

Rețele de calculatoare

Principii

Radu-Lucian Lupșa

Aceasta este ediția electronică a cărții *Rețele de calculatoare*, publicată la Casa Cărții de Știință, în 2008, ISBN: 978-973-133-377-9.

Drepturile de autor aparțin subsemnatului, Radu-Lucian Lupșa.

Subsemnatul, Radu-Lucian Lupșa, acord oricui dorește dreptul de a copia conținutul acestei cărți, integral sau parțial, cu condiția atribuirii corecte autorului și a păstrării acestei notițe.

Cartea poate fi descărcată gratuit de la adresa
<http://www.cs.ubbcluj.ro/~rlupsa/works/retele.pdf>

Cuprins

Principii

Cuprins	5
Prefață	13
1 Introducere	15
1.1 Serviciile oferite de rețea	15
1.2 Principalele elemente ale unei rețele de calculatoare	20
1.3 Premise generale în elaborarea și implementarea protocoalelor în rețele	22
2 Noțiuni de teoria informației	25
2.1 Problema codificării informației pentru un canal discret	26
2.2 Coduri cu proprietatea de prefix	29
2.2.1 Reprezentarea arborescentă a codurilor prefix	29
2.2.2 Decodificarea în cazul codurilor prefix	31
2.2.3 Lungimile cuvintelor unui cod prefix	33
2.3 Coduri optime	39
2.3.1 Cantitatea de informație	40
2.3.2 Lungimea medie a cuvintelor de cod	41
2.3.3 Generarea codului optim prin algoritmul lui Huffman	44
2.3.4 Compresia fișierelor	50
2.4 Coduri detectoare și corectoare de erori	51
2.4.1 Modelul erorilor	52
2.4.2 Principiile codurilor detectoare și corectoare de erori	53
2.4.3 Câteva coduri detectoare sau corectoare de erori	55
2.4.3.1 Bitul de paritate	55
2.4.3.2 Paritate pe linii și coloane	55
2.4.3.3 Coduri polinomiale	56
2.4.4 Coduri detectoare și corectoare de erori în alte domenii	57

3 Nivelul fizic	59
3.1 Problema transmisiei informației la nivelul fizic	59
3.2 Transmiterea semnalelor	60
3.2.1 Modificările suferite de semnale	60
3.2.2 Analiza transmiterii semnalelor cu ajutorul transformatei Fourier	62
3.3 Codificarea informației prin semnale continue	65
3.3.1 Scheme de codificare	65
3.3.2 Modulația	68
3.3.3 Multiplexarea în frecvență	71
3.3.4 Capacitatea maximă a unui canal de comunicație	71
3.4 Transmisia prin perechi de conductoare	72
3.4.1 Construcția cablului	72
3.4.2 Proprietăți ale mediului	74
3.4.3 Legătură magistrală	75
3.4.4 Considerente practice	76
3.5 Transmisia prin unde radio	77
3.5.1 Propagarea undelor	78
3.5.1.1 Polarizarea	78
3.5.1.2 Absorbția și reflexia	79
3.5.1.3 Difrakția	79
3.5.1.4 Interferența undelor	80
3.5.1.5 Divergența undelor	80
3.5.2 Antene	80
3.5.2.1 Directivitatea	81
3.5.2.2 Polarizarea	83
3.5.2.3 Tipuri de antene	83
3.5.3 Raza de acțiune a unei legături radio	83
3.5.3.1 Obstacolele	83
3.5.3.2 Linia orizontului	84
3.5.3.3 Utilizarea sateliților artificiali ai Pământului	84
3.5.3.4 Zgomotul	85
3.5.3.5 Scăderea puterii cu distanța	86
3.5.3.6 Emisia direcționată și polarizată	86
3.5.4 Spectrul radio și alocarea lui	86
3.5.5 Particularități ale sistemelor de comunicație prin radio	88
3.5.5.1 Topologia legăturii	88
3.5.5.2 Fiabilitatea	89
3.5.5.3 Securitatea	89
3.6 Transmisia optică	89
3.6.1 Construcția mediului	90
3.6.1.1 Conectarea fibrelor optice	91
3.6.2 Propagarea semnalului optic	91
3.6.2.1 Moduri de propagare	91

3.6.2.2	Caracteristici ale mediului	92
3.6.2.3	Multiplexarea în lungimea de undă	92
3.6.3	Considerente practice	93
4	Nivelul legăturii de date	95
4.1	Detectarea și corectarea erorilor	96
4.2	Controlul accesului la mediu	97
4.2.1	Protocoale bazate pe asigurarea unui interval exclusiv de emisie	98
4.2.2	Protocoale bazate pe coliziuni și retransmitere	99
4.2.3	Protocoale mixte	101
4.3	Retransmiterea pachetelor pierdute	102
4.3.1	Principiul confirmărilor pozitive și retransmiterilor	103
4.3.2	Trimiterea în avans a mai multor pachete	108
4.3.3	Spațiul numerelor de confirmare	109
4.4	Controlul fluxului	114
4.4.1	Cereri de suspendare și de continuare	115
4.4.2	Mecanismul pas cu pas	115
4.4.3	Mecanism combinat cu retransmiterea pachetelor pierdute	116
4.5	Multiplexarea în timp	117
5	Nivelul rețea și nivelul transport	119
5.1	Retransmiterea datelor de către nodurile intermediare	120
5.1.1	Retransmiterea în rețele bazate pe datagrame	122
5.1.2	Retransmiterea în rețele bazate pe conexiuni	122
5.2	Algoritmi de dirijare	125
5.2.1	Calculul drumurilor cu informații complete despre graful rețelei	127
5.2.2	Calculul drumurilor optime prin schimb de informații de distanță	128
5.2.3	Dirijarea ierarhică	136
5.2.4	Metode particulare de dirijare	139
5.2.4.1	Inundarea	139
5.2.4.2	Învățarea rutelor din adresele sursă ale pachetelor	140
5.2.5	Metode de difuziune	140
5.3	Funcționarea la trafic ridicat	141
5.3.1	Alegerea pachetelor de transmis	142
5.3.2	Controlul congestiei	143
5.3.3	Formarea (limitarea) traficului	144
5.3.4	Rezervarea resurselor	145
5.4	Nivelul transport	146
5.5	Interconectarea rețelelor	147
6	Metode și protocoale criptografice	149
6.1	Asigurarea confidențialității	151
6.1.1	Introducere	151
6.1.2	Refolosirea cheilor	154
6.1.3	Problema spargerii unui cifru	155

6.1.4	Algoritmi de criptare utilizați în practică	157
6.1.5	Criptografie asimetrică (cu cheie publică)	163
6.1.5.1	Utilizarea criptografiei asimetrice	164
6.2	Autentificarea mesajelor	165
6.2.1	Funcții de dispersie criptografice	166
6.2.1.1	Utilizarea funcțiilor de dispersie	167
6.2.2	Funcții de dispersie cu cheie	168
6.2.3	Semnătura digitală	169
6.2.4	Verificarea prospețimii mesajelor	171
6.2.5	Combinarea criptării, autentificării și verificării prospețimii . . .	173
6.3	Stabilirea cheilor	173
6.3.1	Stabilirea cheilor în prezența unui adversar pasiv	176
6.3.1.1	Stabilirea cheilor prin criptografie asimetrică	176
6.3.1.2	Stabilirea cheii prin metoda Diffie-Hellman	177
6.3.1.3	Atacul man-in-the-middle	178
6.3.2	Stabilirea cheilor în prezența unui adversar activ	178
6.3.3	Stabilirea cheilor cu ajutorul unui terț de încredere	180
6.3.4	Certificarea cheilor publice	182
6.3.5	Transportul prin utilizatori umani	183
6.4	Numere aleatoare	185
6.4.1	Generatoare fizice	186
6.4.2	Generatoare de numere pseudoaleatoare	186
6.4.3	Generatoare utilizate în practică	188
6.5	Autentificarea utilizatorilor	188
6.5.1	Stocarea parolelor	188
6.5.2	Parole de unică folosință	189

Protocoale

Cuprins	195
7 Codificări de interes practic	203
7.1 Probleme privind reprezentarea numerelor întregi	203
7.1.1 Reprezentări pe biți	203
7.1.1.1 Bitul	204
7.1.1.2 Șiruri de biți	204
7.1.1.3 Reprezentarea pe biți a numerelor întregi	205
7.1.2 Reprezentări pe octeți	206
7.1.2.1 Octeți	206
7.1.2.2 Șiruri de octeți	208
7.1.2.3 Reprezentarea numerelor pe un număr întreg de octeți . . .	208
7.1.2.4 Reprezentarea numerelor pe un șir arbitrar de biți	210
7.1.3 Probleme privind reprezentarea lungimii șirurilor	212
7.1.4 Alte metode de reprezentare a numerelor întregi	214

7.2	Codificarea textelor	215
7.2.1	Codificarea ASCII	216
7.2.2	Codificările ISO-8859	217
7.2.3	Codificările Unicode	218
7.2.3.1	Codificarea UTF-8	220
7.2.3.2	Codificările UTF-16	220
7.2.3.3	Codificările UTF-32	221
7.3	Reprezentarea datei și orei	221
7.3.1	Măsurarea timpului	222
7.3.2	Obiectivele în alegerea reprezentării timpului în calculator	224
7.3.3	Formate utilizate în practică	225
7.3.3.1	Formatul utilizat de poșta electronică	225
7.3.3.2	ISO-8601 și RFC-3339	226
7.3.3.3	Timpul POSIX	227
7.3.3.4	TAI 64	227
7.4	Recodificări	228
7.4.1	Codificarea hexazecimală	228
7.4.2	Codificarea în baza 64	229
7.4.3	Codificări bazate pe secvențe de evitare	229
8	Programarea în rețea — introducere	231
8.1	Interfața de programare <i>socket BSD</i>	231
8.1.1	Comunicația prin conexiuni	232
8.1.1.1	Deschiderea conexiunii de către client	233
8.1.1.2	Deschiderea conexiunii de către server	233
8.1.1.3	Comunicația propriu-zisă	234
8.1.1.4	Închiderea conexiunii	234
8.1.2	Comunicația prin datagrame	235
8.1.3	Principalele apeluri sistem	237
8.1.3.1	Funcția <code>socket()</code>	237
8.1.3.2	Funcția <code>connect()</code>	237
8.1.3.3	Funcția <code>bind()</code>	238
8.1.3.4	Funcția <code>listen()</code>	239
8.1.3.5	Funcția <code>accept()</code>	239
8.1.3.6	Formatul adreselor	240
8.1.3.7	Interacțiunea dintre <code>connect()</code> , <code>listen()</code> și <code>accept()</code>	242
8.1.3.8	Funcțiile <code>getsockname()</code> și <code>getpeername()</code>	242
8.1.3.9	Funcțiile <code>send()</code> și <code>recv()</code>	243
8.1.3.10	Funcțiile <code>shutdown()</code> și <code>close()</code>	245
8.1.3.11	Funcțiile <code>sendto()</code> și <code>recvfrom()</code>	245
8.1.4	Exemple	246
8.1.4.1	Comunicare prin conexiune	246
8.1.4.2	Comunicare prin datagrame	249
8.2	Formatarea datelor	252

8.2.1	Formate binare	252
8.2.1.1	Tipuri întregi	252
8.2.1.2	Şiruri de caractere şi tablouri	254
8.2.1.3	Variabile compuse (struct -uri)	255
8.2.1.4	Pointeri	257
8.2.2	Formate text	257
8.2.3	Probleme de robusteţe şi securitate	257
8.2.4	Probleme privind costul apelurilor sistem	258
8.3	Probleme de concurenţă în comunicaţie	260
9	Reţele IEEE 802	263
9.1	Reţele IEEE 802.3 (Ethernet)	263
9.1.1	Legături punct la punct prin perechi de conductoare	266
9.1.2	Legături prin fibre optice	272
9.1.3	Legături prin cablu magistrală	274
9.1.4	Repetoarele şi comutatoarele	277
9.1.5	Dirijarea efectuată de comutatoare (switch-uri)	279
9.1.6	Facilităţi avansate ale switch-urilor	279
9.1.6.1	Switch-uri configurabile	279
9.1.6.2	Filtrare pe bază de adrese MAC	280
9.1.6.3	Trunking	280
9.1.6.4	Legături redundante	281
9.1.6.5	Reţele virtuale (VLAN)	281
9.1.7	Considerente privind proiectarea unei reţele	282
9.2	Reţele IEEE 802.11 (Wireless)	283
9.2.1	Arhitectura reţelei	283
9.2.2	Accesul la mediu	285
9.2.3	Generarea pachetelor <i>beacon</i>	286
9.2.4	Securitatea reţelelor 802.11	286
10	Internetul	291
10.1	Arhitectura reţelei	291
10.2	Protocolul IP	292
10.2.1	Structura pachetului IP	293
10.2.2	Bazele dirijării pachetelor IP	294
10.2.2.1	Subreţele şi interfeţe	294
10.2.2.2	Prefixul de reţea	295
10.2.2.3	Tabela de dirijare	296
10.2.3	Scrierea ca text a adreselor şi prefixelor	298
10.2.3.1	Scrierea adreselor IP	298
10.2.3.2	Scrierea prefixelor de reţea	300
10.2.4	Alocarea adreselor IP şi prefixelor de reţea	300
10.2.4.1	Alocarea pe utilizări	301
10.2.4.2	Alocarea adreselor şi dirijarea ierarhică	301

10.2.5	Erori la dirijare și protocolul ICMP	302
10.2.5.1	Pachete nelivrabile	303
10.2.5.2	Diagnosticarea funcționării rutelor	305
10.2.5.3	Ciclarea pachetelor IP	305
10.2.5.4	Congestia	306
10.2.5.5	Redirecționarea	306
10.2.6	Alte chestiuni privind dirijarea pachetelor	307
10.2.6.1	Dimensiunea maximă a pachetelor și fragmentarea	307
10.2.6.2	Calitatea serviciului	308
10.2.7	Configurarea și testarea unei rețele IP locale	309
10.2.7.1	Alegerea parametrilor	309
10.2.7.2	Configurarea parametrilor de rețea pe diverse sisteme de operare	312
10.2.7.3	Testarea și depanarea rețelelor	313
10.3	Nivelul transport	314
10.3.1	Conexiuni cu livrare garantată: protocolul TCP	314
10.3.1.1	Principiul conexiunii TCP	315
10.3.1.2	Comunicația bidirecțională	320
10.3.1.3	Deschiderea și închiderea conexiunii	320
10.3.1.4	Alegerea numărului inițial de secvență	323
10.3.1.5	Închiderea forțată a conexiunii	324
10.3.1.6	Identificarea aplicației destinație	325
10.3.1.7	Correspondența între funcțiile <code>socket()</code> și acțiunile modului TCP	326
10.3.1.8	Controlul fluxului	327
10.3.1.9	Stabilirea time-out-ului pentru retransmiterea pachetelor	327
10.3.1.10	Algoritmul lui Nagle și optimizarea numărului de pachete	328
10.3.1.11	Trimiterea datelor speciale (out of band)	328
10.3.2	Datagrame nesigure: UDP	329
10.4	Identificarea nodurilor după nume: sistemul DNS	330
10.4.1	Numele de domeniu	330
10.4.2	Structura logică a bazei de date DNS	332
10.4.3	Împărțirea în domenii de autoritate	333
10.4.4	Mecanismul de interogare a serverelor	334
10.4.5	Sincronizarea serverelor pentru un domeniu	335
10.4.6	Căutarea numelui după IP	336
10.5	Legăturile directe între nodurile IP	337
10.5.1	Rezolvarea adresei — ARP	337
10.6	Configurarea automată a stațiilor — DHCP	339
10.7	Situații speciale în dirijarea pachetelor	341
10.7.1	Filtre de pachete (firewall)	341
10.7.2	Rețele private	346
10.7.3	Translația adreselor (NAT)	347
10.7.3.1	Translația adresei sursă	347

10.7.3.2	Translația adresei destinație	350
10.7.4	Tunelarea	351
11	Aplicații în rețele	353
11.1	Poșta electronică	353
11.1.1	Formatul mesajelor	354
11.1.1.1	Antetul mesajelor	355
11.1.1.2	Extensii MIME	358
11.1.1.3	Atașarea fișierelor și mesaje din mai multe părți	359
11.1.1.4	Codificarea corpului mesajului și a atașamentelor	360
11.1.2	Transmiterea mesajelor	362
11.1.2.1	Protocolul SMTP	362
11.1.2.2	Determinarea următorului MTA	365
11.1.2.3	Configurarea unui MTA	366
11.1.3	Securitatea poștei electronice	368
11.2	Sesiuni interactive la distanță	371
11.2.1	Protocolul <i>ssh</i>	373
11.2.1.1	Conexiunea <i>ssh</i> protejată criptografic	373
11.2.1.2	Metode de autentificare în <i>ssh</i>	376
11.2.1.3	Multiplexarea conexiunii, tunelarea și aplicații	379
11.2.2	Sistemul X-Window	379
11.3	Transferul fișierelor în rețea	380
11.3.1	Protocolul <i>ftp</i>	381
11.3.2	Protocolul HTTP	382
11.3.2.1	Structura cererilor și a răspunsurilor	383
11.3.2.2	URL-urile	384
11.3.2.3	Alte facilități HTTP	385
11.3.2.4	Proxy HTTP	386
11.3.2.5	Conexiuni securizate: SSL/TLS	387
11.3.2.6	Utilizarea TLS pentru web	389
11.4	PGP/GPG	390
11.4.1	Structura cheilor GnuPG	390
11.4.1.1	Chei primare și subchei	391
11.4.1.2	Utilizatori și identități	392
11.4.1.3	Generarea și modificarea cheilor	392
11.4.1.4	Controlul perioadei de valabilitate a cheilor	393
11.4.1.5	Gestiunea cheilor secrete	395
11.4.2	Transmiterea și certificarea cheilor publice	395
11.4.2.1	Transmiterea cheilor publice	395
11.4.2.2	Verificarea autenticității cheilor	397
11.4.3	Transmiterea mesajelor criptate sau semnate	399
	Bibliografie	401
	Index	405

Prefață

În contextul prezent al dezvoltării rețelelor de calculatoare, este inutil să mai subliniem importanța acestui domeniu.

Lucrarea de față se adresează în principal programatorilor de aplicații în rețea și administratorilor de rețele complexe. Sunt presupuse, din partea cititorului, cunoștințe de bază de programare, precum și privind funcționarea sistemelor de operare.

Ca un avertisment pentru programatori, menționăm că, deși lucrarea tratează chestiuni de nivel mult mai coborât decât cel al platformelor și bibliotecilor utilizate în mod normal în aplicațiile în rețea, este totuși utilă în vederea unei bune înțelegeri a acestor platforme și biblioteci.

Tot ceea ce are legătură într-un fel sau altul cu calculatoare are două caracteristici: se dezvoltă foarte repede și est foarte complex. Rețelele de calculatoare nu fac excepție. Ca urmare, este extrem de ușor pentru oricine să se piardă în nenumăratele detalii în permanentă schimbare.

Considerăm că, în orice domeniu, o bună prezentare trebuie să pornească de la principiile de bază. Principiile de bază se sunt (relativ) simple și evoluează mult mai lent decât construcțiile tehnice elaborate pe baza lor. În consecință, prima parte a lucrării de față, *principii*, este dedicată studierii problemelor ce trebuie rezolvate de o rețea de calculatoare, precum și a principiilor construcției posibilelor soluții ale acestor probleme.

