosește un sistem multiprocesor cu memorie partajată de cost mic. O modalitate de a contracara această pierdere de performanță este un modul frontal care reține unde direcționează fiecare cerere și trimite cererile ulterioare pentru aceeași pagină aceluiași nod. Această abordare specializează fiecare nod în tratarea anumitor pagini astfel încât spațiul destinat pentru cache nu se pierde reținând fiecare fișier în fiecare cache.

O altă problemă cu fermele de servere este aceea că fiecare conexiune TCP a clientului se termină la modulul de intrare, astfel că răspunsul trebuie transmis prin acest modul. Această situație este

evidențiată în fig. 7-23(a), unde cererea sosită (1) și răspunsul transmis (4) trec ambele prin modulul frontal. Câteodată se poate folosi o soluție ingenioasă numită **parsare TCP** (eng.: TCP handoff) pentru a evita această problemă. Cu acest truc, capătul comunicației TCP este „pasat” nodului de procesare astfel că acesta poate replica direct clientului, lucru evidențiat ca (3) în fig. 7-23(b). Această pasare este făcută într-un mod transparent pentru client.



**Fig. 7-23.** (a) Secvență normală de mesaje cerere – răspuns.  
(b) Secvență în care se folosește pasarea TCP.

### URL- Uniform Resource Locators

Am spus de mai multe ori că o pagină de Web poate să conțină referințe la alte pagini. Să explicăm cum sunt implementate aceste referințe. Încă de la crearea Web-ului, a fost clar că pentru a avea o pagină care să indice spre altă pagină este necesar un mecanism care să permită numirea și regăsirea paginilor. În particular, sunt trei întrebări la care trebuie să se răspundă înainte de a se putea afișa o pagină:

1. Cum se numește pagina?
2. Cum este localizată pagina?
3. Cum se face accesul la pagină?

Dacă fiecare pagină ar avea un nume unic, atunci nu ar exista nici o ambiguitate în identificarea paginilor. Totuși, problema nu este încă rezolvată. Să considerăm de exemplu o paralelă între oameni și pagini. În SUA aproape fiecare persoană are un număr de asigurare socială, care este un identificator unic, astfel încât nu există două persoane cu același număr. Totuși, cunoscând numai numărul respectiv, nu există nici o posibilitate de a găsi adresa persoanei respective și sigur nu se poate afla dacă persoanei respective trebuie să i se scrie în Engleză, Spaniolă sau Chineză. Web-ul are practic același fel de probleme.

Soluția aleasă identifică paginile într-un mod care rezolvă toate trei problemele în același timp. Fiecare pagină are un **URL (Uniform Resource Locator)** - adresa uniformă pentru localizarea resurselor care funcționează ca nume al paginii general valabil. Un URL are trei componente: protocolul (cunoscut și sub numele de **schemă**), numele DNS al mașinii pe care este memorat fișierul și un nume local, care indică în mod unic pagina (de obicei numele fișierului care conține pagina). De exemplu, situl de Web al departamentului din care face parte autorul conține un număr de înregistrări video despre universitate și despre orașul Amsterdam. URL-ul paginii cu înregistrările video este:

<http://www.cs.vu.nl/video/index-en.html>

Acest URL este format din trei componente: protocolul (*http*), numele DNS al serverului (*www.cs.vu.nl*) și numele fișierului (*video/index-en.html*), cu semnele de punctuație corespunzătoare. Numele fișierului este o cale relativă la directorul de Web implicit de la *cs.vu.nl*.

Se utilizează notații care reprezintă prescurtări standard. În cazul multor situri, un nume de fișier nul înseamnă implicit pagina principală a organizației. În mod obișnuit, atunci când numele fișierului denotă un director, aceasta implică un fișier numit *index.html*. În sfârșit, *~user/* poate să fie pus în corespondență cu directorul WWW al utilizatorului *user*, și apoi cu fișierul *index.html* în acest director. De exemplu, pagina autorului poate să fie referită ca:

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

chiar dacă de fapt numele propriu-zis al fișierului este *index.html*, implicit în acest director.

Acum ar trebui să fie clar cum funcționează hipertextul. Pentru a face o porțiune de text selectabilă, cel care scrie pagina trebuie să furnizeze două elemente: textul prin care se face selecția și URL-ul paginii care trebuie adusă, dacă textul este selectat. Vom explica sintaxa comenzi mai târziu în acest capitol.

Când se face selecția, programul de navigare căută numele serverului utilizând DNS-ul. Pe baza adresei IP a serverului, programul de navigare stabilește o conexiune TCP spre server. Utilizând această conexiune, se transmite numele fișierului utilizând protocolul specificat. Bingo. Acum se poate să se vadă pagina.

Această schemă URL este deschisă în sensul că este simplu să se utilizeze alte protocoale pentru a se obține diferite tipuri de resurse. De fapt au fost definite URL-uri pentru protocoalele obișnuite, și multe programe de navigare înțeleg aceste protocoale. Forme simplificate ale celor mai obișnuite sunt prezentate în fig. 7-24.

Nume	Utilizat pentru	Exemplu
http	Hipertext (HTML)	<a href="http://www.cs.vu.nl/~ast">http://www.cs.vu.nl/~ast</a>
ftp	FTP	<a href="ftp://ftp.cs.vu.nl/pub/minix/README">ftp://ftp.cs.vu.nl/pub/minix/README</a>
File	Fișier local	<a href="file:///usr/suzanne/prog.c">file:///usr/suzanne/prog.c</a>
news	Grup de știri	<a href="news:AA0134223112@cs.utah.edu">news:AA0134223112@cs.utah.edu</a>
news	Articol de știri	<a href="news:AA0134223112@cs.utah.edu">news:AA0134223112@cs.utah.edu</a>
gopher	Gopher	<a href="gopher://gopher.tc.umn.edu/11/libraries">gopher://gopher.tc.umn.edu/11/libraries</a>
mailto	Trimitere de poșta electronică	<a href="mailto:JohnUser@acm.org">mailto:JohnUser@acm.org</a>
telnet	Conectare la distanță	<a href="telnet://www.w3.org:80">telnet://www.w3.org:80</a>

Fig. 7-24. Câteva URL-uri obișnuite.

Să parcurgem lista rapid. Protocolul *http* este protocolul nativ pentru Web, el este utilizat de către serverele de Web. **HTTP** este o prescurtare pentru **HyperText Transfer Protocol**. Vom examina mai detaliat acest protocol mai târziu în acest capitol.

Protocolul *ftp* este utilizat pentru accesul la fișiere prin FTP (File Transfer Protocol - protocol pentru transferul de fișiere), protocolul Internet de transfer de fișiere. FTP este utilizat de peste douăzeci de ani și este foarte răspândit. Numeroase servere de FTP din toată lumea permit ca de oriunde din Internet să se facă o conectare și să se aducă orice fișier plasat pe un server FTP. Web-ul nu aduce schimbări aici, face doar ca obținerea fișierelor să se facă mai ușor, pentru că FTP are o interfață mai puțin prietenoasă (dar este mai puternic decât HTTP, deoarece permite de exemplu ca un utilizator de pe mașina A să transfere un fișier de pe mașina B pe mașina C).

Este posibil să se facă acces la un fișier local ca la o pagină de Web, fie utilizând protocolul *file* (fișier), fie pur și simplu utilizând numele fișierului. Această abordare este similară utilizării protocolului FTP, dar nu implică existența unui server. Desigur funcționează numai pentru fișiere locale, nu și pentru cele aflate la distanță.

Cu mult înainte de apariția Internet-ului exista sistemul de știri USENET. Acesta este format din aproximativ 30000 de grupuri de știri în care milioane de persoane discută despre o mare varietate de subiecte, adăugând și citind articole legate de subiectul grupului de știri. Protocolul *news* permite citirea unui articol din știri ca și cum ar fi o pagină de Web. Aceasta înseamnă că un program de navigare este în același timp și un cititor de știri. De fapt, multe programe de navigare au butoane sau elemente de meniu care permit citirea știrilor USENET mai ușor decât dacă se utilizează cititoare standard de știri.

Protocolul *news* admite două formate. Primul format specifică un grup de știri și poate să fie utilizat pentru a obține o listă de articole de la un server de știri preconfigurat. Al doilea format cere identificatorul unui articol, de exemplu *AA0134223112@cs.utah.edu*. Programul de navigare aduce articolul de la serverul corespunzător utilizând protocolul **NNTP (Network News Transfer Protocol – Protocol de transfer al știrilor prin rețea)**. Nu vom studia NNTP în această carte, dar este în mare bazat pe SMTP și are un stil similar.

Protocolul *gopher* era utilizat de sistemul Gopher, care a fost proiectat la universitatea Minnesota. Numele este cel al echipei atletice a universității, the Golden Gopher (de asemenea acest nume este utilizat în argou pentru „go for” adică o comandă de aducere). Gopher-ul a precedat Web-ul cu câțiva ani. Era o metodă de regăsire a informației, similară conceptual cu cea utilizată de Web, dar acceptând numai text și imagini. Este considerat depășit și nu se mai folosește în prezent.

Ultimele două protocole nu sunt de fapt protocole pentru aducerea unor pagini de Web, dar sunt utile. Protocolul *mailto* permite transmiterea de poștă dintr-un program de navigare. Pentru a face această operație, se selectează butonul OPEN și se specifică un URL constând din *mailto:* urmat de adresa destinatarului. Majoritatea programelor de navigare vor răspunde prin pornirea unei aplicații de poștă electronică cu adresa și câteva alte câmpuri din antet deja complete.

Protocolul *telnet* este utilizat pentru stabilirea unei conexiuni cu o mașină aflată la distanță. Se utilizează în același fel ca și programul telnet, ceea ce nu constituie o surpriză, deoarece majoritatea programelor de navigare utilizează programul telnet ca aplicație auxiliară.

Pe scurt URL-urile au fost proiectate nu numai pentru a permite utilizatorilor să navegheze prin Web, dar și pentru a utiliza FTP, news, Gopher, e-mail și telnet, ceea ce face inutile interfețele specializate pentru aceste protocole integrând astfel într-un singur program, navigatorul în Web, aproape toate tipurile de acces în Internet. Dacă metoda nu ar fi fost proiectată de un fizician, ar fi putut să pară produsul departamentului de publicitate al unei companii de software.

În ciuda tuturor acestor proprietăți, creșterea Web-ului scoate în evidență și o slăbiciune a metodei utilizării URL-urilor. Pentru o pagină care este foarte des referită, ar fi de preferat să existe mai multe copii pe servere diferite, pentru a reduce traficul în rețea. Problema este că URL-urile nu oferă nici o posibilitate de indicare a unei pagini fără să se specifică unde este localizată pagina respectivă. Nu există nici o metodă pentru a spune ceva de genul: „Vreau pagina xyz, dar nu mă interesează de unde o aduci”. Pentru a rezolva această problemă și a permite multiplicarea paginilor, IETF lucrează la un sistem de **URN (Universal Resource Names - nume universale de resurse)**. Un URN poate să fie privit ca un URL generalizat. Acest subiect este în curs de cercetare, deși o propunere de sintaxă este dată în RFC 2141.

### Lipsa stării și utilizarea cookies

Așa cum am văzut în mod repetat, Web-ul este, în principiu, lipsit de stare. Nu există conceptul unei sesiuni de conectare. Programul de navigare transmite o cerere către server și primește un fișier. Apoi serverul uită că a discutat vreodată cu acel client.

La început, când Web-ul a fost folosit doar pentru obținerea de documente accesibile publicului larg, acest model era perfect adaptat cerințelor. Dar, pe măsură ce Web-ul a început să capete și alte funcții, acest model a dat naștere unor probleme. De exemplu, anumite situri de Web impun clientilor să se înregistreze (și chiar să plătească bani) spre a le utiliza. Ca atare, se pune întrebarea cum pot serverele să distingă între cereri din partea utilizatorilor înregistrați și a celorlalți. Un al doilea exemplu este comerțul electronic. Dacă un utilizator se plimbă printr-un magazin electronic, aruncând din când în când produse în coșul de cumpărături, cum poate serverul să rețină conținutul coșului? Un al treilea exemplu sunt portalurile de Web configurabile cum este Yahoo. Utilizatorii pot configura o pagină inițială detaliată, doar cu informația pe care o doresc (de ex.: valorile acțiunilor la bursă și echipele lor sportive favorite), dar cum poate serverul să afișeze pagina corectă dacă nu știe cine este utilizatorul?

La o primă vedere, se poate crede că serverele pot să urmărească utilizatorii uitându-se la adresele lor IP. Această idee nu funcționează însă. În primul rând, mulți utilizatori lucrează pe calculatoare partajate cu alții utilizatori, în special în cadrul companiilor, iar adresa IP identifică doar calculatorul nu și utilizatorul. În al doilea rând, mult mai grav, mulți dintre cei care oferă servicii de Internet (ISP) utilizează NAT, astfel încât toate pachetele ce pleacă de la orice utilizator folosesc aceeași adresă IP. Din punctul de vedere al serverului, cele câteva de mii de clienți ai unui ISP folosesc aceeași adresă IP.

Pentru a rezolva această problemă, Netscape a proiectat o tehnică mult criticată numită **cookies** (rom. fursecuri). Numele derivă dintr-un argou foarte vechi al programatorilor în care un program apelează o procedură și obține rezultate ce ar putea fi mai târziu pentru a executa ceva. În acest sens, o înregistrare de descriere a unui fișier UNIX sau un identificator al unui obiect Windows reprezintă un cookie. Mecanismul a fost formalizat mai târziu în RFC 2109.

Când un client cere o pagină de Web, serverul poate oferi informații adiționale odată cu pagina cerută. Aceste informații pot include un cookie, care este un fișier (sau șir de caractere) de dimensiune mică (cel mult 4 KB). Programele de navigare stochează cookie-urile oferite într-un director special pentru acestea pe discul clientului, cu excepția cazurilor când utilizatorul a opus utilizarea cookie-urilor. Cookie-urile sunt doar fișiere sau șiruri de caractere, nu programe executabile. În principiu, un cookie ar putea conține un virus, dar deoarece cookie-urile sunt tratate ca date, nu există nici o posibilitate oficială ca virusul să fie executat și să cauzeze probleme. Este însă posibil ca un hacker să exploateze o eroare a programului de navigare și să cauzeze activarea virusului.

Un cookie poate conține până la cinci câmpuri, așa cum se arată în fig. 7-25. Câmpul *Domeniu* spune de unde a sosit cookie-ul. Programele de navigare trebuie să verifice că serverele nu mint în legătură cu domeniul lor. Fiecare domeniu poate stoca cel mult 20 de cookie-uri pentru fiecare client. Câmpul *Cale* reprezintă o cale în structura de directoare a serverului care identifică ce parte a arborelui de fișiere de pe server poate utiliza cookie-ul respectiv. Adesea, acest câmp este /, ceea ce înseamnă întregul arbore.

Domeniu	Cale	Conținut	Expiră	Sigur
toms-casino.com	/	CustomerID=497793521	15-10-02 17:00	Da
joes-store.com	/	Cart1=1-00501;1-07031;2-13721	11-10-02 14:22	Nu
aportal.com	/	Prefs=Stk:SUNW+ORCL;Spt:Jets	31-12-10 23:59	Nu
sneaky.com	/	UserID=3627239101	31-12-12 23:59	Nu

Fig. 7-25. Câteva exemple de cookie-uri

Câmpul *Continut* are forma *nume = valoare*. Atât *nume* cât și *valoare* pot fi orice dorește serverul. Acesta este câmpul în care se stochează conținutul unui cookie.

Câmpul *Expiră* arată când expiră un cookie. Dacă acest câmp este absent, programul de navigare șterge cookie-ul la terminarea execuției. Un astfel de cookie se numește **cookie ne-persistent (non-persistent cookie)**. Dacă se oferă o dată și o oră, cookie-ul se numește **persistent** și este păstrat până la expirare. Timpul de expirare se dă pentru ora Greenwich (Greenwich Mean Time). Pentru a șterge un cookie de pe discul unui client, un server retransmite cookie-ul cu timpul de expirare în trecut.

În sfârșit, câmpul *Sigur* poate indica dacă programul de navigare poate transmite cookie-ul numai unui server sigur. Acest element este utilizat pentru comerțul electronic, aplicații bancare și alte aplicații sigure.

Am văzut cum se obțin cookie-urile, dar cum sunt ele utilizate? Chiar înainte ca un program de navigare să transmită cererea pentru o pagină către un sit Web, el verifică directorul de cookie-uri pentru a vedea dacă există vreun cookie care a fost stocat de către domeniul la care se duce cererea. În caz afirmativ, toate cookie-urile stocate de către acel domeniu sunt incluse în mesajul ce conține cererea. Atunci când serverul le obține, le poate interpreta în orice mod dorește.

Să examinăm acum câteva utilizări posibile pentru cookie-uri. În fig. 7-25, primul cookie a fost stocat de către *toms-casino.com* și este utilizat pentru a identifica utilizatorul. Când clientul se conectează săptămâna următoare pentru a arunca niște bani pe fereastră, programul de navigare transmite cookie-ul, astfel că serverul știe cine este clientul. Odată ce detine numărul de identificare al clientului, serverul poate căuta datele sale într-o bază de date și utilizează aceste informații pentru a construi o pagină de Web potrivită. Depinzând de obiceiurile clientului, această pagină poate reprezenta o masă de poker, o listă a curselor de cai din ziua respectivă sau o mașină de jocuri.

Cel de-al doilea cookie a venit de la *joes-store.com*. Scenariul în acest caz este că utilizatorul se plimbă prin magazin, căutând lucruri de cumpărat. Atunci când găsește un preț bun și execută un clic pe produsul respectiv, serverul construiește un cookie ce conține numărul de bucăți și codul produsului și îl transmite clientului. Pe măsură ce clientul continuă să se plimbe prin magazin, cookie-ul este întors la fiecare nouă pagină cerută. Pe măsură ce cumpărăturile se acumulează, serverul le adaugă la cookie. În figură, coșul de cumpărături conține trei produse, ultimul dintre ele în dublu exemplar. În cele din urmă, când clientul selectează *MERGI LA CASĂ*, cookie-ul, care acum conține lista completă de cumpărături este trimis odată cu cererea. În acest mod, serverul știe exact ce produse au fost cumpărate.

Cel de-al treilea cookie este pentru un portal de Web. Atunci când utilizatorul selectează o legătură către portal, programul de navigare transmite cookie-ul. Aceasta spune portalului să construiască o pagină ce conține valorile acțiunilor pentru Sun Microsystems și Oracle și rezultatele echipei de fotbal New York Jets. Deoarece un cookie poate avea până la 4 KB, există suficient spațiu pentru preferințe mai detaliate în ceea ce privește titluri de articole din ziar, starea vremii, oferte speciale, și.m.d.

Cookie-urile pot fi utilizate și în beneficiul serverului. De exemplu, să presupunem că un server dorește să știe în fiecare moment câți vizitatori a avut și câte pagini au fost vizitate de fiecare dintre aceștia înainte de a părăsi situl. Prima cerere nu va fi însoțită de nici un cookie, astfel că serverul va transmite înapoi un cookie ce conține *Counter=1*. Selectiile ulterioare ale paginilor acestui site vor transmite acest cookie înapoi la server. De fiecare dată, contorul este incrementat și transmis înapoi clientului. Urmărind aceste contoare, serverul poate să vadă câte persoane renunță după vizitarea primei pagini, câte au vizitat două pagini, și.m.d.

Cookie-urile au avut și utilizări greșite. Teoretic, cookie-urile ar trebui să ajungă doar la situl lor de origine, dar spărgătorii au exploatat numeroase erori în programele de navigare pentru a captura

cookie-uri care nu le erau destinate. Deoarece numeroase situri de comerț electronic pun numerele de cărți de credit în cookie-uri, potențialul pentru abuzuri este evident.

