P.C.

1. Nivelurile OSI (Open Systems Interconnection)

Nivelul **fizic** se ocupã de transmiterea biþilor printr-un canal de comunicaþie. Proiectarea trebuie sã garanteze cã atunci când unul din capete trimite un bit 1, acesta e receptat în cealaltã parte ca un bit 1, nu ca un bit 0. Problemele tipice se referã la câþi volþi trebuie utilizaþi pentru a reprezenta un 1 ºi câþi pentru un 0, dacã transmisia poate avea loc simultan în ambele sensuri, cum este stabilitã conexiunea iniþialã ºi cum este întreruptã când au terminat de comunicat ambele pãrþi, câþi pini are conectorul de reþea ºi la ce foloseºte fiecare pin. Aceste aspecte de proiectare au o legãturã strânsã cu interfeþele mecanice, electrice, funcþionale ºi procedurale, ca ºi cu mediul de transmisie situat sub nivelul fizic.

Nivelul **legaturii de date** trateaza erorile de transmisie produse la nivelul fizic, realizand o comunicare corecta intre doua noduri adiacente. Mecanismul utilizat in acest scop este impartirea sirului de biti in cadre, carora le sunt adaugate informatii de control (coduri de verificare, numere de secventa etc).
Cadrele sunt transmise individual putand fi verificate si confirmate de catre receptor. Alte functii ale nivelului se refera la: controlul fluxului datelor (astfel incat transmitatorul sa nu furnizeze date mai rapid decat le poate accepta receptorul) si gestiunea legaturii (stabilirea legaturii, controlul schimbului de date si desfiintarea legaturii).

Nivelul **retea** asigura dirijarea unitatilor de date intre nodurile sursa si destinatar, trecand eventual prin noduri intermediare. Decizia este luata astfel incat sa nu existe in acelasi timp legaturi supraincarcate si legaturi neutilizate, evitandu-se deci congestionarea retelei. O alta functie importanta a nivelului retea este cea de interconectare a retelelor cu arhitecturi diferite.

Nivelul **transport** realizeaza o comunicare sigura intre doua calculatoare gazda, detectand si corectand erorile pe care nivelul retea nu le trateaza. El este important nu numai prin pozitia pe care o ocupa, la mijlocul ierarhiei de nivele, dar si prin functia sa, de a furniza nivelelor superioare o interfata independenta

de tipul retelei utilizate. Functiile sunt realizate de entitati situate in sistemele gazda, fara concursul altor entitati similare intermediare, motiv pentru care se numesc functii "capat la capat" (capetele fiind cele doua entitati de transport corespondente).

Nivelul **sesiune** ofera toate serviciile pentru gestiunea jetoanelor, lasand la latitudinea utilizatorilor semnificatiile asociate acestora. Ele au la baza utilizarea unui mesaj special, numit jeton (token), care poate fi trecut de la un utilizator la altul si a carui posesie ofera detinatorului anumite privilegii: de a transmite date, de a stabili puncte de sincronizare, de a stabili inceputul unei activitati etc.

Nivelul **prezentare** realizeaza transformari ale reprezentarii datelor, astfel incat sa se pastreze semnificatia lor, rezolvandu-se totodata diferentele de sintaxa. Functiile principale se refera la codificarea standard a datelor transmise intre calculatoare cu conventii de reprezentare diferite, la comprimarea textelor, precum si la criptarea/decriptarea acestora in vederea protectiei si securitatii lor.

Nivelul **aplicatie**, cel mai inalt nivel al arhitecturii, are rolul de fereastra de comunicare prin care se fac toate schimburile de date intre utilizatori.
Fiind nivelul care furnizeaza servicii direct aplicatiilor, el cuprinde toate functiile pe care acestea le pot solicita:
- identificarea utilizatorilor cooperanti, autentificarea lor si determinarea disponibilitatii acestora,
- stabilirea calitatii serviciului,
- sincronizarea aplicatiilor cooperante si selectarea modului de dialog,
- stabilirea responsabilitatilor pentru tratarea erorilor,
- identificarea constrangerilor asupra sintaxei datelor,
- transferul informatiei.

2. Serviciile oferite de nivelul legatura de date nivelului retea.

Nivelul legătură de date poate fi proiectat să ofere diferite servicii. Serviciile efective oferite pot varia de la sistem la sistem. Trei posibilităţi de bază, oferite în mod curent, sunt:

1. Serviciu neconfirmat fără conexiune.
2. Serviciu confirmat fără conexiune.
3. Serviciu confirmat orientat-conexiune.

Serviciul neconfirmat fără conexiune constă din aceea că maşina sursă trimite cadre independente către maşina destinaţie, fără ca maşina destinaţie să trebuiască să confirme primirea lor. În acest caz, nu sunt necesare stabilirea şi desfiinţarea unei conexiuni logice. Dacă un cadru este pierdut datorită zgomotului de pe linie, la nivelul legătură de date nu se face nici o încercare pentru recuperarea lui. Această clasă de servicii este adecvată atunci când rata de erori este foarte scăzută, încât recuperarea este lăsată în sarcina nivelurilor superioare. De asemenea, este adecvată pentru traficul de timp real, cum ar fi cel de voce, unde a primi date cu întârziere este mai rău decât a primi date eronate. Majoritatea LAN-urilor utilizează la nivelul legăturii de date servicii neconfirmate fără conexiune.

Următorul pas în ceea ce priveşte siguranţa este **serviciul confirmat fără conexiune**. Atunci când este oferit acest serviciu, încă nu se utilizează conexiuni, dar fiecare cadru trimis este confirmat individual. În acest mod, emiţătorul ştie dacă un cadru a ajuns sau nu cu bine. Dacă nu a ajuns într-un interval de timp specificat, poate fi trimis din nou. Acest serviciu este folositor pentru canale nesigure, cum ar fi sistemele fără fir.

Asigurarea confirmării la nivelul legăturii de date este doar o optimizare, niciodată o cerinţă. Nivelul reţea poate întotdeauna să trimită un mesaj şi să aştepte să fie confirmat. Dacă confirmarea nu apare în timp util, atunci emiţătorul poate retrimite întregul mesaj. Problema cu această strategie este aceea că, de obicei, cadrele au o lungime maximă impusă de hardware, iar pachetele nivelului reţea nu au aceasta limitare. Dacă pachetul mediu este spart în, să zicem, 10 cadre şi 20% din totalul cadrelor sunt pierdute, transmiterea acestuia poate lua

foarte mult timp. În cazul în care cadrele individuale sunt confirmate şi retransmise, pachetele întregi vor fi transmise mult mai rapid. Pe canale sigure, precum fibra optică, costul suplimentar implicat de un astfel de protocol al legăturii de date poate fi nejustificat, dar pe canale fără fir, costul este pe deplin justificat datorită nesiguranţei acestora.

Cel mai sofisticat serviciu pe care nivelul legătură de date îl pune la dispoziţia nivelului reţea este **serviciul orientat-conexiune**. În cazul acestui serviciu, maşinile sursă şi destinaţie stabilesc o conexiune înainte de a transfera date. Fiecare cadru trimis pe conexiune este numerotat şi nivelul legătură de date garantează că fiecare cadru trimis este într-adevăr recepţionat. Mai mult, garantează că fiecare cadru este recepţionat exact o dată şi toate cadrele sunt recepţionate în ordinea corectă. În schimb, în cazul serviciului fără conexiune, este posibil ca, datorită unei confirmări pierdute, un cadru să fie transmis de mai multe ori şi, prin urmare, recepţionat de mai multe ori. Spre deosebire de acesta, serviciul orientat conexiune furnizează proceselor de la nivelul reţea echivalentul unui flux de biţi sigur.

Atunci când este utilizat serviciul orientat conexiune, transferurile au trei faze distincte. În prima fază este stabilită conexiunea, ambele părţi iniţializând variabile şi contoare, utilizate pentru a ţine evidenţa cadrelor care au fost recepţionate şi a celor care nu au fost. În a doua fază, sunt transmise unul sau mai multe cadre. În a treia şi ultima fază, conexiunea este desfiinţată, eliberând variabilele, tampoanele şi alte resurse utilizate la menţinerea conexiunii.

3. Incadrarea la nivelul legatura de date

Deoarece este prea periculos să ne bizuim pe timp pentru a marca începutul şi sfârşitul fiecărui cadru, au fost elaborate alte metode. Vom analiza 3 metode:

1. Numărarea caracterelor.

2. Indicatori cu inserare de octeţi.

3. Indicatori de început şi de sfârşit, cu inserare de biţi.

Prima metodă de încadrare utilizează un câmp din antet pentru a specifica numărul de caractere din cadru. Atunci când nivelul legătură de date de la destinaţie primeşte contorul de caractere, ştie câte caractere urmează şi unde este sfârşitul cadrului. Problema cu acest algoritm este că valoarea contorului poate fi alterată de erori de transmisie. Chiar dacă suma de control este incorectă şi destinaţia ştie că a primit cadru eronat, nu există nici o posibilitate de a determina unde începe următorul cadru. Nu ajută nici trimiterea unui cadru înapoi la sursă, cerând o retransmisie, deoarece destinaţia nu ştie peste câte caractere să sară pentru a începe retransmisia. Din acest motiv, metoda contorizării caracterelor este rar utilizată.

A doua metodă de încadrare înlătură problema resincronizării după o eroare, prin aceea că fiecare cadru începe şi se termină cu o secvenţa specială de octeţi. Iniţial, octeţii ce indicau începutul, respectiv sfârşitul erau diferiţi, dar în ultimii ani s-a trecut la utilizarea unui singur octet, numit octet indicator, atât ca indicator de început, cât şi de sfârşit. În acest fel, dacă receptorul pierde sincronizarea, acesta poate căuta octetul indicator pentru a găsi sfârşitul cadrului. Doi octeţi indicatori consecutivi indică sfârşitul unui cadru şi începutul celui care urmează.

O problemă serioasă cu această metodă apare atunci când se transmit date binare, cum ar fi un obiect sau numere în virgulă mobilă. Se poate întâmpla ca în date să apară octetul folosit ca indicator.

Această situaţie interferează cu procesul de încadrare. O cale de rezolvare a acestei probleme este ca nivelul legătură de date al emiţătorului să insereze un octet special (ESC) înaintea fiecărei apariţii „accidentale" a indicatorului în date. Nivelul legătură de date al receptorului va elimina acest octet special înainte de a pasa datele nivelului reţea. Această tehnică poartă numele de inserare de octeţi (eng.: byte stuffing) sau inserare de caractere (eng.: character stuffing) . Deci, un octet indicator utilizat pentru încadrare poate fi diferenţiat de unul prezent în date prin faptul că este sau nu precedat de un octet special.

Bineînţeles, următoarea întrebare este: Ce se întâmplă dacă un octet special apare în mijlocul datelor? Răspunsul este că pe lângă el se inserează încă un octet special. Deci, un singur octet special face parte dintr-o secvenţă specială, iar un octet special dublu indica faptul ca un singur octet de acest tip a apărut in cadrul datelor. În toate cazurile, secvenţa de octeţi obţinută după eliminarea octeţilor inseraţi este exact secvenţa originală.

Un dezavantaj major al utilizării acestei metode de încadrare este acela că este limitată la utilizarea caracterelor de 8 biţi. Nu toate codurile utilizează caractere de 8 biţi.

A treia metoda de incadrare permite cadrelor de date să conţină un număr arbitrar de biţi şi permite coduri de caractere cu un număr arbitrar de biţi per caracter. Funcţionează astfel: fiecare cadru începe şi se termină cu un şablon special pe biţi, 01111110, numit octet indicator (flag). De fiecare dată când nivelul legătură de date al emiţătorului identifică cinci de unu consecutivi în date, inserează automat un bit 0 în şirul de biţi de rezultaţi. Această inserare de biţi (bit stuffing) este similară inserării de caractere, în care un octet escape este inserat în şirul de caractere de ieşire, înainte de fiecare octet indicator din date.

Atunci când receptorul primeşte o succesiune de cinci biţi 1, urmaţi de un bit 0, extrage automat (adică şterge) bitul 0. La fel ca şi inserarea de caractere, care este

complet transparentă pentru nivelul reţea din ambele calculatoare, aşa este şi inserarea de biţi. Dacă datele utilizator conţin şablonul indicator, 01111110, acest indicator este transmis ca 011111010, dar în memoria receptorului este păstrat ca 01111110.

4. Coduri corectoare de erori

Proiectanţii de reţele au dezvoltat două strategii de bază pentru tratarea erorilor. O modalitate este ca pe lângă fiecare bloc de date trimis să se includă suficientă informaţie redundantă pentru ca receptorul să poată deduce care a fost caracterul transmis. O altă soluţie este să se includă suficientă redundanţă pentru a permite receptorului să constate că a apărut o eroare, dar nu care este eroarea, şi să ceară o retransmisie. Prima strategie utilizează coduri corectoare de erori, iar cea de-a doua utilizează coduri detectoare de erori. Folosirea codurilor corectoare de erori este deseori referită sub numele de corectare de erori în avans (eng.: forward error correction).

Fiecare dintre aceste tehnici se utilizează in situaţii diferite. Pe canale cu siguranţă mare, cum ar fi fibra optică, este mai eficient să utilizăm un cod detector de erori şi să retransmitem blocul în care s-au detectat erori. In cazul canalelor de comunicaţie fără fir, este indicat sa adăugăm destulă informaţie redundantă fiecărui bloc, în loc să ne bazăm pe retransmisie, care poate să fie la rândul său afectată de erori.

Pentru a înţelege cum pot fi tratate erorile, este necesar să privim cu atenţie la ceea ce este de fapt o eroare. În mod normal, un cadru conţine m biţi de date (adică mesaj) şi r biţi redundanţi sau de control. Să considerăm lungimea totală n (adică, n = m + r). O unitate formată din n biţi, care conţine date şi biţi de control, este numită frecvent cuvânt de cod de n biţi (eng.: n-bit codeword).

Date fiind două cuvinte de cod, să zicem, 10001001 şi 10110001, este posibil să determinăm câţi biţi corespunzători diferă. În acest caz diferă 3 biţi. Pentru a determina câţi biţi diferă, aplicăm operatorul SAU EXCLUSIV între cele două cuvinte de cod şi numărăm biţii 1 din rezultat.

Numărul de poziţii binare în care două cuvinte de cod diferă se numeşte distanţa Hamming (Hamming, 1950). Semnificaţia sa este că dacă două cuvinte de cod sunt despărţite de o distanţă Hamming d, sunt necesare d erori de un singur bit pentru a-l converti pe unul în celălalt.

În multe aplicaţii de transmisie de date, toate cele 2m mesaje de date posibile sunt corecte, dar, datorită modului în care sunt calculaţi biţii de control, nu sunt utilizate toate cele 2n cuvinte de cod posibile. Dat fiind algoritmul pentru calculul biţilor de control, este posibil să construim o listă completa de cuvinte de cod permise şi din această listă să găsim cele două cuvinte de cod a căror distanţă Hamming este minimă. Această distanţă este distanţa Hamming a codului complet.

Proprietăţile detectoare şi corectoare de erori ale unui cod depind de distanţa sa Hamming. Pentru a detecta d erori, este nevoie de un cod cu distanţă d + 1, deoarece cu un asemenea cod nu există nici o modalitate ca d erori de un singur bit să poată modifica un cuvânt de cod corect într-un alt cuvânt de cod corect. Atunci când receptorul vede un cuvânt de cod incorect, poate spune că s-a produs o eroare de transmisie. Similar, pentru a corecta d erori, este nevoie de un cod cu distanţă 2d + 1, deoarece în acest mod cuvintele de cod corecte sunt atât de distanţate, încât, chiar cu d modificări, cuvântul de cod originar este totuşi mai apropiat decât alte cuvinte de cod şi va fi unic determinat.

Să ne imaginăm că dorim să proiectăm un cod cu m biţi de mesaj şi r biţi de control care ne va permite să corectăm toate erorile singulare. Pentru fiecare din cele 2m mesaje corecte există n cuvinte de cod eronate, aflate la distanţă 1 de el. Acestea sunt formate prin inversarea sistematică a fiecăruia dintre cei n biţi din cuvântul de cod de n biţi format din el. Astfel, fiecare din cele 2m mesaje corecte necesită n+1 şabloane asociate. Cum numărul total de şabloane este 2n, trebuie să avem (n+1) 2m <= 2n. Utilizând n=m+r, această condiţie devine (m + r + 1)<= 2r. Dându-se m, acesta impune o limită inferioară asupra numărului de biţi de control necesari pentru a corecta erorile singulare.

Această limită inferioară teoretică poate fi, de fapt, atinsă utilizând o metodă atribuită lui Hamming (1950). Biţii cuvântului de cod sunt numerotaţi

consecutiv, începând cu bitul 1 de la marginea din stânga, bitul 2 imediat la dreapta sa, etc. Biţii care sunt puteri ale lui 2 (1, 2, 4, 8, 16 etc.) sunt biţi de control. Restul (3, 5, 6, 7, 9 etc.) sunt completaţi cu cei m biţi de date. Fiecare bit de control forţează ca paritatea unui grup de biţi, inclusiv el însuşi, să fie pară (sau impară).

Un bit poate fi inclus în mai multe calcule de paritate. Pentru a vedea la care biţi de control contribuie bitul de date din poziţia k, rescriem k ca o sumă de puteri ale lui 2. De exemplu, 11= 1+2+8 şi 29=1+4+8+16. Un bit este verificat de acei biţi de control care apar în dezvoltarea sa (de exemplu, bitul 11 este verificat de biţii 1, 2 şi 8).

Când soseşte un cuvânt de cod, receptorul iniţializează un contor la 0. Acesta examinează apoi fiecare bit de control, k (k = 1, 2, 4, 8...) pentru a vedea dacă are paritatea corectă. Dacă nu, adaugă k la contor. Dacă, după ce au fost examinaţi toţi biţii de control, contorul este 0 (adică, dacă toţi biţii au fost corecţi), cuvântul de cod este acceptat ca valid. Dacă valoarea contorului este nenulă, ea reprezintă numărul bitului incorect. De exemplu, dacă biţii de control 1, 2 şi 8 sunt eronaţi, atunci bitul inversat este 11, deoarece este singurul verificat de biţii 1, 2 şi 8.

Codurile Hamming pot corecta numai erori singulare. Totuşi, există un artificiu care poate fi utilizat pentru a permite codurilor Hamming să corecteze erorile în rafală. O secvenţă de k cuvinte de cod consecutive este aranjată ca o matrice, având câte un cuvânt de cod pe fiecare linie. În mod normal, datele ar fi transmise linie cu linie, de la stânga la dreapta. Pentru a corecta erorile în rafală, datele vor trebui transmise pe coloane, începând cu coloana cea mai din stânga. Când au fost trimişi toţi cei k biţi, este transmisă a doua coloană şi aşa mai departe, aşa cum se arată in fig. 3-7. Atunci când un cadru ajunge la receptor, matricea este reconstruită, coloană cu coloană. Dacă a apărut o eroare în rafală, de lungime k, va fi afectat cel mult un bit din fiecare dintre cele k cuvinte de cod, dar codul Hamming poate corecta o eroare pe cuvânt de cod, aşa încât întregul bloc poate fi refăcut. Această metodă utilizează kr biţi de control pentru a face blocuri de km biţi de date imune la erorile în rafală de lungime k sau mai mică.

5. Coduri detectoare de erori

Codurile corectoare de erori sunt adesea utilizate pe canale fără fir, care sunt cunoscute ca fiind predispuse la erori, în comparaţie cu firele de cupru sau fibra optică. Fără coduri detectoare de erori, comunicaţia ar fi greu de realizat. In cazul firelor de cupru sau a fibrei optice rata erorilor este mult mai mică, aşa că detectarea erorilor şi retransmisia este de obicei mai eficientă aici ca metodă de tratare a erorilor care apar ocazional.

Dacă unui bloc i se adaugă un singur bit de paritate şi blocul este puternic deformat de o eroare în rafală lungă, probabilitatea ca eroarea să fie detectată este de numai 0.5, ceea ce este greu de acceptat. Şansele pot fi îmbunătăţite considerabil dacă fiecare bloc transmis este privit ca o matrice dreptunghiulară de n biţi lăţime şi k biţi înălţime, după cum am arătat mai sus. Pentru fiecare coloană este calculat un bit de paritate, care este adăugat într-o nouă linie de la sfârşitul matricei. Matricea este apoi transmisă linie cu linie. La sosirea blocului, receptorul verifică toţi biţii de paritate. Dacă oricare din ei este greşit, va cere o retransmisie a blocului. Retransmisii succesive sunt cerute daca este nevoie, până când întregul bloc este recepţionat fără erori de paritate.

Această metodă poate detecta o singură rafală de lungime n, cu numai un bit pe coloană modificat. O rafală de lungime n+1 va trece totuşi nedetectată dacă primul şi ultimul bit sunt inversaţi, iar toţi ceilalţi biţi sunt corecţi (o eroare în rafală nu înseamnă că toţi biţii sunt greşiţi, ci că cel puţin primul şi ultimul sunt greşiţi). Dacă blocul este puternic deformat de o rafală lungă sau de rafale scurte multiple, probabilitatea ca oricare din cele n coloane să aibă, accidental, paritatea corectă este 0.5, deci probabilitatea ca un bloc eronat să fie acceptat atunci când nu ar trebui este 2 la -n.

Cu toate că schema de mai sus poate fi uneori adecvată, în practică este larg utilizată o altă metodă: **codul polinomial** (cunoscut şi sub numele de cod cu redundanţă ciclică, eng.: **cyclic redundancy code**). Codurile polinomiale sunt bazate pe tratarea şirurilor de biţi ca reprezentări de polinoame cu coeficienţi 0 şi 1. Un cadru de k biţi este văzut ca o listă de coeficienţi pentru un polinom cu k termeni, de la xk-1 la x0. Se spune că un astfel de polinom este de gradul k-1. Bitul cel mai semnificativ (cel mai din stânga) este coeficientul lui xk-1; următorul bit este coeficientul lui xk-2 ş.a.m.d. De exemplu, 110001 are şase biţi şi ei reprezintă un polinom cu şase termeni cu coeficienţii 1, 1, 0, 0, 0 şi 1: x5+x4+x0. Aritmetica polinomială este de tip modulo 2, în conformitate cu regulile teoriei algebrice. Nu există transport la adunare şi nici împrumut la scădere. Atât adunările cât şi scăderile sunt identice cu SAU EXCLUSIV. Împărţirea lungă este făcută ca în binar cu excepţia faptului că scăderea este realizată modulo 2, ca mai sus. Despre un împărţitor se spune că „intră" într-un deîmpărţit dacă deîmpărţitul are tot atâţia biţi ca împărţitorul.

Atunci când este utilizată metoda codului polinomial, emiţătorul şi receptorul se pun de acord în avans asupra unui polinom generator G(x). Atât bitul cel mai semnificativ cât şi cel mai puţin semnificativ trebuie să fie 1. Pentru a calcula suma de control pentru un cadru cu m biţi, corespunzător polinomului M(x), cadrul trebuie să fie mai lung decât polinomul generator. Ideea este de a adăuga o sumă de control la sfârşitul cadrului, astfel încât polinomul reprezentat de cadrul cu sumă de control să fie divizibil prin G(x). Când receptorul preia cadrul cu suma de control, încearcă să-l împartă la G(x). Dacă se obţine un rest, înseamnă că a avut loc o eroare de transmisie.

Algoritmul pentru calculul sumei de control este următorul:

1. Fie r gradul lui G(x). Se adaugă r biţi 0 la capătul mai puţin semnificativ al cadrului, aşa încât acesta va conţine acum n+r biţi şi va corespunde polinomului xrM(x).

2. Se împarte şirul de biţi ce corespund lui G(x) într-un şir de biţi corespunzând lui xrM(x), utilizând împărţirea modulo 2.

3. Se scade restul (care are întotdeauna r sau mai puţini biţi) din şirul de biţi corespunzând lui xrM(x), utilizând scăderea modulo 2. Rezultatul este cadrul cu sumă de control ce va fi transmis. Numim polinomul său T(x).

Să analizăm puterea acestei metode. Ce tipuri de erori vor fi detectate? Să ne imaginăm că apare o eroare de transmisie, aşa încât în loc să sosească şirul de biţi pentru T(x), ajunge T(x) + E(X). Fiecare bit din E(x) corespunde unui bit care a fost inversat. Dacă în E(x) există k biţi 1, aceasta înseamnă că au apărut k erori de un singur bit.

La recepţia cadrului cu sumă de control, receptorul îl împarte prin G(x); aceasta înseamnă că va calcula [T(x) + E(x)]/G(x). T(x)/G(x) este 0, aşa încât rezultatul calculului este pur şi simplu E(x)/G(x). Acele erori care se întâmplă să corespundă unor polinoame care îl au ca factor pe G(x) vor scăpa; toate celelalte vor fi detectate.

6. Protocoalele elementare ale legaturii de date

*Protocolul Start-Stop

-transmiţătorul trimite un nou cadru numai după recepţia confirmării pozitive a cadrulului precedent.

*Protocol simplex fară restrictii.

Cea mai simplă variantă de protocol are în vedere următoarele consideraţii:

• utilizatorul A vrea să transmită date lui B folosind o legătură sigură, simplex;

• A reprezintă o sursă inepuizabilă de date, astfel încît transmiţătorul nu trebuie să aştepte niciodată la preluarea datelor pentru transmisie;

• B reprezintă un consumator ideal, care preia imediat datele de la receptor ori de cîte ori acesta i le dă;

• canalul fizic de comunicaţie este fără erori.

*Protocol simplex start-stop.

În a doua variantă, considerăm în continuare canalul fără erori, dar utilizatorul B nu poate accepta date în orice ritm. Este necesar controlul fluxului, care în acest caz se realizează printr-o reacţie a receptorului către transmiţător, permiţîndu-i

acestuia din urmă să transmită următorul cadru. Reacţia are forma unui cadru fictiv.

***Protocol simplex pentru un canal cu erori.**

S-ar părea că în cazul unui canal cu erori o uşoară modificare a protocolului anterior este suficientă. Receptorul transmite un cadru de confirmare doar pentru cadre corecte. În caz de eroare, transmiţătorul nu primeşte confirmarea într-un timp prestabilit şi ca urmare retransmite cadrul. Funcţia de armare a ceasului şi de permisiune a evenimentului TimeOut este: void StartCeas(NrSecv); iar funcţia de dezarmare a ceasului este: void StopCeas (NrSecv);

Parametrul lor permite asocierea unui eveniment TimeOut cu cadrul avînd un număr de secvenţă specificat. Această variantă prezintă pericolul duplicării cadrelor, în cazul în care se pierde confirmarea mesajului şi nu mesajul (transmiţătorul retransmite cadrul precedent, pe care receptorul îl ia drept cadru nou). Evitarea acestei erori se face prin includerea unui număr de secvenţă în antetul cadrului. Deoarece în acest caz, numărul de secvenţă trebuie să diferenţieze doar două cadre succesive, este suficient un contor de un bit, numerele de secvenţa 0 şi 1 ale cadrelor succesive alternînd.

Pentru justificare, fie cadrele succesive m, m+1, m+2 cu numerele de secvenţă respectiv 0, 1 şi 0. După transmisia cadrului m (0), poate urma m (0), sau m+1 (1), după cum s-a primit sau nu confirmarea. În nici un caz nu urmează m+2 (0), care presupune confirmarea corectă a lui m (0). Deci dacă receptorul a primit corect m (0) şi primeşte un nou cadru cu număr de secvenţă 0, el nu poate fi decît m (0) şi în consecinţă îl ignoră.

***Protocol cu fereastra glisanta**

-nu aşteaptă confirmarea cadrelor precedente pentru a transmite cadre noi.

Protocoalele anterioare sînt simplex. Pentru o transmisie duplex ar fi necesare două legături distincte, una pentru fiecare sens. Ca alternativă, putem utiliza cadrele de confirmare pentru transmiterea datelor în sens opus. Dacă nu sînt date, se transmite doar confirmarea. Apar deci mai multe feluri de cadre, diferenţiabile printr-un cîmp din antet (fel).

Aceste elemente sînt cuprinse în protocoalele cu fereastră glisantă. Esenţa lor este prezentată în continuare. Transmiţătorul menţine o listă cu numerele de secvenţă ale cadrelor transmise dar neconfirmate, alcătuind fereastra transmiţătorului. Cînd trebuie transmis un cadru, i se ataşează numărul de secvenţă următor (fereastra se măreşte). Cînd se recepţionează o confirmare, marginea inferioară a ferestrei se deplasează cu o unitate. Cadrele se păstrează de transmiţător pînă la confirmarea lor.

Receptorul păstrează o listă cu numerele de secvenţă ale cadrelor pe care le poate accepta (fereastra receptorului). Cadrele din fereastră sînt acceptate, celelalte sînt ignorate. Cînd soseşte un cadru cu număr de secvenţă egal cu marginea inferioară, fereastra este deplasată cu o poziţie (prin modificarea ambelor margini), păstrîndu-se dimensiunea constantă.