Partea a doua a lucrării, *protocele*, prezintă câteva dintre cele mai răspândite protocele și mecanisme utilizate în rețelele de calculatoare. Ea este construită pentru a oferi cititorului o privire de ansamblu asupra protocelelor studiate. Această privire de ansamblu poate fi suficientă pentru unii cititori, în caz contrar fiind probabil necesară citirea efectivă a standardelor.

Lucrarea de față este rodul experienței autorului în activități legate de administrarea rețelei de calculatoare a Departamentului de Informatică al Facultății de Matematică și Informatică din cadrul Universității Babeș-Bolyai Cluj-Napoca, în predarea unui curs de *Rețele de calculatoare* la această fac-

ultate, precum și din activitatea de cercetare desfășurată de-a lungul anilor, în special de nevoile practice din cadrul contractului de cercetare PNII 11003/2007 - *Sistem decizional bazat pe tehnici de tip multi-agent pentru generarea, optimizarea și managementul registrelor nationale de boli cronice netransmisibile - CRONIS*. Seturile mari de date ce se vehiculează în sistemul medical, precum și nevoia de confidențialitate și securitate a lor, cer o foarte bună cunoaștere și punere în practică a noțiunilor legate de codificarea informației, de metode și protocoale criptografice, de aplicații în rețele etc.

Capitolul 1

Introducere

Prin *rețea de calculatoare* înțelegem un sistem (constând din componente hard și soft) care interconectează niște calculatoare, permițând unor programe ce se execută pe aceste calculatoare să comunice între ele.

De notat că, în uzul comun, termenul de rețea de calculatoare mai are și sensul de *sistem de calcul*, construit din mai multe calculatoare interconectate într-o rețea, care se comportă ca un sistem unitar, de exemplu, prezintă aceleași conturi de utilizatori pe toate calculatoarele.

1.1. Serviciile oferite de rețea

Se spune că orice problemă bine formulată este pe jumătate rezolvată. Prin urmare, pentru început, vom stabili mai exact ce se dorește de la o rețea de calculatoare.

Într-o rețea de calculatoare avem mai multe calculatoare pe care se execută procese utilizator. Rolul rețelei este de-a oferi acestor procese posibilitatea de-a comunica între ele. Din punctul de vedere al programatorului acestor procese, rețeaua oferă niște funcții, din nucleul sistemului de operare sau din biblioteci standard, apelabile de către aceste procese (fig. 1.1). Ansamblul acestor funcții constituie *interfața de programare* (engl. *API — Application Programming Interface*) a rețelei.

Principalele funcții oferite de rețea, apelabile de către un proces utilizator, sunt o funcție care trimite date de la procesul curent spre partenerul sau partenerii de comunicație și o funcție care recepționează datele trimise spre procesul curent. În aceste funcții este necesară desemnarea destinatarului spre care procesul emițător dorește transmiterea datelor, respectiv a emițătorului dinspre care procesul receptor solicită să primească date. În acest scop, fiecare

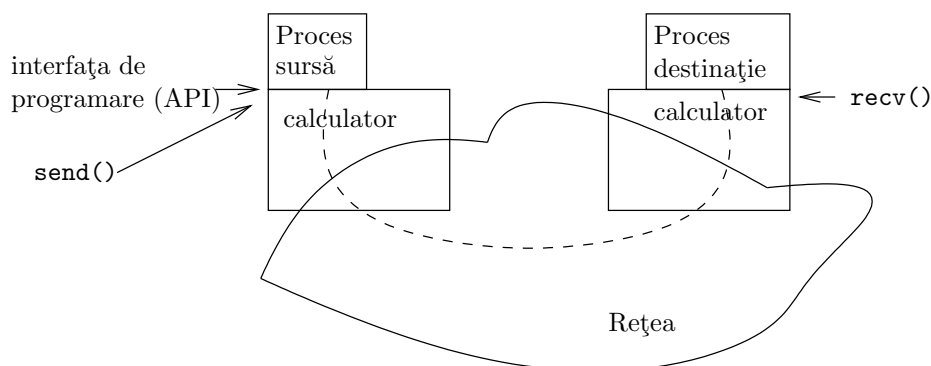


Figura 1.1: Rețeaua de calculatoare, din punctul de vedere al proceselor aplicație. Funcționalitatea rețelei este oferită prin funcții apelabile din procesele utilizator. Rețeaua oferă o aplicațiilor o cale de transmisie a datelor (linia punctată). Construcția efectivă a rețelei nu este vizibilă aplicațiilor.

entitate ce poate comunica în rețea trebuie să aibă asociată o *adresă* (un șir de biți, construit după anumite reguli, identificând unic o anumită entitate).

Pe lângă aceste funcții de bază, rețeaua mai oferă funcții pentru configurarea diferiților parametri. O parte dintre acești parametri fixează rolul și locul diverselor componente în cadrul rețelei (de exemplu, fiecare calculator trebuie să-și cunoască propria adresă). Alți parametri sunt legați de calitatea serviciilor oferite de rețea (debit de transfer de date, timp de propagare, etc).

Datele transmise de procesele utilizator sunt de obicei șiruri arbitrare de octeți. Rolul rețelei este de-a transmite întocmai șirul de octeți trimis de procesul sursă către procesul destinație. Semnificația, pentru procesul destinație, a unui șir de octeți transmis face obiectul unei înțelegeri (*protocol*) între procesele utilizator. La proiectarea rețelei nu ne interesează ce fac procesele utilizator cu datele transferate; la proiectarea programelor utilizator nu ne interesează cum lucrează rețeaua pentru a transmite datele.

În continuare vom trece în revistă principalele caracteristici ale serviciului oferit de rețea proceselor de aplicație.

O comunicație poate fi, după numărul destinatarilor:

- *punct la punct*, dacă există un singur destinatar. În mod obișnuit, destinatarul este selecționat explicit de către procesul emițător; o astfel de comunicație este numită *unicast*. Uneori însă, de exemplu în cazul în care un serviciu este oferit de mai multe *servere*, echivalente din punctul de vedere al clientului, este favorabil ca rețeaua să aleagă destinatarul comunicației, în funcție de distanța față de emițător, dintr-o mulțime

specificată de destinatari posibili. Un astfel de comunicație se numește *anycast*.

- *difuziune*, dacă există mai mulți destinatari. Distingem *difuziune completă* (engl. *broadcast*), în care destinatari sunt toate calculatoarele dintr-o rețea, și *difuziune selectivă* (engl. *multicast*), în care destinatarii sunt o submulțime aleasă a calculatoarelor din rețea.

Serviciul de comunicație oferit de rețea poate fi de tip *conexiune* sau de tip transport de *datagrame*:

- În cazul *conexiunilor*, în cadrul comunicației între două procese se disting trei faze:
 - deschiderea conexiunii, în cadrul căreia sunt făcute niște pregătiri, inclusiv alocarea unor resurse pentru comunicație;
 - comunicația propriu-zisă, în care unul sau ambele procese transmite un șir de pachete sau de biți celuilalt proces;
 - închiderea conexiunii, în cadrul căreia se eliberează resursele alocate la deschidere.
- În cazul transportului de datagrame, procesul emițător pregătește un ansamblu, numit *datagramă* (prin analogie cu *telegramă*), cuprinzând un șir de biți destinat procesului receptor și anumite informații necesare livrării (adresa destinatarului). Apoi transmite datagrama rețelei de calculatoare, care o transmite procesului receptor. Mai multe datagrame trimise de același proces sursă către același proces destinație sunt transmise independent una de alta, ceea ce duce, în general, la posibilitatea inversării ordinii de recepție față de ordinea de emisie a datagramelor.

Principalii parametri de calitate ai serviciului oferit de rețea sunt:

Capacitatea de transport oferită de rețea, sau *debitul maxim acceptat*, este raportul dintre numărul de biți transportați în cadrul unei comunicații și timpul în care aceștia sunt transmiși. Echivalent, capacitatea este inversul duratei medii între trecerea, printr-un punct dat al rețelei, a doi biți consecutivi ai unei comunicații.

Timpul de transfer a unui bloc de date este timpul scurs de la trecerea, printr-un punct dat, a primului bit al blocului până la trecerea, prin același punct, a ultimului bit. Timpul de transfer este egal cu raportul dintre dimensiunea blocului și debitul cu care se face transferul.

Capacitatea oferită de rețea unei legături poate să varieze datorită variației debitului altor comunicații care partajează aceleași echipamente.

Există aplicații, de exemplu legate de transfer de fișiere, pentru care este important ca rețeaua să ofere o capacitate medie cât mai mare. Pentru alte aplicații, cum ar fi telefonie, transmisia video (de exemplu pentru teleconferințe) sau alte aplicații în timp real, este important să nu scadă niciodată capacitatea legăturii sub o anumită valoare minimă, însă o capacitate mai mare nu este utilă.

Timpul de propagare între două entități este timpul scurs între momentul în care entitatea sursă emite un bit și momentul în care acel bit ajunge la destinație. Timpul de propagare rezultă din însumarea timpului de propagare a semnalului de-a lungul mediului de comunicație cu diverșii timpi de așteptare a datelor în diverse zone tampon. De remarcat că timpul de propagare a semnalului este egal cu distanța de la emițător la receptor împărțită la viteza de propagare a semnalului, iar viteza de propagare nu poate depăși viteza luminii în vid; din acest motiv, de exemplu, timpul de propagare prin legături prin satelit nu poate fi mai scurt de câteva zecimi de secundă.

Timpul scurs de la începutul transmisiei unui bloc de date de către emițător până la finalul recepției blocului de către receptor este egal cu suma dintre timpul de transfer și timpul de propagare.

Uneori, în loc de timpul de propagare se utilizează o altă mărime, *timpul dus-întors*, care este timpul scurs de la transmiterea unui mesaj de către o partenerul de comunicație până la primirea răspunsului din partea acestuia. Timpul dus-întors este suma dintre timpurile de propagare pentru cele două sensuri și timpul de procesare pentru crearea răspunsului.

Evident, timpul de propagare e bine să fie cât mai scurt, însă diferite aplicații au cerințe diferite:

- La unele aplicații timpul de propagare nu este prea important. De exemplu, la transferul unui fișier mare, la care oricum timpul de transfer este mare, timpul de propagare influențează foarte puțin timpul total necesar transmiterii fișierului.
- La difuzarea de materiale audio sau video, un timp de propagare mare nu este deranjant, însă este important ca el să fie constant în timp. Aceasta pentru că nu este deranjant dacă o transmisie de televiziune este cu câteva secunde în întârziere față de evenimentele transmise, însă este important să nu fie momente în care imaginea „îngheață“ datorită creșterii timpului de propagare și momente în care imaginea „sare înainte“ datorită scurtării timpului de propagare.

- Timpul de propagare (sau, echivalent, timpul dus-întors) este important să fie scurt în special pentru aplicații în care entitățile ce comunică transmit mesaje scurte și trebuie să aștepte răspunsul la mesajul precedent pentru a putea genera mesajul următor. Exemple de astfel de aplicații sunt: telefonie, videoconferințe, sesiuni interactive la distanță.

Posibilitatea existenței erorilor de transmisie: Erorile de transmisie apar ca urmare a diverselor perturbații ce afectează transmiterea semnalelor. Există metode de-a micșora oricât de mult probabilitatea ca un mesaj să fie afectat de erori, însă niciodată această probabilitate nu poate fi făcută zero (probabilitatea unei erori poate fi făcută însă mai mică decât, de exemplu, probabilitatea unui cataclism devastator care să distrugă toată rețeaua). Metodele de reducere a probabilității erorilor de transmisie sunt studiate în § 2.4 și § 4.1.

Transmisia sigură înseamnă ca fiecare mesaj al entității sursă să ajungă exact într-un singur exemplar la destinație (să nu se piardă și să nu fie duplicat) și mai multe mesaje transmise de către o aceeași sursă spre o aceeași destinație să ajungă la destinație în ordinea în care au fost transmise de sursă. Mesajele se pot pierde datorită erorilor de transmisie, a supraaglomerării sau a defectării unor echipamente din rețea sau chiar din cauză că emițătorul transmite cu debit mai mare decât este capabil receptorul să preia informația transmisă. Duplicarea sau inversarea mesajelor pot fi cauzate de modificări ale configurației sau încărcării rețelei în timpul trecerii pachetelor prin rețea. Realizarea transmisiei sigure este studiată în § 4.3 și § 4.4.

Transmisia sigură este evident utilă, însă vine cu un anumit cost. Cel mai adesea, costul este creșterea și fluctuația timpului de propagare, deoarece mesajele pierdute trebuie retransmise. La o transmisie audio-video, este adesea preferabilă păstrarea unui timp de propagare redus, cu prețul pierderii, din când în când, a unor fracțiuni de secundă de material audio-video.

Securitatea comunicației înseamnă că un adversar care controlează o parte din rețea să nu poată obține informația transmisă, să nu poată modifica datele transmise fără ca acest lucru să fie detectat de către receptor și să nu poată impersona vreuna dintre entitățile ce comunică. Securitatea comunicației se obține prin metode criptografice, studiate în capitolul 6.

1.2. Principalele elemente ale unei rețele de calculatoare

Pentru ca două dispozitive aflate la distanță unul de celălalt să poată comunica, este nevoie ca cele două dispozitive să fie legate printr-un *mediu de comunicație* care permite propagarea variației unei mărimi fizice. Mediul fizic, împreună cu dispozitivele de adaptare între reprezentarea locală a informației și reprezentarea pe mediul de transmisie constituie *nivelul fizic* al rețelei. Nivelul fizic este deci un modul care permite transmisia unui șir de biți între două dispozitive legate direct unul de celălalt. Constructiv, nivelul fizic este constituit din: cablul electric, fibra optică sau, după caz, antenele de emisie-recepție, eventuale amplificatoare sau repetitoare, plăcile de rețea din calculatoare și *driver*-ele plăcilor de rețea. Construcția nivelului fizic este studiată în capitolul 3.

De obicei, serviciul oferit de nivelul fizic suferă de anumite neajunsuri, cum ar fi probabilitatea mare a erorilor și transmisia nesigură. Pentru contracararea acestora, de-o parte și de alta a nivelului fizic se plasează câte un modul de adaptare; aceste două module constituie *nivelul legăturii de date*. Nivelul legăturii de date este construit parțial prin hard (parte a plăcii de rețea) și parțial prin soft (parte a *driver*-ului plăcii de rețea). Construcția nivelului legăturii de date este studiată în capitolul 4.

Nivelul fizic împreună cu nivelul legăturii de date oferă o legătură bună între două calculatoare conectate direct printr-un mediu fizic. Ar fi neeconomic să cerem să existe o legătură directă între oricare două calculatoare; este preferabil să putem transmite date prin intermediul unui lanț de calculatoare (sau alte dispozitive) legate fizic fiecare cu următorul din lanț. Realizarea unei astfel de legături cade în sarcina *nivelului rețea*, constituit din câte un modul în fiecare calculator al rețelei. Modulul de rețea este construit prin soft, în nucleul sistemului de operare al fiecărui calculator din rețea. Construcția și funcționarea nivelului rețea este studiată în capitolul 5.

De obicei, serviciul oferit direct de către nivelul rețea nu poate fi utilizat direct de către programele utilizator. De aceea, între modulul de rețea și programul utilizator se mai interpune un modul, constituind (împreună cu modulul omolog de pe calculatorul partener de comunicații) *nivelul transport*. Nivelul transport este constituit din părți ale nucleului sistemului de operare și, uneori, biblioteci legate în programele utilizator.

Relațiile dintre aceste componente sunt reprezentate în figura 1.2.

Fiecare dintre nivele oferă nivelului superior o interfață care cuprinde în principal funcții de trimitere și de recepție a datelor. Aceste funcții sunt

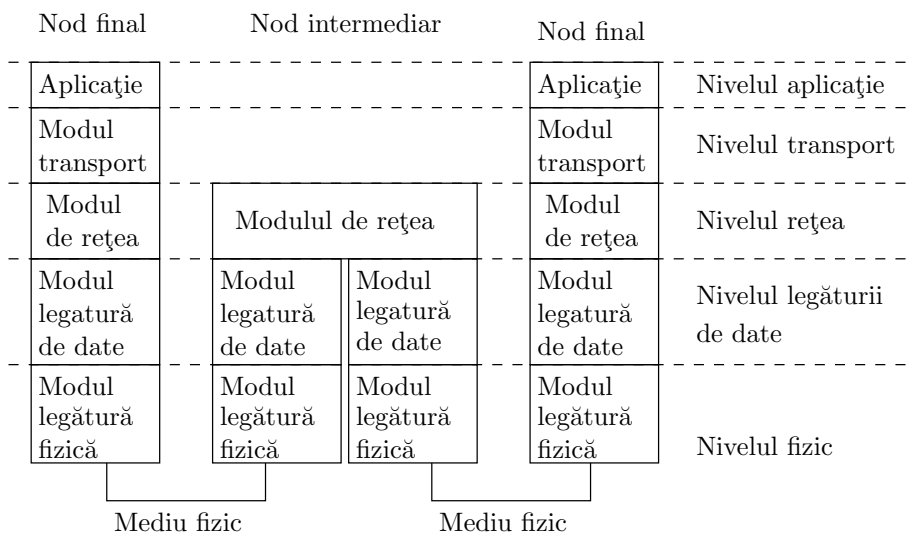


Figura 1.2: Componentele unei părți dintr-o rețea de calculatoare. Sunt figurate doar componentele implicate în comunicația dintre două aplicații. Cele două aplicații se execută pe două calculatoare între care nu există o legătură directă, dar există o legătură printr-un nod intermediar.

similare celor oferite de rețea aplicațiilor (așa cum am văzut în § 1.1), dar serviciile oferite sunt mai primitive. Astfel, nivelul fizic oferă nivelului legăturii de date servicii de transfer de date, dar numai între calculatoare conectate direct și cu riscul ca datele să fie alterate în timpul transferului sau să se piardă complet. Nivelul legăturii de date oferă nivelului rețea servicii de transfer de date mai sigure, dar în continuare cu restricția că transferul este posibil doar între calculatoare conectate direct. Nivelul rețea oferă nivelului transport servicii de transfer de date între orice două calculatoare din rețea, dar încă neadecvate utilizării directe de către aplicații (lipsa transmisiei sigure, comunicație posibilă doar pentru un singur proces aplicație la un moment dat, etc.).

Construcția fiecăruia dintre nivele este independentă de construcția celorlalte (contează doar interfața dintre ele și parametrii de calitate a serviciului oferit de un nivel celui imediat superior). De exemplu, în proiectarea nivelului rețea, nu ne interesează nici ce aplicații vor utiliza rețeaua (aceleși nivel rețea din Internet este utilizat de aplicații de poșta electronică, web, telefonie prin Internet și videoconferințe), nici cum este construit nivelul fizic (perechi de conductoare, fibre optice sau legături radio prin satelit).

Modulele, de pe același nivel, din noduri diferite își transmit unul altuia (utilizând în acest scop serviciile oferite de nivelul inferior) două tipuri

de date: *datele utile* a căror transfer este cerut de nivelul superior și *date de control* necesare coordonării activităților modulelor din cadrul nivelului. Regulile de reprezentare a acestor date, de organizare a acestora în mesaje, precum și regulile după care se trimit mesajele între modulele aceleiași nivel alcătuiesc *protocolul de comunicație* al nivelului respectiv.