Un mod de utilizare controversat al cookie-urilor este colectarea, în secret, de informații privind obiceiurile de navigare pe Web ale utilizatorilor. Mecanismul funcționează în modul următor. O companie de publicitate, să spunem Sneaky Ads. (*rom. sneaky = viclean*) contactează un număr de situri de Web importante și pune anunțuri publicitare ale clienților săi pe paginile respectivelor situri, pentru care plătește celor ce dețin siturile o anumită sumă. În loc să ofere sitului un fișier GIF sau JPEG pentru a fi amplasat pe fiecare pagină, le oferă un URL ce trebuie adăugat la fiecare pagină. Fiecare din aceste URL-uri conține un număr unic în partea rezervată fișierului, cum ar fi

<http://www.sneaky.com/382674902342.gif>

Atunci când un utilizator vizitează o pagină *P* ce conține un asemenea anunț publicitar, programul de navigare obține fișierul HTML și vede legătura către imaginea de la [www.sneaky.com](http://www.sneaky.com), săcă să transmită cererea pentru imagine acestui server. Serverul întoarce un fișier GIF conținând anunțul publicitar, împreună cu un cookie ce conține un număr unic de identificare pentru utilizator, 36271239101 în fig. 7-25. Compania Sneaky înregistrează faptul că utilizatorul cu acest număr de înregistrare a vizitat pagina *P*. Aceasta este ușor de realizat având în vedere faptul că fișierul cerut (382674902342.gif) este menționat doar în pagina *P*. Desigur că anunțul în sine poate apărea pe o mie de alte pagini, dar de fiecare dată cu un nume de fișier diferit. Compania Sneaky colectează probabil doar câțiva bănuți de la compania ce a realizat produsul de fiecare dată când transmite anunțul publicitar.

Mai târziu, când utilizatorul vizitează o altă pagină de Web care conține unul din anunțurile publicitare ale companiei Sneaky, după ce programul de navigare a obținut fișierul HTML de la server, vede referința către, să spunem, <http://www.sneaky.com/493654919923.gif> și cere acest fișier. Deoarece există deja un cookie de la domeniul *sneaky.com*, programul de navigare include cookie-ul companiei Sneaky ce conține și numărul unic de identificare al utilizatorului. Compania Sneaky știe acum o a doua pagină vizitată de utilizator.

Cu timpul, compania Sneaky poate construi un profil complet al obiceiurilor de navigare ale utilizatorului, deși acesta nu a efectuat nici un clic pe vreun anunț publicitar. Desigur, acest profil nu include încă numele utilizatorului (deși conține adresa IP a acestuia, informație care poate fi suficientă pentru a deduce numele din alte baze de date). În cazul în care utilizatorul oferă vreodată numele său unui site care cooperează cu Sneaky, un profil complet ce include și numele este acum gata de vânzare pentru oricine dorește să-l cumpere. Vânzarea acestor informații poate fi suficient de profitabilă pentru ca Sneaky să poată plasa mai multe anunțuri publicitare pe mai multe situri Web și astfel să colecteze și mai multe informații. Cea mai ascunsă parte a acestei povestiri este că majoritatea utilizatorilor nu știu nimic despre această colectare de informații și chiar ar putea să se creadă în siguranță, pentru că nu au selectat nici un anunț publicitar.

Și dacă Sneaky vrea să fie și mai vicleană, anunțul publicitar poate să nu fie unul clasic. Un „anunț” format dintr-un singur pixel de culoarea fondului (și deci invizibil) are exact același efect ca și anunțul propriu-zis: programul de navigare trebuie să obțină imaginea gif de 1 x 1 pixeli și să îl livreze toate cookie-urile care au fost transmise de domeniul de origine al pixelului.

Pentru a menține impresia de intimitate, o serie de utilizatori își configurorează programele de navigare pentru a refuza toate cookie-urile. Această acțiune poate duce însă la probleme cu siturile de Web legitime ce folosesc cookie-uri. Pentru a rezolva această problemă, utilizatorii instalează câteodată software care „mânâncă cookie-uri” (*cookie-eating software*). Acestea sunt programe speciale care inspectează fiecare cookie la sosire și îl acceptă sau refuză în funcție de opțiunile utilizatorului

(de ex.: în ce situri Web se poate avea încredere). Această metodă oferă utilizatorului un control fin asupra căror cookie-uri sunt acceptate și care sunt refuzate. Programele de navigare moderne cum ar fi Mozilla ([www.mozilla.org](http://www.mozilla.org)) au un nivel de control complex asupra cookie-urilor.

### 7.3.2 Documente Web statice

Fundamentul Web-ului este transferul de pagini de Web de la server la client. În forma cea mai simplă, paginile de Web sunt statice, adică doar fișiere existente pe un server, ce așteaptă să fie cerute. În acest sens, chiar și o înregistrare video este o pagină de Web statică, deoarece este doar un fișier. În această secțiune vom privi paginile de Web statice în detaliu. În secțiunea următoare vom examina conținutul dinamic.

#### HTML--HyperText Markup Language

Paginile de Web sunt în prezent scrise într-un limbaj numit **HTML (HyperText Markup Language)**. HTML permite utilizatorilor să producă pagini de Web care conțin text, imagini și referințe la alte pagini de Web. HTML este un limbaj de marcare, un limbaj care descrie cum trebuie să fie formatare textele. Termenul de „marcare” provine din timpurile vechi, când editorii făceau marcaje pe documente pentru a indica tipografului - în acele timpuri un om - ce font-uri să folosească și.a.m.d. Limbajele de marcare conțin comenzi explicite pentru formatare. De exemplu, în HTML, **<b> înseamnă început de mod aldin, și </b>** înseamnă terminarea utilizării modului aldin. Avantajul utilizării unui limbaj de marcare față de unul în care nu se utilizează marcarea explicită constă din faptul că este simplu de scris un program de navigare care să interpreteze comenziile de marcare. TeX și troff sunt două exemple foarte cunoscute de limbaje de marcare.

Prin standardizarea și includerea comenziilor de marcare în fiecare fișier HTML, devine posibil ca orice program de navigare să poată să citească și să formeze orice pagină Web. Posibilitatea formatarii paginii receptioane este foarte importantă, deoarece o pagină poate să fie construită pe un ecran cu 1600 x 1200 pixeli utilizând culori codificate cu 24 de biți, dar s-ar putea să fie necesară afișarea într-o mică fereastră de pe un ecran cu 640 x 320 pixeli și utilizând culori codificate pe 8 biți.

În cele ce urmează se face o scurtă introducere în limbajul HTML, doar pentru a oferi o idee despre subiect. Cu toate că este posibil să se construiască documente HTML utilizând orice editor, și mulți fac asta, este posibil și să se utilizeze editoare HTML speciale care pot să facă toată munca (desigur, în mod corespunzător utilizatorul are mai puțin control asupra detaliilor produsului final).

O pagină Web corect formată conține o zonă de cap și o zonă de corp, cuprinse între marcajele (tag-uri) **<html>** și **</html>**, dar majoritatea programelor de navigare ignoră absența acestor marcaje. Așa cum se vede în fig. 7-26(a), capul este cuprins între marcajele **<head>** și **</head>**, iar corpul între marcajele **<body>** și **</body>**. Comenziile cuprinse între aceste marcaje se numesc **directive**. Majoritatea marcajelor HTML au acest format, adică, **<ceva>** pentru a indica începutul a ceva și **</ceva>** pentru a marca sfârșitul. Numeroase exemple de fișiere HTML sunt disponibile. Majoritatea programelor de navigare au o opțiune **VIEW SOURCE** (afișarea sursei) sau ceva similar. Selectarea acestei opțiuni afișează pagina curentă în format HTML în loc de forma interpretată.

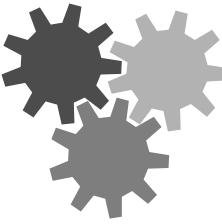
Marcajele pot să fie scrise cu litere mici sau mari. Adică **<head>** și **<HEAD>** înseamnă același lucru, dar versiuni mai noi ale standardului cer existența doar a primei forme. Cum este dispus textul în documentul HTML este nesemnificativ. Programele de navigare ignoră spațiile și trecerile la rând nou, deoarece textul trebuie să fie formatat pentru a corespunde zonei de afișare curente. Cores-

punzător, se pot utiliza spații pentru a face documentele HTML mai ușor de citit, ceva ce ar fi necesar pentru majoritatea documentelor. Liniile albe nu pot să fie utilizate pentru separarea paragrafeelor, deoarece sunt pur și simplu ignorate. Este necesară utilizarea unor marcaje explicite.

```
<html>
<head> <title> AMALGAMATED WIDGET, INC. </title></head>
<body> <h1> Welcome to AWI's Home Page </h1>
<img SRC="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's</b>
home page. We hope <i> you </i> will find all the information you need here.
<p> Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by FAX. </p>
<hr>
<h2> Product Information </h2>
<ul>
    <li> <a href="http://widget.com/products/big" > Big widgets </a>
    <li> <a href="http://widget.com/products/little" > Little widgets </a>
</ul>
<h2> Telephone Numbers </h2>
<ul>
    <li> By telephone: 1-800-WIDGETS,
    <li> By fax: 1-415-765-4321
</ul>
</body>
</html>
```

(a)

**Welcome to AWI's Home Page**



We are so happy that you have chosen to visit **Amalgamated Widget's** home page. We hope you will find all the information you need here.

Below we have links to information about our many fine products. You can order electronically (by WWW), by telephone, or by FAX.

---

**Product Information**

- [Big widgets](http://widget.com/products/big)
- [Little widgets](http://widget.com/products/little)

**Telephone numbers**

- 1-800-WIDGETS
- 1-415-765-4321

(b)

**Fig.7-26.** (a) Un exemplu simplu de pagină de Web. (b) Pagina formatată.

Unele marcaje au parametri (care au nume), numiți **attribute**. De exemplu:

```

```

este un maraj, `<img>`, având parametrul `src` cu valoarea `abc` și parametrul `alt` cu valoarea `foobar`. Pentru fiecare maraj, standardul HTML oferă o listă a parametrilor care pot să fie utilizati, dacă este cazul și care este semnificația lor. Deoarece parametrii au nume, ordinea în care se dau valorile parametrilor nu este semnificativă.

Din punct de vedere tehnic, documentele HTML sunt scrise utilizând setul de caractere ISO 8859-1 Latin-1, dar pentru utilizatorii ale căror tastaturi suportă numai codul ASCII, se pot utiliza secvențe de caractere pentru reprezentarea caracterelor speciale cum ar fi è. Lista caracterelor speciale este precizată în standard. Toate încep cu caracterul „&” și se termină cu „;”. De exemplu &egrave; produce è iar &eacute; produce é. Deoarece `< >` și `&` au semnificații speciale, pot să fie reprezentate numai utilizând secvențele speciale de caractere corespunzătoare, `&lt;`; `&gt;`; și `&amp;`.

Principalul element din zona de cap este titlul care este cuprins între `<title>` și `</title>`, dar aici pot să apară și alte tipuri de informații. Titlul nu este afișat pe pagină. Unele programe de navigare îl utilizează pentru a eticheta fereastra în care se afișează pagina respectivă.

Să analizăm și alte particularități prezente în exemplul din fig. 7-26. Toate marcajele utilizate în fig. 7-26 și încă alte câteva sunt prezentate în fig. 7-27. Titlurile de capitol sunt generate de marajul `<hn>`, unde *n* este o cifră între 1 și 6. `<h1>` este titlul cel mai important; `<h6>` este cel mai puțin important.

Marcaj	Descriere
<code>&lt;html&gt; ... &lt;/html&gt;</code>	Delimită textul scris în HTML
<code>&lt;head&gt; ... &lt;/head&gt;</code>	Delimită zona de cap
<code>&lt;title&gt; ... &lt;/title&gt;</code>	Definește titlul (nu este afișat de programul de navigare)
<code>&lt;body&gt; ... &lt;/body&gt;</code>	Delimită zona de corp
<code>&lt;h<i>n</i>&gt; ... &lt;/h<i>n</i>&gt;</code>	Delimită un titlu de nivel <i>n</i>
<code>&lt;b&gt; ... &lt;/b&gt;</code>	Textul ... o să fie afișat cu aldine
<code>&lt;i&gt; ... &lt;/i&gt;</code>	Textul ... o să fie afișat cu cursiv
<code>&lt;center&gt; ... &lt;/center&gt;</code>	Centrează ... pe pagină orizontal
<code>&lt;ul&gt; ... &lt;/ul&gt;</code>	Delimită o listă neordonată
<code>&lt;ol&gt; ... &lt;/ol&gt;</code>	Delimită o listă ordonată (numerotată)
<code>&lt;li&gt; ... &lt;/li&gt;</code>	Delimită un elemente într-o listă ordonată sau neordonată
<code>&lt;br&gt;</code>	Trecere la linie nouă
<code>&lt;p&gt;</code>	Început de paragraf
<code>&lt;hr&gt;</code>	Linie orizontală
<code>&lt;img src="..."&gt;</code>	Se încarcă o imagine
<code>&lt;a href="..."&gt; ... &lt;/a&gt;</code>	Se definește o hiper-legătură

Fig. 7-27. O selecție de marcaje uzuale. Unele mai au și alți parametri.

Depinde de programul de navigare să prezinte aceste titluri în mod diferit pe ecran. De obicei, titlurile cu număr mai mic vor fi afișate utilizând caractere mai mari. Programul de navigare poate să utilizeze culori diferite pentru fiecare nivel de titlu. De obicei, pentru titlurile marcate cu `<h1>` se utilizează litere mari scrise cu aldine cu cel puțin o linie liberă înainte și după. Corespunzător, pentru titlurile marcate cu `<h2>`, se utilizează caractere mai mici cu mai puțin spațiu lăsat înainte și după.

Marcajele `<b>` și `<i>` sunt utilizate pentru a indica modurile aldini și respectiv cursiv. Dacă programul de navigare nu poate să afișeze aceste tipuri de caractere, va utiliza un alt mod de a le reprezenta, de exemplu utilizând culori sau video-invers.

Limbajul HTML oferă diferite mecanisme pentru construirea de liste, inclusiv liste conținute în alte liste. Listele încep cu `<ul>` sau `<ol>`, `<li>` fiind folosit pentru a marca începutul elementelor în ambele cazuri. Marcajul `<ul>` indică începutul unei liste neordonate. Elementele individuale, care sunt marcate în sursă cu `<li>`, sunt reprezentate precedate de buline (•). O variantă a acestui mecanism este `<ol>`, care descrie o listă ordonată. Când se utilizează acest marcaj, textele precedate de `<li>` sunt numerotate de către programul de navigare. Marcajele `<ul>` și `<ol>` au aceeași sintaxă și efecte asemănătoare.

Marcajele `<br>`, `<p>` și `<hr>` indică o separare între diferențele părți ale textului. Formatul precis este descris în pagina de stil (vezi mai jos) asociată cu pagina curentă. Marcajul `<br>` forțează trecerea la linie nouă. De obicei programele de navigare nu inserează o linie liberă după `<br>`. Marcajul `<p>` reprezintă un început de paragraf, pentru care se va inseră o linie nouă și eventual se va face o indentare. (În mod teoretic, există și `</p>` pentru a indica sfârșitul de paragraf, dar este foarte rar utilizat; majoritatea celor care scriu în HTML nici nu știu că acest marcaj există). Ultimul marcaj, `<hr>`, forțează trecerea la linie nouă și desenează o linie orizontală.

HTML permite includerea de imagini în paginile de Web. Marcajul `<img>` arată că pe poziția curentă din pagină se va include o imagine. Marcajul poate să aibă o serie de parametri. Parametrul `src` indică URL-ul imaginii. Standardul HTML nu specifică ce formate grafice sunt permise. În practică, toate programele de navigare acceptă fișiere în format GIF, multe pot lucra și cu fișiere în format JPEG. Programele de navigare pot să lucreze și cu alte formate, dar o astfel de extensie este o sabie cu două tăișuri. Dacă, de exemplu un utilizator este obișnuit cu un program de navigare care suportă fișiere în format BMP, este posibil ca el să includă astfel de fișiere în pagina sa de Web și să fie uimit că alte programe de navigare ignoră imaginile sale minunate.

Alți parametri pentru `<img>` sunt `align`, care controlează modul în care se aliniaază imaginea față de limita de jos a textului (`top`, `middle`, `bottom`), `alt`, care furnizează textul afișat în locul imaginii dacă utilizatorul dezactivează opțiunea de afișare a imaginilor și `ismap`, un indicator care anunță că imaginea este o hartă selectabilă.

În sfârșit, să considerăm hiper-legăturile, care utilizează marcajele `<a>` (anchor) și `</a>`. Ca și `<img>`, `<a>` are diverse parametri, printre care `href` (URL-ul) și `name` (numele hiper-legăturii). Textul cuprins între `<a>` și `</a>` este afișat. Dacă este selectat, atunci se utilizează hiper-legătura pentru a se aduce o nouă pagină. În locul textului se poate pune și un marcaj `<img>`, caz în care, cu un clic pe imagine, se va activa legătura.

De exemplu, să considerăm următorul fragment HTML:

```
<a href="http://www.nasa.gov">NASA's home page </a>
```

Când se afișează acest fragment, pe ecran apare:

NASA's home page

Dacă utilizatorul execută un clic pe acest text, programul de navigare aduce și afișează pagina al cărei URL este `http://www.nasa.gov`.

Ca al doilea exemplu, să considerăm

```
<a href="http://www.nasa.gov"></a>
```

Când se afișează pagina va apărea o imagine (o navetă spațială). Executând un clic pe imagine, se va aduce pagina NASA, la fel ca și în cazul în care în exemplul anterior a fost selectat textul. Dacă utilizatorul a dezactivat opțiunea de afișare a imaginilor, atunci în loc de imagine va fi afișat textul NASA.

```

<html>
<head><title> A sample page with a table </title></head>
<body>
<table border=1 rules=all>
<caption> Some differences between HTML Versions </caption>
<col align=left>
<col align=center>
<col align= center >
<col align= center >
<col align= center >
<tr> <th>Item <th>HTML 1.0 <th>HTML 2.0 <th>HTML 3.0 <th> HTML 4.0 </tr>
<tr> <th> Hyperlinks <td> x <td> x <td> x <td> x </tr>
<tr> <th> Images <td> x <td> x <td> x <td> x </tr>
<tr> <th> Lists <td> x <td> x <td> x <td> x </tr>
<tr> <th> Active Maps and Images <td> &nbsp; <td> x <td> x <td> x </tr>
<tr> <th> Forms <td> &nbsp; <td> x <td> x <td> x </tr>
<tr> <th> Equations <td> &nbsp; <td> &nbsp; <td> x <td> x </tr>
<tr> <th> Toolbars <td> &nbsp; <td> &nbsp; <td> x <td> x </tr>
<tr> <th> Tables <td> &nbsp; <td> &nbsp; <td> x <td> x </tr>
<tr> <th> Accesibility features <td> &nbsp; <td> &nbsp; <td> &nbsp; <td> x </tr>
<tr> <th> Object embedding <td> &nbsp; <td> &nbsp; <td> &nbsp; <td> x </tr>
<tr> <th> Scripting <td> &nbsp; <td> &nbsp; <td> &nbsp; <td> x </tr>
</table>
</body>
</html>

```

(a)

**Some differences between HTML Versions**

<b>Item</b>	<b>HTML 1.0</b>	<b>HTML 2.0</b>	<b>HTML 3.0</b>	<b>HTML 4.0</b>
Hyperlinks	x	x	x	x
Images	x	x	x	x
Lists	x	x	x	x
Active Maps and Images		x	x	x
Forms		x	x	x
Equations			x	x
Toolbars			x	x
Tables			x	x
Accessibility features				x
Object embedding				x
Scripting				x

(b)

**Fig. 7-28.** (a) O tabelă HTML, (b) Un rezultat posibil.

Pentru marcajul `<a>` se poate utiliza parametrul *name* pentru a fixa o hiper-legătură, care să fie referită din pagină. De exemplu, unele pagini de Web încep cu o tabelă de conținut selectabilă. Prin execuția unui clic pe o intrare în tabela de conținut, se va trece direct la secțiunea corespunzătoare din pagină.

HTML evoluează continuu. În versiunile HTML 1.0 și HTML 2.0 nu existau tabele, dar au fost adăugate în HTML 3.0. O tabelă HTML este formată din una sau mai multe linii, fiecare fiind formată din una sau mai multe **celule**. Celulele pot să conțină diferite tipuri de informații, inclusiv text, figuri, iconițe, fotografii și chiar tabele. Celulele pot să fie alipite, de exemplu un titlu poate să se întândă peste mai multe coloane. Autorii paginilor au control asupra modului în care se face afișarea, inclusiv alinierea, stilul bordurii, marginile celulelor, dar programul de navigare este cel care hotărăște de fapt cum se face afișarea.

O descriere în HTML a unei tabele este prezentată în fig. 7-28(a), iar efectul posibil este prezentat în fig. 7-28(b). Acest exemplu prezintă câteva din facilitățile de descriere a tabelelor în HTML. Tabelele încep cu marcajul <table>. Se pot specifica informații suplimentare pentru a descrie proprietățile generale ale tabelei.

Marcajul <caption> poate să fie utilizat pentru a furniza un titlu tabelei. Fiecare linie începe cu marcajul <tr> (Table Row - linie în tabelă). Celulele individuale sunt marcate cu <th> (Table Header - titlu de coloană), <td> (Table Data - date în tabelă). Diferențierea este necesară pentru a permite programului de navigare să le afișeze diferit, aşa cum se vede și din exemplul considerat.