***Protocol cu fereastră de dimensiune unu.**

Ambele ferestre fiind unitare, transmiţătorul trebuie să aştepte confirmarea înainte de a transmite un nou cadru (la fel ca în cazul precedent). Deoarece algoritmii transmiţătorului şi receptorului sînt aproape identici, dăm o singură descriere pentru ambele staţii. Fiecare staţie realizează ciclic următoarele operaţii:

- recepţia unui cadru,
- prelucrarea şirului de cadre recepţionate,
- prelucrarea şirului de cadre transmise,
- transmiterea sau retransmiterea unui cadru impreună cu confirmarea cadrului recepţionat corect.

O situaþie necorespunzãtoare apare dacã ambele staþii transmit un cadru iniþial. Pentru a corecta acest neajuns, o staþie trebuie sã o preceadã pe cealaltã prin acþiunile sale. Pentru aceasta, doar una trebuie sã execute transmisia unui cadru în afara buclei principale.

Cîmpul conf conţine numărul ultimului cadru recepţionat corect. Dacă el corespunde cadrului aflat în transfer, transmiţătorul preia alt pachet pentru transmisie. Altfel, se transmite acelaşi pachet.

Cîmpul secv conţine numărul de secvenţă al cadrului. Dacă el coincide cu cel aşteptat, se transmite un pachet nivelului reţea. Staţiile nu se blochează şi nu acceptă duplicate. Să presupunem de exemplu că A încearcă să trimită cadrul 0 lui B, iar B încearcă să trimită cadrul 0 lui A. Presupunem că timpul de aşteptare (timeout) al lui A este prea scurt, deci A trimite repetat cadre (0, 1, A0). Cînd primul cadru valid ajunge la B, va fi acceptat, punîndu-se CadruAsteptat pe 1. Celelalte cadre sînt ignorate, avînd secv=0. Mai mult, duplicatele au conf = 1, în timp ce B aşteaptă conf = 0 pentru a scoate un nou pachet de la reţea. Ca urmare, B va continua să transmită repetat un cadru cu (0, 0,...). Unul din cadre ajunge la A, care începe să transmită un nou pachet.

***Protocoale cu fereastră supraunitară de transmisie.**

În cazul unor canale cu întîrzieri mari de transmisie, aşteptarea confirmării fiecărui cadru înainte de transmiterea următorului poate conduce la o eficienţă scăzută. Astfel, la un canal cu o întîrziere de propagare dus-întors de 500 msec şi o viteza de 50 Kbps, transmiterea unui cadru de 1000 de biţi durează efectiv doar 20 de milisecunde din cele 520 necesare ciclului complet de transmitere şi recepţie a confirmării.

Performanţa se poate ameliora permiţînd transmiterea unui cadru înainte de a se primi confirmarea cadrelor anterioare. În exemplul precedent, dacă transmiţătorul trimite cadrele unul după altul, el va recepţiona confirmarea primului cadru după transmiterea celui de al 26-lea. Erorile nu pot fi sesizate şi corectate imediat de transmiţător. De obicei, el află despre producerea unei erori după ce a transmis şi alte cadre. Receptorul, care trebuie să păstreze secvenţialitatea cadrelor pe care le livrează utilizatorului, are două posibilităţi de tratare a cadrelor primite după eroare.

O cale este de ignorare a acestora, fără transmitere de confirmări. Ea corespunde unei ferestre de recepţie unitare. În acest caz, transmiţătorul va fi obligat să retransmită toate cadrele începînd cu cel eronat. A doua cale este memorarea cadrelor recepţionate corect, urmînd a fi retransmise selectiv doar cadrele eronate. Ea corespunde unei ferestre a receptorului mai mare ca unu.

***Protocol cu retransmitere neselectivă.**

Fereastra maximă a transmiţătorului poate fi de MaxSecv cadre, deşi există MaxSecv+1 numere de secvenţă distincte. Justificăm această restricţie prin următorul scenariu, în care presupunem că MaxSecv=7:

1. Transmiţătorul trimite cadrele 0..7;
2. Toate cadrele sînt recepţionate şi confirmate;
3. Toate confirmările sînt pierdute;
4. Transmiţătorul retrimite la time-out toate cadrele;
5. Receptorul acceptă duplicatele.

În acest protocol renunţăm la presupunerea că utilizatorul este o sursă ideala de pachete şi includem mecanismul de reglare a fluxului de date: cînd utilizatorul are un pachet de transmis el provoacă un eveniment "ReteaPregatita"; cînd o staţie are MaxSecv cadre neconfirmate, ea interzice reţelei să-i transmită alte pachete prin procedura "DezactivRetea"; cînd numărul de pachete scade, ea anulează restricţia prin funcţia "ActivRetea".

Transmiţătorul păstrează cadrele neconfirmate pentru o eventuală retransmisie. Confirmarea cadrului n provoacă automat confirmarea cadrelor n-1, n-2,... anterioare, ceeace compensează eventualele pierderi ale confirmărilor acestor cadre. Deoarece mai multe cadre îşi asteaptă confirmarea, se utilizează ceasuri separate pentru diferite mesaje. Fiecare ceas este pornit la transmiterea unui cadru şi este oprit la recepţia confirmării sale. La producerea unui eveniment TimeOut, toate cadrele aflate în memoria tampon a transmiţătorului sînt retrimise.

***Protocol cu retransmitere selectiva.**

În acest caz, fereastra receptorului este mai mare ca unu. Cînd se primeşte un cadru avînd număr de secvenţă în fereastră, el este acceptat şi memorat de nivelul legătură de date, nefiind comunicat utilizatorului decît după livrarea cadrelor dinaintea sa.

Fereastra receptorului nu poate fi egală cu cea a transmiţătorului aşa cum rezultă din următorul scenariu, pentru MaxSecv = 7 :

1. Transmiţătorul trimite cadrele 0..6
2. Cadrele sînt recepţionate şi fereastra receptorului devine 7,0, 1, 2, 3, 4, 5
3. Toate confirmările sînt pierdute
4. Transmiţătorul retrimite cadrul 0 la time-out
5. Receptorul acceptă cadrul 0 aflat în fereastra şi cere cadrul 7

6. Transmiţătorul află că toate cadrele sale au fost transmise corect şi trimite cadrele 7, 0, 1, 2, 3, 4, 5
7. Receptorul acceptă cadrele, cu excepţia lui 0, pentru care are deja un cadru recepţionat. Ca urmare, ignoră acest cadru, luînd în locul lui dulpicatul cadrului 0 anterior.

Eroarea se produce deoarece ferestrele succesive ale receptorului au numere de secvenţă comune, neputîndu-se face diferenţa între cadrele noi şi duplicatele celor vechi. Pentru a elimina aceste suprapuneri, fereastra receptorului trebuie să fie cel mult jumătate din gama numerelor de secvenţă.

Fereastra receptorului determină şi numărul de tampoane de recepţie, ca de altfel şi numărul de ceasuri necesare. Cînd canalul este puternic încărcat, confirmările cadrelor transmise de la staţia A la staţia B pot fi comunicate odată cu datele vehiculate de la B la A. Dacă traficul de la B la A este scăzut, se recurge la transmiterea unor cadre speciale de confirmare. Transmiterea confirmărilor nu se face imediat după recepţia unui cadru corect, ci după un interval de timp convenabil stabilit şi numai dacă în acest interval nu a apărut posibilitatea transmiterii confirmării într-un cadru de date de la B la A. Terminarea intervalului de timp are drept corespondent evenimentul "ReteaLibera". Pentru a îmbunătăţi eficienţa transmisiei, dacă receptorul detectează un cadru eronat, el poate transmite o confirmare negativă "nak", care reprezintă o cerere explicită de retransmitere a cadrului specificat în "nak". Receptorul trebuie să evite transmiterea mai multor cadre "nak" pentru acelaşi cadru aşteptat.

7. HDLC - Controlul de nivel înalt al legăturii de date

În această secţiune vom examina un grup de protocoale strâns legate, puţin mai vechi, dar care sunt încă foarte utilizate. Ele sunt toate derivate din protocolul pentru legătura de date utilizat în lumea mainframe-urilor IBM, numit SDLC (Synchronous Data Link Control, rom.: protocolul de control sincron al legăturii de date). După ce a dezvoltat SDLC, IBM l-a supus examinării ANSI şi ISO pentru acceptare ca standard SUA şi, respectiv, internaţional. ANSI a modificat protocolul, astfel încât acesta a devenit ADCCP (Advanced Data Communication Control Procedure, rom.: procedură de control avansat al comunicaţiilor de date), iar ISO l-a modificat şi a produs HDLC (High-level Data Link Control, rom.: control de nivel înalt al legăturii de date). CCITT a adoptat şi modificat HDLC pentru al său

LAP (Link Access Procedure, rom.: procedură de acces la legătură) care este parte a standardului pentru interfaţa de reţea X.25, dar, mai târziu l-a modificat din nou, rezultând LAPB, în scopul de a-l face mai compatibil cu o versiune ulterioară de HDLC. Un lucru frumos în ceea ce priveşte standardele este că sunt multe, dintre care poţi alege. În plus, dacă nu îţi place nici unul dintre ele, poţi aştepta modelul care va apărea anul viitor.

Aceste protocoale se bazează pe aceleaşi principii. Toate sunt orientate pe biţi şi folosesc inserarea de biţi pentru transparenţa datelor. Ele diferă doar în puncte minore, şi totuşi supărătoare. Discuţia care urmează, despre protocoalele orientate pe biţi, intenţionează a fi o introducere generală. Pentru detaliile specifice fiecărui protocol, consultaţi definiţia corespunzătoare.

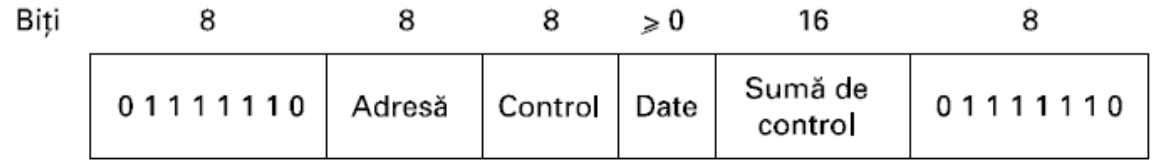


Fig. 3-24. Format de cadru pentru protocoalele orientate pe biți.

Toate protocoalele orientate pe biţi folosesc structura de cadru prezentată în fig. 3-24. Câmpul Adresă este primul ca importanţă pentru liniile cu terminale multiple, unde el este folosit pentru a identifica unul dintre terminale. Pentru liniile punct-la-punct, el este folosit uneori pentru a deosebi comenzile de răspunsuri.

Câmpul Control este folosit pentru numere de secvenţă, confirmări şi alte scopuri, după cum se va arăta în continuare.

Câmpul Date poate conţine informaţii arbitrare. Poate avea lungime arbitrară, cu toate că eficienţa sumei de control scade odată cu creşterea lungimii cadrului, datorită creşterii probabilităţii de apariţie a erorilor în rafală.

Câmpul Suma de Control este o variantă CRC (Cyclic Redundancy Code - cod ciclic redundant).

Cadrul este delimitat cu o altă secvenţă indicator (01111110). Pe liniile punct-la-punct inactive secvenţele indicator sunt transmise continuu. Un cadru minim conţine trei câmpuri şi are în total 32 de biţi, excluzând indicatorii de la capete.

Există trei tipuri de cadre: Informaţie, Supervizor şi Nenumerotat. Conţinutul câmpului Control pentru fiecare dintre aceste trei tipuri este prezentat în fig. 3-25. Acest protocol foloseşte o fereastră glisantă, cu un număr de secvenţă reprezentat pe 3 biţi. În fereastră pot fi păstrate, la un moment dat, până la şapte cadre neconfirmate.

Câmpul Secvenţă din fig. 3-25(a) este numărul de secvenţă al cadrului. Câmpul Următor este o confirmare ataşată. Oricum, toate protocoalele aderă la convenţia că, în loc să ataşeze numărul ultimului cadru recepţionat corect, să folosească numărul primului cadru nerecepţionat (adică următorul cadru aşteptat). Opţiunea pentru ultimul cadru primit sau următorul cadru recepţionat este arbitrară; nu are importanţă ce convenţie este utilizată, dacă este folosită cu consecvenţă.

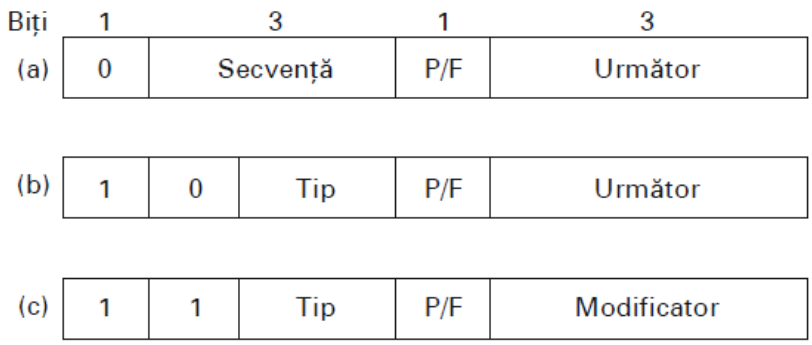


Fig. 3-25. Câmpul Control pentru (a) un cadru de informație, (b) un cadru de supervizare, (c) un cadru nenumerotat.

Bitul P/F înseamnă Test/Final (eng.: Poll/Final). El este folosit atunci când un calculator (sau un concentrator) interoghează un grup de terminale. Când este folosit ca P, calculatorul invită terminalul să trimită date. Toate cadrele trimise de terminal, cu excepţia celui final, au bitul P/F setat pe P. Pentru cadrul final bitul este setat la F.

În câteva dintre protocoale, bitul P/F este folosit pentru a forţa cealaltă maşină să trimită imediat un cadru Supervizor, în loc să aştepte fluxul invers la care să se ataşeze informaţia despre fereastră. Bitul are de asemenea câteva utilizări minore referitoare la cadrele nenumerotate.

Numeroasele tipuri de cadre Supervizor sunt diferenţiate prin câmpul Tip. Tipul 0 este un cadru de confirmare (numit oficial RECEIVE READY) folosit pentru a indica următorul cadru aşteptat. Cadrul este folosit atunci când nu există flux invers care să poată fi folosit pentru ataşare.

Tipul 1 este un cadru de confirmare negativă (oficial numit REJECT). Este folosit pentru a indica detecţia unei erori de transmisie. Câmpul Următor indică primul cadru din secvenţă ce nu a fost recepţionat corect (deci cadrul ce trebuie retransmis). Transmiţătorului i se cere să retransmită toate cadrele neconfirmate, începând cu Următor-ul.

Tipul 2 este RECEIVE NOT READY. El confirmă toate cadrele, cu excepţia lui Următor, exact ca RECEIVE READY, dar spune transmiţătorului să oprească transmisia. RECEIVE NOT READY este destinat să semnaleze anumite probleme temporare apărute la receptor, cum ar fi lipsa zonelor tampon, şi nu ca o alternativă la controlul fluxului cu fereastră glisantă. Când problema a fost rezolvată, receptorul trimite un RECEIVE READY, REJECT sau anumite cadre de control.

Tipul 3 este SELECTIVE REJECT. El cere retransmiterea, însă doar pentru cadrul specificat. Din acest punct de vedere este folositor atunci când dimensiunea ferestrei transmiţătorului este jumătate sau mai puţin din dimensiunea spaţiului secvenţei. Astfel, dacă receptorul doreşte să păstreze cadre care erau în afara secvenţei pentru posibila folosire ulterioară, el poate să forţeze retransmiterea oricărui cadru, folosind SELECTIVE REJECT. HDLC şi ADCCP permit acest tip de cadru, dar SDLC şi LAPB nu îl permit (adică nu există Selective Reject) şi cadrele de tipul 3 nu sunt definite.

Cea de-a treia clasă o reprezintă cadrul Nenumerotat. El este folosit uneori în scopuri de control, dar poate fi folosit şi pentru transportul datelor atunci când se recurge la un serviciu nesigur, neorientat pe conexiune. Diversele tipuri de protocoale orientate pe biţi diferă considerabil aici, spre deosebire de celelalte două tipuri, unde erau aproape identice. Pentru a indica tipul cadrului sunt disponibili cinci biţi, dar nu sunt folosite toate cele 32 de posibilităţi.

Toate protocoalele furnizează o comandă, DISC (DISConnect), care permite ca o maşină să anunţe că se va opri. De asemenea există o comandă ce permite ca o maşină, care tocmai s-a reconectat, să-şi anunţe prezenţa şi să forţeze resetarea tuturor numerelor de secvenţă la zero. Această comandă poartă numele de SNRM (Set Normal Response Mode - stabileşte modul normal de răspuns). Din nefericire, „modul normal de răspuns" numai normal nu este. Este un mod neechilibrat (adică asimetric) în care unul din capetele liniei este master iar celălalt este slave. SNRM datează din timpurile când comunicaţia datelor presupunea un terminal neinteligent comunicând cu un calculator gazdă puternic, ceea ce este, evident, asimetric. Pentru a face protocolul mai potrivit cazurilor în care cei doi parteneri sunt egali, HDLC şi LAPB au o comandă suplimentară, SABM (Set Asynchronous Balanced Mode - stabileşte modul asincron echilibrat), care resetează linia şi declară ambii parteneri ca fiind egali. De asemenea, aceste protocoale au comenzile SABME şi SNRME, care sunt identice cu SABM şi, respectiv, SNRM, cu excepţia faptului că ele permit folosirea unui format extins pentru cadru, care utilizează numere de secvenţă pe 7 biţi în locul unora pe 3 biţi.

A treia comandă prevăzută de toate protocoalele este FRMR (FRaMe Reject), folosită pentru a indica sosirea unui cadru cu suma de control corectă, dar cu semantică imposibilă. Exemple de semantică imposibilă sunt cadru de tipul 3 Supervizor în LAPB, un cadru mai scurt de 32 de biţi, un cadru de control nepermis, confirmarea unui cadru care a fost în afara ferestrei etc. Cadrele FRMR conţin un câmp de date de 24 de biţi care arată ceea ce a fost eronat la cadrul respectiv. Datele includ câmpul de control al cadrului eronat, parametrii ferestrei şi o colecţie de biţi folosiţi pentru a semnala erori specifice.

Cadrele de control pot fi pierdute sau deteriorate ca şi cadrele de date, de aceea şi ele trebuie confirmate. În acest scop este furnizat un cadru special de control, numit UA (Unnumbered Acknowledgement). Deoarece poate exista un singur cadru de control neconfirmat, nu există niciodată ambiguităţi asupra cadrului care este confirmat. Cadrele de control rămase sunt folosite pentru iniţializare, interogare şi raportarea stării. Există, de asemenea, un cadru de control care poate conţine informaţii arbitrare, UI (Unnumbered Information).

Aceste date nu sunt livrate nivelului reţea, ci sunt destinate a fi primite chiar de nivelul legătură de date.

8. Servicii furnizate de nivelul retea nivelului transport

Nivelul reţea furnizează servicii nivelului transport la interfaţa dintre cele două niveluri. O întrebare importantă este ce fel de servicii furnizează nivelul reţea nivelului transport. Serviciile nivelului reţea 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 şi topologia ruterelor existente.

3. Adresele de reţea disponibile la nivelul transport trebuie să folosească o schemă de numerotare uniformă, chiar în cadrul reţelelor LAN şi WAN.

Obiectivele fiind stabilite, proiectantul nivelului reţea are o mare libertate în a scrie specificaţiile detaliate ale serviciilor oferite nivelului transport. Această libertate degenerează adesea într-o aprigă bătălie între două tabere opuse. Problema centrală a discuţiei este dacă nivelul reţea 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 şi nimic mai mult. În viziunea lor (bazată pe experienţa a aproape 30 de ani de exploatare a unei reţele de calculatoare în funcţiune), subreţeaua este inerent nesigură, indiferent cum ar fi proiectată. De aceea calculatoarele gazdă trebuie să accepte faptul că reţeaua este nesigură şi să facă controlul erorilor (i.e., detecţia şi corecţia erorii) şi controlul fluxului ele însele.

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

Cealaltă tabără (reprezentată de companiile de telefoane) afirmă că subreţeaua trebuie să asigure un serviciu orientat pe conexiune sigur. Ei susţin că 100 de ani de experienţă cu sistemul telephonic mondial reprezintă un ghid excelent. În această perspectivă, calitatea serviciului este elementul dominant, şi într-o subreţea fără conexiuni, calitatea serviciului este dificil de obţinut, în special pentru trafic in timp real cum ar fi voce şi imagine.

9. Subretea datagrama si subretea cu circuite virtuale

Dacă este oferit un serviciu neorientat pe conexiune, atunci pachetele sunt trimise in subreţea individual şi dirijate independent de celelalte. Nu este necesară nici o iniţializare prealabilă. În acest context, pachetele sunt numite frecvent **datagrame** (**datagrams**) (prin analogie cu telegramele), iar subreţeaua este numită **subreţea 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 destinaţie. Această conexiune este numită **VC**

(**virtual circuit, circuit virtual**), prin analogie cu circuitele fizice care se stabilesc în sistemul telefonic, iar subreţeaua este numită **subreţea cu circuite virtuale** (**virtual-circuit subnet**).

Să vedem cum funcţionează o **subreţea datagramă**. Să presupunem că procesul P1 din fig. 5-2 are un mesaj lung pentru procesul P2. El transmite mesajul nivelului transport, cu instrucţiunile 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 şi transferă rezultatul nivelului reţea, probabil o altă procedură din cadrul sistemului de operare.

Să presupunem că mesajul este de patru ori mai lung decât dimensiunea maximă a unui pachet, aşa că nivelul reţea trebuie să îl spargă în patru pachete, 1, 2, 3, şi 4 şi 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 telecomunicaţii. Fiecare ruter are o tabelă internă care îi spune unde să trimită pachete pentru fiecare destinaţie posibilă. Fiecare intrare în tabelă este o pereche compusă din destinaţie şi linia de ieşire folosită pentru acea destinaţie. Pot fi folosite doar linii conectate direct. De exemplu, în fig. 5-2, A are doar două linii de ieşire – către B şi C – astfel că fiecare pachet ce vine trebuie trimis către unul dintre aceste rutere, chiar dacă ultima destinaţie este alt ruter. Tabela de rutare iniţială a lui A este prezentată în figură sub eticheta „iniţial”.

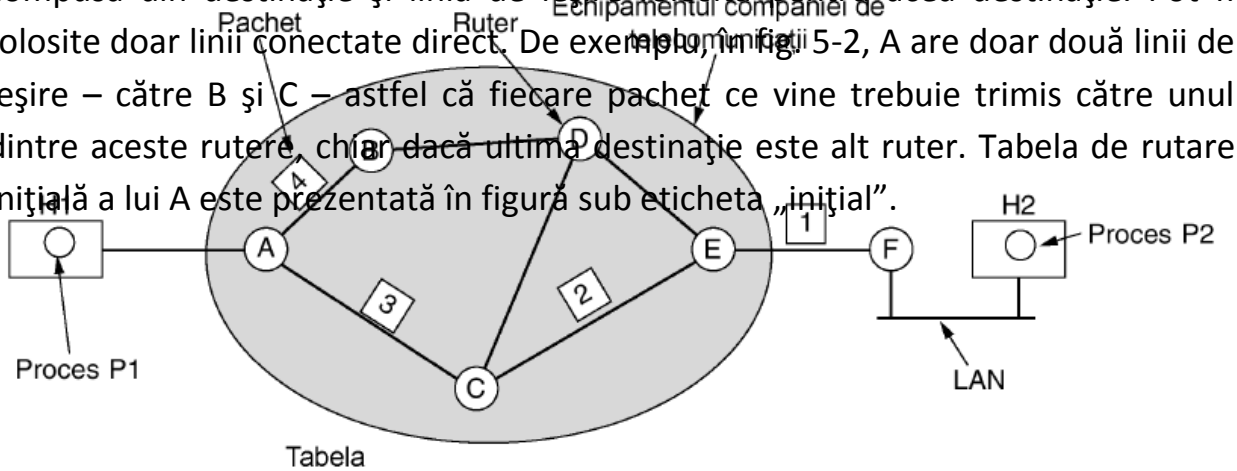


Tabela ruterului A

inițial		ulterior	
A	–	A	–
B	B	B	B
C	C	C	C
D	B	D	B
E	C	E	B
F	C	F	B

Dest. Rută

Tabela ruterului C

A	A
B	A
C	–
D	D
E	E
F	E

Tabela ruterului E

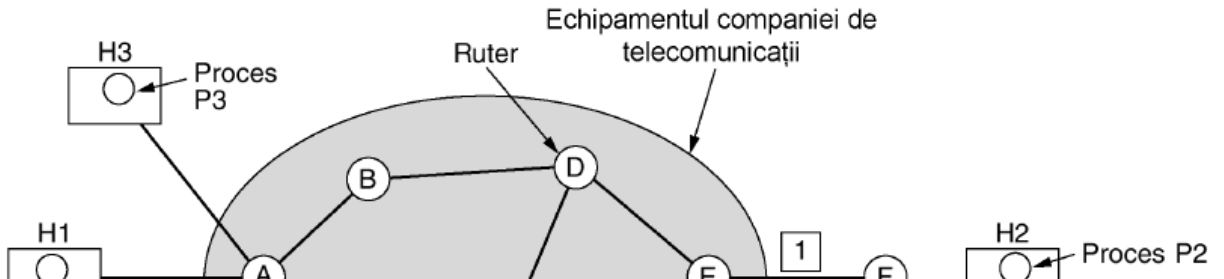
A	C
B	D
C	C
D	D
E	–
F	F

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

Cum au ajuns la A, pachetele 1, 2 şi 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 şi apoi către F. Când a ajuns la F, a fost încapsulat într-un cadru al nivelului legătură de date şi trimis către calculatorul gazdă H2 prin reţeaua LAN.

Totuşi, ceva diferit s-a întâmplat cu pachetul 4. Când a ajuns la A a fost trimis către ruterul B, chiar dacă şi 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 a aflat despre o congestie undeva pe calea ACE şi şi-a actualizat tabela de rutare, aşa cum apare sub eticheta „mai târziu”. Algoritmul ce administrează tabelele şi ia deciziile de rutare se numeşte algoritm de rutare (routing algorithm).

Pentru **serviciile orientate conexiune**, avem nevoie de o subreţea cu circuite virtuale. Să vedem cum funcţionează 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 stabileşte o conexiune, se alege o cale între maşina sursă şi maşina destinaţie, ca parte componentă a iniţializării conexiunii şi aceasta este memorată în tabelele ruterelor. Acea cale este folosită pentru tot traficul de pe conexiune, exact în acelaşi mod în care funcţionează sistemul telefonic. Atunci când conexiunea este eliberată, este închis şi circuitul virtual. În cazul serviciilor orientate conexiune, fiecare pachet poartă un identificator care spune cărui circuit virtual îi aparţine.



De exemplu, să considerăm situaţia 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 iniţializează conexiunea şi aceasta este singura conexiune) şi indică subreţelei să stabilească circuitul virtual. Aceasta conduce la a doua linie din tabele. Observaţi că apare un conflict deoarece deşi A poate distinge uşor pachetele conexiunii 1 de la H1 de pachetele conexiunii 1 de la H3, C nu poate face asta. Din acest motiv, A asociază un identificator de conexiune diferit pentru traficul de ieşire 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 numeşte comutarea etichetelor (label switching).

Comparaţie între subreţele cu circuite virtuale şi subreţele datagramă

Problemă	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.

10. Algoritmi de dirijare

Algoritmul de dirijare (**routing algorithm**) este acea parte a software-ului nivelului reţea care răspunde de alegerea liniei de ieşire pe care trebuie trimis un pachet recepţionat. Dacă subreţeaua foloseşte intern datagrame, această decizie trebuie luată din nou pentru fiecare pachet recepţionat, deoarece este posibil ca cea mai bună rută să se fi modificat între timp. Dacă subreţeaua foloseşte circuite virtuale, deciziile de dirijare sunt luate doar la iniţializarea 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 funcţiune pentru o întreagă sesiune utilizator (de exemplu o sesiune de conectare de la un terminal -login- sau un transfer de fişiere).

Uneori este util să se facă distincţia între dirijare, care înseamnă alegerea căii care va fi folosită, şi retransmitere, care se referă la ceea ce se întâmplă atunci când soseşte un pachet. Se poate spune despre un ruter că rulează intern două procese. Unul dintre ele preia fiecare pachet care soseşte, căutând în tabela de

dirijare linia de ieşire folosită pentru el. Acesta este procesul de retransmitere (forwarding). Celălalt proces se ocupă de completarea şi actualizarea tabelei de rutare. Aici intervine algoritmul de dirijare.