Funcționarea corectă a unei rețele necesită respectarea, de către toate modulele implicate, a protocoalelor de comunicație stabilite.

1.3. Premise generale în elaborarea și implementarea protocoalelor în rețele

Pe lângă rațiunile pur funcționale, studiate pe larg în capitolele următoare, în elaborarea și implementarea protocoalelor intervin rațiuni practice, pe care le vom înșira pe scurt în continuare:

- Deoarece o rețea este formată din multe componente, frecvența cu care se întâmplă ca cel puțin o componentă a unei rețele să nu funcționeze corect este mare. Este necesar ca o defecțiune să afecteze cât mai puțin din rețea, iar componentele a căror defectare duce la căderea întregii rețele trebuie să fie cât mai puține, eventual nici una.
- Găsirea unei pene într-un sistem complex este, în general, dificilă. Rețeaua trebuie să ofere mecanisme prin care orice defecțiune să fie ușor de localizat.
- Implementări diferite ale unui protocol se pot abate în moduri diferite de la specificația protocolului. Este bine ca mici abateri ale partenerului de comunicație să fie tolerate. Rezultă de aici principiul că o implementare trebuie să fie *strictă cu ceea ce transmite și tolerantă cu ceea ce recepționează*.
- Rețeaua trebuie să funcționeze astăzi, sau, *un plan bun azi este mai bun decât un plan perfect mâine* (maximă atribuită generalului american George Patton, circa 1944). Momentul standardizării unui protocol este extrem de delicat: dacă este standardizat înainte ca problema de rezolvat să fie bine înțeleasă și soluțiile posibile bine analizate, rezultă un protocol prost; dacă standardizarea apare prea târziu, după ce s-a răspândit deja un protocol acceptabil, există riscul creerii unui protocol perfect, dar pe care nu-l folosește nimeni deoarece înlocuirea sistemelor existente ar fi mai scumpă decât avantajul adus de protocolul mai bun.
- Protocoalele totuși evoluează, iar oprirea întregii rețele în vederea schimbării echipamentelor afectate de schimbarea protocolului nu este rezonabilă.

Ca urmare, la o schimbare de protocol trebuie avut în vedere existența unei perioade de tranziție în timpul căreia echipamentele noi trebuie să poată comunica cu cele vechi. Tranziția este mult ușurată dacă protocolul vechi prevede anumite facilități. O posibilitate este ca în protocol să se prevadă o fază de negociere în care fiecare entitate anunță ce versiuni de protocol și ce extensii de protocol cunoaște, iar apoi comunicația decurge conform versiunii celei mai recente și cu cele mai multe extensii suportate de ambii parteneri. Altă posibilitate este stabilirea, de la prima versiune a protocolului, a acțiunilor unui dispozitiv, ce implementează o versiune veche a protocolului, la primirea unui mesaj neprevăzut în acea versiune.

- Cerințe diferite ale diferitelor aplicații duc la tendința de-a elabora protocoale complexe, care să satisfacă pe toată lumea. Protocoale complexe duc la implementări scumpe și cu riscuri mari de-a avea erori. Este preferabil un protocol care să ofere câteva operații simple care să poată fi combinate după dorința aplicației ce-l utilizează. Dacă o astfel de abordare nu este fezabilă, ducând la un protocol prea complex, se recurge la protocoale ce au posibilitatea de-a fi implementate doar parțial; metodele utilizabile în acest scop sunt similare cu cele descrise mai sus pentru facilitarea evoluției protocoalelor.

Capitolul 2

Noțiuni de teoria informației

Teoria informației se ocupă cu studiul metodelor de codificare a informației în vederea transmiterii sau stocării acesteia. În cadrul teoriei informației se studiază și cum se poate măsura cantitatea de informație transmisă într-un mesaj și cum se poate măsura eficiența unei anumite codificări.

Prin *informație* înțelegem cunoștințele unei entități.

În cele ce urmează, ne va interesa problema transmiterii unei informații de la o *sursă* la o *destinație*. Informația de transmis nu este cunoscută inițial nici de destinație, nici de sistemul de transmitere. Ca urmare, *a priori* informația de transmis poate fi văzută ca o variabilă aleatoare.

Comunicația dintre sursă și destinație se desfășoară prin intermediul unui *canal de comunicație*. Canalul de comunicație este capabil să transmită fie o mărime variabilă în timp, numită *semnal* (în esență, o funcție reală continuă), caz în care canalul este numit *continuu*, fie un șir de simboluri dintr-o mulțime finită, caz în care canalul este numit *discret*.

Deoarece canalul nu poate transmite direct informația sursei, între sursă și canal avem nevoie de un dispozitiv, numit *emițător*, care transformă informația utilă, produsă de sursă, într-un semnal sau, după caz, într-un șir de simboluri. Similar, între canal și destinație se plasează un dispozitiv, numit *receptor*, al cărui rol este de-a efectua operația inversă, și anume de-a extrage din semnal sau din șirul de simboluri informația utilă pentru destinație (fig. 2.1).

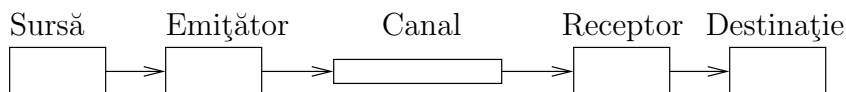


Figura 2.1: Transmisia informației de la sursă la destinație

Semnalul sau, după caz, șirul de simboluri ce tranzitează canalul se numește *reprezentarea informației*. Regulile de corespondență dintre informația utilă și reprezentarea sa poartă denumirea de *schemă de reprezentare a informației*, *schemă de codificare a informației* sau *cod*.

Ca exemplu, o limbă scrisă este o schemă de reprezentare a informației, pentru un canal discret a cărui mulțime de simboluri conține literele alfabetului limbii respective, precum și spațiul și semnele de punctuație. Un text scris într-o limbă este o *reprezentare a informației*, iar conceptele din textul respectiv sunt efectiv *informația* conținută în text.

Ca un al doilea exemplu, limba vorbită este o altă schemă de reprezentare a informației, canalul pentru care este construită fiind de tip continuu.

Schema de codificare a informației se presupune că este stabilită în prealabil și este cunoscută atât emițătorului cât și receptorului. De asemenea, în construcția schemei de reprezentare a informației se ține cont de caracteristicile canalului și de caracteristici generale ale informațiilor ce trebuie să se poată transmite, însă la elaborarea ei nu se cunosc informațiile ce trebuie efectiv transmise. De exemplu, la elaborarea unei scheme de codificare a literelor dintr-un text utilizând un canal ce poate transmite doar simbolurile 0 și 1 se poate ține cont de frecvența obișnuită a literelor într-un text, dar nu și de textul efectiv de transmis.

Restul capitolului tratează scheme de reprezentare a informației pentru canale discrete. Vom studia în continuare:

- proprietăți generale ale codurilor,
- problema minimizării numărului de simboluri necesare a fi transmise prin canal, precum și măsurarea cantității de informație,
- problema codificării în cazul în care canalul alterează șirul de simboluri pe care îl transmite (canal cu perturbații).

2.1. Problema codificării informației pentru un canal discret

În cazul unui canal discret, canalul poate transmite un șir de simboluri dintr-o mulțime S , numită *mulțimea simbolurilor de cod* sau *alfabetul canalului*. Elementele lui S se numesc *simboluri de cod* sau, scurt, *simboluri*. Mulțimea S este finită și are cel puțin două elemente. De regulă $S = \{0, 1\}$.

Pentru șirurile de simboluri de cod vom utiliza următoarele notații:

- S^* reprezintă mulțimea șirurilor finite de elemente din S .
- $u \cdot v$ reprezintă concatenarea șirurilor u și v .

- $|u|$ reprezintă lungimea șirului u ; avem $|u \cdot v| = |u| + |v|$, $\forall u, v \in S^*$.
- ε este șirul vid; avem $|\varepsilon| = 0$ și $u \cdot \varepsilon = \varepsilon \cdot u = u$, $\forall u \in S^*$.

Informația transmisă de către sursă constă dintr-un șir de *mesaje*. Fiecare mesaj este un element dintr-o mulțime M de mesaje posibile. Mesajele provin din universul utilizatorului sistemului; ele pot fi propoziții, litere, numere, etc.

Mulțimea de mesaje M este nevidă și cel mult numărabilă. De cele mai multe ori M este finită.

Definiția 2.1 Numim funcție de codificare sau cod orice funcție injectivă $c : M \rightarrow S^*$, unde M este mulțimea de mesaje, cel mult numărabilă, iar S este mulțimea simbolurilor de cod, finită și având cel puțin două elemente.

Fiecare mesaj $m \in M$ va fi codificat prin șirul $c(m) \in S^*$.

Definiția 2.2 Numim cuvânt de cod orice șir de simboluri de cod $w \in S^*$ cu proprietatea că există un mesaj $m \in M$ astfel încât $w = c(m)$.

Numim mulțimea cuvintelor de cod mulțimea $W = c(M)$.

Un șir de mesaje $(m_1, \dots, m_k) \in M^*$ (unde M^* desemnează mulțimea șirurilor finite de mesaje din M) va fi codificat prin șirul format prin concatenarea codificărilor mesajelor:

$$c(m_1) \cdot c(m_2) \cdot \dots \cdot c(m_k).$$

De remarcat că în urma concatenării se pierd delimitările dintre codificările mesajelor individuale. Ca urmare, pentru ca receptorul să poată decodifica fără ambiguități orice transmisie a emițătorului este necesară o proprietate suplimentară a codului, aceea de-a fi unic decodabil:

Definiția 2.3 Un cod $c : M \rightarrow S^*$ se numește:

- cod unic decodabil, dacă funcția $\hat{c} : M^* \rightarrow S^*$ dată prin

$$\hat{c}(m_1, m_2, \dots, m_k) = c(m_1) \cdot c(m_2) \cdot c(m_k) \quad (2.1)$$

este injectivă.

- cod cu proprietatea de prefix sau cod prefix, dacă nu există $m_1, m_2 \in M$, cu $m_1 \neq m_2$, astfel încât $c(m_1)$ să fie prefix pentru $c(m_2)$ și în plus $c(m) \neq \varepsilon$, $\forall m \in M$.

- cod de lungime fixă, dacă există o constantă $l \in \mathbb{N} \setminus \{0\}$ astfel încât $|c(m)| = l, \forall m \in M$; valoarea l se numește lungimea codului;

Propoziția 2.4 *Au loc următoarele proprietăți:*

1. Orice cod de lungime fixă este cod prefix.
2. Orice cod prefix este unic decodabil.

Demonstrația este imediată.

EXEMPLUL 2.1: Considerăm mulțimea mesajelor $M = \{a, b, c, d\}$ și mulțimea simbolurilor de cod $S = \{0, 1\}$. Următorul cod are proprietatea de prefix.

$$\begin{aligned} a &\mapsto 0 \\ b &\mapsto 101 \\ c &\mapsto 11 \\ d &\mapsto 100 \end{aligned}$$

EXEMPLUL 2.2: Următorul cod, obținut prin oglindirea cuvintelor codului din exemplul anterior, este unic decodabil dar nu are proprietatea de prefix:

$$\begin{aligned} a &\mapsto 0 \\ b &\mapsto 101 \\ c &\mapsto 11 \\ d &\mapsto 001 \end{aligned}$$

Codul nu este prefix întrucât cuvântul de cod 0 care este codificarea mesajului a este prefix al cuvântului de cod 001 care este codificarea mesajului d .

De notat că un cod obținut prin oglindirea cuvintelor unui cod prefix se numește *cod suffix* și întotdeauna este unic decodabil.

EXEMPLUL 2.3: Codul de mai jos nu este unic decodabil:

$$\begin{aligned} a &\mapsto 0 \\ b &\mapsto 1 \\ c &\mapsto 01 \end{aligned}$$

Codul nu este unic decodabil întrucât șirul de simboluri de cod 01 poate fi codificarea mesajului c sau a șirului de mesaje ab .

2.2. Coduri cu proprietatea de prefix

Deși simple, codurile de lungime fixă nu sunt adecvate în următoarele două cazuri:

- pentru obținerea unui cod eficient, adică având cuvinte cât mai scurte, dacă probabilitățile diverselor mesaje din M sunt diferite (M este mulțimea mesajelor sursei);
- dacă M nu este finită (de exemplu, M este mulțimea numerelor naturale).

În aceste situații, trebuie să ne extindem la clase mai largi decât cea a codurilor de lungime fixă. Așa cum vom vedea în continuarea paragrafului de față, clasa codurilor prefix este suficientă în situațiile enumerate mai sus și, în același timp, permite decodificarea destul de simplă a transmisiei.

2.2.1. Reprezentarea arborescentă a codurilor prefix

Unui cod prefix $c : M \rightarrow S^*$ i se poate atașa un arbore în care:

- pentru fiecare nod intern, muchiile descendente sunt cel mult în număr de $|S|$ și sunt etichetate cu simboluri distincte din S ;
- fiecare frunză este etichetată cu câte un mesaj distinct din M ;
- cuvântul de cod al unui mesaj este format din simbolurile de cod ale muchiilor de pe lanțul ce unește rădăcina cu frunza atașată mesajului.

Construcția arborelui se face conform algoritmului 2.1 (*Generează_arbore*).

EXEMPLUL 2.4: Pentru codul din exemplul 2.1 arborele este reprezentat în figura 2.2.

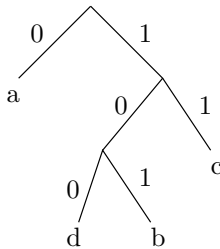


Figura 2.2: Arborele atașat unui cod prefix

EXEMPLUL 2.5: Fie codul prefix pentru mulțimea mesajelor

$$M = \{a, b, c, d, e, f, g, h\}$$

Algoritmul *Generează_arbore*

intrarea: M mulţime finită nevidă

$c : M \rightarrow S^*$ cod prefix

ieşirea: T arborele asociat codului c

algoritmul:

crează T format doar din rădăcină

$r :=$ rădăcina lui T

pentru $m \in M$ execută

$(s_1, \dots, s_l) := c(m)$

$x := r$

pentru $i := 1, l$ execută

dacă nu există muchie descendentă de la x etichetată cu s_i atunci

dacă x are asociat un mesaj atunci

eroare: c nu este cod este prefix

sfârşit dacă

crează y descendent al lui x şi etichetează (x, y) cu s_i

sfârşit dacă

$x :=$ descendentul lui x pe muchia etichetată s_i

sfârşit pentru

dacă x nu e frunză atunci

eroare: c nu este cod este prefix

sfârşit dacă

asociază m nodului x

sfârşit pentru

sfârşit algoritm

Algoritmul 2.1: Generarea arborelui asociat unui cod prefix

și mulțimea simbolurilor de cod $S = \{0, 1, 2\}$:

a	↔	0
b	↔	10
c	↔	11
d	↔	12
e	↔	200
f	↔	201
g	↔	21
h	↔	22

Arborele atașat este reprezentat în figura 2.3.

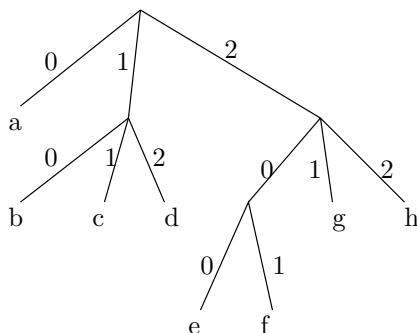


Figura 2.3: Arborele atașat codului prefix din exemplul 2.5.

2.2.2. Decodificarea în cazul codurilor prefix

Dacă avem un șir de mesaje codificat printr-un cod prefix, decodificarea se poate face prin algoritmul 2.2. Acesta rulează în timp proporțional cu numărul de simboluri de cod din reprezentarea datelor de decodificat.

De remarcat că fiecare mesaj este decodificat de îndată ce ultimul simbol din reprezentarea sa a fost citit și prelucrat. Acest lucru este posibil numai pentru codurile prefix; din acest motiv, codurile prefix se mai numesc și *coduri instantanee*.

EXEMPLUL 2.6: Fie codul prefix din exemplul 2.5 (vezi fig. 2.3) și fie șirul de decodificat:

$$s = 0112000$$

Algoritmul *Decodează*

intrarea: T arborele unui cod prefix $c : M \rightarrow S^*$

$s = (s_1, s_2, \dots, s_l) \in S^*$ un şir finit de simboluri de cod

ieşirea: $m = (m_1, m_2, \dots, m_k) \in M^*$ şirul mesajelor a căror codificare este s_1, \dots, s_l

algoritmul:

$m := \varepsilon$

$x :=$ rădăcina lui T

pentru $i := 1, l$ execută

 dacă nu există muchie descendentă de la x etichetată cu s_i atunci

 eroare: s nu este concatenare de cuvinte de cod

 sfârşit dacă

$x :=$ descendentul ui x pe muchia etichetată cu s_i

 dacă x este frunză atunci

 adaugă la m mesajul asociat lui x

$x :=$ rădăcina lui T

 sfârşit dacă

sfârşit pentru

dacă x nu este rădăcina lui T atunci

 eroare: s nu este concatenare de cuvinte de cod

sfârşit dacă

sfârşit algoritm

Algoritmul 2.2: Decodificarea unei reprezentări printr-un cod prefix

Decodificarea se face astfel: La început x este rădăcina arborelui. Luăm din șirul s primul element; acesta are valoarea 0. Coborâm în arbore de-a lungul muchiei etichetate cu 0 și ajungem la frunza etichetată „a”. Deoarece am ajuns la o frunză, punem mesajul din eticheta frunzai — adică „a” — în șirul de mesaje decodificat și revenim la rădăcină. Urmează simbolul de cod 1; coborâm de-a lungul muchiei 1 și ajungem în nodul părinte ale nodurilor „b”, „c” și „d”. Urmează simbolul 1; coborâm de-a lungul muchiei 1 și ajungem la frunza „c”; adăugăm „c” la șirul de mesaje și revenim la rădăcină. Continuând în același fel, vom obține în continuare mesajele „e” și „a”. Șirul de mesaje transmis este deci „acea”.

2.2.3. Lungimile cuvintelor unui cod prefix

În cele ce urmează, vom examina o condiție necesară și suficientă pentru existența unui cod prefix cu lungimi date ale cuvintelor, iar apoi vom arăta că această condiție este de asemenea necesară pentru existența unui cod unic decodabil.

Teorema 2.5 *Fiind dată o mulțime de mesaje M cel mult numărabilă și o mulțime de simboluri S finită având cel puțin 2 elemente distincte, pentru orice cod $c : M \rightarrow S^*$ cu proprietatea de prefix, lungimile cuvintelor de cod $l_i = |c(i)|$, $i \in M$, satisfac următoarea inegalitate (inegalitatea lui Kraft):*

$$\sum_{i \in M} |S|^{-l_i} \leq 1 \quad (2.2)$$

și, reciproc, dacă numerele naturale $(l_i)_{i \in M}$ satisfac inegalitatea (2.2) atunci există un cod prefix $c : M \rightarrow S^*$ având lungimile cuvintelor $|c(i)| = l_i$, $\forall i \in M$.

Demonstrație. Vom nota în continuare $d = |S|$ și $K = \sum_{m \in M} d^{-l_m}$.