În tabele se pot utiliza alte marcaje. Acestea includ posibilitatea de a specifica alinieri orizontale sau verticale ale celulelor, alinierea în cadrul celulei, margini, gruparea de celule, unități și multe altele.

În HTML 4.0 au fost adăugate noi elemente. Acestea includ elemente ce fac paginile mai accesibile utilizatorilor cu handicap, înglobarea obiectelor (o generalizare a marcajului <img> astfel încât alte obiecte să poată fi înglobate în pagini), suport pentru limbaje de scripturi (pentru a permite conținut dinamic) și multe altele.

Când un sit de Web este complex, fiind format din multe pagini produse de autori diferiți ce lăză pentru aceeași companie, este adesea de dorit să existe o modalitate pentru a împiedica moduri de prezentare diferite în pagini diferite. Această problemă poate fi rezolvată utilizând **paginile de stil (style sheets)**. Atunci când se utilizează pagini de stil, paginile individuale nu mai folosesc stiluri fizice, cum sunt modurile aldin și cursiv. Autorii pot acum să utilizeze stiluri logice, cum sunt <dn> (definiție), <em> (evidențiere), <strong> (evidențiere accentuată) și <var> (variabile de program). Stilurile logice sunt definite în pagina de stil, pentru care există o referință la începutul fiecărei pagini. În acest fel, toate paginile au același stil și dacă administratorul sitului (*Webmaster*) decide să schimbe stilul <strong> din stil cursiv de 14 puncte tipografice, culoare albastră în stil aldin, 18 puncte tipografice, culoare roz tipător, tot ceea ce trebuie să facă este să schimbe o singură definiție pentru a converti întregul sit Web. O pagină de stil poate fi comparată cu o directivă #include într-un program C: schimbarea unei macrodefiniții în fișierul inclus determină schimbarea în toate fișierele program ce includ respectivul header.

## Formulare

HTML 1.0 funcționa într-o singură direcție. Utilizatorii puteau să aducă o pagină de la furnizorii de informație, dar era foarte dificil să se transmită informație în sens invers. Pe măsură ce tot mai multe organizații comerciale au început să utilizeze Web-ul, a apărut o puternică cerere pentru comunicația în dublu sens. De exemplu, multe companii vor să poată prelua comenzi pentru produse utilizând paginile lor de Web, furnizorii de software vor să distribuie programe prin intermediul Web-ului, clienții să își completeze fișele de înregistrare prin același mijloc, iar companiile care oferă servicii de căutare în Web au nevoie ca utilizatorii de servicii să poată să introducă cuvintele pe baza cărora se face căutarea.

Acest gen de cereri a dus la includerea **formularelor** începând cu HTML 2.0. Formularele conțin casete și butoane care permit utilizatorilor să completeze informații sau să facă selecții și apoi să transmită informațiile la proprietarul paginii. În acest scop se utilizează marcajul `<input>`. Acesta are o varietate de parametri care determină mărimea, tipul, și modul de afișare a casetei utilizate. Cele mai obișnuite sunt câmpuri în care utilizatorul poate să introducă text, casete care pot să fie selectate, hărți active, butoane *submit*. Exemplul din fig. 7-29 prezintă câteva dintre aceste posibilități.

```

<html>
<head><title> AWI CUSTOMER ORDERING FORM </title></head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/widgetorder" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street Address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size=4>
Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10>
expires <input name="expires" size=4>
M/C <input name="cc" type=radio value="mastercard">
VISA <input name="cc" type=radio value="visacard"> </p>
<p> Widget size Big <input name="product" type=radio value="expensive">
Little <input name="product" type=radio value="cheap">
Ship by express courier <input name="express" type=checkbox> </p>
<p> <input type=submit value="submit order"> </p>
Thank you for ordering an AWI widget, the best widget money can buy!
</form>
</body>
</html>

```

(a)

## Widget Order Form

Name	<input type="text"/>		
Street address	<input type="text"/>		
City	<input type="text"/>	State	<input type="text"/>
Country	<input type="text"/>		
Credit card #	<input type="text"/>	Expires	<input type="text"/>
M/C	<input type="radio"/>	Visa	<input type="radio"/>
Widget size	Big <input type="radio"/>	Little <input type="radio"/>	Ship by express courier <input type="checkbox"/>
<input type="button" value="Submit order"/>			
Thank you for ordering an AWI widget, the best widget money can buy!			

(b)

**Fig. 7-29.** (a) Un formular de comandă HTML. (b) Pagina formatată.

Să începem discuția parcurgând exemplul. Ca orice formular, și acesta este cuprins între marcajele `<form>` și `</form>`. Textele care nu sunt incluse între marcaje sunt afișate. Într-un formular poate să fie utilizat orice marcat obișnuit (de exemplu `<b>`) În acest formular sunt utilizate trei tipuri de caseți.

Prima casetă din formular apare după textul „Name”. Caseta are lățimea de 46 de caractere și utilizatorul va introduce un sir care va fi memorat în variabila *customer* pentru prelucrări ulterioare. Marcajul `<p>` indică programului de navigare să afișeze ceea ce urmează pe o linie nouă, chiar dacă mai este loc pe linia curentă. Utilizând `<p>` și alte marcaje care controlează dispunerea textului, creatorul paginii poate să controleze cum arată formularul pe ecran.

Pe linia următoare se solicită adresa utilizatorului, având cel mult 40 de caractere, pe o linie separată. Urmează o linie pe care se solicită orașul, statul și țara. Aici nu se utilizează marcajul `<p>`, deci programul de navigare o să le afișeze pe toate pe aceeași linie, dacă încap. Din punctul de vedere al programului de navigare, paragraful curent conține șase elemente: trei siruri alternând cu trei caseți. El le afișează pe aceeași linie de la stânga la dreapta, trecând la o linie nouă ori de câte ori pe linia curentă nu mai încape următorul element. Astfel, este posibil ca pe un ecran 1600 x 1200 să încapă toate cele trei siruri și casetele corespunzătoare, în timp ce pe un ecran 1024 x 768 ele pot să fie distribuite în două linii. În cazul cel mai defavorabil cuvântul „Country” este la capătul liniei, iar caseta asociată este la începutul liniei următoare. Nu există nici o posibilitate de a forța programul de navigare să afișeze caseta lângă text.

Următoarea linie solicită numărul cărtii de credit și data sa de expirare. Transmiterea numerelor cărtiilor de credit prin Internet trebuie să se facă numai dacă s-au luat măsurile de securitate adecvate. Vom discuta despre aceste măsuri în cap. 8.

După data de expirare, întâlnim un element nou: butoane radio. Acestea sunt utilizate atunci când trebuie să se facă o alegere între mai multe alternative. Modelul care se utilizează aici este cel al unui aparat de radio care are butoane pentru selecția scalelor. Programul de navigare afișează aceste caseți într-o formă care permite utilizatorului să le selecteze sau să le deselecteze prin execuția unui clic (sau utilizând tastatura). Selecția uneia dintre ele le deselectează pe toate celelalte care fac parte din același grup. Modul de afișare depinde de programul de navigare. „Widget size” utilizează de asemenea două butoane. Cele două grupuri sunt diferențiate prin câmpul *name* și nu printr-un domeniu static de genul `<radiobutton> ... </radiobutton>`.

Parametrii *value* sunt utilizati pentru a arăta care buton a fost apăsat. În funcție de opțiunea aleasă pentru cartea de credit, variabila *cc* va avea ca valoare sirul „mastercard” sau „visacard”.

După cele două seturi de butoane, urmează opțiunea referitoare la modul de transport, reprezentată de o casetă de tip *checkbox*. Aceasta poate să fie pe poziția selectată sau nu. Spre deosebire de cazul butoanelor radio, unde poate să fie selectat un singur buton dintr-un set, fiecare casetă de tip *checkbox* poate să fie selectată sau nu, independent de celelalte. De exemplu, când se comandă o piță utilizând pagina de Web Electropizza, utilizatorul poate să aleagă sardele și ceapă și ananas (dacă le suportă), dar nu poate să aleagă mică și medie și mare pentru aceeași piță. Conținutul corespunzător piței va fi reprezentat de trei butoane diferite de tip *checkbox*, în timp ce dimensiunea va fi reprezentată de un set de butoane radio.

Pe de altă parte, în cazul în care lista din care se face alegerea este foarte lungă, butoanele radio devin dificil de utilizat. Din acest motiv, marcajele `<select>` și `</select>` sunt utilizate pentru a prezenta o listă de alternative, utilizând semantica corespunzătoare unor butoane radio (dacă nu se utilizează parametrul *multiple*, caz în care semantica este cea de la casele de tip

(*checkbox*). Unele programe de navigare afișează opțiunile cuprinse între `<select>` și `</select>` ca un meniu derulant.

Am văzut jumătate din tipurile standard pentru marcajul `<input>`: *radio* și *checkbox*. De fapt, am văzut și un al treilea: *text*. Deoarece acesta este tipul implicit, nu am mai utilizat parametrul `type = text`, dar puteam să o facem. Alte două tipuri sunt *password* și *textarea*. O casetă *password* funcționează la fel ca o casetă *text*, numai că nu se face afișarea caracterelor introduse. O casetă *textarea* este similară unei casete *text*, numai că va conține mai multe linii.

Întorcându-ne la exemplul din fig. 7-29, urmează butonul *submit*. Când este selectat acest buton, informația introdusă de către utilizator este transmisă la calculatorul de pe care provine formularul. Similar altor tipuri, *submit* este un cuvânt cheie pe care îl înțelege programul de navigare. Sirul `value` reprezintă în acest caz eticheta butonului și se afișează. Toate casetele pot să aibă valori, dar am avut nevoie de această facilitate numai aici. Pentru casetele *text*, conținutul câmpului `value` este afișat o dată cu formularul, dar utilizatorul poate să îl modifice sau să îl steargă. Casetele *checkbox* și *radio* pot să fie inițializate, utilizând însă un parametru numit *checked* (deoarece parametrul `value` oferă un text, dar nu indică o selecție).

Atunci când utilizatorul selectează butonul „*submit order*”, programul de navigare împachetează informația colectată într-o singură linie lungă și o transmite serverului pentru procesare. Pentru a separa câmpurile, se utilizează caracterul `&`; caracterul `+` reprezintă spațiu. De exemplu, răspunsul la formular poate să arate ca în fig. 7-30:

(împărțit aici în trei linii din motive de aliniere pagină)

```
customer=John+Doe&address=100+Main+St.&city=White+Plain&
state=NY&country=USA&cardno=1234567890&expires6/98&cc=mastercard&
product=cheap&express=on
```

**Fig. 7-30.** Un răspuns posibil de la programul de navigare către server cu informațiile complete de utilizator

Sirul va fi transmis la server ca o singură linie, nu ca trei. Dacă un *checkbox* nu este selectat, el este omis din sir. Este problema serverului să interpreze sirul respectiv. Vom discuta cum se poate realiza acest lucru mai târziu în acest capitol.

### XML și XSL

Limbajul HTML, cu sau fără formulare, nu conferă nici o structură paginilor de Web. De asemenea, el amestecă conținutul cu informații despre formatul paginii. Pe măsură ce comerțul electronic și alte aplicații devin din ce în ce mai răspândite, apare o cerere din ce în ce mai mare pentru structurarea paginilor de Web și separarea conținutului de informațiile de format. De exemplu, un program care caută pe Web prețul cel mai bun al unei cărți sau al unui CD trebuie să analizeze multe pagini de Web căutând numele produsului și prețul său. Cu paginile de Web în format HTML este foarte dificil ca un program să își dea seama unde se află numele și unde se află prețul.

Din acest motiv, consorțiu W3C a dezvoltat îmbunătățiri ale limbajului HTML pentru a permite paginilor de Web să fie structurate în vederea procesării automate. Două limbiage noi au fost dezvoltate în acest scop. Mai întâi, **XML (eXtensible Markup Language)** descrie conținutul într-un mod

structurat și apoi, **XSL (eXtensible Style Language)** descrie formatul independent de conținut. Ambele limbaje reprezintă subiecte mari și complexe, ca atare scurta introducere care urmează atinge tangențial subiectul, deși ar trebui să ofere o idee despre modul cum funcționează.

Să considerăm documentul XML dat ca exemplu în fig. 7-31. Acesta definește o structură numită *book\_list*, care reprezintă o listă de cărți. Fiecare carte (*book*) are trei câmpuri, titlul, autorul și anul publicării. Aceste structuri sunt extrem de simple. Sunt permise structurile ce conțin câmpuri repetitive (de ex.: mai mulți autori), câmpuri optionale (de ex.: titlul CD-ROM-ului inclus) și câmpuri la alegere (de ex.: URL-ul unei librării dacă respectiva carte mai este disponibilă sau URL-ul unui sit de licitații dacă nu mai este disponibilă pe piață).

În acest exemplu, fiecare din cele trei câmpuri este o entitate indivizibilă, dar se permite subdiviziarea ulterioară a unui câmp. De exemplu, câmpul autor putea fi alcătuit aşa cum urmează, spre a oferi un control mai fin la căutare și formatare:

```
<author>
    <first_name>Andrew</first_name>
    <last_name>Tannenbaum</last_name>
</author>
```

Fiecare câmp poate fi împărțit în sub-câmpuri și sub-sub-câmpuri până la o adâncime arbitrară.

Întregul fișier din fig. 7-31 definește o listă de cărți ce conține trei cărți. Nu spune nimic despre modul cum se poate afișa pagina de Web pe ecran. Pentru a oferi informații de formatare avem nevoie de un alt fișier, *book\_list.xsl*, ce conține definiția XSL. Acest fișier este o pagină de stil care spune cum se poate afișa pagina. (Există alternative la paginile de stil, cum ar fi conversia XML în HTML, dar aceste alternative depășesc subiectul acestei cărți.)

```
<?xml version="1.0" ?>
<?xmlstylesheet type="text/xsl" href="book_list.xsl"?>

<book_list>
    <book>
        <title>Computer Networks, 4/e </title>
        <author> Andrew S. Tannenbaum </author>
        <year> 2003 </year>
    </book>
    <book>
        <title> Modern Operating Systems, 2/e </title>
        <author> Andrew S. Tannenbaum </author>
        <year> 2001 </year>
    </book>
    <book>
        <title> Structured Computer Organization 4/e </title>
        <author> Andrew S. Tannenbaum </author>
        <year> 1999 </year>
    </book>
</book_list>
```

**Fig. 7-31.** O pagină de Web simplă în format XML

```

<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
<html>
<body>
<table border="2">
<tr>
<th> Title </th>
<th> Author </th>
<th> Year </th>
</tr>
<xsl:for-each select="book_list/book">
<tr>
<td> <xsl:value-of select="title"/> </td>
<td> <xsl:value-of select="author"/> </td>
<td> <xsl:value-of select="year"/> </td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

**Fig. 7-32.** O pagină de stil în XSL

Un exemplu de fișier XSL pentru formatarea conținutului din fig. 7-31 este dat în fig. 7-32. După câteva declarații necesare ce includ URL-ul standardului XSL, fișierul conține marcaje începând cu `<html>` și `<body>`. Acestea definesc începutul unei pagini de Web, ca de obicei. Urmează apoi o definiție de tabel ce include titlurile celor trei coloane. Să observăm că în plus față de marcajele `<th>` există și marcaje `</th>`, lucru care nu ne-a preocupat până acum. Specificațiile XML și XSL sunt mult mai stricte decât specificația HTML. Ele statuează că reiectarea fișierelor incorecte din punct de vedere sintactic este obligatorie, chiar dacă programul de navigare poate determina ce a intenționat proiectantul paginii Web. Un program de navigare care acceptă fișiere XML sau XSL incorekte din punct de vedere sintactic și repară eroarea el însuși este neconform cu standardul și va fi reiectat la un test de conformanță cu standardele. Programelor de navigare li se permite însă să identifice exact eroarea. Această măsură întrucâtiva draconică este necesară pentru a rezolva problema numărului imens de pagini de Web scrise neglijent, care există în prezent.

#### Instrucțiunea

```
<xsl:for-each select="book_list/book">
```

este comparabilă cu o instrucțiune `for` în C. Execuția ei determină programul de navigare să execute corpul buclei (terminată de `</xsl:for-each>`) câte o dată pentru fiecare carte. Fiecare iterare afișează cinci linii: `<tr>`, titlul, autorul și anul și `</tr>`. După această buclă, sunt transmise la ieșire marcajele de închidere `</body>` și `</html>`. Rezultatul operației programului de navigare de a interpreta această pagină de stil este același ca și în cazul când pagina de Web ar fi conținut direct tabelul. Totuși, în acest format, programele pot analiza fișierul XML și găsi cu ușurință cărțile publicate du-

pă 2000, de exemplu. Merită subliniat faptul că deși fișierul nostru XSL conținea un fel de buclă, paginile de Web în XML și XSL sunt în continuare statice deoarece conțin pur și simplu instrucțiuni pentru programul de navigare despre modul de afișare a paginii, la fel ca și paginile HTML. Desigur, pentru a folosi XML și XSL, programul de navigare trebuie să fie capabil să interpreteze XML și XSL, dar marea majoritate a acestora au deja această capacitate. Nu este încă foarte clar dacă XSL va prelua paginile de stil tradiționale.

Nu am arătat cum se poate face acest lucru, dar limbajul XML permite proiectantului de situri Web să creeze fișiere de definiție în care structurile sunt definite în avans. Aceste fișiere de definiții pot fi incluse unele în altele, făcând posibilă construcția de pagini Web complexe. Pentru informații suplimentare despre aceasta și multe alte caracteristici ale limbajelor XML și XSL, consultați una din multele cărți despre acest subiect. Două exemple sunt (Livingston, 2002; și Williamson, 2001).

Înainte de a încheia discuția despre XML și XSL, merită să comentăm pe marginea luptei ideologice din interiorul consorțiului WWW și a comunității dezvoltatorilor de pagini Web. Scopul inițial al limbajului HTML era să specifică *structura* documentului și nu *modul de afișare*. De exemplu,

```
<h1> Deborah's Photos </h1>
```

instruiește programul de navigare să sublinieze titlul, dar nu spune nimic despre tipului font-ului, dimensiune sau culoare. Acestea sunt lăsate pe seama programului de navigare, care cunoaște proprietățile ecranului (de ex.: câți pixeli are). Totuși, mulți dezvoltatori de pagini Web doreau controlul absolut asupra modalității în care erau afișate paginile lor, astfel că au fost adăugate noi marcaje la HTML pentru a controla modul de afișare, cum ar fi

```
<font face="helvetica" size="24" color="red"> Deborah's Photos </font>
```

De asemenea, au fost adăugate modalități de a controla poziționarea precisă pe ecran. Această abordare este că nu este portabilă, ceea ce reprezintă desigur o problemă. Deși o pagină poate fi afișată perfect de programul de navigare cu care este dezvoltată, un alt program de navigare, sau chiar altă versiune a aceluiași program sau o altă rezoluție a ecranului poate fi un dezastru. XML este parțial o încercare de întoarcere la scopul originar de a specifica doar structura nu modalitatea de afișare a documentului. Totuși, XSL este oferit în plus, pentru a controla modul de afișare. Ambele formate pot avea însă utilizări eronate. Puteți conta pe asta.

XML poate fi utilizat și în alte scopuri decât acela de a descrie pagini de Web. O utilizare din ce în ce mai frecventă este aceea de limbaj de comunicare între aplicații. În particular, **SOAP (Simple Object Access Protocol – Protocol simplu pentru accesul la obiecte)** este o modalitate de a executa apeluri de tip RPC între aplicații într-un mod independent de limbaj și de sistem. Clientul construiește cererea ca mesaj XML și o transmite serverului, utilizând protocolul HTTP (descriș mai departe). Serverul trimite înapoi un răspuns sub formă de mesaj XML. În acest mod pot comunica aplicații de pe sisteme heterogene.

### XHTML – eXtended HyperText Markup Language

Limbajul HTML continuă să evolueze pentru a se conforma noilor cereri. Multe persoane din acest domeniu cred că în viitor majoritatea dispozitivelor cu acces la Web nu vor fi calculatoarele personale, ci dispozitive cu conexiuni fără fir, de tip PDA. Aceste dispozitive au memorie limitată pentru programe de navigare mari cu metode euristicice ce încearcă să trateze într-un anumit mod paginile de Web incorecte din punct de vedere sintactic. Astfel, următorul pas după HTML 4 este un limbaj care este Foarte Selectiv. Acest limbaj este numit **XHTML (eXtended HyperText Markup Language, rom.: Limbaj extins de marcaje hipertext)** mai degrabă decât HTML 5, pentru că, de

fapt, este HTML 4 reformulat în XML. Prin aceasta vrem să spunem că marcaje precum `<h1>` nu au nici o însemnatate prin ele însăși. Pentru a obține efectul din HTML 4 este nevoie de o definiție în fișierul XSL. XHTML este noul standard pentru Web și ar trebui folosit pentru toate paginile de Web noi pentru a asigura un maxim de portabilitate pe diverse platforme și programe de navigare.

Există șase diferențe majore și mai multe diferențe minore între XHTML și HTML 4. Să trecem acum în revistă diferențele majore. Mai întâi, paginile XHTML și programele de navigare trebuie să se supună în mod strict standardului. Gata cu paginile de Web de proastă calitate. Această proprietate a fost moștenită din XML.

În al doilea rând, toate marcajele și atributele trebuie să fie scrise cu litere mici. Marcaje ca `<HTML>` nu sunt valide în XHTML. Folosirea marcajelor precum `<html>` este acum obligatorie. Similar, `<img SRC="pic001.jpg">` este interzis pentru că are în componentă un atribut scris cu litere mari.

În al treilea rând, sunt necesare marcaje de terminare, chiar și pentru `</p>`. Pentru marcaje care nu au un marcat natural de terminare, cum ar fi `<br>`, `<hr>` și `<img>`, un caracter / trebuie să preceadă caracterul de terminare „>”, de exemplu