Indiferent dacă ruta se alege independent pentru fiecare pachet sau doar la stabilirea unei noi conexiuni, un algoritm de dirijare trebuie să aibă anumite proprietăţi: corectitudine, simplitate, robusteţe, stabilitate, echitate, optimalitate.

Algoritmii de dirijare pot fi grupaţi în două mari clase: **neadaptivi** şi **adaptivi**. **Algoritmii neadaptivi** (**nonadaptive algorithms**) nu îşi bazează deciziile de dirijare pe măsurători sau estimări ale traficului şi topologiei curente. Astfel, alegerea căii folosite pentru a ajunge de la nodul I la nodul J (oricare ar fi I şi J) se calculează în avans, off-line şi parvine ruterului la iniţializarea reţelei. Această procedură se mai numeşte şi **dirijare statică** (**static routing**).

Algoritmii adaptivi (**adaptive algorithms**), prin contrast, îşi modifică deciziile de dirijare pentru a reflecta modificările de topologie şi de multe ori şi pe cele de trafic. Algoritmii adaptivi diferă prin locul de unde îşi iau informaţia (de exemplu local, de la un ruter vecin sau de la toate ruterele), prin momentul la care schimbă rutele (de exemplu la fiecare ΔT secunde, când se schimbă încărcarea sau când se schimbă topologia) şi prin metrica folosită pentru optimizare (de exemplu distanţa, numărul de salturi sau timpul estimat pentru tranzit).

Algoritmi statici:

- **Algoritmul lui Dijkstra:** Fiecare nod este etichetat (în paranteze) cu distanţa de la nodul sursă până la el, de-a lungul celei mai bune căi cunoscute. Iniţial nu se cunoaşte nici o cale, aşa că toate nodurile vor fi etichetate cu infinit. Pe măsură ce se execută algoritmul şi se găsesc noi căi, etichetele se pot schimba, reflectând căi mai bune. O etichetă poate fi fie temporară, fie permanentă. Iniţial toate etichetele sunt temporare. Atunci când se descoperă că o etichetă reprezintă cea mai scurtă cale posibilă de la sursă către acel nod, ea devine permanentă şi nu se mai schimbă ulterior.

- **Inundarea:** fiecare pachet recepţionat este trimis mai departe pe fiecare linie de ieşire, cu excepţia celei pe care a sosit. Este evident că inundarea generează un mare număr de pachete duplicate, de fapt un număr infinit dacă nu se iau unele măsuri pentru a limita acest proces.
    - O astfel de măsură este păstrarea unui contor de salturi în antetul fiecărui pachet, contor care este decrementat la fiecare salt şi care face ca pachetul să fie distrus când contorul atinge valoarea zero. Ideal ar fi ca acest contor să fie iniţializat cu lungimea căii de la sursă la destinaţie. Dacă emiţătorul nu cunoaşte lungimea căii, poate iniţializa contorul la valoarea cea mai defavorabilă, adică diametrul subreţelei.
    - O metodă alternativă pentru limitarea inundării este identificarea pachetelor care au fost deja inundate, pentru a preîntâmpina trimiterea lor a doua oară. O cale pentru a realiza acest scop este ca ruterul sursă să plaseze un număr de secvenţă în fiecare pachet pe care îl primeşte de la calculatorul gazdă asociat. Fiecare ruter necesită menţinerea unei liste pentru fiecare ruter sursă, cu numerele de secvenţă provenite de la acel ruter sursă şi care au fost deja trimise mai departe. Dacă soseşte un pachet care se află în listă, el nu mai este trimis mai departe. Pentru a limita creşterea lungimii listei, fiecare listă trebuie însoţită de un contor, k, care semnifică faptul că toate numerele de secvenţă până la k au fost deja tratate. La recepţia unui pachet este uşor să se verifice dacă este un duplicat, caz în care este distrus. Evident, lista cu numere mai mici decât k nu este necesară, deoarece k o rezumă.
- **Inundarea selectiva:** ruterele nu trimit fiecare pachet recepţionat pe fiecare legătură de ieşire, ci doar pe acele linii care duc aproximativ în direcţia potrivită. De obicei sunt puţine motive pentru a trimite un pachet spre partea de vest a reţelei folosind o legătură spre est, decât dacă topologia reţelei este cu totul deosebită şi ruterul este sigur de acest lucru.

Algoritmi dinamici:

- **Algoritmul de dirijare cu vectori distanţă (distance vector routing)**
    - Presupune că fiecare ruter menţine o tabelă (de exemplu un vector) care păstrează cea mai bună distanţă cunoscută spre fiecare destinaţie şi linia care trebuie urmată pentru a ajunge acolo. Aceste tabele sunt actualizate prin schimbul de informaţii între nodurile vecine.
    - Algoritmul de dirijare cu vectori distanţă este cunoscut şi sub alte nume, cel mai des algoritmul distribuit de dirijare **Bellman-Ford** şi algoritmul **Ford-Fulkerson**, după numele cercetătorilor care l-au propus.
    - În dirijarea pe baza vectorilor distanţă, fiecare ruter păstrează o tabelă de dirijare conţinând câte o intrare pentru fiecare ruter din subreţea. Această intrare are două părţi: linia de ieşire preferată care se foloseşte pentru destinaţia respectivă şi o estimare a timpului sau distanţei până la acea destinaţie. Metrica folosită poate fi numărul de salturi, întârzierea în milisecunde, numărul total de pachete care aşteaptă în cozi de-a lungul căii, sau ceva asemănător.
    - Se presupune că ruterul cunoaşte „distanţa" spre fiecare dintre vecinii săi. Dacă se foloseşte metrica salturilor, distanţa este doar de un salt. Dacă metrica folosită este cea a lungimii cozilor de aşteptare, ruterul examinează pur şi simplu lungimile acestor cozi. Dacă metrica este cea a întârzierilor, ruterul o poate măsura direct prin pachete speciale ECHO, în care receptorul va marca doar timpul curent (ştampila de timp) şi le va trimite înapoi cât mai repede posibil.
    - Ca exemplu, să presupunem că se foloseşte metrica întârzierilor şi că ruterul cunoaşte întârzierea spre fiecare dintre vecinii săi. O dată la fiecare T msec fiecare ruter trimite spre fiecare vecin o listă a estimărilor proprii spre fiecare destinaţie. De asemenea el recepţionează o listă similară de la fiecare vecin. Să presupunem că una dintre aceste tabele tocmai a sosit de la vecinul X, cu Xi fiind estimarea lui X despre cât timp este necesar pentru a ajunge la ruterul i. Dacă ruterul ştie că întârzierea spre X este m msec, el ştie de asemenea că poate atinge ruterul i trecând prin X în Xi + m msec.

Făcând aceste calcule pentru fiecare vecin, un ruter poate stabili care estimare pare a fi cea mai bună, pentru a folosi această estimare, împreună cu linia corespunzătoare în noua tabelă de dirijare. Este de remarcat faptul că vechea tabelă de dirijare nu intervine în calcule.

- **Problema numararii la infinit**
  - Dirijarea folosind vectori distanţă funcţionează în teorie, însă în practică are o limitare importantă: deşi ea converge spre rezultatul corect, o face foarte lent. În particular, ea reacţionează rapid la veştile bune, dar foarte lent la cele rele. Să considerăm un ruter care are un cel mai bun drum spre destinaţia X foarte lung. Dacă la următorul schimb de informaţii, vecinul său A raportează brusc o întârziere mică spre X, ruterul va comuta şi va folosi linia spre A pentru a dirija traficul spre X. Astfel, într-o singură schimbare a vectorului, vestea bună a fost luată în considerare. Într-o subreţea având calea cea mai lungă de lungime N salturi, după N schimburi fiecare ruter va afla despre liniile şi ruterele nou apărute.
  - Să considerăm acum situaţia în care toate liniile şi ruterele sunt iniţial în funcţiune. Ruterele B, C, D şi E au distanţele spre A respectiv de 1, 2, 3, 4. Brusc, A se opreşte sau, alternativ, linia dintre A şi B este întreruptă, ceea ce reprezintă efectiv acelaşi lucru din punctul de vedere al lui B.
  - La primul schimb de pachete, B nu primeşte nimic de la A. Din fericire, C spune: „Nici o problemă. Eu ştiu o cale spre A de lungime 2.” Însă B nu ştie că această cale a lui C trece prin B însuşi. După cunoştinţele lui B, C ar putea avea zece linii, toate cu căi separate de lungime 2 spre A. Prin urmare B va crede că poate ajunge la A prin C pe o cale de lungime 3. D şi E nu îşi actualizează intrările proprii pentru A la primul schimb.
  - La al doilea schimb, C remarcă faptul că fiecare dintre vecinii săi pretinde a avea o cale de lungime 3 spre A. El va alege la întâmplare unul dintre aceşti vecini şi va înregistra noua distanţă spre A ca fiind 4.

        - Din această figură se poate deduce de ce veştile rele circulă mai lent: orice ruter va avea întotdeauna o valoare cu cel mult unu mai mare decât valoarea minimă a vecinilor săi. Treptat, toate ruterele vor ajunge la infinit, însă numărul de schimburi necesar depinde de valoarea numerică folosită pentru a reprezenta valoarea infinit. Din această cauză este recomandat să se aleagă infinitul, ca fiind lungimea celei mai mari căi, plus 1. Dacă metrica este întârzierea în timp, atunci nu este definite nici o limită superioară, astfel încât este necesară o valoare mare pentru a preveni considerarea unui drum cu întârziere mare ca fiind un drum defect. Nu este deloc surprinzător că această problem este numită problema numărării la infinit (the count to infinity problem).
        - Miezul problemei este că atunci când X îi spune lui Y că are o cale spre o destinaţie, Y nu are de unde să ştie dacă se află el însuşi pe acea cale.

- **Dirijarea folosind starea legaturilor**
    - Ideea algoritmului bazat pe starea legăturilor este simplă şi poate fi formulată în 5 puncte. Fiecare ruter trebuie să facă următoarele:
        - Să descopere care sunt vecinii săi şi afle adresele de reţea ale acestora.
        - Să măsoare întârzierea sau costul până la fiecare din vecinii săi.
        - Să pregătească un pachet prin care anunţă pe toată lumea că tocmai a terminat de cules datele despre vecini.
        - Să trimită acest pachet către toate celelalte rutere.
        - Să calculeze cea mai scurtă cale spre fiecare ruter.
    - Ca urmare, întreaga topologie şi toate întârzierile sunt măsurate experimental şi distribuite spre fiecare ruter. Apoi se poate rula

algoritmul lui Dijkstra pentru a afla cea mai scurtă cale către fiecare ruter.

- **Determinarea vecinilor:** Când un ruter este pus în funcţiune, prima sa sarcină este să afle care sunt vecinii săi. El realizează aceasta prin trimiterea unui pachet special HELLO pe fiecare linie prin care este legat la alt ruter. Ruterul de la celălalt capăt trebuie să răspundă anunţând într-un pachet identitatea sa. Aceste nume trebuie să fie unice global, pentru că dacă mai târziu un ruter află că trei rutere sunt conectate toate la F, este esenţial ca acesta să poată determina dacă cele trei se referă la acelaşi F.
- **Măsurarea costului liniei:**
  - Algoritmul de dirijare bazat pe starea legăturilor cere ca fiecare ruter să ştie, sau cel puţin să aibă o estimare rezonabilă, a întârzierii către fiecare dintre vecinii săi. Cel mai direct mod de a afla acest lucru este de a trimite un pachet special ECHO pe linie, cerând ca ruterul partener să-l trimită înapoi imediat. Măsurând timpul în care pachetul se întoarce (round-trip time) şi împărţindu-l la doi, ruterul iniţiator poate avea o estimare rezonabilă a întârzierii. Pentru rezultate şi mai bune, testul poate fi repetat de mai multe ori, folosindu-se apoi valoarea medie obţinută. Bineînţeles, această metodă presupune implicit că întârzierile sunt simetrice, dar nu mereu este aşa.
  - O problemă interesantă este dacă să se considere sau nu încărcarea reţelei la măsurarea întârzierii. Pentru a ţine cont de încărcare, timpul de revenire trebuie măsurat din momentul în care pachetul ECHO este pus în coadă. Pentru a ignora încărcarea, ceasul se poate porni în momentul în care pachetul ECHO ajunge pe prima poziţie din coadă.
- **Construirea pachetelor cu starea legăturilor:**
  - De îndată ce a fost colectată informaţia necesară pentru realizarea schimbului, se poate trece la pasul următor, fiecare ruter construind un pachet care conţine toate datele. Pachetul începe cu identitatea expeditorului, urmată de un număr de

secvenţă, vârstă şi o listă a vecinilor. Pentru fiecare vecin se specifică întârzierea asociată.

- Construirea pachetelor cu starea legăturilor se face uşor. Partea mai dificilă este să se determine când să fie construite ele. O posibilitate este ca ele să fie construite periodic, adică la intervale regulate. O altă posibilitate este atunci când se produce un eveniment semnificativ, cum ar fi scoaterea din funcţiune a unui vecin sau a unei linii, sau repunerea lor în funcţiune, sau modificarea semnificativă a proprietăţilor lor.

- **Distribuirea pachetelor cu starea legăturilor:**
  - Cea mai complicată parte a algoritmului este distribuirea sigură a pachetelor cu starea legăturilor. De îndată ce pachetele sunt distribuite şi instalate, ruterele care primesc primele pachete îşi vor schimba rutele. În consecinţă, rutere diferite ar putea folosi versiuni diferite ale topologiei, ceea ce poate duce la apariţia unor inconsistenţe, bucle, maşini inaccesibile sau a altor probleme.
  - Ideea fundamentală este folosirea inundării pentru a distribui pachetele cu starea legăturilor. Pentru a avea controlul inundării, fiecare pachet conţine un număr de secvenţă care este incrementat la fiecare nou pachet trimis. Ruterele păstrează evidenţa tuturor perechilor (ruter sursă, număr secvenţă) pe care le văd. La sosirea unui nou pachet cu starea legăturilor, el este căutat în lista pachetelor deja văzute. Dacă pachetul este nou, el este retrimis pe toate liniile, cu excepţia celei pe care a sosit. Dacă este un duplicat, pachetul este distrus. Dacă pachetul sosit are un număr de secvenţă mai mic decât cel mai mare număr de secvenţă detectat, atunci el este rejectat ca fiind învechit.
  - Acest algoritm are câteva probleme, dar ele sunt tratabile. În primul rând, dacă numerele de secvenţă ating valoarea maximă posibilă şi reîncep de la valori mici, se pot produce confuzii. Soluţia este folosirea numerelor de secvenţă pe 32

biţi. Considerând un pachet de starea legăturilor pe secundă, ar trebui 137 ani pentru a se reîncepe de la valori mici, astfel încât această posibilitate poate fi ignorată.

- În al doilea rând, dacă un ruter se defectează, el va pierde evidenţa numerelor de secvenţă. Dacă va reîncepe de la 0, următorul pachet va fi rejectat ca duplicat.
- În al treilea rând, dacă numărul de secvenţă este alterat şi se recepţionează 65540 în loc de 4 (eroare de modificare a unui bit), pachetele de la 5 la 65540 vor fi rejectate ca fiind învechite, deoarece se consideră că numărul de secvenţă curent este 65540.
- Soluţia tuturor acestor probleme este includerea vârstei pachetului după numărul de secvenţă şi decrementarea sa la fiecare secundă. Când vârsta ajunge la zero, informaţia de la ruterul respective este distrusă. În mod normal un nou pachet apare, să zicem, la fiecare 10 sec., astfel încât informaţia de la ruter expiră doar dacă ruterul este oprit (sau dacă şase pachete consecutive s-au pierdut, un lucru extrem de improbabil). Câmpul Age este decrementat de asemenea de fiecare ruter la începutul procesului de inundare, pentru a se asigura că nici un pachet nu se poate pierde sau nu poate supravieţui o perioadă nelimitată (un pachet având vârsta zero este distrus).

- **Calcularea noilor rute:**
  - De îndată ce un ruter a acumulat un set complet de pachete cu starea legăturilor, el poate construi graful întregii subreţele, deoarece fiecare legătură este reprezentată. De fapt, fiecare legătură este reprezentată de două ori, o dată pentru fiecare direcţie. Cele două valori pot fi mediate sau folosite separat.
  - Acum poate fi folosit local algoritmul lui Dijkstra, pentru a construi cea mai scurtă cale către toate destinaţiile posibile. Rezultatul acestui algoritm poate fi trecut în tabelele de dirijare, iar apoi operaţiile normale pot fi reluate.

11. Dirijarea prin difuzare

În unele aplicaţii, calculatoarele gazdă au nevoie să trimită mesaje către mai multe sau către toate celelalte calculatoare gazdă. De exemplu, un serviciu de distribuire a rapoartelor meteorologice, de actualizare a cursului acţiunilor sau transmisiuni radio în direct ar putea funcţiona mai bine prin difuzarea datelor către toate maşinile, fiind la latitudinea celor interesate de date să le recepţioneze. Trimiterea simultană a unui pachet către toate destinaţiile se numeşte difuzare (broadcast); mai multe metode pentru realizarea ei au fost propuse.

O metodă de difuzare care nu are cerinţe speciale pentru subreţea este ca sursa să trimită un pachet distinct către fiecare destinaţie. Metoda este nu numai consumatoare de lăţime de bandă, dar ea cere ca sursa să deţină o listă completă a tuturor destinaţiilor. S-ar putea ca în practică aceasta să fie singura metodă utilizabilă, însă ea este metoda cea mai puţin dorită.

Inundarea este un alt candidat evident. Deşi inundarea este nepotrivită pentru comunicaţia obişnuită capăt la capăt, pentru difuzare ar trebui luată serios în considerare, mai ales dacă nici una dintre metodele ce vor fi descrise în continuare nu este aplicabilă. Problema folosirii inundării ca metodă de difuzare este aceeaşi cu problema ivită la folosirea sa ca algoritm de dirijare capăt la capăt: generează prea multe pachete şi consumă lăţime de bandă prea mare.

Al treilea algoritm este dirijarea multidestinaţie (multidestination routing). Dacă se foloseşte această metodă, fiecare pachet conţine fie o listă a destinaţiilor, fie o hartă de biţi care indică destinaţiile dorite. Atunci când un pachet ajunge la un ruter, ruterul verifică toate destinaţiile pentru adetermina setul liniilor de ieşire pe care trebuie trimis pachetul. (O linie de ieşire este selectată dacă este calea cea mai bună pentru cel puţin o destinaţie.) Ruterul generează o nouă copie a pachetului pentru fiecare linie de ieşire folosită şi include în fiecare pachet doar acele destinaţii care folosesclinia respectivă. Efectul este partiţionarea mulţimii destinaţiilor între liniile de ieşire. După un număr suficient de salturi, fiecare pachet va conţine o singură destinaţie şi poate fi tratat ca un pachet normal. Dirijarea multidestinaţie este asemănătoare trimiterii separate a mai multor

pachete către adresele destinaţie, cu excepţia faptului că atunci când mai multe pachete trebuie să urmeze aceeaşi cale, unul dintre ele plăteşte tot drumul, iar celelalte călătoresc gratuit.

Al patrulea algoritm de difuzare utilizează explicit arborele de scufundare al ruterului care iniţiază difuzarea sau orice alt arbore de acoperire util. Un arbore de acoperire (spanning tree) este un subset al subreţelei care include toate ruterele şi nu conţine bucle. Dacă fiecare ruter cunoaşte care din liniile sale participă la arborele de acoperire, el poate copia un pachet de difuzare recepţionat pe toate liniile de ieşire care fac parte din arborele de acoperire, cu excepţia celei pe care a fost recepţionat. Această metodă asigură o utilizare deosebit de eficientă a lăţimii de bandă, generând numărul minim de pachete necesare pentru a rezolva problema. Singura problemă este că, pentru a fi aplicabilă, fiecare ruter trebuie să deţină cunoştinţe despre un anume arbore de acoperire. Uneori această informaţie este disponibilă (de exemplu la dirijarea bazată pe starea legăturilor), iar alteori nu este disponibilă (de exemplu la dirijarea cu vectori distanţă).

Ultimul nostru algoritm cu difuzare este o încercare de a aproxima comportamentul precedentului, chiar şi atunci când ruterele nu ştiu absolut nimic despre arborele de acoperire. Ideea, numită trimiterea pe calea inversă (reverse path forwarding) este remarcabil de simplă odată ce a fost indicată. Când un pachet de difuzare ajunge la un ruter, acesta verifică dacă pachetul a sosit pe linia pe care se trimit de obicei pachete către sursa difuzării. Dacă este aşa, este o şansă foarte mare ca însuşi pachetul de difuzare să fi urmat cea mai bună cale, fiind astfel prima copie care ajunge la ruter. Aceasta fiind situaţia, ruterul trimite pachetul pe toate liniile de ieşire, cu excepţia celei pe care a sosit. Dacă însă pachetul a sosit pe altă linie decât cea preferată pentru a ajunge la sursă, pachetul este distrus, fiind considerat un posibil duplicat.

12. Dirijarea cu trimitere multipla (multicast)

Unele aplicaţii necesită ca procese aflate la mari distanţe unele de altele să lucreze în grup, de exemplu, un grup de procese care implementează o bază de date distribuită. În aceste situaţii, este adesea necesar ca un proces să trimită un

mesaj către toţi ceilalţi membri ai grupului. Dacă grupul este mic, el poate trimite fiecărui partener un mesaj capăt la capăt. Dacă grupul este mare, această strategie este costisitoare. Uneori se poate folosi difuzarea, dar folosirea difuzării pentru a anunţa 1000 de maşini dintr-o reţea cu un milion de noduri este ineficientă, deoarece majoritatea receptorilor nu sunt interesaţi de mesaj (sau chiar mai rău, sunt foarte interesaţi, dar nu trebuie să vadă mesajul). De aceea, avem nevoie de o modalitate de a trimite mesaje spre grupuri bine definite.

Trimiterea unui mesaj către un astfel de grup se numeşte multicasting, iar algoritmul de dirijare asociat se numeşte **dirijare multicast** (**multicast routing**). În această secţiune, vom descrie o modalitate de a realiza dirijarea multicast.

Dirijarea multicast necesită managementul grupului. Trebuie să existe modalităţi de a crea şi distruge grupuri şi de a permite proceselor să intre în grupuri sau să le părăsească. Cum se realizează aceste funcţii nu este treaba algoritmului de dirijare. Important pentru algoritmul de dirijare este că atunci când un proces se ataşează unui grup, el trebuie să informeze gazda sa despre aceasta. Este important ca ruterele să ştie căror grupuri le aparţin calculatoarele gazdă asociate. Fie calculatoarele gazdă trebuie să anunţe ruterul asociat la producerea unei modificări în alcătuirea grupurilor, fie ruterul trebuie să interogheze periodic aceste calculatoare gazdă. În ambele cazuri, ruterul află din ce grupuri fac parte calculatoarele gazdă. Ruterele îşi informează vecinii, astfel că informaţia se propagă prin subreţea.

Pentru a realiza dirijarea multicast, fiecare ruter calculează arborele de acoperire care acoperă toate celelalte rutere din subreţea. Atunci când un proces trimite un pachet multicast către un grup, primul ruter îşi examinează arborele de acoperire şi îl retează, eliminând toate liniile care nu conduc către calculatoare gazdă, membre ale grupului. Pachetele multicast sunt dirijate doar de-a lungul arborelui de acoperire corespunzător.

Sunt posibile mai multe moduri de retezare a arborelui de acoperire. Cel mai simplu se poate folosi dacă se utilizează dirijarea bazată pe starea legăturilor şi fiecare ruter cunoaşte întreaga topologie a subreţelei, inclusiv apartenenţa calculatoarelor gazdă la grupuri. Atunci arborele poate fi retezat pornind de la

sfârşitul fiecărei căi, mergând spre rădăcină şi eliminând toate ruterele care nu aparţin grupului respectiv.

În cazul dirijării folosind vectori distanţă, poate fi aplicată o altă strategie de retezare a arborelui. Algoritmul de bază folosit este trimiterea pe calea inversă. Oricum, ori de câte ori un ruter fără nici un calculator gazdă interesat de un anume grup şi fără nici o conexiune la alte rutere primeşte un mesaj multicast pentru acel grup, el va răspunde cu un mesaj PRUNE (retezare), anunţând expeditorul să nu îi mai trimită mesaje multicast pentru acel grup. Când un ruter care nu are printre calculatoarele gazdă nici un membru al vreunui grup primeşte astfel de mesaje pe toate liniile sale, el poate de asemenea să răspundă cu un mesaj PRUNE. În acest fel subreţeaua este retezată recursiv.

13. Dirijarea pentru calculatoare gazdă mobile

Calculatoarele gazdă care nu se mişcă niciodată se numesc staţionare. Ele sunt conectate la reţea prin fire de cupru sau fibre optice. Prin contrast, putem distinge alte două categorii de calculatoare gazdă. Calculatoarele gazdă migratoare sunt de fapt calculatoare gazdă staţionare, dar care, din când în când, se mută dintr-un loc fixat în altul şi care folosesc reţeaua doar atunci când sunt conectate fizic. Calculatoarele gazdă călătoare efectuează calcule în timp ce se deplasează şi doresc să menţină legătura în timpul deplasării. Vom folosi termenul gazdă mobilă pentru a desemna fiecare dintre ultimele două categorii, adică toate calculatoarele gazdă care sunt departe de casă şi doresc totuşi să fie conectate.

Fiecare domeniu are unul sau mai mulţi agenţi pentru străini (foreign agent), procese ce ţin evidenţa tuturor gazdelor mobile care vizitează domeniul. În plus, fiecare domeniu are un agent local (local agent) care ţine evidenţa calculatoarelor gazdă cu domiciliul în domeniul respectiv, dar care momentan vizitează alte domenii.

Când un nou calculator gazdă pătrunde într-un domeniu, fie prin conectarea la acesta (ataşarea fizică la LAN), fie prin plimbarea printr-o celulă,

trebuie să se înregistreze la agentul pentru străini din domeniul respectiv. Procedura de înregistrare se desfăşoară de obicei astfel:

- Periodic, fiecare agent pentru străini difuzează un pachet anunţându-şi existenţa şi adresa. Un utilizator mobil nou sosit trebuie să aştepte unul dintre aceste mesaje, însă dacă niciunul nu soseşte într-un interval rezonabil de timp, calculatorul mobil poate difuza un pachet care spune: „Este vreun agent pentru străini prin zonă?”
- Calculatorul mobil se înregistrează la agentul pentru străini precizând adresa sa permanentă, adresa actuală a nivelului legătură de date, precum şi unele informaţii de securitate.
- Agentul pentru străini contactează apoi agentul local al domeniului din care provine utilizatorul mobil şi îi spune: „Unul dintre calculatoarele tale gazdă se află chiar aici.” Mesajul agentului pentru străini către agentul local conţine adresa de reţea a agentului pentru străini. El mai conţine de asemenea şi informaţia de securitate, pentru a convinge agentul local că gazda mobilă este într-adevăr acolo.
- Agentul local examinează informaţia de securitate, care conţine şi o ştampilă de timp, pentru a dovedi că a fost generată în ultimele câteva secunde. Dacă este mulţumit, atunci anunţă agentul pentru străini că poate trece la acţiune.
- Când agentul pentru străini primeşte confirmarea de la agentul local, el creează o nouă intrare în tabela sa şi anunţă utilizatorul mobil că a fost înregistrat.

Ideal, atunci când un calculator gazdă părăseşte un domeniu, acest lucru trebuie anunţat, pentru a fi scos din evidenţă, însă mulţi utilizatori pur şi simplu închid calculatorul când termină.

Agentul local face două lucruri cand primeste un pachet pentru gazda mobila. În primul rând, încapsulează pachetul în câmpul de informaţie utilă (payload) al unui pachet de trimis, pe care îl expediază apoi către agentul pentru străini. Acest mecanism se numeşte tunelare. După ce recepţionează pachetul

încapsulat, agentul pentru străini extrage pachetul iniţial din câmpul payload şi îl trimite către calculatorul gazdă mobil drept cadru al nivelului legătură de date.

În al doilea rând, agentul local anunţă expeditorul mesajului ca de acum încolo să trimită pachetele adresate gazdei mobile încapsulându-le în câmpul de informaţie utilă (payload) al unor pachete trimise explicit agentului pentru străini, în loc să le trimită la adresa de domiciliu a calculatorului gazdă mobil. Pachetele următoare vor putea fi dirijate direct către utilizator, prin intermediul agentului pentru străini, evitând trecerea prin locaţia de domiciliu.

14. Dirijarea în reţele AD HOC

Reţelele de noduri ce întâmplător se află aproape unul de celălalt sunt numite reţele **ad hoc** (**ad hoc network**) sau **MANET** (**Mobile Ad hoc NETworks**, reţele mobile ad hoc).