Vom demonstra întâi prima implicație, pentru cazul în care mulțimea mesajelor M este finită. Demonstrația va fi construită prin inducție după maximul k al lungimilor cuvintelor de cod ($k = \max_{m \in M} l_m$).

Pentru $k = 1$, înseamnă că toate cuvintele de cod sunt de lungime 1 și în consecință sunt în număr de cel mult d . Ca urmare

$$K = \sum_{m \in M} d^{-1} = |M| \cdot d^{-1} \leq d \cdot d^{-1} = 1.$$

Presupunând inegalitatea lui Kraft adevărată pentru coduri de lungime maximă $k = k_0$, pentru un $k_0 \in \mathbf{N}^*$ arbitrar, să demonstrăm că are loc și pentru coduri de lungime maximă $k = k_0 + 1$. Pentru aceasta, să construim mulțimile de mesaje

$$M_x = \{m \in M : \text{primul simbol din } c(m) \text{ este } x\}, \quad x \in S.$$

Se observă imediat că $(M_x)_{x \in S}$ sunt disjuncte două câte două și că reuniunea lor este M . Ca urmare

$$K = \sum_{x \in S} \sum_{m \in M_x} d^{-l_m}.$$

Pentru fiecare $x \in M$, restricția lui c la M_x , $c|_{M_x}$, este de asemenea un cod prefix. Distingem în continuare trei cazuri:

- Dacă M_x are cel puțin 2 elemente, rezultă că toate cuvintele de cod ale elementelor din M_x au lungime mai mare sau egală cu 2, deoarece în caz contrar singurul cuvânt de cod de lungime 1, anume x , ar fi prefix pentru toate celelalte. Eliminând din toate cuvintele de cod primul simbol obținem un nou cod prefix pentru M_x . Acest cod prefix are toate cuvintele de cod lungime cel mult k_0 și ca urmare, conform ipotezei de inducție, satisface inegalitatea lui Kraft, adică

$$\sum_{m \in M_x} d^{-(l_m-1)} \leq 1,$$

de unde

$$\sum_{m \in M_x} d^{-l_m} \leq \frac{1}{d}.$$

- Dacă M_x are un singur element, cuvântul de cod asociat acestui element are lungime cel puțin 1 și ca urmare din nou

$$\sum_{m \in M_x} d^{-l_m} \leq \frac{1}{d}.$$

- Dacă $M_x = \emptyset$, avem $\sum_{m \in M_x} d^{-l_m} = 0 \leq \frac{1}{d}$.

Însumând acum pentru toate submulțimile M_x , obținem:

$$K = \sum_{x \in S} \sum_{m \in M_x} d^{-l_m} \leq \sum_{x \in S} \frac{1}{d} = 1.$$

În cazul unei mulțimi M numărabile, construim

$$M_l = \{m \in M : |c(m)| \leq l\}, \quad l \in \mathbb{N}$$

și notăm

$$K_l = \sum_{m \in M_l} d^{-l_m}.$$

Deoarece, pentru fiecare $l \in \mathbb{N}$, $c|_{M_l}$ este un cod prefix, rezultă $K_l \leq 1$, $\forall l \in \mathbb{N}$. Dar $(K_l)_{l \in \mathbb{N}}$ este un subșir al șirului sumelor parțiale ale unei

Algoritmul *Construiește_cod*

intrarea: $(l_m)_{m \in M} \subseteq \mathbb{N}$ satisfăcând (2.2)

ieșirea: $c : M \rightarrow S^*$ cod prefix cu $|c(m)| = l_m, \forall m \in M$

algoritmul:

```

     $E := \{\varepsilon\}$ 
    pentru  $l = 1, \max_{m \in M} l_m$  execută
         $E' := \emptyset$ 
        pentru  $w \in E$  execută
            pentru  $x \in S$  execută
                 $E' := E' \cup \{w \cdot x\}$ 
            sfârșit pentru
        sfârșit pentru
     $E := E'$ 
    pentru  $m \in M : l_m = l$  execută
         $c(m) :=$  o valoare arbitrară din  $E$ 
         $E := E \setminus \{c(m)\}$ 
    sfârșit pentru
sfârșit pentru
sfârșit algoritm

```

Algoritmul 2.3: Construcția unui cod prefix cu lungimi date ale cuvintelor de cod

permutări a seriei cu termeni pozitivi $\sum_{m \in M} d^{-l_m}$. De aici rezultă că seria este convergentă și suma ei K este la rândul ei mai mică sau egală cu 1.

Să demonstrăm acum reciproca, și anume că inegalitatea lui Kraft implică existența unui cod prefix. Construcția codului va fi realizată de algoritmul 2.3. Demonstrăm în continuare corectitudinea acestui algoritm.

Vom nota în cele ce urmează cu E_k valoarea lui E în cadrul iterației $l = k$ imediat după execuția instrucțiunii $E := E'$.

Mai întâi, pentru a demonstra că lungimile cuvintelor de cod sunt într-adevăr cele dorite, să arătăm că toate cuvintele din E_k au lungime k . Într-adevăr, la prima iterație cuvintele din E_1 se obțin prin concatenarea câte unui simbol din S la șirul vid. Apoi, cuvintele din E_{k+1} se obțin din cuvintele rămase din E_k după atribuirea unora ca și cuvinte de cod prin adăugarea la final a câte unui simbol din S . Ca urmare, cuvintele din E_{k+1} sunt de lungime k .

Să arătăm acum că se obține un cod prefix. Dacă un cuvânt din E_k este atribuit unui mesaj, cuvântul de cod respectiv este eliminat din E_k . Cuvintele ce vor fi atribuite în continuare pot avea prefixe de lungime k doar dintre cuvintele rămase în E_k .

Mai trebuie arătat că există întotdeauna în E o valoare de atribuit lui $c(m)$. Pentru aceasta, vom arăta că

$$\sum_{\substack{m \in M \\ l_m \geq k}} d^{k-l_m} \leq |E_k| \quad (2.3)$$

La prima iterație, $|E_k| = d$ și

$$\sum_{\substack{m \in M \\ l_m \geq k}} d^{k-l_m} = d \cdot K \leq d = |E|$$

Presupunând că (2.3) are loc la iterația cu $l = k$, la iterația următoare, în care $l = k + 1$, avem

$$\begin{aligned} \sum_{\substack{m \in M \\ l_m \geq k+1}} d^{k+1-l_m} &= d \cdot \sum_{\substack{m \in M \\ l_m \geq k+1}} d^{k-l_m} = \\ &= d \left(\sum_{\substack{m \in M \\ l_m \geq k}} d^{k-l_m} - \sum_{\substack{m \in M \\ l_m = k}} d^{k-l_m} \right) = \\ &\leq d(|E_k| - |\{m \in M : l_m = k\}|) = \\ &= |E_{k+1}| \end{aligned}$$

unde ultima egalitate rezultă din modul de construcție a lui E_{k+1} din E_k prin eliminarea unui număr de elemente egal cu numărul de cuvinte de

cod de lungime k urmată de înlocuirea fiecărui cuvânt rămas cu d cuvinte obținute prin adăugarea fiecărei litere posibile din S .

Observăm acum că suma din inegalitatea (2.3) are un număr de termeni de valoare 1 egal cu numărul de cuvinte de lungime k de obținut și, ca urmare, există în E_k suficiente cuvinte. \diamond

EXEMPLUL 2.7: Dorim construirea unui cod prefix pentru mulțimea $M = \{a, b, c, d, e\}$ și mulțimea de simboluri de cod $S = \{0, 1\}$ cu următoarele lungimi ale cuvintelor de cod: $l_a = 3$, $l_b = 1$, $l_c = 3$, $l_d = 3$, $l_e = 3$.

Rezolvare: mai întâi verificăm dacă este satisfăcută inegalitatea lui Kraft:

$$\sum_{m \in M} |S|^{-l_m} = 2^{-3} + 2^{-1} + 2^{-3} + 2^{-3} + 2^{-3} = 1 \leq 1,$$

inegalitatea este satisfăcută și prin urmare există un cod prefix.

Construcția propriu-zisă este arătată în figura 2.4. Cerculețele desemnează nodurile corespunzătoare elementelor din mulțimea E .

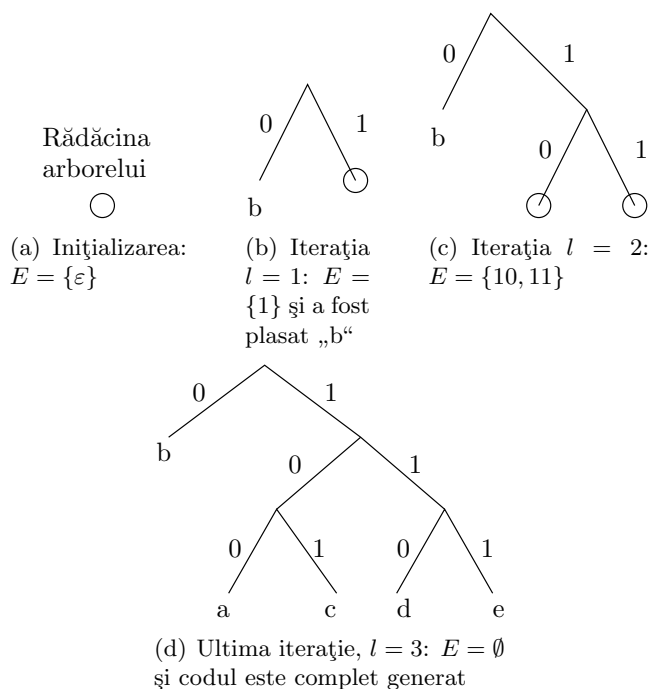


Figura 2.4: Construcția unui cod prefix cu lungimi fixate ale cuvintelor de cod (exemplul 2.7)

Vom arăta în continuare că inegalitatea lui Kraft este o condiție necesară pentru existența codurilor unic decodabile, nu doar a celor prefix. Avem:

Teorema 2.6 (McMillan) Pentru orice cod unic decodabil $c : M \rightarrow |S|$ are loc inegalitatea:

$$\sum_{m \in M} d^{-l_m} \leq 1 \quad (2.4)$$

unde $l_m = |c(m)|$, $m \in M$ și $d = |S|$.

Demonstrație. Considerăm mai întâi cazul când M este finită. Să notăm cu $E = \sum_{m \in M} d^{-l_m}$. Să luăm un $k \in \mathbb{N}^*$ arbitrar și să calculăm:

$$\begin{aligned} E^k &= \sum_{(m_1, \dots, m_k) \in M^k} d^{-l_{m_1}} \cdot \dots \cdot d^{-l_{m_k}} \\ &= \sum_{(m_1, \dots, m_k) \in M^k} d^{-(l_{m_1} + \dots + l_{m_k})} \end{aligned} \quad (2.5)$$

Regrupăm acum termenii din (2.5) după valorile sumei $l_{m_1} + \dots + l_{m_k}$. Pentru aceasta, vom nota cu $N(k, l)$ numărul de termeni din dezvoltarea (2.5) pentru care $l_{m_1} + \dots + l_{m_k} = l$. Cu alte cuvinte,

$$N(k, l) = |\{(m_1, \dots, m_k) \in M^k : l_{m_1} + \dots + l_{m_k} = l\}|.$$

Mai observăm că

$$k \leq l_{m_1} + \dots + l_{m_k} \leq l_{\max} \cdot k$$

unde l_{\max} este maximul lungimii cuvintelor de cod ($l_{\max} = \max_{m \in M} l_m$).

Obținem:

$$E^k = \sum_{l=k}^{l_{\max} \cdot k} N(k, l) \cdot d^{-l}. \quad (2.6)$$

Să observăm acum că $N(k, l)$ este numărul de șiruri de k mesaje pentru care lungimea codificării șirului este l . Deoarece codul este unic decodabil, aceste codificări sunt distincte și ca urmare $N(k, l)$ este cel mult egal cu numărul de șiruri distincte de l simboluri de cod, adică

$$N(k, l) \leq d^l.$$

Înlocuind în (2.6), obținem:

$$E^k \leq \sum_{l=k}^{l_{\max} \cdot k} d^l \cdot d^{-l} = l_{\max} \cdot k - k + 1 \leq l_{\max} \cdot k, \quad (2.7)$$

adică

$$E^k \leq l_{\max} \cdot k. \quad (2.8)$$

Această inegalitate are loc pentru orice $k \in \mathbb{N}^*$. Dacă am avea $E > 1$, atunci pentru un k suficient de mare am avea $E^k > l_{\max} \cdot k$; prin urmare $E \leq 1$.

Dacă M este numărabilă, construim mulțimile

$$M_k = \{m \in M : |c(m)| \leq k\}, \forall k \in \mathbb{N}$$

și notăm $E_k = \sum_{m \in M_k} d^{-l_m}$. Pentru fiecare $k \in \mathbb{N}$, M_k este finită și $c|_{M_k}$ este un cod unic decodabil. Ca urmare, $E_k \leq 1$ pentru fiecare $k \in \mathbb{N}$. Observăm acum că $E = \lim_{k \rightarrow \infty} E_k \leq 1$. \diamond

Corolarul 2.7 *Pentru orice cod unic decodabil, există un cod prefix cu aceleași lungimi ale cuvintelor de cod.*

2.3. Coduri optime

Deoarece stocarea sau transmiterea fiecărui simbol de cod implică un cost (timp necesar transmisiei, spațiu fizic pe suportul de informație, etc), este natural să căutăm un cod pentru care numărul de simboluri de cod necesare transmiterii șirului de mesaje al sursei este cât mai mic. Se impun însă câteva precizări cu privire la această minimizare.

Mai întâi, codul trebuie elaborat necunoscând informația particulară pe care urmează s-o trimită sursa. Prin urmare, nu se poate cere minimizarea lungimii reprezentării informației transmise efectiv de sursă. Se va minimiza deci numărul mediu de biți necesari reprezentării unui mesaj al sursei.

În al doilea rând, acest număr mediu de biți se consideră în sens probabilistic, de valoare medie a unei variabile aleatoare. Anume, fiecare mesaj al sursei poate fi considerat o variabilă aleatoare cu valori din mulțimea M de mesaje ale sursei. Lungimea reprezentării mesajului este de asemenea o variabilă aleatoare, a cărei valoare medie este ceea ce dorim să minimizăm.

Probabilitățile diferitelor mesaje ale sursei se pot estima pe diverse căi fie analizând teoretic fenomenele pe baza cărora funcționează sursa, fie analizând statistic șiruri de mesaje trimise de sursă. Ca exemplu, dacă mesajele sursei sunt litere ce alcătuiesc un text într-o anumită limbă, se poate determina statistic frecvența fiecărei litere, precum și frecvențele unor succesiuni de litere.

2.3.1. Cantitatea de informație

Cantitatea de informație purtată de un mesaj este o măsură a incertitudinii pe care destinatarul o avea imediat înainte de primirea mesajului și care este eliminată în urma primirii mesajului.

Cantitatea de informație purtată de un mesaj trebuie deci să fie mică dacă pentru destinatar evenimentul anunțat de mesaj era aproape sigur și mare dacă este un eveniment total neașteptat. Este de dorit, de asemenea, ca măsura informației să fie aditivă, în sensul că privind ca un singur mesaj o succesiune de două mesaje, cantitatea de informație purtată de mesajul compus să fie suma cantităților de informație purtate de cele două mesaje separat.

Așa cum vom vedea în continuare, cantitatea de informație purtată de un mesaj va fixa o limită inferioară teoretică a numărului de simboluri de cod necesare codificării mesajului.

De notat că cantitatea de informație nu are nici o legătură cu utilitatea informației.

Definiția 2.8 *Fie o sursă care emite un șir de mesaje $m_1, m_2, \dots, m_t \in M$. Cantitatea de informație adusă de mesajul m_t este*

$$\text{info}(m_t) = -\log_2 \Pr(m_t | m_1, m_2, \dots, m_{t-1}).$$

Altfel spus, cantitatea de informație adusă de un mesaj m_t în contextul (adică urmând după) m_1, m_2, \dots, m_{t-1} este minus logaritmul probabilității ca al t -lea mesaj să fie m_t , condiționată de faptul că mesajele precedente au fost m_1, m_2, \dots, m_{t-1} .

În cazul unei surse *ergotice*, adică pentru care probabilitatea ca un mesaj să aibă o anumită valoare este independentă de mesajele anterioare și de poziția (numărul de ordine) mesajului în șirul de mesaje, putem, pentru fiecare $m \in M$, să notăm cu p_m probabilitatea ca un anumit mesaj din șirul de mesaje să aibă valoarea m . Atunci cantitatea de informație adusă de un mesaj m este $\text{info}(m) = -\log_2 p_m$.

Unitatea de măsură pentru cantitatea de informație este *bitul*.

A nu se confunda bitul cu sensul de unitate de măsură pentru cantitatea de informație cu bitul cu sensul de cifră binară. Există o legătură între aceste noțiuni, și anume, așa cum vom vedea, pentru a transmite un bit de informație avem nevoie cel puțin de un bit (cifră binară).

EXEMPLUL 2.8: Dacă emițătorul anunță receptorului rezultatul aruncării unei monede, mesajul *a căzut cu fața în sus* poartă o cantitate de informație egală cu $-\log_2 \frac{1}{2} = -(-1) = 1$ bit.

EXEMPLUL 2.9: În textul acestei lucrări, 10,7% dintre litere sunt „a”, și doar 1,1% sunt „b”. Cu aceste cunoștințe, receptorul se va aștepta de la fiecare literă să fie „a” cu probabilitate de 10,7% și „b” cu probabilitate de 1,1%. În aceste condiții, fiecare literă „a” poartă $-\log_2 0,107 \approx 3,224$ biți de informație, și fiecare literă „b” poartă $-\log_2 0,011 \approx 6,5$ biți.

EXEMPLUL 2.10: Presupunem că emițătorul informează receptorul asupra rezultatului aruncării unui zar. Dacă emițătorul trimite mesajul *numărul este între 1 și 4* cantitatea de informație este $-\log_2 \frac{4}{6} \approx 0,58$ biți. Dacă emițătorul anunță acum că numărul este 3, probabilitatea acestui caz, cu informațiile disponibile imediat înainte, este $\frac{1}{4}$, de unde cantitatea de informație purtată de mesajul *numărul este 3* este $-\log_2 \frac{1}{4} = 2$ biți. Să observăm că, dacă emițătorul ar fi spus de la început *numărul este 3*, cantitatea de informație transmisă ar fi fost $-\log_2 \frac{1}{6} \approx 2,58$ biți.

Definiția 2.9 Fie o sursă de informație ce emite mesaje dintr-o mulțime M , fiecare mesaj $m \in M$ având o probabilitate p_m de-a fi emis. Se numește entropia sursei de informație cantitatea

$$H = - \sum_{m \in M} p_m \cdot \log p_m \quad (2.9)$$

Cu alte cuvinte, entropia este cantitatea medie de informație per mesaj.

2.3.2. Lungimea medie a cuvintelor de cod

Definiția 2.10 Fie o sursă ce emite mesaje dintr-o mulțime M . Pentru fiecare $m \in M$, fie p_m probabilitatea mesajului m și fie $c : M \rightarrow S^*$ un cod unic decodabil. Se numește lungimea medie a cuvintelor codului c valoarea

$$\bar{l} = \sum_{m \in M} p_m \cdot |c(m)|.$$

Definiția 2.11 Un cod unic decodabil $c : M \rightarrow S^*$ se numește cod optim dacă lungimea medie a cuvintelor sale este mai mică sau egală decât lungimea medie a cuvintelor oricărui cod unic decodabil $c' : M \rightarrow S^*$.