```

```

În al patrulea rând, atributele trebuie să fie conținute între ghilimele. De exemplu,

```
<img SRC="pic001.jpg" height=500 />
```

nu mai este permis. Valoarea 500 trebuie pusă între ghilimele, cum este numele fișierului JPEG, chiar dacă 500 este doar un număr.

În al cincilea rând, marcajele trebuie să se conțină unul pe altul într-un mod corespunzător. În trecut, acest lucru nu era necesar atât timp cât starea finală atinsă era corectă. De exemplu,

```
<center> <b> Vacation Pictures </b> </center>
```

era legal. În XHTML nu mai este. Marcajele trebuie închise în ordinea inversă în care au fost deschise.

În al șaselea rând, fiecare document trebuie să-și specifică tipul documentului. De exemplu, am văzut acest lucru în fig. 7-32. Pentru o discuție asupra schimbărilor, fie ele majore sau minore, vezi [www.w3.org](http://www.w3.org).

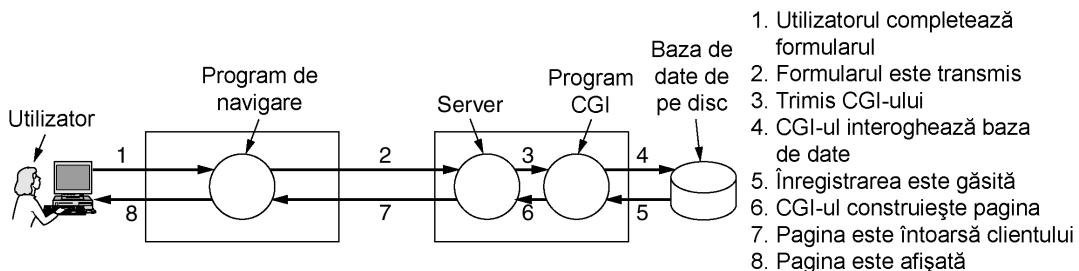
### 7.3.3 Documente Web dinamice

Până acum, modelul pe care l-am folosit este cel din Fig. 6-6: clientul transmite numele fișierului către server, care apoi întoarce fișierul. La începutul Web-ului, tot conținutul era de fapt static în acest mod (doar fișiere). Totuși, în ultimii ani, din ce în ce mai mult conținut a devenit dinamic, adică generat la cerere și nu doar stocat pe disc. Generarea de conținut poate avea loc fie la server, fie la client. Să examinăm acum pe rând fiecare din aceste cazuri.

#### Generare dinamică de pagini de Web la server

Pentru a vedea de ce este necesară generarea de conținut la server, să luăm în considerare utilizarea formularelor, aşa cum a fost descrisă mai devreme. Atunci când un utilizator completează un formular și apasă butonul *submit*, se transmite un mesaj către server, mesaj ce arată că are în interior conținutul unui formular, împreună cu acele câmpuri complete de utilizator. Acest mesaj nu este numele unui fișier ce trebuie întors. În schimb, acest mesaj trebuie să fie oferit unui program, sau

script, pentru a fi procesat. De obicei, procesarea implică folosirea informațiilor oferite de utilizator pentru căutarea unei înregistrări într-o bază de date de pe discul serverului și generarea unei pagini HTML personalizate pentru a fi trimisă înapoi clientului. De exemplu, într-o aplicație de comerț electronic, atunci când utilizatorul face un clic pe *MERGI LA CASĂ*, programul de navigare întoarce cookie-ul ce conține produsele din coșul de cumpărături, dar la server trebuie apelat un program, sau script, care procesează acest cookie și generează o pagină HTML ca răspuns. Pagina HTML ar putea afișa un formular ce conține lista de produse din coș și ultima adresă de expediere cunoscută a utilizatorului, împreună cu o cerere de verificare a informațiilor și de specificare a modalității de plată. Etapele necesare pentru procesarea informației dintr-un formular HTML sunt ilustrate în fig. 7-33.



**Fig. 7-33. Etapele de procesare a informației dintr-un formular HTML**

Modalitatea tradițională de a trata formularele și alte pagini de Web interactive este sistemul numit **CGI** (Common Gateway Interface – Interfață comună de conversie). Aceasta este o interfață standardizată ce permite serverelor de Web să discute cu programele din fundal și cu scripturile care acceptă o intrare (de ex.: formulare) și să genereze pagini HTML ca răspuns. De obicei, aceste programe de fundal sunt scripturi scrise în limbajul Perl deoarece scripturile Perl sunt mai ușor și mai rapid de scris decât programele (cel puțin dacă știi să programezi în Perl). În mod obișnuit, ele sunt localizate într-un director numit *cgi-bin*, care este vizibil în URL. Câteodată un alt limbaj de scripturi, Python, este utilizat în loc de Perl.

Ca un exemplu de cât de frecvent lucrează CGI, să considerăm cazul unui produs al companiei Truly Great Products Company (*rom. Compania Produselor cu Adevarat Bune*) care vine cu o fișă de înregistrare a produsului pentru garanție. În loc de a completa această fișă, clientului i se spune să meargă la [www.tgpc.com](http://www.tgpc.com) pentru a se înregistra on-line. Pe acea pagină există o hiper-legătură care spune

Apăsați aici pentru a va înregistra produsul

Această legătură indică spre un script Perl, să spunem [www.tgpc.com/cgi-bin/reg.perl](http://www.tgpc.com/cgi-bin/reg.perl). Când acest script este executat fără parametri, transmite înapoi o pagină HTML conținând formularul de înregistrare. Atunci când utilizatorul completează formularul și face un clic pe *submit* se transmite un mesaj acestui script, mesaj ce conține valorile completeate după modelul din fig. 7-30. Scriptul Perl analizează parametrii, adaugă o înregistrare în baza de date pentru noul client și transmite înapoi o pagină HTML ce conține numărul de înregistrare și un număr de telefon de la serviciul de asistență. Aceasta nu este singura modalitate de a trata formularele, dar este o modalitate des întâlnită. Există un număr mare de cărți despre scrierea scripturilor CGI și programarea în Perl. Câteva exemple sunt: (Hanegan, 2001; Lash, 2002; și Meltzer și Michalski, 2001).

Scripturile CGI nu sunt singura modalitate de a genera conținut dinamic la server. O altă modalitate des întâlnită este de a îngloba mici scripturi în paginile HTML și a lăsa serverul să le execute pentru a genera pagina. Un limbaj popular pentru scrierea acestor scripturi este **PHP (PHP: Hypertext Processor; rom.: Procesor Hipertext)**. Pentru a fi folosit, serverul trebuie să înțeleagă PHP (exact cum programul de navigare trebuie să înțeleagă XML pentru a interpreta paginile de Web scrise în XML). De obicei, serverele se așteaptă ca paginile de Web ce conțin PHP să aibă extensia *php* mai degrabă decât *html* sau *htm*.

Un mic script PHP este ilustrat în fig. 7-34; ar trebui să funcționeze pe orice server care are PHP instalat. Conține HTML normal cu excepția scriptului PHP dintre marcajele <?php ... ?>. Ceea ce generează este o pagină de Web ce afișează ceea ce știe despre programul de navigare care o apelează. Programele de navigare trimit de obicei o serie de informații odată cu cererea lor (și orice cookie aplicabil) și această informație este pusă în variabila *HTTP\_USER\_AGENT*. Când acest program este pus în fișierul *test.php* în directorul de WWW al companiei ABCD, tastând URL-ul *www.abcd.com/test.php* se va afișa o pagină de Web care spune utilizatorului ce program de navigare, ce limbă și ce sistem de operare folosește.

```
<html>
<body>

<h2> This is what I know about you </h2>
<?php echo $HTTP_USER_AGENT ?>

</body>
</html>
```

Fig. 7-34. O pagină HTML cu PHP înglobat

PHP este folositor în special la tratarea formularelor și este mai simplu de utilizat decât scripturile CGI. Ca un exemplu al modului său de funcționare, să considerăm exemplul din fig. 7-35(a). Această figură conține o pagină HTML normală cu un formular în interior. Singurul lucru neobișnuit la această pagină este prima linie, care spune că fișierul *action.php* va fi invocat pentru a trata parametrii după ce utilizatorul a completat și transmis formularul. Pagina afișează două căsuțe de text, una cerând numele și cealaltă vîrstă. După ce aceste căsuțe au fost completate și formularul transmis, serverul analizează sirul de caractere de forma celui din fig. 7-30, punând numele în variabila *name* și vîrsta în variabila *age*. Începe apoi să proceseze fișierul *action.php*, arătat în fig. 7-35(b) pentru obținerea răspunsului. În timpul procesării acestui fișier sunt executate comenzi PHP. Dacă utilizatorul a completat valorile „Barbara” și „24” în căsuțele formularului, fișierul transmis înapoi este cel dat în fig. 7-35(c). Astfel, tratarea formularelor devine extrem de simplă în PHP.

Deși PHP este ușor de utilizat, este de fapt un limbaj de programare puternic, orientat pe interfațarea dintre Web și o bază de date a serverului. Suportă variabile, siruri de caractere, vectori și majoritatea structurilor de control din C, dar un sistem de I/E mult mai puternic decât *printf*. PHP este un program public (open source) și disponibil gratuit. A fost conceput special să lucreze bine cu Apache, care este de asemenea gratuit și care este cel mai larg utilizat server de Web din lume. Pentru mai multe informații despre PHP, vezi (Valade, 2002).

Am văzut până acum două moduri diferite de a genera pagini HTML dinamice: script-urile CGI și PHP-ul înglobat. Există și o a treia tehnică, numită **JSP (JavaServer Pages)**, care este similară cu PHP, cu excepția faptului că partea dinamică este scrisă în limbajul de programare Java în loc de

PHP. Paginile ce folosesc această tehnică au în numele fișierului extensia *jsp*. O altă tehnică, **ASP (Active Server Pages)**, este versiunea de la Microsoft a PHP și JavaServer Pages. Pentru generarea conținutului dinamic folosește limbajul de script proprietar al Microsoft-ului, Visual Basic Script. Paginile ce folosesc această tehnică au extensia *asp*. Alegerea dintre *PHP*, *JSP*, și *ASP* are în general mai mult de-a face cu politici (open-source vs. Sun vs. Microsoft) decât cu tehnologia, cele trei limbi-je fiind comparabile. Colecția de tehnologii pentru generarea din zbor a conținutului este uneori denumită **HTML dinamic**.

### Generare dinamică de pagini de Web la client

Scripturile CGI, PHP, JSP și ASP rezolvă problema formularelor și a interacțiunilor cu bazele de date din server. Toate pot să accepte informații care vin din formulare, să caute informații într-o sau mai multe baze de date și să genereze pagini HTML cu rezultate. Ceea ce nu poate face nici unul dintre scripturi este să răspundă la mișcările mouse-ului sau să interacționeze direct cu utilizatorii. În acest scop, este necesar ca scripturile să fie înglobate în paginile HTML care sunt executate mai degrabă pe mașina clientului, decât pe mașina serverului. Începând cu HTML 4.0, astfel de scripturi erau permise folosind marcajul <script>. Cel mai popular limbaj de script la client este **JavaScript**, așa că o să aruncăm o scurtă privire asupra lui.

```
<html>
<body>
<form action="action.php" method="post">
<p> Introduceti numele: <input type="text" name="name"> </p>
<p> Introduceti varsta: <input type="text" name="age"> </p>
<input type="submit">
</form>
</body>
</html>
```

(a)

```
<html>
<body>
<h1> Raspuns: </h1>
Hello <?php echo $name; ?>.
Prezicere: anul urmator veti avea <?php echo $age + 1; ?> ani.
</body>
</html>
```

(b)

```
<html>
<body>
<h1> Raspuns: </h1>
Buna, Barbara.
Prezicere: anul urmator veti avea 25 ani.
</body>
</html>
```

(c)

**Fig. 7-35.** (a) O pagină Web ce conține un formular.

(b) Un script PHP pentru afișarea dinamică a form-ului.

(c) Ieșirea scriptului PHP când datele de intrare sunt „Barbara” și respectiv 24.

```
<html>
<head>
<script language="javascript" type="text/javascript">
function response(test_form) {
    var person = test_form.name.value;
    var years = eval(test_form.age.value) + 1;
    document.open();
    document.writeln("<html> <body>");
    document.writeln("Hello " + person + ".<br>");
    document.writeln("Prezicere: la anul vei avea " + years + ".");
    document.writeln("</body> </html>");
    document.close();
}
</script>
</head>

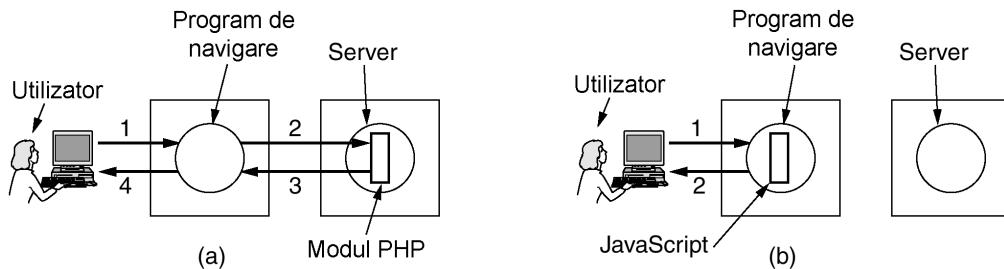
<body>
<form>
Introduceti numele: <input type="text" name="name">
<p>
Introduceti varsta: <input type="text" name="age">
<p>
<input type="button" value="submit" onclick="response(this.form)">
</form>
</body>
</html>
```

**Fig. 7-36.** Folosirea JavaScript Pentru procesarea unui formular.

JavaScript este un limbaj de script, *foarte* inspirat din câteva idei ale limbajului de programare Java. Nu este cu siguranță Java. Ca alte limbaje de script, el este un limbaj de nivel foarte înalt. De exemplu, într-o singură linie din JavaScript este posibil să se creeze o fereastră de dialog, să se aștepte introducerea de text și să se memoreze sirul rezultat într-o variabilă. Astfel de caracteristici de nivel înalt fac din JavaScript un limbaj ideal pentru crearea de pagini Web interactive. Pe de altă parte, faptul că nu este standardizat și că se modifică mai repede ca o muscă prinse într-o mașină cu raze X, fac extrem de dificilă scrierea de programe JavaScript care să funcționeze pe toate platforme, dar poate într-o zi se va stabiliza.

Un exemplu de program în JavaScript, este cel din fig. 7-36. Ca și în fig. 7-35(a), apare un formular în care se cer numele și vârsta și care calculează vârsta persoanei în anul următor. Corpul este aproape la fel ca în exemplul PHP, principala diferență fiind declararea butonului de trimis a datelor și asocierea unei funcții cu acest buton. Această funcție spune programului de navigare să invoke scriptul *response* la o apăsare de buton și să-i trimită formularul ca parametru.

Complet nouă aici este declararea funcției JavaScript *response* în antetul fișierului HTML, o zonă în mod normal rezervată titlurilor, culorilor de fundal și aşa mai departe. Această funcție extrage valoarea câmpului *name* din formular și o păstrează ca sir în variabila *person*. De asemenea extrage valoarea câmpului *age*, o convertește la un întreg prin folosirea funcției *eval*, o incrementează cu 1 și reține rezultatul în *years*. Apoi deschide un document pentru ieșire în care face trei scrieri, folosind metoda *writeln*, și închide documentul. Documentul este un fișier HTML, după cum se poate vedea din diversele marcaje HTML din el. Programul de navigare afișează apoi documentul pe ecran.



**Fig. 7-37.** (a) Scripting la server cu PHP. (b) Scripting la client cu JavaScript.

Este foarte important de înțeles că în timp ce fig. 7-35 și fig. 7-36 arată similar, ele sunt procesate total diferit. În fig. 7-35, după ce utilizatorul a apăsat butonul *submit*, programul de navigare strângе informația într-un sir lung, în stilul celui din fig. 7-30 și îl trimite serverului care a trimis pagina. Serverul vede numele fișierului PHP și îl execută. Scriptul PHP produce o nouă pagină HTML și acea pagină este trimisă înapoi programului de navigare pentru afișare. Cu fig. 7-36, când butonul *submit* este apăsat, programul de navigare interpretează o funcție JavaScript conținută pe pagină. Totul este făcut local, în programul de navigare. Nu se face nici un contact cu serverul. Ca o consecință, rezultatul este tipărit teoretic instantaneu, în timp ce cu PHP, poate exista o întârziere de câteva secunde înainte ca HTML-ul rezultat să ajungă la client. Diferența între utilizarea scripturilor la server și utilizarea acestora la client este ilustrată în fig. 7-37, inclusiv pașii implicați. În ambele cazuri, pașii numeroși încep după afișarea formularului. Pasul 1 constă din acceptarea datelor de intrare ale utilizatorului. Apoi urmează procesarea acestora, care diferă în cele două cazuri.

```

<html>
<head>
<script language="javascript" type="text/javascript">
function response(test_form) {
    function factorial(n) { if (n==0) return 1; else return n * factorial(n-1); }
    var r = eval(test_form.number.value);           // r = argument introdus de la tastatura
    document.myform.mytext.value = "Aici sunt rezultatele.\n";
    for (var i = 1; i <= r; i++)                  // tipărește o linie de la 1 la r
        document.myform.mytext.value += (i + "!=" + factorial(i) + "\n");
}
</script>
</head>

<body>
<form name="myform">
Introduceti un numar: <input type="text" name="number">
<input type="button" value="calcul factorial" onclick="response(this.form)">
<p>
<textarea name="mytext" rows=25 cols=50> </textarea>
</form>
</body>
</html>

```

**Fig. 7-38.** Un program JavaScript pentru calculul și afișarea factorialului.

Această diferență nu înseamnă că JavaScript este mai bun ca PHP. Utilizările lor sunt complet diferite. PHP (și, prin implicație, JSP și ASP) sunt utilizate când este necesară o interacțiune cu o bază de date aflată la distanță. JavaScript este utilizat când interacțiunea se face cu utilizatorul la calculatorul clientului. Este cu siguranță posibil (și des întâlnit) să existe pagini HTML care folosesc atât PHP cât și JavaScript, deși evident nu pot face același lucru pe același buton, sau să dețină același buton.

JavaScript este un limbaj de programare matur, cu toată puterea limbajelor C și Java. Are variabile, șiruri, vectori, obiecte, funcții, și toate structurile de control obișnuite. Are, de asemenea, un număr mare de facilități specifice paginilor Web, inclusiv abilitatea de a lucra cu ferestre și cadre, setarea și obținerea de cookie-uri, lucrul cu formulare, și cu hiper-legături. Un exemplu de program JavaScript care utilizează o funcție recursivă este dat în fig. 7-38.

JavaScript poate, de asemenea, să urmărească mișcarea mouse-ului peste obiectele afișate. Multe pagini Web ce conțin JavaScript au proprietatea că atunci când mouse-ul se mișcă peste un text sau o imagine, se întâmplă ceva. Deseori, imaginea se schimbă sau apare dintr-o dată un meniu. Acest tip de comportament este ușor de programat în JavaScript și conduce la pagini de Web foarte dinamice. În fig. 7-39 este dat un exemplu.

