

P.C.

1. Nivelurile OSI (Open Systems Interconnection)

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

Nivelul **legaturii de date** tratează erorile de transmisie produse la nivelul fizic, realizând o comunicare corectă între două noduri adiacente. Mecanismul utilizat în acest scop este împartirea sirului de biți în cadre, cărora le sunt adăugate informații de control (coduri de verificare, numere de secvență etc).

Cadrele sunt transmise individual putând fi verificate și confirmate de către receptor. Alte funcții ale nivelului se referă la: controlul fluxului datelor (astfel încât transmitatorul să nu furnizeze date mai rapid decât le poate accepta receptorul) și gestiunea legaturii (stabilirea legaturii, controlul schimbului de date și desființarea legaturii).

Nivelul **rețea** asigură dirijarea unităților de date între nodurile sursă și destinatar, trecând eventual prin noduri intermediare. Decizia este luată astfel încât să nu existe în același timp legături supraincercate și legături neutilizate, evitându-se deci congestiunea rețelei. O altă funcție importantă a nivelului rețea este cea de interconectare a rețelelor cu arhitecturi diferite.

Nivelul **transport** realizează o comunicare sigură între două calculatoare gazdă, detectând și corectând erorile pe care nivelul rețea nu le tratează. El este important nu numai prin poziția pe care o ocupă, la mijlocul ierarhiei de nivele, dar și prin funcția sa, de a furniza nivelelor superioare o interfață independentă

de tipul rețelei utilizate. Funcțiile sunt realizate de entități situate în sistemele gazda, fără concursul altor entități similare intermediare, motiv pentru care se numesc funcții "capăt la capăt" (capetele fiind cele două entități de transport corespondente).

Nivelul **sesiune** oferă toate serviciile pentru gestiunea jetoanelor, lăsând la latitudinea utilizatorilor semnificațiile asociate acestora. Ele au la bază utilizarea unui mesaj special, numit jeton (token), care poate fi trecut de la un utilizator la altul și a cărui posesie oferă detinatorului anumite privilegii: de a transmite date, de a stabili puncte de sincronizare, de a stabili începutul unei activități etc.

Nivelul **prezentare** realizează transformări ale reprezentării datelor, astfel încât să se păstreze semnificația lor, rezolvându-se totodată diferențele de sintaxă. Funcțiile principale se referă la codificarea standard a datelor transmise între calculatoare cu convenții de reprezentare diferite, la comprimarea textelor, precum și la criptarea/decriptarea acestora în vederea protecției și securității lor.

Nivelul **aplicatie**, cel mai înalt nivel al arhitecturii, are rolul de fereastră de comunicare prin care se fac toate schimburile de date între utilizatori.

Fiind nivelul care furnizează servicii direct aplicațiilor, el cuprinde toate funcțiile pe care acestea le pot solicita:

- identificarea utilizatorilor cooperanți, autentificarea lor și determinarea disponibilității acestora,
- stabilirea calității serviciului,
- sincronizarea aplicațiilor cooperante și selectarea modului de dialog,
- stabilirea responsabilităților pentru tratarea erorilor,
- identificarea constrângerilor asupra sintaxei datelor,
- transferul informației.

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 posibilități 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 mașina sursă trimite cadre independente către mașina destinație, fără ca mașina destinație să trebuiască să confirme primirea lor. În acest caz, nu sunt necesare stabilirea și desființarea 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 privește siguranța 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, emițătorul știe 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 cerință. Nivelul rețea poate întotdeauna să trimită un mesaj și să aștepte să fie confirmat. Dacă confirmarea nu apare în timp util, atunci emițătorul poate retrimite întregul mesaj. Problema cu această strategie este aceea că, de obicei, cadrele au o lungime maximă impusă de hardware, iar pachetele nivelului rețea nu au aceasta limitare. Dacă pachetul mediu este spart în, să zicem, 10 cadre și 20% din totalul cadrelor sunt pierdute, transmiterea acestuia poate lua

foarte mult timp. În cazul în care cadrele individuale sunt confirmate și 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 nesiguranței acestora.

Cel mai sofisticat serviciu pe care nivelul legătură de date îl pune la dispoziția nivelului rețea este **serviciul orientat-conexiune**. În cazul acestui serviciu, mașinile sursă și destinație stabilesc o conexiune înainte de a transfera date. Fiecare cadru trimis pe conexiune este numerotat și nivelul legătură de date garantează că fiecare cadru trimis este într-adevăr recepționat. Mai mult, garantează că fiecare cadru este recepționat exact o dată și toate cadrele sunt recepționate 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 și, prin urmare, recepționat de mai multe ori. Spre deosebire de acesta, serviciul orientat conexiune furnizează proceselor de la nivelul rețea echivalentul unui flux de biți sigur.

Atunci când este utilizat serviciul orientat conexiune, transferurile au trei faze distincte. În prima fază este stabilită conexiunea, ambele părți inițializând variabile și contoare, utilizate pentru a ține evidența cadrelor care au fost recepționate și a celor care nu au fost. În a doua fază, sunt transmise unul sau mai multe cadre. În a treia și ultima fază, conexiunea este desființată, eliberând variabilele, tampoanele și alte resurse utilizate la menținerea conexiunii.

3. Incadrarea la nivelul legatura de date

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

1. Numărarea caracterelor.

2. Indicatori cu inserare de octeți.

3. Indicatori de început și de sfârșit, cu inserare de biți.

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 destinație primește contorul de caractere, știe câte caractere urmează și unde este sfârșitul cadrului. Problema cu acest algoritm este că valoarea contorului poate fi alterată de erori de transmisie. Chiar dacă suma de control este incorectă și destinația știe 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 destinația nu știe 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 și se termină cu o secvență specială de octeți. Inițial, octeții ce indicau începutul, respectiv sfârșitul erau diferiți, dar în ultimii ani s-a trecut la utilizarea unui singur octet, numit octet indicator, atât ca indicator de început, cât și de sfârșit. În acest fel, dacă receptorul pierde sincronizarea, acesta poate căuta octetul indicator pentru a găsi sfârșitul cadrului. Doi octeți indicatori consecutivi indică sfârșitul unui cadru și 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 situație interferează cu procesul de încadrare. O cale de rezolvare a acestei probleme este ca nivelul legătură de date al emițătorului să insereze un octet special (ESC) înaintea fiecărei apariții „accidentale” a indicatorului în date. Nivelul legătură de date al receptorului va elimina acest octet special înainte de a pasa datele nivelului rețea. Această tehnică poartă numele de inserare de octeți (eng.: byte stuffing) sau inserare de caractere (eng.: character stuffing) . Deci, un octet indicator utilizat pentru încadrare poate fi diferențiat de unul prezent în date prin faptul că este sau nu precedat de un octet special.

Bineînțeles, 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 secvență specială, iar un octet special dublu indica faptul ca un singur octet de acest tip a apărut în cadrul datelor. În toate cazurile, secvența de octeți obținută după eliminarea octeților inserați este exact secvența originală.

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

A treia metoda de încadrare permite cadrelor de date să conțină un număr arbitrar de biți și permite coduri de caractere cu un număr arbitrar de biți per caracter. Funcționează astfel: fiecare cadru începe și se termină cu un șablon special pe biți, 01111110, numit octet indicator (flag). De fiecare dată când nivelul legătură de date al emițătorului identifică cinci de unu consecutivi în date, inserează automat un bit 0 în șirul de biți de rezultați. Această inserare de biți (bit stuffing) este similară inserării de caractere, în care un octet escape este inserat în șirul de caractere de ieșire, înainte de fiecare octet indicator din date.

Atunci când receptorul primește o succesiune de cinci biți 1, urmați de un bit 0, extrage automat (adică șterge) bitul 0. La fel ca și inserarea de caractere, care este

complet transparentă pentru nivelul rețea din ambele calculatoare, așa este și inserarea de biți. Dacă datele utilizator conțin șablonul indicator, 01111110, acest indicator este transmis ca 011111010, dar în memoria receptorului este păstrat ca 01111110.

4. Coduri corectoare de erori

Proiectanții de rețele au dezvoltat două strategii de bază pentru tratarea erorilor. O modalitate este ca pe lângă fiecare bloc de date trimis să se includă suficientă informație redundantă pentru ca receptorul să poată deduce care a fost caracterul transmis. O altă soluție este să se includă suficientă redundanță pentru a permite receptorului să constate că a apărut o eroare, dar nu care este eroarea, și 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 în situații diferite. Pe canale cu siguranță mare, cum ar fi fibra optică, este mai eficient să utilizăm un cod detector de erori și să retransmitem blocul în care s-au detectat erori. În cazul canalelor de comunicație fără fir, este indicat să adăugăm destulă informație redundantă fiecărui bloc, în loc să ne bazăm pe retransmisie, care poate să fie la rândul său afectată de erori.

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

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

Numărul de poziții binare în care două cuvinte de cod diferă se numește distanța Hamming (Hamming, 1950). Semnificația sa este că dacă două cuvinte de cod sunt despărțite de o distanță Hamming d , sunt necesare d erori de un singur bit pentru a-l converti pe unul în celălalt.

În multe aplicații de transmisie de date, toate cele 2^m mesaje de date posibile sunt corecte, dar, datorită modului în care sunt calculați biții de control, nu sunt utilizate toate cele 2^n cuvinte de cod posibile. Dat fiind algoritmul pentru calculul biților de control, este posibil să construim o listă completă de cuvinte de cod permise și din această listă să găsim cele două cuvinte de cod a căror distanță Hamming este minimă. Această distanță este distanța Hamming a codului complet.

Proprietățile detectoare și corectoare de erori ale unui cod depind de distanța sa Hamming. Pentru a detecta d erori, este nevoie de un cod cu distanță $d + 1$, 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 distanță $2d + 1$, deoarece în acest mod cuvintele de cod corecte sunt atât de distanțate, încât, chiar cu d modificări, cuvântul de cod original este totuși mai apropiat decât alte cuvinte de cod și va fi unic determinat.

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

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

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

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

Când sosește un cuvânt de cod, receptorul inițializează un contor la 0. Acesta examinează apoi fiecare bit de control, k ($k = 1, 2, 4, 8, \dots$) pentru a vedea dacă are paritatea corectă. Dacă nu, adaugă k la contor. Dacă, după ce au fost examinați toți biții de control, contorul este 0 (adică, dacă toți biții au fost corecți), cuvântul de cod este acceptat ca valid. Dacă valoarea contorului este nenulă, ea reprezintă numărul bitului incorect. De exemplu, dacă biții de control 1, 2 și 8 sunt eronați, atunci bitul inversat este 11, deoarece este singurul verificat de biții 1, 2 și 8.

Codurile Hamming pot corecta numai erori singulare. Totuși, există un artificiu care poate fi utilizat pentru a permite codurilor Hamming să corecteze erorile în rafală. O secvență 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 trimiși toți cei k biți, este transmisă a doua coloană și așa mai departe, așa cum se arată în 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, așa încât întregul bloc poate fi refăcut. Această metodă utilizează kr biți de control pentru a face blocuri de km biți 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 comparație cu firele de cupru sau fibra optică. Fără coduri detectoare de erori, comunicația ar fi greu de realizat. În cazul firelor de cupru sau a fibrei optice rata erorilor este mult mai mică, așa că detectarea erorilor și 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 și 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. Șansele pot fi îmbunătățite considerabil dacă fiecare bloc transmis este privit ca o matrice dreptunghiulară de n biți lățime și k biți înălțime, după cum am arătat mai sus. Pentru fiecare coloană este calculat un bit de paritate, care este adăugat într-o nouă linie de la sfârșitul matricei. Matricea este apoi transmisă linie cu linie. La sosirea blocului, receptorul verifică toți biții de paritate. Dacă oricare din ei este greșit, va cere o retransmisie a blocului. Retransmisii succesive sunt cerute dacă este nevoie, până când întregul bloc este recepționat 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 totuși nedetectată dacă primul și ultimul bit sunt inversați, iar toți ceilalți biți sunt corecți (o eroare în rafală nu înseamnă că toți biții sunt greșiți, ci că cel puțin primul și ultimul sunt greșiți). 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^{-n} .

Cu toate că schema de mai sus poate fi uneori adecvată, în practică este larg utilizată o altă metodă: **codul polinomial** (cunoscut și sub numele de cod cu redundanță ciclică, eng.: **cyclic redundancy code**). Codurile polinomiale sunt bazate pe tratarea șirurilor de biți ca reprezentări de polinoame cu coeficienți 0 și 1. Un cadru de k biți este văzut ca o listă de coeficienți pentru un polinom cu k termeni, de la x^{k-1} la x^0 . Se spune că un astfel de polinom este de gradul $k-1$. Bitul cel mai semnificativ (cel mai din stânga) este coeficientul lui x^{k-1} ; următorul bit este coeficientul lui x^{k-2} ș.a.m.d. De exemplu, 110001 are șase biți și ei reprezintă un polinom cu șase termeni cu coeficienții 1, 1, 0, 0, 0 și 1: $x^5+x^4+x^0$. Aritmetica polinomială este de tip modulo 2, în conformitate cu regulile teoriei algebrei. Nu există transport la adunare și nici împrumut la scădere. Atât adunările cât și scăderile sunt identice cu SAU EXCLUSIV. Împărțirea lungă este făcută ca în binar cu excepția faptului că scăderea este realizată modulo 2, ca mai sus. Despre un împărțitor se spune că „intră” într-un deîmpărțit dacă deîmpărțitul are tot atâția biți ca împărțitorul.

Atunci când este utilizată metoda codului polinomial, emițătorul și receptorul se pun de acord în avans asupra unui polinom generator $G(x)$. Atât bitul cel mai semnificativ cât și cel mai puțin semnificativ trebuie să fie 1. Pentru a calcula suma de control pentru un cadru cu m biți, corespunzător polinomului $M(x)$, cadrul trebuie să fie mai lung decât polinomul generator. Ideea este de a adăuga o sumă de control la sfârșitul 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 obține 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 biți 0 la capătul mai puțin semnificativ al cadrului, așa încât acesta va conține acum $n+r$ biți și va corespunde polinomului $x^r M(x)$.
2. Se împarte șirul de biți ce corespund lui $G(x)$ într-un șir de biți corespunzând lui $x^r M(x)$, utilizând împărțirea modulo 2.

3. Se scade restul (care are întotdeauna r sau mai puțini biți) din șirul de biți 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, așa încât în loc să sosească șirul de biți 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 biți 1, aceasta înseamnă că au apărut k erori de un singur bit.

La recepția 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, așa încât rezultatul calculului este pur și 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 legăturii de date

***Protocolul Start-Stop**

-transmițătorul trimite un nou cadru numai după recepția confirmării pozitive a cadrului precedent.

***Protocol simplex fără restricții.**

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

- utilizatorul A vrea să transmită date lui B folosind o legătură sigură, simplex;
- A reprezintă o sursă inepuizabilă de date, astfel încât transmițătorul nu trebuie să aștepte niciodată la preluarea datelor pentru transmisie;
- B reprezintă un consumator ideal, care preia imediat datele de la receptor ori de câte ori acesta i le dă;
- canalul fizic de comunicație 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 reacție a receptorului către transmițător, permițându-i

acestui din urmă să transmită următorul cadru. Reacția are forma unui cadru fictiv.

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

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

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

Pentru justificare, fie cadrele succesive m , $m+1$, $m+2$ cu numerele de secvență respectiv 0, 1 și 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) și primește un nou cadru cu număr de secvență 0, el nu poate fi decât m (0) și în consecință îl ignoră.

***Protocol cu fereastra glisanta**

-nu așteaptă 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, diferențiabile printr-un câmp din antet (fel).

Aceste elemente sînt cuprinse în protocoalele cu fereastră glisantă. Esența lor este prezentată în continuare. Transmițătorul menține o listă cu numerele de secvență ale cadrelor transmise dar neconfirmate, alcătuiind fereastra transmițătorului. Cînd trebuie transmis un cadru, i se atașează numărul de secvență următor (fereastra se mărește). Cînd se recepționează o confirmare, marginea inferioară a ferestrei se deplasează cu o unitate. Cadrele se păstrează de transmițător pînă la confirmarea lor.

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

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

Ambele ferestre fiind unitare, transmițătorul trebuie să aștepte confirmarea înainte de a transmite un nou cadru (la fel ca în cazul precedent).

Deoarece algoritmi transmițătorului și receptorului sînt aproape identici, dăm o singură descriere pentru ambele stații. Fiecare stație realizează ciclic următoarele operații:

- recepția unui cadru,
- prelucrarea șirului de cadre recepționate,
- prelucrarea șirului de cadre transmise,
- transmiterea sau retransmiterea unui cadru împreună cu confirmarea cadrului recepționat corect.

O situație necorespunzătoare apare dacă ambele stații transmit un cadru inițial. Pentru a corecta acest neajuns, o stație trebuie să o preceadă pe cealaltă prin acțiunile sale. Pentru aceasta, doar una trebuie să execute transmisia unui cadru în afara buclei principale.

Cîmpul conf conține numărul ultimului cadru recepționat corect. Dacă el corespunde cadrului aflat în transfer, transmițătorul preia alt pachet pentru transmisie. Altfel, se transmite același pachet.

Cîmpul secv conține numărul de secvență al cadrului. Dacă el coincide cu cel așteptat, se transmite un pachet nivelului rețea. Stațiile nu se blochează și 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 așteptare (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 CadruAșteptat pe 1. Celelalte cadre sînt ignorate, avînd secv=0. Mai mult, duplicatele au conf = 1, în timp ce B așteaptă conf = 0 pentru a scoate un nou pachet de la rețea. 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, așteptarea confirmării fiecărui cadru înainte de transmiterea următorului poate conduce la o eficiență scăzută. Astfel, la un canal cu o întîrziere de propagare dus-întors de 500 msec și o viteză de 50 Kbps, transmiterea unui cadru de 1000 de biți durează efectiv doar 20 de milisecunde din cele 520 necesare ciclului complet de transmitere și recepție a confirmării.

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

O cale este de ignorare a acestora, fără transmitere de confirmări. Ea corespunde unei ferestre de recepție unitare. În acest caz, transmițătorul va fi obligat să retransmită toate cadrele începînd cu cel eronat. A doua cale este memorarea cadrelor recepționate 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 transmițătorului poate fi de MaxSecv cadre, deși există $\text{MaxSecv}+1$ numere de secvență distincte. Justificăm această restricție prin următorul scenariu, în care presupunem că $\text{MaxSecv}=7$:

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

În acest protocol renunțăm la presupunerea că utilizatorul este o sursă ideală de pachete și includem mecanismul de reglare a fluxului de date: cînd utilizatorul are un pachet de transmis el provoacă un eveniment "ReteaPregatita"; cînd o stație are MaxSecv cadre neconfirmate, ea interzice rețelei să-i transmită alte pachete prin procedura "DezactivRetea"; cînd numărul de pachete scade, ea anulează restricția prin funcția "ActivRetea".