Au fost propuşi diferiţi algoritmi de dirijare pentru reţelele ad hoc. Unul dintre cei mai interesanţi este algoritmul de dirijare **AODV** (**Ad hoc On-demand Distance Vector**, vectori distanţă ad hoc la cerere). Este o rudă îndepărtată a algoritmului cu vectori distanţă Bellman-Ford dar adaptat pentru un mediu mobil şi care ia în considerare limitele lărgimii de bandă şi durata scurtă de funcţionare a unei baterii în acest mediu. O altă caracteristică neobişnuită este faptul că acesta este un algoritm la cerere, adică determină o cale către o anumită destinaţie doar atunci când cineva doreşte să trimită un pachet către acea destinaţie.

La orice moment de timp, o reţea ad hoc poate fi descrisă de un graf de noduri (rutere + gazde). Două noduri sunt conectate (de exemplu, există un arc ce le uneşte în graf) dacă pot comunica direct folosind radioul. Algoritmul AODV menţine o tabelă în fiecare nod, în care cheia este destinaţia, şi care dă informaţii despre acea destinaţie, inclusiv cărui vecin trebuie trimise pachetele pentru a ajunge la destinaţie. Operatii care se realizeaza in nodurile intermediare atat pe ruta de la dus, cat si la intoarcere :

Adresa sursei	ID cerere	Adresa destinației	Numărul de sec-vență al sursei	Numărul de sec-vență al destinației	Contorul de salturi

Fig. 5-21. Formatul unui pachet ROUTE REQUEST

Adresa sursei	Adresa destinației	Numărul de sec-vență al destinației	Contorul de salturi	Durata de viață

Fig. 5-22. Formatul pachetului ROUTE REPLY

15. Protocolul IP

O datagramă IP constă dintr-o parte de antet şi o parte de text. Antetul are o parte fixă de 20 de octeţi şi o parte opţională 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 puţin semnificativ primul). Pe maşinile de tip little endian, este necesară o conversie prin program atât la transmisie cât şi la recepţie.

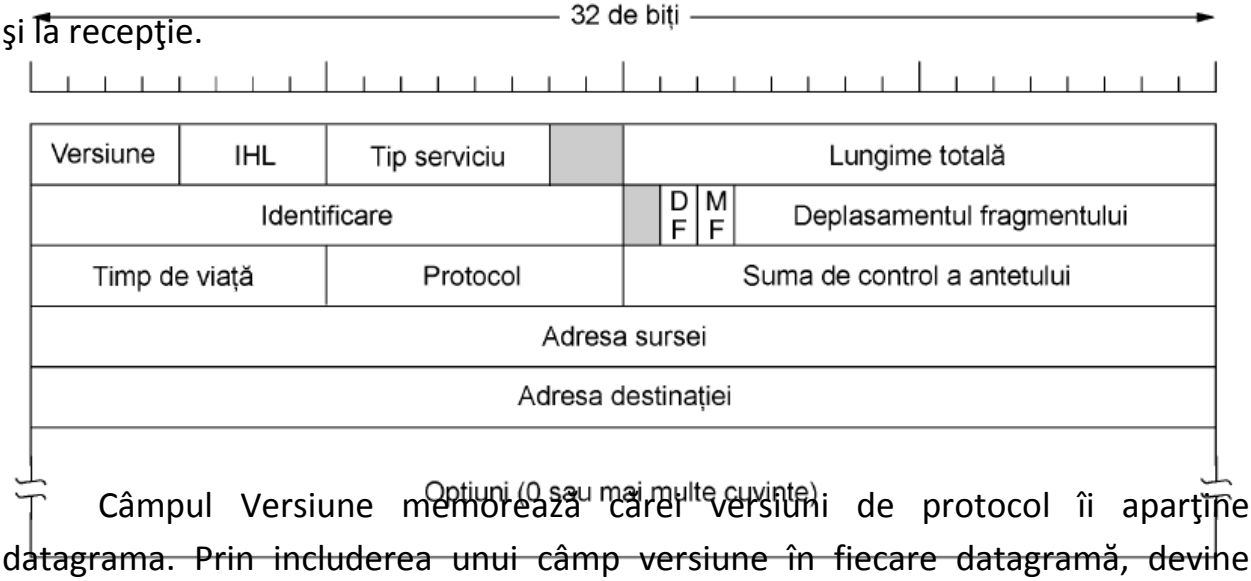


Fig. 5-53. Antetul IPv4 (protocolul Internet).

Câmpul Versiune memorează cărei versiuni de protocol îi aparţine datagrama. Prin includerea unui câmp versiune în fiecare datagramă, devine posibil ca tranziţia între versiuni să dureze ani de zile, cu unele maşini rulând vechea versiune, iar altele rulând-o pe cea nouă. La ora actuală are loc o tranziţie de la IPv4 la IPv6, care deja durează de câţiva ani şi în nici un caz nu s-a apropiat de final.

Din moment ce lungimea antetului nu este constantă, un câmp din antet, IHL, este pus la dispoziţie pentru a spune cât de lung este antetul, în cuvinte de 32 de octeţi. Valoarea minimă este 5, care se aplică atunci când nu sunt prezente opţiuni. Valoarea maximă a acestui câmp de 4 biţi este 15, ceea ce limitează antetul la 60 de octeţi şi, astfel, câmpul de opţiuni la 40 de octeţi. Pentru unele opţiuni, cum ar fi cea care înregistrează calea pe care a mers un pachet, 40 de octeţi sunt mult prea puţini, făcând această opţiune nefolositoare.

Câmpul Tip serviciu este unul dintre puţinele câmpuri care şi-a schimbat sensul (oarecum) dea lungul anilor. A fost şi este în continuare menit să diferenţieze diferitele clase de servicii. Sunt posibile diferite combinaţii de fiabilitate şi viteză. Pentru vocea digitizată, livrarea rapidă are prioritate faţă de transmisia corectă. Pentru transferul de fişiere, transmisia fără erori este mai importantă decât transmisia rapidă.

La început, câmpul de 6 biţi conţinea (de la stânga la dreapta), un câmp Precedenţă de trei biţi şi trei indicatori, D, T şi R. Câmpul Precedenţă reprezintă prioritatea, de la 0 (normal) la 7 (pachet de control al reţelei). Cei trei biţi indicatori permiteau calculatorului gazdă să specifice ce îl afectează cel mai mult din mulţimea {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 şi o întârziere mare sau o linie dedicată cu o productivitate scăzută şi o întârziere mică. În practică, ruterele curente ignoră adesea întregul câmp Tip serviciu.

Până la urmă, IETF a cedat şi a modificat puţin câmpul pentru a-l adapta la servicii diferenţiate. Şase dintre biţi sunt folosiţi pentru a indica căreia dintre clasele de servicii descrise mai devreme îi aparţine pachetul. Aceste clase includ patru priorităţi de introducere în coadă, trei probabilităţi de respingere şi clasele istorice.

Lungimea totală include totul din datagramă - atât antet cât şi date. Lungimea maximă este de 65535 octeţi. În prezent, această limită superioară este tolerabilă, dar în viitoarele reţele cu capacităţi de gigaocteţi vor fi necesare datagrame mai mari.

Câmpul Identificare este necesar pentru a permite gazdei destinaţie să determine cărei datagrame îi aparţine un nou pachet primit. Toate fragmentele unei datagrame conţin aceeaşi valoare de Identificare.

Urmează un bit nefolosit şi apoi două câmpuri de 1 bit. DF însemnă Don't Fragment (A nu se fragmenta). Acesta este un ordin dat ruterelor ca să nu fragmenteze datagrama pentru că destinaţia nu este capabilă să reasambleze piesele la loc. De exemplu, când un calculator porneşte, memoria sa ROM poate cere să i se trimită o imagine de memorie ca o singură datagramă. Prin marcarea datagramei cu bitul DF, emiţătorul ştie că aceasta va ajunge într-o singură bucată, chiar dacă aceasta înseamnă că datagrama trebuie să evite o reţea cu pachete mai mici pe calea cea mai bună şi să aleagă o rută suboptimală. Toate maşinile trebuie să accepte fragmente de 576 octeţi sau mai mici.

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

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

Câmpul Timp de viaţă este un contor folosit pentru a limita durata de viaţă a pachetelor. Este prevăzut să contorizeze timpul în secunde, permiţând un timp maxim de viaţă de 255 secunde. El trebuie să fie decrementat la fiecare salt (hop - trecere dintr-o reţea în alta) şi 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 şi se trimite înapoi la gazda sursă un pachet de avertisment. Această facilitate previne hoinăreala la infinit a datagramelor, ceea ce se poate întâmpla dacă tabelele de dirijare devin incoerente.

Când nivelul reţea a asamblat o datagramă completă, trebuie să ştie ce să facă cu ea. Câmpul Protocol spune cărui proces de transport trebuie să o predea. TCP este o posibilitate, dar tot aşa sunt şi UDP şi alte câteva. Numerotarea protocoalelor este globală la nivelul întregului Internet.

Suma de control a antetului verifică numai antetul. O astfel de sumă de control este utilă pentru detectarea erorilor generate de locaţii de memorie proaste din interiorul unui ruter. Algoritmul este de a aduna toate jumătăţile de cuvinte, de 16 biţi, atunci când acestea sosesc, folosind aritmetică în complement faţă de unu şi păstrarea complementului faţă de unu al rezultatului. Pentru scopul acestui algoritm, se presupune că la sosire suma de control a antetului este zero. Acest algoritm este mai robust decât folosirea unei adunări normale. Observaţi că suma de control a antetului trebuie recalculate la fiecare salt, pentru că întotdeauna se schimbă cel puţin un câmp (câmpul timp de viaţă), dar se pot folosi trucuri pentru a accelera calculul.

Adresa sursei şi Adresa destinaţiei indică numărul de reţea şi numărul de gazdă. Vom discuta adresele Internet în secţiunea următoare.

Câmpul Opţiuni a fost proiectat pentru a oferi un subterfugiu care să permită versiunilor viitoare ale protocolului să includă informaţii care nu sunt prezente în proiectul original, pentru a permite cercetătorilor să încerce noi idei şi pentru a evita alocarea unor biţi din antet pentru informaţii folosite rar. Opţiunile sunt de lungime variabilă. Fiecare începe cu un cod de un octet care identifică opţiunea. Unele opţiuni sunt urmate de un câmp de un octet reprezentând lungimea opţiunii, urmat de unul sau mai mulţi octeţi de date. Câmpul Opţiuni este completat până la un multiplu de 4 octeţi.

16. Adrese IP

Fiecare gazdă şi ruter din Internet are o adresă IP, care codifică adresa sa de reţea şi de gazdă. Combinaţia este unică: în principiu nu există două maşini cu

aceeaşi adresă IP. Toate adresele IP sunt de 32 de biţi lungime şi sunt folosite în câmpurile Adresă sursă şi Adresă destinaţie 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 interfaţă de reţea, deci dacă o gazdă este în două reţele, trebuie să folosească două adrese IP. Totuşi în practică, cele mai multe gazde sunt conectate la o singură reţea şi deci au o adresă IP.

Timp de mai multe decenii, adresele IP erau împărţite în cinci categorii ilustrate în fig. 5-55. Acest model de alocare a fost denumit clase de adrese. Nu mai este folosit, dar referinţele la acest model sunt în continuare des întâlnite în literatură.

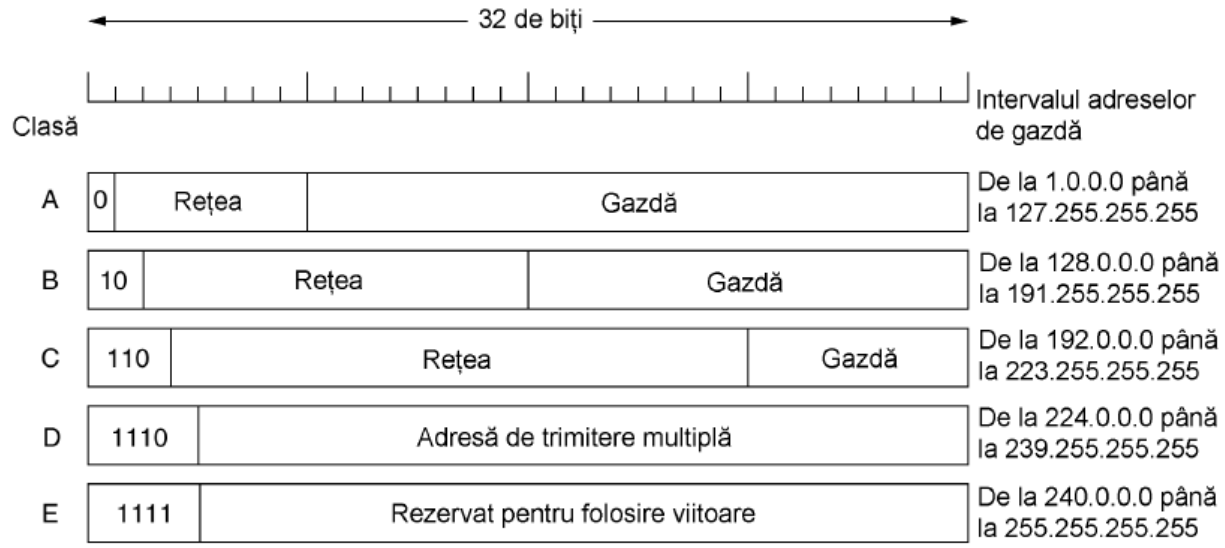


Fig. 5-55. Formatul adreselor IP.

Adresele de reţea, care sunt numere de 32 de biţi, sunt scrise în mod uzual în notaţia zecimală cu punct. În acest format, fiecare din cei 4 octeţi 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 şi cea mai mare este 255.255.255.255.

Valorile 0 şi -1 au semnificaţii speciale. Valoarea 0 înseamnă reţeaua curentă sau gazda curentă. Valoarea -1 este folosită ca o adresă de difuzare pentru a desemna toate gazdele din reţeaua indicată.

Insuficienta adreselor IP este o problema. Soluţia care a fost implementată acum şi care a dat Internet-ului un pic de spaţiu de manevră este **CIDR** (**Classless InterDomain Routing** - Dirijarea fără clase între domenii). Ideea de la baza CIDR, este de a aloca adresele IP rămase, în blocuri de dimensiune variabilă, fără a se ţine cont de clase. Dacă un site are nevoie, să zicem, de 2000 de adrese, îi este dat un bloc de 2048 adrese la o graniţă de 2048 de octeţi.

Renunţarea 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 biţi pentru a obţine un număr de clasă de 4 biţi. O ramificaţie cu 16 căi sorta pachetele în A, B, C şi D (dacă era suportat), cu 8 cazuri pentru clasa A, patru cazuri pentru clasa B, două cazuri pentru clasa C şi câte unul pentru D şi E. Programul pentru fiecare clasă masca numărul de reţea de 8, 16 sau 24 de biţi şi îl alinia la dreapta într-un cuvânt de 32 de biţi. Numărul de reţea era apoi căutat în tabela pentru A, B sau C, de obicei indexat pentru reţelele A şi B şi folosind dispersia (hashing) pentru reţelele C. Odată intrarea găsită, se putea găsi linia de ieşire şi pachetul era retransmis.

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

17. NAT – Translatarea adreselor de retea

Adresele IP sunt insuficiente. Un ISP ar putea avea o adresă /16 (anterior de clasă B) oferindu-i 65.534 numere de staţii. Dacă are mai mulţi clienţi, are o problemă.

Problema epuizării adreselor IP nu este o problemă teoretică care ar putea apărea cândva în viitorul îndepărtat. A apărut aici şi acum. Soluţia pe termen lung este ca tot Internetul să migreze la IPv6, care are adrese de 128 de biţi. Tranziţia se desfăşoară încet, dar vor trece ani până la finalizarea procesului. În consecinţă, 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 reţea).

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 primeşte o adresă IP unică, care este folosită pentru traficul intern. Totuşi, atunci când un pachet părăseşte compania şi 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 conţinând 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 dispoziţie 16.777.216 adrese (cu excepţia adreselor 0 şi –1, ca de obicei) şi este alegerea obişnuită a majorităţii companiilor, chiar dacă nu au nevoie de aşa de multe adrese.

În interiorul companiei fiecare maşină are o adresă unică de forma 10.x.y.z. Totuşi, când un pachet părăseşte compania, trece printr-o **unitate NAT** (eng.: **NAT box**)care converteşte adresa IP internă la adresa IP reală a 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 ştie unitatea NAT cu care adresă să o înlocuiască 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 emiţător, dar numai 1 bit este încă nefolosit. În principiu, o nouă opţiune ar putea fi creată pentru a reţine adevărata adresă a sursei, dar acest lucru ar necesita schimbarea codului din protocolul IP de pe toate staţiile din întregul Internet pentru a putea prelucra noua opţiune. Aceasta nu este o alternativă promiţătoare pentru o rezolvare rapidă.

În cele ce urmează se prezintă ceea ce s-a întâmplat cu adevărat. Proiectanţii NAT au observat că marea majoritate a pachetelor IP aveau conţinut TCP sau UDP. Ambele au antete ce conţin un port sursă şi un port destinaţie. În continuare vom discuta doar despre porturi TCP, dar exact aceleaşi lucruri se aplică şi la UDP. Porturile sunt numere întregi pe 16 biţi care indică unde începe şi unde se termină o conexiune TCP. Aceste câmpuri pun la dispoziţie câmpul necesar funcţionării NAT.

Atunci când un proces doreşte să stabilească o conexiune TCP cu un proces de la distanţă, se ataşează unui port TCP nefolosit de pe maşina sa. Acesta se numeşte portul sursă şi spune codului TCP unde să trimită pachetele care vin şi aparţin acestei conexiuni. Procesul furnizează şi un port destinaţie 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 clienţii să le poată localiza. Fiecare mesaj TCP care pleacă conţine atât un port sursă cât şi un port destinaţie. Cele două porturi permit identificarea proceselor care folosesc conexiunea la cele două capete.

Folosind câmpul Port sursă rezolvăm problema de corespondenţă. De fiecare dată când un pachet pleacă, el intră în unitatea NAT şi 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 translatare a unităţii NAT, care are 65.536 intrări. Această tabelă conţine adresa IP iniţială şi portul iniţial. În final, sunt recalculate şi inserate în pachet sumele de control ale antetelor IP şi TCP. Câmpul Port sursă trebuie înlocuit pentru că s-ar putea întâmpla, de exemplu, ca staţiile 10.0.0.1 şi 10.0.0.2 să aibă amândouă conexiuni care să folosească portul 5000, deci câmpul Port sursă nu este suficient pentru a identifica procesul emiţător.

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

Multe persoane din comunitatea IP o consideră un monstru pe faţa pământului. Rezumate succint, iată câteva dintre obiecţii :

- În primul rând,NAT violează modelul arhitectural al IP-ului, care afirmă că fiecare adresă IP identifică unic o singură maşină din lume. Întreaga structură software a Internetului este construită pornind de la acest fapt. Cu NAT, mii de maşini pot folosi (şi folosesc) adresa 10.0.0.1.

- În al doilea rând, NAT schimbă natura Internetului de la o reţea fără conexiuni la un fel de reţea cu conexiuni.
- În al treilea rând, NAT încalcă cea mai fundamentală regulă a protocoalelor pe niveluri: nivelul *k* nu poate face nici un fel de presupuneri referitor la ceea ce a pus nivelul *k+1* în câmpul de informaţie utilă. Principiul de bază este să se păstreze niveluri independente.
- În al patrulea rând, procesele din Internet nu sunt obligate să folosească TCP sau UDP.
- În al cincilea rând, anumite aplicaţii introduc adrese IP în corpul mesajului. Receptorul extrage aceste adrese şi le foloseşte. De vreme ce NAT nu ştie nimic despre aceste adrese, nu le poate înlocui, deci orice încercare de a le folosi de către cealaltă parte va eşua.

Acestea şi alte probleme ale NAT sunt discutate în RFC 2993. În general, opozanţii NAT spun că rezolvând problema insuficienţei adreselor IP cu o soluţie temporară şi urâtă, presiunea pentru a implementa soluţia reală, adică tranziţia la IPv6, este redusă şi acesta este un lucru rău.

18. Protocolul mesajelor de control din Internet

Operarea Internet-ului este strâns monitorizată de către rutere. Atunci când se întâmplă ceva neobişnuit, evenimentul este raportat prin **ICMP** (**Internet Control Message Protocol** – protocolul mesajelor de control din Internet), care este folosit şi 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
Problemă de parametru	Câmp invalid în antet
Oprire sursă	Pachet de șoc
Redirectare	Învață un ruter despre geografie
Cerere de ecou	Întreabă o mașină dacă este activă
Răspuns ecou	Da, sunt activă
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 DESTINAŢIE INACCESIBILĂ (DESTINATION UNREACHABLE) este folosit atunci când subreţeaua sau un ruter nu pot localiza destinaţia, sau un pachet cu bitul DF nu poate fi livrat deoarece o reţea cu „pachete mici” îi stă în cale.

Mesajul TIMP DEPĂŞIT (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 emiţătoare 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 aşteptat să încetinească ritmul de transmisie. Este folosit arareori, deoarece când apare congestie, aceste pachete au tendinţa de a turna mai mult gaz pe foc.

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

Mesajele CERERE ECOU (ECHO REQUEST) şi RĂSPUNS ECOU (ECHO REPLY) sunt folosite pentru a vedea dacă o anumită destinaţie este accesibilă şi activă. Este de aşteptat ca la recepţia mesajului ECOU, destinaţia să răspundă printr-un mesaj RĂSPUNS ECOU.

Mesajele CERERE AMPRENTĂ DE TIMP (TIMESTAMP REQUEST) şi RĂSPUNS AMPRENTĂ DE TIMP (TIMESTAMP REPLY) sunt similare, cu excepţia faptului că în răspuns sunt înregistrate timpul de sosire a mesajului şi de plecare a răspunsului. Această facilitate este folosită pentru a măsura performanţele reţelei.

19. Protocolul de rezoluţie a adresei: ARP

Deşi fiecare maşină din Internet are una sau mai multe adrese IP, acestea nu pot fi folosite de fapt pentru trimiterea pachetelor deoarece hardware-ul nivelului legăturii de date nu înţelege adresele Internet. Actualmente, cele mai multe gazde sunt ataşate la un LAN printr-o placă de interfaţă care înţelege numai adresele LAN. De exemplu, fiecare placă Ethernet fabricată până acum vine cu o adresă Ethernet de 48 biţi. Fabricanţii plăcilor Ethernet cer un spaţiu de adrese de la o autoritate centrală pentru a se asigura că nu există două plăci cu aceeaşi adresă (pentru a evita conflictele care ar apărea dacă cele două plăci ar fi în acelaşi LAN). Plăcile trimit şi primesc cadre pe baza adresei Ethernet de 48 biţi. Ele nu ştiu absolut nimic despre adresele IP pe 32 de biţi.

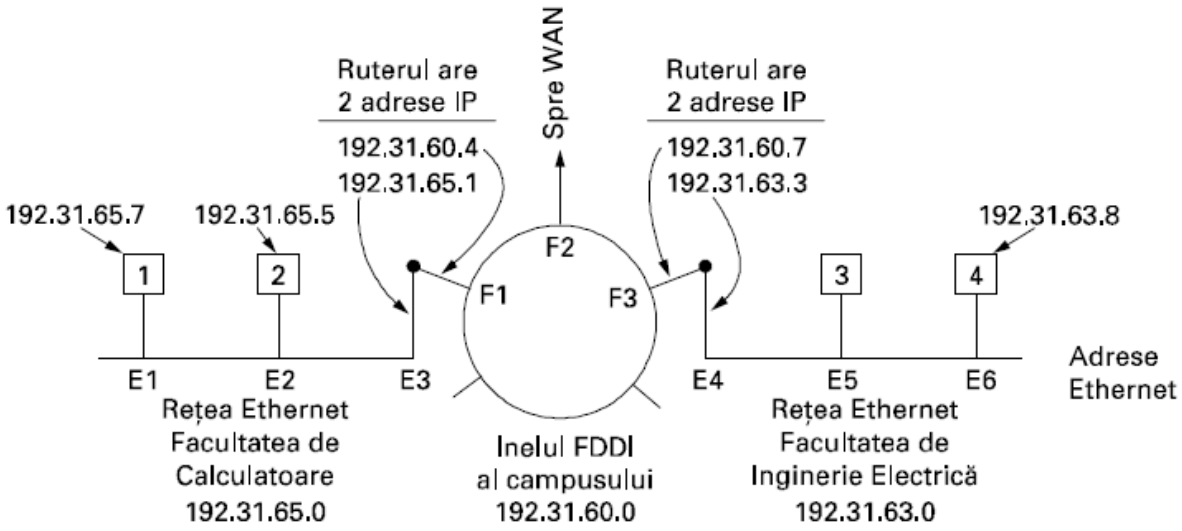


Fig. 5-62. Trei rețele /24 interconectate: două rețele Ethernet și un inel FDDI.

Se pune atunci întrebarea: Cum sunt transformate adresele IP în adrese la nivelul legăturii de date, ca de exemplu Ethernet? Să începem prin a vedea cum trimite un utilizator de pe gazda 1 un pachet unui utilizator de pe gazda 2. Presupunem că expeditorul ştie 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) care 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 adresă şi văd că destinatarul se află în propria reţea, dar au nevoie de un mijloc prin care să determine adresa Ethernet a destinatarului. O soluţie este să avem undeva în sistem un fişier de configurare care transformă adresele IP în adrese Ethernet. Această soluţie este posibilă, desigur, dar pentru organizaţii cu mii de maşini, menţinerea fişierelor actualizate este o acţiune consumatoare de timp şi care poate genera erori.

O soluţie mai bună este ca gazda 1 să trimită un pachet de difuzare în reţeaua Ethernet întrebând: „Cine este proprietarul adresei IP 192.31.65.5?”. Pachetul de difuzare va ajunge la toate maşinile din reţeaua Ethernet 192.31.65.0 şi fiecare îşi 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 şi a primi răspunsul se numeşte **ARP** (**Address Resolution Protocol** - Protocolul de rezoluţie a adresei). Aproape toate maşinile din Internet îl folosesc. Avantajul folosirii ARP faţă de fişierele de configurare îl reprezintă simplitatea. Administratorul de sistem nu trebuie să facă prea multe, decât să atribuie fiecărei maşini o adresă IP şi să hotărască măştile subreţelelor. 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 informaţie utilă şi îl lansează pe reţeaua Ethernet. Placa Ethernet a gazdei 2 detectează acest cadru, recunoaşte că este un cadru pentru ea, îl ia repede şi generează o întrerupere. Driverul Ethernet extrage pachetul IP din informaţia utilă şi îl trimite programelor IP, care văd că este corect adresat şi îl prelucrează.

Pentru a face ARP-ul mai eficient sunt posibile mai multe optimizări. Pentru început, la fiecare execuţie a ARP, maşina păstrează rezultatul pentru cazul în care are nevoie să contacteze din nou aceeaşi maşină în scurt timp. Data viitoare va găsi local corespondentul adresei, evitându-se astfel necesitatea unei a doua difuzări. În multe cazuri, gazda 2 trebuie să trimită înapoi un răspuns, ceea ce o

forţează să execute ARP, pentru a determina adresa Ethernet a expeditorului. Această difuzare

ARP poate fi evitată obligând gazda 1 să includă în pachetul ARP corespondenţa dintre adresa sa IP şi 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 maşinile din reţeaua Ethernet pot memora această relaţie în memoria ARP locală.

Altă optimizare este ca fiecare maşină să difuzeze corespondenţa sa de adrese la pornirea maşinii. 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 totuşi soseşte un răspuns (neaşteptat), înseamnă că două maşini au aceeaşi adresă IP. Noua maşină ar trebui să-l informeze pe administratorul de sistem şi să nu pornească.

20. DHCP

Ca şi RARP şi 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 acelaşi 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.

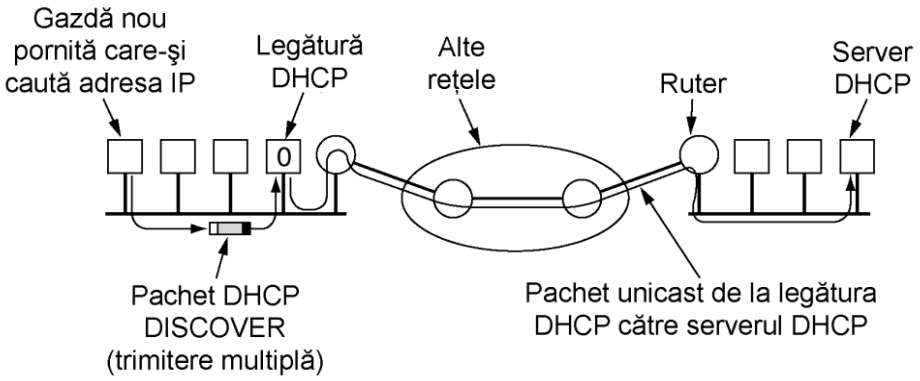


Fig. 5-63. Funcţionarea DHCP.