```
<html>
<head>
<script language="javascript" type="text/javascript">
if (!document.myurl) document.myurl = new Array();
document.myurl[0] = "http://www.cs.vu.nl/ast/im/kitten.jpg";
document.myurl[1] = "http://www.cs.vu.nl/ast/im/puppy.jpg";
document.myurl[2] = "http://www.cs.vu.nl/ast/im/bunny.jpg";
function pop(m) {
    var urx = "http://www.cs.vu.nl/ast/im/cat.jpg";
    popupwin = window.open(document.myurl[m], "mywind", "width=250,height=250");
}
</script>
</head>
<body>
<p><a href="#" onMouseover="pop(0); return false;"> Kitten </a> </p>
<p><a href="#" onMouseover="pop(1); return false;"> Puppy </a> </p>
<p><a href="#" onMouseover="pop(2); return false;"> Bunny </a> </p>
</body>
</html>
```

**Fig. 7-39.** O pagină Web interactivă care răspunde la mișcarea mouse-ului.

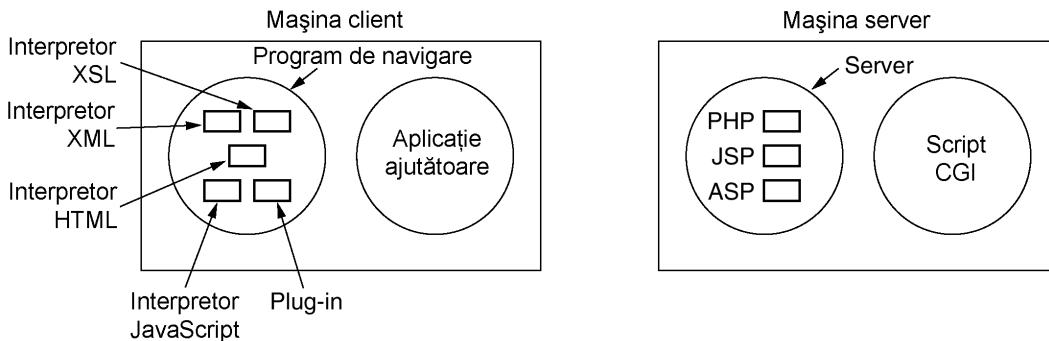
JavaScript nu este singura cale de a face paginile Web foarte interactive. Altă metodă populară este bazată pe folosirea **applet-urilor**. Acestea sunt mici programe Java care au fost compilate într-un cod mașină pentru un calculator virtual numit **JVM (Java Virtual Machine – Mașina Virtuală Java)**. Applet-urile pot fi incluse în paginile HTML (între <applet> și </applet>) și sunt interpretate de programe de navigare care cunosc JVM. Deoarece applet-urile nu sunt executate, ci interpretate, interpretorul Java poate să le împiedice să facă Lucruri Rele. Cel puțin în teorie. În practică, autorii de applet-uri au găsit și exploatat un sir aproape nesfărșit de erori în bibliotecile Java de I/E.

Răspunsul Microsoft la applet-urile Java de la Sun au fost paginile Web cu **controale Active-X (Active-X controls)**, care sunt programe compilate pentru o mașină Pentium și sunt executate direct

în hardware. Această proprietate le face mult mai rapide și mai flexibile decât applet-urile interpretate, pentru că pot face orice poate face un program. Când Internet Explorer vede un control Active-X într-o pagină Web, îl descarcă, îi verifică identitatea și îl execută. Totuși, descărcarea și execuția de programe străine ridică probleme de securitate, la care ne vom referi în cap. 8.

Din moment ce aproape toate programele de navigare pot să interpreteze atât programe Java cât și JavaScript, un programator care vrea să facă o pagină Web foarte interactivă va putea să aleagă între două tehnici, iar dacă nu se dorește portabilitatea pe mai multe platforme, poate să aleagă și Active-X. Ca o regulă generală, programele JavaScript sunt mai ușor de scris, applet-urile Java se execută mai rapid iar controalele Active-X cel mai rapid dintre toate. De asemenea, din moment ce toate programele de navigare implementează exact aceeași JVM, dar nu există două programe de navigare care să știe aceeași versiune de JavaScript, applet-urile Java sunt mai portabile decât programele JavaScript. Pentru mai multe detalii despre JavaScript, există multe cărți, fiecare cu multe (deseori peste 1000) pagini. Câteva exemple sunt (Easttom, 2001; Harris 2001; și McFerdries, 2001).

Înainte de a părăsi subiectul conținutului dinamic al Web-ului, să recapitulăm pe scurt ce am atins până acum. Pagini Web complete se pot genera din mers, folosind diverse script-uri pe mașina server. Odată ce sunt primite de programul de navigare, ele sunt tratate ca pagini HTML normale și sunt doar afișate. Script-urile pot fi scrise în Perl, PHP, JSP sau ASP, după cum este arătat în fig. 7-40.



**Fig. 7-40.** Diverse moduri de a genera și afișa conținut.

Generarea conținutului dinamic este posibilă și în partea clientului. Paginile Web pot fi scrise în XML și convertite la HTML conform unui fișier XSL. Programele JavaScript pot să efectueze diverse calcule. În sfârșit, plug-in-urile (plug-ins) și aplicațiile ajutătoare (helper applications) pot fi folosite pentru afișarea conținutului într-o varietate de forme.

### 7.3.4 HTTP – HyperText Transfer Protocol

Protocolul de transfer utilizat pe Web este **HTTP (HyperText Transfer Protocol, rom.:Protocol de Transfer al Hipertextului)**. Acesta specifică ce mesaje pot trimite clientii către servere și ce răspunsuri primesc înapoi. Fiecare interacțiune constă dintr-o cerere ASCII, urmată de un răspuns MIME conform RFC 822. Toți clientii și toate serverele trebuie să se supună acestui protocol. Este definit în RFC 2616. În această secțiune vom trata câteva din proprietățile sale cele mai importante.

## Conexiuni

Modul ușual prin care un program de navigare contactează un server este de a stabili o conexiune TCP pe portul 80 pe mașina serverului, deși această procedură nu este cerută formal. Avantajul de a folosi TCP este că nici programele de navigare și nici serverele nu trebuie să se preocupe de mesajele pierdute, mesajele duplicate, mesajele lungi, sau mesajele de confirmare. Toate aceste aspecte sunt tratate de implementarea TCP.

În HTTP 1.0, după ce conexiunea era stabilită, o singură cerere era transmisă și un singur răspuns era primit înapoi. Apoi conexiunea TCP era eliberată. Într-o lume în care pagina tipică Web constă în întregime din text HTML, această metodă era adecvată. În câțiva ani însă, o pagină medie Web conținea un număr mare de iconițe, imagini și alte lucruri plăcute vederii, astfel că stabilirea unei conexiuni TCP pentru a prelua o singură iconiță a devenit un mod foarte costisitor de a opera.

Această observație a dus la apariția HTTP 1.1, care suportă **conexiuni persistente**. Cu ele, este posibilă stabilirea unei conexiuni TCP, trimiterea unei cereri și obținerea unui răspuns, apoi trimiterea unor cereri adiționale și obținerea de răspunsuri adiționale. Prin distribuirea pornirii și eliberării unei conexiuni TCP peste mai multe cereri, supraîncărcarea relativă datorată TCP-ului este mult mai mică pe fiecare cerere. Este de asemenea posibilă trimiterea cererilor prin mecanismul pipeline, adică trimiterea cererii 2 înainte ca răspunsul la cererea 1 să fi sosit.

## Metode

Cu toate că HTTP a fost proiectat pentru utilizarea în Web, el a fost creat intenționat mai general decât era necesar în perspectiva aplicațiilor orientate pe obiecte. Pentru aceasta sunt suportate operațiile, denumite **metode**, care fac mai mult decât cele care doar cer o pagină Web. Această considerație generală a permis apariția SOAP. Fiecare cerere constă din una sau mai multe linii de text ASCII, în care primul cuvânt din prima linie este numele metodei cerute. Metodele incorporate sunt listate în fig. 7-41. Pentru accesarea unor obiecte generale, metode adiționale specifice obiectelor pot fi de asemenea disponibile. În numele metodelor este semnificativă utilizarea literelor mari/mici, de exemplu *GET* este o metodă acceptată, dar nu și *get*.

Metoda	Descriere
GET	Cerere de citire a unei pagini Web
HEAD	Cerere de citire a antetului unei pagini de Web
PUT	Cerere de memorare a unei pagini de Web
POST	Adăugarea la o resursă anume (de exemplu o pagină de Web)
DELETE	Ștergerea unei pagini de Web
TRACE	Tipărirea cererii care a sosit
CONNECT	Rezervat pentru o utilizare în viitor
OPTIONS	Interrogarea anumitor opțiuni

Fig. 7-41. Metode de cerere standard pentru HTTP.

Metoda *GET* cere serverului să transmită pagina (prin care noi înțelegem obiect, în cel mai general caz, dar în practică de obicei doar un fișier). Pagina este codată corespunzător în MIME. Marea majoritate a cererilor către servere Web sunt metode *GET*. Forma ușuală a metodei *GET* este

GET fișier HTTP-1.1

unde *fișier* denumește resursa (fișierul) ce va fi adusă, și 1.1 este versiunea de protocol utilizat.

Metoda *HEAD* cere doar antetul mesajului, fără să ceară și pagina propriu-zisă. Această metodă poate să fie utilizată pentru a afla când s-a făcut ultima modificare, pentru a obține informații pentru indexare, sau numai pentru a verifica corectitudinea unui URL.

Metoda *PUT* este inversă metodei *GET*: în loc să citească o pagină, o scrie. Această metodă permite crearea unei colecții de pagini de Web pe un server la distanță. Corpul cererii conține pagina. Pagina poate să fie codificată utilizând MIME, caz în care liniile care urmează după *PUT* pot include *Content-Type* și antete de autentificare, pentru a demonstra că într-adevăr cel care face cererea are dreptul de a realiza operația cerută.

Similară metodei *PUT* este metoda *POST*. Îf ea conține un URL, dar în loc să înlocuiască date existente, noile date se vor adăuga într-un mod generalizat. De exemplu, se poate transmite un mesaj la un grup de știri sau adăuga un fișier la un sistem de informare în rețea. În practică, nici *PUT* și nici *POST* nu sunt utilizate prea mult.

*DELETE* realizează ce era de așteptat: ștergerea unei pagini. Ca și la *PUT*, autentificarea și drepturile de acces joacă aici un rol important. Nu există nici o garanție asupra succesului operației *DELETE*, deoarece chiar dacă serverul dorește să execute ștergerea, fișierul poate să aibă atrbute care să interzică serverului HTTP să îl modifice sau să îl șteargă.

Metoda *TRACE* este pentru verificarea corectitudinii. Ea cere serverului să trimită înapoi cererea. Această metodă este utilă când cererile nu sunt procesate corect și clientul vrea să știe ce fel de cerere a ajuns de fapt la server.

Metoda *CONNECT* nu este utilizată în prezent. Este rezervată pentru utilizări ulterioare.

Metoda *OPTIONS* asigură o modalitate pentru client de a interoga serverul despre proprietățile acestuia sau despre cele ale unui anumit fișier.

Fiecare cerere obține un răspuns ce constă din linia de stare și posibile informații suplimentare (de exemplu, o parte sau toată pagina Web). Linia de stare conține un cod de stare de trei cifre, indicând dacă cererea a fost satisfăcută și dacă nu, cauza. Prima cifră este utilizată pentru împărțirea răspunsurilor în cinci mari grupuri, ca în fig. 7-42. Codurile 1xx sunt utilizate în practică foarte rar. Codurile 2xx indică tratarea cu succes a cererii și conținutul (dacă există) este returnat. Codurile 3xx spun clientului să caute în altă parte, prin folosirea unui URL diferit, sau în propria memorie ascunsă (discutată mai târziu). Codurile 4xx indică insuccesul cererii din cauza unei erori la client, precum o cerere invalidă sau o pagină inexistentă. În fine, erorile 5xx indică o problemă în server, datorată codului său sau unei supraîncărcări temporare.

Cod	Semnificație	Exemple
1xx	Informatie	100 = serverul acceptă tratarea cererii de la client
2xx	Succes	200 = cerere reușită; 204 = nu există conținut
3xx	Redirectare	301 = pagină mutată; 304 = pagina din memoria ascunsă este încă validă
4xx	Eroare la client	403 = pagină interzisă; 404 = pagina nu a fost găsită
5xx	Eroare la server	500 = eroare internă la server; 503 = încearcă mai târziu

Fig. 7-42. Grupuri de răspunsuri ale codurilor de stare.

### Antete de mesaje

Linia de cerere (de exemplu linia cu metoda *GET*) poate fi urmată de linii adiționale cu mai multe informații. Acestea poartă numele de **antete de cerere**. Această informație poate fi comparată cu parametrii unui apel de procedură. Răspunsurile pot avea de asemenea **antete de răspuns**. Anumite antete pot fi folosite în orice sens. O selecție a celor mai importante este dată în fig. 7-43.

Antetul *User-Agent* permite clientului să informeze serverul asupra programului său de navigare, sistemului de operare și altor proprietăți. În fig. 7-34 am văzut că serverul avea în mod magic această informație și că o poate obține la cerere într-un script PHP. Antetul este utilizat de client pentru a-i asigura serverului această informație.

Antet	Tip	Descriere
User-Agent	Cerere	Informație asupra programului de navigare și a platformei
Accept	Cerere	Tipul de pagini pe care clientul le poate trata
Accept-Charset	Cerere	Seturile de caractere care sunt acceptabile la client
Accept-Encoding	Cerere	Codificările de pagini pe care clientul le poate trata
Accept-Language	Cerere	Limbajele naturale pe care clientul le poate trata
Host	Cerere	Numele DNS al serverului
Authorization	Cerere	O listă a drepturilor clientului
Cookie	Cerere	Trimite un cookie setat anterior înapoi la server
Date	Ambele	Data și ora la care mesajul a fost trimis
Upgrade	Ambele	Protocolul la care transmіtătorul vrea să comute
Server	Răspuns	Informație despre server
Content-Encoding	Răspuns	Cum este codat conținutul (de exemplu, gzip)
Content-Language	Răspuns	Limbajul natural utilizat în pagină
Content-Length	Răspuns	Lungimea paginii în octeți
Content-Tzpe	Răspuns	Tipul MIME al paginii
Last-Modified	Răspuns	Ora și data la care pagina a fost ultima dată modificată
Location	Răspuns	O comandă pentru client pentru a trimite cererea în altă parte
Accept-Ranges	Răspuns	Serverul va accepta cereri în anumite limite de octeți
Set-Cookie	Răspuns	Serverul vrea să salveze un cookie la client

Fig. 7-43. Câteva antete de mesaje HTTP.

Cele patru antete *Accept* spun serverului ce este dispus clientul să accepte în cazul în care acesta are un repertoriu limitat despre ceea ce este acceptabil. Primul antet specifică ce tipuri MIME sunt acceptate (de exemplu, text/html). Al doilea reprezintă setul de caractere (de exemplu ISO-8859 sau Unicode-1-1). Al treilea se referă la metode de compresie (de exemplu, gzip). Al patrulea indică un limbaj natural (de exemplu, spaniola). Dacă serverul are mai multe pagini din care poate să aleagă, el poate utiliza această informație pentru a furniza clientului pagina pe care o cauță. Dacă nu poate satisface cererea, este întors un cod de eroare și cererea eșuează.

Antetul *Host* denumește serverul. El este luat din URL. Antetul este obligatoriu. Este utilizat deoarece anumite adrese IP pot servi mai multe nume de DNS și serverul are nevoie de o anumită modalitate de a spune cărui calculator să-i trimită cererea.

Antetul *Authorization* este necesar pentru protecția paginilor. În acest caz, clientul trebuie să demonstreze că are dreptul de a vedea pagina cerută. Acest header este utilizat în acest scop.

Deși cookie-urile sunt tratate în RFC 2109 mai mult decât în RFC 2616, și ele au două antete. Antetul *Cookie* este utilizat de clienți pentru a întoarce serverului un cookie care a fost anterior trimis de o mașină aflată în domeniul serverului.

Antetul *Date* poate fi utilizat în ambele sensuri și conține ora și data la care a fost trimis mesajul. Antetul *Upgrade* este folosit pentru a face mai ușoară crearea unei tranziții către o viitoare (posibil incompatibilă) versiune a protocolului HTTP. Aceasta permite clientului, să anunțe ce anume suportă, și serverului să afirme ceea ce folosește.

Acum am ajuns la antetele utilizate exclusiv de către server în răspunsul cererilor. Primul, *Server*, permite serverului să spună cine este și câteva proprietăți, dacă dorește.

Următoarele patru antete, toate începând cu *Content-*, permit serverului să descrie proprietățile paginii pe care o transmite.

Antetul *Last-Modified* spune când a fost modificată ultima dată pagina. Acest antet joacă un rol important în mecanismul de memorie ascunsă.

Antetul *Location* este utilizat de server pentru a informa clientul că ar trebui să utilizeze un alt URL. Acesta poate fi folosit dacă pagina a fost mutată, sau pentru a da permisiunea mai multor URL-uri de a referi aceeași pagină (posibil pe servere diferite). Este de asemenea utilizată pentru companiile care au o pagină de Web principală în domeniul *com*, dar care redirecționează clienții la o pagină națională sau regională în funcție de adresa lor IP sau limba preferată.

Dacă o pagină este foarte mare, un client mic poate nu o dorește dintr-o dată. Unele servere acceptă cereri în anumite intervale de octeți, astfel că pagina poate fi citită în mai multe unități mai mici. Antetul *Accept-Ranges* anunță asentimentul severului de a trata acest tip de cerere parțială de pagini.

Al doilea antet pentru cookie, *Set-Cookie*, se referă la modul în care serverele trimit cookie-uri la clienti. Este de așteptat salvarea cookie-ului de către client și returnarea acestuia la cereri ulterioare ale serverului.

### Exemplu de utilizare HTTP

Deoarece HTTP este un protocol ASCII, este destul de ușor pentru o persoană aflată la un terminal (ca opus al programului de navigare) să vorbească direct cu serverele de Web. Este necesară doar o conexiune TCP la portul 80 pe server. Cititorii sunt încurajați să încerce personal acest scenariu (preferabil dintr-un sistem UNIX, deoarece anumite sisteme nu returnează starea conexiunii).