Transmițătorul păstrează cadrele neconfirmate pentru o eventuală retransmisie. Confirmarea cadrului n provoacă automat confirmarea cadrelor $n-1$, $n-2$,... anterioare, ceea ce compensează eventualele pierderi ale confirmărilor acestor cadre. Deoarece mai multe cadre își asteaptă confirmarea, se utilizează ceasuri separate pentru diferite mesaje. Fiecare ceas este pornit la transmiterea unui cadru și este oprit la recepția confirmării sale. La producerea unui eveniment Timeout, toate cadrele aflate în memoria tampon a transmițătorului sînt retrimise.

***Protocol cu retransmitere selectiva.**

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

Fereastra receptorului nu poate fi egală cu cea a transmițătorului așa cum rezultă din următorul scenariu, pentru $\text{MaxSecv} = 7$:

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

6. Transmițătorul află că toate cadrele sale au fost transmise corect și trimite cadrele 7, 0, 1, 2, 3, 4, 5

7. Receptorul acceptă cadrele, cu excepția lui 0, pentru care are deja un cadru recepționat. Ca urmare, ignoră acest cadru, luînd în locul lui duplicatul cadrului 0 anterior.

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

Fereastra receptorului determină și numărul de tampoane de recepție, ca de altfel și numărul de ceasuri necesare. Cînd canalul este puternic încărcat, confirmările cadrelor transmise de la stația A la stația 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 recepția unui cadru corect, ci după un interval de timp convenabil stabilit și 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 îmbunătăți eficiența 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 același cadru așteptat.

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

În această secțiune vom examina un grup de protocoale strâns legate, puțin 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 și ISO pentru acceptare ca standard SUA și, respectiv, internațional. ANSI a modificat protocolul, astfel încât acesta a devenit ADCCP (Advanced Data Communication Control Procedure, rom.: procedură de control avansat al comunicațiilor de date), iar ISO l-a modificat și a produs HDLC (High-level Data Link Control, rom.: control de nivel înalt al legăturii de date). CCITT a adoptat și modificat HDLC pentru al său

LAP (Link Access Procedure, rom.: procedură de acces la legătură) care este parte a standardului pentru interfața de rețea 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 privește standardele este că sunt multe, dintre care poți alege. În plus, dacă nu îți place nici unul dintre ele, poți aștepta modelul care va apărea anul viitor.

Aceste protocoale se bazează pe aceleași principii. Toate sunt orientate pe biți și folosesc inserarea de biți pentru transparența datelor. Ele diferă doar în puncte minore, și totuși supărătoare. Discuția care urmează, despre protocoalele orientate pe biți, intenționează a fi o introducere generală. Pentru detaliile specifice fiecărui protocol, consultați definiția corespunzătoare.

Biți	8	8	8	≥ 0	16	8
	0 1 1 1 1 1 0	Adresă	Control	Date	Sumă de control	0 1 1 1 1 1 0

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

Toate protocoalele orientate pe biți folosesc structura de cadru prezentată în fig. 3-24. Câmpul Adresă este primul ca importanță 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 secvență, confirmări și alte scopuri, după cum se va arăta în continuare.

Câmpul Date poate conține informații arbitrare. Poate avea lungime arbitrară, cu toate că eficiența sumei de control scade odată cu creșterea lungimii cadrului, datorită creșterii probabilității de apariție a erorilor în rafală.

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

Cadrul este delimitat cu o altă secvență indicator (01111110). Pe liniile punct-la-punct inactive secvențele indicator sunt transmise continuu. Un cadru minim conține trei câmpuri și are în total 32 de biți, excluzând indicatorii de la capete.

Există trei tipuri de cadre: Informație, Supervizor și Nenumerotat. Conținutul câmpului Control pentru fiecare dintre aceste trei tipuri este prezentat în fig. 3-25. Acest protocol folosește o fereastră glisantă, cu un număr de secvență reprezentat pe 3 biți. În fereastră pot fi păstrate, la un moment dat, până la șapte cadre neconfirmate.

Câmpul Secvență din fig. 3-25(a) este numărul de secvență al cadrului. Câmpul Următor este o confirmare atașată. Oricum, toate protocoalele aderă la convenția că, în loc să atașeze numărul ultimului cadru recepționat corect, să folosească numărul primului cadru nerecepționat (adică următorul cadru așteptat). Opțiunea pentru ultimul cadru primit sau următorul cadru recepționat este arbitrară; nu are importanță ce convenție este utilizată, dacă este folosită cu consecvență.



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 excepția 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 forța cealaltă mașină să trimită imediat un cadru Supervizor, în loc să aștepte fluxul invers la care să se atașeze informația despre fereastră. Bitul are de asemenea câteva utilizări minore referitoare la cadrele nenumerate.

Numeroasele tipuri de cadre Supervizor sunt diferențiate prin câmpul Tip.

Tipul 0 este un cadru de confirmare (numit oficial RECEIVE READY) folosit pentru a indica următorul cadru așteptat. Cadrul este folosit atunci când nu există flux invers care să poată fi folosit pentru atașare.

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

Tipul 2 este RECEIVE NOT READY. El confirmă toate cadrele, cu excepția lui Următor, exact ca RECEIVE READY, dar spune transmițătorului să oprească transmisia. RECEIVE NOT READY este destinat să semnaleze anumite probleme temporare apărute la receptor, cum ar fi lipsa zonelor tampon, și 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 transmițătorului este jumătate sau mai puțin din dimensiunea spațiului secvenței. Astfel, dacă receptorul dorește să păstreze cadre care erau în afara secvenței pentru posibila folosire ulterioară, el poate să forțeze retransmiterea oricărui cadru, folosind SELECTIVE REJECT. HDLC și ADCCP permit acest tip de cadru, dar SDLC și LAPB nu îl permit (adică nu există Selective Reject) și 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 și pentru transportul datelor atunci când se recurge la un serviciu nesigur, neorientat pe conexiune. Diversele tipuri de protocoale orientate pe biți diferă considerabil aici, spre deosebire de celelalte două tipuri, unde erau aproape identice. Pentru a indica tipul cadrului sunt disponibili cinci biți, dar nu sunt folosite toate cele 32 de posibilități.

Toate protocoalele furnizează o comandă, DISC (DISConnect), care permite ca o mașină să anunțe că se va opri. De asemenea există o comandă ce permite ca o mașină, care tocmai s-a reconectat, să-și anunțe prezența și să forțeze resetarea tuturor numerelor de secvență la zero. Această comandă poartă numele de SNRM (Set Normal Response Mode - stabilește 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 comunicația 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 și LAPB au o comandă suplimentară, SABM

(Set Asynchronous Balanced Mode - stabilește modul asincron echilibrat), care resetează linia și declară ambii parteneri ca fiind egali. De asemenea, aceste protocoale au comenzile SABME și SNRME, care sunt identice cu SABM și, respectiv, SNRM, cu excepția faptului că ele permit folosirea unui format extins pentru cadru, care utilizează numere de secvență pe 7 biți în locul unora pe 3 biți.

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 biți, un cadru de control nepermis, confirmarea unui cadru care a fost în afara ferestrei etc. Cadrele FRMR conțin un câmp de date de 24 de biți care arată ceea ce a fost eronat la cadrul respectiv. Datele includ câmpul de control al cadrului eronat, parametrii ferestrei și o colecție de biți folosiți pentru a semnala erori specifice.

Cadrele de control pot fi pierdute sau deteriorate ca și cadrele de date, de aceea și 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 ambiguități asupra cadrului care este confirmat. Cadrele de control rămase sunt folosite pentru inițializare, interogare și raportarea stării. Există, de asemenea, un cadru de control care poate conține informații arbitrare, UI (Unnumbered Information).

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

8. Servicii furnizate de nivelul rețea nivelului transport

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

1. Serviciile trebuie să fie independente de tehnologia ruterului.
2. Nivelul transport trebuie să fie independent de numărul, tipul și topologia ruterelor existente.
3. Adresele de rețea disponibile la nivelul transport trebuie să folosească o schemă de numerotare uniformă, chiar în cadrul rețelelor LAN și WAN.

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

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

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

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

9. Subrețea datagrama și subrețea cu circuite virtuale

Dacă este oferit un serviciu neorientat pe conexiune, atunci pachetele sunt trimise în subrețea individual și dirijate independent de celelalte. Nu este necesară nici o inițializare prealabilă. În acest context, pachetele sunt numite frecvent **datagrame (datagrams)** (prin analogie cu telegramele), iar subrețeaua este numită **subrețea datagramă (datagram subnet)**. Dacă este folosit serviciul orientat conexiune, atunci, înainte de a trimite pachete de date, trebuie stabilită o cale de la ruterul sursă la ruterul destinație. Această conexiune este numită **VC**

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

Să vedem cum funcționează o **subrețea datagramă**. Să presupunem că procesul P1 din fig. 5-2 are un mesaj lung pentru procesul P2. El transmite mesajul nivelului transport, cu instrucțiunile de livrare către procesul P2 aflat pe calculatorul gazdă H2. Codul nivelului transport rulează pe calculatorul gazdă H1, de obicei în cadrul sistemului de operare. Acesta inserează la începutul mesajului un antet corespunzător nivelului transport și transferă rezultatul nivelului rețea, probabil o altă procedură din cadrul sistemului de operare.

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

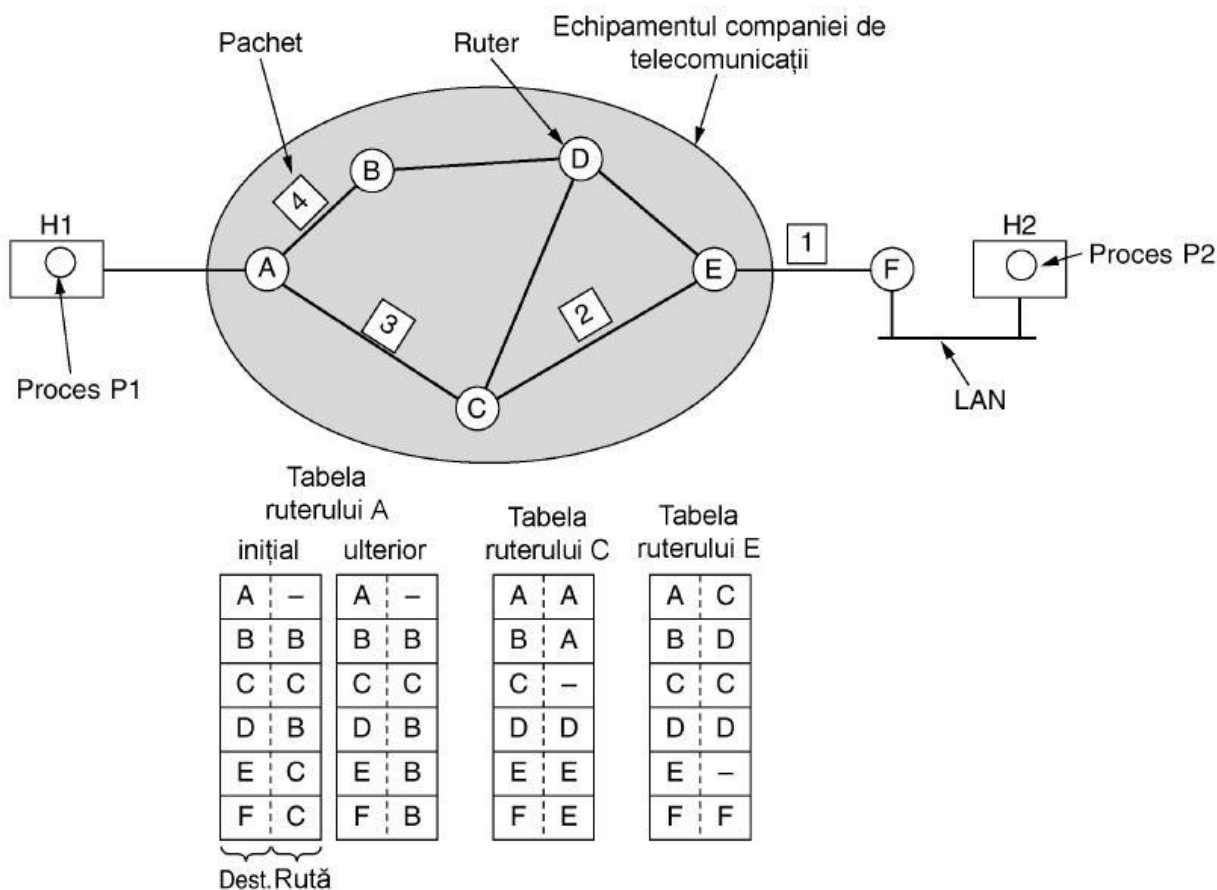


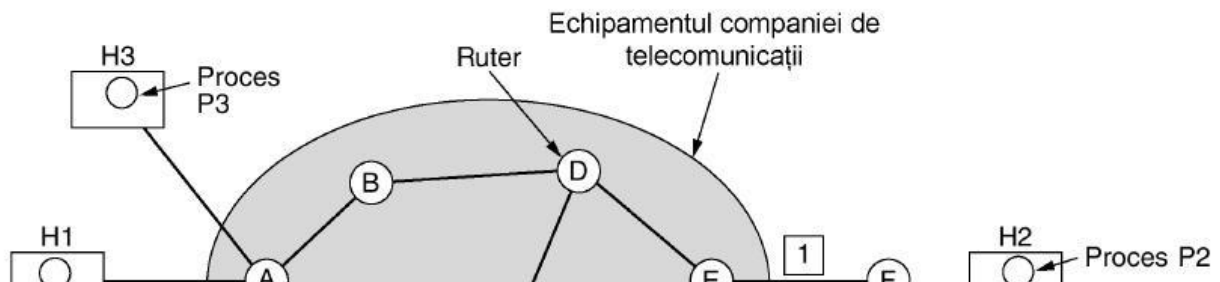
Fig. 5-2. Dirijarea într-o subrețea datagramă.

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

Totuși, ceva diferit s-a întâmplat cu pachetul 4. Când a ajuns la A a fost trimis către ruterul B, chiar dacă și el este destinat tot lui F. Dintr-un motiv oarecare, A a decis să trimită pachetul 4 pe o rută diferită de cea urmată de primele trei. Poate că a aflat despre o congestie undeva pe calea ACE și și-a actualizat tabela de rutare, așa cum apare sub eticheta „mai târziu”. Algoritmul ce administrează tabelele și ia deciziile de rutare se numește algoritm de rutare (routing algorithm).

Pentru **serviciile orientate conexiune**, avem nevoie de o subrețea cu circuite

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



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

Acum să vedem ce se întâmplă dacă H3 vrea, de asemenea, să stabilească o conexiune cu H2. Alege identificatorul de conexiune 1 (deoarece inițializează conexiunea și aceasta este singura conexiune) și indică subrețelei să stabilească circuitul virtual. Aceasta conduce la a doua linie din tabele. Observați că apare un conflict deoarece deși A poate distinge ușor pachetele conexiunii 1 de la H1 de pachetele conexiunii 1 de la H3, C nu poate face asta. Din acest motiv, A asociază un identificator de conexiune diferit pentru traficul de ieșire al celei de a doua conexiuni. Pentru evitarea conflictelor de acest gen ruterele trebuie să poată înlocui identificatorii de conexiune în pachetele care pleacă. În unele contexte, aceasta se numește comutarea etichetelor (label switching).

Comparație între subrețele cu circuite virtuale și subrețele 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. Comparatie între subrețele datagramă și subrețele cu circuite virtuale.

10. Algoritmi de dirijare

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

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

dirijare linia de ieșire folosită pentru el. Acesta este procesul de retransmitere (forwarding). Celălalt proces se ocupă de completarea și actualizarea tabelului 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 proprietăți: corectitudine, simplitate, robustețe, stabilitate, echitate, optimalitate.

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

Algoritmii adaptivi (adaptive algorithms), prin contrast, își modifică deciziile de dirijare pentru a reflecta modificările de topologie și de multe ori și pe cele de trafic. Algoritmii adaptivi diferă prin locul de unde își iau informația (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) și prin metrica folosită pentru optimizare (de exemplu distanța, numărul de salturi sau timpul estimat pentru tranzit).

Algoritmi statici:

- **Algoritmul lui Dijkstra:** Fiecare nod este etichetat (în paranteze) cu distanța de la nodul sursă până la el, de-a lungul celei mai bune căi cunoscute. Inițial nu se cunoaște nici o cale, așa că toate nodurile vor fi etichetate cu infinit. Pe măsură ce se execută algoritmul și se găsesc noi căi, etichetele se pot schimba, reflectând căi mai bune. O etichetă poate fi fie temporară, fie permanentă. Inițial 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 și nu se mai schimbă ulterior.

- **Inundarea:** fiecare pachet recepționat este trimis mai departe pe fiecare linie de ieșire, cu excepția 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 și care face ca pachetul să fie distrus când contorul atinge valoarea zero.

Ideal ar fi ca acest contor să fie inițializat cu lungimea căii de la sursă la destinație. Dacă emițătorul nu cunoaște lungimea căii, poate inițializa contorul la valoarea cea mai defavorabilă, adică diametrul subrețelei.
 - 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 secvență în fiecare pachet pe care îl primește de la calculatorul gazdă asociat. Fiecare ruter necesită menținerea unei liste pentru fiecare ruter sursă, cu numerele de secvență provenite de la acel ruter sursă și care au fost deja trimise mai departe. Dacă sosește un pachet care se află în listă, el nu mai este trimis mai departe. Pentru a limita creșterea lungimii listei, fiecare listă trebuie însoțită de un contor, k , care semnifică faptul că toate numerele de secvență până la k au fost deja tratate. La recepția unui pachet este ușor 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 recepționat pe fiecare legătură de ieșire, ci doar pe acele linii care duc aproximativ în direcția potrivită. De obicei sunt puține motive pentru a trimite un pachet spre partea de vest a rețelei folosind o legătură spre est, decât dacă topologia rețelei este cu totul deosebită și ruterul este sigur de acest lucru.