Pentru a-şi afla adresa IP, o maşină tocmai pornită difuzează un pachet DHCP DISCOVER. Agentul de legătură DHCP din LAN interceptează toate difuzările DHCP.

Atunci când găseşte un pachet DHCP DISCOVER, îl trimite ca pachet unicast serverului DHCP, posibil într-o reţea depărtată. Singura informaţie 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 părăseşte reţeaua şi 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 reuşeşte să facă cererea sau dacă cererea este respinsă, gazda nu va mai putea folosi adresa IP care îi fusese dată mai devreme.

21. Protocolul de dirijare folosit de porţile interioare: OSPF

Internet-ul este construit dintr-un număr mare de sisteme autonome(AS). Fiecare AS este administrat de o organizaţie diferită şi poate folosi propriul algoritm de dirijare în interior. De exemplu, reţelele interne ale companiilor *X*, *Y* şi *Z* ar fi văzute ca trei AS-uri dacă toate ar fi în Internet. Toate trei pot folosi intern algoritmi de dirijare diferiţi. Cu toate acestea, existenţa standardelor, chiar şi pentru dirijarea internă, simplifică implementarea la graniţele dintre AS-uri şi permite reutilizarea codului. **OSPF** (**Open Shortest Path First** - protocol public (deschis) bazat pe calea cea mai scurtă) a devenit un standard în 1990. Mulţi producători de rutere oferă support pentru el şi acesta a devenit principalul protocol de porţi interioare.

OSPF suportă trei tipuri de conexiuni şi reţele:

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

OSPF funcţionează prin abstractizarea colecţiei de reţele, rutere şi linii reale într-un graf orientat în care fiecărui arc îi este atribuit un cost (distanţă, 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 direcţie. Ponderile lor pot fi diferite. O reţea multiacces este reprezentată de un nod pentru reţeaua însăşi plus câte un nod pentru fiecare ruter. Arcele de la nodul reţea la rutere au pondere 0 şi sunt omise din graf.

Un sistem autonom :

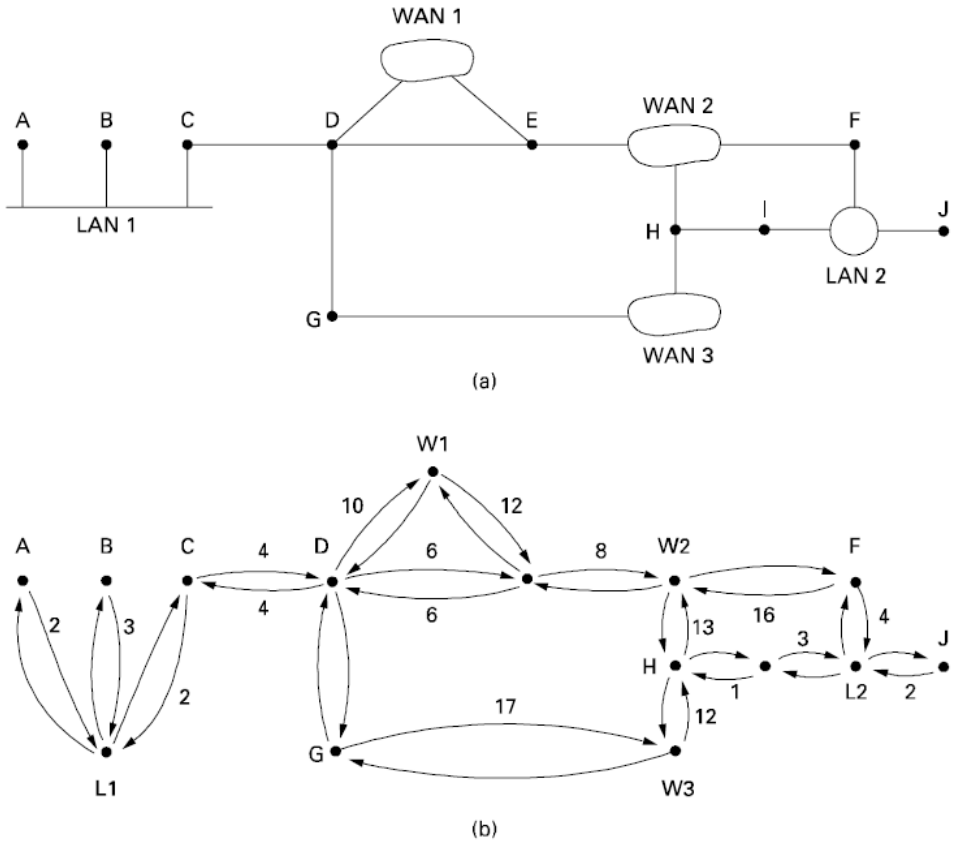


Fig. 5-64. (a) Un sistem autonom. (b) O reprezentare de tip graf a lui (a).

Multe din AS-urile din Internet sunt foarte mari şi nu sunt simplu de administrat. OSPF le permite să fie divizate în **zone** numerotate, unde o zonă este o reţea sau o mulţime de reţele învecinate. Zonele nu se suprapun şi nu este necesar să fie exhaustive, în sensul că unele rutere pot să nu aparţină nici unei zone. O zonă este o generalizare a unei subreţele. În afara zonei, topologia şi 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 tunele, 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 şi are un cost. Fiecare ruter care este conectat la două sau mai multe zone aparţine coloanei vertebrale. Analog cu celelalte zone, topologia coloanei vertebrale nu este vizibilă din afara coloanei vertebrale.

În interiorul unei zone, fiecare ruter are aceeaşi bază de date pentru starea legăturilor şi foloseşte acelaşi 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 puţin unul. Un ruter care conectează două zone are nevoie de bazele de date pentru ambele zone şi 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 şi interASuri. Rutele intrazonale sunt cele mai uşoare, din moment ce ruterul sursă ştie întotdeauna calea cea mai scurtă spre ruterul destinaţie. Dirijarea interzonală se desfăşoară întotdeauna în trei paşi: drum de la sursă la coloana vertebrală; drum de-a lungul coloanei vertebrale până la zona destinaţie; drum la destinaţie. Acest algoritm forţează o configuraţie de tip stea pentru OSPF, coloana vertebrală fiind concentratorul (hub), iar celelalte zone fiind spiţele. Pachetele sunt dirijate de la sursă la destinaţie „ca atare”. Ele nu sunt încapsulate sau trecute prin tunel, cu excepţia cazului în care merg spre o zonă a cărei unică conexiune la coloana vertebrală este un tunel.

Tip mesaj	Descriere
Hello	Folosit pentru descoperirea vecinilor
Actualizare Stare Legătură	Emițătorul 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.

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

OSPF funcţionează prin schimb de informaţii între rutere adiacente, care nu este acelaşi lucru cu schimbul de informaţii 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 situaţie, un ruter este ales ca ruter desemnat. Se spune că el este adiacent cu toate celelalte rutere din LAN-ul său şi schimbă informaţii cu ele. Ruterele vecine care nu sunt adiacente nu schimbă informaţii între ele. De asemenea, este actualizat în permanenţă şi un ruter desemnat de rezervă pentru a uşura tranziţia dacă ruterul desemnat primar se defectează şi trebuie să fie înlocuit imediat.

În timpul funcţionării normale, fiecare ruter inundă periodic cu mesaje ACTUALIZARE STARE LEGĂTURĂ (Link State Update) fiecare ruter adiacent. Acest mesaj indică starea sa şi furnizează costurile folosite în baza de date topologică. Mesajele de inundare sunt confirmate pentru a le face sigure. Fiecare mesaj are un număr de secvenţă, 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 îşi revine sau când costul acesteia se modifică.

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

Fiecare partener poate cere informaţii 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 şi astfel, noua informaţie este răspândită în zonă. Toate aceste mesaje sunt trimise ca simple pachete IP.

22. Protocolul de dirijare pentru porţi externe: BGP

În cadrul unui singur AS, protocolul de dirijare recomandat este OSPF (deşi, desigur, nu este singurul folosit). Între AS-uri se foloseşte un protocol diferit, **BGP** (**Border Gateway Protocol** – Protocolul porţilor de graniţă). Între AS-uri este necesar un protocol diferit pentru că scopurile unui protocol pentru porţi interioare şi ale unui protocol pentru porţi exterioare sunt diferite. Tot ce trebuie să facă un protocol pentru porţi interioare este să mute pachetele cât mai eficient de la sursă la destinaţie. El nu trebuie să-şi facă probleme cu politica.

Ruterele ce folosesc protocolul de porţi exterioare trebuie să ţină cont într-o mare măsură de politică. De exemplu, un AS al unei corporaţii poate dori facilitatea de a trimite pachete oricărui site Internet şi să recepţioneze pachete de la orice site Internet. Cu toate acestea, poate să nu dorească să asigure tranzitarea pentru pachetele originare dintr-un AS străin destinate unui AS strain 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.

Protocoalele pentru porţi externe, în general şi BGP în particular, au fost proiectate pentru a permite forţarea multor tipuri de politici de dirijare pentru traficul între AS-uri. Politicile tipice implică consideraţii 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 porneşte 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 destinaţie.

5. Traficul care pleacă sau ajunge la IBM nu trebuie să tranziteze Microsoft.

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

23. IPv6

IPv6 menţine caracteristicile bune ale IP-ului, le elimină sau atenuează pe cele rele şi adaugă unele noi acolo unde este nevoie. În general, IPv6 nu este compatibil cu IPv4, dar el este compatibil cu celelalte protocoale Internet auxiliare, incluzând TCP, UDP, ICMP, IGMP, OSPF, BGP şi DNS, câteodată fiind necesare mici modificări (majoritatea pentru a putea lucra cu adrese mai lungi).

În primul rând şi cel mai important, IPv6 are adrese mai lungi decât IPv4. Ele au o lungime de 16 octeţi, ceea ce rezolvă problema pentru a cărei soluţionare a fost creat IPv6: să furnizeze o sursă efectiv nelimitată de adrese Internet.

A doua mare îmbunătăţire a lui IPv6 este simplificarea antetului. El conţine numai 7 câmpuri (faţă de 13 în IPv4). Această schimbare permite ruterelor să prelucreze pachetele mai rapid, îmbunătăţind astfel productivitatea şi întârzierea.

A treia mare îmbunătăţire a fost suportul mai bun pentru opţiuni. Această schimbare a fost esenţială în noul antet, deoarece câmpurile care erau necesare anterior sunt acum opţionale. În plus, modul în care sunt reprezentate opţiunile este diferit, uşurând ruterelor saltul peste opţiunile 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 porţia sa de poveşti de ziar despre copii precoce de 12 ani care îşi 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 îmbunătăţi securitatea. Autentificarea şi confidenţialitatea sunt trăsături cheie ale noului IP. Ulterior ele au fost adaptate şi în IPv4, astfel că în domeniul securităţii diferenţele nu mai sunt aşa de mari.

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

Antetul IPv6

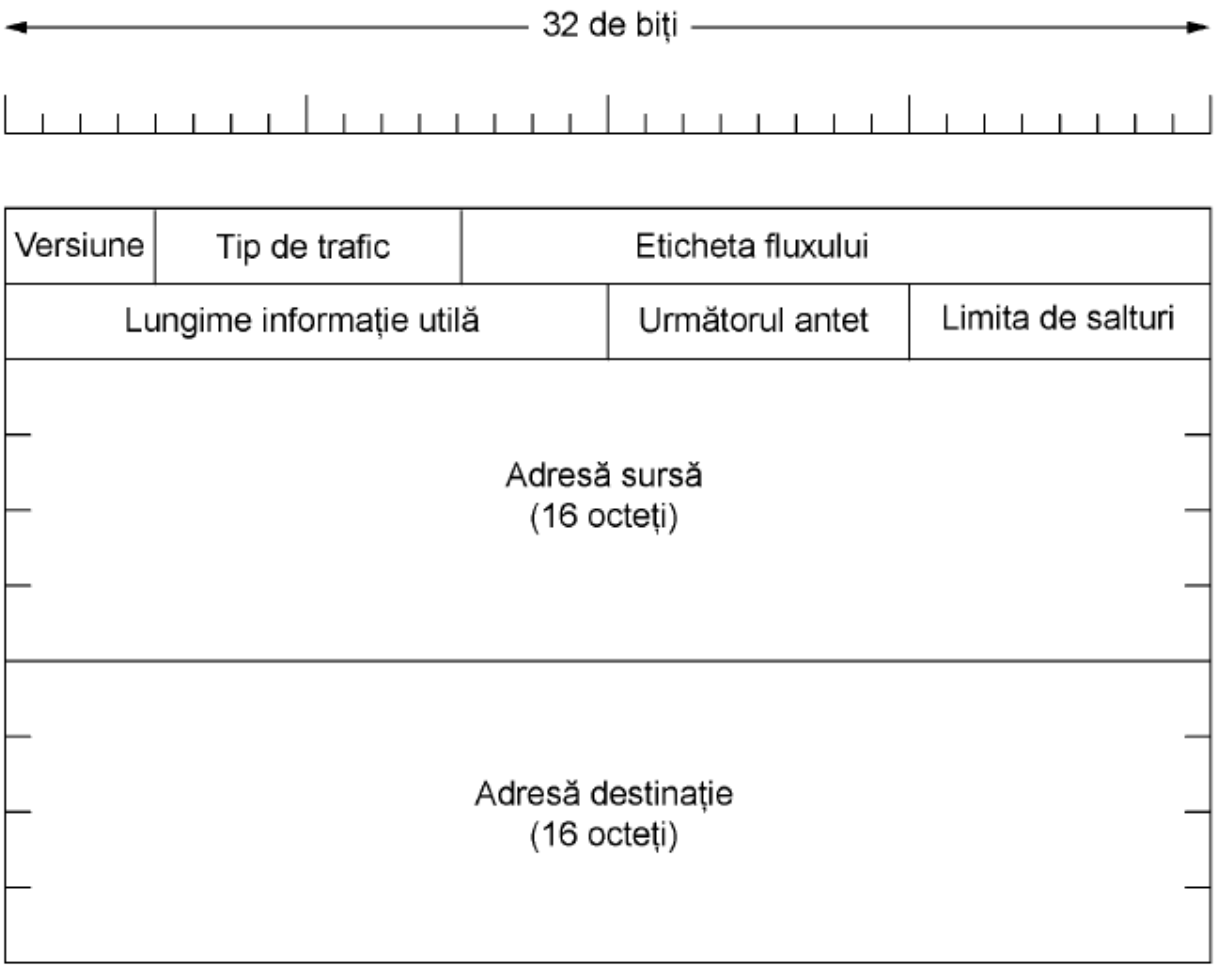


Fig. 5-68. Antetul fix IPv6 (obligatoriu).

Câmpul Versiune este întotdeauna 6 pentru IPv6 (şi 4 pentru IPv4).

Câmpul Tip de trafic (Traffic class) este folosit pentru a distinge între pachetele care au diverse cerinţe 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 desfăşoară 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 şi unei destinaţii să stabilească o pseudo-conexiune cu proprietăţi şi cerinţe particulare. De exemplu, un şir de pachete de la un proces de pe o anumită gazdă sursă către un anumit proces pe o anumită gazdă destinaţie poate avea cerinţe de întârziere stricte şi din acest motiv necesită capacitate de transmisie rezervată. Fluxul poate fi stabilit în avans şi 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 subreţele cu datagrame şi garanţiile unei subreţele cu circuite virtuale. Fiecare flux este desemnat de adresa sursă, adresa destinaţie şi numărul de flux, aşa încât, între o pereche dată de adrese IP pot exista mai multe fluxuri active în acelaşi timp . De asemenea, în acest mod, chiar dacă două fluxuri venind de la gazde diferite, dar cu acelaşi număr de flux trec prin acelaşi ruter, ruterul va fi capabil să le separe folosind adresele sursă şi destinaţie. Se aşteaptă ca numerele de flux să fie alese aleator, în loc de a fi atribuite secvenţial începând cu 1, pentru că se aşteaptă ca ruterele să le folosească în tabele de dispersie.

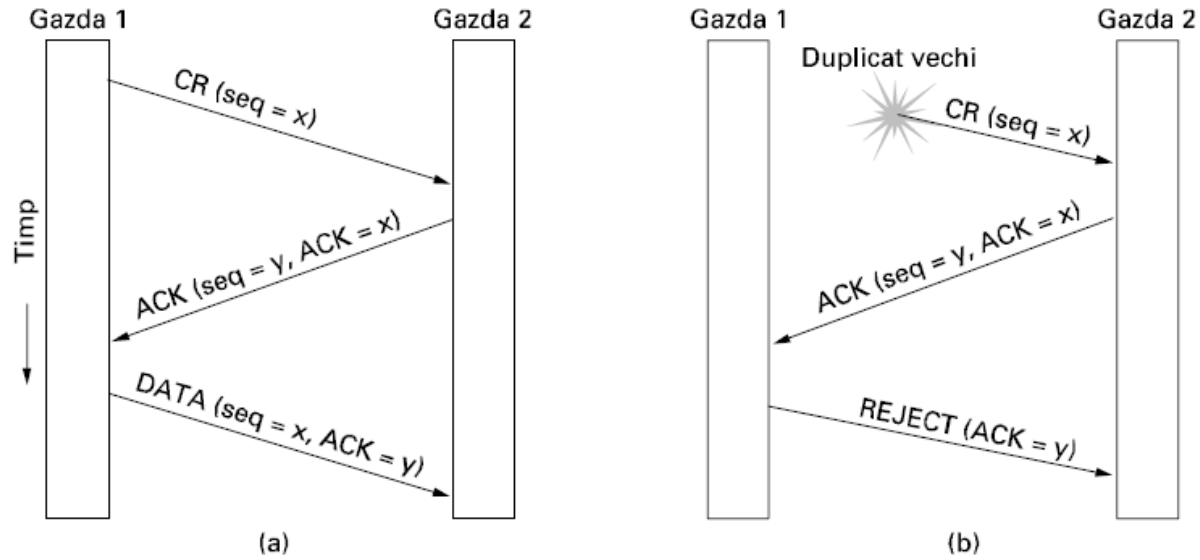
Câmpul Lungimea informaţiei utile spune câţi octeţi urmează după antetul de 40 de octeţi din fig. 5-68. Numele a fost schimbat faţă de câmpul Lungime totală din IPv4 deoarece semnificaţia este uşor modificată: cei 40 de octeţi nu mai sunt parte a lungimii (aşa cum erau înainte).

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

Câmpul Limita salturilor este folosit pentru a împiedica pachetele să trăiască veşnic. El este, în practică, identic cu câmpul Timp de viaţă din IPv4, şi anume un câmp care este decrementat la fiecare salt dintr-o reţea 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 şi Adresă destinaţie.

24. Stabilirea si eliberarea conexiunii la nivelul transport



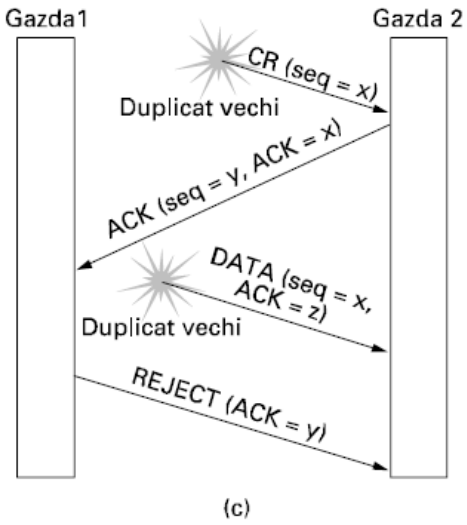


Fig. 6-11. Trei scenarii posibile de stabilire a conexiunii pentru un protocol cu înțelegere în trei pași. CR reprezintă CONNECTION REQUEST. (a) Cazul normal, (b) Un duplicat vechi al unui mesaj CONNECTION REQUEST apare când nu trebuie, (c) Sunt duplicate atât CONNECTION REQUEST cât și CONNECTION ACCEPTED.

Deoarece TPDU-urile de control pot şi ele să fie întârziate, pot apărea probleme atunci când entităţile de transport încercă să cadă de acord asupra numărului iniţial de secvenţă. Să presupunem, de exemplu, că, pentru a stabili o conexiune, gazda 1 trimite un mesaj CONNECTION REQUEST conţinând numărul de secvenţă iniţial propus şi portul destinaţie gazdei 2. Acesta va confirma mesajul trimiţând 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 înţelegere în trei paşi (three-way handshake). Acest protocol nu necesită ca ambele părţi să înceapă să trimită acelaşi număr de secvenţă, deci poate fi utilizat şi împreună cu alte metode de sincronizare decât ceasul global. Procedura normală de iniţiere a conexiunii este exemplificată în fig. 6-11(a). Gazda 1 alege un număr de secvenţă x şi trimite un TPDU CONNECTION REQUEST care conţine x gazdei 2. Gazda 2 răspunde cu CONNECTION ACK, confirmând x şi anunţând numărul său iniţial de secvenţă, y. În cele din urmă gazda 1 confirmă alegerea lui y gazdei 2 în primul mesaj de date pe care îl trimite.

Eliberarea unei conexiuni este mai uşoară decât stabilirea ei. Totuşi, există mai multe dificultăţi decât ne-am aştepta. Aşa cum am mai amintit, există două moduri de a termina o conexiune: eliberare simetrică şi eliberare asimetrică. Sistemul telefonic foloseşte eliberarea asimetrică: atunci când unul din interlocutori închide, conexiunea este întreruptă. Eliberarea simetrică priveşte conexiunea ca pe două conexiuni separate unidirecţionale şi cere ca fiecare să fie eliberată separat.

Eliberarea asimetrică este bruscă şi 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 destinaţie, gazda 2 trimite DISCONNECT REQUEST . În acest caz, conexiunea va fi eliberată şi vor fi pierdute date.

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

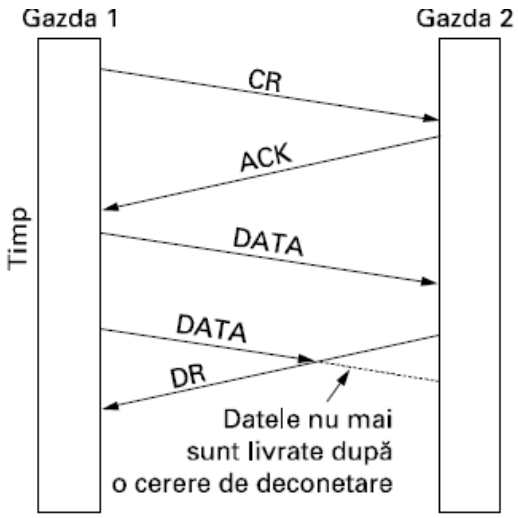


Fig. 6-12. Deconectare bruscă cu pierdere de date. CR= CONNECTION REQUEST, ACK=CONNECTION ACCEPTED , DR=DISCONNECT REQUEST.

Eliberarea simetrică este utilă atunci când fiecare proces are o cantitate fixă de date de trimis şi ştie bine când trebuie să transmită şi când a terminat. În alte

situaţii însă, nu este deloc uşor de determinat când trebuie eliberată conexiunea şi când a fost trimis tot ce era de transmis. S-ar putea avea în vedere un protocol de tipul următor: atunci când 1 termină, trimite ceva de tipul: Am terminat. Ai terminat şi tu? Dacă gazda 2 răspunde: Da, am terminat. Închidem! conexiunea poate fi eliberată în condiţii bune.

Din nefericire, acest protocol nu merge întotdeauna. Binecunoscuta problemă a celor două armate este similară acestei situaţii: să ne imaginăm că armată albă şi-a pus tabăra într-o vale. Pe amândouă crestele 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-şi sincronizeze atacul. Totuşi singura lor posibilitate de comunicaţie este să trimită un mesager care să străbată valea. Mesagerul poate fi capturat de armata albă şi mesajul poate fi pierdut (adică vor trebui să utilizeze un canal de comunicaţie 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 ştie dacă răspunsul său a ajuns sau nu la destinaţie. Dacă nu a ajuns, armata 1 nu va ataca, deci ar fi o prostie din partea lui să intre în luptă. Să încercăm să îmbunătăţim protocolul, transformându-l într-unul cu înţelegere în trei paşi. Iniţiatorul 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 ştie dacă confirmarea sa a ajuns la destinaţie şi 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 lovi de aceleaşi probleme. De fapt, poate fi demonstrat că nu există un protocol care să funcţioneze.

Pentru a vedea legătura problemei celor două armate cu problema eliberării conexiunii este suficient să înlocuim ‘atac’ cu ‘deconectare’. Dacă niciuna din părţi 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 dispuşi să ne asumăm mai multe riscuri atunci când este vorba de eliberarea conexiunii decât atunci când este vorba de atacarea armatei albe, aşa încât situaţia nu este întru totul fără speranţă. Fig. 6-14 prezintă patru scenarii de eliberare a conexiunii folosind un protocol cu confirmare în trei timpi. Deşi acest protocol nu este infailibil, el este în general adecvat.

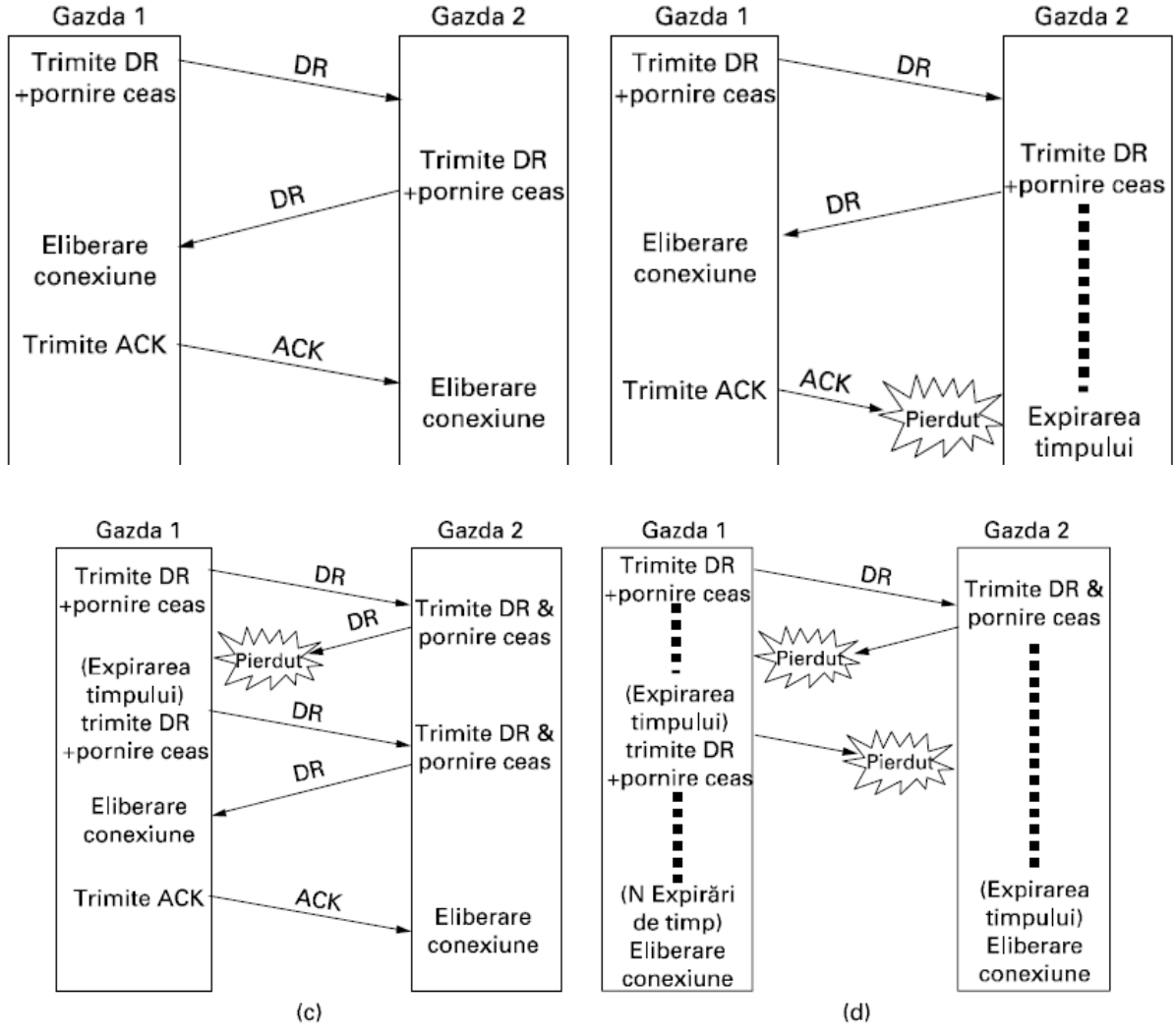


Fig. 6-14. Patru cazuri posibile la eliberarea conexiunii: (a)Cazul normal cu confirmare în trei timpi. (b)Ultima confirmare este pierdută. (c) Răspunsul este pierdut. (d) Răspunsul şi următoarele cereri de deconectare sunt pierdute. (DR=DISCONNECT REQUEST).