Există următoarea limită inferioară pentru lungimea medie a cuvintelor de cod:

Teorema 2.12 *Fie o sursă ce emite mesaje dintr-o mulțime M , fie H entropia sursei și fie $c : M \rightarrow S^*$ un cod unic decodabil. Atunci lungimea medie \bar{l} a cuvintelor codului c satisface*

$$\bar{l} \geq \frac{H}{\log_2 |S|}. \quad (2.10)$$

În particular, dacă $|S| = 2$, atunci rezultă $\bar{l} \geq H$. Cu alte cuvinte avem nevoie cel puțin de un simbol binar (un bit) pentru a transmite un bit de informație.

Definiția 2.13 *Se numește eficiența unui cod raportul $\eta = \frac{H}{\bar{l} \log_2 |S|}$, unde H este entropia sursei, \bar{l} este lungimea medie a cuvintelor de cod, iar S este mulțimea simbolurilor de cod.*

Se numește redundanța relativă valoarea $1 - \eta$.

Eficiența și redundanța relativă sunt numere cuprinse între 0 și 1.

Valoarea minimă, dată teorema 2.12, pentru lungimea medie a cuvintelor de cod poate fi atinsă efectiv, adică se poate obține eficiența $\eta = 1$, doar în anumite cazuri. Motivul pentru care ea nu poate fi întotdeauna atinsă este dată de natura discretă a simbolurilor de cod. Ideal, lungimea cuvintelor de cod ar trebui să fie $l_m = -\log_{|S|} p_m$. Pentru aceste valori inegalitatea lui Kraft este satisfăcută:

$$\sum_{m \in M} |S|^{-l_m} = \sum_{m \in M} |S|^{-(-\log_{|S|} p_m)} = \sum_{m \in M} p_m = 1 \leq 1,$$

prin urmare ar exista un cod unic decodabil și limita din teorema 2.12 ar fi atinsă:

$$\begin{aligned} \bar{l} &= \sum_{m \in M} p_m \cdot (-\log_{|S|} p_m) = - \sum_{m \in M} p_m \cdot \frac{\log_2 p_m}{\log_2 |S|} \\ &= \frac{1}{\log_2 |S|} \cdot \left(- \sum_{m \in M} p_m \cdot \log_2 p_m \right) = \frac{H}{\log_2 |S|}. \end{aligned}$$

Acest lucru se poate realiza însă numai dacă $l_m = -\log_{|S|} p_m$ sunt toate întregi.

În cazul general putem doar să alegem ca lungimi ale cuvintelor de cod valorile mai mari, $l_m = \lceil -\log_{|S|} p_m \rceil$. Pentru aceste valori avem

$$-\log_{|S|} p_m \leq l_m < -\log_{|S|} p_m + 1$$

de unde rezultă:

Teorema 2.14 *Fie o sursă ergotică ce emite mesaje dintr-o mulțime M , fie H entropia sursei și fie S o mulțime de simboluri de cod. Atunci există un cod $c : M \rightarrow S^*$ unic decodabil a cărui lungime medie \bar{l} a cuvintelor de cod satisface*

$$\frac{H}{\log_2 |S|} \leq \bar{l} < \frac{H}{\log_2 |S|} + 1. \quad (2.11)$$

Rezultatul teoremei precedente poate fi îmbunătățit dacă în loc să considerăm mesajele sursei ca fiind mesajele din M considerăm succesiuni de mesaje din M , construim un cod pentru acestea din urmă și determinăm raportul dintre lungimea medie a cuvântului de cod și numărul de mesaje din M codificate prin acesta. În detaliu, construcția este următoarea:

Fixăm $k \in \mathbb{N}$. Considerăm o a doua sursă, ale cărei mesaje vor fi succesiuni de k mesaje ale sursei originale. Mulțimea de mesaje ale noii surse este prin urmare M^k . Probabilitățile mesajelor sunt $p_{(m_1, \dots, m_k)} = p_{m_1} \cdot \dots \cdot p_{m_k}$. Vom nota cu H_k entropia noii surse. Avem

$$\begin{aligned} H_k &= - \sum_{(m_1, \dots, m_k) \in M^k} p_{(m_1, \dots, m_k)} \log_2 p_{(m_1, \dots, m_k)} = \\ &= - \sum_{(m_1, \dots, m_k) \in M^k} p_{m_1} \cdot \dots \cdot p_{m_k} \cdot (\log_2 p_{m_1} + \dots + \log_2 p_{m_k}) = \\ &= - \sum_{i=1}^k \sum_{(m_1, \dots, m_k) \in M^k} p_{m_1} \cdot \dots \cdot p_{m_k} \cdot \log_2 p_{m_i} = \\ &= \sum_{i=1}^k \left(\sum_{(m_1, \dots, m_{i-1}, m_{i+1}, \dots, m_k) \in M^{k-1}} p_{m_1} \cdot \dots \cdot p_{m_{i-1}} \cdot p_{m_{i+1}} \cdot \dots \cdot p_{m_k} \right) \cdot \\ &\quad \cdot \left(- \sum_{m_i \in M} p_{m_i} \cdot \log_2 p_{m_i} \right) = \\ &= \sum_{i=1}^k 1 \cdot H = \\ &= k \cdot H \end{aligned}$$

Conform teoremei 2.14, există un cod $c : M^k \rightarrow S^*$ pentru care lungimea medie a cuvintelor de cod, $\bar{l}^{(k)}$, satisface

$$\frac{H_k}{\log_2 |S|} \leq \bar{l}^{(k)} < \frac{H_k}{\log_2 |S|} + 1.$$

Numărul mediu de simboluri de cod utilizate pentru a transmite un mesaj din M este $\frac{\overline{l^{(k)}}}{k}$, care este delimitat de

$$\frac{H}{\log_2 |S|} \leq \frac{\overline{l^{(k)}}}{k} < \frac{H}{\log_2 |S|} + \frac{1}{k}.$$

Prin urmare, pentru orice $\varepsilon > 0$, putem alege un $k \in \mathbb{N}$ astfel încât codificând câte k mesaje succesive din M să obținem un număr de simboluri pe mesaj încadrat între

$$\frac{H}{\log_2 |S|} \leq \frac{\overline{l^{(k)}}}{k} < \frac{H}{\log_2 |S|} + \varepsilon.$$

2.3.3. Generarea codului optim prin algoritmul lui Huffman

Ne vom ocupa în continuare de generarea efectivă a unui cod optim pentru o sursă cu probabilități cunoscute ale mesajelor. Algoritmul cel mai utilizat pentru aceasta este algoritmul lui Huffman (algoritmul 2.4).

Ca idee de bază, algoritmul lui Huffman construiește arborele unui cod prefix în modul următor: pleacă de la n arbori (n fiind numărul de mesaje) fiecare constând doar din rădăcină, după care unește câte $|S|$ arbori ($|S|$ fiind numărul de simboluri de cod) ca subarbori ai unui nod nou creat. La fiecare unire, se iau arborii cu sumele probabilităților mesajelor asociate cele mai mici; în caz de egalitate între probabilități, se iau oricare dintre arborii de probabilități egale. Algoritmul se termină în momentul în care rămâne un singur arbore.

Dacă $|S| > 2$ și n nu este de forma $(|S| - 1)k + 1$ cu $k \in \mathbb{N}$, astfel că nu s-ar putea uni de fiecare dată exact $|S|$ arbori, la prima unire se vor uni $(n - 2) \bmod (|S| - 1) + 2$ arbori, astfel încât la toate celelalte uniri să se unească câte $|S|$ arbori și în final să rămână exact un arbore.

EXEMPLUL 2.11: Fie o sursă având mulțimea mesajelor posibile

$$M = \{a, b, c, d, e\}$$

cu probabilitățile corepunzătoare $p_a = 0,35$, $p_b = 0,15$, $p_c = 0,15$, $p_d = 0,15$, $p_e = 0,20$ și fie alfabetul canalului $S = \{0, 1\}$. Generarea codului optim se face astfel (vezi fig. 2.5):

- În prima fază creem noduri izolate corespunzătoare mesajelor sursei (fig. 2.5(a));
- Alegem două noduri cu cele mai mici probabilități și le unim. Acestea pot fi „b” cu „c”, „b” cu „d” sau „c” cu „d”. Oricare dintre alegeri duce la un

Algoritmul Huffman

intrarea: M mulțime finită de mesaje

$p_m \in (0, 1)$, $m \in M$, probabilitățile mesajelor; $\sum_{m \in M} p_m = 1$ $S = \{s_1, s_2, \dots, s_d\}$ mulțime finită de simboluri de cod, $d \geq 2$

ieșirea: $c : M \rightarrow S^*$ cod prefix

algoritmul:

$E := M$

$d' := (|M| - 2) \bmod (|S| - 1) + 2$

cât timp $|E| > 1$ execută

alege $e_1, \dots, e_{d'} \in E$ cu $p_{e_i} \leq p_{e^*}$, $\forall i \in \{1, \dots, d'\}$, $\forall e^* \in E \setminus \{e_1, \dots, e_{d'}\}$

crează t unic

pentru $i \in \{1, \dots, d'\}$ execută

pune e_i ca fiu al lui t

$s_{(t, e_i)} := s_i$

sfârșit pentru

$p_t := \sum_{i=1}^{d'} p_{e_i}$

$E := (E \setminus \{e_1, \dots, e_{d'}\}) \cup \{t\}$

$d' := d$

sfârșit cât timp

$c :=$ codul prefix asociat unicului arbore din E

sfârșit algoritm

Algoritmul 2.4: Algoritmul lui Huffman

cod optim. Să alegem „b” cu „c”. Calculăm și probabilitatea arborelui rezultat: $0,15 + 0,15 = 0,3$. (fig. 2.5(b)).

- În continuare unim din nou arborii de probabilități minime; acum aceștia sunt „d” și „e” (fig. 2.5(c)).
- Avem acum două posibilități: arborele ce conține pe „b” și pe „c” poate fi unit fie cu arborele format din „a”, fie cu arborele format din „d” și „e”. Alegem a doua variantă.
- În final unim cei doi arbori rămași.

Avem acum codurile mesajelor: $c(a) = 0$, $c(b) = 100$, $c(c) = 101$, $c(d) = 110$, $c(e) = 111$. Lungimea medie a cuvintelor de cod este

$$\bar{l} = 0,35 \cdot 1 + 0,15 \cdot 3 + 0,15 \cdot 3 + 0,15 \cdot 3 + 0,2 \cdot 3 = 2,3$$

Pentru comparație, entropia este

$$H = -0,35 \log_2 0,35 - 0,15 \log_2 0,15 - 0,15 \log_2 0,15 - 0,15 \log_2 0,15 - 0,2 \log_2 0,2 \approx 2,226121$$

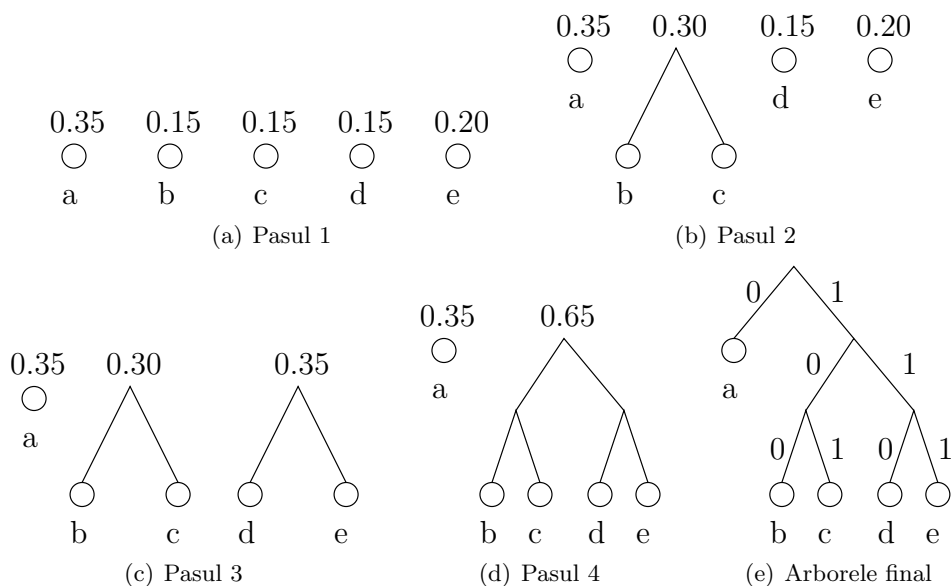


Figura 2.5: Funcționarea algoritmului Huffman, exemplul 2.11

Dacă la pasul 4 s-ar fi ales cealaltă posibilitate, ar fi rezultat mulțimea de arbori din figura 2.6(a) și în final arborele asociat codului prefix din figura 2.6(b). Să observăm că se obține exact aceeași lungime medie a cuvintelor de cod:

$$\bar{l} = 0,35 \cdot 2 + 0,15 \cdot 3 + 0,15 \cdot 3 + 0,15 \cdot 2 + 0,2 \cdot 2 = 2,3$$

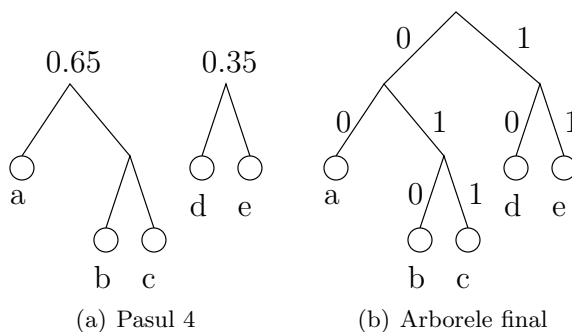


Figura 2.6: Variantă alternativă pentru pașii 4 și 5 (exemplul 2.11)

EXEMPLUL 2.12: Fie o sursă având mulțimea mesajelor posibile

$$M = \{a, b, c, d, e, f\}$$

cu probabilitățile corepsunzătoare $p_a = 0,4$, $p_b = 0,15$, $p_c = 0,15$, $p_d = 0,1$, $p_e = 0,1$, $p_f = 0,1$ și fie alfabetul canalului $S = \{0, 1, 2\}$.

Construcția codului prin algoritmul lui Huffman este prezentată în figura 2.7. Lungimea medie a cuvintelor de cod este $\bar{l} = 1,6$, entropia este $H \approx 2,346439$ și avem

$$\frac{H}{\log_2 |S|} \approx \frac{2,346439}{1,5849625} \approx 1,4804382 \leq 1,6 = \bar{l}$$

Teorema 2.15 *Codul obținut prin algoritmul Huffman este optim.*

Pentru demonstrație avem nevoie de câteva leme ce descriu proprietăți ale unui cod optim. În cele ce urmează vom nota cu $L(c)$ lungimea medie a cuvintelor unui cod c .

Lema 2.16 *Fie M mulțimea mesajelor sursei, fie p_m , $m \in M$, probabilitățile mesajelor sursei, fie S alfabetul canalului și fie $c : M \rightarrow S^*$ un cod optim. Pentru orice mesaje $m_1, m_2 \in M$, dacă $p_{m_1} < p_{m_2}$ atunci $|c(m_1)| \geq |c(m_2)|$.*

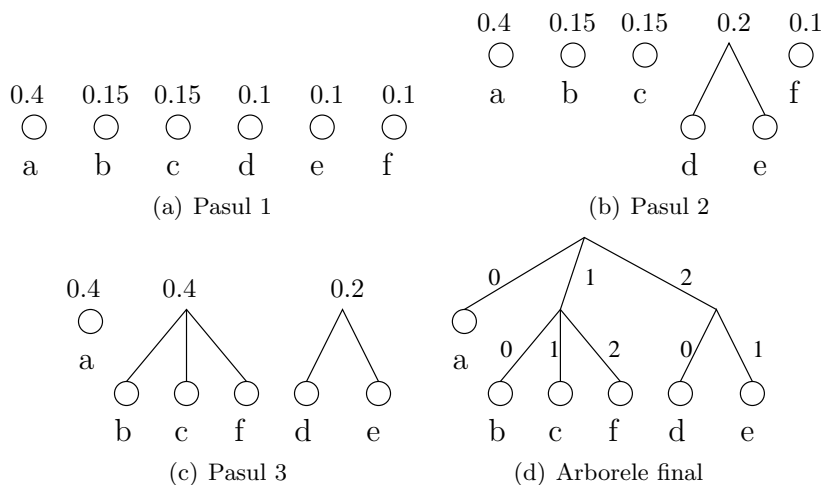


Figura 2.7: Funcționarea algoritmului lui Huffman, exemplul 2.12

Demonstrație. Presupunem contrariul: $\exists m_1, m_2 \in M, p_{m_1} < p_{m_2}$ și $|c(m_1)| < |c(m_2)|$. Construim atunci un alt cod, $c' : M \rightarrow S^*$, prin interschimbarea cuvintelor de cod asociate mesajelor m_1 și m_2 :

$$c'(m) = \begin{cases} c(m_2) & , m = m_1 \\ c(m_1) & , m = m_2 \\ c(m) & , m \in M \setminus \{m_1, m_2\} \end{cases}$$

Avem

$$\begin{aligned} L(c') &= \sum_{m \in M} p_m \cdot |c'(m)| = \\ &= L(c) - p_{m_1}|c(m_1)| - p_{m_2}|c(m_2)| + p_{m_1}|c(m_2)| + p_{m_2}|c(m_1)| = \\ &= L(c) + (p_{m_1} - p_{m_2})(|c(m_2)| - |c(m_1)|) < \\ &< L(c) \end{aligned}$$

adică c' are lungimea cuvintelor de cod mai mică decât c , de unde rezultă că c nu este cod optim. \diamond

Lema 2.17 Fie M mulțimea mesajelor sursei, $|M| \geq 2$, fie S alfabetul canalului, fie $c : M \rightarrow S^*$ un cod optim și fie l_{\max} lungimea celui mai lung cuvânt al codului c ($l_{\max} = \max_{m \in M} |c(m)|$). Atunci există cel puțin $(n - 2) \bmod (|S| - 1) + 2$ cuvinte de cod de lungime l_{\max} .

Demonstrație. Conform corolarului 2.7, există un cod prefix cu aceleași lungimi ale cuvintelor de cod ca și codul c . Deoarece ne interesează doar

lungimile cuvintelor de cod, putem, fără a restrânge generalitatea, să presupune că c este cod prefix.