Algoritmi dinamici:

- **Algoritmul de dirijare cu vectori distanță (distance vector routing)**

- Presupune că fiecare ruter menține o tabelă (de exemplu un vector) care păstrează cea mai bună distanță cunoscută spre fiecare destinație și linia care trebuie urmată pentru a ajunge acolo. Aceste tabele sunt actualizate prin schimbul de informații între nodurile vecine.
- Algoritmul de dirijare cu vectori distanță este cunoscut și sub alte nume, cel mai des algoritmul distribuit de dirijare **Bellman-Ford** și algoritmul **Ford-Fulkerson**, după numele cercetătorilor care l-au propus.
- În dirijarea pe baza vectorilor distanță, fiecare ruter păstrează o tabelă de dirijare conținând câte o intrare pentru fiecare ruter din subrețea. Această intrare are două părți: linia de ieșire preferată care se folosește pentru destinația respectivă și o estimare a timpului sau distanței până la acea destinație. Metrica folosită poate fi numărul de salturi, întârzierea în milisecunde, numărul total de pachete care așteaptă în cozi de-a lungul căii, sau ceva asemănător.
- Se presupune că ruterul cunoaște „distanța” spre fiecare dintre vecinii săi. Dacă se folosește metrica salturilor, distanța este doar de un salt. Dacă metrica folosită este cea a lungimii cozilor de așteptare, ruterul examinează pur și 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 (ștampila de timp) și le va trimite înapoi cât mai repede posibil.
- Ca exemplu, să presupunem că se folosește metrica întârzierilor și că ruterul cunoaște 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 destinație. De asemenea el recepționează o listă similară de la fiecare vecin. Să presupunem că una dintre aceste tabele tocmai a sosit de la vecinul X , cu X_i fiind estimarea lui X despre cât timp este necesar pentru a ajunge la ruterul i . Dacă ruterul știe că întârzierea spre X este m msec, el știe de asemenea că poate atinge ruterul i trecând prin X în $X_i + m$ 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 distanță funcționează în teorie, însă în practică are o limitare importantă: deși ea converge spre rezultatul corect, o face foarte lent. În particular, ea reacționează rapid la veștile bune, dar foarte lent la cele rele. Să considerăm un ruter care are un cel mai bun drum spre destinația X foarte lung. Dacă la următorul schimb de informații, vecinul său A raportează brusc o întârziere mică spre X, ruterul va comuta și 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 subrețea având calea cea mai lungă de lungime N salturi, după N schimburi fiecare ruter va afla despre liniile și ruterele nou apărute.
- Să considerăm acum situația în care toate liniile și ruterele sunt inițial în funcțiune. Ruterele B, C, D și E au distanțele spre A respectiv de 1, 2, 3, 4. Brusc, A se oprește sau, alternativ, linia dintre A și B este întreruptă, ceea ce reprezintă efectiv același lucru din punctul de vedere al lui B.
- La primul schimb de pachete, B nu primește nimic de la A. Din fericire, C spune: „Nici o problemă. Eu știu o cale spre A de lungime 2.” Însă B nu știe că această cale a lui C trece prin B însuși. După cunoștințele 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 și E nu își 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 acești vecini și va înregistra noua distanță spre A ca fiind 4.

- Din această figură se poate deduce de ce veștile 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 problemă 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 destinație, Y nu are de unde să știe dacă se află el însuși pe acea cale.

- **Dirijarea folosind starea legăturilor**

- Ideea algoritmului bazat pe starea legăturilor este simplă și poate fi formulată în 5 puncte. Fiecare ruter trebuie să facă următoarele:
 - Să descopere care sunt vecinii săi și afle adresele de rețea ale acestora.
 - Să măsoare întârzierea sau costul până la fiecare din vecinii săi.
 - Să pregătească un pachet prin care anunță 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 și toate întârzierile sunt măsurate experimental și 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 funcțiune, 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 anunțând 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 esențial ca acesta să poată determina dacă cele trei se referă la același F.
- **Măsurarea costului liniei:**
 - Algoritmul de dirijare bazat pe starea legăturilor cere ca fiecare ruter să știe, sau cel puțin 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) și împărțindu-l la doi, ruterul inițiator poate avea o estimare rezonabilă a întârzierii. Pentru rezultate și mai bune, testul poate fi repetat de mai multe ori, folosindu-se apoi valoarea medie obținută. Bineînțeles, această metodă presupune implicit că întârzierile sunt simetrice, dar nu mereu este așa.
 - O problemă interesantă este dacă să se considere sau nu încărcarea rețelei la măsurarea întârzierii. Pentru a ține 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 poziție din coadă.
- **Construirea pachetelor cu starea legăturilor:**
 - De îndată ce a fost colectată informația necesară pentru realizarea schimbului, se poate trece la pasul următor, fiecare ruter construind un pachet care conține toate datele. Pachetul începe cu identitatea expeditorului, urmată de un număr de

secvență, vârstă și o listă a vecinilor. Pentru fiecare vecin se specifică întârzierea asociată.

- Construirea pachetelor cu starea legăturilor se face ușor. 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 funcțiune a unui vecin sau a unei linii, sau repunerea lor în funcțiune, sau modificarea semnificativă a proprietăților 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 și instalate, ruterele care primesc primele pachete își vor schimba rutele. În consecință, rutere diferite ar putea folosi versiuni diferite ale topologiei, ceea ce poate duce la apariția unor inconsistențe, bucle, mașini inaccesibile sau a altor probleme.
- Ideea fundamentală este folosirea inundării pentru a distribui pachetele cu starea legăturilor. Pentru a avea controlul inundării, fiecare pachet conține un număr de secvență care este incrementat la fiecare nou pachet trimis. Ruterele păstrează evidența tuturor perechilor (ruter sursă, număr secvență) 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 excepția celei pe care a sosit. Dacă este un duplicat, pachetul este distrus. Dacă pachetul sosit are un număr de secvență mai mic decât cel mai mare număr de secvență detectat, atunci el este rejectat ca fiind învechit.
- Acest algoritm are câteva probleme, dar ele sunt tratabile. În primul rând, dacă numerele de secvență ating valoarea maximă posibilă și reîncep de la valori mici, se pot produce confuzii. Soluția este folosirea numerelor de secvență pe 32

biți. 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 evidența numerelor de secvență. Dacă va reîncepe de la 0, următorul pachet va fi rejectat ca duplicat.
- În al treilea rând, dacă numărul de secvență este alterat și se recepționează 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 secvență curent este 65540.
- Soluția tuturor acestor probleme este includerea vârstei pachetului după numărul de secvență și decrementarea sa la fiecare secundă. Când vârsta ajunge la zero, informația de la ruterul respective este distrusă. În mod normal un nou pachet apare, să zicem, la fiecare 10 sec., astfel încât informația de la ruter expiră doar dacă ruterul este oprit (sau dacă șase 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 supraviețui 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 subrețele, deoarece fiecare legătură este reprezentată. De fapt, fiecare legătură este reprezentată de două ori, o dată pentru fiecare direcție. 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 destinațiile posibile.
Rezultatul acestui algoritm poate fi trecut în tabelele de dirijare, iar apoi operațiile normale pot fi reluate.

11. Dirijarea prin difuzare

În unele aplicații, 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 acțiunilor sau transmisiuni radio în direct ar putea funcționa mai bine prin difuzarea datelor către toate mașinile, fiind la latitudinea celor interesate de date să le recepționeze. Trimiterea simultană a unui pachet către toate destinațiile se numește difuzare (broadcast); mai multe metode pentru realizarea ei au fost propuse.

O metodă de difuzare care nu are cerințe speciale pentru subrețea este ca sursa să trimită un pachet distinct către fiecare destinație. Metoda este nu numai consumatoare de lățime de bandă, dar ea cere ca sursa să dețină o listă completă a tuturor destinațiilor. S-ar putea ca în practică aceasta să fie singura metodă utilizabilă, însă ea este metoda cea mai puțin dorită.

Inundarea este un alt candidat evident. Deși inundarea este nepotrivită pentru comunicația obișnuită 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 aceeași cu problema ivită la folosirea sa ca algoritm de dirijare capăt la capăt: generează prea multe pachete și consumă lățime de bandă prea mare.

Al treilea algoritm este dirijarea multidestinație (multidestination routing). Dacă se folosește această metodă, fiecare pachet conține fie o listă a destinațiilor, fie o hartă de biți care indică destinațiile dorite. Atunci când un pachet ajunge la un ruter, ruterul verifică toate destinațiile pentru a determina setul liniilor de ieșire pe care trebuie trimis pachetul. (O linie de ieșire este selectată dacă este calea cea mai bună pentru cel puțin o destinație.) Ruterul generează o nouă copie a pachetului pentru fiecare linie de ieșire folosită și include în fiecare pachet doar acele destinații care folosesc linia respectivă. Efectul este partiționarea mulțimii destinațiilor între liniile de ieșire. După un număr suficient de salturi, fiecare pachet va conține o singură destinație și poate fi tratat ca un pachet normal. Dirijarea multidestinație este asemănătoare trimiterii separate a mai multor

pachete către adresele destinație, cu excepția faptului că atunci când mai multe pachete trebuie să urmeze aceeași cale, unul dintre ele plătește tot drumul, iar celelalte călătoresc gratuit.

Al patrulea algoritm de difuzare utilizează explicit arborele de scufundare al ruterului care inițiază difuzarea sau orice alt arbore de acoperire util. Un arbore de acoperire (spanning tree) este un subset al subrețelei care include toate ruterele și nu conține bucle. Dacă fiecare ruter cunoaște care din liniile sale participă la arborele de acoperire, el poate copia un pachet de difuzare recepționat pe toate liniile de ieșire care fac parte din arborele de acoperire, cu excepția celei pe care a fost recepționat. Această metodă asigură o utilizare deosebit de eficientă a lății de bandă, generând numărul minim de pachete necesare pentru a rezolva problema. Singura problemă este că, pentru a fi aplicabilă, fiecare ruter trebuie să dețină cunoștințe despre un anumit arbore de acoperire. Uneori această informație este disponibilă (de exemplu la dirijarea bazată pe starea legăturilor), iar alteori nu este disponibilă (de exemplu la dirijarea cu vectori distanță).

Ultimul nostru algoritm cu difuzare este o încercare de a aproxima comportamentul precedentului, chiar și atunci când ruterele nu știu 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 trimite de obicei pachete către sursa difuzării. Dacă este așa, este o șansă foarte mare ca însuși pachetul de difuzare să fi urmat cea mai bună cale, fiind astfel prima copie care ajunge la ruter. Aceasta fiind situația, ruterul trimite pachetul pe toate liniile de ieșire, cu excepția 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 multiplă (multicast)

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

mesaj către toți ceilalți 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 anunța 1000 de mașini dintr-o rețea cu un milion de noduri este ineficientă, deoarece majoritatea receptorilor nu sunt interesați de mesaj (sau chiar mai rău, sunt foarte interesați, 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 numește multicasting, iar algoritmul de dirijare asociat se numește **dirijare multicast (multicast routing)**. În această secțiune, vom descrie o modalitate de a realiza dirijarea multicast.

Dirijarea multicast necesită managementul grupului. Trebuie să existe modalități de a crea și distruge grupuri și de a permite proceselor să intre în grupuri sau să le părăsească. Cum se realizează aceste funcții nu este treaba algoritmului de dirijare. Important pentru algoritmul de dirijare este că atunci când un proces se atașează unui grup, el trebuie să informeze gazda sa despre aceasta. Este important ca ruterele să știe căror grupuri le aparțin calculatoarele gazdă asociate. Fie calculatoarele gazdă trebuie să anunțe 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 își informează vecinii, astfel că informația se propagă prin subrețea.

Pentru a realiza dirijarea multicast, fiecare ruter calculează arborele de acoperire care acoperă toate celelalte rutere din subrețea. Atunci când un proces trimite un pachet multicast către un grup, primul ruter își examinează arborele de acoperire și 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 și fiecare ruter cunoaște întreaga topologie a subrețelei, inclusiv apartenența calculatoarelor gazdă la grupuri. Atunci arborele poate fi retezat pornind de la

sfârșitul fiecărei căi, mergând spre rădăcină și eliminând toate ruterele care nu aparțin grupului respectiv.

În cazul dirijării folosind vectori distanță, poate fi aplicată o altă strategie de rețezare 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 și fără nici o conexiune la alte rutere primește un mesaj multicast pentru acel grup, el va răspunde cu un mesaj PRUNE (retezare), anunțând 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 primește astfel de mesaje pe toate liniile sale, el poate de asemenea să răspundă cu un mesaj PRUNE. În acest fel subrețeaua este retezată recursiv.

13. Dirijarea pentru calculatoare gazdă mobile

Calculatoarele gazdă care nu se mișcă niciodată se numesc staționare. Ele sunt conectate la rețea 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 staționare, dar care, din când în când, se mută dintr-un loc fixat în altul și care folosesc rețeaua doar atunci când sunt conectate fizic. Calculatoarele gazdă călătorești efectuează calcule în timp ce se deplasează și doresc să mențină 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 și doresc totuși să fie conectate.

Fiecare domeniu are unul sau mai mulți agenți pentru străini (foreign agent), procese ce țin evidența tuturor gazdelor mobile care vizitează domeniul.

În plus, fiecare domeniu are un agent local (local agent) care ține evidența 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 (atașarea fizică la LAN), fie prin plimbarea printr-o celulă,

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

- Periodic, fiecare agent pentru străini difuzează un pachet anunțându-și existența și adresa. Un utilizator mobil nou sosit trebuie să aștepte unul dintre aceste mesaje, însă dacă niciunul nu sosește 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 și unele informații de securitate.
- Agentul pentru străini contactează apoi agentul local al domeniului din care provine utilizatorul mobil și îi spune: „Unul dintre calculatoarele tale gazdă se află chiar aici.” Mesajul agentului pentru străini către agentul local conține adresa de rețea a agentului pentru străini. El mai conține de asemenea și informația de securitate, pentru a convinge agentul local că gazda mobilă este într-adevăr acolo.
- Agentul local examinează informația de securitate, care conține și o ștampilă de timp, pentru a dovedi că a fost generată în ultimele câteva secunde. Dacă este mulțumit, atunci anunță agentul pentru străini că poate trece la acțiune.
- Când agentul pentru străini primește confirmarea de la agentul local, el creează o nouă intrare în tabela sa și anunță utilizatorul mobil că a fost înregistrat.

Ideal, atunci când un calculator gazdă părăsește un domeniu, acest lucru trebuie anunțat, pentru a fi scos din evidență, însă mulți utilizatori pur și simplu închid calculatorul când termină.

Agentul local face două lucruri când primește un pachet pentru gazda mobilă. În primul rând, încapsulează pachetul în câmpul de informație utilă (payload) al unui pachet de trimis, pe care îl expediază apoi către agentul pentru străini. Acest mecanism se numește tunelare. După ce recepționează pachetul

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

În al doilea rând, agentul local anunță expeditorul mesajului ca de acum încolo să trimită pachetele adresate gazdei mobile încapsulându-le în câmpul de informație 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 locația de domiciliu.

14. Dirijarea în rețele AD HOC

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

Au fost propuși diferiți algoritmi de dirijare pentru rețelele ad hoc. Unul dintre cei mai interesanți este algoritmul de dirijare **AODV** (**Ad hoc On-demand Distance Vector**, vectori distanță ad hoc la cerere). Este o rudă îndepărtată a algoritmului cu vectori distanță Bellman-Ford dar adaptat pentru un mediu mobil și care ia în considerare limitele lărgimii de bandă și durata scurtă de funcționare a unei baterii în acest mediu. O altă caracteristică neobișnuită este faptul că acesta este un algoritm la cerere, adică determină o cale către o anumită destinație doar atunci când cineva dorește să trimită un pachet către acea destinație.

La orice moment de timp, o rețea ad hoc poate fi descrisă de un graf de noduri (rutere + gazde). Două noduri sunt conectate (de exemplu, există un arc ce le unește în graf) dacă pot comunica direct folosind radioul. Algoritmul AODV menține o tabelă în fiecare nod, în care cheia este destinația, și care dă informații despre acea destinație, inclusiv cărui vecin trebuie trimise pachetele pentru a ajunge la destinație. 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 secvență al sursei	Numărul de secvență al destinației	Contorul de salturi
---------------	-----------	--------------------	-------------------------------	------------------------------------	---------------------

Fig. 5-21. Formatul unui pachet ROUTE REQUEST

Adresa sursei	Adresa destinației	Numărul de secvență 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 și o parte de text. Antetul are o parte fixă de 20 de octeți și o parte opțională cu lungime variabilă. Formatul antetului este prezentat în fig. 5-53. El este transmis în ordinea big endian (cel mai semnificativ primul): de la stânga la dreapta, începând cu bitul cel mai semnificativ al câmpului Versiune. (Procesorul SPARC este de tip big endian; Pentium este de tip little endian - cel mai puțin semnificativ primul). Pe mașinile de tip little endian, este necesară o conversie prin program atât la transmisie cât și la recepție.

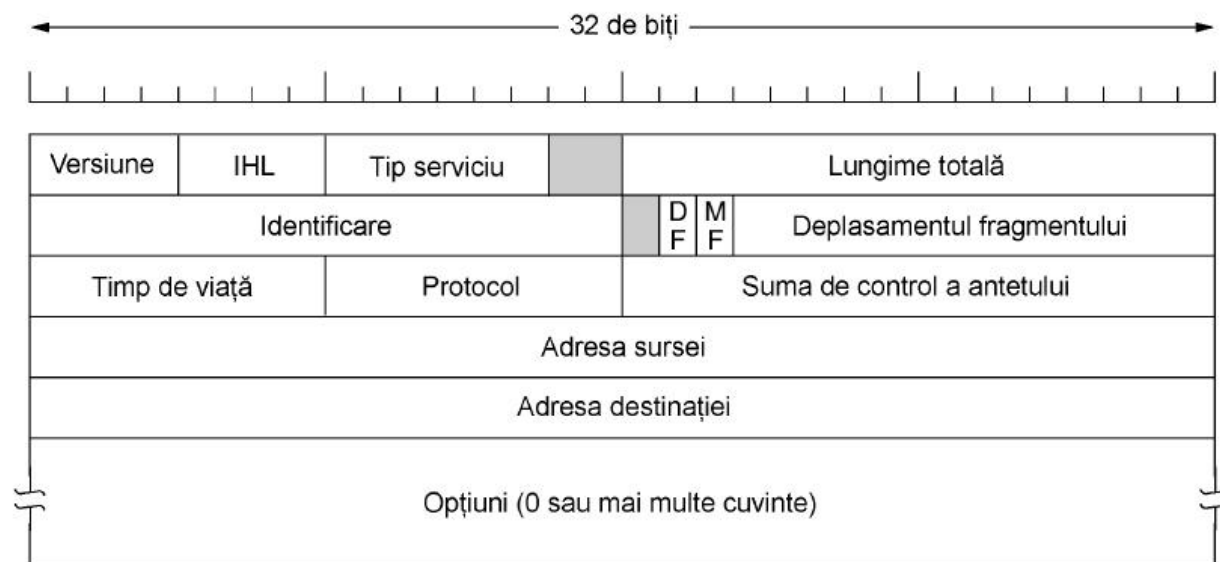


Fig. 5-53. Antetul IPv4 (protocolul Internet).

Câmpul Versiune memorează cărei versiuni de protocol îi aparține datagrama. Prin includerea unui câmp versiune în fiecare datagramă, devine posibil ca tranziția între versiuni să dureze ani de zile, cu unele mașini rulând vechea

versiune, iar altele rulând-o pe cea nouă. La ora actuală are loc o tranziție de la IPv4 la IPv6, care deja durează de câțiva ani și 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 dispoziție pentru a spune cât de lung este antetul, în cuvinte de 32 de octeți. Valoarea minimă este 5, care se aplică atunci când nu sunt prezente opțiuni. Valoarea maximă a acestui câmp de 4 biți este 15, ceea ce limitează antetul la 60 de octeți și, astfel, câmpul de opțiuni la 40 de octeți. Pentru unele opțiuni, cum ar fi cea care înregistrează calea pe care a mers un pachet, 40 de octeți sunt mult prea puțini, făcând această opțiune nefolositoare.

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

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

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

Lungimea totală include totul din datagramă - atât antet cât și date. Lungimea maximă este de 65535 octeți. În prezent, această limită superioară este tolerabilă, dar în viitoarele rețele cu capacități de gigaocteți vor fi necesare datagrame mai mari.