25. Protocolul UDP

Setul de protocoale Internet suportă un protocol de transport fără conexiune, **UDP** (**User Datagram Protocol** – Protocol cu Datagrame Utilizator). UDP oferă aplicaţiilor o modalitate de a trimite datagrame IP încapsulate şi 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 octeţi urmat de informaţia utilă . Antetul este prezentat în fig. 6-23. Cele două porturi servesc la identificarea punctelor terminale ale maşinilor sursă şi destinaţie. Când ajunge un pachet UDP, conţinutul său este predat procesului ataşat portului destinaţie. Această ataşare are loc atunci când se foloseşte o simplă procedură de nume sau ceva asemănător, aşa cum am văzut în fig. 6-6 pentru TCP (procesul de legătură este acelaşi pentru UDP). De fapt, valoarea cea mai importantă dată de existenţa UDP-ului faţă de folosirea doar a IPului simplu, este aceea a adăugării porturilor sursă şi destinaţie. Fără câmpurile portului, nivelul de transport nu ar şti ce să facă cu pachetul. Cu ajutorul lor, segmentele se livrează corect.

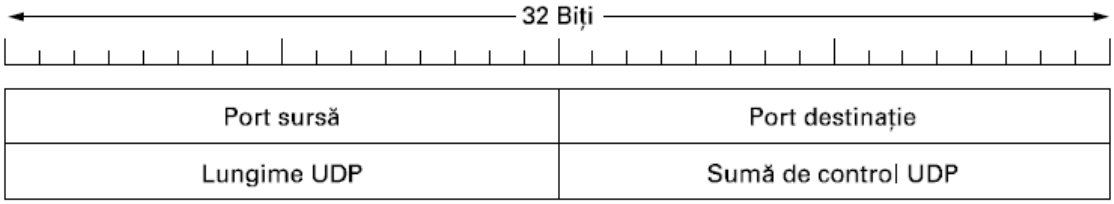


Fig. 6-23. Antetul UDP.

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

Câmpul lungime UDP include antetul de 8 octeţi şi datele. Câmpul sumă de control UDP este optional şi stocat ca 0 (zero) dacă nu este calculat (o valoare de adevăr 0 rezultată în urma calculelor **este memorată ca un şir de biţi 1).** Dezactivarea acestuia este o prostie, excepţie făcând cazul în care calitatea informaţiei chiar nu contează (de exemplu, transmisia vocală digitalizată).

Merită probabil menţionate, în mod explicit, unele dintre lucrurile pe care UDP-ul nu le face. Nu realizează controlul fluxului, controlul erorii, sau retransmiterea unui segment incorect primit. Toate acestea depind de procesele utilizatorului. Ceea ce face este să ofere protocolului IP o interfaţă cu facilităţi adăugate de demultiplexare a mai multor procese, folosind porturi. Aceasta este tot ceea ce face UDP-ul. Pentru aplicaţiile 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 situaţiilor client-server. Deseori, clientul trimite o cerinţă scurtă server-ului şi aşteaptă înapoi un răspuns scurt. Dacă se pierde ori cererea ori răspunsul, clientul poate pur şi simplu să încerce din nou după ce a expirat timpul. Nu numai că va fi mai simplu codul, dar sunt necesare şi mai puţine mesaje (câte unul în fiecare direcţie) decât la un protocol care solicită o iniţializare iniţială.

O aplicaţie care foloseşte UDP-ul în acest fel este DNS (Domain Name System, rom: Sistem de rezolvare de nume). 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, conţinând numele gazdă, către un server DNS. Serverul răspunde cu un pachet UDP conţinând adresa de IP a gazdei. Nu este necesară nici o iniţializare în avans şi nici o închidere de sesiune. Doar două mesaje traversează reţeaua.

26. 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 aplicaţiilor multimedia în timp real. În particular, având în

vedere că radioul pe internet, telefonia pe Internet, muzica la cerere, video-conferinţele, video la cerere şi alte aplicaţii multimedia au devenit mai răspândite, oamenii au descoperit că fiecare aplicaţie a folosit, mai mult sau mai puţin, acelaşi protocol de transport în timp real. Treptat a devenit clar faptul că un protocol generic de transport în timp real, pentru aplicaţii multimedia, ar fi o idee bună. Aşa a luat naştere **RTP**-ul (**Real-time Transport Protocol**, rom: Protocol de transport în timp real).

Poziţia RTP-ului în stiva de protocoale este oarecum ciudată. S-a hotărât să se pună RTP-ul în spaţiul utilizator şi să se ruleze (în mod normal) peste UDP. El funcţionează după cum urmează. Aplicaţiile multimedia constau în aplicaţii audio, video, text şi posibil alte fluxuri. Acestea sunt trimise bibliotecii RTP, care se află în spaţiul utilizator împreună cu aplicaţia. Apoi, această bibliotecă multiplexează fluxurile şi 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 şi încapsulate în pachete IP. Dacă computer-ul se găseşte într-o reţea Ethernet, pachetele IP sunt puse apoi în cadre Ethernet, pentru transmisie.

Ca o consecinţă a acestei proiectări, este cam dificil de spus în ce nivel este RTP-ul. Cum rulează în spaţiul utilizator şi este legat la programul aplicaţie, în mod cert arată ca un protocol de aplicaţie. Pe de altă parte, este un protocol generic independent de aplicaţie, care doar furnizează facilităţi 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 aplicaţie.

Funcţia 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 destinaţie (unicasting) sau către destinaţii multiple (multicasting). Deoarece RTP-ul foloseşte numai UDP normal, pachetele sale nu sunt tratate în mod special de către rutere, decât dacă sunt activate anumite facilităţi de calitatea serviciilor din IP. În particular, nu există garanţii speciale referitoare la livrare, bruiaj etc. Fiecărui pachet trimis în fluxul RTP i se dă un număr cu unu mai mare decât al predecesorului său. Această numerotare permite destinaţiei să

stabilească dacă lipsesc unele pachete. Dacă un pachet lipseşte, cea mai bună decizie ce poate fi luată de către destinaţie este de a aproxima valoarea lipsă prin interpolare. Retransmiterea nu este o opţiune practică având în vedere că pachetul retransmis va ajunge probabil prea târziu pentru a fi util. Ca o consecinţă, RTP-ul nu are control al fluxului, control al erorii, nu are confirmări şi nu are mecanism pentru a cere retransmiterea.

Antetul RTP este ilustrat în fig. 6-26. Acesta constă din trei cuvinte de 32 biţi şi eventual unele extensii. Primul cuvânt conţine câmpul Versiune, care este deja la 2.

Bitul P indică faptul că pachetul a fost extins la un multiplu de 4 octeţi. Ultimul octet extins ne spune câţi octeţi au fost adăugaţi. Bitul X indică prezenţa unui antet extins. Formatul şi semnificaţia 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 cerinţe neprevăzute.

Câmpul CC arată câte surse contribuabile sunt prezente, de la 0 la 15. Bitul M este un bit de marcare specific aplicaţiei. Poate fi folosit pentru a marca începutul unui cadru video, începutul unui cuvânt într-un canal audio sau altceva ce aplicaţia înţelege. Câmpul Tip informaţie utilă indică ce algoritm de codare a fost folosit (de exemplu 8 biţi audio necompresaţi, MP3, etc). Din moment ce

fiecare pachet transportă acest câmp, codarea se poate schimba în timpul transmisiei.

Numărul de secvenţă 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 şti când este făcut primul eşantion din pachet. Această valoare poate să ajute la reducerea bruiajului la receptor prin separarea redării de momentul ajungerii pachetului. Identificatorul sursei de sincronizare spune cărui flux îi aparţine pachetul. Este metoda utilizată pentru a multiplexa şi demultiplexa mai multe fluxuri de date într-un singur flux de pachete UDP. În sfârşit, dacă există, identificatorii sursei care contribuie sunt folosiţi când în studio există mixere. În acest caz, mixerul este sursa de sincronizare şi fluxurile, fiind amestecate, apar aici.

27. TCP - Transport Communication Protocol

TCP (Transport Communication Protocol - protocol de comunicaţie de nivel transport) a fost proiectat explicit pentru a asigura un flux sigur de octeţi de la un capăt la celălalt al conexiunii într-o inter-reţea nesigură. O inter-reţea diferă de o reţea propriu-zisă prin faptul că diferite părţi ale sale pot diferi substanţial în topologie, lărgime de bandă, întârzieri, dimensiunea pachetelor şi alţi parametri. TCP a fost proiectat să se adapteze în mod dinamic la proprietăţile inter-reţelei şi să fie robust în ceea ce priveşte mai multe tipuri de defecte.

O conexiune TCP este un flux de octeţi şi nu un flux de mesaje. Dimensiunile mesajelor nu se conservă de la un capăt la celălalt. Atunci când o aplicaţie trimite date către TCP, TCP-ul le poate expedia imediat sau le poate reţine într-un tampon (în scopul colectării unei cantităţi mai mari de informaţie pe care să o expedieze toată odată), după bunul său plac. Cu toate acestea, câteodată, aplicaţia doreşte ca informaţia să fie expediată imediat. De exemplu, să presupunem că un utilizator este conectat la o maşină de la distanţă. După ce a fost terminată o linie de comandă şi s-a tastat Return, este esenţial ca linia să fie imediat expediată către maşina de la distanţă şi să nu fie memorată până la terminarea următoarei linii. Pentru a forţa expedierea, aplicaţia poate folosi indicatorul PUSH, care îi semnalează TCP-ului să nu întârzie procesul de transmisie.

O ultimă caracteristică a serviciului TCP care merită menţionată aici constă în informaţia urgentă. Atunci când un utilizator apasă tasta DEL sau CTRL-C pentru a întrerupe o prelucrare la distanţă, aflată deja în execuţie, aplicaţia emiţător plasează o informaţie de control în fluxul de date şi o furnizează TCP-ului împreună cu indicatorul URGENT. Acest eveniment impune TCP-ului întreruperea acumulării de informaţie şi transmisia imediată a întregii informaţii disponibile deja pentru conexiunea respectivă. Atunci când informaţia urgentă este recepţionată la destinaţie, aplicaţia receptoare este întreruptă astfel încât, eliberată de orice altă activitate, aplicaţia să poată citi fluxul de date şi să poată regăsi informaţia urgentă. Sfârşitul informaţiei urgente este marcat, astfel încât aplicaţia să ştie când se termină informaţia. Începutul informaţiei urgente nu este marcat. Este sarcina aplicaţiei să determine acest început.

Protocolul de bază utilizat de către entităţile TCP este protocolul cu fereastră glisantă. Atunci când un emiţător transmite un segment, el porneşte un cronometru. Atunci când un segment ajunge la destinaţie, entitatea TCP receptoare trimite înapoi un segment (cu informaţie utilă, dacă aceasta există sau fără, în caz contrar) care conţine totodată şi numărul de secvenţă următor pe care aceasta se aşteaptă să-l recepţioneze. Dacă cronometrul emiţătorului depăşeşte o anumită valoare înaintea primirii confirmării, emiţătorul retransmite segmentul neconfirmat.

În fig. 6-29 este prezentată structura unui segment TCP. Fiecare segment începe cu un antet format dintr-o structură fixă de 20 de octeţi. Antetul fix poate fi urmat de un set de opţiuni associate antetului. În continuarea opţiunilor, dacă ele există, pot urma până la 65.535 - 20 - 20 = 65.495 de octeţi de date, unde primul 20 reprezintă antetul IP, iar al doilea antetul TCP. Segmente care nu conţin octeţi de date sunt nu numai permise, dar şi utilizate în mod frecvent pentru confirmări şi mesaje de control.

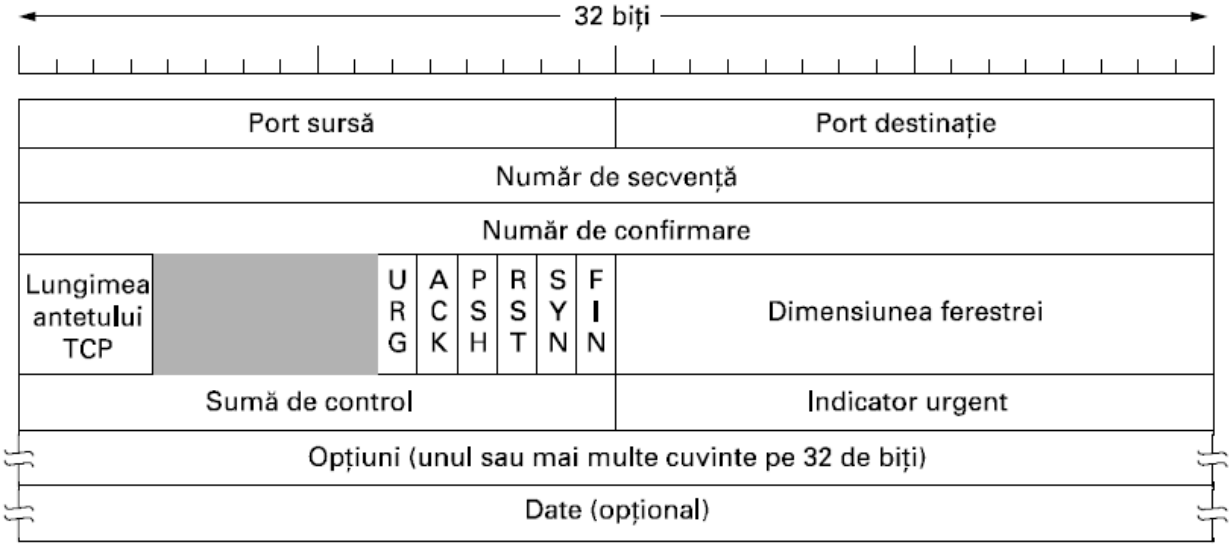


Fig. 6-29. Antetul TCP.

Câmpurile Port sursă şi Port destinaţie identifică punctele finale ale conexiunii. Porturile general cunoscute sunt definite la www.iana.org, dar fiecare gazdă le poate aloca pe celelalte după cum doreşte. Un port formează împreună cu adresa IP a maşinii sale un unic punct de capăt (eng.: end point) de 48 de biţi. Conexiunea este identificată de punctele de capăt ale sursei şi destinaţiei.

Câmpurile Număr de secvenţă şi Număr de confirmare au semnificaţia funcţiilor lor uzuale. Trebuie observat că cel din urmă indică octetul următor aşteptat şi nu ultimul octet recepţionat în mod corect. Ambele câmpuri au lungimea de 32 de biţi, deoarece într-un flux TCP fiecare bit de informaţie este numerotat.

Lungimea antetului TCP indică numărul de cuvinte de 32 de biţi care sunt conţinute în antetul TCP. Această informaţie este utilă, deoarece câmpul Opţiuni

este de lungime variabilă, proprietate pe care o transmite astfel şi antetului. Tehnic vorbind, acest câmp indică în realitate începutul datelor din segment, măsurat în cuvinte de 32 de biţi, dar cum acest număr este identic cu lungimea antetului în cuvinte, efectul este acelaşi.

Urmează un câmp de şase biţi care este neutilizat. Faptul că acest câmp a supravieţuit intact mai mult de un sfert de secol este o mărturie despre cât de bine a fost proiectat TCP-ul. Protocoale mai prost concepute ar fi avut nevoie de el pentru a corecta erori ale proiectării iniţiale.

Urmează acum şase indicatori de câte un bit. URG este poziţionat pe 1 dacă Indicatorul Urgent este valid. Indicatorul Urgent este folosit pentru a indica deplasamentul în octeţi faţă de numărul curent de secvenţă la care se găseşte informaţia urgentă. O astfel de facilitate ţine locul mesajelor de întrerupere. Aşa cum am menţionat deja anterior, această facilitate reprezintă esenţa modului în care emiţătorul poate transmite un semnal receptorului fără ca TCP-ul în sine să fie cauza întreruperii.

Bitul ACK este poziţionat pe 1 pentru a indica faptul că Numărul de confirmare este valid. În cazul în care ACK este poziţionat pe 0, segmentul în discuţie nu conţine o confirmare şi câmpul Număr de confirmare este ignorat.

Bitul PSH indică informaţia FORŢATĂ. Receptorul este rugat respectuos să livreze aplicaţiei informaţia respectivă imediat ce este recepţionată şi să nu o memoreze în aşteptarea umplerii tampoanelor de comunicaţie (lucru care, altminteri, ar fi făcut din raţiuni de eficienţă).

Bitul RST este folosit pentru a desfiinţa o conexiune care a devenit inutilizabilă datorită defecţiunii unei maşini 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, recepţionarea unui segment având acest bit poziţionat indică o problemă care trebuie tratată în funcţie de context.

Bitul SYN este utilizat pentru stabilirea unei conexiuni. Cererea de conexiune conţine SYN = 1 şi ACK = 0 pentru a indica faptul că acel câmp suplimentar de confirmare nu este utilizat. Răspunsul la o astfel de cerere conţine

o confirmare, având deci SYN = 1 şi ACK = 1. În esenţă, bitul SYN este utilizat pentru a indica o CERERE DE CONEXIUNE şi o CONEXIUNE ACCEPTATĂ, bitul ACK făcând distincţia între cele două posibilităţi.

Bitul FIN este folosit pentru a încheia o conexiune. El indică faptul că emiţătorul nu mai are nici o informaţie de transmis. Cu toate acestea, după închiderea conexiunii, un proces poate recepţiona în continuare date pe o durată nedefinită. Ambele segmente, SYN şi FIN, conţin numere de secvenţă şi 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 octeţi care pot fi trimişi, începând de la octetul confirmat. Un câmp Fereastră de valoare 0 este perfect legal şi spune că octeţii până la Număr de confirmare - 1 inclusiv au fost recepţionaţi, dar receptorul doreşte cu ardoare o pauză, aşa că mulţumeşte frumos, dar pentru moment nu doreşte continuarea transferului. Permisiunea de expediere poate fi acordată ulterior de către receptor prin trimiterea unui segment având acelaşi Număr de confirmare, dar un câmp Fereastră cu o valoare nenulă.

Este de asemenea prevăzută o Sumă de control, în scopul obţinerii unei fiabilităţi extreme. Această sumă de control este calculată pentru antet, informaţie şi pseudo-antetul conceptual prezentat în fig. 6-30. În momentul calculului, Suma de control TCP este poziţionată 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 biţi în complement faţă de 1 şi aplicând apoi încă o dată complementul faţă de 1 asupra sumei. În acest mod, atunci când receptorul aplică acelaşi calcul asupra întregului segment, inclusiv asupra Sumei de control, rezultatul ar trebui să fie 0.

Pseudo-antetul conţine adresele IP ale maşinii sursă şi destinaţie, de 32 de biţi fiecare, numărul de protocol pentru TCP (6) şi numărul de octeţi al segmentului TCP (incluzând şi 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 însăşi ierarhia protocolului, deoarece adresa IP aparţine nivelului IP şi nu nivelului TCP.

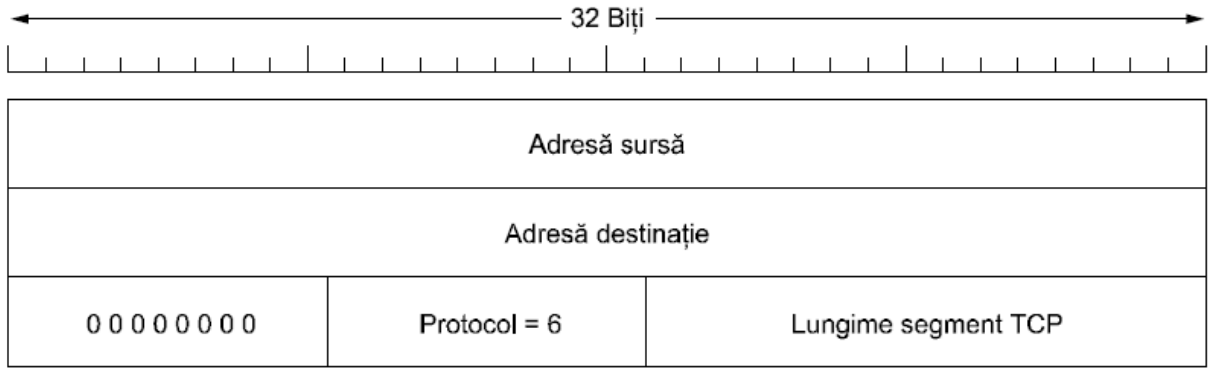


Fig. 6-30. Pseudo-antetul inclus în suma de control TCP.

Câmpul Opţiuni a fost proiectat pentru a permite adăugarea unor facilităţi suplimentare neacoperite de antetul obişnuit. Cea mai importantă opţiune este aceea care permite fiecărei maşini să specifice încărcarea maximă de informaţie utilă TCP pe care este dispusă să o accepte. Utilizarea segmentelor de dimensiune mare este mai eficientă decât utilizarea segmentelor de dimensiune mica datorită amortizării antetului de 20 de octeţi prin cantitatea mai mare de informaţie utilă. Cu toate acestea, este posibil ca maşini mai puţin performante să nu fie capabile să manevreze segmente foarte mari. În timpul iniţializării conexiunii, fiecare parte anunţă dimensiunea maximă acceptată şi aşteaptă de la partener aceeaşi informaţie. Câştigă cel mai mic dintre cele două numere. Dacă o maşină nu foloseşte această opţiune, cantitatea implicită de informaţie utilă este de 536 octeţi. Toate maşinile din Internet trebuie să accepte segmente de dimensiune 536 + 20 = 556 octeţi. Dimensiunea maximă a segmentului nu trebuie să fie aceeaşi în cele două direcţii.

În TCP conexiunile sunt stabilite utilizând „înţelegerea în trei paşi”. Pentru a stabili o conexiune, una din părţi - să spunem serverul - aşteaptă în mod pasiv o cerere de conexiune prin execuţia primitivelor LISTEN şi 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 şi numărul de port la care doreşte să se conecteze, dimensiunea maximă

a segmentului TCP pe care este dispusă să o accepte şi, opţional, o informaţie utilizator (de exemplu o parolă). Primitiva CONNECT trimite un segment TCP având bitul SYN poziţionat şi bitul ACK nepoziţionat, după care aşteaptă un răspuns.

Atunci când soseşte la destinaţie un segment, entitatea TCP receptoare verifică dacă nu cumva există un proces care a executat LISTEN pe numărul de port specificat în câmpul Port destinaţie. În caz contrar, trimite un răspuns cu bitul RST poziţionat, pentru a refuza conexiunea.

Dacă există vreun proces care ascultă la acel port, segmentul TCP recepţionat 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 secvenţa de segmente TCP transferate în caz de funcţionare normală. De notat că un segment SYN consumă un octet din spaţiul numerelor de secvenţă, astfel încât confirmarea să poată fi făcută fără ambiguităţi.

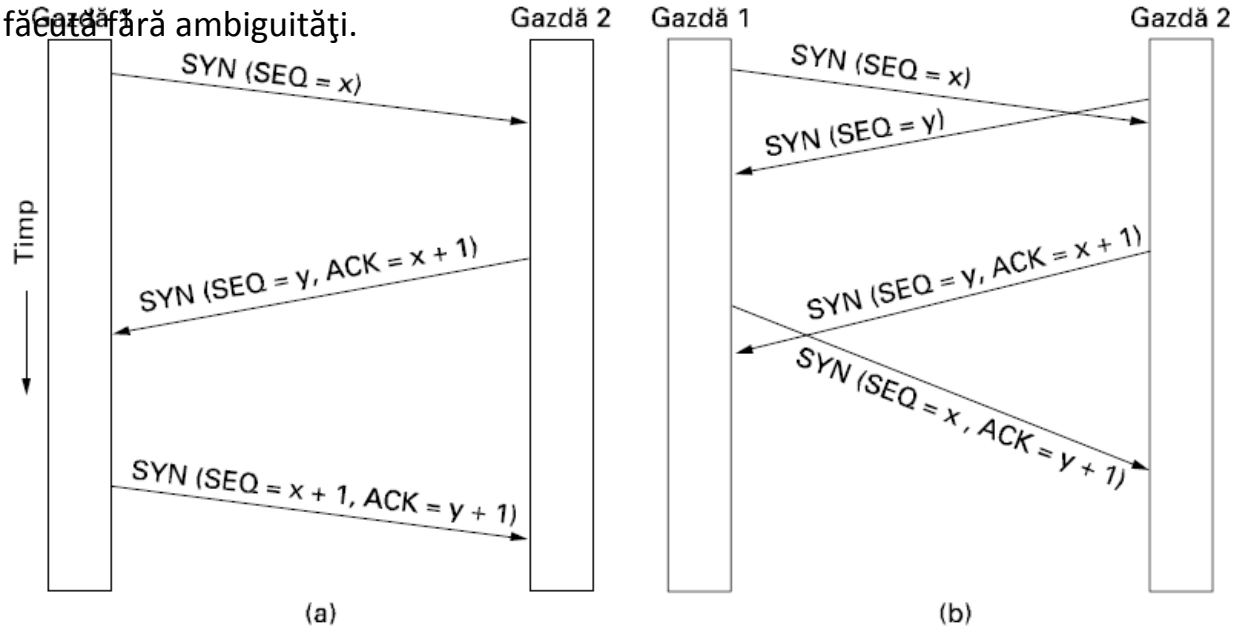


Fig. 6-31. (a) Stabilirea unei conexiuni TCP în cazul normal. (b) Coliziunea apelurilor.

Deşi conexiunile TCP sunt bidirecţionale, pentru a înţelege cum sunt desfiinţate conexiunile, cel mai bine este să ni le imaginăm sub forma unei perechi de legături unidirecţionale. Fiecare legătură unidirecţională 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 informaţie 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, informaţia poate fi transferată în continuare, pentru un timp nedefinit, în celălalt sens. Conexiunea este desfiinţată atunci când ambele direcţii au fost oprite. În mod normal, pentru a elibera o conexiune sunt necesare patru segmente TCP: câte un FIN şi un ACK pentru fiecare sens. Cu toate acestea, este posibil ca primul ACK şi cel de-al doilea FIN să fie cuprinse în acelaşi segment reducând astfel numărul total la trei.

Pentru a evita problema celor două armate, sunt utilizate cronometre. Dacă un răspuns la un FIN nu este recepţionat pe durata a cel mult două cicluri de maxime de viaţă ale unui pachet, emiţătorul FIN-ului eliberează conexiunea. Cealaltă parte va observa în final că nimeni nu mai pare să asculte la celălalt capăt al conexiunii, şi va elibera conexiunea în urma expirării unui interval de timp. Această soluţie nu este perfectă, dar având în vedere faptul că o soluţie perfectă este teoretic imposibilă, va trebui să ne mulţumim cu ce avem. În realitate astfel de probleme apar foarte rar.

Administrarea ferestrei în TCP nu este direct legată de confirmări, aşa cum se întâmplă la cele mai multe protocoale de nivel legătură de date. De exemplu, să presupunem că receptorul are un tampon de 4096 octeţi, aşa cum se vede în fig. 6-34. Dacă emiţătorul transmite un segment de 2048 de octeţi care este recepţionat corect, receptorul va confirma segmentul. Deoarece acum tamponul acestuia din urmă mai are liberi doar 2048 octeţi (până când aplicaţia şterge nişte date din acest tampon), receptorul va anunţa o fereastră de 2048 octeţi începând de la următorul octet aşteptat. Acum, emiţătorul transmite alţi 2048 octeţi, care sunt confirmaţi, dar fereastra oferită este 0. Emiţătorul trebuie să se oprească până când procesul aplicaţie de pe maşina receptoare a şters nişte date din tampon, moment în care TCP poate oferi o fereastră mai mare.

Atunci când fereastra este 0, în mod normal emiţătorul nu poate să transmită segmente, cu două excepţii. În primul rând, informaţia urgentă poate fi trimisă, de exemplu pentru a permite utilizatorului să oprească procesele rulând

pe maşina de la distanţă. În al doilea rând, emiţătorul poate trimite un segment de un octet pentru a determina receptorul să renunţe următorul octet aşteptat şi dimensiunea ferestrei. Standardul TCP prevede în mod explicit această opţiune pentru a preveni interblocarea în cazul în care se întâmplă ca anunţarea unei ferestre să fie vreodată pierdută.