Considerăm arborele asociat codului c . Vom numi *numărul de poziții libere* ale unui nod intern (un nod ce are cel puțin un fiu) valoarea $|S|$ minus numărul de fii. Observăm următoarele:

- Cu excepția penultimului nivel, fiecare nod intern are zero poziții libere. Într-adevăr, în caz contrar s-ar putea muta o frunză de pe ultimul nivel ca descendent al nodului cu cel puțin o poziție liberă; prin această operație ar scădea lungimea cuvântului de cod corespunzător și ca urmare ar scădea lungimea medie a cuvintelor de cod, contrazicând ipoteza că c este optim.
- Suma numerelor pozițiilor libere ale nodurilor penultimului nivel este cel mult $|S| - 2$. Dacă arborele are înălțime 1, atunci unicul nod intern este rădăcina, aceasta are cel puțin 2 fii, deoarece $|M| \geq 2$, și, în consecință, numărul pozițiilor libere este cel mult $|S| - 2$. Considerăm acum un arbore de înălțime cel puțin 2 și să presupunând prin absurd că am avea $|S| - 1$ poziții libere. Fie t un nod intern de pe penultimul nivel și fie k numărul de descendenți ai săi. Nodul t are $|S| - k$ poziții libere, deci mai rămân cel puțin $k - 1$ poziții libere la celelalte noduri. Mutăm $k - 1$ dintre descendenții lui t pe poziții libere ale altor noduri ale penultimului nivel; lungimile cuvintelor de cod se păstrează. Acum t are un singur descendent. Putem elimina nodul t subordonând unicul său descendent direct parintelui lui t ; în acest fel lungimea cuvântului de cod corespunzător scade cu 1 și lungimea medie a cuvântului de cod scade cu o valoare nenulă, ceea ce contrazice din nou ipoteza că c e optim.

Pentru un arbore cu k noduri interne și cu numărul total de poziții libere 0, numărul de frunze, care este egal cu numărul n de mesaje, este $n = k \cdot (|S| - 1) + 1$. Acest lucru se demonstrează imediat prin inducție după k . Dacă arborele are în total j poziții libere, prin completarea acestora cu frunze ar rezulta un arbore cu 0 poziții libere și $n + j$ frunze; prin urmare

$$n = k \cdot (|S| - 1) + 1 - j$$

Notând $q = |S| - j - 2$, avem

$$n = k \cdot (|S| - 1) + q - |S| + 3 = (k - 1) \cdot (|S| - 1) + 2 + q$$

Deoarece $0 \leq j \leq |S| - 2$ rezultă $0 \leq q \leq |S| - 2$ de unde

$$q = (n - 2) \bmod (|S| - 1)$$

Penultimul nivel conține cel puțin un nod intern, de unde rezultă că pe ultimul nivel există cel puțin $|S| - j$ frunze. Cum $|S| - j = q + 2$ rezultă că pe ultimul nivel avem cel puțin

$$q + 2 = (n - 2) \bmod (|S| - 1) + 2$$

frunze.◇

Demonstrația teoremei 2.15. Fie n numărul de mesaje. Vom demonstra prin inducție după numărul $k = \left\lceil \frac{n-1}{|S|-1} \right\rceil$.

Pentru $k = 1$, adică $n \leq |S|$, algoritmul lui Huffman face o singură unificare, rezultând cuvinte de cod de lungime 1 pentru toate mesajele. Un astfel de cod este optim, deoarece cuvinte de cod de lungime mai mică decât 1 nu sunt permise.

Presupunem acum că algoritmul Huffman generează codul optim pentru un k dat și să-i demonstrăm optimalitatea pentru $k + 1$. Să luăm deci o mulțime de mesaje M cu $k(|S| - 1) + 1 \leq |M| \leq (k + 1)(|S| - 1)$, să notăm cu p_m , $m \in M$, probabilitățile mesajelor, să notăm cu c_h codul generat de algoritmul lui Huffman și cu c_o un cod prefix optim pentru aceeași mulțime de mesaje și aceleași probabilități și să notăm cu $L(c_h)$, respectiv $L(c_o)$ lungimile medii ale cuvintelor de cod corespunzătoare. Avem de demonstrat că $L(c_h) \leq L(c_o)$.

Deoarece c_o este un cod optim, aplicând lema 2.17 deducem că c_o are cel puțin $(n - 2) \bmod (|S| - 1) + 2$ cuvinte de lungime maximă. Din lema 2.16, deducem că acestea sunt cuvintele corespunzătoare mesajelor cu probabilitățile cele mai mici, adică fie mesajele $e_1, \dots, e_{d'}$ alese de algoritmul lui Huffman pentru prima unificare, fie mesaje de aceleași probabilități; în al doilea caz putem, prin interschimbări de cuvinte de cod, să facem ca cele $(n - 2) \bmod (|S| - 1) + 2$ cuvinte de lungime maximă din c_o să fie cele alese în prima etapă a algoritmului lui Huffman, fără ca prin aceasta să pierdem optimalitatea lui c_o . De asemenea, prin interschimbări de cuvinte de cod, putem face ca celor $(n - 2) \bmod (|S| - 1) + 2$ mesaje alese de algoritmul lui Huffman să le corespundă prin c_o cuvinte de cod ce diferă doar prin ultimul simbol.

Creem acum un cod $c'_o : (M \setminus \{e_1, \dots, e_{d'}\}) \cup \{t\} \rightarrow S^*$, unde t este un obiect nou introdus, dând ca valoare pentru $c(t)$ prefixul comun al lui $c(e_1), \dots, c(e_{d'})$. În același mod, creem un cod c'_h pornind de la c_h . Observăm acum că, notând $p_t = \sum_{i=1}^{d'} p_{e_i}$, avem $L(c'_o) = L(c_o) - p_t$ și analog, $L(c'_h) = L(c_h) - p_t$. Să mai remarcăm că c'_h este codul produs de algoritmul lui Huffman pentru mulțimea de mesaje $(M \setminus \{e_1, \dots, e_{d'}\}) \cup \{t\}$ și, conform ipotezei de inducție, el este optim; prin urmare $L(c'_h) \leq L(c'_o)$. De aici rezultă $L(c_h) \leq L(c_o)$, deci codul obținut prin algoritmul lui Huffman este optim.◇

2.3.4. Compresia fișierelor

Codarea optimală este ceea ce face orice program de compresie a fișierelor. Algoritmul Huffman este folosit aproape de orice algoritm de compresie, însă de regulă nu direct asupra octeților din datele de comprimat.

Algoritmii de compresie utilizați în practică se folosesc și de dependențele între octeții succesivi.

Utilizarea oricărui cod presupune că receptorul cunoaște codul folosit de emițător. Transmiterea separată a codului către receptor riscă să contrabalanseze câștigul obținut prin codare optimală. *Metodele adaptative* presupun că emițătorul începe emisia cu un cod standard, după care îl modifică pentru a-l optimiza conform frecvențelor observate în date. Dacă algoritmul de generare a codului este fixat și codul folosit la un moment dat depinde doar de datele trimise (codate) deja, atunci receptorul poate recalcula codul folosit de emițător (folosind același algoritm ca și emițătorul).

De notat că nici un cod nu poate folosi mai puțini biți pentru codare decât cantitatea de informație transmisă. În lipsa redundanței, nu e posibilă compresia. Ca o consecință, nici un program de compresie nu poate comprima un șir aleator de octeți.

2.4. Coduri detectoare și corectoare de erori

Vom studia în cele ce urmează problema transmisiei informației în situația unui canal discret, dar care alterează șirul de simboluri de cod transmise. În practică, o astfel de alterare este efectul zgomotului ce se suprapune peste semnalul transmis de nivelul fizic (vezi capitolul 3); din acest motiv un astfel de canal se numește *canal cu zgomot* sau *canal cu perturbații*.

Pentru transmiterea corectă a datelor printr-un canal cu perturbații este necesar un mecanism care să permită fie *detectarea* fie *corectarea* erorilor de transmisie. Ambele mecanisme permit receptorului să determine dacă un cuvânt de cod a fost transmis corect sau a fost alterat de către canal. În cazul unui cuvânt alterat:

- *detectarea erorilor* presupune că receptorul informează destinația de acest lucru;
- *corectarea erorilor* presupune că receptorul determină cuvântul de cod cel mai probabil să fi fost transmis de către emițător și dă sursei mesajul corespunzător aceluși cuvânt.

Ca principiu, atât detectarea cât și corectarea erorilor se bazează pe un cod în care nu orice secvență (de lungime adecvată) de simboluri de cod este cuvânt de cod și, ca urmare, alterările cele mai probabile ale șirului de simboluri transmis conduc la secvențe de simboluri de cod care nu constituie cuvinte de cod. Desigur, întotdeauna rămâne posibilitatea ca erorile de transmisie să transforme un cuvânt de cod în alt cuvânt de cod și, ca urmare, erorile

să scape nedetectate. Cu un cod bine ales, însă, probabilitatea unei erori nedetectate poate fi făcută suficient de mică. Evident, pentru aceasta este necesar ca mulțimea cuvintelor de cod să fie o submulțime „rară“ a mulțimii secvențelor de simboluri de cod.

Prin urmare, posibilitățile de detectare a erorilor țin de construcția codului. De aici denumirea de *cod detector de erori*, respectiv *cod corector de erori*. Deoarece la orice cod detector sau corector de erori mulțimea șirurilor de cuvinte de cod este o submulțime strictă a mulțimii șirurilor arbitrare de simboluri de cod, rezultă că orice cod detector sau corector de erori are redundanță.

În cele ce urmează vom considera alfabetul canalului $S = \{0, 1\}$.

2.4.1. Modelul erorilor

Construcția codului detector sau corector de erori trebuie făcută în așa fel încât să facă suficient de mică probabilitatea unei erori nedetectate. Este deci esențială construcția unui model probabilistic al erorilor, adică determinarea, pentru fiecare modificare a șirului de simboluri transmis de canal, a probabilității corespunzătoare.

Distingem următoarele tipuri de erori:

- *erori individuale*, care schimbă valoarea unui bit din 0 în 1 sau reciproc;
- *rafale de erori*, care schimbă o parte dintr-un șir de biți (nu neapărat toți). Lungimea rafalei este numărul de biți dintre primul și ultimul bit modificat;
- *erori de sincronizare*, care determină pierderea unui bit sau introducerea unui bit, împreună cu decalarea corespunzătoare a biților următori.

Transmisia unui șir de biți poate fi afectată simultan de mai multe erori distincte.

O modelare simplă a erorilor este aceea în care se presupune că există doar erori individuale și că probabilitatea ca o eroare să afecteze un bit este aceeași pentru toți biții și independentă de valorile biților și de pozițiile celorlalte erori. Cu alte cuvinte, fiecare bit are o probabilitate p să fie inversat (dacă emițătorul a transmis un 1 receptorul să primească 0 și dacă emițătorul a transmis 1 receptorul să primească 0) și $1 - p$ să fie transmis corect.

Erorile fiind independente, probabilitatea ca o secvență de l biți să se transmită corect este $p_0 = (1 - p)^n$, probabilitatea ca acea secvență să fie afectată de exact o eroare este $p_1 = lp(1 - p)^{l-1} \approx lp$, probabilitatea să se producă două erori este $p_2 = \frac{l(l-1)}{2}p^2(1 - p)^{l-2}$ și, în general, probabilitatea

să se producă exact k erori este

$$p_k = \frac{l!}{k!(l-k)!} p^k (1-p)^{l-k},$$

conform distribuției binomiale.

Observăm că, întrucât $p \ll 1$, pentru l suficient de mic avem $p_0 \gg p_1 \gg p_2 \gg \dots$, adică probabilitatea de-a avea mai mult de câteva erori este extrem de mică.

2.4.2. Principiile codurilor detectoare și corectoare de erori

Vom analiza doar cazul codurilor de lungime fixă pentru mulțimea de simboluri $S = \{0, 1\}$. Notăm cu l lungimea cuvintelor de cod. Prin urmare, mulțimea cuvintelor de cod, W , este o submulțime a mulțimii șirurilor de simboluri de cod de lungime l : $W \subseteq \{0, 1\}^l$.

Ca model al erorilor, considerăm că avem doar erori individuale, independente (cazul studiat în paragraful anterior).

Deoarece nu avem erori de sincronizare și deoarece toate cuvintele de cod au aceeași lungime l , receptorul poate departaja cuvintele de cod succesive, independent de erorile de transmisie survenite. Ne vom pune deci doar problema detectării sau corectării erorilor ce afectează un cuvânt de cod de lungime fixă l .

Întrucât probabilitatea de-a avea k sau mai multe erori scade foarte repede o dată cu creșterea lui k , se alege o valoare k astfel încât probabilitatea de-a avea k sau mai multe erori este neglijabil de mică și se construiește codul presupunând că nu se produc mai mult de $k - 1$ erori.

Definiția 2.18 Spunem despre codul $c : M \rightarrow \{0, 1\}^l$ că detectează k erori individuale dacă, pentru orice cuvânt de cod $w \in W = c(M)$, prin transformarea lui w ca urmare a k sau mai puține erori, cuvântul rezultat w' nu este cuvânt de cod: $w' \notin W$.

Pentru a determina numărul de erori detectate de un cod, definim următoarele:

Definim pe $\{0, 1\}^l$ o funcție distanță:

$$d(u, v) = \sum_{i=1}^l |u_i - v_i|,$$

unde $u = (u_1, u_2, \dots, u_l)$ și $v = (v_1, v_2, \dots, v_l)$. Astfel, distanța între două cuvinte este numărul de erori individuale necesare pentru a transforma primul cuvânt în cel de-al doilea.

Notăm acum

$$d_{\min}(W) = \min_{\substack{u,v \in W \\ u \neq v}} d(u,v),$$

unde W este mulţimea cuvintelor de cod ale codului considerat.

Propoziţia 2.19 *Fie codul $c : M \rightarrow \{0,1\}^l$ şi $W = c(M)$. Codul c detectează k erori dacă şi numai dacă $d_{\min}(W) \geq k + 1$.*

Să examinăm acum codurile corectoare de erori.

Definiţia 2.20 *Spunem despre codul $c : M \rightarrow \{0,1\}^l$ că corectează k erori individuale dacă, pentru orice cuvânt de cod $w \in W = c(M)$, prin transformarea lui w ca urmare a k sau mai puţine erori cuvântul rezultat w' are proprietatea că w este cel mai apropiat cuvânt de w' din W :*

$$\forall w^s \in W, d(w', w^s) \geq d(w', w).$$

Propoziţia 2.21 *Fie codul $c : M \rightarrow \{0,1\}^l$ şi $W = c(M)$. Codul c corectează k erori dacă şi numai dacă $d_{\min}(W) \geq 2k + 1$.*

Să analizăm acum eficienţa codului. De obicei, datele utile pentru un cod detector sau corector de erori sunt şiruri de biţi, obţinuţi prin codificarea datelor din universul aplicaţiei. Ca urmare, mulţimea mesajelor este mulţimea şirurilor de n biţi, $M = \{0,1\}^n$, pentru o valoare n dată. Mesajele sunt echiprobabile, probabilitatea oricărui mesaj fiind aceeaşi: $p_m = \frac{1}{|M|} = 2^{-n}$, $\forall m \in M$. Ca urmare, eficienţa codului este

$$\frac{H}{l} = \frac{n}{l}.$$

Să mai notăm că $|M| = |W| = 2^n$.

Construcţia efectivă a unui cod detector sau corector de erori cuprinde două aspecte:

- construcţia unei mulţimi $W \subseteq \{0,1\}^l$ cu $d_{\min}(W)$ suficient de mare pentru numărul de erori de detectat sau corectat şi, totodată, având $\frac{\log_2 |W|}{l}$ cât mai mare pentru o eficienţă cât mai mare a codului.
- găsirea unor algoritmi eficienţi pentru codificare şi pentru detectarea erorilor (adică verificarea apartenenţei unui şir de l biţi la W) şi eventual corectarea erorilor (adică găsirea celui mai apropiat cuvânt din W faţă de un şir de l biţi dat).

2.4.3. Câteva coduri detectoare sau corectoare de erori

Descriem în continuare, pe scurt, câteva coduri detectoare sau corectoare de erori. În descrierea lor vom utiliza notațiile din paragraful precedent.

În general, mulțimea cuvintelor de cod W este astfel aleasă încât șirul primilor n dintre cei l biți să poată lua oricare dintre cele 2^n valori posibile, iar ultimii $l - n$ biți sunt unic determinați de primii n biți. Primii n biți din cuvântul de cod poartă denumirea de *informație utilă*, iar ultimii $l - n$ biți poartă numele de *biți de control*.

Pentru un astfel de cod, emițătorul primește de la sursă n biți ce constituie informația utilă, calculează cei $l - n$ biți de control aplicând un algoritm asupra informației utile și transmite prin canal informația utilă urmată de biții de control. Receptorul citește informația utilă și biții de control; pentru detectarea erorilor aplică același algoritm ca și emițătorul asupra informației utile citite și verifică dacă rezultatul coincide cu biții de control citați.

2.4.3.1. Bitul de paritate

La codul cu bit de paritate se alege $l = n + 1$. Există două sisteme, *paritate pară* (engl. *even parity*), în care W este definită ca fiind mulțimea șirurilor de l biți conținând număr par de valori 1, și *paritate impară* (engl. *odd parity*), în care W este mulțimea șirurilor de l biți conținând un număr impar de valori 1. Unicul bit de control se mai numește *bit de paritate*.

Se vede imediat că $d_{\min}(W) = 2$ și prin urmare bitul de paritate detectează o eroare și nu poate corecta nici o eroare.

Bitul de paritate se calculează numărând biții cu valoare 1 din informația utilă și verificând dacă este par sau impar.

EXEMPLUL 2.13: Pentru codul cu paritate pară și $n = 7$, șirul de biți 1010110 (informație utilă) se codifică 10101100 (bitul de control este 0). Șirul 1110110 se codifică 11101101 (bit de control 1). Șirul 11001100 este cuvântul de cod corespunzător informației utile 1100110. Șirul 11001101 nu este cuvânt de cod valid.

EXEMPLUL 2.14: Pentru codul cu paritate impară și $n = 7$, șirul de biți 1010110 se codifică 10101101 (bitul de control este 1). Șirul 1110110 se codifică 11101100 (bit de control 1). Șirul 11001100 nu este cuvânt de cod valid. Șirul 11001101 este cuvântul de cod corespunzător informației utile 1100110.

2.4.3.2. Paritate pe linii și coloane

La un astfel de cod informația utilă se consideră a fi o matrice $n_1 \times n_2$ de biți, cu n_1 și n_2 fixați. Ca urmare $n = n_1 \cdot n_2$. Codul are $l = (n_1 + 1) \cdot (n_2 +$

1). Cuvintele de cod sunt văzute ca fiind matrici $(n_1 + 1) \times (n_2 + 1)$ în care ultima linie și ultima coloană cuprind biții de control. Mulțimea cuvintelor de cod este mulțimea matricilor $(n_1 + 1) \times (n_2 + 1)$ în care pe fiecare linie și pe fiecare coloană numărul de valori 1 este par.

Se poate arăta ușor că $d_{\min}(W) = 4$, prin urmare codul detectează 3 erori sau corectează 1 eroare.

Codificarea și detectarea erorilor se face calculând bitul de paritate pentru fiecare linie și pentru fiecare coloană. De remarcat că ultimul bit din matrice trebuie calculat fie ca bit de paritate al biților de paritate ai liniilor, fie ca bit de paritate ai biților de paritate ai coloanelor; ambele variante duc la același rezultat.

EXEMPLUL 2.15: Pentru $n_1 = n_2 = 4$, șirul 1011010111001111 se codifică astfel:

$$\begin{array}{cccc|c} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 1 & 1 \end{array}$$

Astfel, cuvântul de cod rezultat este șirul: 1011101010110001111011011.

Pentru corectarea erorilor, se caută mai întâi liniile și coloanele care încalcă paritatea. Presupunând că s-a produs o singură eroare, va exista exact o linie și o coloană. Bitul eronat este la intersecția liniei și coloanei găsite.