Câmpul Identificare este necesar pentru a permite gazdei destinație să determine cărei datagramă îi aparține un nou pachet primit. Toate fragmentele unei datagramă conțin aceeași valoare de Identificare.

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

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

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

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

Când nivelul rețea a asamblat o datagramă completă, trebuie să știe ce să facă cu ea. Câmpul Protocol spune cărui proces de transport trebuie să o predea. TCP este o posibilitate, dar tot așa sunt și UDP și 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 locații de memorie proaste din interiorul unui ruter. Algoritmul este de a aduna toate jumătățile de cuvinte, de 16 biți, atunci când acestea sosesc, folosind aritmetică în complement față de unu și păstrarea complementului față de unu al rezultatului. Pentru scopul acestui algoritm, se presupune că la sosire suma de control a antetului este zero. Acest algoritm este mai robust decât folosirea unei adunări normale. Observați că suma de control a antetului trebuie recalculată la fiecare salt, pentru că întotdeauna se schimbă cel puțin un câmp (câmpul timp de viață), dar se pot folosi trucuri pentru a accelera calculul.

Adresa sursei și Adresa destinației indică numărul de rețea și numărul de gazdă. Vom discuta adresele Internet în secțiunea următoare.

Câmpul Opțiuni a fost proiectat pentru a oferi un subterfugiu care să permită versiunilor viitoare ale protocolului să includă informații care nu sunt prezente în proiectul original, pentru a permite cercetătorilor să încerce noi idei și pentru a evita alocarea unor biți din antet pentru informații folosite rar. Opțiunile sunt de lungime variabilă. Fiecare începe cu un cod de un octet care identifică opțiunea. Unele opțiuni sunt urmate de un câmp de un octet reprezentând lungimea opțiunii, urmat de unul sau mai mulți octeți de date. Câmpul Opțiuni este completat până la un multiplu de 4 octeți.

16. Adrese IP

Fiecare gazdă și ruter din Internet are o adresă IP, care codifică adresa sa de rețea și de gazdă. Combinația este unică: în principiu nu există două mașini cu

aceeași adresă IP. Toate adresele IP sunt de 32 de biți lungime și sunt folosite în câmpurile Adresă sursă și Adresă destinație ale pachetelor IP. Este important de observat că o adresă IP nu se referă de fapt la o gazdă. Se referă de fapt la o interfață de rețea, deci dacă o gazdă este în două rețele, trebuie să folosească două adrese IP. Totuși în practică, cele mai multe gazde sunt conectate la o singură rețea și deci au o adresă IP.

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

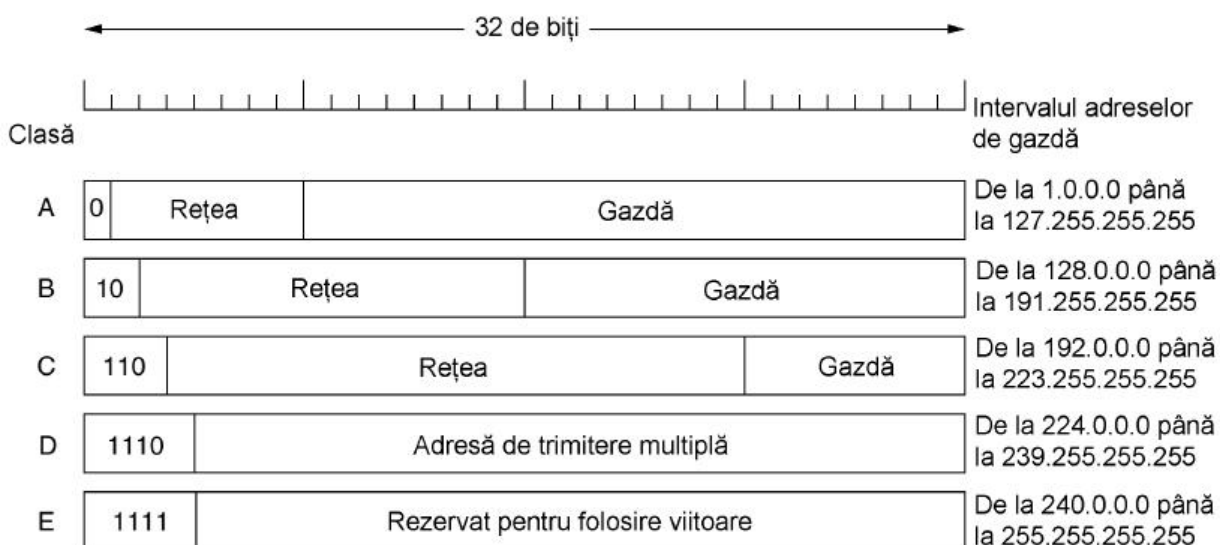


Fig. 5-55. Formatul adreselor IP.

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

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

Insuficienta adreselor IP este o problema. Soluția care a fost implementată acum și care a dat Internet-ului un pic de spațiu de manevră este **CIDR (Classless InterDomain Routing** - Dirijarea fără clase între domenii). Ideea de la baza CIDR, este de a alocă adresele IP rămase, în blocuri de dimensiune variabilă, fără a se ține cont de clase. Dacă un site are nevoie, să zicem, de 2000 de adrese, îi este dat un bloc de 2048 adrese la o graniță de 2048 de octeți.

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

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

17. NAT – Traducerea adreselor de rețea

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

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

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

10.0.0.0 - 10.255.255.255/8 (16.777.216 gazde)

172.16.0.0 - 172.31.255.255/12 (1.048.576 gazde)

192.168.0.0 - 192.168.255.255/16 (65.536 gazde)

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

În interiorul companiei fiecare mașină are o adresă unică de forma 10.x.y.z. Totuși, când un pachet părăsește compania, trece printr-o **unitate NAT** (eng.: **NAT box**) care convertește adresa IP internă 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 știe 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 emițător, dar numai 1 bit este încă nefolosit. În principiu, o nouă opțiune ar putea fi creată pentru a reține adevărata adresă a sursei, dar acest lucru ar necesita schimbarea codului din protocolul IP de pe toate stațiile din întregul Internet pentru a putea prelucra noua opțiune. Aceasta nu este o alternativă promițătoare pentru o rezolvare rapidă.

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

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

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

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

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

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

- În al doilea rând, NAT schimbă natura Internetului de la o rețea fără conexiuni la un fel de rețea cu conexiuni.
- 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 informație 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 aplicații introduc adrese IP în corpul mesajului. Receptorul extrage aceste adrese și le folosește. De vreme ce NAT nu știe nimic despre aceste adrese, nu le poate înlocui, deci orice încercare de a le folosi de către cealaltă parte va eșua.

Acestea și alte probleme ale NAT sunt discutate în RFC 2993. În general, opozanții NAT spun că rezolvând problema insuficienței adreselor IP cu o soluție temporară și urâtă, presiunea pentru a implementa soluția reală, adică tranziția la IPv6, este redusă și 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 neobișnuit, evenimentul este raportat prin **ICMP (Internet Control Message Protocol)** – protocolul mesajelor de control din Internet), care este folosit și pentru testarea Internet-ului. Sunt definite aproape o duzină de tipuri de mesaje ICMP. Cele mai importante sunt enumerate în fig. 5-61. Fiecare tip de mesaj ICMP este încapsulat într-un pachet IP.

Tipul mesajului	Descriere
Destinație inaccesibilă	Pachetul nu poate fi livrat
Timp depășit	Câmpul timp de viață a ajuns la 0
Problemă de parametru	Câmp invalid în antet
Oprire sursă	Pachet de șoc
Redirectare	Învăță 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 DESTINAȚIE INACCESIBILĂ (DESTINATION UNREACHABLE) este folosit atunci când subrețeaua sau un ruter nu pot localiza destinația, sau un pachet cu bitul DF nu poate fi livrat deoarece o rețea cu „pachete mici” îi stă în cale.

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

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

Mesajul OPRIRE SURSĂ (SOURCE QUENCH) a fost folosit pe vremuri pentru a limita traficul gazdelor care trimiteau prea multe pachete. Când o gazdă primea acest mesaj, era de așteptat să încetinească ritmul de transmisie. Este folosit arareori, deoarece când apare congestie, aceste pachete au tendința 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 greșit. Este folosit de ruter pentru a spune gazdei emițătoare despre eroarea probabilă.

Mesajele CERERE ECOU (ECHO REQUEST) și RĂSPUNS ECOU (ECHO REPLY) sunt folosite pentru a vedea dacă o anumită destinație este accesibilă și activă. Este de așteptat ca la recepția mesajului ECOU, destinația să răspundă printr-un mesaj RĂSPUNS ECOU.

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

19. Protocolul de rezoluție a adresei: ARP

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

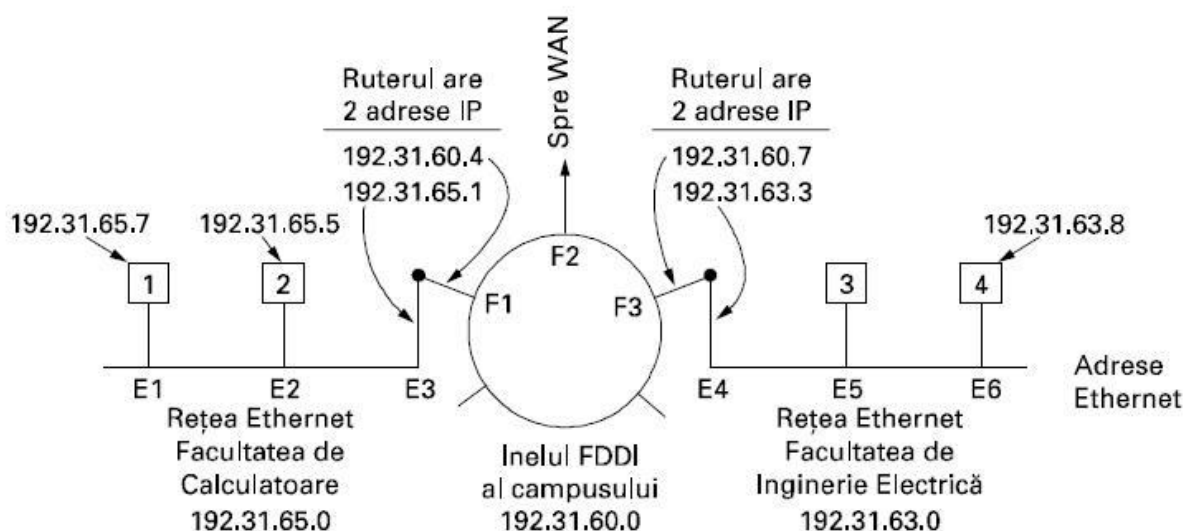


Fig. 5-62. Trei rețele /24 interconectate: două rețele Ethernet și un inel FDDI.

Se pune atunci întrebarea: Cum sunt transformate adresele IP în adrese la nivelul legăturii de date, ca de exemplu Ethernet? Să începem prin a vedea cum trimite un utilizator de pe gazda 1 un pachet unui utilizator de pe gazda 2.

Presupunem că expeditorul știe numele destinatarului, ceva de genul `ary@eagle.cs.uni.edu`. Primul pas este aflarea adresei IP a gazdei 2, cunoscută ca `eagle.cs.uni.edu`. Această căutare este făcută de sistemul numelor de domenii (DNS) 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 și văd că destinatarul se află în propria rețea, dar au nevoie de un mijloc prin care să determine adresa Ethernet a destinatarului. O soluție este să avem undeva în sistem un fișier de configurare care transformă adresele IP în adrese Ethernet. Această soluție este posibilă, desigur, dar pentru organizații cu mii de mașini, menținerea fișierelor actualizate este o acțiune consumatoare de timp și care poate genera erori.

O soluție mai bună este ca gazda 1 să trimită un pachet de difuzare în rețeaua Ethernet întrebând: „Cine este proprietarul adresei IP 192.31.65.5?”. Pachetul de difuzare va ajunge la toate mașinile din rețeaua Ethernet 192.31.65.0 și fiecare își va verifica adresa IP. Numai gazda 2 va răspunde cu adresa sa Ethernet (E2). În acest mod gazda 1 află că adresa IP 192.31.65.5 este pe gazda cu adresa Ethernet E2. Protocolul folosit pentru a pune astfel de întrebări și a primi răspunsul se numește **ARP (Address Resolution Protocol - Protocolul de rezoluție a adresei)**. Aproape toate mașinile din Internet îl folosesc. Avantajul folosirii ARP față de fișierele de configurare îl reprezintă simplitatea. Administratorul de sistem nu trebuie să facă prea multe, decât să atribuie fiecărei mașini o adresă IP și să hotărască măștile subrețelelor. ARP-ul face restul.

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

Pentru a face ARP-ul mai eficient sunt posibile mai multe optimizări. Pentru început, la fiecare execuție a ARP, mașina păstrează rezultatul pentru cazul în care are nevoie să contacteze din nou aceeași mașină în scurt timp. 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 forțează să execute ARP, pentru a determina adresa Ethernet a expeditorului. Această difuzare

ARP poate fi evitată obligând gazda 1 să includă în pachetul ARP corespondența dintre adresa sa IP și adresa Ethernet. Când pachetul ARP ajunge la gazda 2, perechea (192.31.65.7, E1) este memorată local de ARP pentru o folosire ulterioară. De fapt, toate mașinile din rețeaua Ethernet pot memora această relație în memoria ARP locală.

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

20. DHCP

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

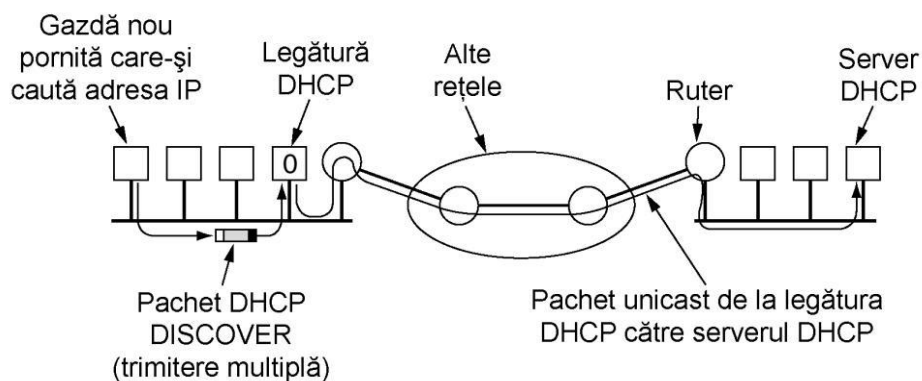


Fig. 5-63. Funcționarea DHCP.

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

Atunci când găsește un pachet DHCP DISCOVER, îl trimite ca pachet unicast serverului DHCP, posibil într-o rețea depărtată. Singura informație de care are nevoie agentul este adresa IP a serverului DHCP.

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

21. Protocolul de dirijare folosit de porțile interioare: OSPF

Internet-ul este construit dintr-un număr mare de sisteme autonome(AS). Fiecare AS este administrat de o organizație diferită și poate folosi propriul algoritm de dirijare în interior. De exemplu, rețelele interne ale companiilor X, Y și Z ar fi văzute ca trei AS-uri dacă toate ar fi în Internet. Toate trei pot folosi intern algoritmi de dirijare diferiți. Cu toate acestea, existența standardelor, chiar și pentru dirijarea internă, simplifică implementarea la granițele dintre AS-uri și permite reutilizarea codului. **OSPF (Open Shortest Path First** - protocol public

(deschis) bazat pe calea cea mai scurtă) a devenit un standard în 1990. Mulți producători de rutere oferă support pentru el și acesta a devenit principalul protocol de porți interioare.

OSPF suportă trei tipuri de conexiuni și rețele:

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

OSPF funcționează prin abstractizarea colecției de rețele, rutere și linii reale

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

Un sistem autonom :

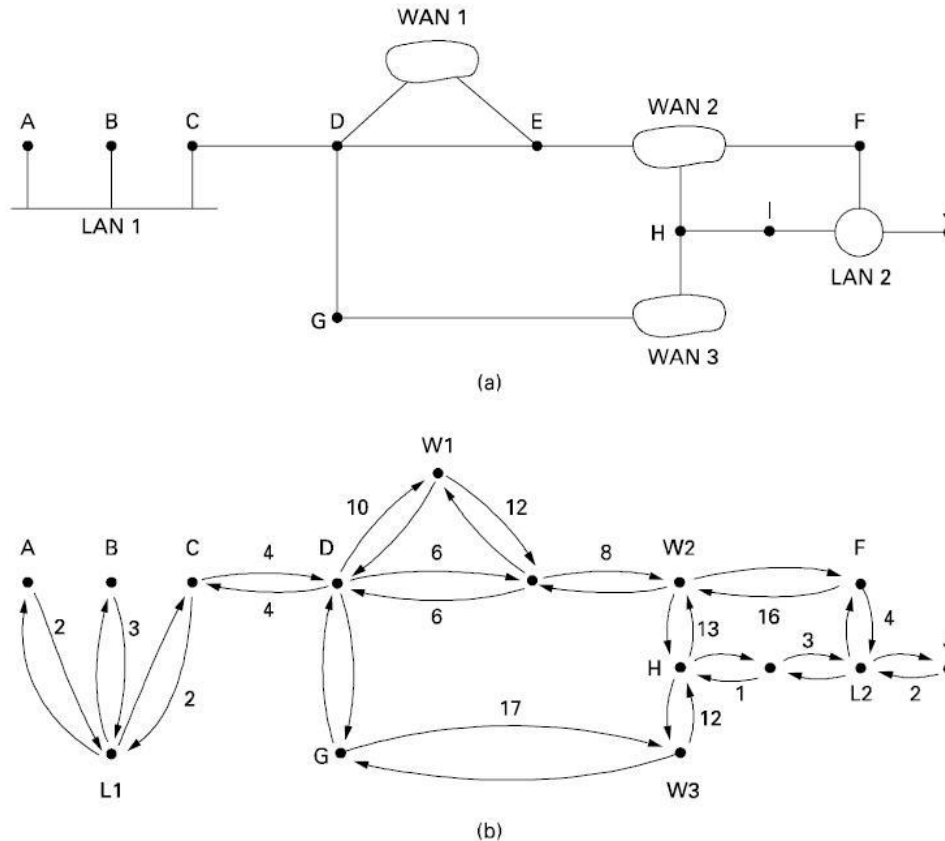


Fig. 5-64. (a) Un sistem autonom. (b) O reprezentare de tip graf a lui (a).

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

Orice AS are o zonă **de coloană vertebrală**, numită zona 0. Toate zonele sunt conectate la coloana vertebrală, eventual prin 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 și are un cost. Fiecare ruter care este conectat la două sau mai multe zone aparține coloanei vertebrale. Analog cu celelalte zone, topologia coloanei vertebrale nu este vizibilă din afara coloanei vertebrale.