TCP Tranzacţional

Mai devreme în acest capitol am analizat apelul de proceduri la distanţă ca modalitate de a implementa sistemele client-server. Dacă atât cererea, cât şi răspunsul sunt suficient de mici încât să se potrivească în pachete simple şi operaţia este idempotentă, UDP-ul poate fi uşor utilizat. Totuşi, dacă aceste condiţii nu sunt îndeplinite, utilizarea UDP-ului este mai puţin atractivă. De exemplu, dacă răspunsul este unul lung, atunci datagramele trebuie sa fie secvenţiate şi trebuie iniţiat un mechanism pentru a retransmite datagramele pierdute. De fapt, aplicaţiei 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 eficienţa. Secvenţa 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. A se reţine că acesta este cazul ideal. În cazul cel mai rău, cererea clientului şi FIN-ul sunt confirmate separat, precum sunt răspunsul server-ului şi FIN-ul.

Întrebarea care apare imediat este dacă există vreo posibilitate de a combina eficienţa 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 tranzacţional).

Ideea principală este aceea de a modifica secvenţa standard de iniţializare a conexiunii astfel încât să permită, la nivel înalt, transferul de date în timpul iniţializării. Protocolul T/TCP este prezentat în fig. 6-40(b). Primul pachet al clientului conţine bitul SYN, cererea în sine şi FIN-ul. De fapt acesta spune: vreau să stabilesc o conexiune, aici sunt datele şi am terminat Când serverul primeşte cererea, caută sau calculează răspunsul şi 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 şi protocolul ia sfârşit în (după) trei mesaje.

În orice caz, dacă rezultatul este mai mare de 1 pachet, serverul are de asemenea şi opţiunea de a nu seta bitul FIN, caz în care poate trimite pachete multiple înainte de a-i închide direcţia.

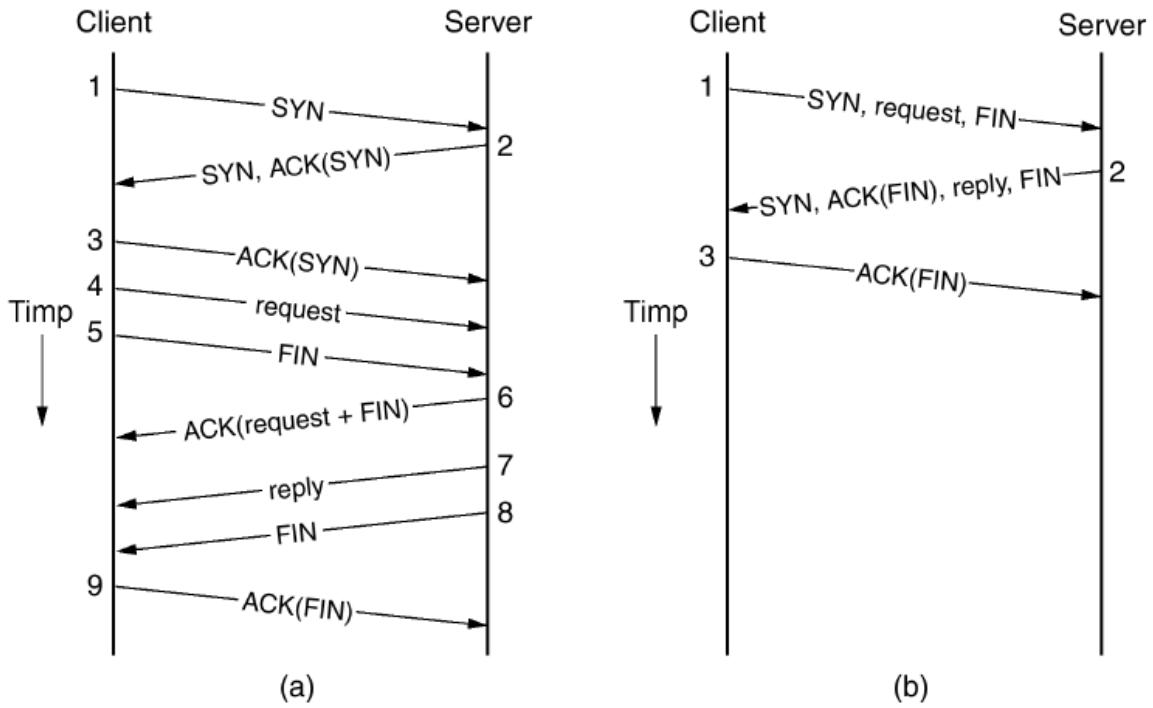


Fig. 6-40. (a) RPC folosind TCP clasic ; (b) RPC folosind T/TCP

28. DNS

Încă de la ARPANET exista un fişier host.txt care cuprindea toate sistemele gazdă şi adresele lor IP. În fiecare noapte, toate gazdele îl preluau de la site-ul unde era păstrat. Pentru o reţea formată din câteva sute de maşini mari, cu divizarea timpului, această abordare era destul de rezonabilă.

Totuşi, atunci când la reţea au fost conectate mii de staţii de lucru, toţi şi-au dat seama că această abordare nu putea să funcţioneze la nesfârşit. În primul rând dimensiunea fişierului ar deveni prea mare. Cu toate acestea şi chiar mai important, conflictele de nume de sisteme gazdă ar apărea în permanenţă dacă nu ar fi administrate centralizat, ceva de negândit într-o reţea internaţională de dimensiuni uriaşe din cauza încărcării şi a latenţei. Pentru a rezolva aceste probleme, a fost inventat **DNS** (**Domain Name System** - Sistemul numelor de domenii).

Esenţa DNS-ului constă într-o schemă ierarhică de nume de domenii şi a unui sistem de baze de date distribuite pentru implementarea acestei scheme de nume. În principal este utilizat pentru a pune în corespondenţă numele sistemelor gazdă şi adresele destinaţiilor de e-mail cu adresele IP, dar poate fi utilizat şi pentru alte scopuri.

Foarte pe scurt, DNS este utilizat după cum urmează. Pentru a stabili corespondenţa dintre un nume şi o adresă IP, programul de aplicaţie apelează o procedură de bibliotecă numită resolver, transferându-i numele ca parametru. Resolver-ul trimite un pachet UDP la serverul DNS local, care caută numele şi returnează adresa IP către resolver, care o returnează apelantului. Înarmat cu adresa IP, programul poate stabili o conexiune TCP cu destinaţia sau îi poate trimite pachete UDP.

Spaţiul de nume DNS

Conceptual, Internetul este divizat în peste 200 domenii de nivel superior, fiecare domeniu cuprinzând mai multe sisteme gazdă. Fiecare domeniu este partiţionat în subdomenii şi acestea sunt, la rândul lor, partiţionate ş.a.m.d. Toate aceste domenii pot fi reprezentate ca un arbore. Frunzele arborelui reprezintă domenii care nu au subdomenii (dar, bineînţeles, conţin sisteme). Un domeniu

frunză poate conţine un singur sistem gazdă sau poate reprezenta o firmă, deci să conţină mii de sisteme gazdă.

Domeniile de pe primul nivel se împart în două categorii: generice şi de ţări. Domeniile generice sunt com (comercial), edu (instituţii educaţionale), gov (guvernul federal al SUA), int (organizaţii internaţionale), mil (forţele armate ale SUA), net (furnizori Internet) şi org (organizaţii nonprofit). Domeniile de ţări includ o intrare pentru fiecare ţară.

Fiecare domeniu este identificat prin calea în arbore de la el la domeniul (fără nume) rădăcină. Componentele sunt separate prin puncte (pronunţat „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 înţelesul adevărat. În ambele cazuri, un nume de domeniu se referă la un anumit nod din arbore şi la toate nodurile de sub el.

Numele de domenii nu fac distincţie între litere mici şi litere mari, astfel edu, Edu, sau EDU înseamnă acelaşi lucru. Componentele numelor pot avea o lungime de cel mult 63 caractere, iar întreaga cale de nume nu trebuie să depăşească 255 de caractere.

Înregistrări de resurse

Fiecărui domeniu, fie că este un singur calculator gazdă, fie un domeniu de nivel superior, îi poate fi asociată o mulţime de înregistrări de resurse (resource records). Pentru un singur sistem gazdă, cea mai obişnuită 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 acelui nume. Astfel, adevărata

funcţie a DNS este să realizeze corespondenţa dintre numele de domenii şi înregistrările de resurse.

O înregistrare de resursă este un 5-tuplu. Cu toate că, din raţiuni de eficienţă, 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_viaţă 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 şi fiecare copie a bazei de date păstrează informaţii 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_viaţă (time_to_live) dă o indicaţie despre cât de stabilă este înregistrarea. Informaţia care este foarte stabilă are asigurată o valoare mare, cum ar fi 86400 (numărul de secunde dintr-o zi). Informaţiei instabile îi este atribuită o valoare mică, cum ar fi 60 (1 minut).

Al treilea câmp dintr-o înregistrare de resursă este Clasa (class). Pentru informaţiile legate de Internet este tot timpul IN. Pentru alte informaţii 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	Semnificaţie	Valoare
SOA	Start autoritate	Parametrii pentru această zonă
A	Adresa IP a unui sistem gazdă	Întreg pe 32 de biţi
MX	Schimb de poştă	Prioritate, domeniu dispus să accepte poştă 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 şi sistem de operare în ASCII
TXT	Text	Text ASCII neinterpretat

Fig. 7-2. Principalele tipuri de înregistrări de resurse DNS.

În final ajungem la câmpul Valoare. Acest câmp poate fi un număr, un nume de domeniu sau un şir ASCII. Semantica depinde de tipul de înregistrare.

Servere de nume

Teoretic, un singur server de nume poate conţine întreaga bază de date DNS şi 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.

Pentru a evita problemele asociate cu existenţa unei singure surse de informaţie, spaţiul de nume DNS este împărţit în zone care nu se suprapun. Fiecare zonă conţine câte o parte a arborelui precum şi numele serverelor care păstrează informaţia autorizată despre acea zonă. În mod normal, o zonă va avea un server de nume primar, care preia informaţia dintr-un fişier de pe discul propriu şi unul sau mai multe servere de nume secundare, care iau informaţiile de pe serverul primar. Pentru a îmbunătăţi 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 şi unde să fie plasate.

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 jurisdicţia 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 şi astfel este întotdeauna corectă. Înregistrările autorizate se deosebesc de înregistrările din memoria ascunsă, care pot fi expirate.

Dacă, totuşi, domeniul se află la distanţă, iar local nu este disponibilă nici o informaţie despre domeniul cerut, atunci serverul de nume trimite un mesaj de cerere la serverul de nume de pe primul nivel al domeniului solicitat. Apoi se

merge din subdomeniu in subdomeniu pana se gaseste inregistrarea cautata care e returnata pe traseul invers.

Odată ce aceste înregistrări de resurse ajung înapoi la serverul de nume initial, ele vor fi depuse în memoria ascunsă, pentru a fi folosite ulterior. Totuşi, această informaţie nu este autorizată, deoarece orice schimbare făcută la un domeniu nu se va propaga spre toate serverele care au folosit-o. Din acest motiv intrările în memoria ascunsă nu ar trebui să aibă viaţă prea lungă. Acesta este motivul pentru care câmpul Timp_de_viaţă este inclus în fiecare înregistrare de resursă. El informează serverele de nume aflate la distanţă cât timp să menţină înregistrările în memoria ascunsă. Dacă o anumită maşină are de ani de zile aceeaşi adresă IP, această informaţie ar putea fi păstrată timp de o zi. Pentru informaţii mai volatile este mai sigur ca înregistrările să fie eliminate după câteva secunde sau un minut.

De menţionat că metoda de interogare descrisă aici este cunoscută ca metoda de **interogare recursiva** (**recursive query**), deoarece fiecare server care nu are informaţia cerută o caută în altă parte şi raportează. Este posibilă şi o altă variantă. În acest caz, atunci când o cerere nu poate fi rezolvată local, cererea eşuează, dar este întors numele următorului server de pe calea ce trebuie încercată. Unele servere nu implementează interogarea recursivă şi întorc întotdeauna numele următorului server la care să se încerce.

De asemenea merită menţionat faptul că atunci când un client DNS nu reuşeşte să primească un răspuns înainte de expirarea timpului de căutare, data viitoare va încerca un alt server. Se presupune că serverul este probabil nefuncţional, nu că cererea sau răspunsul s-au pierdut.

29. Posta electronica

Arhitectură şi servicii

În această secţiune vom furniza o prezentare de ansamblu a ceea ce pot face sistemele de poştă electronică şi cum sunt ele organizate. Aceste sisteme constau de obicei din două subsisteme: agenţii-utilizator, care permit utilizatorilor să citească şi să trimită scrisori prin poşta electronică şi agenţii de transfer de mesaje, care transportă mesajele de la sursă la destinaţie. Agenţii-utilizator sunt programe locale, care furnizează o metodă de a interacţiona cu sistemul de e-mail bazată pe comenzi, meniuri sau grafică. Agenţii 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 poştă electronică pun la dispoziţie cinci funcţii de bază. Să aruncăm o privire asupra lor.

Compunerea se referă la procesul de creare a mesajelor şi a răspunsurilor. Deşi pentru corpul mesajului poate fi folosit orice editor de texte, sistemul însuşi poate acorda asistenţă la adresare şi la completarea numeroaselor câmpuri antet ataşate fiecărui mesaj. De exemplu, când se răspunde la un mesaj, sistemul poate extrage adresa iniţiatorului din mesajul primit şi 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 destinaţa, sau la o maşină intermediară, emiterea mesajului şi eliberarea conexiunii. Sistemul de poştă ar trebui să facă acest lucru singur, fără a deranja utilizatorul.

Raportarea se referă la informarea iniţiatorului despre ce s-a întâmplat cu mesajul. A fost livrat? A fost respins? A fost pierdut? Există numeroase aplicaţii în care confirmarea livrării este importantă şi poate avea chiar semnificaţie juridică.

Afişarea mesajelor primite este necesară pentru ca utilizatorii să-şi poată citi poşta. Uneori sunt necesare conversii sau trebuie apelat un program de vizualizare special; de exemplu, dacă mesajul este un fişier PostScript, sau voce digitizată. Se mai încearcă uneori şi conversii simple şi formatări.

Dispoziţia este pasul final şi se referă la ceea ce face receptorul cu mesajul, după ce l-a primit. Posibilităţile includ eliminarea sa înainte de a-l citi, aruncarea sa după citire, salvarea sa ş.a.m.d. Ar trebui de asemenea să fie posibilă regăsirea şi recitirea de mesaje deja salvate, trimiterea lor mai departe, sau procesarea lor în alte moduri.

Majoritatea sistemelor permit utilizatorilor să-şi creeze **cutii poştale** (**mailboxes**) pentru a păstra mesajele sosite. Sunt necesare comenzi de creare şi distrugere a cutiilor poştale, de inspectare a conţinutului acestora, de inserare şi de ştergere de mesaje din cutii poştale ş.a.m.d.

Managerii de companii au adesea nevoie să trimită un acelaşi mesaj fiecărui subordonat, client sau furnizor. Acest lucru dă naştere ideii de **listă de poştă** (**mailing list**), care este o listă de adrese de poştă electronică. Când un mesaj este trimis la lista de poştă, copii identice ale sale sunt expediate fiecăruia dintre cei de pe listă.

Alte caracteristici evoluate sunt copii la indigo, poştă de prioritate mare, poştă secretă (criptată), receptori alternativi, dacă cel primar nu este disponibil, şi posibilitatea de a permite secretarelor să se ocupe de poşta primită de şefii lor.

O idee fundamentală în toate sistemele moderne de e-mail este distincţia dintre **plic** şi **conţinutul** său. Plicul încapsulează mesajul. Conţine toată informaţia necesară pentru transportul mesajului, cum ar fi destinaţia, adresa, prioritatea, nivelul de securitate, toate acestea fiind distincte de mesajul în sine.

Agenţii de transfer de mesaje folosesc plicul pentru rutare (dirijare), aşa cum face şi oficiul poştal. Mesajul din interiorul plicului conţine două părţi: **antetul** şi **corpul**. Antetul conţine informaţie de control pentru agenţii utilizator. Corpul mesajului se adresează în întregime utilizatorului uman.

Trimiterea poştei electronice

Pentru a trimite un mesaj prin poşta electronică, un utilizator trebuie să furnizeze mesajul, adresa destinaţie, şi eventual alţi câţiva parametri. Mesajul poate fi produs cu un editor de texte de sine stătător, cu un program de procesare de text sau, eventual, cu un editor de texte specializat, construit în interiorul agentului utilizator. Adresa de destinaţie trebuie să fie într-un format cu care agentul utilizator să poată lucra. Mulţi agenţi-utilizator solicită adrese de forma utilizator@adresă-dns.

Majoritatea sistemelor de e-mail acceptă liste de poştă, astfel că un utilizator poate trimite, cu o singură comandă, un acelaşi mesaj tuturor persoanelor dintr-o listă. Dacă lista de poştă este păstrată local, agentul-utilizator poate pur şi simplu să trimită câte un mesaj separat fiecăruia dintre receptorii doriţi. Dacă lista este păstrată la distanţă, atunci mesajele vor fi expandate acolo.

Citirea poştei electronice

În mod obişnuit, când este lansat un agent-utilizator, înainte de a afişa ceva pe ecran, el se va uita în cutia poştală a utilizatorului după mesajele care sosesc. Apoi poate anunţa numărul de mesaje din cutie, sau poate afişa pentru fiecare mesaj câte un rezumat de o linie, pentru ca apoi să aştepte o comandă.

Continutul unei cutii postale poate avea mai multe campuri: primul câmp reprezintă numărul mesajului. Al doilea câmp, Marcaje, poate conţine un K, însemnând că mesajul nu este nou, dar a fost citit anterior şi păstrat în cutia poştală; un A, însemnând că deja s-a răspuns la acest mesaj; şi/sau un F, însemnând că mesajul a fost trimis mai departe altcuiva. Sunt de asemenea

posibile şi alte marcaje. Al treilea câmp specifică lungimea mesajului şi al patrulea spune cine a trimis mesajul. Din moment ce el este pur şi simplu extras din mesaj, acest câmp poate conţine prenume, nume complete, iniţiale, nume de cont, sau orice altceva şi-a ales transmiţătorul să pună. În sfârşit, câmpul Subiect specifică despre ce este mesajul, într-un scurt rezumat. Persoanele care omit să includă un camp Subiect adesea descoperă că răspunsurile la scrisorile lor tind să nu obţină prioritate maximă.

Formatele mesajelor

Mesajele constau dintr-un plic simplu (descris în RFC 821), un număr de câmpuri antet, o linie goală şi apoi corpul mesajului. Fiecare câmp antet se compune (din punct de vedere logic) dintr-o singură linie de text ASCII, conţinând numele câmpului, două puncte, şi, pentru majoritatea câmpurilor, o valoare.

La o utilizare normală, agentul-utilizator construieşte un mesaj şi îl transmite agentului de transfer de mesaje, care apoi foloseşte unele dintre câmpurile antet pentru a construi plicul efectiv, o combinaţie oarecum demodată de mesaj şi plic.

Principalele câmpuri antet, legate de transportul de mesaje: Câmpul To: oferă adresa DNS a receptorului primar. Este permisă de asemenea existenţa de receptori multipli. Câmpul Cc: dă adresa oricărui receptor secundar. În termenii livrării, nu este nici o diferenţă între un receptor primar şi unul secundar. Este în întregime o deosebire psihologică, ce poate fi importantă pentru persoanele implicate, dar este neimportantă pentru sistemul de poştă. Câmpul Bcc: (Blind carbon copy - copie confidenţială la indigo) este la fel ca Cc:, cu excepţia că această linie este ştearsă din toate copiile trimise la receptorii primari şi secundari. Acest element permite utilizatorilor să trimită copii unei a treia categorii de receptori, fără ca cei primari şi secundari să ştie acest lucru. Următoarele două câmpuri, From: şi Sender:, precizează cine a scris şi respectiv cine a trimis mesajul. Acestea pot să nu fie identice.

O linie conţinând Received: este adăugată de fiecare agent de transfer de mesaje de pe traseu. Linia conţine identitatea agentului, data şi momentul de

timp la care a fost primit mesajul şi alte informaţii care pot fi utilizate pentru găsirea defecţiunilor în sistemul de dirijare.

Câmpul Return-Path: este adăugat de agentul final de transfer de mesaje şi are în intenţie să indice cum se ajunge înapoi la transmiţător. În teorie, această informaţie poate fi adunată din toate antetele Received: (cu excepţia numelui cutiei poştale a transmiţătorului), dar rareori este completată aşa şi de obicei conţine chiar adresa transmiţătorului. Câmpul Reply-To: este uneori utilizat când nici persoana care a compus mesajul, nici cea care l-a trimis nu vrea să vadă răspunsul.

MIME - Multipurpose Internet Mail Extensions

Ideea fundamentală a MIME este să continue să folosească formatul RFC 822, dar să adauge structură corpului mesajului şi să definească reguli de codificare pentru mesajele non-ASCII. Deoarece respectă RFC 822, mesajele MIME pot fi trimise utilizând programele şi protocoalele de poştă existente. Tot ceea ce trebuie modificat sunt programele de transmitere şi recepţie, pe care utilizatorii le pot face ei înşişi.

MIME defineşte cinci noi antete de mesaje, aşa cum se arată în fig. 7-11.

Antet	Conținut
MIME-Version:	Identifică versiunea de MIME
Content-Description:	Șir adresat utilizatorului care spune ce este în mesaj
Content-Id:	Identificator unic
Content-Transfer-Encoding:	Cum este împachetat corpul pentru transmisie
Content-Type:	Natura mesajului

Fig. 7-11. Antetele RFC 822 adăugate de către MIME.

Transferul mesajelor

SMTP – Simple Mail Transfer Protocol(Protocol simplu de transfer de poştă)

În cadrul Internetului poşta electronică este livrată prin stabilirea de către maşina sursă a unei conexiuni TCP la portul 25 al maşinii de destinaţie. La acest

port se află un demon de e-mail care ştie SMTP (Simple Mail Transfer Protocol). Acest demon acceptă conexiunile şi copiază mesajele de la ele în cutiile poştale corespunzătoare. Dacă mesajul nu poate fi livrat, se returnează transmiţătorului un raport de eroare conţinând prima parte a mesajului nelivrat.

SMTP este un protocol simplu de tip ASCII. După stabilirea conexiunii TCP la portul 25, maşina transmiţătoare, operând în calitate de client, aşteaptă ca maşina receptoare, operând ca server, să vorbească prima. Serverul începe prin a trimite o linie de text, declarându-şi identitatea şi spunând dacă este pregătit sau nu să primească mesaje. Dacă nu este, clienţii eliberează conexiunea şi încearcă din nou mai târziu.

Dacă serverul este dispus să primească e-mail, clientul anunţă de la cine vine scrisoarea şi cui îi este adresată. Dacă un asemenea receptor există la destinaţie, serverul îi acordă clientului permisiunea să trimită mesajul. Apoi clientul trimite mesajul şi serverul îl confirmă. În general nu este necesară ataşarea unei sume de control deoarece TCP furnizează un flux sigur de octeţi. Dacă mai există şi alte mesaje, acestea sunt trimise tot acum. Când schimbul de mesaje, în ambele direcţii, s-a încheiat, conexiunea este eliberată.

Chiar dacă protocolul SMTP este bine definit, mai pot apărea câteva probleme. O problem 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 şi client, unul din ei poate renunţa, în timp ce celălalt este încă ocupat, întrerupând conexiunea în mod neaşteptat. În sfârşit, în unele situaţii, pot fi lansate schimburi infinite de mesaje.

Pentru a atinge câteva dintre aceste probleme, în RFC 2821 s-a definit protocolul SMTP extins (ESMTP). Clienţii care doresc să-l utilizeze trebuie să trimită iniţial un mesaj EHLO în loc de HELO. Dacă acesta este rejectat, atunci serverul este unul standard de tip SMTP şi clientul va trebui să se comporte în modul obişnuit. Dacă EHLO este acceptat, înseamnă ca sunt permise noile comenzi şi noii parametri.

Livrarea finală

Până acum, am presupus că toţi utilizatorii lucrează pe maşini capabile să trimită şi să primească e-mail. După cum am văzut, e-mail-ul este livrat prin stabilirea unei conexiuni TCP între expeditor şi destinatar şi apoi prin trimiterea e-mail-ului prin ea. Acest model a funcţionat bine zeci de ani, atât timp cât toate calculatoarele din ARPANET (şi mai târziu din Internet) erau, de fapt, conectate la reţea şi gata să accepte conexiuni TCP.

Totuşi, odată cu apariţia celor care accesează Internet-ul folosind un modem cu care se conectează la ISP-ul lor, acest lucru nu mai ţine. Problema este următoarea: Ce se întâmplă când Elinor vrea să-i trimită Carolynei un e-mail şi Carolyn nu este conectată la reţea în acel moment? Elinor nu va putea să stabilească o conexiune TCP cu Carolyn şi astfel, nu va putea utiliza protocolul SMTP.

O soluţie este ca agentul de transfer de mesaje de pe o maşină ISP să accepte e-mail-ul pentru clienţii săi şi să-l stocheze în cutiile lor poştale pe o maşină a ISP-ului. Din moment ce acest agent poate fi conectat la reţea tot timpul, se poate trimite e-mail 24 de ore pe zi.

POP3

Din nefericire, această soluţie dă naştere altei probleme: cum îşi ia utilizatorul e-mail-ul de la agentul de transfer de mesaje al ISP-ului? Soluţia acestei probleme este crearea unui alt protocol care să permită agenţilor de transfer mesaje (aflaţi pe calculatoarele clienţilor) să contacteze agentul de transfer mesaje (de pe o maşină ISP) şi 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 poştă, versiunea 3).

POP3 începe când utilizatorul porneşte programul cititor de poştă (mail reader). Acesta sună la ISP (în caz că nu există deja o conexiune) şi stabileşte 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 se referă la admiterea utilizatorului în sistem (login).
2. Tranzacţionare tratează colectarea e-mail-urilor şi marcarea lor pentru ştergere din cutia poştală.
3. Actializare se ocupă cu ştergerea efectivă a mesajelor.

Deşi este adevărat 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 şi golesc cutia poştală. Ca urmare, practic singura copie rămâne înregistrată pe discul utilizatorului. Dacă acesta se strică, toate e-mail-urile pot fi pierdute definitiv.

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 şi larg folosit datorită simplităţii şi robusteţii sale. Totuşi, există în industria calculatoarelor un adevăr bine înrădăcinat, acela că imediat ce un lucru funcţionează bine, cineva va începe să ceară mai multe facilităţi (şi să aibă mai multe probleme). Asta s-a întâmplat şi cu e-mail-ul. De exemplu, multă lume are un singur cont de e-mail la serviciu sau la şcoală şi vrea săl acceseze de pe PC-ul de acasă, de pe calculatorul portabil în călătoriile de afaceri şi din Internet café-uri în vacanţe. 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 maşini, mai mult sau mai puţin la întâmplare, unele dintre ele nefiind ale utilizatorului.

Acest dezavantaj a dat naştere unei alternative a protocolului de livrare finală, **IMAP** (**Internet Message Access Protocol** – Protocol pentru accesul mesajelor în Internet). Spre deosebire de POP3, care în mod normal presupune că utilizatorul îşi va goli căsuţa poştală la fiecare conectare şi va lucra deconectat de la reţea (off-line) după aceea, IMAP presupune că tot email-ul va rămâne pe server oricât de mult, în mai multe căsuţe poştale.

IMAP prevede mecanisme extinse pentru citirea mesajelor sau chiar a părţilor de mesaje, o facilitate folositoare când se utilizează un modem încet pentru citirea părţii textuale a unui mesaj cu mai multe părţi audio şi 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 şi manipularea mai multor cutii poştale pe server. Astfel, un utilizator poate păstra o cutie poştală pentru fiecare corespondent şi poate muta aici mesajele din inbox după ce acestea au fost citite.

IMAP are multe facilităţi, ca de exemplu posibilitatea de a se referi la un mesaj nu prin numărul de sosire, ca în fig. 7-8, ci utilizând atribute (de exemplu, dă-mi primul mesaj de la Bobbie). Spre deosebire de POP3, IMAP poate de asemenea să accepte atât expedierea mesajelor spre destinaţie cât şi livrarea mesajelor venite.

Stilul general al protocolului IMAP este similar cu cel al POP3-ului, cu excepţia faptului că există zeci de comenzi. Serverul IMAP ascultă pe portul 143. În fig. 7-17 este prezentată o comparaţie între POP3 şi IMAP. E bine de notat, totuşi, că nu toate ISP-urile oferă ambele protocoale şi că nu toate programele de e-mail le suportă pe amândouă. Aşadar, atunci când alegeţi un program de e-mail, este important să aflaţi ce protocoale suportă şi să vă asiguraţi că ISP-ul oferă cel puţin 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 poştale	Nu	Da
Cine face copii de siguranţă la cutiile poştale	Utilizatorul	ISP-ul
Bun pentru utilizatorii mobili	Nu	Da
Controlul utilizatorului asupra scrisorilor preluate	Mic	Mare
Descărcare parţială 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 creştere

Fig. 7-17. O comparaţie între POP3 şi IMAP.