EXEMPLUL 2.16: Șirul 101001101011010011000111111101 nu este cuvânt de cod:

$$\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 0 & 1 \end{array}$$

Se observă că paritatea nu este respectată de linia a 2-a și de prima coloană. Prin urmare, primul bit de pe linia a 2 este eronat, fiind 0 în original. Datele utile sunt deci: 1010010101100111.

2.4.3.3. Coduri polinomiale

Oricărui șir de biți $v = (v_1, \dots, v_k) \in \{0, 1\}^k$ i se asociază un polinom de grad cel mult $k - 1$:

$$v(X) = v_1X^{k-1} + v_2X^{k-2} + \dots + v_{k-1}X + v_k.$$

Coefficienții acestui polinom sunt considerați ca elemente ale corpului $F_2 = (\{0, 1\}, +, \cdot)$, unde $+$ este operația *sau exclusiv*, iar \cdot este operația *și*, cu tabelele de mai jos:

$$\begin{array}{c|cc} + & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array} \quad \begin{array}{c|cc} \cdot & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array}$$

De remarcat că polinoamele peste orice corp păstrează multe din proprietățile polinoamelor „obișnuite“, în particular se poate defini la fel adunarea, scăderea și înmulțirea și are loc teorema împărțirii cu rest.

Pentru construcția unui cod polinomial, se alege un așa-numit *polinom generator* $g(X)$ de grad $l - n$ (reamintim că l este lungimea cuvintelor de cod, iar n este numărul de biți ai informației utile; $n < l$). Mulțimea cuvintelor de cod W se definește ca mulțimea șirurilor de l biți cu proprietatea că polinomul asociat șirului este divizibil cu $g(X)$.

Șirul biților de control se calculează astfel:

- se construiește polinomul $i(X)$ asociat informației utile,
- se calculează $r(X)$ ca fiind restul împărțirii lui $i(X) \cdot X^{l-n}$ la $g(X)$
- șirul biților de control este șirul de $l - n$ biți al cărui polinom asociat este $r(X)$.

Pentru a ne convinge de corectitudinea algoritmului de mai sus, să observăm că obținem ca și cuvânt de cod un șir de forma $i_1, \dots, i_n, r_1, \dots, r_{l-n}$ al cărui polinom asociat este

$$\begin{aligned} v(X) &= i_1 X^{l-1} + \dots + i_n X^{l-n} + r_1 X^{l-n-1} + \dots + r_{l-n} = \\ &= i(X) \cdot X^{l-n} + r(X). \end{aligned}$$

Deoarece $r(X)$ este restul împărțirii lui $i(X) \cdot X^{l-n}$ la $g(X)$, rezultă că polinomul $i(X) \cdot X^{l-n} - r(X)$ este divizibil cu $g(X)$. Deoarece în F_2 avem că $1 + 1 = 0$ rezultă că $r(X) = -r(X)$. De aici rezultă că $v(X)$ este divizibil cu $g(X)$.

Codurile polinomiale sunt mult utilizate datorită simplității construcției unor circuite (hardware) care calculează biții de control.

Dacă se dorește corectarea erorilor, se observă că pozițiile erorilor nu depind decât de restul împărțirii polinomului asociat șirului de biți recepționat, $v'(X)$, la $g(X)$.

2.4.4. Coduri detectoare și corectoare de erori în alte domenii

Ne întâlnim cu coduri detectoare sau corectoare de erori și în situații mai puțin legate de calculatoare.

Limbaajul natural conţine multă redundanţă; ca urmare permite detectarea şi coerctarea multor „erori de tipar“, după cum vă puteţi convinge uoşr citind această frază. Din păcate însă, nu garantează detectarea nici măcar a unei singure erori; sunt cazuri în care o singură eroare poate schimba radical sensul unei fraze.

Transmisia vocii prin radio sau prin telefonie analogică este în general zgomotoasă şi adesea cu distorsiuni puternice. Ca urmare, riscul erorilor de transmisie este ridicat. Cum, pe de altă parte, diverse indicative cum ar fi numere de telefon, numere de înmatriculare, ş.a.m.d. nu conţin redundanţă, la transmiterea acestora cifrele se pronunţă cu anumite modificări, iar pentru litere se pronunţă un cuvânt întreg, dintr-un set standardizat, care începe cu litera ce se doreşte a fi transmisă. De exemplu, 2 minute se va pronunţa *doi minute*, pentru a evita confuzia *două-nouă*; de asemenea 7 se pronunţă *şapte*. Ca un alt exemplu, în engleză, indicativul ROT209 se va pronunţa *Romeo Oscar Tango Two Zero Niner*.

În sfârşit, codul numeric personal (CNP), codul IBAN, ISBN-ul de pe cărţi şi alte asemenea coduri de identificare ce sunt transmise frecvent prin intermediul unor operatori umani au o cifră de control.

Capitolul 3

Nivelul fizic

3.1. Problema transmisiei informației la nivelul fizic

Sarcina nivelului fizic este aceea de-a transmite un șir de biți (sau, în general, un șir de simboluri) produs de o *sursă* către o *destinație*. Sursa și destinația se află la distanță una față de cealaltă.

Sursa și destinația sunt „clienții” sistemului de comunicație; nivelul fizic trebuie să fie capabil să transmită datele în folosul acestora.

Șirul de biți ce trebuie transmis poartă denumirea de *date utile*.

Pentru îndeplinirea scopului său, nivelul fizic dispune de un *mediu de transmisie*. Mediul de transmisie se întinde de la amplasamentul sursei până la amplasamentul destinației și este capabil să transmită la distanță o anumită acțiune fizică.

Nivelul fizic cuprinde trei elemente: mediul de transmisie, *emițătorul* și *receptorul* (vezi fig. 3.1). *Emițătorul* primește biții de la sursă și, în conformitate cu valorile lor, acționează asupra mediului. *Receptorul* sesizează acțiunile emițătorului asupra mediului și reconstituie șirul de biți produs de sursă. Șirul de biți reconstituit este livrat destinației.

Mărimea fizică ce măsoară acțiunea produsă de emițător și transmisă de către mediu până la receptor și care este utilizată efectiv ca purtătoare a informației se numește *semnal*. Semnalul este întotdeauna analizat ca o funcție continuă de timp.

Mărimea fizică utilizată ca semnal este aleasă de proiectantul sistemului de comunicații dintre acele mărimi pe care mediul ales le poate propaga în condiții bune. De exemplu, pentru transmisia prin perechi de conductoare, semnalul poate fi tensiunea electrică dintre conductoare sau intensitatea curentului prin conductoare.

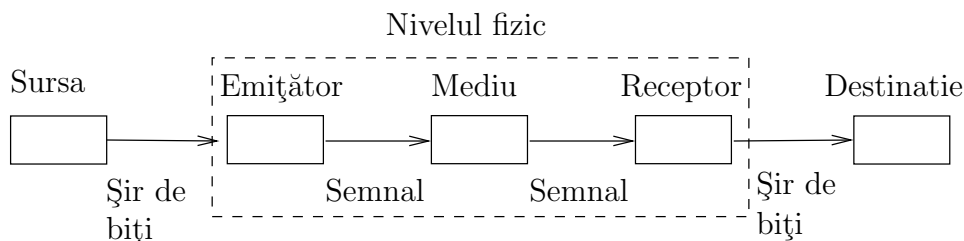


Figura 3.1: Modelarea transmisiei la nivel fizic

Emitătorul transformă șirul de biți recepționat într-un semnal adecvat transmiterii prin mediul de comunicație. Receptorul efectuează operația inversă. Corespondența dintre șirurile de biți posibile și semnalele corespunzătoare poartă denumirea de *schemă de codificare a informației prin semnal continuu*.

Schema de codificare utilizată trebuie să fie aceeași pentru emițător și receptor.

Mediul de transmisie modifică în general semnalul transmis, astfel că semnalul primit de receptor de la mediu nu este identic cu semnalul aplicat de emițător asupra mediului. Vom arăta în § 3.2 care sunt transformările suferite de semnal în timpul propagării. Schema de codificare a informației trebuie să țină cont de aceste modificări. O parte din schemele folosite vor fi studiate în § 3.3.

În continuarea acestui capitol vom trece în revistă problemele specifice legate de transmiterea semnalelor și de codificarea informației prin semnale. O analiză riguroasă a acestor probleme depășește cu mult cadrul acestei lucrări. Prezentarea de față are ca scop familiarizarea cu noțiunile și problemele respective, în vederea înțelegerii soluțiilor existente, limitărilor lor, parametrilor specificați în documentațiile privind echipamentele folosite și, mai ales, posibilității comunicării cu specialiștii în domeniul electronicii și comunicațiilor.

3.2. Transmiterea semnalelor

3.2.1. Modificările suferite de semnale

Pentru a studia modificările suferite de semnale în timpul propagării prin mediul de transmisie, vom considera în principal cazul transmiterii tensiunii electrice printr-o pereche de conductoare.

Semnalul măsurat la joncțiunea dintre emițător și mediu se numește *semnal emis* și îl vom nota cu $U_e(t)$, unde t este timpul. Semnalul măsurat

la joncțiunea dintre mediu și receptor se numește *semnal recepționat* și îl vom nota cu $U_r(t)$.

Transformările suferite de semnal sunt următoarele:

Întârzierea constă în faptul că semnalul recepționat urmează cu o anumită întârziere semnalul emis. Cu notațiile de mai sus și neglijând fenomenele ce vor fi descrise la punctele următoare, avem $U_r(t) = U_e(t - \Delta t)$. Durata Δt se numește *întârziere (de propagare)* sau *timp de propagare*. Întârzierea are valoarea $\Delta t = \frac{l}{v}$, unde l este lungimea mediului iar v este viteza de propagare a semnalului. Viteza de propagare a semnalului depinde de natura mediului de transmisie. La transmisia prin conductoare, v depinde numai de materialul izolator dintre conductoare și, pentru materialele folosite în mod curent, are valoarea aproximativă $v \approx 2/3c = 2 \cdot 10^8$ m/s, unde c este viteza luminii în vid.

atenuarea constă în faptul că semnalul recepționat are amplitudine mai mică decât cel emis. Neglijând întârzierea, are loc $U_r(t) = g \cdot U_e(t)$, cu $0 < g < 1$. Ținând cont și de întârziere, avem $U_r(t) = g \cdot U_e(t - \Delta t)$. Numărul $1/g$ se numește *factor de atenuare în tensiune*.

De cele mai multe ori atenuarea unui semnal este exprimată prin *factorul de atenuare în putere*, numit pe scurt *factor de atenuare*, definit ca raportul dintre puterea semnalului emis și a celui recepționat. În cazul perechii de conductoare, deoarece puterea este proporțională cu pătratul tensiunii (raportul tensiune/intensitate fiind aproximativ constant), factorul de atenuare în putere este egal cu $1/g^2$.

Prin conectarea unul după celălalt a mai multor medii de transmisie, factorul de atenuare a mediului rezultat este produsul factorilor de atenuare ai componentelor. Din acest motiv, în loc de factorul de atenuare se folosește adesea logaritmul său: logaritmul factorului de atenuare rezultat este suma logaritmilor, în aceeași bază, ai factorilor de atenuare ai componentelor.

Logaritmul factorului de atenuare se numește pe scurt *atenuare*.

Valoarea logaritmului depinde de baza utilizată, baze diferite ducând la valori proporționale. Deoarece schimbarea bazei de logaritmare are un efect similar cu schimbarea unității de măsură pentru o mărime fizică, după valoarea logaritmului se scrie o pseudo-unitate de măsură ce arată de fapt baza de logaritmare utilizată. Pentru logaritmul în baza zece, pseudo-unitatea de măsură folosită este *belul*, având simbolul B. Pseudo-unitatea de măsură utilizată curent este *decibelul*, având simbolul dB. Avem $1 \text{ B} = 10 \text{ dB}$. O valoare exprimată în decibeli (dB) o putem vedea, echivalent, fie ca valoarea logaritmului în baza

10 înmulțită cu 10, fie ca valoarea logaritmului în baza $10^{1/10}$. De exemplu, dacă factorul de atenuare este $(1/g^2) = 10$, logaritmul său este $1 \text{ B} = 10 \text{ dB}$. Dacă factorul de atenuare este 2, logaritmul său (atenuarea) este $\log_{10} 2 \text{ B} \approx 0,3 \text{ B} = 3 \text{ dB}$.

Puterea semnalului emis se măsoară în watti (W) sau miliwatti (mW). Adesea, este specificată nu puterea ci logaritmul puterii: se ia numărul ce reprezintă puterea, în miliwatti, și logaritmul său se exprimă în decibeli. Pseudo-unitatea de măsură corespunzătoare reprezentării de mai sus se numește decibel-miliwatt, având simbolul (neconform regulilor Sistemului Internațional de Masuri și Unități) dBm. Ca exemple: o putere de emisie de 1 mW poate fi scrisă și 0 dBm, o putere de 1 W se scrie 30 dBm, iar 0,1 mW se scrie ca -10 dBm .

Puterea minimă a semnalului recepționat, pentru care receptorul este capabil să decodifice corect semnalul, se numește *pragul de sensibilitate* al receptorului. Ca și puterea emițătorului, pragul de sensibilitate se poate exprima în miliwatti sau în decibel-miliwatti.

distorsiunea este o modificare deterministă a semnalului recepționat față de cel emis, diferită de întârziere și atenuare. (O modificare este deterministă dacă, oricâteori transmitem un același semnal, modificarea se manifestă identic.) Mai multe detalii despre distorsiuni vor fi date în § 3.2.2.

zgomotele sunt modificări nedeterminate ale semnalului recepționat, cauzate de factori externi sistemului de transmisie (fulgere, întrerupătoare electrice, alte sisteme de transmisie de date, alte echipamente electronice) sau de factori interni cu manifestare aleatoare (mișcarea de agitație termică a atomilor din dispozitivele electornice).

Zgomotul se exprimă ca diferența dintre semnalul recepționat efectiv și semnalul ce ar fi recepționat în lipsa zgomotului. *Raportul semnal/zgomot* este raportul dintre puterea semnalului și puterea corespunzătoare zgomotului. Uneori termenul de raport semnal/zgomot este utilizat și pentru logaritmul raportului semnal/zgomot; de obicei nu este pericol de confuzie deoarece logaritmul este exprimat în decibeli, în timp ce raportul semnal/zgomot nu are unitate de măsură.

O categorie specială de zgomot este *diafonia*, care este un zgomot provenit din semnalul transmis pe un mediu de transmisie vecin.

3.2.2. Analiza transmiterii semnalelor cu ajutorul transformatei Fourier

Considerăm un dispozitiv electronic, care are o intrare și o ieșire

(fig. 3.2). În particular, o pereche de conductoare folosită pentru transmisie poate fi considerată un astfel de dispozitiv, capetele dinspre emițător constituind intrarea, iar cele dinspre receptor, ieșirea.



Figura 3.2: Un dispozitiv cu o intrare și o ieșire

Tensiunea de la ieșire depinde de tensiunea de la intrare, însă în general depinde de tot istoricul ei. Altfel spus, comportamentul dispozitivului poate fi descris de un operator L (reamintim că un operator este o funcție definită pe un spațiu de funcții cu valori tot într-un spațiu de funcții). Acest operator primește ca argument funcția timp-tensiune U_i care caracterizează semnalul de intrare. Valoarea operatorului este funcția timp-tensiune $U_e = L(U_i)$ care caracterizează semnalul de ieșire.

Multe dispozitive electronice au un comportament *liniar*, adică operatorul L care le caracterizează este un operator liniar. Reamintim că un operator L este liniar dacă, pentru orice funcții f și g și pentru orice scalari α și β , are loc

$$L(\alpha f + \beta g) = \alpha L(f) + \beta L(g).$$

Pentru un dispozitiv liniar, dacă semnalul de intrare $U_i(t)$ poate fi descompus ca o sumă de forma

$$U_i(t) = \alpha_1 U_{i1}(t) + \alpha_2 U_{i2}(t) + \dots + \alpha_n U_{in}(t),$$

atunci pentru semnalul de ieșire avem

$$U_e(t) = L(U_i)(t) = \alpha_1 U_{e1}(t) + \alpha_2 U_{e2}(t) + \dots + \alpha_n U_{en}(t),$$

unde $U_{e1} = L(U_{i1})$, $U_{e2} = L(U_{i2})$, ..., $U_{en} = L(U_{in})$.

Dispozitivele liniare au proprietatea că, dacă semnalul de intrare este *sinusoidal*, adică

$$U_i(t) = U_0 \cdot \cos(2\pi ft + \phi),$$

atunci semnalul de ieșire este tot sinusoidal și, mai mult,

$$U_e(t) = g(f) \cdot U_0 \cdot \cos(2\pi ft + \phi - \theta(f)),$$

unde $g(f)$ și $\theta(f)$ depind doar de cum este construit dispozitivul și de frecvența f a semnalului.

Orice semnal se poate scrie unic ca o sumă de semnale sinusoidale. (Nota: condițiile matematice asupra semnalului, și alte detalii se găsesc în lucrările de specialitate, de exemplu [Crstici et al. 1981]; aici facem doar o prezentare semi-intuitivă).

Un semnal *periodic* de perioadă T se poate descompune în așa-numita *serie Fourier*:

$$U(t) = \sum_{k=0}^{\infty} a_k \cos\left(\frac{2\pi k}{T}t + \phi_k\right).$$

Un semnal *limitat în timp*, adică nul în afara unui interval finit $[0, T]$, se poate descompune sub forma:

$$U(t) = \int_0^{\infty} a(f) \cdot \cos(2\pi ft + \phi(f)) df. \quad (3.1)$$

Notă: relația (3.1) este dată de obicei sub forma numită *transformata Fourier inversă*:

$$U(t) = \int_{\mathbb{R}} \hat{U}(f) \cdot e^{2\pi i f t} df, \quad (3.2)$$

unde \hat{U} este o funcție complexă care se numește *transformata Fourier* a funcției U .

Relația (3.1) spune că semnalul U se poate scrie ca o sumă de sinusoidale cu diferite frecvențe f având amplitudinile $a(f)$ și defazaajul (decalajul sinusoidalei de-a lungul axei Ox) egal cu $\phi(f)$.

Frecvențele f pentru care amplitudinile corespunzătoare $a(f)$ sunt nenule alcătuiesc *spectrul* semnalului.

Pentru un dispozitiv liniar, semnalul de ieșire se poate calcula descompunând în sinusoidale semnalul de intrare, calculând efectul dispozitivului asupra fiecărei sinusoidale în parte și însumând în final ieșirile:

$$\begin{aligned}
U_e(t) &= L(U_i(t)) = \\
&= L\left(\int_0^{\infty} a_i(f) \cdot \cos(2\pi ft + \phi_i(f))df\right) = \\
&= \int_0^{\infty} L(a_i(f) \cdot \cos(2\pi ft + \phi_i(f)))df = \\
&= \int_0^{\infty} a_i(f) \cdot g(f) \cdot \cos(2\pi ft + \phi_i(f) - \theta(f))df,
\end{aligned} \tag{3.3}$$

unde $a_i(f)$ și $\phi_i(f)$ sunt funcțiile $a(f)$ și $\phi(f)$ din descompunerea, conform relației (3.1), a semnalului de intrare U_i .

Comportamentul unui dispozitiv liniar este deci complet definit de funcțiile $g(f)$ și $\theta(f)$.