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

În timpul operării normale pot fi necesare trei tipuri de căi: intrazonale, interzonale și interASuri. Rutele intrazonale sunt cele mai ușoare, din moment ce ruterul sursă știe întotdeauna calea cea mai scurtă spre ruterul destinație. Dirijarea interzonală se desfășoară întotdeauna în trei pași: drum de la sursă la coloana vertebrală; drum de-a lungul coloanei vertebrale până la zona destinație; drum la destinație. Acest algoritm forțează o configurație de tip stea pentru OSPF, coloana vertebrală fiind concentratorul (hub), iar celelalte zone fiind spițele. Pachetele sunt dirijate de la sursă la destinație „ca atare”. Ele nu sunt încapsulate sau trecute prin tunel, cu excepția cazului în care merg spre o zonă a cărei unică conexiune la coloana vertebrală este un tunel.

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

Ruterele din același LAN sunt toate vecine.

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

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

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

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

22. Protocolul de dirijare pentru porți externe: BGP

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

Ruterele ce folosesc protocolul de porți exterioare trebuie să țină cont într-o mare măsură de politică. De exemplu, un AS al unei corporații poate dori facilitatea de a trimite pachete oricărui site Internet și să recepționeze 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 porți externe, în general și BGP în particular, au fost proiectate pentru a permite forțarea multor tipuri de politici de dirijare pentru traficul între AS-uri. Politicile tipice implică considerații politice, de securitate sau economice. Câteva exemple de constrângeri de dirijare sunt:

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

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

23. IPv6

IPv6 menține caracteristicile bune ale IP-ului, le elimină sau atenuează pe cele rele și 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 și DNS, câteodată fiind necesare mici modificări (majoritatea pentru a putea lucra cu adrese mai lungi).

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

A doua mare îmbunătățire a lui IPv6 este simplificarea antetului. El conține numai 7 câmpuri (față de 13 în IPv4). Această schimbare permite rutelor să prelucreză pachetele mai rapid, îmbunătățind astfel productivitatea și întârzierea.

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

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

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

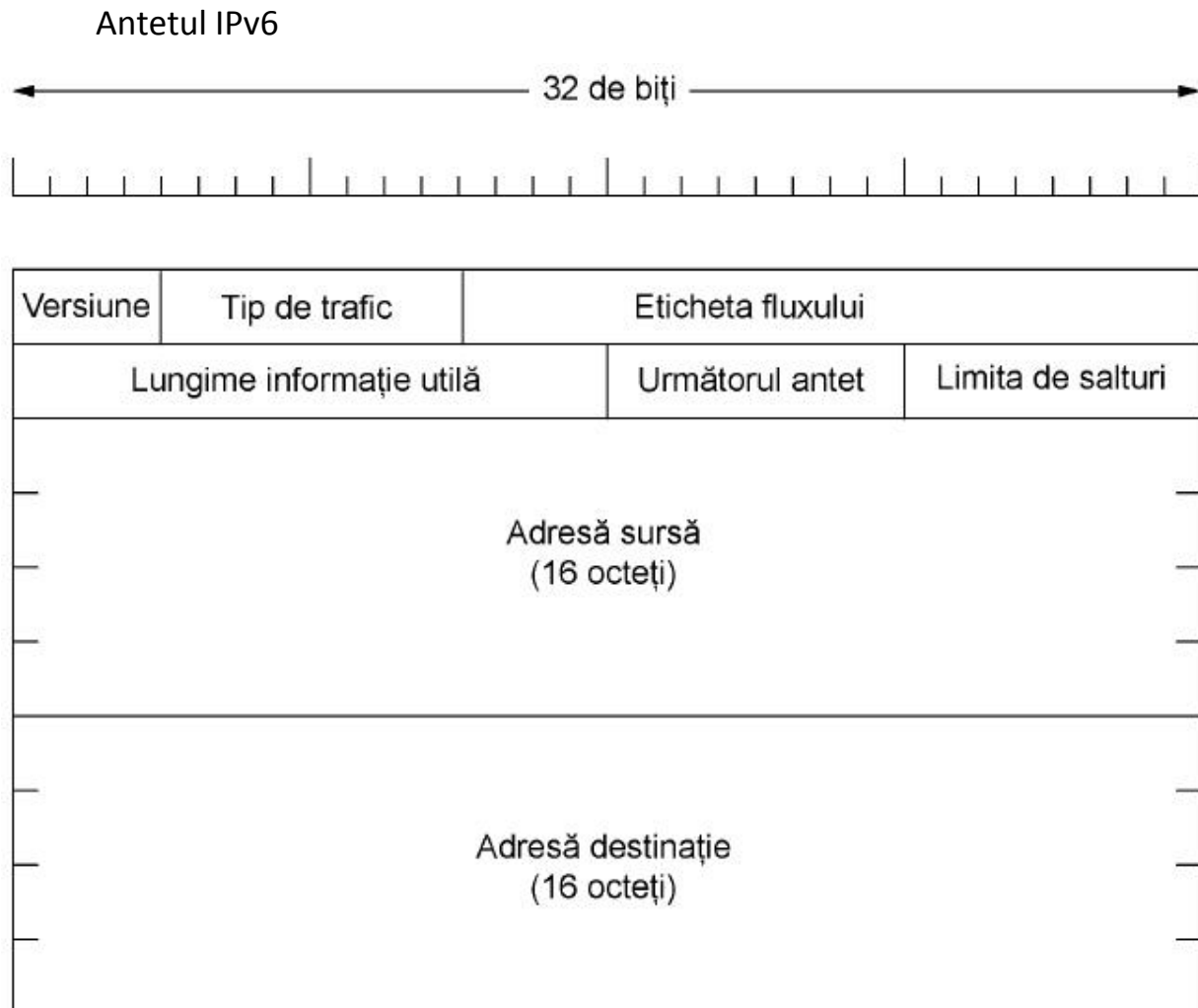


Fig. 5-68. Antetul fix IPv6 (obligatoriu).

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

Câmpul Tip de trafic (Traffic class) este folosit pentru a distinge între pachetele care au diverse cerințe de livrare în timp real. Un câmp cu acest scop a

existat în IP de la început, dar a fost implementat sporadic de către rutere. În acest moment se desfășoară experimente pentru a determina cum poate fi utilizat cel mai bine pentru transmisii multimedia.

Câmpul Eticheta fluxului este încă experimental, dar va fi folosit pentru a permite unei surse și unei destinații să stabilească o pseudo-conexiune cu proprietăți și cerințe particulare. De exemplu, un șir de pachete de la un proces de pe o anumită gazdă sursă către un anumit proces pe o anumită gazdă destinație poate avea cerințe de întârziere stricte și din acest motiv necesită capacitate de transmisie rezervată. Fluxul poate fi stabilit în avans și poate primi un identificator. Când apare un pachet cu o Etichetă a fluxului diferită de zero, toate ruterele pot să o caute în tabelele interne pentru a vedea ce tip de tratament special necesită. Ca efect, fluxurile sunt o încercare de a combina două moduri: flexibilitatea unei subrețele cu datagrame și garanțiile unei subrețele cu circuite virtuale. Fiecare flux este desemnat de adresa sursă, adresa destinație și numărul de flux, așa încât, între o pereche dată de adrese IP pot exista mai multe fluxuri active în același timp. De asemenea, în acest mod, chiar dacă două fluxuri venind de la gazde diferite, dar cu același număr de flux trec prin același ruter, ruterul va fi capabil să le separe folosind adresele sursă și destinație. Se așteaptă ca numerele de flux să fie alese aleator, în loc de a fi atribuite secvențial începând cu 1, pentru că se așteaptă ca ruterele să le folosească în tabele de dispersie.

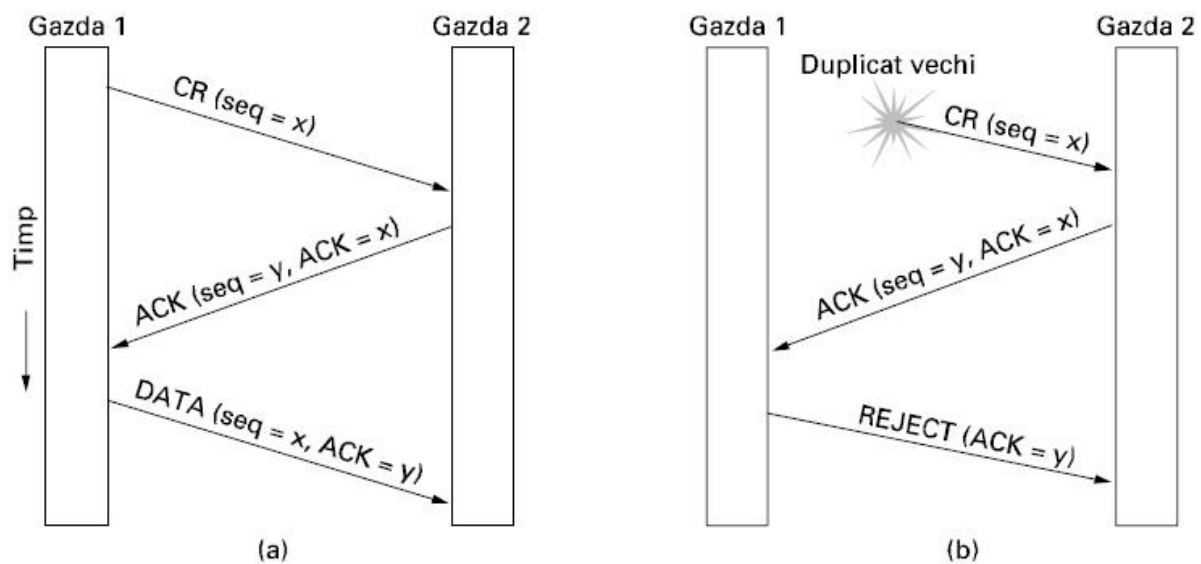
Câmpul Lungimea informației utile spune câți octeți urmează după antetul de 40 de octeți din fig. 5-68. Numele a fost schimbat față de câmpul Lungime totală din IPv4 deoarece semnificația este ușor modificată: cei 40 de octeți nu mai sunt parte a lungimii (așa cum erau înainte).

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

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

Apoi urmează câmpurile Adresă sursă și Adresă destinație.

24. Stabilirea si eliberarea conexiunii la nivelul transport



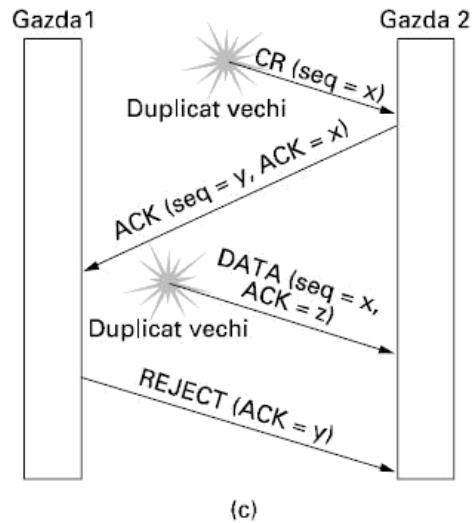


Fig. 6-11. Trei scenarii posibile de stabilire a conexiunii pentru un protocol cu înțelegere în trei pași. CR reprezintă CONNECTION REQUEST. (a) Cazul normal, (b) Un duplicat vechi al unui mesaj CONNECTION REQUEST apare când nu trebuie, (c) Sunt duplicate atât CONNECTION REQUEST cât și CONNECTION ACCEPTED.

Deoarece TPDU-urile de control pot și ele să fie întârziate, pot apărea probleme atunci când entitățile de transport încercă să cadă de acord asupra numărului inițial de secvență. Să presupunem, de exemplu, că, pentru a stabili o conexiune, gazda 1 trimite un mesaj CONNECTION REQUEST conținând numărul de secvență inițial propus și portul destinație gazdei 2. Acesta va confirma mesajul trimițând înapoi un TPDU CONNECTION ACCEPTED. Dacă TPDU-ul CONNECTION REQUEST este pierdut, dar un duplicat întârziat al unui alt CONNECTION REQUEST va ajunge la gazda 2, atunci conexiunea nu va fi stabilită corect.

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

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

Eliberarea asimetrică este bruscă și poate genera pierderi de date. Să considerăm scenariul din fig. 6-12. După stabilirea conexiunii, gazda 1 trimite un TPDU care ajunge corect la gazda 2. Gazda 1 mai trimite un TPDU dar, înainte ca acesta să ajungă la destinație, gazda 2 trimite DISCONNECT REQUEST. În acest caz, conexiunea va fi eliberată și vor fi pierdute date.

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

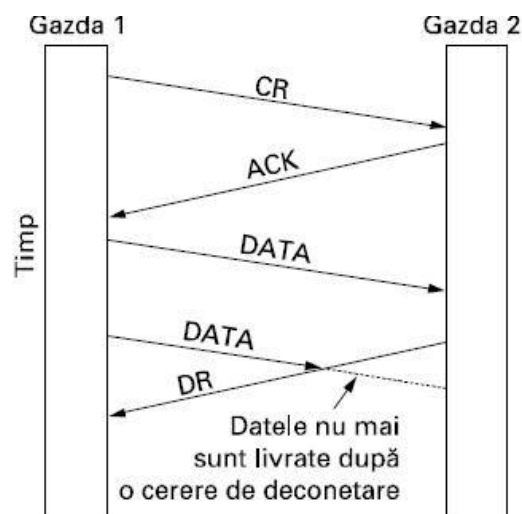


Fig. 6-12. Deconectare bruscă cu pierdere de date. CR= CONNECTION REQUEST, ACK=CONNECTION ACCEPTED, DR=DISCONNECT REQUEST.

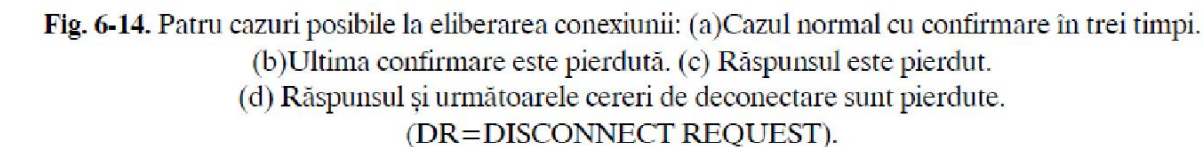
Eliberarea simetrică este utilă atunci când fiecare proces are o cantitate fixă de date de trimis și știe bine când trebuie să transmită și când a terminat. În alte situații însă, nu este deloc ușor de determinat când trebuie eliberată conexiunea și când a fost trimis tot ce era de transmis. S-ar putea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

Din nefericire, acest protocol nu merge întotdeauna. Binecunoscuta problemă a celor două armate este similară acestei situații: să ne imaginăm că armata albă și-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-și sincronizeze atacul. Totuși singura lor posibilitate de comunicație este să trimită un mesager care să străbată valea. Mesagerul poate fi capturat de armata albă și mesajul poate fi pierdut (adică vor trebui să utilizeze un canal de comunicație nesigur). Problema este următoarea: există vreun protocol care să permită armatelor albastre să învingă?

Să presupunem că comandantul primei armate albastre trimite un mesaj: „Propun să atacăm pe 29 martie”, mesajul ajunge la armata 2 al cărei comandant răspunde: „De acord” iar răspunsul ajunge înapoi la armata 1. Va avea loc atacul în acest caz? Probabil că nu, deoarece comandantul armatei 2 nu știe dacă răspunsul său a ajuns sau nu la destinație. Dacă nu a ajuns, armata 1 nu va ataca, deci ar fi o prostie din partea lui să intre în luptă. Să încercăm să îmbunătățim protocolul, transformându-l într-unul cu înțelegere în trei pași. Inițiatorul propunerii de atac trebuie să confirme răspunsul. Presupunând că nici un mesaj nu este pierdut, armata 2 va avea confirmarea, dar comandantul armatei 1 va ezita acum. Până la urmă, el nu știe dacă confirmarea sa a ajuns la destinație și este sigur că dacă aceasta nu a ajuns, armata 2 nu va ataca. Am putea să încercăm un protocol cu confirmare în patru timpi, dar ne-am lovit de aceleași probleme. De fapt, poate fi

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



(b) Ultima confirmare este pierdută. (c) Răspunsul este pierdut.
(d) Răspunsul și 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 aplicațiilor o modalitate de a trimite datagrame IP încapsulate și de a le transmite fără a fi nevoie să stabilească o conexiune. UDP este descris în RFC 768.

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



Fig. 6-23. Antetul UDP.

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

Câmpul lungime UDP include antetul de 8 octeți și datele. Câmpul sumă de control UDP este optional și stocat ca 0 (zero) dacă nu este calculat (o valoare de adevăr 0 rezultată în urma calculelor **este memorată ca un șir de biți 1**). Dezactivarea acestuia este o prostie, excepție făcând cazul în care calitatea informației chiar nu contează (de exemplu, transmisia vocală digitalizată).

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

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

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

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 aplicațiilor multimedia în timp real. În particular, având în

vedere că radioul pe internet, telefonia pe Internet, muzica la cerere, video-conferințele, video la cerere și alte aplicații multimedia au devenit mai răspândite, oamenii au descoperit că fiecare aplicație a folosit, mai mult sau mai puțin, același protocol de transport în timp real. Treptat a devenit clar faptul că un protocol generic de transport în timp real, pentru aplicații multimedia, ar fi o idee bună. Așa a luat naștere **RTP-ul (Real-time Transport Protocol, rom: Protocol de transport în timp real)**.

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

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

Funcția de bază a RTP-ului este multiplexarea mai multor fluxuri de date în timp real într-un singur flux de pachete UDP. Fluxul UDP poate fi transmis către o singură destinație (unicasting) sau către destinații multiple (multicasting). Deoarece RTP-ul folosește numai UDP normal, pachetele sale nu sunt tratate în mod special de către rutere, decât dacă sunt activate anumite facilități de calitate a serviciilor din IP. În particular, nu există garanții speciale referitoare la livrare, bruij etc. Fiecărui pachet trimis în fluxul RTP i se dă un număr cu unu mai mare decât al predecesorului său. Această numerotare permite destinației să

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

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

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

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

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

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

27. TCP - Transport Communication Protocol

TCP (Transport Communication Protocol - protocol de comunicație de nivel transport) a fost proiectat explicit pentru a asigura un flux sigur de octeți de la un capăt la celălalt al conexiunii într-o inter-rețea nesigură. O inter-rețea diferă de o rețea propriu-zisă prin faptul că diferite părți ale sale pot diferi substanțial în topologie, lărgime de bandă, întârzieri, dimensiunea pachetelor și alți parametri. TCP a fost proiectat să se adapteze în mod dinamic la proprietățile inter-rețelei și să fie robust în ceea ce privește mai multe tipuri de defecte.

O conexiune TCP este un flux de octeți și nu un flux de mesaje.

Dimensiunile mesajelor nu se conservă de la un capăt la celălalt. Atunci când o aplicație trimite date către TCP, TCP-ul le poate expedia imediat sau le poate reține într-un tampon (în scopul colectării unei cantități mai mari de informație pe care să o expedieze toată odată), după bunul său plac. Cu toate acestea, câteodată, aplicația dorește ca informația să fie expediată imediat. De exemplu, să presupunem că un utilizator este conectat la o mașină de la distanță. După ce a fost terminată o linie de comandă și s-a tastat Return, este esențial ca linia să fie imediat expediată către mașina de la distanță și să nu fie memorată până la terminarea următoarei linii. Pentru a forța expedierea, aplicația poate folosi indicatorul PUSH, care îi semnalează TCP-ului să nu întârzie procesul de transmisie.