30. HTTP – HyperText Transfer Protocol

Conexiuni

Modul uzual prin care un program de navigare contactează un server este de a stabili o conexiune TCP pe portul 80 pe maşina serverului, deşi această procedură nu este cerută formal. Avantajul de a folosi TCP este că nici programele de navigare şi nici serverele nu trebuie sa se preocupe de mesajele pierdute, mesajele duplicate, mesajele lungi, sau mesajele de confirmare. Toate aceste aspect sunt tratate de implementarea TCP.

În HTTP 1.0, după ce conexiunea era stabilită, o singură cerere era transmisă şi un singur răspuns era primit înapoi. Apoi conexiunea TCP era eliberată. Într-o lume în care pagina tipică Web consta în întregime din text HTML, această metodă era adecvată. În câţiva ani însă, o pagină medie Web conţinea un număr mare de iconiţe, imagini şi alte lucruri plăcute vederii, astfel că stabilirea unei conexiuni TCP pentru a prelua o singură iconiţă a devenit un mod foarte costisitor de a opera.

Această observaţie a dus la apariţia HTTP 1.1, care suportă conexiuni persistente. Cu ele, este posibilă stabilirea unei conexiuni TCP, trimiterea unei cereri şi

obţinerea unui răspuns, apoi trimiterea unor cereri adiţionale şi obţinerea de răspunsuri adiţionale. Prin distribuirea pornirii şi 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 intenţionat mai general decât era necesar în perspectiva aplicaţiilor orientate pe obiecte. Pentru aceasta sunt suportate operaţiile, denumite metode, care fac mai mult decât cele care doar cer o pagină Web. Această consideraţie generală a permis apariţia 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 adiţionale specifice obiectelor pot fi de asemenea disponibile.

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	Interogarea anumitor opțiuni

Fig. 7-41. Metode de cerere standard pentru HTTP.

Metoda GET cere serverului să trimită pagina (prin care noi înţelegem obiect, în cel mai general caz, dar în practică de obicei doar un fişier). Pagina este codată corespunzător în MIME. Marea majoritate a cererilor către servere Web sunt metode GET. Forma uzuală a metodei GET este: **GET fişier HTTP-1.1**, unde fişier denumeşte resursa (fişierul) ce va fi adusă, şi 1.1 este versiunea de protocol utilizat.

Metoda HEAD cere doar antetul mesajului, fără să ceară şi pagina propriu-zisă. Această metodă poate să fie utilizată pentru a afla când s-a făcut ultima modificare, pentru a obţine informaţii pentru indexare, sau numai pentru a verifica corectitudinea unui URL.

Metoda PUT este inversa metodei GET: în loc să citească o pagină, o scrie. Această metodă permite crearea unei colecţii de pagini de Web pe un server la distanţă. Corpul cererii conţine pagina. Pagina poate să fie codificată utilizând MIME, caz în care liniile care urmează după PUT pot include Content-Type şi antete de autentificare, pentru a demonstra că într-adevăr cel care face cerereaare dreptul de a realiza operaţia cerută.

Similară metodei PUT este **metoda POST**. Şi ea conţine 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 ştiri sau adăuga un fişier la un sistem de informare în reţea. În practică, nici PUT şi nici POST nu sunt utilizate prea mult.

Metoda DELETE realizează ce era de aşteptat: ştergerea unei pagini. Ca şi la PUT, autentificarea şi drepturile de acces joacă aici un rol important. Nu există nici o garanţie asupra succesului operaţiei DELETE, deoarece chiar dacă serverul doreşte să execute ştergerea, fişierul poate să aibă attribute care să interzică serverului HTTP să îl modifice sau să îl şteargă.

Metoda TRACE este pentru verificarea corectitudinii. Ea cere serverului să trimită înapoi cererea. Această metodă este utilă când cererile nu sunt procesate corect şi clientul vrea să ştie 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 proprietăţile acestuia sau despre cele ale unui anumit fişier.

Fiecare cerere obţine un **răspuns** ce constă din **linia de stare** şi posibile **informaţii suplimentare** (de exemplu, o parte sau toată pagina Web). Linia de

stare conţine un cod de stare de trei cifre, indicând dacă cererea a fost satisfăcută şi dacă nu, cauza. Prima cifră este utilizată pentru împărţirea răspunsurilor în cinci mari grupuri, ca în fig. 7-42.

Cod	Semnificație	Exemple
1xx	Informație	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 adiţionale cu mai multe informaţii. Acestea poartă numele de antete de cerere. Această informaţie 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 selecţie a celor mai importante este dată în fig. 7-43.

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 transmițătorul vrea să comute
Server	Răspuns	Informație despre server
Content-Encoding	Răspuns	Cum este codat conținutului (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.

31. FTP

Descriere :

Standard pentru transfer de fisiere (RFC959). Precede TCP/IP, dar a fost adaptat ulterior la TCP/IP. Este un protocol general, cu urmatoarele caracterisitci:

– Independent de SO si de hardware

– Transfera fisiere oarecare

– Gestioneaza drepturile asupra fisierelor si restrictiile de acces

- Poate rula fie interactiv fie automat.

Protocol permite actiuni de:

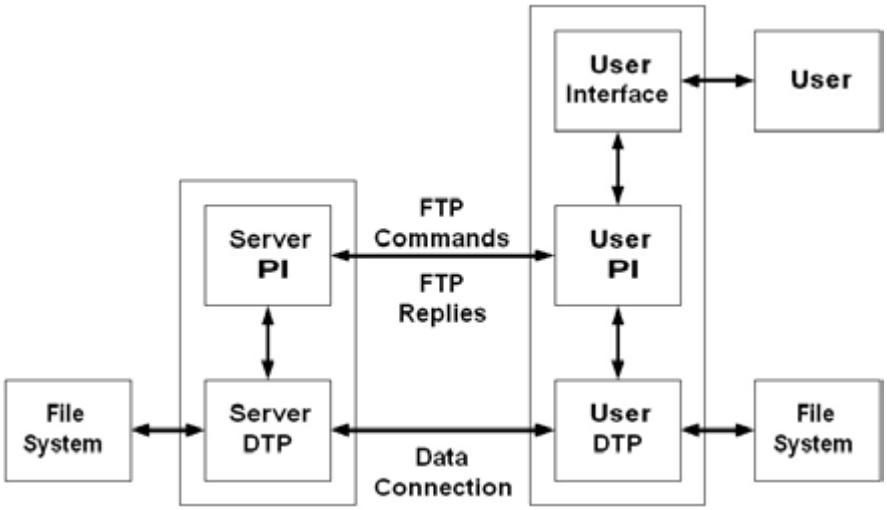
– Listarea continutului unui director

– Modificarea directorului curent

– Aducerea unui fisier

– Trimiterea unui fisier

Modelul FTP



Modelul FTP foloseste doua conexiuni: una de control si una de date. PI – Protocol Interpreter implementeaza protocolul FTP, iar DTP - Data Transfer Process DTP realizeaza transferul. User PI (Protocol Interpreter) initiaza conexiunea de control (lucreaza dupa protocolul telnet). Dupa stabilirea ei, **comenzile** generate de User PI sunt transmise serverului, care transmite **raspunsuri** standard generate de Server PI.

Comenzile FTP specifica parametrii pentru conexiunea de date, cum ar fi portul de date, modul de transfer (stream, block, compressed), tipul reprezentarii

(text, binar etc.). De regula, User DTP (User Data Transfer Protocol) trebuie sa asigure un "listen" la portul desemnat. Acest port poate fi in acelasi sistem cu User PI sau in alt sistem unde se asigura ascultarea. Conexiunea de date poate fi folosita pentru transmitere si receptie simultana de date.

Cateva Comenzi / Raspunsuri FTP

Comanda Descriere

ABOR Abort proces de conexiune de date
ALLO <bytes> Aloca spatiu fisier pe server
CWD <dir path> Schimba director de lucru pe server
DELE <filename> Sterge fisier specificat, pe server
MODE <mode> Mod de transfer (S=stream, B=block, C=compressed).
MKD <directory> Creaza director specificat, pe server
QUIT De-logheaza de pe server
TYPE <data type> Tip date (A-ASCII, E-EBCDIC, B-binar)
USER <username> username pentru login
PASS <password> parola pentru login

Raspuns Descriere

150 Deschide conexiunea
200 OK
220 Serviciu pregatit
221 De-logare retea
226 Inchide conexiunea de date
250 Actiune terminata

Conexiuni :

Conexiunea de date

Serverul la distanta accepta stabilirea unei conexiuni de control de la client, dupa care se stabileste conexiunea de date. Se pot folosi doua metode:

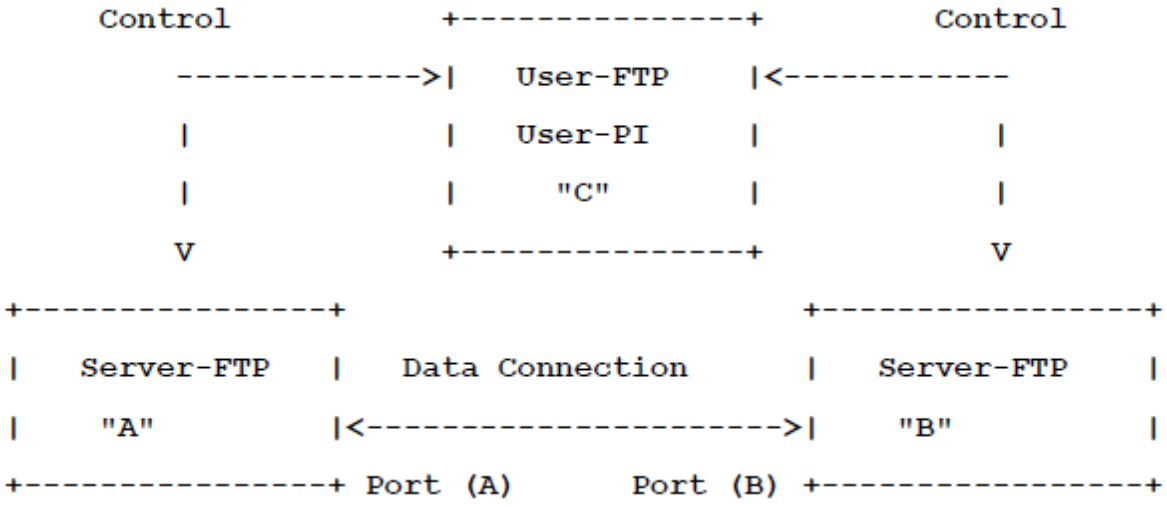
- Activa – serverul se conecteaza la client, clientul specifica o adresa si un port;
  serverul initiaza conexiunea (ex. port 1931, adresa 192.168.1.2)

Client: PORT 192,168,1,2,7,139
Server: 200 PORT command successful.

- Pasiva – clientul initiaza conexiunea, clientul cere serverului sa asculte la o adresa si un port (care nu este portul sau standard); serverul comunica adresa si portul (ex. port 4023, adresa 172.16.62.36);

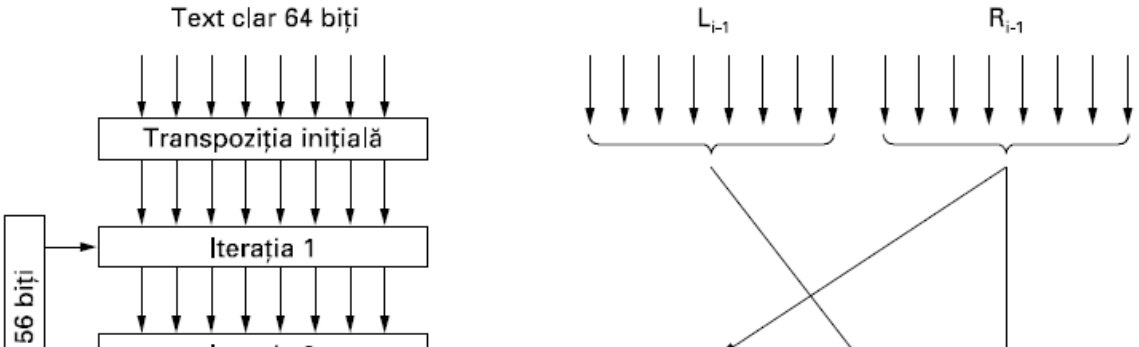
Client: PASV
Server: Entering Passive Mode (172,16,62,36,133,111)

Interactiune server-server
Un User PI poate stabili conexiuni de control pentru a face transferul de date intre alte doua servere, nici unul local. Mecanismul se numeste transfer "third party" sau transfer "proxy". Utilizatorul din C poate initia o conexiune de control primara cu serverul A si o conexiune de control secundara cu serverul B. Apoi, utilizatorul din C stabileste o conexiune de date intre A si B, prin care se transfera datele direct intre cele doua servere. Controlul se face prin intermediul lui C care joaca rolul de intermediar. Cand trebuie transferate date intre A si B, User PI din C seteaza conexiunile de control si apoi transmite lui A o comanda de conectare pasiva PASV prin care-i spune sa asculte la portul sau de date (nu sa initieze o conexiune). Cand User PI primeste o confirmare la PASV (care specifica identitatea lui A si portul la care asculta), User PI din C transmite catre B portul lui A intr-o comanda PORT si primeste de la B un raspuns. User PI din C transmite comenzile de transfer de date catre A si B. B initiaza conexiunea si transferul incepe.



32. Algoritmi cu cheie secreta

DES – Data Encryption Standard



O prezentare generală a DES este făcută în fig. 8-7(a). Textul clar este criptat în blocuri de câte 64 de biţi, rezultând blocuri de 64 de biţi de text cifrat. Algoritmul, care este parametrizat cu o cheie de 56 de biţi, are 19 runde distincte. Prima rundă este o transpoziţie independentă de cheie, aplicată asupra textului clar de 64 de biţi. Ultima rundă este exact inversa acestei transpoziţii. Penultima rundă schimbă cei mai din stânga 32 de biţi cu cei mai din dreapta 32 de biţi. Cele 16 runde rămase sunt funcţional identice dar sunt parametrizate de funcţii de cheie diferite. Algoritmul a fost proiectat pentru a permite ca decriptarea să se facă cu aceeaşi cheie ca şi criptarea, o proprietate necesară în orice algoritm cu cheie secretă. Paşii sunt parcurşi în ordine inversă.

Funcţionarea unuia dintre paşii intermediari este ilustrată în fig. 8-7(b). Fiecare rundă ia două intrări de 32 de biţi şi produce două ieşiri de 32 de biţi. Ieşirea din stânga este o simplă copie a intrării din dreapta. Ieşirea din dreapta rezultă în urma unui SAU exclusiv (XOR) bit cu bit între intrarea din stânga şi o funcţie depinzând de intrarea din dreapta şi de o cheie pentru această rundă, Ki. Toată complexitatea rezidă în această funcţie.

Funcţia constă din patru paşi, parcurşi în secvenţă. În primul rând, este construit un număr de 48 de biţi, E, prin expandarea celor 32 de biţi ai lui Ri-1 în concordanţă cu o regulă de transpoziţie fixă şi de duplicare. În al doilea rând, E şi Ki sunt combinate prin XOR. Ieşirea este apoi împărţită în opt grupuri de câte 6 biţi şi fiecare dintre acestea este introdus într-o cutie S diferită. Fiecare dintre cele 64 de intrări posibile într-o cutie S este pusă în corespondenţă cu o ieşire de 4 biţi. În final, aceşti 8 × 4 biţi sunt trecuţi printr-o cutie P.

În fiecare din cele 16 iteraţii este folosită o cheie diferită. Înainte de începerea algoritmului este aplicată o transpoziţie de 56 de biţi asupra cheii. Chiar înainte de începerea fiecărei iteraţii, cheia este partiţionată în două unităţi de câte 28 de biţi, fiecare dintre ele este rotită la stânga cu un număr de biţi depinzând de numărul iteraţiei. Ki este derivat din această cheie rotită prin aplicarea unei transpoziţii pe 56 de biţi asupra ei. La fiecare rundă este extrasă şi permutată o altă submulţime de 48 de biţi din cei 56 de biţi.

Triplu DES

Încă din 1979, IBM a realizat că lungimea cheii DES era prea mică şi a conceput un mod de a o creşte efectiv, folosind tripla criptare (Tuchman, 1979). Metoda aleasă, care de atunci a fost încorporată în Standardul Internaţional 8732, este ilustrată în fig. 8-8. Aici sunt folosite două chei şi trei runde.

În prima rundă textul clar este în mod obişnuit criptat cu K1. În a doua rundă, este rulat DES în mod de decriptare, folosind cheia K2. În final, este efectuată o altă criptare cu K1.

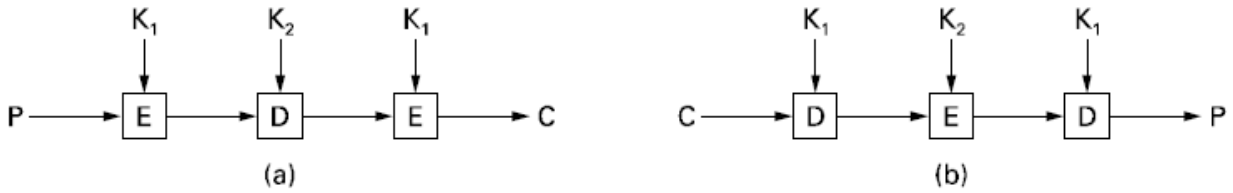


Fig. 8-8. (a)Tripla criptare folosind DES. (b) Decriptarea.

Această proiectarea dă naştere imediat la două întrebări. Prima, de ce sunt folosite doar două chei în loc de trei? A doua, de ce este folosită succesiunea de transformări EDE (eng.: Encrypt Decrypt Encrypt, rom.: Criptare Decriptare Criptare), în loc de EEE (eng.: Encrypt Encrypt Encrypt, rom.: Criptare Criptare Criptare)? Motivul pentru care sunt utilizate două chei este Acela că chiar şi cei mai paranoici criptografi admit că 112 biţi sunt suficienţi pentru aplicaţiile comerciale de rutină, la momentul actual. A extinde la 168 de biţi înseamnă a adăuga o supraîncărcare inutilă pentru gestiunea şi transportul unei alte chei cu un câştig real redus.

Motivul pentru criptare, decriptare şi apoi iar criptare este compatibilitatea cu produsele existente ce folosesc sisteme DES cu o singură cheie. Atât funcţia de criptare cât şi cea de decriptare sunt corespondenţe între mulţimi de numere pe 64 de biţi. Din punct de vedere criptografic, cele două corespondenţe sunt la fel de puternice. Totuşi, folosind EDE în locul EEE, un calculator ce utilizează tripla criptare poate comunica cu unul ce foloseşte criptarea simplă, folosind K1=K2.

AES – Advanced Encryption Standard (Rijndael)

Rijndael permite lungimi de chei şi mărimi de blocuri de la 128 de biţi la 256 de biţi în paşi de câte 32 biţi. Lungimea cheii şi lungimea blocului pot fi alese în mod independent. Cu toate acestea, AES specifică faptul că mărimea blocului trebuie să fie de 128 de biţi şi lungimea cheii trebuie să fie de 128, 192, sau 256 de biţi. E îndoielnic că cineva va folosi vreodată cheile de 192 de biţi, astfel că de fapt, AES are două variante: bloc de 128 de biţi cu cheie de 128 de biţi şi bloc de 128 de biţi cu cheie de 256 de biţi.

Ca şi DES, Rijndael foloseşte substituţie şi permutări, ca şi runde multiple. Numărul de runde depinde de mărimea cheii şi de mărimea blocului, fiind 10 pentru o cheie de 128 de biţi cu blocuri de 128 biţi şi mărindu-se până la 14 pentru cheia cu cea mai mare dimensiune şi blocul cu cea mai mare dimensiune. Totuşi, spre deosebire de DES, toate operaţiile sunt la nivel de octet, pentru a permite implementări eficiente hardware şi software. Codul este dat în fig. 8-9.

```
#define LENGTH 16                                   /* # octeți în blocul de date sau în cheie */
#define NROWS 4                                     /* număr de linii din stare */
#define NCOLS 4                                     /* număr de coloane din stare */
#define ROUNDS 10                                   /* număr de iterații */
typedef unsigned char byte;                         /* întreg fără semn pe 8 biți */

rijndael(byte plaintext[LENGTH], byte ciphertext[LENGTH], byte key[LENGTH])
{
  int r;                                            /* index pentru iterare */
  byte state[NROWS][NCOLS];                         /* starea curentă */
  struct {byte k[NROWS][NCOLS];} rk[ROUNDS + 1];    /* cheile pentru runde */

  expand_key(key, rk);                              /* construiește cheile pentru runde */
  copy_plaintext_to_state(state, plaintext);        /* inițializează starea curentă */
  xor_roundkey_into_state(state, rk[0]);            /* face XOR între cheie și stare */

  for (r = 1; r <= ROUNDS; r++) {
    substitute(stare);                              /* aplică substituția fiecărui octet */
    rotate_rows(state);                             /* rotește rândul i cu i octeți */
    if (r < ROUNDS) mix_columns(state);             /* funcție de amestecare */
    xor_roundkey_into_state(state, rk[r]);          /* face XOR între cheie și stare */
  }
  copy_state_to_ciphertext(ciphertext, state);      /* întoarce rezultatul */
}
```

Fig. 8-9. Rijndael în linii generale

Funcţia rijndael are trei parametri. Aceştia sunt: plaintext, un vector de 16 octeţi conţinând datele de intrare, ciphertext, un vector de 16 octeţi în care va fi introdus rezultatul cifrat, şi key, cheia de 16 octeţi. Pe durata calculelor, starea curentă a datelor e păstrată într-un vector de octeţi, state, a cărui mărime este NROWS × NCOLS. Pentru blocuri de 128 de octeţi, acest vector este de 4 × 4 octeţi. Întregul bloc de date de 128 de biţi poate fi stocat în 16 octeţi.

Vectorul state este iniţializat cu textul clar şi este modificat la fiecare pas al calculului. În anumiţi paşi, este realizată substituţia octet-cu-octet. În alţii, octeţii sunt permutaţi în interiorul vectorului. Sunt folosite şi alte transformări. La sfârşit, conţinutul lui state reprezintă textul cifrat.

Codul începe prin expandarea cheii în 11 vectori de aceeaşi lungime cu starea. Aceştia sunt memoraţi în rk, care este un vector de structuri, fiecare structură conţinând un vector stare. Unul dintre aceştia va fi folosit la începutul calculului şi ceilalţi 10 vor fi folosiţi în timpul celor 10 runde, câte unul în fiecare rundă. Calculul cheilor de runde din cheia de criptare este prea complicat pentru a

fi prezentat aici. Este de ajuns să spunem că cheile de runde sunt produse prin rotiri repetate şi aplicarea de operaţii XOR asupra unor grupuri de biţi din cheie.

Următorul pas este acela de a copia textul clar în vectorul state astfel încât să poată fi procesat pe perioada rundelor. Acesta este copiat în ordinea coloanelor, cu primii patru octeţi în coloana 0, următorii patru octeţi în coloana 1 şi aşa mai departe. Atât coloanele cât şi liniile sunt numerotate pornind de la 0, deşi rundele sunt numerotate pornind de la 1.

Mai este un pas înainte de a începe calculul principal: rk[0] este combinat prin XOR în state, octet cu octet. Cu alte cuvinte fiecare octet din cei 16 aflaţi în state este înlocuit cu rezultatul aplicării operaţiei XOR asupra sa şi asupra octetului corespunzător din rk[0].

Şi acum urmează partea cea mai interesantă. Bucla execută 10 iteraţii, câte una pe rundă, transformând state la fiecare iteraţie. Fiecare rundă constă în patru paşi. Pasul 1 realizează o substituţie octet-cu-octet asupra lui state. Pe rând, fiecare octet este folosit ca index într-o cutie S pentru a-l înlocui valoarea prin conţinutul corespunzător din acea cutie S. Acest pas este un cifru de substituţie monoalfabetică directă. Spre deosebire de DES, care are mai multe cutii S, Rijndaek are doar o cutie S.

Pasul 2 roteşte la stânga fiecare din cele 4 rânduri. Rândul 0 este rotit cu 0 octeţi (nu e schimbat), rândul 1 este rotit cu 1 octet, rândul 2 este rotit cu 2 octeţi şi rândul 3 este rotit cu 3 octeţi. Acest pas difuzează conţinutul datelor curente în jurul blocului.

Pasul 3 amestecă fiecare coloană independent de celelalte. Această amestecare este realizată prin înmulţire de matrice, în care noua coloană este produsul dintre vechea coloană şi o matrice constantă, multiplicarea fiind realizată folosind câmpul finit Galois, GF(28). Deşi acest lucru poate părea complicat, există un algoritm care permite fiecărui element al noii coloane să fie calculat folosind două căutări în tabele şi trei operaţii XOR.

În fine, pasul 4 aplică operaţia XOR pentru cheia din runda curentă şi vectorul stare.

Deoarece fiecare pas e reversibil, decriptarea poate fi realizată prin rularea algoritmului de la coadă la cap. Oricum, este posibilă şi o schemă prin care decriptarea poate fi realizată prin rularea algoritmului de criptare nemodificat, dar folosind tabele diferite.

Algoritmul a fost proiectat nu doar pentru o securitate foarte solidă, dar şi pentru o viteză foarte mare. O bună implementare software pe o maşină la 2 GHz ar trebui să atingă o rată de criptare de 700 Mbps, ceea ce este suficient de rapid pentru a criptarea peste 100 de fişiere video MPEG-2 în timp real. Implementările hardware sunt chiar mai rapide.

33. Algoritmi cu cheie publica - RSA

Datorită posibilelor avantaje ale criptografiei cu chei publice, mulţi cercetători au lucrat din greu la acest subiect şi au fost deja publicaţi câţiva algoritmi. O metodă bună a fost descoperită de un grup de la MIT (Rivest et. al, 1978). Ea este cunoscută prin iniţialele numelor celor trei descoperitori (Rivest, Shamir, Adelman): **RSA**. Metoda a supravieţuit tuturor încercărilor de a o sparge timp de mai mult de un sfert de secol şi este considerată foarte puternică. Multe aplicaţii de securitate se bazează pe ea. Dezavantajul major al acesteia este că necesită chei de cel puţin 1024 de biţi pentru o securitate bună (spre deosebire de 128 biţi pentru algoritmii cu cheie simetrică), ceea ce o face destul de lentă.

Metoda RSA este bazată pe câteva principii din teoria numerelor. Vom rezuma mai jos modul în care se foloseşte această metodă:

1. Se aleg două numere prime, p şi q, (de obicei de 1024 biţi).

2. Se calculează $n = p \times q$ şi $z = (p - 1) \times (q-1)$.

3. Se alege un număr relativ prim cu z şi este notat cu d.

4. Se găseşte e astfel încât $e \times d = 1 \bmod z$.

Cu aceşti parametri calculaţi în avans, suntem gata să începem criptarea. Împărţim textul clar (privit ca şir de biţi) în blocuri, astfel încât fiecare mesaj de text clar, P, să intre în intervalul $0 \leq P < n$. Aceasta poate fi făcută grupând textul clar în blocuri de câte k biţi, unde k este cel mai mare număr întreg pentru care inegalitatea $2k < n$ este adevărată.

Pentru a cripta mesajul P, se calculează $C = Pe \pmod n$. Pentru a decripta C, se calculează $P = Cd \pmod n$. Se poate demonstra că pentru toţi P din intervalul specificat, criptarea şi decriptarea sunt funcţii inverse una alteia. Pentru a realiza criptarea este nevoie de e şi n. Pentru a realiza decriptarea este nevoie de d şi n. De aceea, cheia publică constă din perechea (e, n) iar cheia private din perechea (d, n).

Securitatea metodei este bazată pe dificultatea factorizării numerelor mari. Dacă un criptanalist ar putea factoriza numărul n (public cunoscut), el ar putea găsi p şi q, iar din acestea pe z. Cu z şi e cunoscuţi, criptanalistul îl poate calcula pe d folosind algoritmul lui Euclid. Din fericire, matematicienii au încercat să factorizeze numere mari de cel puţin 300 de ani şi experienţa acumulată sugerează că aceasta este o problemă mai mult decât dificilă.

Trebuie subliniat că folosirea RSA în modul descris este similară folosirii unui algoritm simetric în modul ECB - blocuri de intrare identice conduc la blocuri de ieşire identice. De aceea este necesară o anumită formă de înlănţuire pentru criptarea datelor. Totuşi, în practică, cele mai multe sisteme bazate pe RSA folosesc criptografia cu cheie publică în principal pentru distribuirea cheilor de sesiune de unică folosinţă utilizate pentru un algoritm simetric ca AES sau triplu DES. RSA este prea lent pentru a cripta eficient volume mari de date, dar este folosit mult la distribuţia de chei.