```
Trying 4.17.168.6...
Connected to www.ietf.org.
Escape character is '^'.
HTTP/1.1 200 OK
Date: Wed, 08 May 2002 22:54:22 GMT
Server: Apache/1.3.20 (Unix) mod_ssl/2.8.4 OpenSSL/0.9.5a
Last-Modified: Mon, 11 Sep 2000 13:56:29 GMT
ETag: "2a79d-c8b-39bce48d"
Accept-Ranges: bytes
Content-Length: 3211
Content-Type: text/html
X-Pad: avoid browser bug

<html>
<head>
<title>IETF RFC Page</title>
<script language="javascript">
function url() {
var x = document.form1.number.value
if (x.length == 1) { x = "000" + x }
if (x.length == 2) { x = "00" + x }
if (x.length == 3) { x = "0" + x }
document.form1.action = "/rfc/rfc" + x + ".txt"
document.form1.submit
}
</script>
</head>
```

**Fig. 7-44.** Primele linii ale fișierului *www.ietf.org/rfc.html*.

Următoarea secvență de comenzi va realiza acest lucru:

```
telnet www.ietf.org 80 >log  
GET /rfc.html HTTP/1.1  
Host: www.ietf.org  
close
```

Această secvență de comenzi pornește o conexiune telnet (adică TCP) pe portul 80 al serverului Web al IETF, [www.ietf.org](http://www.ietf.org). Rezultatul sesiunii este redirectat către fișierul *log* pentru o inspecție ulterioară. Apoi urmează comanda *GET* denumind fișierul și protocolul. Următoarea linie este an-tetul obligatoriu *Host*. Linia rămasă liberă este de asemenea cerută. Ea semnalează serverului că nu mai sunt antete de cerere. Comanda *close* indică programului de telnet să întrerupă conexiunea.

Fișierul *log* poate fi inspectat folosind un editor. Acesta ar trebui să înceapă similar cu liniile de cod din fig. 7-44, cu excepția unei modificări recente de către IETF.

Primele trei linii reprezintă ieșirea programului telnet, și nu de la serverul la distanță. Linia ce începe cu HTTP/1.1 este răspunsul IETF prin care spune că este dispus să vorbească HTTP/1.1 cu tine. Apoi urmează un număr de antete și apoi conținutul. Am văzut deja toate antetele, cu excepția lui *ETag* care este un identificator de pagină unic, referitor la memoria ascunsă, și *X-Pad* care nu este standardizat, probabil o cale de scăpare pentru vreun program de navigare cu erori.

### 7.3.5 Îmbunătățiri ale performanței

Popularitatea Web-ului aproape că a fost propria sa distrugere. Servere, rutere și linii folosite de Web sunt adesea supraîncărcate. Multă lume a început să denumească WWW-ul ca World Wide Wait (rom: aşteptare de întindere planetară). Ca o consecință a acestor întârzieri fără sfârșit, cercetătorii au dezvoltat diverse tehnici pentru îmbunătățirea performanțelor. Vom examina acum trei dintre ele: memoria ascunsă, replicarea serverelor și rețelele de livrare a conținutului.

#### Memoria ascunsă

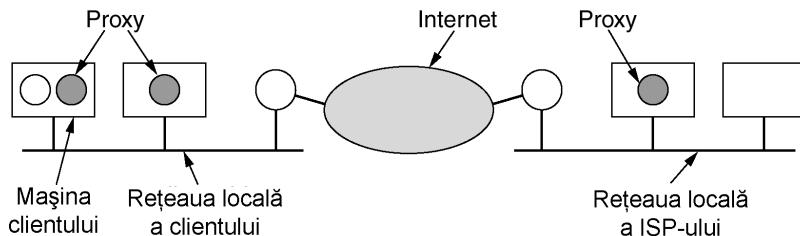
Un mod simplu de a îmbunătăți performanța este de a salva paginile care au fost cerute pentru cazul în care ele vor fi utilizate din nou. Această tehnică este efectivă în special pentru paginile care sunt vizitate foarte mult, ca [www.yahoo.com](http://www.yahoo.com) și [www.cnn.com](http://www.cnn.com). Paginile pot fi păstrate pentru utilizări ulterioare în **memoria ascunsă** (eng.: *cache*). Procedura uzuală este ca un proces, denumit **proxy** (rom.: **delegat**), să întrețină această memorie. Pentru a utiliza memoria ascunsă, un program de navigare poate fi configurat să adreseze toate cererile de pagini proxy-ului, și nu serverului real unde se află pagina respectivă. Dacă proxy-ul are pagina, o returnează imediat. Dacă nu, aduce pagina de la server, o adaugă în memoria ascunsă pentru utilizarea ulterioară și o întoarce clientului care a cerut-o.

Două întrebări importante aferente memoriei ascunse sunt:

1. Cine ar trebui să dețină memoria ascunsă?
2. Cât de mult timp ar trebui să stea paginile în memoria ascunsă?

Există mai multe răspunsuri la prima întrebare. PC-urile individuale de obicei rulează proxy-uri, deci pot să caute rapid pagini vizitate anterior. Într-un LAN al unei companii, proxy-ul este în general o mașină ce poate fi accesată de toate mașinile din acel LAN, astfel că dacă un utilizator se uită la o anumită pagină și apoi alt utilizator din același LAN vrea aceeași pagină, ea poate fi adusă din memoria ascunsă a proxy-ului. Multe ISP-uri rulează proxy-uri, pentru a accelera accesul clientilor

lor. De obicei toate memoriiile ascunse operează în același timp, deci cererile se duc inițial la proxy-ul local. Dacă eșuează, proxy-ul local cere pagina proxy-ului din LAN. Dacă și aceasta eșuează, proxy-ul din LAN încearcă la proxy-ul ISP-ului. Ultimul trebuie să reușească să aducă paginile, fie din memoria sa ascunsă, fie de la o memorie ascunsă de nivel superior, fie de la însuși serverul ce deține pagina. O schemă ce include mai multe memorii ascunse care pot fi încercate în secvență este denumită **memorie ascunsă ierarhică (hierarchical caching)**. O posibilă implementare este ilustrată în fig. 7-45.



**Fig. 7-45.** Memorie ascunsă ierarhică cu 3 proxy-uri .

Cât timp ar trebui paginile să rămână în memoria ascunsă este un pic mai dificil de aflat. Anumite pagini nu ar trebui să fie păstrate deloc în memoria ascunsă. De exemplu, o pagină ce conține prețurile pentru cele mai active 50 de acțiuni la bursă se schimbă la fiecare secundă. Dacă s-ar păstra în memoria ascunsă, un utilizator ce ia o astfel de copie ar lua date **vechi** (adică depășite). Pe de altă parte, din momentul în care schimbul de acțiuni s-a închis pe ziua respectivă, pagina va rămâne validă ore sau zile, până când începe următoarea licitație. Astfel, eficiența menținerii unei pagini în memoria ascunsă poate varia foarte mult în timp.

Problema-cheie în determinarea eliminării unei pagini din memoria ascunsă este legată de vechimea pe care utilizatorii sunt dispuși să o accepte (din moment ce paginile din memoria ascunsă sunt ținute pe disc, cantitatea de memorare consumată reprezintă rareori o problemă). Dacă un proxy elimină repede paginile, el va întoarce rar o pagină veche, dar nu va fi prea eficient (adică va avea o rată scăzută de succes). Dacă păstrează paginile prea mult, poate avea o rată mai mare dar cu prețul de a întoarce deseori pagini cu informație expirată.

Există două abordări în tratarea acestei probleme. Prima utilizează o euristică pentru a ști cât timp să mențină fiecare pagină. O euristică des întâlnită este cea în care se ține cont de antetul *Last-Modified* (vezi fig. 7-43). Dacă o pagină a fost modificată cu o oră în urmă, se ține în memoria ascunsă o oră. Dacă a fost modificată cu un an în urmă, este evident o pagină foarte veche (de exemplu, o listă a zeilor din mitologia greacă și cea romană), deci poate fi păstrată în memoria ascunsă pentru un an, cu o probabilitate rezonabilă că nu se va modifica în decursul anului. Deși această euristică funcționează bine în practică, ea întoarce, totuși, pagini vechi din când în când.

Cealaltă abordare este mai costisitoare dar elimină posibilitatea paginilor vechi prin utilizarea unor caracteristici speciale ale RFC 2616 care tratează administrarea memoriei ascunse. Una din cele mai utilizate caracteristici este antetul de cerere *If-Modified-Since*, pe care un proxy poate să-l trimite unui server. El specifică pagina pe care o vrea proxy-ul și momentul la care pagina din memoria ascunsă a fost modificată ultima dată (din antetul *Last-Modified*). Dacă pagina nu a mai fost modificată de atunci, serverul trimite înapoi un scurt mesaj *Not Modified* (codul de stare 304 din fig. 7-42), care indică proxy-ului să utilizeze pagina din memoria ascunsă. Dacă pagina a mai fost modificată de atunci, este returnată noua pagină. În timp ce pentru această abordare este întotdeauna ne-

voie de un mesaj de cerere și unul de răspuns, mesajul de răspuns va fi foarte scurt când intrarea în memoria ascunsă este încă validă.

Aceste două abordări pot fi combinate ușor. Pentru primul  $\Delta T$  după aducerea paginii, proxy-ul doar returnează pagina clientilor ce o cer. După ce pagina a stat un timp în memoria ascunsă, proxy-ul utilizează mesajul *If-Modified-Since* pentru a verifica valabilitatea acesteia. Alegerea  $\Delta T$  implică invariabil o euristică, depinzând de cât de mult timp a trecut de la modificarea paginii.

Paginile Web cu conținut dinamic (de exemplu, cele generate de un script PHP) nu ar trebui niciodată păstrate în memoria ascunsă, deoarece parametrii pot fi diferiți la următoarea accesare. Pentru a trata aceasta și alte cauze, există un mecanism general prin care un server instruiește toate proxy-urile până la client să nu folosească din nou pagina curentă până nu îi verifică valabilitatea. Acest mecanism poate fi folosit de asemenea pentru orice pagină pasibilă să fie modificată curând. Diverse mecanisme de control al memoriei ascunse sunt definite în RFC 2616.

Mai există o abordare în îmbunătățirea performanței, memoria ascunsă proactivă. Când un proxy aduce o pagină de la un server, el poate inspecta pagina pentru a vedea dacă ea conține hiperlegături. Dacă este aşa, poate să lanseze cereri serverelor corespunzătoare pentru a preîncărca în memoria ascunsă paginile la care punctează, pentru cazul în care va fi nevoie de ele. Această tehnică poate reduce timpul de acces pentru cererile ulterioare, dar poate, la fel de bine, să inunde liniile de comunicație cu pagini care nu vor fi niciodată necesare.

În mod clar, memoria ascunsă în Web este departe de a fi banală. Mult mai multe se pot spune despre ea. De fapt, s-au scris cărți întregi pe această temă, de exemplu (Rabinovich și Spatscheck, 2002; și Wessels, 2001). Dar este timpul ca noi să trecem la un alt subiect.

### Replicarea serverelor

Memoria ascunsă este o tehnică orientată spre client pentru îmbunătățirea performanțelor, dar există și tehnici orientate pe server. Cea mai întâlnită abordare pentru îmbunătățirea performanțelor serverelor este replicarea conținutului lor în mai multe locuri separate. Această tehnică este cîteodată denumită **oglindire (mirroring)**.

Într-o utilizare tipică a oglindirii într-o companie, pagina principală de Web conține câteva imagini cu legături către siturile Web regionale, de exemplu siturile din est, vest, nord și sud. Utilizatorul urmează legătura cea mai apropiată. Din acel moment, toate cererile se duc la serverul selectat.

Siturile oglindite sunt în general complet statice. Compania decide unde să plaseze copiile, dispune de un server în fiecare regiune, și plasează (mai mult sau mai puțin) întregul conținut în fiecare loc (omnipotend, probabil, dezapezitoarele în situl de la Miami și sezlongurile în situl din Anchorage). Alegerea siturilor rămâne în general stabilă luni sau chiar ani de zile.

Din păcate, Web-ul prezintă un fenomen cunoscut ca **aglomerare bruscă (flash crowds)** în care un sit Web care era anterior necunoscut, nevizitat, aproape mort, devine dintr-o dată centrul universului. De exemplu, până pe 6 noiembrie 2000, situl Web al secretarului de stat din Florida, [www.dos.state.fl.us](http://www.dos.state.fl.us), informa tacit despre întâlnirile cabinetului statului Florida și oferea instrucțiuni pentru a deveni notar în Florida. Dar pe 7 noiembrie 2000, când președinția Statelor Unite depindea dintr-o dată de câteva mii de voturi disputate în câteva provincii din Florida, a devenit unul din primele 5 situri Web din lume. Evident, nu a putut suporta încărcarea și aproape că a murit strivit sub ea.

Este necesar un mod prin care un sit Web, ce observă dintr-o dată o cerere masivă a traficului, să se cloneze automat în atâtea locații cât este necesar și să păstreze aceste situri operaționale până când trece furtuna, moment în care oprește majoritatea sau chiar totalitatea acestora. Pentru a avea

această abilitate, un sit are nevoie de o înțelegere prealabilă cu o companie care deține mai multe situri, prin care se pot crea replici la cerere și pentru care se plătește în funcție de capacitatea pe care o folosește în realitate.

O strategie și mai flexibilă este de a crea replici dinamice la nivel de pagini, în funcție de unde vine traficul. Câteva cercetări pe această temă sunt raportate în (Pierre s.a., 2001; și Pierre s.a., 2002).

### Rețele de livrare de conținut

Culmea capitalismului este că cineva a descoperit cum să câștige bani din World Wide Wait. Funcționează în felul următor. Companiile denumite **CDN (Content Delivery Networks, rom: rețele de livrare de conținut)** vorbesc cu deținătorii conținutului (situri muzicale, ziar, și alții care doresc să facă disponibil conținutul ușor și rapid) și se oferă să livreze acest conținut utilizatorilor finali în mod eficient, contra cost. După semnarea contractului, deținătorul conținutului oferă CDN-ului conținutul sitului său Web pentru preprocesare (care va fi discutată imediat) și apoi distribuție.

Apoi CDN vorbește cu un număr mare de ISP-uri și se oferă să îi plătească bine pentru dreptul de a plasa un server administrat la distanță, plin de conținut valoros, pe LAN-urile lor. Nu numai că este o sursă de venit, dar asigură de asemenea clientilor ISP-urilor timp de răspuns excelent pentru a ajunge la conținutul CDN-ului, oferind astfel ISP-ului un avantaj competitiv față de alte ISP-uri care nu au acceptat oferta CDN-ului. În aceste condiții, colaborarea cu un CDN este ceva la mintea cocoșului pentru ISP. Ca o consecință, cele mai mari CDN-uri au mai mult de 10.000 de servere răspândite în toată lumea.

```
<html>
<head><title>Furry Video</title></head>
<body>
<h1>Furry Video's Product List</h1>
<p>Click below for free samples.</p>
<a href="bears.mpg">Bears Today</a><br>
<a href="bunnies.mpg">Funny Bunnies</a><br>
<a href="mice.mpg">Nice Mice</a><br>
</body>
</html>
```

(a)