O ultimă caracteristică a serviciului TCP care merită menționată aici constă în informația urgentă. Atunci când un utilizator apasă tasta DEL sau CTRL-C pentru a întrerupe o prelucrare la distanță, aflată deja în execuție, aplicația emițător plasează o informație de control în fluxul de date și o furnizează TCP-ului împreună cu indicatorul URGENT. Acest eveniment impune TCP-ului întreruperea acumulării de informație și transmisia imediată a întregii informații disponibile deja pentru conexiunea respectivă. Atunci când informația urgentă este recepționată la destinație, aplicația receptoare este întreruptă astfel încât, eliberată de orice altă activitate, aplicația să poată citi fluxul de date și să poată regăsi informația urgentă. Sfârșitul informației urgente este marcat, astfel încât aplicația să știe când se termină informația. Începutul informației urgente nu este marcat. Este sarcina aplicației să determine acest început.

Protocolul de bază utilizat de către entitățile TCP este protocolul cu fereastră glisantă. Atunci când un emițător transmite un segment, el pornește un cronometru. Atunci când un segment ajunge la destinație, entitatea TCP receptoare trimite înapoi un segment (cu informație utilă, dacă aceasta există sau fără, în caz contrar) care conține totodată și numărul de secvență următor pe care aceasta se așteaptă să-l recepționeze. Dacă cronometrul emițătorului depășește o anumită valoare înaintea primirii confirmării, emițătorul retransmite segmentul neconfirmat.

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

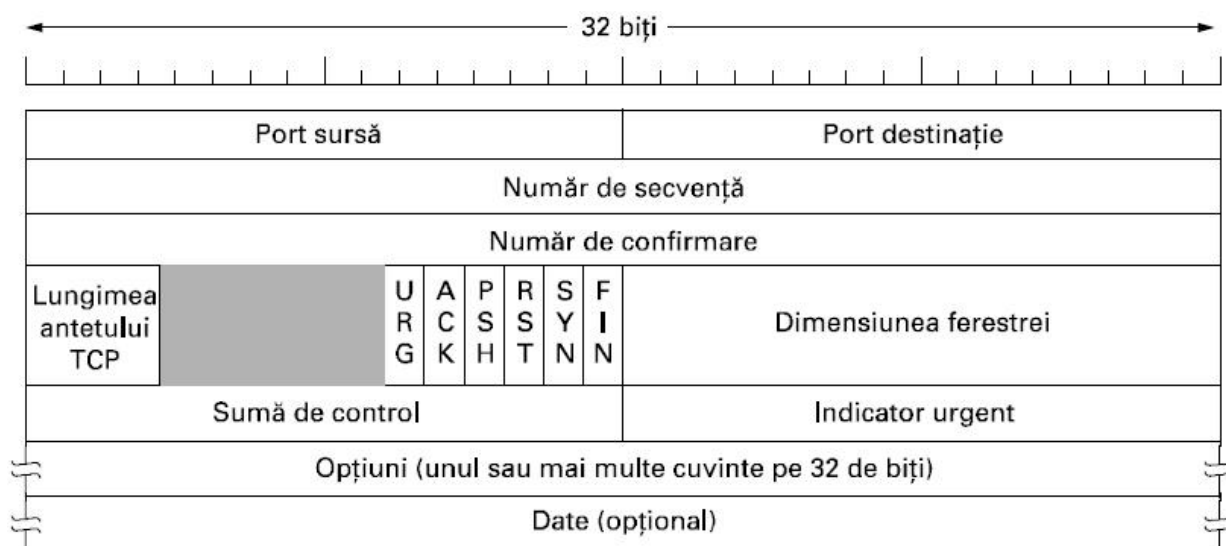


Fig. 6-29. Antetul TCP.

Câmpurile Port sursă și Port destinație identifică punctele finale ale conexiunii. Porturile general cunoscute sunt definite la www.iana.org, dar fiecare gazdă le poate alocă pe celelalte după cum dorește. Un port formează împreună cu adresa IP a mașinii sale un unic punct de capăt (eng.: end point) de 48 de biți. Conexiunea este identificată de punctele de capăt ale sursei și destinației.

Câmpurile Număr de secvență și Număr de confirmare au semnificația funcțiilor lor uzuale. Trebuie observat că cel din urmă indică octetul următor așteptat și nu ultimul octet recepționat în mod corect. Ambele câmpuri au lungimea de 32 de biți, deoarece într-un flux TCP fiecare bit de informație este numerotat.

Lungimea antetului TCP indică numărul de cuvinte de 32 de biți care sunt conținute în antetul TCP. Această informație este utilă, deoarece câmpul Opțiuni

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

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

Urmează acum șase indicatori de câte un bit. URG este poziționat pe 1 dacă Indicatorul Urgent este valid. Indicatorul Urgent este folosit pentru a indica deplasamentul în octeți față de numărul curent de secvență la care se găsește informația urgentă. O astfel de facilitare ține locul mesajelor de întrerupere. Așa cum am menționat deja anterior, această facilitare reprezintă esența modului în care emițătorul poate transmite un semnal receptorului fără ca TCP-ul în sine să fie cauza întreruperii.

Bitul ACK este poziționat pe 1 pentru a indica faptul că Numărul de confirmare este valid. În cazul în care ACK este poziționat pe 0, segmentul în discuție nu conține o confirmare și câmpul Număr de confirmare este ignorat.

Bitul PSH indică informația FORȚATĂ. Receptorul este rugat respectuos să livreze aplicației informația respectivă imediat ce este recepționată și să nu o memoreze în așteptarea umplerii tampoanelor de comunicație (lucru care, altminteri, ar fi făcut din rațiuni de eficiență).

Bitul RST este folosit pentru a desființa o conexiune care a devenit inutilizabilă datorită defecțiunii unei mașini sau oricărui alt motiv. El este de asemenea utilizat pentru a refuza un segment invalid sau o încercare de deschidere a unei conexiuni. În general, recepționarea unui segment având acest bit poziționat indică o problemă care trebuie tratată în funcție de context.

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

o confirmare, având deci $\text{SYN} = 1$ și $\text{ACK} = 1$. În esență, bitul SYN este utilizat pentru a indica o CERERE DE CONEXIUNE și o CONEXIUNE ACCEPTATĂ, bitul ACK făcând distincția între cele două posibilități.

Bitul FIN este folosit pentru a încheia o conexiune. El indică faptul că emițătorul nu mai are nici o informație de transmis. Cu toate acestea, după închiderea conexiunii, un proces poate recepționa în continuare date pe o durată nedefinită. Ambele segmente, SYN și FIN, conțin numere de secvență și astfel este garantat faptul că ele vor fi prelucrate în ordinea corectă.

În TCP, fluxul de control este tratat prin ferestre glisante de dimensiune variabilă. Câmpul Fereastră indică numărul de octeți care pot fi trimiși, începând de la octetul confirmat. Un câmp Fereastră de valoare 0 este perfect legal și spune că octeții până la Număr de confirmare - 1 inclusiv au fost recepționați, dar receptorul dorește cu ardoare o pauză, așa că mulțumește frumos, dar pentru moment nu dorește continuarea transferului. Permisivitatea de expediere poate fi acordată ulterior de către receptor prin trimiterea unui segment având același Număr de confirmare, dar un câmp Fereastră cu o valoare nenulă.

Este de asemenea prevăzută o Sumă de control, în scopul obținerii unei fiabilități extreme. Această sumă de control este calculată pentru antet, informație și pseudo-antetul conceptual prezentat în fig. 6-30. În momentul calculului, Suma de control TCP este poziționată pe zero, iar câmpul de date este completat cu un octet suplimentar nul, dacă lungimea sa este un număr impar.

Algoritmul de calcul al sumei de control este simplu, el adunând toate cuvintele de 16 biți în complement față de 1 și aplicând apoi încă o dată complementul față de 1 asupra sumei. În acest mod, atunci când receptorul aplică același calcul asupra întregului segment, inclusiv asupra Sumei de control, rezultatul ar trebui să fie 0.

Pseudo-antetul conține adresele IP ale mașinii sursă și destinație, de 32 de biți fiecare, numărul de protocol pentru TCP (6) și numărul de octeți al segmentului TCP (incluzând și antetul). Prin includerea pseudo-antetului în calculul sumei de control TCP se pot detecta pachetele care au fost preluate

eronat, dar procedând astfel, este negată însăși ierarhia protocolului, deoarece adresa IP aparține nivelului IP și nu nivelului TCP.



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

Câmpul Opțiuni a fost proiectat pentru a permite adăugarea unor facilități suplimentare neacoperite de antetul obișnuit. Cea mai importantă opțiune este aceea care permite fiecărei mașini să specifice încărcarea maximă de informație utilă TCP pe care este dispusă să o accepte. Utilizarea segmentelor de dimensiune mare este mai eficientă decât utilizarea segmentelor de dimensiune mica datorită amortizării antetului de 20 de octeți prin cantitatea mai mare de informație utilă. Cu toate acestea, este posibil ca mașini mai puțin performante să nu fie capabile să manevreze segmente foarte mari. În timpul inițializării conexiunii, fiecare parte anunță dimensiunea maximă acceptată și așteaptă de la partener aceeași informație. Câștigă cel mai mic dintre cele două numere. Dacă o mașină nu folosește această opțiune, cantitatea implicită de informație utilă este de 536 octeți. Toate mașinile din Internet trebuie să accepte segmente de dimensiune $536 + 20 = 556$ octeți. Dimensiunea maximă a segmentului nu trebuie să fie aceeași în cele două direcții.

În TCP conexiunile sunt stabilite utilizând „înțelegerea în trei pași”. Pentru a stabili o conexiune, una din părți - să spunem serverul - așteaptă în mod pasiv o cerere de conexiune prin execuția primitivelor LISTEN și ACCEPT, putând specifica o sursă anume sau nici o sursă în mod particular.

Cealaltă parte - să spunem clientul - execută o primitivă CONNECT, indicând

adresa IP și numărul de port la care dorește să se conecteze, dimensiunea maximă a segmentului TCP pe care este dispusă să o accepte și, opțional, o informație utilizator (de exemplu o parolă). Primitiva CONNECT trimite un segment TCP având bitul SYN poziționat și bitul ACK nepoziționat, după care așteaptă un răspuns.

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

Dacă există vreun proces care ascultă la acel port, segmentul TCP recepționat va fi dirijat către procesul respectiv. Acesta poate accepta sau refuza conexiunea. Dacă o acceptă, trimite înapoi expeditorului un segment de confirmare. În fig. 6-31(a) este reprezentată secvența de segmente TCP transferate în caz de funcționare normală. De notat că un segment SYN consumă un octet din spațiul numerelor de secvență, astfel încât confirmarea să poată fi făcută fără ambiguități.

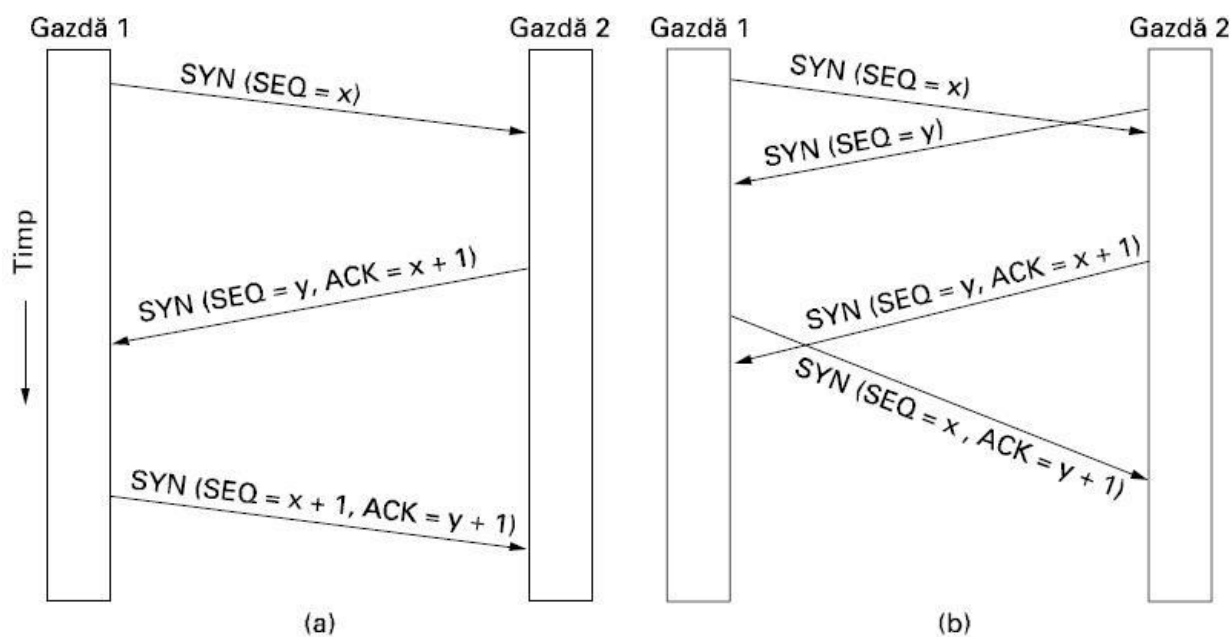


Fig. 6-31. (a) Stabilirea unei conexiuni TCP în cazul normal. (b) Coliziunea apelurilor.

Deși conexiunile TCP sunt bidirecționale, pentru a înțelege cum sunt desființate conexiunile, cel mai bine este să ni le imaginăm sub forma unei perechi de legături unidirecționale. Fiecare legătură unidirecțională este eliberată independent de perechea sa. Pentru eliberarea unei conexiuni, orice partener poate expedia un segment TCP având bitul FIN setat, lucru care indică faptul că nici o informație nu mai urmează să fie transmisă. Atunci când FIN-ul este confirmat, sensul respectiv de comunicare este efectiv oprit pentru noi date. Cu toate acestea, informația poate fi transferată în continuare, pentru un timp nedefinit, în celălalt sens. Conexiunea este desființată atunci când ambele direcții au fost oprite. În mod normal, pentru a elibera o conexiune sunt necesare patru segmente TCP: câte un FIN și un ACK pentru fiecare sens. Cu toate acestea, este posibil ca primul ACK și cel de-al doilea FIN să fie cuprinse în același segment reducând astfel numărul total la trei.

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

Administrarea ferestrei în TCP nu este direct legată de confirmări, așa cum se întâmplă la cele mai multe protocoale de nivel legătură de date. De exemplu, să presupunem că receptorul are un tampon de 4096 octeți, așa cum se vede în fig. 6-34. Dacă emițătorul transmite un segment de 2048 de octeți care este recepționat corect, receptorul va confirma segmentul. Deoarece acum tamponul acestuia din urmă mai are liberi doar 2048 octeți (până când aplicația șterge niște date din acest tampon), receptorul va anunța o fereastră de 2048 octeți începând de la următorul octet așteptat. Acum, emițătorul transmite alți 2048 octeți, care sunt confirmați, dar fereastra oferită este 0. Emițătorul trebuie să se oprească până când procesul aplicație de pe mașina receptoare a șters niște date din tampon, moment în care

TCP poate oferi o fereastră mai mare.

Atunci când fereastra este 0, în mod normal emițătorul nu poate să transmită segmente, cu două excepții. În primul rând, informația urgentă poate fi trimisă, de exemplu pentru a permite utilizatorului să oprească procesele rulând pe mașina de la distanță. În al doilea rând, emițătorul poate trimite un segment de un octet pentru a determina receptorul să renunțe următorul octet așteptat și dimensiunea ferestrei. Standardul TCP prevede în mod explicit această opțiune pentru a preveni interblocarea în cazul în care se întâmplă ca anunțarea unei ferestre să fie vreodată pierdută.

TCP Tranzacțional

Mai devreme în acest capitol am analizat apelul de proceduri la distanță ca modalitate de a implementa sistemele client-server. Dacă atât cererea, cât și răspunsul sunt suficient de mici încât să se potrivească în pachete simple și operația este idempotentă, UDP-ul poate fi ușor utilizat. Totuși, dacă aceste condiții nu sunt îndeplinite, utilizarea UDP-ului este mai puțin atractivă. De exemplu, dacă răspunsul este unul lung, atunci datagramele trebuie să fie secvențiate și trebuie inițiat un mecanism pentru a retransmite datagramele pierdute. De fapt, aplicației îi este cerut să reinventeze TCP-ul.

În mod cert, acest lucru nu este atractiv, dar nici utilizarea TCP-ului în sine nu este atractivă. Problema este eficiența. Secvența normală a pachetelor pentru a face un RPC peste TCP este prezentată în fig. 6-40(a). În cel mai bun caz sunt necesare nouă pachete. A se reține că acesta este cazul ideal. În cazul cel mai rău, cererea clientului și FIN-ul sunt confirmate separat, precum sunt răspunsul serverului și FIN-ul.

Întrebarea care apare imediat este dacă există vreo posibilitate de a combina eficiența RPC-ului folosind UDP (doar 2 mesaje) cu fiabilitatea TCP-ului.

Răspunsul este: aproape că da. Se poate realiza cu o variantă experimentală de TCP numită T/TCP (Transactional TCP, rom: TCP tranzacțional).

Ideea principală este aceea de a modifica secvența standard de inițializare a conexiunii astfel încât să permită, la nivel înalt, transferul de date în timpul

inițializării. Protocolul T/TCP este prezentat în fig. 6-40(b). Primul pachet al clientului conține bitul SYN, cererea în sine și FIN-ul. De fapt acesta spune: vreau să stabilesc o conexiune, aici sunt datele și am terminat. Când serverul primește cererea, caută sau calculează răspunsul și alege modul în care să răspundă.

Dacă răspunsul încapă într-un pachet, dă răspunsul din fig. 6-40(b), care spune: confirm FIN-ul tău, iată răspunsul, iar eu am terminat. Clientul confirmă apoi FIN-ul server-ului și protocolul ia sfârșit în (după) trei mesaje.

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

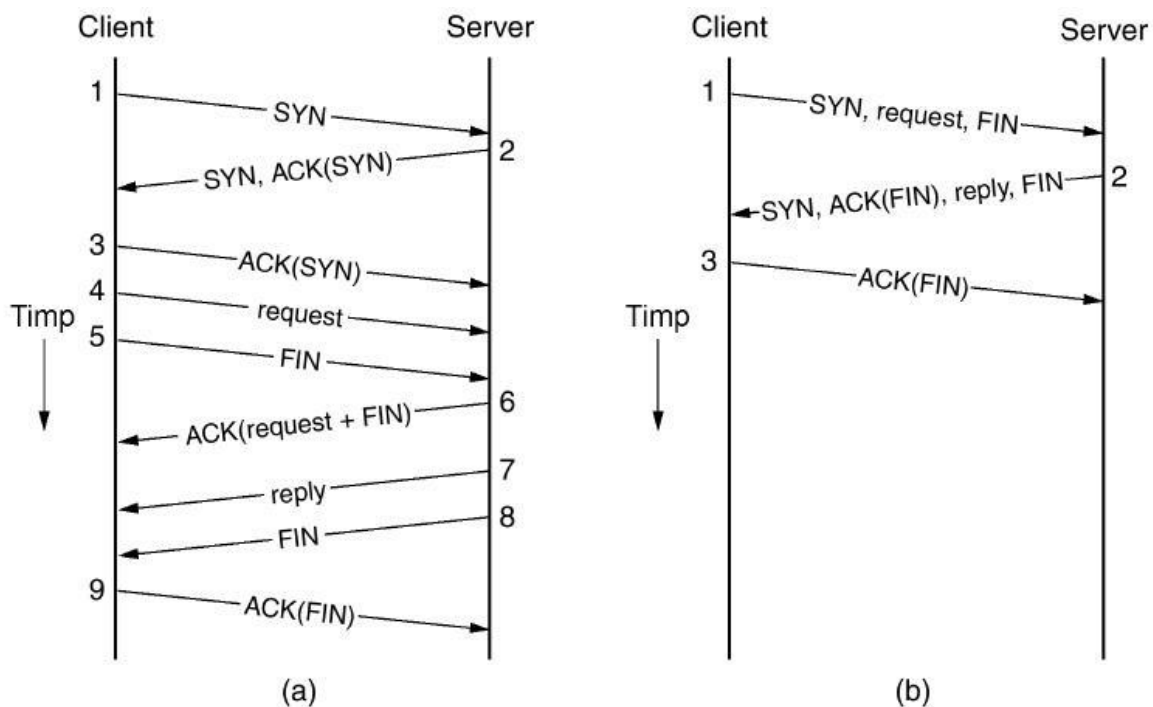


Fig. 6-40. (a) RPC folosind TCP clasic ; (b) RPC folosind T/TCP

28. DNS

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

Totuși, atunci când la rețea au fost conectate mii de stații de lucru, toți și-au dat seama că această abordare nu putea să funcționeze la nesfârșit. În primul rând dimensiunea fișierului ar deveni prea mare. Cu toate acestea și chiar mai important, conflictele de nume de sisteme gazdă ar apărea în permanență dacă nu ar fi administrate centralizat, ceva de negândit într-o rețea internațională de dimensiuni uriașe din cauza încărcării și a latenței. Pentru a rezolva aceste probleme, a fost inventat **DNS (Domain Name System - Sistemul numelor de domenii)**.

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

Foarte pe scurt, DNS este utilizat după cum urmează. Pentru a stabili corespondența dintre un nume și o adresă IP, programul de aplicație 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 și returnează adresa IP către resolver, care o returnează apelantului. Înarmat cu adresa IP, programul poate stabili o conexiune TCP cu destinația sau îi poate trimite pachete UDP.

Spațiul de nume DNS

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

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

Domeniile de pe primul nivel se împart în două categorii: generice și de țări. Domeniile generice sunt com (comercial), edu (instituții educaționale), gov (guvernul federal al SUA), int (organizații internaționale), mil (forțele armate ale SUA), net (furnizori Internet) și org (organizații nonprofit). Domeniile de țări includ o intrare pentru fiecare țară.

Fiecare domeniu este identificat prin calea în arbore de la el la domeniul (fără nume) rădăcină. Componentele sunt separate prin puncte (pronunțat „dot”). Astfel, departamentul tehnic de la Sun Microsystems ar putea fi eng.sun.com, în loc de numele în stil UNIX /com/sun/eng. De notat că această numire ierarhică face ca eng.sun.com să nu intre în conflict cu posibila utilizare a lui eng din eng.yale.edu, care ar putea fi folosit pentru departamentul de limba engleză de la Yale.

Numele de domenii pot fi absolute sau relative. Un nume absolut de domeniu se termină cu un punct (de exemplu, eng.sun.com.), în timp ce unul relativ nu. Numele relative trebuie interpretate în context pentru a le determina înțelesul adevărat. În ambele cazuri, un nume de domeniu se referă la un anumit nod din arbore și la toate nodurile de sub el.

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

Înregistrări de resurse

Fiecărui domeniu, fie că este un singur calculator gazdă, fie un domeniu de nivel superior, îi poate fi asociată o mulțime de înregistrări de resurse (resource records). Pentru un singur sistem gazdă, cea mai obișnuită înregistrare de resursă este chiar adresa IP, dar există multe alte tipuri de înregistrări de resurse. Atunci când un resolver trimite un nume de domeniu către un DNS, ceea ce va primi ca răspuns sunt înregistrările de resurse asociate acelui nume. Astfel, adevărata

funcție a DNS este să realizeze corespondența dintre numele de domenii și înregistrările de resurse.

O înregistrare de resursă este un 5-tuplu. Cu toate că, din rațiuni de eficiență, înregistrările de resurse sunt codificate binar, în majoritatea expunerilor ele sunt prezentate ca text ASCII, câte o înregistrare de resursă pe linie. Formatul pe care îl vom utiliza este următorul: **Nume_domeniu Timp_de_viață Clasă Tip**

Valoare

Nume_domeniu (domain_name) precizează domeniul căruia i se aplică această înregistrare. În mod normal există mai multe înregistrări pentru fiecare domeniu și fiecare copie a bazei de date păstrează informații despre mai multe domenii. Acest câmp este utilizat ca cheie de căutare primară pentru a satisface cererile. Ordinea înregistrărilor în baza de date nu este semnificativă.

Câmpul Timp_de_viață (time_to_live) dă o indicație despre cât de stabilă este înregistrarea. Informația care este foarte stabilă are asigurată o valoare mare, cum ar fi 86400 (numărul de secunde dintr-o zi). Informației instabile îi este atribuită o valoare mică, cum ar fi 60 (1 minut).

Al treilea câmp dintr-o înregistrare de resursă este Clasa (class). Pentru informațiile legate de Internet este tot timpul IN. Pentru alte informații pot fi folosite alte coduri, însă în practică acestea se întâlnesc rar.

Câmpul Tip (type) precizează tipul înregistrării. Cele mai importante tipuri sunt prezentate în fig. 7-2.

Tip	Semnificație	Valoare
SOA	Start autoritate	Parametrii pentru această zonă
A	Adresa IP a unui sistem gazdă	Întreg pe 32 de biți
MX	Schimb de poștă	Prioritate, domeniu dispus să accepte poștă electronică
NS	Server de Nume	Numele serverului pentru acest domeniu
CNAME	Nume canonic	Numele domeniului
PTR	Pointer	Pseudonim pentru adresa IP
HINFO	Descriere sistem gazdă	Unitate centrală și sistem de operare în ASCII
TXT	Text	Text ASCII neinterpretat

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

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

Servere de nume

Teoretic, un singur server de nume poate conține întreaga bază de date DNS și poate să răspundă tuturor cererilor. În practică, acest server poate fi atât de încărcat, încât să devină de neutilizat. În afară de aceasta, dacă se defectează, va fi afectat întregul Internet.

Pentru a evita problemele asociate cu existența unei singure surse de informație, spațiul de nume DNS este împărțit în zone care nu se suprapun. Fiecare zonă conține câte o parte a arborelui precum și numele serverelor care păstrează informația autorizată despre acea zonă. În mod normal, o zonă va avea un server de nume primar, care preia informația dintr-un fișier de pe discul propriu și unul sau mai multe servere de nume secundare, care iau informațiile de pe serverul primar. Pentru a îmbunătăți fiabilitatea, unele servere pentru o zonă pot fi plasate în afara zonei.

Plasarea limitelor unei zone este la latitudinea administratorului ei. Această decizie este luată în mare parte bazându-se pe câte servere de nume sunt dorite și unde să fie plasate.

Atunci când un resolver are o cerere referitoare la un nume de domeniu, el transferă cererea unuia din serverele locale de nume. Dacă domeniul căutat este sub jurisdicția serverului de nume, cum ar fi ai.cs.yale.edu, care este sub cs.yale.edu, el reîntoarce înregistrări de resurse autorizate. O înregistrare autorizată (authoritative record) este cea care vine de la autoritatea care administrează înregistrarea și astfel este întotdeauna corectă. Înregistrările autorizate se deosebesc de înregistrările din memoria ascunsă, care pot fi expirate.

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

merge din subdomeniu în subdomeniu pana se gaseste înregistrarea 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. Totuși, această informație 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 viață prea lungă. Acesta este motivul pentru care câmpul `Timp_de_viață` este inclus în fiecare înregistrare de resursă. El informează serverele de nume aflate la distanță cât timp să mențină înregistrările în memoria ascunsă. Dacă o anumită mașină are de ani de zile aceeași adresă IP, această informație ar putea fi păstrată timp de o zi. Pentru informații mai volatile este mai sigur ca înregistrările să fie eliminate după câteva secunde sau un minut.

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

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

29. Posta electronica

Arhitectură și servicii

În această secțiune vom furniza o prezentare de ansamblu a ceea ce pot face sistemele de poștă electronică și cum sunt ele organizate. Aceste sisteme constau de obicei din două subsisteme: agenții-utilizator, care permit utilizatorilor să citească și să trimită scrisori prin poșta electronică și agenții de transfer de mesaje, care transportă mesajele de la sursă la destinație. Agenții-utilizator sunt programe locale, care furnizează o metodă de a interacționa cu sistemul de e-mail bazată pe comenzi, meniuri sau grafică. Agenții de transfer de mesaje sunt, de regulă, demoni de sistem, adică procese care se execută în fundal. Sarcina lor este să transfere mesajele prin sistem.

În general, sistemele de poștă electronică pun la dispoziție cinci funcții de bază. Să aruncăm o privire asupra lor.

Compunerea se referă la procesul de creare a mesajelor și a răspunsurilor. Deși pentru corpul mesajului poate fi folosit orice editor de texte, sistemul însuși poate acorda asistență la adresare și la completarea numeroaselor câmpuri antet atașate fiecărui mesaj. De exemplu, când se răspunde la un mesaj, sistemul poate extrage adresa inițiatorului din mesajul primit și o poate insera automat în locul potrivit din cadrul răspunsului.

Transferul se referă la deplasarea mesajului de la autor la receptor. În mare, aceasta necesită stabilirea unei conexiuni la destinația, sau la o mașină intermediară, emiterea mesajului și eliberarea conexiunii. Sistemul de poștă ar trebui să facă acest lucru singur, fără a deranja utilizatorul.

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

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

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

Majoritatea sistemelor permit utilizatorilor să-și creeze **cutii poștale (mailboxes)** pentru a păstra mesajele sosite. Sunt necesare comenzi de creare și distrugere a cutiilor poștale, de inspectare a conținutului acestora, de inserare și de ștergere de mesaje din cutii poștale ș.a.m.d.

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

Alte caracteristici evolute sunt copii la indigo, poștă de prioritate mare, poștă secretă (criptată), receptori alternativi, dacă cel primar nu este disponibil, și posibilitatea de a permite secretarelor să se ocupe de poșta primită de șefii lor.

O idee fundamentală în toate sistemele moderne de e-mail este distincția dintre **plic** și **conținutul** său. Plicul încapsulează mesajul. Conține toată informația necesară pentru transportul mesajului, cum ar fi destinația, adresa, prioritatea, nivelul de securitate, toate acestea fiind distincte de mesajul în sine.

Agenții de transfer de mesaje folosesc plicul pentru rutare (dirijare), așa cum face și oficiul poștal. Mesajul din interiorul plicului conține două părți: **antetul** și **corpul**. Antetul conține informație de control pentru agenții utilizator. Corpul mesajului se adresează în întregime utilizatorului uman.

Trimiterea poștei electronice

Pentru a trimite un mesaj prin poșta electronică, un utilizator trebuie să furnizeze mesajul, adresa destinație, și eventual alți câțiva parametri. Mesajul poate fi produs cu un editor de texte de sine stătător, cu un program de procesare de text sau, eventual, cu un editor de texte specializat, construit în interiorul agentului utilizator. Adresa de destinație trebuie să fie într-un format cu care agentul utilizator să poată lucra. Mulți agenți-utilizator solicită adrese de forma utilizator@adresă-dns.

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

Citirea poștei electronice

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

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

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

Formatele mesajelor

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

La o utilizare normală, agentul-utilizator construiește un mesaj și îl transmite agentului de transfer de mesaje, care apoi folosește unele dintre câmpurile antet pentru a construi plicul efectiv, o combinație oarecum demodată de mesaj și plic.

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

O linie conținând Received: este adăugată de fiecare agent de transfer de mesaje de pe traseu. Linia conține identitatea agentului, data și momentul de

timp la care a fost primit mesajul și alte informații care pot fi utilizate pentru găsirea defecțiunilor în sistemul de dirijare.

Câmpul Return-Path: este adăugat de agentul final de transfer de mesaje și are în intenție să indice cum se ajunge înapoi la transmițător. În teorie, această informație poate fi adunată din toate antetele Received: (cu excepția numelui cutiei poștale a transmițătorului), dar rareori este completată așa și de obicei conține chiar adresa transmițătorului. 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 și să definească reguli de codificare pentru mesajele non-ASCII. Deoarece respectă RFC 822, mesajele MIME pot fi trimise utilizând programele și protocoalele de poștă existente. Tot ceea ce trebuie modificat sunt programele de transmitere și recepție, pe care utilizatorii le pot face ei înșiși.

MIME definește cinci noi antete de mesaje, așa 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 poștă)

În cadrul Internetului poșta electronică este livrată prin stabilirea de către mașina sursă a unei conexiuni TCP la portul 25 al mașinii de destinație. La acest

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

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

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

Chiar dacă protocolul SMTP este bine definit, mai pot apărea câteva probleme. O problemă este legată de lungimea mesajelor. Unele implementări mai vechi nu pot să lucreze cu mesaje mai mari de 64KB. O altă problemă se referă la expirări de timp (timeout). Dacă acestea diferă pentru server și client, unul din ei poate renunța, în timp ce celălalt este încă ocupat, întrerupând conexiunea în mod neașteptat. În sfârșit, în unele situații, pot fi lansate schimburi infinite de mesaje.

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

Livrarea finală

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

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

O soluție este ca agentul de transfer de mesaje de pe o mașină ISP să accepte e-mail-ul pentru clienții săi și să-l stocheze în cutiile lor poștale pe o mașină a ISP-ului. Din moment ce acest agent poate fi conectat la rețea tot timpul, se poate trimite e-mail 24 de ore pe zi.

POP3

Din nefericire, această soluție dă naștere altei probleme: cum își ia utilizatorul e-mail-ul de la agentul de transfer de mesaje al ISP-ului? Soluția acestei probleme este crearea unui alt protocol care să permită agenților de transfer mesaje (aflați pe calculatoarele clienților) să contacteze agentul de transfer mesaje (de pe o mașină ISP) și să facă posibilă copierea e-mail-ului de la ISP la utilizator. Un astfel de protocol este **POP3 (Post Office Protocol Version 3-Protocol de poștă, versiunea 3)**.

POP3 începe când utilizatorul pornește programul cititor de poștă (mail reader). Acesta sună la ISP (în caz că nu există deja o conexiune) și stabilește o conexiune TCP cu agentul de transfer de mesaje, prin portul 110. Odată ce conexiunea a fost stabilită, protocolul POP3 trece succesiv prin următoarele trei stări:

1. Autorizare se referă la admiterea utilizatorului în sistem (login).
2. Tranzacționare tratează colectarea e-mail-urilor și marcarea lor pentru ștergere din cutia poștală.
3. Actualizare se ocupă cu ștergerea efectivă a mesajelor.

Deși 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 și golesc cutia poștală. Ca urmare, practic singura copie rămâne înregistrată pe discul utilizatorului. Dacă acesta se strică, toate e-mail-urile pot fi pierdute definitiv.

IMAP

Pentru un utilizator cu un singur cont de e-mail, la un singur ISP, care este tot timpul accesat de la un singur PC, POP3 este bun și larg folosit datorită simplității și robusteții sale. Totuși, există în industria calculatoarelor un adevăr bine înrădăcinat, acela că imediat ce un lucru funcționează bine, cineva va începe să ceară mai multe facilități (și să aibă mai multe probleme). Asta s-a întâmplat și cu e-mail-ul. De exemplu, multă lume are un singur cont de e-mail la serviciu sau la școală și vrea săl acceseze de pe PC-ul de acasă, de pe calculatorul portabil în călătoriile de afaceri și din Internet café-uri în vacanțe. Cu toate că POP3 permite asta, din moment ce în mod normal el descarcă toate mesajele la fiecare conectare, rezultatul constă în răspândirea e-mail-ului utilizatorului pe mai multe mașini, mai mult sau mai puțin la întâmplare, unele dintre ele nefiind ale utilizatorului.

Acest dezavantaj a dat naștere unei alternative a protocolului de livrare finală, **IMAP (Internet Message Access Protocol – Protocol pentru accesul mesajelor în Internet)**. Spre deosebire de POP3, care în mod normal presupune că utilizatorul își va goli căsuța poștală la fiecare conectare și va lucra deconectat de la rețea (off-line) după aceea, IMAP presupune că tot email-ul va rămâne pe server oricât de mult, în mai multe căsuțe poștale.

IMAP prevede mecanisme extinse pentru citirea mesajelor sau chiar a părților de mesaje, o facilitate folositoare când se utilizează un modem încet pentru citirea părții textuale a unui mesaj cu mai multe părți audio și video de mari dimensiuni. Întrucât premisa de folosire este că mesajele nu vor fi transferate pe calculatorul utilizatorului în vederea stocării permanente, IMAP asigură mecanisme pentru crearea, distrugerea și manipularea mai multor cutii poștale pe server. Astfel, un utilizator poate păstra o cutie poștală pentru fiecare corespondent și poate muta aici mesajele din inbox după ce acestea au fost citite.

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

Stilul general al protocolului IMAP este similar cu cel al POP3-ului, cu excepția faptului că există zeci de comenzi. Serverul IMAP ascultă pe portul 143. În fig. 7-17 este prezentată o comparație între POP3 și IMAP. E bine de notat, totuși, că nu toate ISP-urile oferă ambele protocoale și că nu toate programele de e-mail le suportă pe amândouă. Așadar, atunci când alegeți un program de e-mail, este important să aflați ce protocoale suportă și să vă asigurați că ISP-ul oferă cel puțin unul din ele.

Caracteristica	POP3	IMAP
Unde este definit protocolul	RFC 1939	RFC 2060
Portul TCP folosit	110	143
Unde este stocat e-mail-ul	PC-ul utilizatorului	Server
Unde este citit e-mail-ul	Off-line	On-line
Timpul necesar conectării	Mic	Mare
Folosirea resurselor serverului	Minimă	Intensă
Mai multe cutii poștale	Nu	Da
Cine face copii de siguranță la cutiile poștale	Utilizatorul	ISP-ul
Bun pentru utilizatorii mobili	Nu	Da
Controlul utilizatorului asupra scrisorilor preluate	Mic	Mare
Descărcare parțială a mesajelor	Nu	Da
Volumul discului alocat (disk quota) este o problemă	Nu	Ar putea fi în timp
Simplu de implementat	Da	Nu
Suport răspândit	Da	În creștere

Fig. 7-17. O comparație între POP3 și IMAP.