```
<html>
<head><title>Furry Video</title></head>
<body>
<h1>Furry Video's Product List</h1>
<p>Click below for free samples.</p>
<a href="http://cdn-server.com/furryvideo/bears.mpg">Bears Today</a><br>
<a href="http://cdn-server.com/furryvideo/bunnies.mpg">Funny Bunnies</a><br>
<a href="http://cdn-server.com/furryvideo/mice.mpg">Nice Mice</a><br>
</body>
</html>
```

(b)

**Fig. 7-46.** (a) Pagina Web originală. (b) Aceeași pagină după transformare.

Cu un conținut replicat pe mii de situri în lumea întreagă, există în mod clar un potențial ridicat pentru îmbunătățirea performanțelor. Cu toate acestea, pentru o funcționare bună, trebuie să existe o modalitate prin care să se redirecteze cererea clientului la cel mai apropiat server CDN, preferabil unul aflat la ISP-ul clientului. De asemenea, această redirectare trebuie făcută fără modificarea DNS-ului sau a oricărei alte părți a infrastructurii standard a Internet-ului. O descriere puțin simplificată despre cum lucrează Akamai, cel mai mare CDN, este oferită în continuare.

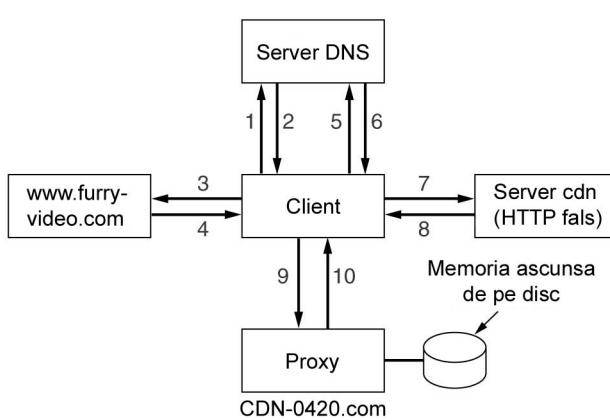
Întregul proces începe din momentul în care furnizorul conținutului trimite CDN-ului situl său Web. Apoi CDN-ul trece fiecare pagină printr-un preprocesor care înlocuiește toate URL-urile cu unele modificate. Modelul de lucru din spatele acestei strategii este acela că situl Web al furnizorului de conținut este constituit din multe pagini foarte mici (doar text HTML), dar că aceste pagini au de obicei referințe către fișiere mari, precum imagini, audio și video. Paginile HTML modificate sunt păstrate pe serverul furnizorului de conținut și sunt aduse în mod obișnuit; doar imaginile, comunicațiile audio și video merg pe serverele CDN-ului.

Pentru a vedea cum funcționează această schemă în realitate, se consideră pagina Web a lui Fury (rom.: îmblânit) Video din fig. 7-46(a). După preprocesare, ea este transformată în fig. 7-46(b) și plasată pe serverul Fury Video ca [www.furryvideo.com/index.html](http://www.furryvideo.com/index.html).

Când un utilizator introduce ca URL [www.furryvideo.com](http://www.furryvideo.com), DNS-ul întoarce adresa IP a sitului Fury Video, permitând ca pagina (HTML) principală să fie adusă în mod obișnuit. Când se face clic pe oricare din hiper-legături, programul de navigare cere DNS-ului să caute *cdn-server.com*, ceea ce acesta chiar face. Programul de navigare trimite apoi o cerere HTTP către această adresă IP, conțând pe faptul că va primi înapoi un fișier MPEG.

Aceasta nu se întâmplă deoarece *cdn-server.com* nu găzduiește nici un conținut. În schimb, acesta este pe serverul fals de HTTP al CDN-ului. El examinează numele fișierului și numele serverului pentru a afla care pagină este necesară cărui furnizor de conținut. De asemenea, examinează adresa IP a cererii sosite și o caută în baza sa de date pentru a determina unde este posibil să se afle utilizatorul. Cu o astfel de informație, el determină care servere CDN de conținut pot oferi utilizatorului serviciul cel mai bun. Această decizie este dificilă deoarece serverul situat geografic cel mai aproape poate să nu fie cel mai apropiat în termeni de topologie de rețea, iar cel mai apropiat în termeni de topologie de rețea poate fi foarte aglomerat în acel moment. După ce se face o alegere, *cdn-server.com* trimite înapoi un răspuns cu codul de stare 301 și un antet *Location* cu URL-ul fișierului pe serverul CDN de conținut cel mai apropiat de client. Pentru acest exemplu, să presupunem că URL-ul este [www.CDN-0420.com/furryvideo/bears.mpg](http://CDN-0420.com/furryvideo/bears.mpg). Programul de navigare procesează apoi acest URL în mod normal pentru a obține fișierul MPEG real.

Pașii urmăți sunt ilustrați în fig. 7-47. Primul pas este căutarea [www.furryvideo.com](http://www.furryvideo.com) pentru a obține adresa lui IP. După aceea, pagina HTML poate fi adusă și afișată în mod obișnuit. Pagina conține trei hiper-legături la *cdn-server* [vezi fig. 7-46(b)]. Când, să zicem, este selectată prima, este căutată (pasul 5) și returnată (pasul 6) adresa sa de DNS. Când o cerere pentru *bears.mpg* este trimisă la *cdn-server* (pasul 7), clientul este înștiințat că trebuie să se ducă de fapt la *CDN-0420.com* (pasul 8). Când face acest lucru (pasul 9), i se dă fișierul din memoria ascunsă a proxy-ului (pasul 10). Proprietatea care face ca întregul mecanism să funcționeze este pasul 8, serverul fals de HTTP redirectând clientul la un proxy CDN aflat aproape de client.



1. Caută www.furryvideo.com
2. Este întoarsă adresa IP a lui Fury
3. Pagina HTML este cerută de la Fury
4. Este întoarsă o pagină HTML
5. După selectia cu mouse-ul, se căută cdn-server.com
6. Este întoarsă adresa IP a lui cdn-server
7. Bears.mpg este cerut de la cdn-server
8. Clientul este redirectionat către CDN-0420
9. Este cerut fișierul bears.mpg
10. Fișierul bears.mpg este întors din memoria ascunsă.

**Fig. 7-47.** Pași în căutarea unui URL când se folosește un CDN

Serverul CDN la care este redirectat clientul este în mod tipic un proxy cu o memorie ascunsă preîncărcată cu conținutul cel mai important. Dacă totuși cineva cere un fișier care nu este în memoria ascunsă, acesta este adus de la serverul real și dispus în memoria ascunsă pentru o utilizare ulterioră. Făcând din serverul de conținut un proxy și nu o replică completă, CDN are abilitatea de a schimba dimensiunea discului, timpul de preîncărcare și diversi parametri de performanță.

Mai multe despre rețele de livrare a conținutului găsiți în (Hull, 2002; și Rabinovich și Spatscheck, 2002).

### 7.3.6 Web-ul fără fir

Există un interes considerabil pentru dispozitivele mici, portabile, capabile să acceseze Web-ul printr-o legătură fără fir. De fapt, primii pași în această direcție au fost deja făcuți. Cu siguranță că vor fi o mulțime de schimbări în acest domeniu în anii ce vin, dar tot merită să examinăm câteva din ideile actuale legate de web-ul fără fir, pentru a vedea unde suntem acum și încotro ne putem îndrepta. Ne vom concentra asupra primelor două sisteme Web fără fir de scară largă care au spart piața: WAP și i-mode.

#### WAP-The Wireless Application Protocol (Protocolul pentru aplicații fără fir)

O dată ce Internet-ul și telefoanele mobile au devenit lucruri comune, nu a durat mult până când cuiva i-a venit ideea să le combine într-un telefon mobil cu ecran încorporat pentru acces fără fir la poșta electronică și la Web. Acel „cineva” a fost consorțiul condus inițial de Nokia, Ericsson, Motorola și phone.com (fosta Unwired Planet) și care acum se laudă cu sute de membri. Sistemul se numește **WAP** (Wireless Application Protocol - protocolul pentru aplicații fără fir).

Un dispozitiv WAP poate fi un telefon mobil îmbunătățit, PDA, sau calculator portabil fără servicii pentru voce. Specificația le permite pe toate și multe altele. Ideea de bază este să se folosească infrastructura digitală fără fir existentă. Utilizatorii pot accesa o poartă (eng.: gateway) WAP prin legătura fără fir și îi pot trimite cereri de pagini Web. Apoi, poarta controlează memoria ascunsă pentru pagina cerută. Dacă există, o trimite; dacă nu există, o ia de pe Internet-ul cu fir. În esență, această înseamnă că WAP 1.0 este un sistem cu comutare de circuite cu o taxă de

conectare pe minut relativ mare. Pentru a scurta o poveste lungă, oamenilor nu le-a plăcut să aceseze Internet-ul pe un ecran mic și plătind la minut, astfel că WAP-ul a fost un fel de nereușită (deși au mai fost și alte probleme). În orice caz, WAP-ul și competitorul sau, i-mode (prezentat mai jos), par să conveargă spre o aceeași tehnologie, astfel că WAP 2.0 ar mai putea să fie un mare succes. Întrucât WAP 1.0 a fost prima încercare pentru Internet-ul fără fir, merită să fie descris cel puțin pe scurt.

WAP este de fapt o stivă de protocoale pentru accesarea Web-ului, optimizată pentru conexiuni cu o bandă de transfer mică folosind dispozitive fără fir ce au un procesor lent, puțină memorie și un ecran mic. Aceste cerințe sunt evident diferite de cele pentru un PC standard de birou, scenariu care duce la niște diferențe între protocoale. Nivelurile sunt prezentate în fig. 7-48.

Mediul aplicațiilor fără fir (WAE)
Protocolul sesiune fără fir (WSP)
Protocolul tranzacție fără fir (WTP)
Securitatea la nivelul transport fără fir (WTLS)
Protocolul pentru datagrame fără fir (WDP)
Nivelul fizic (GSM, CDMA, D-AMPS, GPRS, etc.)

Fig. 7-48. Stiva de protocoale WAP.

Nivelul cel mai de jos suportă toate sistemele de telefonie mobilă existentă, inclusiv GSM, D-AMPS și CDMA. Rata de transfer pentru WAP 1.0 este de 9600 bps. Deasupra acestora se află protocolul pentru datagrame, **WDP (Wireless Datagram Protocol** - protocolul pentru datagrame fără fir), care este de fapt UDP. Apoi vine un nivel pentru securitate, evident necesar într-un sistem fără fir. WTLS este un subset al SSL-ului de la Netscape, la care ne vom uita în cap. 8. Deasupra acestuia este un nivel tranzacție sigură sau nesigură, care se ocupă de cereri și răspunsuri. Acest nivel înlocuiește TCP, care nu este folosit peste legătura prin aer din motive legate de eficiență. Apoi vine un nivel sesiune, care este similar cu HTTP/1.1, dar cu câteva restricții și extensii pentru motive de optimizare. Deasupra acestuia se află micro-programul de navigare (WAE).

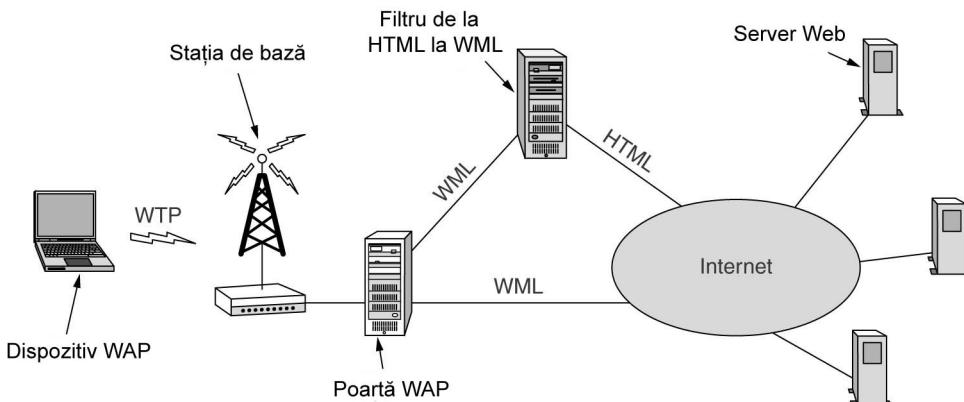


Fig. 7-49. Arhitectura WAP.

În afara costului, celălalt aspect care cu siguranță a afectat acceptarea WAP-ului este faptul că nu folosește HTML. În locul acestuia, nivelul WAE folosește un limbaj de marcare numit **WML** (**Wireless Markup Language** – limbajul de marcare fără fir), care este o aplicație a XML. Drept consecință, în principiu, un dispozitiv WAP nu poate accesa decât acele pagini care au fost convertite la WML. Oricum, având în vedere că asta restricționează mult valoarea WAP-ului, arhitectura reclamă un filtru direct de la HTML la WML pentru a crește multimea paginilor disponibile. Arhitectura este ilustrată în fig. 7-49.

Ca să fim corecți, WAP-ul a fost, probabil, puțin înaintea vremii sale. Când WAP-ul a fost lansat prima dată, XML abia era cunoscut în afara W3C și astfel presa a anunțat lansarea sa spunând **WAP NU FOLOSEȘTE HTML**. Un titlu mai clar ar fi fost: **WAP DEJA FOLOSEȘTE NOUL STANDARD HTML**. Dar cum răul fusese făcut, a fost greu de reparat și WAP 1.0 nu a prins niciodată. Vom rediscuta WAP-ul după ce ne vom uita mai întâi la competitorul său major.

### I-Mode

În timp ce un consorțiu multi-industrial de companii de telecomunicații și de calculatoare a fost ocupat cu construcția unui standard deschis folosind cea mai avansată versiune de HTML disponibilă, alte dezvoltări aveau loc în Japonia. Acolo, o japoneză, Mari Matsunaga, a inventat o altă soluție pentru Web-ul fără fir numită **i-mode** (**information mode** – modul informație). Ea a convins divizia „fără fir” a fostului monopol de telefonie japoneză că ideea sa era corectă și în februarie 1999 NTT DoCoMo (în traducere literală: Compania Japoneză pentru Telefoane și Telegraf oriunde te-ai duce) a lansat serviciul în Japonia. În 3 ani a avut peste 35 de milioane de abonați japonezi, care puteau accesa peste 40.000 de situri Web speciale i-mode. În plus, a mai făcut ca majoritatea companiilor de telecomunicații să saliveze după succesul său financiar, mai ales datorită faptului că WAP nu părea să duca niciieri. Să vedem acum ce este i-mode și cum funcționează.

Sistemul i-mode are trei componente de bază: un nou sistem de transmisie, un nou telefon și un nou limbaj pentru proiectarea paginilor Web. Sistemul de transmisie constă în două rețele separate: rețeaua de telefonie mobilă cu comutare de circuite existentă (oarecum comparabilă cu D-AMPS) și o nouă rețea cu comutare de pachete construită în mod special pentru serviciile i-mode. Modul voce folosește rețeaua cu comutare de circuite și este taxat la fiecare minut de conectare. I-mode folosește rețeaua cu comutare de pachete și este întotdeauna activ (la fel ca la ADSL sau la cablu), astfel că nu există taxarea pentru timpul de conectare. În locul acesta, există o taxă pentru fiecare pachet trimis. Momentan nu este posibil să fie folosite ambele rețele în același timp.

Telefoanele arată ca niște telefoane mobile cărora li s-a adăugat un mic ecran. NTT DoCoMo promovează masiv dispozitivele i-mode ca fiind mai degrabă telefoane mobile decât terminale Web fără fir, deși ele chiar asta sunt. De fapt, probabil că majoritatea clienților nici nu sunt conștienți că sunt conectați la Internet. Ei consideră dispozitivele lor i-mode ca fiind telefoane mobile cu facilități sporite. Pentru a păstra acest model de i-mode la nivel de serviciu, telefoanele nu pot fi programate de utilizatori, deși ele conțin echivalentul unui PC din 1995 și ar putea probabil rula Windows 95 sau UNIX.

Când telefonul i-mode este pornit, utilizatorului îi este prezentată o listă cu categoriile de servicii aprobată oficial. Sunt mult peste 1000 de servicii grupate în aproximativ 20 de categorii. Fiecare serviciu, care este de fapt un mic sit Web i-mode, este oferit de către o companie independentă. Categoriile importante din meniul oficial includ poșta electronică, știri, meteo, sport, jocuri, cumpărături, hărți, horoscop, distrație, călătorii, ghiduri regionale, tonuri ale soneriei, rețete, jocuri de noroc, servicii bancare și cotațiile bursei. Serviciul este oarecum orientat către adolescenți și oameni de 20-

30 de ani, care au tendință să se atașeze de jucăriile electronice, mai ales dacă sunt în culori frumos asortate. Simplul fapt că peste 40 de companii vând tonuri ale soneriei spune ceva. Cea mai populară aplicație este poșta electronică, care permite mesaje de până la 500 de octeți și din acest motiv este văzută ca o mare îmbunătățire față de SMS (Short Message Service – serviciul de mesaje scurte) care permite mesaje de numai 160 de octeți. Jocurile sunt și ele populare.

Sunt de asemenea peste 40.000 de situri Web i-mode, dar ele trebuie accesate mai degrabă scriindu-se URL-ul lor, decât selectându-le dintr-un meniu. Dintr-un punct de vedere, lista oficială este ca un portal Internet care permite altor situri Web să fie accesate prin selecție în loc să li se scrie URL-ul.

NTT DoCoMo controlează îndeaproape serviciile oficiale. Pentru a fi acceptat pe listă, un serviciu trebuie să îndeplinească o serie de criterii publice. De exemplu, un serviciu nu trebuie să aibă o influență negativă asupra societății, dicționarele japonez-englez trebuie să aibă suficiente cuvinte, serviciile cu tonuri pentru sonerie trebuie să adauge frecvent noi tonuri și nici un sit nu poate să promoveze comportarea vicioasă sau să se reflecte negativ asupra NTT DoCoMo (Frangle, 2002). Cele 40.000 de situri Internet pot face orice vor ele.

Modelul afacerii i-mode este atât de diferit de acela al Internet-ului conventional încât merită explicat. Taxa pentru abonamentul de bază i-mode este de câțiva dolari pe lună. Cum există o taxă pentru fiecare pachet primit, abonamentul de bază include și un mic număr de pachete. Ca alternativă, clientul poate opta pentru un abonament cu mai multe pachete gratuite, cu o taxă pe pachet ce scade repede pe măsură ce trece de la 1 MB pe lună la 10 MB pe lună. Dacă pachetele gratuite sunt folosite până la jumătatea lunii, pot fi cumpărate on-line alte pachete adiționale.

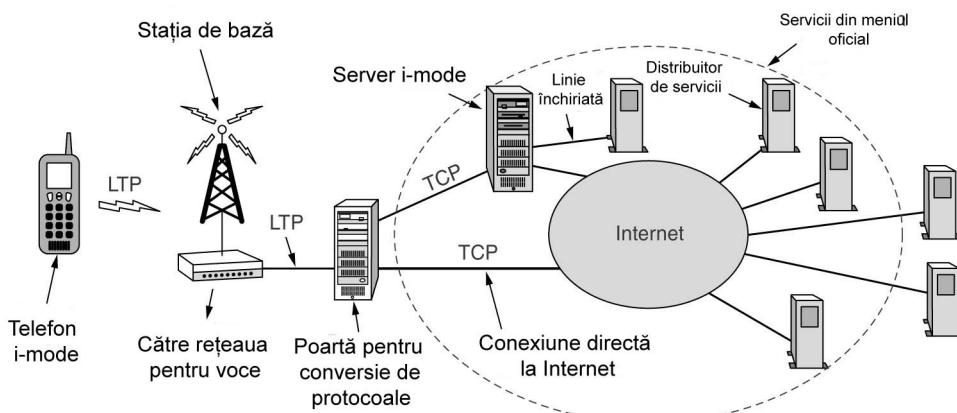
Pentru a folosi un serviciu trebuie să te abonezi la el, fapt care se realizează printr-o simplă selecție și introducerea codului PIN personal. Majoritatea serviciilor oficiale costă în jur de 1\$-2\$ pe lună. NTT DoCoMo adaugă taxa la factura de telefon și transferă 91% celui care oferă serviciul, păstrând 9%. Dacă un serviciu neoficial are 1 milion de clienți, trebuie să trimită 1 milion de facturi de (aproximativ) 1\$ în fiecare lună. Dacă acel serviciu devine oficial, NTT DoCoMo se ocupă de taxare și transferă lunar 910.000\$ în contul din bancă al serviciului. A nu avea de manipulat note de plată este un mare stimulent pentru ca cineva să devină distribuitor oficial de servicii, ceea ce generează venituri mai mari pentru NTT DoCoMo. De asemenea, fiind oficial ajungi în meniul inițial, ceea ce face situl tău mult mai ușor de găsit. Factura utilizatorului include convorbirile, taxele de abonamente i-mode, taxele de abonamente pentru servicii și pachetele suplimentare.

În ciuda succesului său masiv în Japonia, nu este de loc clar că i-mode va prinde și în SUA și Europa. Din unele puncte de vedere, situația din Japonia este diferită de aceea din Vest. În primul rând, majoritatea potențialilor clienți din Vest (spre exemplu adolescentii, studenții și oamenii de afaceri) au deja un PC cu ecran mare acasă și aproape sigur o conexiune la Internet de cel puțin 56 Kbps, adesea mult mai rapidă. În Japonia, puțini oameni au PC-uri conectate la Internet acasă, pe de o parte din cauza lipsei de spațiu, dar și din cauza taxelor exorbitante ale NTT pentru serviciile de telefonie locală (unde în jur de 700\$ pentru instalarea unei linii și 1.50\$ pe oră pentru convorbiri locale). Pentru majoritatea utilizatorilor, i-mode este singura lor conexiune la Internet.

În al doilea rând, locuitorii din Vest nu sunt obișnuiți să plătească 1\$ pe lună pentru a accesa situl Web al CNN, 1\$ pe lună pentru a accesa situl Yahoo, 1\$ pe lună pentru a accesa situl Google și aşa mai departe, fără să mai menționez câțiva dolari pentru fiecare MB descărcat. Majoritatea distribuitorilor de Internet din Vest au acum o taxă fixă pe lună, independentă de utilizarea reală, în mare măsură ca răspuns la cererea clientilor.

În al treilea rând, pentru mulți japonezi, perioada de vârf în care folosesc i-mode este perioada în care se deplasează la sau de la serviciu sau școală în tren sau în metrou. În Europa, mai puțini oameni se deplasează cu trenul decât în Japonia, iar în SUA abia dacă se deplasează cățiva. Folosirea i-mode acasă, lângă un calculator cu un monitor de 17 țoli, conexiune ADSL de 1 Mbps și toti megaocetii gratuiți, nu se prea justifică. Cu toate acestea, nimeni nu a prezis imensa popularitate a telefoanelor mobile în general, astfel că i-mode mai poate încă să-și găsească o nișă în Vest.

Așa cum am menționat mai sus, telefoanele i-mode folosesc rețea cu comutare de circuite existentă pentru voce și o nouă rețea cu comutare de pachete pentru date. Rețea pentru date se bazează pe CDMA și transmite pachete de 128 de octeți la 9600 bps. O diagramă a rețelei este dată în fig. 7-50. Telefoanele folosesc **LTP (Lightweight Transport Protocol** – protocol simplificat de transport) pe o legătură prin aer până la o poartă pentru conversie de protocoale. Poarta are o conexiune de bandă largă prin fibră optică la serverul i-mode, care este conectat la toate serviciile. Când utilizatorul selectează un serviciu din meniu oficial, cererea este trimisă serverului i-mode, care ține majoritatea paginilor în memoria ascunsă pentru a-și spori performanța. Cererile pentru situri care nu sunt în meniu oficial ocolește serverul i-mode și merg direct pe Internet.



**Fig. 7-50.** Structura rețelei de date i-mode, arătând protocoalele de transport.

Telefoanele actuale au procesoare care funcționează la aproximativ 100 MHz, câțiva megaocetii de memorie ROM rapidă, poate 1 MB RAM și un ecran mic încorporat. I-mode necesită un ecran de cel puțin 72x94 pixeli, dar unele dispozitive mai mari au chiar 120x160 pixeli. Ecranele au de obicei culori pe 8 biți, ceea ce permite 256 de culori. Aceasta nu este suficient pentru fotografii, dar este adevarat pentru desenarea de linii și imagini animate simple. Cum nu există mouse, navigarea pe ecran se face cu săgețile direcționale.

Modulul de interacțiune cu utilizatorul		
Elemente de intrare	Interpretor cHTML	Java
Coordonator simplu de ferestre		
Comunicație de rețea		
Sistem de operare în timp real		

**Fig. 7-51.** Structura aplicațiilor i-mode.

Structura aplicațiilor este prezentată în fig. 7-51. Nivelul cel mai de jos conține un sistem simplu de operare în timp real pentru controlul echipamentelor. Apoi vine un modul pentru comunicarea pe rețea, folosind protocolul proprietar al NTT DoCoMo, LTP. Deasupra acestuia vine un simplu coordonator de ferestre care se ocupă de text și de imaginile simple (fișiere GIF). Cu ecranele având doar aproximativ 120x160 de pixeli în cel mai bun caz, nu sunt prea multe de coordonat.

Al patrulea nivel conține interpretorul de pagini Web (de exemplu, programul de navigare). În modul nu folosește întregul HTML, ci numai un subset al acestuia, numit cHTML (**compact HTML** – HTML compact), bazat în mare pe HTML 1.0. Acest nivel permite de asemenea și aplicații ajutătoare și elemente de intrare, la fel cum fac și programele de navigare pentru PC-uri. O aplicație ajutătoare standard este un interpretor pentru o versiune puțin modificată a JVM.

La nivelul cel mai înalt se află modulul de interacțiune cu utilizatorul, care controlează comunicația cu acesta.

Să ne uităm acum mai în detaliu la cHTML. După cum am menționat, este aproximativ HTML 1.0, cu câteva omisiuni și câteva extensii pentru a fi folosit cu telefoane mobile. A fost trimis la W3C pentru standardizare, dar W3C nu a arătat interes pentru el, aşa că probabil va rămâne un produs privat.

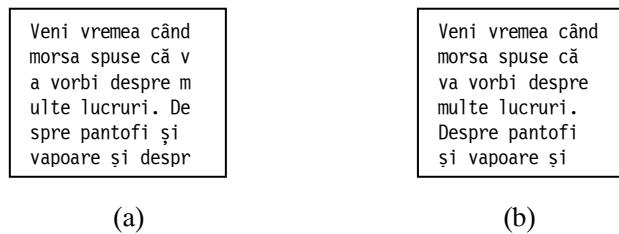
Majoritatea etichetelor de bază HTML sunt permise, inclusiv aici <html>, <head>, <title>, <body>, <hn>, <center>, <ul>, <ol>, <@>, <li>, <br>, <p>, <hr>, <img>, <form> și <input>. Etichetele <b> și <i> nu sunt permise.

Eticheta <a> este permisă pentru legarea la alte pagini, dar cu schema adițională *tel* pentru formarea numerelor de telefon. Din punct de vedere *tel* este analog cu *mailto*. Când o hiper-legătură folosind schema *mailto* este selectată, programul de navigare deschide un formular pentru a trimite un mesaj electronic către destinația marcată în legătură. Când este selectată o hiper-legătură cu schema *tel*, programul de navigare formează numărul de telefon. Spre exemplu, o agenda de telefon poate conține imagini simple ale unor persoane. Când se selectează una dintre acestea, telefonul îl va suna pe el sau pe ea. RFC 2806 prezintă URL-urile pentru telefoane.

Programul de navigare cHTML este limitat în alte feluri. El nu suportă JavaScript, cadre, foi de stil, culori sau imagini de fundal. De asemenea, nu suportă imagini JPEG, deoarece acestea se decompresionează în prea mult timp. Sunt permise applet-urile Java, dar sunt (în prezent) limitate la 10 KB din cauza vitezei mici de transmisie a legăturii prin aer.

Deși NTT DoCoMo a eliminat câteva etichete HTML, a și adăugat unele noi. Eticheta <blink> face ca textul să se afișeze și apoi să dispară. Deși pare ciudat să interzici eticheta <b> (motivând că siturile Web nu ar trebui să se ocupe de prezentarea conținutului) și apoi să adaugi <blink> care nu se referă decât la prezentarea conținutului, asta au făcut. O altă etichetă nouă este <marquee>, care își deplasează conținutul pe ecran precum un monitor de bursă.

Un element nou este atributul *align* al etichetei <br>. Este necesar, deoarece pentru un ecran ce are de obicei 6 rânduri de câte 16 caractere, există un mare pericol ca rândurile să fie rupte în două, ca în fig. 7-52(a). *Align* ajută la reducerea acestei probleme și conduce la ceva ce seamănă mai degrabă cu fig. 7-52(b). Este interesant să notăm că limba japoneză nu suferă când cuvintele sunt rupte între linii. Pentru textul kanji, ecranul este împărțit într-un caroaj dreptunghiular de celule de dimensiune 9x10 pixeli sau 12x12 pixeli, în funcție de caracterele suportate. Fiecare celulă conține exact un caracter kanji, care este echivalentul unui cuvânt în engleză. Despartirea mai multor cuvinte pe mai multe linii este înțotdeauna admisă în japoneză.



**Fig. 7-52.** Lewis Caroll începe într-un ecran de 16x6.

Deși limba japoneză are zeci de mii de kanji, NTT DoCoMo a inventat 166 noi, numite **emoji**, cu un factor de amuzament ridicat – de fapt, niște pictograme precum zâmbitorii din fig. 7-6. Acestea includ simboluri pentru semnele astrologice, bere, hamburger, parc de amuzament, zi de naștere, telefon mobil, câine, pisica, Crăciun, inimă rănită, sărut, stare de spirit, somnolență, și desigur, unul semnificând ceva simpatic.

Un alt nou element este posibilitatea de a permite utilizatorilor să selecteze hiper-legături folosind tastatura, proprietate cu siguranță importantă pentru un calculator fără mouse. Un exemplu despre cum este folosit acest atribut este prezentat în fișierul cHTML din fig. 7-53.