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 mașina serverului, deși această procedură nu este cerută formal. Avantajul de a folosi TCP este că nici programele de navigare și nici serverele nu trebuie să se preocupe de mesajele pierdute, mesajele duplicate, mesajele lungi, sau mesajele de confirmare. Toate aceste aspect sunt tratate de implementarea TCP.

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

Această observație a dus la apariția HTTP 1.1, care suportă conexiuni persistente. Cu ele, este posibilă stabilirea unei conexiuni TCP, trimiterea unei cereri și

obținerea unui răspuns, apoi trimiterea unor cereri adiționale și obținerea de răspunsuri adiționale. Prin distribuirea pornirii și eliberării unei conexiuni TCP peste mai multe cereri, supraîncărcarea relativă datorată TCP-ului este mult mai mică pe fiecare cerere. Este de asemenea posibilă trimiterea cererilor prin mecanismul pipeline, adică trimiterea cererii 2 înainte ca răspunsul la cererea 1 să fi sosit.

Metode

Cu toate că HTTP a fost proiectat pentru utilizarea în Web, el a fost creat intenționat mai general decât era necesar în perspectiva aplicațiilor orientate pe obiecte. Pentru aceasta sunt suportate operațiile, denumite metode, care fac mai mult decât cele care doar cer o pagină Web. Această considerație generală a permis apariția SOAP. Fiecare cerere constă din una sau mai multe linii de text

ASCII, în care primul cuvânt din prima linie este numele metodei cerute. Metodele încorporate sunt listate în fig. 7-41. Pentru accesarea unor obiecte generale, metode adiționale 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 înțelegem obiect, în cel mai general caz, dar în practică de obicei doar un fișier). Pagina este codată corespunzător în MIME. Marea majoritate a cererilor către servere Web sunt metode GET. Forma uzuală a metodei GET este: **GET fișier HTTP-1.1**, unde fișier denumește resursa (fișierul) ce va fi adusă, și 1.1 este versiunea de protocol utilizat.

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

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

Similară metodei PUT este **metoda POST**. Și ea conține un URL, dar în loc să înlocuiască date existente, noile date se vor adăuga într-un mod generalizat. De exemplu, se poate transmite un mesaj la un grup de știri sau adăuga un fișier la un sistem de informare în rețea. În practică, nici PUT și nici POST nu sunt utilizate prea mult.

Metoda DELETE realizează ce era de așteptat: ștergerea unei pagini. Ca și la PUT, autentificarea și drepturile de acces joacă aici un rol important. Nu există nici o garanție asupra succesului operației DELETE, deoarece chiar dacă serverul dorește să execute ștergerea, fișierul poate să aibă attribute care să interzică serverului HTTP să îl modifice sau să îl șteargă.

Metoda TRACE este pentru verificarea corectitudinii. Ea cere serverului să trimită înapoi cererea. Această metodă este utilă când cererile nu sunt procesate corect și clientul vrea să știe ce fel de cerere a ajuns de fapt la server.

Metoda CONNECT nu este utilizată în prezent. Este rezervată pentru utilizări ulterioare.

Metoda OPTIONS asigură o modalitate pentru client de a interoga serverul despre proprietățile acestuia sau despre cele ale unui anumit fișier.

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

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

Cod	Semnificație	Exemple
1xx	Informație	100 = serverul acceptă tratarea cererii de la client
2xx	Succes	200 = cerere reușită; 204 = nu există conținut
3xx	Redirecționare	301 = pagină mutată; 304 = pagina din memoria ascunsă este încă validă
4xx	Eroare la client	403 = pagină interzisă; 404 = pagina nu a fost găsită
5xx	Eroare la server	500 = eroare internă la server; 503 = încearcă mai târziu

Fig. 7-42. Grupuri de răspunsuri ale codurilor de stare.

Antete de mesaje

Linia de cerere (de exemplu linia cu metoda GET) poate fi urmată de linii adiționale cu mai multe informații. Acestea poartă numele de antete de cerere. Această informație poate fi comparată cu parametrii unui apel de procedură. Răspunsurile pot avea de asemenea antete de răspuns. Anumite antete pot fi folosite în orice sens. O selecție a celor mai importante este dată în fig. 7-43.

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 transmitătorul 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-Type	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 :

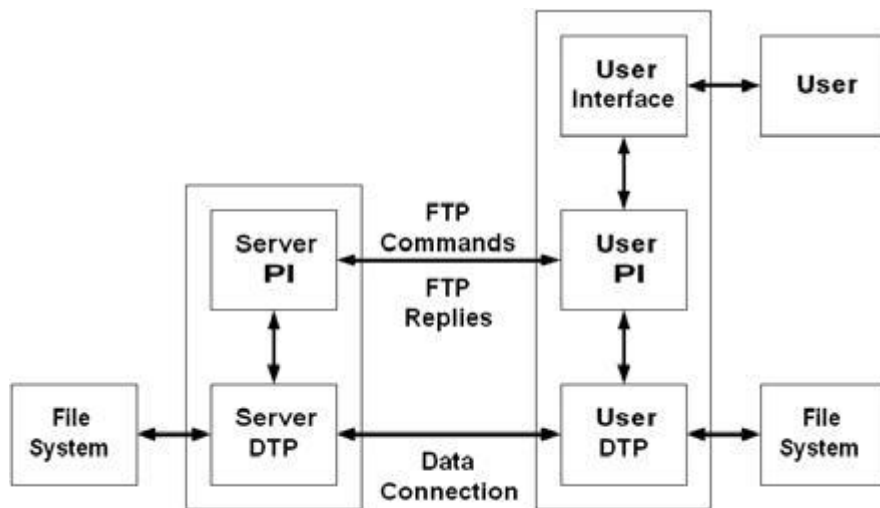
Standard pentru transfer de fisiere (RFC959). Precede TCP/IP, dar a fost adaptat ulterior la TCP/IP. Este un protocol general, cu urmatoarele caracteristici:

- Independent de SO si de hardware
- Transfera fisiere oarecare
- Gestioneaza drepturile asupra fisierelelor 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 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.

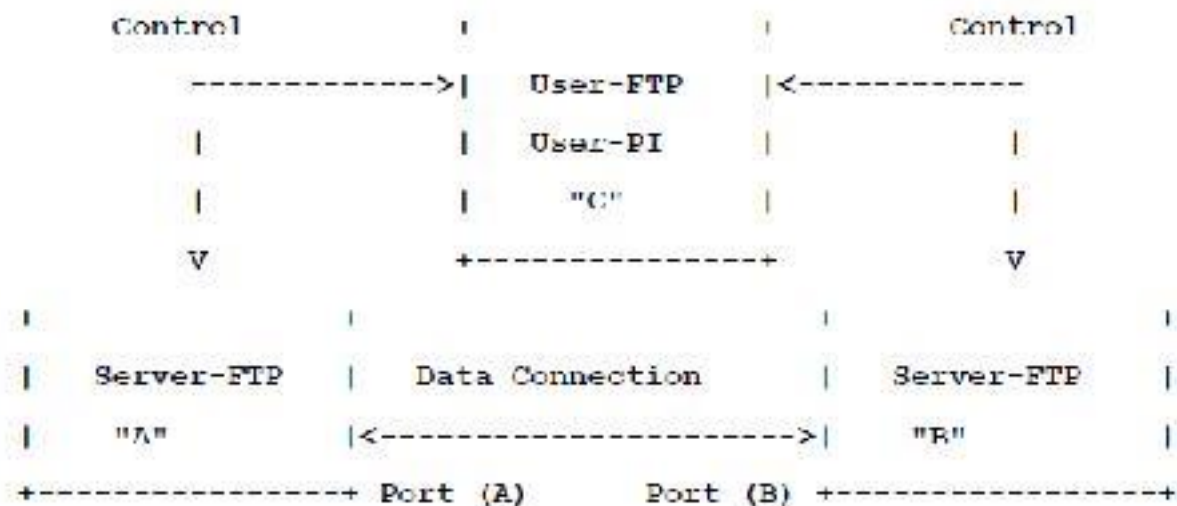
- Pasiva – clientul initiaza conexiunea, clientul cere serverului sa asculte la o adresa si un port (care nu este portul sau standard); serverul comunica adresa si portul (ex. port 4023, adresa 172.16.62.36);

Client: PASV

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

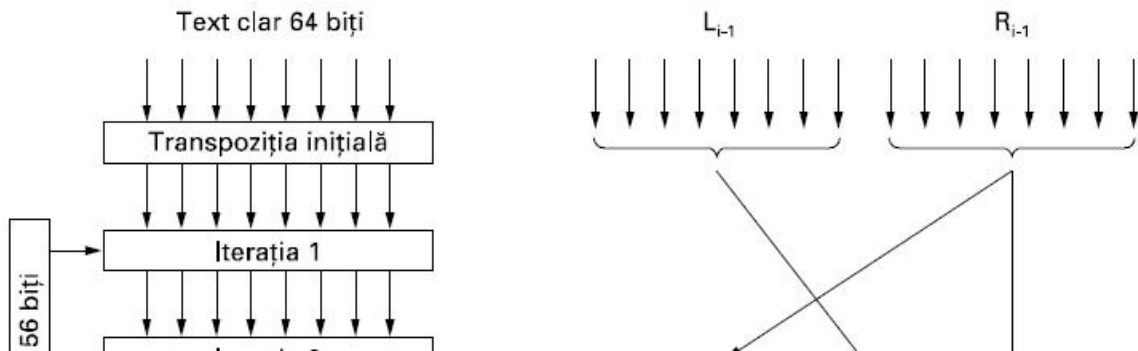
Interactiune server-server

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 biți, rezultând blocuri de 64 de biți de text cifrat.

Algoritmul, care este parametrizat cu o cheie de 56 de biți, are 19 runde distincte. Prima rundă este o transpoziție independentă de cheie, aplicată asupra textului clar de 64 de biți. Ultima rundă este exact inversa acestei transpoziții. Penultima rundă schimbă cei mai din stânga 32 de biți cu cei mai din dreapta 32 de biți. Cele 16 runde rămase sunt funcțional identice dar sunt parametrizate de funcții de cheie diferite. Algoritmul a fost proiectat pentru a permite ca decriptarea să se facă cu aceeași cheie ca și criptarea, o proprietate necesară în orice algoritm cu cheie secretă. Pașii sunt parcurși în ordine inversă.

Funcționarea unuia dintre pașii intermediari este ilustrată în fig. 8-7(b). Fiecare rundă ia două intrări de 32 de biți și produce două ieșiri de 32 de biți. Ieșirea din stânga este o simplă copie a intrării din dreapta. Ieșirea din dreapta rezultă în urma unui SAU exclusiv (XOR) bit cu bit între intrarea din stânga și o funcție depinzând de intrarea din dreapta și de o cheie pentru această rundă, K_i . Toată complexitatea rezidă în această funcție.

Funcția constă din patru pași, parcurși în secvență. În primul rând, este construit un număr de 48 de biți, E, prin expandarea celor 32 de biți ai lui R_{i-1} în concordanță cu o regulă de transpoziție fixă și de duplicare. În al doilea rând, E și K_i sunt combinate prin XOR. Ieșirea este apoi împărțită în opt grupuri de câte 6 biți și fiecare dintre acestea este introdus într-o cutie S diferită. Fiecare dintre cele 64 de intrări posibile într-o cutie S este pusă în corespondență cu o ieșire de 4 biți. În final, acești 8×4 biți sunt trecuți printr-o cutie P.

În fiecare din cele 16 iterații este folosită o cheie diferită. Înainte de începerea algoritmului este aplicată o transpoziție de 56 de biți asupra cheii. Chiar înainte de începerea fiecărei iterații, cheia este partiționată în două unități de câte 28 de biți, fiecare dintre ele este rotită la stânga cu un număr de biți depinzând de numărul iterației. K_i este derivat din această cheie rotită prin aplicarea unei transpoziții pe 56 de biți asupra ei. La fiecare rundă este extrasă și permutată o altă submulțime de 48 de biți din cei 56 de biți.

Triplu DES

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

În prima rundă textul clar este în mod obișnuit criptat cu K_1 . În a doua rundă, este rulat DES în mod de decriptare, folosind cheia K_2 . În final, este efectuată o altă criptare cu K_1 .

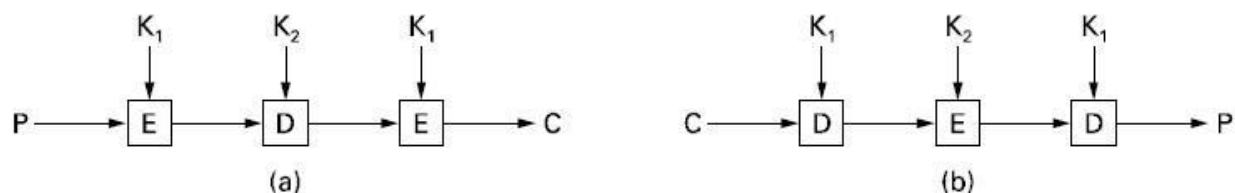


Fig. 8-8. (a) Tripla criptare folosind DES. (b) Decriptarea.

Această proiectare dă naștere 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 și cei mai paranoici criptografi admit că 112 biți sunt suficienți pentru aplicațiile comerciale de rutină, la momentul actual. A extinde la 168 de biți înseamnă a adăuga o supraîncărcare inutilă pentru gestiunea și transportul unei alte chei cu un câștig real redus.

Motivul pentru criptare, decriptare și apoi iar criptare este compatibilitatea cu produsele existente ce folosesc sisteme DES cu o singură cheie. Atât funcția de criptare cât și cea de decriptare sunt corespondențe între mulțimi de numere pe 64 de biți. Din punct de vedere criptografic, cele două corespondențe sunt la fel de puternice. Totuși, folosind EDE în locul EEE, un calculator ce utilizează tripla criptare poate comunica cu unul ce folosește criptarea simplă, folosind $K_1=K_2$.

AES – Advanced Encryption Standard (Rijndael)

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

Ca și DES, Rijndael folosește substituție și permutări, ca și runde multiple. Numărul de runde depinde de mărimea cheii și de mărimea blocului, fiind 10 pentru o cheie de 128 de biți cu blocuri de 128 biți și mărindu-se până la 14 pentru cheia cu cea mai mare dimensiune și blocul cu cea mai mare dimensiune.

Totuși, spre deosebire de DES, toate operațiile sunt la nivel de octet, pentru a permite implementări eficiente hardware și 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(state);        /* 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

Funcția rijndael are trei parametri. Aceștia sunt: plaintext, un vector de 16 octeți conținând datele de intrare, ciphertext, un vector de 16 octeți în care va fi introdus rezultatul cifrat, și key, cheia de 16 octeți. Pe durata calculelor, starea curentă a datelor e păstrată într-un vector de octeți, state, a cărui mărime este $NROWS \times NCOLS$. Pentru blocuri de 128 de octeți, acest vector este de 4×4 octeți. Întregul bloc de date de 128 de biți poate fi stocat în 16 octeți.

Vectorul state este inițializat cu textul clar și este modificat la fiecare pas al calculului. În anumiți pași, este realizată substituția octet-cu-octet. În alții, octeții sunt permutați în interiorul vectorului. Sunt folosite și alte transformări. La sfârșit, conținutul lui state reprezintă textul cifrat.

Codul începe prin expandarea cheii în 11 vectori de aceeași lungime cu starea. Aceștia sunt memorați în rk, care este un vector de structuri, fiecare structură conținând un vector stare. Unul dintre aceștia va fi folosit la începutul calculului și ceilalți 10 vor fi folosiți în timpul celor 10 runde, câte unul în fiecare rundă. Calculul cheilor de rundă din cheia de criptare este prea complicat pentru a

fi prezentat aici. Este de ajuns să spunem că cheile de rundă sunt produse prin rotiri repetate și aplicarea de operații XOR asupra unor grupuri de biți 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 octeți în coloana 0, următorii patru octeți în coloana 1 și așa mai departe. Atât coloanele cât și liniile sunt numerotate pornind de la 0, deși 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 aflați în state este înlocuit cu rezultatul aplicării operației XOR asupra sa și asupra octetului corespunzător din $rk[0]$.

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

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

Pasul 3 amestecă fiecare coloană independent de celelalte. Această amestecare este realizată prin înmulțire de matrice, în care noua coloană este produsul dintre vechea coloană și o matrice constantă, multiplicarea fiind realizată folosind câmpul finit Galois, $GF(2^8)$. Deși acest lucru poate părea complicat, există un algoritm care permite fiecărui element al noii coloane să fie calculat folosind două căutări în tabele și trei operații XOR.

În fine, pasul 4 aplică operația XOR pentru cheia din runda curentă și vectorul stare.

Deoarece fiecare pas e reversibil, decriptarea poate fi realizată prin rularea algoritmului de la coadă la cap. Oricum, este posibilă și 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 și pentru o viteză foarte mare. O bună implementare software pe o mașină la 2 GHz ar trebui să atingă o rată de criptare de 700 Mbps, ceea ce este suficient de rapid pentru a cripta peste 100 de fișiere 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, mulți cercetători au lucrat din greu la acest subiect și au fost deja publicați câțiva algoritmi. O metodă bună a fost descoperită de un grup de la MIT (Rivest et. al, 1978). Ea este cunoscută prin inițialele numelor celor trei descoperitori (Rivest, Shamir, Adelman): **RSA**. Metoda a supraviețuit tuturor încercărilor de a o sparge timp de mai mult de un sfert de secol și este considerată foarte puternică. Multe aplicații de securitate se bazează pe ea. Dezavantajul major al acesteia este că necesită chei de cel puțin 1024 de biți pentru o securitate bună (spre deosebire de 128 biți pentru algoritmi 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 folosește această metodă:

1. Se aleg două numere prime, p și q , (de obicei de 1024 biți).
2. Se calculează $n = p \times q$ și $z = (p - 1) \times (q - 1)$.
3. Se alege un număr relativ prim cu z și este notat cu d .
4. Se găsește e astfel încât $e \times d = 1 \bmod z$.

Cu acești parametri calculați în avans, suntem gata să începem criptarea. Împărțim textul clar (privit ca șir de biți) 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 biți, unde k este cel mai mare număr întreg pentru care inegalitatea $2k < n$ este adevărată.

Pentru a cripta mesajul P , se calculează $C = P^e \pmod{n}$. Pentru a decripta C , se calculează $P = C^d \pmod{n}$. Se poate demonstra că pentru toți P din intervalul specificat, criptarea și decriptarea sunt funcții inverse una alteia. Pentru a realiza criptarea este nevoie de e și n . Pentru a realiza decriptarea este nevoie de d și n . De aceea, cheia publică constă din perechea (e, n) iar cheia privată 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 și q , iar din acestea pe z . Cu z și e cunoscuți, criptanalistul îl poate calcula pe d folosind algoritmul lui Euclid. Din fericire, matematicienii au încercat să factorizeze numere mari de cel puțin 300 de ani și experiența 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 ieșire identice. De aceea este necesară o anumită formă de înlănțuire pentru criptarea datelor. Totuși, în practică, cele mai multe sisteme bazate pe RSA folosesc criptografia cu cheie publică în principal pentru distribuirea cheilor de sesiune de unică folosință 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 distribuția de chei.