```
<html>
<body>
<h1> Selectează o opțiune </h1>
<a href="messages.chtml" accesskey="1"> Verifică poșta vocală </a> <br>
<a href="mail.chtml" accesskey="2"> Verifică poșta electronică </a> <br>
<a href="games.chtml" accesskey="3"> Joacă un joc </a>
</body>
</html>
```

**Fig. 7-53.** Un exemplu de fișier cHTML.

Deși partea client este oarecum limitată, serverul i-mode este un calculator de-a dreptul răsunător, cu toate soneriele și fluierele obișnuite. El suportă CGI, Perl, PHP, JSP, ASP și tot ceea ce serverele Web suportă de obicei.

Caracteristica	WAP	I-mode
Ce este	Stivă de protocoale	Serviciu
Dispozitiv	Telefon, PDA, calculator portabil	Telefon
Tipul de acces	Prin telefon	Tot timpul activ
Rețeaua de suport	Cu comutare de circuite	Două: circuite + pachete
Rata de transfer	9600 bps	9600 bps
Ecranul	Monocrom	Color
Limbajul de marcare	WML (aplicație XML)	cHTML
Limbajul script	WMLScript	Nici unul
Taxarea utilizatorilor	Pe minut	Pe pachet
Plata pentru cumpărături	Cu cartea de credit	Pe factura de telefon
Pictograme	Nu	Da
Standardizare	Standard deschis al forumului WAP	Proprietar NTT DoCoMo
Locul de utilizare	Europa, Japonia	Japonia
Utilizatorul tipic	Omul de afaceri	Adolescentă

**Fig. 7-54.** O comparație între prima generație de WAP și i-mode.

O comparație rapidă între WAP și i-mode așa cum au fost ele implementate în sistemele de prima generație este dată în fig. 7-54. În timp ce câteva diferențe pot părea mici, adesea ele sunt importante. Spre exemplu, cei de 15 ani nu au cărți de credit, astfel că posibilitatea de a cumpăra lucruri prin comerțul electronic și de a fi taxați abia când primesc nota de plată la telefon reprezintă un stimulent pentru interesul lor asupra sistemului.

Pentru alte informații despre i-mode citiți (Frengle, 2002; și Vacca, 2002).

### Web-ul fără fir de generația a două

WAP 1.0, bazat pe standarde recunoscute internațional, trebuia să fie o unealtă importantă pentru oamenii cu afaceri în derulare. A eșuat. I-mode a fost o jucărie electronică pentru adolescenții japonezi folosind numai elemente proprietare. A fost un mare succes. Ce s-a întâmplat după asta? Fiecare parte a învățat câte ceva din Web-ul fără fir de primă generație. Consorțiu WAP a învățat că important este conținutul. Să nu ai un număr mare de situri Web care îți înțeleg limbajul de marcă este fatal. NTT DoCoMo a învățat că un sistem închis, proprietar, strâns legat de dispozitive mici și de cultura japoneză nu este un bun produs pentru export. Concluzia pe care ambele părți au tras-o este că pentru a convinge un număr mare de situri Web să-și pună conținutul în formatul tău, trebuie să ai un limbaj de marcă deschis, stabil și care este universal acceptat. Războaiele asupra formatelor nu sunt bune pentru progres.

Ambele servicii sunt aproape de a intra în cea de-a două generație a tehnologiei Web fără fir. WAP 2.0 a venit primul, așa că îl vom folosi pe acesta ca exemplu. WAP 1.0 avea unele lucruri bune și acestea au fost continuante. Unul dintre acestea este că WAP poate folosi o mulțime de rețele diferite. Prima generație folosea rețele cu comutare de circuite, dar rețelele cu comutare de pachete au fost întotdeauna o alternativă și încă mai sunt. Sistemele de a două generație vor folosi probabil comutarea de pachete, spre exemplu, GPRS-ul. Un altul este că WAP a fost inițial proiectat pentru a suporta o mare varietate de dispozitive, de la telefoane mobile până la calculatoare portabile puternice, și încă mai este.

WAP 2.0 are și câteva caracteristici noi. Cele mai importante sunt:

1. Modelul de livrare a paginilor (push), alături de cel de cerere (pull).
2. Suport pentru integrarea telefoniei în aplicații.
3. Mesagerie multimedia.
4. Includerea a 264 de pictograme.
5. Interfață cu un dispozitiv de memorare.
6. Suport pentru plug-in-uri în browser.

Modelul de cerere este bine cunoscut: clientul cere o pagină și o primește. Modelul de livrare (push) suportă livrarea datelor fără a fi cerute, precum ținerea la curent cu cotațiile la bursă sau alertele de trafic.

Vocea și datele încep să se contopească, iar WAP 2.0 le suportă într-o mulțime de moduri. Am văzut un astfel exemplu mai devreme, la capacitatea i-mode-ului de a lega o iconă sau un text de pe ecran cu un număr de telefon ce trebuie format. Odată cu poșta electronică și telefonia este suportată și mesageria multimedia.

Marea popularitate a emoji-ului i-mode a stimulat consorțiu WAP să inventeze 264 emoji proprii. Categoriile includ animale, utilaje casnice, îmbrăcăminte, emoții, mâncăruri, corpul uman, genuri, hărți, muzică, plante, sporturi, timp, unelte, vehicule, arme și meteo. Destul de interesant este faptul că standardul pur și simplu numește fiecare pictogramă; nu dă harta de pixeli reală, probabil

de teamă că reprezentarea într-o cultură a „somnolenței” sau a „îmbrățișării” să nu insulte altă cultură. I-mode nu a avut această problemă fiind îndreptat către o singură țară.

A oferi o interfață de stocare nu înseamnă că fiecare telefon cu WAP 2.0 va veni cu un mare disc fix. Memoria ROM rapidă este, de asemenea, un dispozitiv de stocare. O cameră fără fir cu facilități WAP ar putea folosi memoria ROM rapidă pentru stocarea temporară a imaginii înainte de a transmite cele mai bune cadre pe Internet.

În fine, plug-in-urile pot extinde capabilitățile programului de navigare. Este oferit, de asemenea, un limbaj scriptic.

Mai multe diferențe tehnice sunt de asemenea prezente în WAP 2.0. Două dintre cele mai importante se referă la stiva de protocole și la limbajul de marcăre. WAP 2.0 continuă să suporte vechea stivă de protocole din fig. 7-48, dar de asemenea suportă și stiva standard a Internet-ului cu TCP și ©/1.0. Cu toate acestea, patru schimbări minore (dar compatibile) au fost aduse TCP-ului (pentru a simplifica codul): (1) Folosirea unei ferestre fixe de 64 KB, (2) lipsa unui start lent, (3) MTU-ul maxim de 1500 de octeți și (4) un algoritm de retransmisie ușor modificat. TLS este protocolul pentru securitatea la nivel transport standardizat de IETF; îl vom examina în Cap. 8. Mai multe dispozitive inițiale vor conține probabil ambele stive, cum se arată în fig. 7-55.

XHTML	
WSP	©
WTP	TLS
WTLS	TCP
WDP	IP
Nivelul fizic	Nivelul fizic
Stiva de protocole	Stiva de protocole
WAP 1.0	WAP 2.0

Fig. 7-55. WAP 2.0 suportă două stive de protocole.

Cealaltă diferență tehnică față de WAP 1.0 este limbajul de marcăre. WAP 2.0 suportă XHTML Basic, care este potrivit pentru dispozitivele mici fără fir. Întrucât NTT DoCoMo a fost de asemenea de acord să suporte acest subset, proiectanții de situri Web pot folosi acest format știind că paginile lor vor funcționa atât pe Internet-ul fix cât și pe toate dispozitivele fără fir. Aceste decizii vor încheia războaiele legate de formatul limbajului de marcăre care au împiedicat dezvoltarea industriei Web fără fir.

Modulul	Obligatoriu?	Funcția	Exemple de etichete
Structură	Da	Structura documentelor	body, head, html, title
Text	Da	Informații	br, code, dfn, em, hn, kbd, p, strong
Hiper-text	Da	Hiper-legături	a
Liste	Da	Liste de articole	dl, dt, dd, ol, ul, li
Formulare	Nu	Formulare de completat	form, input, label, option, textarea
Tabele	Nu	Tabele dreptunghiulare	caption, table, td, th, tr
Imagini	Nu	Imagini	img
Obiecte	Nu	Applet-uri, hărți, etc.	object, param
Meta-informații	Nu	Informații suplimentare	meta
Legături	Nu	Similar cu <a>	link
Bază	Nu	URL-ul de start	base

Fig. 7-56. Modulele și etichetele din XHTML Basic.

Câteva cuvinte despre XHTML Basic sunt poate necesare. Acesta este gândit pentru telefoane mobile, televiziune, PDA-uri, dispozitive pentru vânzarea automată, pagere, mașini, jocuri mecanice și chiar ceasuri. Din acest motiv nu suportă foi de stil, scripturi sau cadre, însă cunoaște majoritatea etichetelor standard. Acestea sunt grupate în 11 module. Unele sunt obligatorii; altele sunt opționale. Toate sunt definite în XML. Modulele și câteva exemple de etichete sunt listate în fig. 7-56. Nu baleiat toate exemplele de etichete, însă mai multe informații se pot găsi la [www.w3.org](http://www.w3.org).

În ciuda înțelegerii asupra folosirii XHTML Basic, o amenințare pândește WAP și i-mode: 802.11. A doua generație a Web-ului fără fir ar trebui să funcționeze la 384 Kbps, mult mai mult decât cei 9600 bps ai primei generații, dar și mult mai puțin decât cei 11 Mbps sau 54 Mbps oferiti de 802.11. Firește, 802.11 nu este omniprezent, dar pe măsură ce mai multe restaurante, hoteluri, magazine, companii, aeroporturi, stații de autobuz, muzeu, universități, spitale și alte organizații decid să instaleze stații de bază pentru angajații și clienții lor, se va ajunge probabil la o suficientă acoperire în zonele urbane astfel încât oamenii să dorească să meargă pentru o cafea sau pentru a trimite un mesaj electronic la o braserie aflată la câteva blocuri distanță, dar cu 802.11 instalat. Firmele pot adăuga automat embleme 802.11 alături de embleme care arată ce cărți de credit acceptă și asta din același motiv: pentru a atrage clienți. Hărțile orașelor (firește, descărcabile) pot marca zonele acoperite cu verde și pe cele neconectate cu roșu, astfel ca oamenii să se poată deplasa de la o stație de bază la alta, aşa cum nomazii se mutau de la o oază la alta în desert.

Deși braseriile pot instala repede stații de bază 802.11, fermierilor probabil că nu le va fi la fel de ușor, deci acoperirea va fi zonală și limitată la zonele centrale ale orașelor, din cauza răspândirii limitate a semnalului 802.11 (câteva sute de metri în cel mai bun caz). Aceasta poate duce la dispozitive fără fir cu două moduri, care folosesc 802.11 dacă pot prinde un semnal și recurg la WAP dacă nu.

## 7.4 MULTIMEDIA

Deși Web-ul fără fir este o tehnologie nouă și incitantă, ea nu este singura. Pentru mulți, multimedia reprezintă sarea și piperul rețelelor de calculatoare. Mintile ascuțite văd imense provocări tehnice în furnizarea de video (interactiv) la cerere în fiecare casă. Gulerele albe văd un profit imens în acestea. Întrucât multimedia necesită o lățime de bandă mare, este destul de greu să fie făcută să funcționeze prin conexiuni fixe. Chiar și calitatea video VHS prin legătura fără fir este la distanță de câțiva ani, aşa că discuția noastră se va axa asupra sistemelor conectate.

Literal, multimedia înseamnă două sau mai multe media. Dacă editorul acestei cărți voia să se alăture interesului la modă despre multimedia, el putea anunța că lucrarea folosește tehnologia multimedia. În fond, aceasta conține două media: textul și grafica (desenele). Cu toate acestea, atunci când majoritatea oamenilor se referă la multimedia, de fapt ei se referă la combinarea între două sau mai multe **media continue** (continuous media), adică media care trebuie să se desfășoare într-un interval bine definit, de obicei folosind interacțiunea cu utilizatorul. În practică, cele două media sunt audio și video, adică sunete plus filme.

Oricum, mulți vorbesc adesea despre sunetele audio pure, precum telefonia prin Internet sau radio-ul prin Internet, ca și cum ar fi tot multimedia, ceea ce cu siguranță nu sunt. De fapt, un termen mai bun ar fi **fluxuri media** (streaming media), dar vom urma multimea și vom considera sunetele audio în timp real ca fiind tot multimedia. În secțiunile următoare vom examina modul în care calcu-

respectivă. Când un pachet ajunge prin tunelul „bun”, este copiat pe toate celelalte tuneluri care nu s-au auto-tăiat anterior. Dacă toate celelalte tuneluri s-au auto-tăiat și canalul din insula locală nu este interesat, m-ruterul trimite un mesaj de tăiere înapoi prin canalul „bun”. În acest fel, trimiterea multiplă se adaptează automat și merge doar unde este dorită.

PIM-SM (modul rar), descris în RFC 2362, lucrează diferit. Aici ideea este de a preveni saturarea Internetului, doar pentru că trei persoane din Berkeley vor să ţină o conferință peste o adresă de clasă D. PIM-SM funcționează prin fixarea unor puncte de întâlnire. Fiecare dintre sursele dintr-un grup cu trimitere multiplă PIM-SM își trimit pachetele la punctele de întâlnire. Orice sit interesat în atașare, cere unui punct de întâlnire să-i seteze un tunel. În acest mod, tot traficul PIM-SM este transportat prin transmitere simplă, în loc de transmitere multiplă. PIM-SM devine tot mai popular și MBONE migrează către folosirea lui. Pe măsură ce PIM-SM devine mai mult folosit, MOSPF dispare treptat. Pe de altă parte, însuși MBONE pare într-un fel să stagneze și probabil niciodată nu va deveni foarte popular.

Totuși, multimedia prin rețea este încă un domeniu foarte interesant, care evoluează rapid, chiar dacă MBONE nu devine un succes uriaș. Zilnic sunt anunțate noi tehnologii și aplicații. Mai mult, transmiterea multiplă și calitatea serviciului funcționează împreună, după cum se prezintă în (Striegel și Manimaran, 2002). Alt subiect fierbinte este transmisia multiplă fără fir (wireless) (Gossain et. al., 2002). Întregul domeniu al transmisiunilor multiple și orice este legat de el va rămâne, probabil, important pentru următorii ani.

## 7.5 REZUMAT

Atribuirea numelor în Internet folosește o schemă ierarhică, numită sistemul numelor de domenii (DNS). La nivelul superior, există bine cunoscutele domenii generice, inclusiv *com* și *edu*, precum și cele aproximativ 200 domenii pentru țări. DNS este implementat ca un sistem de baze de date distribuite, cu servere în întreaga lume. DNS păstrează înregistrări cu adrese IP, centre de messagerie și alte informații. Prin interogarea unui server DNS, un proces poate stabili corespondența dintre un nume de domeniu Internet și o adresă IP folosită pentru a comunica cu acel domeniu.

E-mail este una din cele două aplicații foarte populare din Internet. Oricine, de la copii la bunici, poate folosi acum. Cele mai multe sisteme de poștă electronică din lume folosesc sistemul definit în RFC 2821 și 2822. Mesajele trimise în acest sistem folosesc antete de sistem ASCII pentru definirea proprietăților mesajului. Materiale cu diverse tipuri conținut pot fi transmise folosind MIME. Mesajele sunt transmise folosind SMTP, care lucrează făcând o conexiune TCP de la sistemul sursă la cel de destinație și livrând în mod direct e-mail-ul peste conexiunea TCP.

O altă aplicație foarte populară pentru Internet este World Wide Web. Web-ul este un sistem pentru legarea documentelor "hipertext". Inițial, fiecare document era o pagină scrisă în HTML, cu posibile hiper-legături la alte documente. Azi, XML câștigă teren în fața HTML. De asemenea, un mare volum de informație este generat dinamic, folosind script-uri executate de server (eng.:server-side scripts) (PHP, JSP și ASP), precum și script-uri executate de client (eng.: client-side scripts) (de remarcat aici Javascript). Un program de navigare poate afișa documentul stabilind o conexiune TCP cu serverul său, cerând documentul și apoi închizând conexiunea. Aceste mesaje de cerere conțin o mulțime de antete pentru asigurarea informației suplimentare. Folosirea memoriei ascunse,

replicarea și rețelele de livrare a conținutului sunt folosite pe scară largă pentru a îmbunătăți performanțele Web-ului.

Web-ul fără fir este de-abia la început. Primele sisteme sunt WAP și i-mode, fiecare cu ecrane mici și lungime de bandă limitate, dar cele din generația următoare vor fi mai puternice.

Multimedia este și ea o stea pe firmamentul rețelelor. Se permite ca semnalele video și audio să fie digitizate și transportate electronic pentru afișare. Audio necesită mai puțină lărgime de bandă, astfel că este mai avansat. Fluxurile audio, radio prin Internet și vocea prin IP (voice over IP) sunt acum o realitate, iar noile aplicații apar permanent. Video la cerere este un domeniu de viitor, de mare interes. În sfârșit, Mbone este un serviciu experimental, bazat pe televiziunea și radioul digital trimise peste Internet.

## 7.6 PROBLEME

1. Multe calculatoare ale unor firme au trei identificatori universali, unici. Care sunt ei?
2. După informațiile date în fig. 7-3, *little-sister.cs.vu.nl* se află într-o rețea de clasă A, B, sau C?
3. În fig. 7-3, nu este nici un punct după *rowboat*? De ce nu?
4. Ghiciți ce ar putea să însemne smiley-ul :-X (uneori scris ca :-#).
5. DNS folosește UDP în loc de TCP. Pachetele DNS pierdute nu pot fi recuperate automat. Cauzează acest lucru probleme, și dacă da, cum sunt ele rezolvate?
6. În plus față de problema pierderilor, pachetele UDP au o dimensiune maximă, uneori ajungând chiar la minimum 576 octeți. Ce se întâmplă când numele DNS căutat depășește această dimensiune? Poate fi trimis în două pachete?
7. Se poate ca o mașină cu un singur nume DNS să aibă mai multe adrese IP? Cum ar putea să se întâpte acest lucru?
8. Este posibil ca un calculator să aibă două nume DNS care aparțin de două domenii de nivel înalt? Dacă da, dați un exemplu plauzibil. Dacă nu, explicați de ce.
9. Numărul de companii cu site Web a crescut exploziv în ultimii ani. Ca rezultat, mii de companii au fost înregistrate în domeniul *com*, ducând la o încărcare mare a serverului pentru acest domeniu. Sugerați o cale de a diminua această problemă fără a schimba schema de nume (adică fără a introduce noi domenii de nivel înalt). Este permis ca soluția să impună schimbarea codului de la client.
10. Unele sisteme de e-mail suportă în antet câmpul *Content Return*: El specifică dacă corpul mesajului trebuie să fie returnat în cazul imposibilității livrării. Acest câmp aparține plicului sau conținutului?