Un semnal este nedistorsionat dacă și numai dacă, pentru toate frecvențele f din spectrul semnalului, $g(f)$ este constantă și $\theta(f)$ este proporțional cu f , adică există constantele g_0 și θ_0 astfel încât

$$\begin{cases} g(f) = g_0 \\ \theta(f) = \theta_0 f \end{cases} \tag{3.4}$$

pentru toate frecvențele f din spectrul semnalului.

În practică, condiția (3.4) este satisfăcută, cu o aproximație acceptabilă, doar pentru frecvențe care se încadrează într-un anumit interval $f \in [f_{\min}, f_{\max}]$. Acest interval se numește *banda de trecere* a dispozitivului. În consecință, dacă spectrul semnalului de intrare se încadrează în banda de trecere a dispozitivului, semnalul de ieșire va prezenta distorsiuni acceptabil de mici.

Diferența $f_{\max} - f_{\min}$ se numește *lățimea de bandă* a dispozitivului.

De exemplu, banda de trecere a unei linii telefonice este cuprinsă între aproximativ 300 Hz și 3 kHz.

3.3. Codificarea informației prin semnale continue

3.3.1. Scheme de codificare

Cea mai simplă codificare este aceea în care împărțim timpul în intervale de durată fixată Δt (pe care o numim *lungimea unui bit*) și, pe durata

fiecărui bit, semnalul emis va avea o anumită valoare — de exemplu 12 V — dacă bitul are valoarea 1 și o altă valoare — de exemplu 0 V — dacă bitul are valoarea 0 (vezi fig. 3.3).

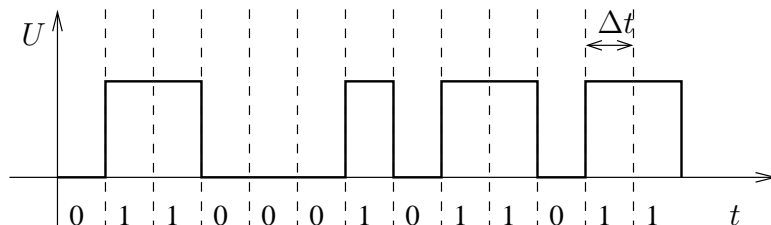


Figura 3.3: Codificarea directă

Receptorul determină intervalele corespunzătoare biților și măsoară semnalul la mijlocul fiecărui interval. Dacă tensiunea este mai mare decât o valoare numită *prag* — pentru exemplul nostru se poate lua ca prag 3 V — receptorul decide că bitul respectiv are valoarea 1, iar în caz contrar decide că bitul are valoarea 0.

Valoarea pragului poate fi fixă sau poate fi stabilită dinamic în funcție de amplitudinea semnalului recepționat pentru a ține cont de atenuare.

Să observăm că receptorul trebuie să fie sincronizat cu emițătorul, adică să examineze semnalul recepționat la mijlocul intervalului corespunzător unui bit. Acest lucru se poate face — însă este adesea nepractic — transmițând un al doilea semnal, de sincronizare, pe un mediu separat (adică folosind o altă pereche de fire).

Sincronizarea se poate face și pe baza semnalului util, dacă receptorul dispune de un ceas suficient de precis. În acest scop, receptorul va măsura timpul cât semnalul este „sus“ (peste prag) și va determina de câte ori se cuprinde în acest interval durata unui bit. Numărul de biți consecutivi identici trebuie să fie limitat, căci receptorul nu va putea distinge între n biți și $n + 1$ biți consecutivi având aceeași valoare, dacă n este prea mare.

Limitarea numărului de biți identici consecutivi se poate face în mai multe feluri:

Codificarea Manchester. Semnalul are una sau două tranziții pentru fiecare interval corespunzător unui bit. O tranziție la mijlocul intervalului arată valoarea bitului: tranziția este în sus pentru 1 și în jos pentru 0. Pentru a face posibil ca doi biți consecutivi să aibă aceeași valoare, la începutul intervalului corespunzător unui bit mai poate să apară o tranziție (fig. 3.4).

Rețelele Ethernet de 10 Mbit/s utilizează codificarea Manchester.

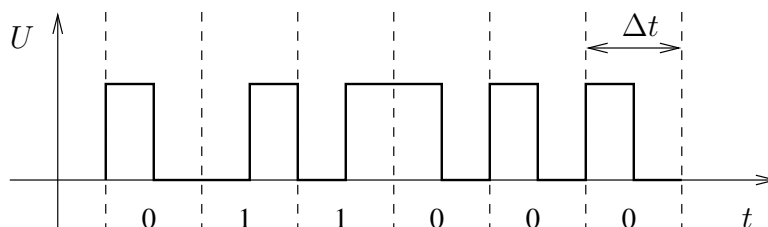


Figura 3.4: Codificarea Manchester

Codificarea Manchester diferențială. Semnalul are o tranziție la începutul fiecărui interval de bit. Dacă bitul este 1 atunci semnalul mai are o tranziție la mijlocul intervalului (fig. 3.5).

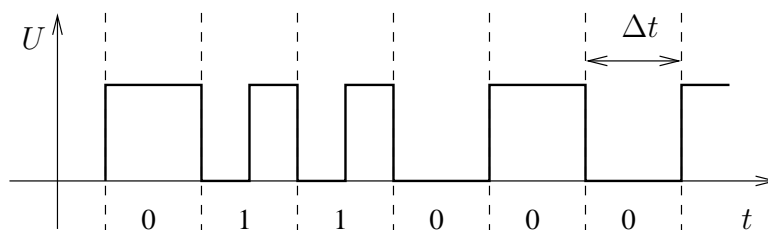


Figura 3.5: Codificarea Manchester diferențială

Codurile de grup sunt o familie de coduri construite după următoarea schemă:

Se fixează un număr n (valori uzuale: $n = 4$ sau $n = 8$); șirul transmis trebuie să aibă ca lungime un multiplu de n biți.

Se fixează o tabelă de corespondență care asociază fiecăruia dintre cele 2^n șiruri de n biți posibile un șir de m biți, unde $m > n$ este fixat, cu restricția ca între cei m biți să nu fie prea multe valori egale consecutive. Codul este determinat de numerele n și m și de această tabelă.

Șirul de biți de codificat se codifică astfel: mai întâi, fiecare grup de n biți consecutivi se înlocuiește cu șirul de m biți asociat. Apoi șirul de biți astfel obținut se codifică direct, un bit 0 fiind reprezentat printr-o valoare a tensiunii și un bit 1 prin altă valoare.

Rețelele Ethernet de 100 Mbit/s utilizează un cod de grup cu $n = 4$ și $m = 5$.

Să examinăm acum cerințele privind banda de trecere a mediului necesară pentru transmiterea semnalelor de mai sus.

Semnalele de formă rectangulară descrise mai sus au spectru infinit (spectrul lor nu este mărginit superior). Trecute printr-un mediu de comunicație care are o lățime de bandă finită, semnalele vor fi „rotunjite“ mai mult sau mai puțin.

Să notăm cu τ durata elementară a unui palier al semnalului ideal (durata minimă în care semnalul ideal are o valoare constantă). Pentru codificarea directă, $\tau = \Delta t$; pentru codificarile Manchester și Manchester diferențială, $\tau = \frac{1}{2}\Delta t$.

Dacă banda de trecere a mediului include intervalul $[0, \frac{1}{2\tau}]$, atunci mediul păstrează suficient din forma semnalului pentru ca receptorul să poată decodifica informația transmisă. Dacă frecvența maximă a benzii de trecere este mai mică decât $\frac{1}{2\tau}$, atunci un semnal rectangular care are, alternativ, un timp τ o valoare și următorul timp τ cealaltă valoare va fi distorsionat atât de mult încât „urcușurile“ și „coborâșurile“ semnalului nu vor mai putea fi identificate de către receptor și ca urmare informația purtată nu mai poate fi obținută.

Pentru un mediu dat, cu o bandă de trecere dată, există, prin urmare, o valoare minimă a lui τ pentru care receptorul poate extrage informația utilă din semnalul recepționat. Dacă limita superioară a benzii de trecere este f_{\max} , valoarea minimă este $\tau = \frac{1}{2f_{\max}}$.

Diversele codificări studiate mai sus au diferite rapoarte k între durata medie a unui bit și valoarea lui τ . La codificarea directă, durata unui bit este egală cu τ și deci $k = 1$. La codificarile Manchester și Manchester diferențială, durata unui bit este 2τ și avem $k = 2$. La codurile de grup, durata unui bit util este $\frac{m}{n}\tau$ și avem $k = \frac{m}{n}$. Debitul maxim cu care se pot transmite datele este $\frac{f_{\max}}{k}$.

3.3.2. Modulația

Există situații în care este necesar ca spectrul semnalului să ocupe o bandă departe de frecvența zero. Aceasta se poate întâmpla fie pentru că circuitele sau mediul de transmisie nu pot transmite frecvențele apropiate de zero (este de exemplu cazul transmiterii prin unde radio), fie pentru a putea transmite mai multe semnale pe același mediu prin multiplexare în frecvență (vezi § 3.3.3).

În aceste situații, semnalul rezultat direct în urma uneia dintre schemele de codificare descrise în paragraful precedent nu poate fi transmis direct. O posibilă soluție este *modulația*, descrisă în continuare.

Semnalul transmis efectiv este de forma:

$$U(t) = a \cdot \sin(2\pi ft + \phi),$$

unde unul dintre parametri a , f sau ϕ variază în timp, în funcție de semnalul original, rezultat direct din codificare.

Semnalul original îl numim *semnal primar* sau *semnal modulator*.

Semnalul sinusoidal, rezultat pentru valorile „de repaus” ale parametrilor a , f și ϕ , se numește *semnal purtător*, iar frecvența f de repaus se numește *frecvență purtătoare* și o vom nota în continuare cu f_p .

Semnalul rezultat în urma modulației se numește *semnal modulat*.

Operația de construcție a semnalului modulat pornind de la semnalul primar se numește *modulație*. Operația inversă, de obținere a semnalului primar dându-se semnalul modulat, se numește *demodulație*.

După parametrul modificat, avem:

modulația de amplitudine (prescurtat *MA*, engl. *amplitude modulation*, *AM*), care constă în modificarea amplitudinii a . Semnalul transmis este deci:

$$U(t) = U_0 \cdot s(t) \cdot \sin(2\pi f_p t),$$

unde $s(t)$ este semnalul modulator. Pentru ca amplitudinea $a = U_0 \cdot s(t)$ să fie mai mare decât 0, asupra semnalului $s(t)$ se impune restricția $s(t) > 0$.

Se observă că modulația în amplitudine este liniară (modulația sumei a două semnale $a + b$ este suma rezultatelor modulației independente pentru a și b).

Dacă semnalul modulator este sinusoidal

$$s(t) = 1 + m \cdot \sin(2\pi f_s t + \phi)$$

atunci

$$\begin{aligned} U(t) &= U_0 \cdot s(t) \cdot \sin(2\pi f_p t) = \\ &= U_0 \cdot (\sin(2\pi f_p t) + m \cdot \sin(2\pi f_s t + \phi) \cdot \sin(2\pi f_p t)) = \\ &= U_0 \cdot \left(\sin(2\pi f_p t) + \right. \\ &\quad \left. + \frac{m}{2} \cos(2\pi(f_p - f_s)t - \phi) - \frac{m}{2} \cos(2\pi(f_p + f_s)t + \phi) \right) \end{aligned} \quad (3.5)$$

adică în urma modulației în amplitudine cu un semnal sinusoidal de frecvență f_s se obține o sumă de trei semnale sinusoidale având frecvențele $f_p - f_s$, f_p și $f_p + f_s$.

Din liniaritatea modulației în amplitudine și din relația (3.5) deducem că, pentru un semnal modulator având un anumit spectru, spectrul semnalului modulat conține frecvența purtătoare și două *benzi laterale*, stângă și dreaptă, acestea cuprinzând diferențele, respectiv sumele, dintre frecvența purtătoare și frecvențele din spectrul semnalului primar.

Întrucât spectrul semnalului modulat este simetric în jurul frecvenței purtătoare, de fapt doar una dintre benzile laterale poartă informație utilă. Din acest motiv, adesea se suprimă total sau parțial de la transmisie una dintre benzile laterale.

modulația de frecvență (prescurtat *MF*, engl. *frequency modulation*, *FM*), care constă în modificarea frecvenței f în jurul frecvenței purtătoare f_p .

Semnalul transmis are forma

$$U(t) = U_0 \cdot \sin(2\pi \cdot (f_p + m \cdot s(t)) \cdot t)$$

unde, din nou, f_p este *frecvența purtătoare*, $s(t)$ este semnalul modulator, iar m este o constantă. Semnalul modulator trebuie să respecte restricția $m \cdot s(t) s_0 \ll f_p$.

Analiza spectrului unui semnal modulat în frecvență este mult mai dificilă decât în cazul modulației în amplitudine.

modulația de fază, care constă în modificarea fazei ϕ .

Semnalul transmis are forma

$$U(t) = U_0 \cdot \sin(2\pi f_p t + m \cdot s(t))$$

Este evident că, întrucât receptorul nu are de obicei un reper absolut de timp, el nu poate detecta decât variațiile de fază ale semnalului recepționat. Ca urmare, o valoare constantă a lui $s(t)$ nu poate fi deosebită de zero și, mai mult, nici variații lente ale lui $s(t)$ nu pot fi detectate. În consecință, spectrul lui $s(t)$ nu poate conține frecvențe prea apropiate de 0.

Există și posibilitatea de-a varia simultan doi sau chiar toți cei trei parametri. *Modulația în cuadratură* constă în varierea simultană a amplitudinii a și a fazei ϕ , pentru a transmite simultan două semnale utile s_1 și s_2 .

Semnalul modulat are forma

$$\begin{aligned} U(t) &= U_0 \cdot \sqrt{s_1(t)^2 + s_2(t)^2} \cdot \sin\left(2\pi f_p t + \arctg \frac{s_1(t)}{s_2(t)}\right) \\ &= U_0 \cdot ((s_1(t)) \cos(2\pi f_p t) + (s_2(t)) \sin(2\pi f_p t)) \end{aligned}$$

3.3.3. Multiplexarea în frecvenţă

Multiplexarea, în general, constă în transmiterea mai multor semnale independente prin acelaşi mediu de transmisie.

Două semnale ale căror spectre se încadrează în benzi disjuncte pot fi separate cu ajutorul unor dispozitive numite *filtre* (de frecvenţă).

Multiplexarea în frecvenţă constă în transmiterea simultană prin acelaşi mediu a unor semnale având spectre încadrate în benzi disjuncte.

Emitătoarele produc semnale cu spectre disjuncte prin modulaţie utilizând frecvenţe purtătoare diferite. De notat că diferenţele între frecvenţele purtătoare trebuie să fie mai mari decât lăţimile de bandă necesare transmisiei semnalelor corespunzătoare.

Fiecare receptor trebuie să fie dotat cu un filtru care să lase să treacă doar banda utilizată de emiţătorul corespunzător.

3.3.4. Capacitatea maximă a unui canal de comunicaţie

Banda de trecere a mediului de transmisie împreună cu raportul semnal/zgomot determină o limită superioară a debitului transmisiei. Limitarea este independentă de schema de codificare utilizată pentru transmisie şi ca urmare este valabilă pentru orice schemă de codificare ne-am putea imagina.

Este util să avem în vedere existenţa acestei limite, în acelaşi fel în care cunoaşterea principiului conservării energiei ne foloseşte pentru a nu încerca construcţia unui perpetuum mobile — încercare din start sortită eşecului.

Pentru un mediu cu lăţimea de bandă Δf şi cu raportul semnal/zgomot s/n , debitul maxim de informaţie ce poate fi transmis este proporţional cu $\Delta f \cdot \log(s/n + 1)$. Acest rezultat provine din următoarele două observaţii:

1. Teorema de eşantionare a lui Shannon spune că un semnal al cărui spectru se încadrează într-un interval $[0, f_{\max})$ este unic determinat de valorile sale la momente de timp situate la intervale egale cu $\Delta t = \frac{1}{2f_{\max}}$ unul de altul.

Ca urmare, un semnal al cărui spectru este inclus în intervalul $[0, f_{\max})$ nu poate purta mai multă informaţie decât eşantioanele semnalului luate la interval $\frac{1}{2f_{\max}}$ unul de altul.

2. În prezența zgomotului, receptorul nu poate distinge între două valori posibile ale semnalului la un anumit moment de timp decât dacă diferența dintre cele două valori este mai mare decât amplitudinea zgomotului.

Ca urmare, cantitatea de informație purtată de un eșantion este limitată la o valoare proporțională cu $\log(s/n + 1)$.

Deoarece pentru o schemă de codificare fixată există o relație de proporționalitate între lățimea de bandă a mediului și debitul maxim al transmisiei, debitul maxim al transmisiei unui echipament de comunicație se numește uneori în mod impropiu tot *lățime de bandă* sau *bandă de trecere*.

3.4. Transmisia prin perechi de conductoare

La transmisia prin perechi de conductoare, mediul constă din două conductoare izolate între ele. Semnalul este considerat a fi tensiunea electrică între conductoare.

3.4.1. Construcția cablului

Conductoarele trebuie realizate dintr-un material cu conductivitate electrică ridicată. Aproape în toate cazurile materialul folosit este cuprul.

Izolația dintre conductoare trebuie să nu absoarbă multă energie dacă este plasată într-un câmp electric variabil. În acest scop doar anumite substanțe sunt potrivite. Materialele utilizate cel mai frecvent sunt polietilena și politetrafluoretilena (cunoscută sub numele de *Teflon*TM). Policlorura de vinil (PVC), utilizată adesea la izolarea conductoarelor de alimentare cu energie electrică, absoarbe prea mult din puterea unui semnal de frecvență mare; din această cauză nu se poate folosi în circuite de semnal. Aerul este cel mai bun izolator, dar nu oferă susținere mecanică.

Ca formă și dispunere relativă, există trei construcții utilizate:

- *Perechea simplă*, în care conductoarele sunt paralele unul față de celălalt. Conductoarele pot fi alcătuite dintr-o singură sârmă de cupru, sau — pentru a fi mai flexibile — dintr-un mănunchi de sârme subțiri. Fiecare conductor este învelit într-un strat izolator.

Adesea, mai multe perechi de conductoare sunt duse împreună, în paralel, formând un cablu. În cadrul unui cablu, este posibil ca un conductor să fie comun, partajat între două sau mai multe circuite de semnalizare. În acest caz, n circuite utilizează $n + 1$ conductoare, în loc de $2n$ câte sunt în cazul în care perechile sunt complet separate.

Avantajul este, evident, reducerea costului, iar dezavantajul este mărirea diafoniei între circuite.

Din cauza diafoniei și sensibilității la zgomote, perechea simplă se utilizează doar pe distanțe mici.

- *Perechea torsadată* (engl. *twisted pair*), în care conductoarele sunt răsucite unul în jurul celuilalt. Rolul răsucirii este de-a micșora interacțiunea cu câmpul electromagnetic înconjurător, adică micșorarea zgomotului indus de un câmp electromagnetic înconjurător și, totodată, micșorarea câmpului electromagnetic produs de semnalul ce trece prin perechea de conductoare. Acest lucru este important în special pentru micșorarea diafoniei cu celelalte perechi de conductoare din același cablu. În afară de răsucire, restul construcției este identică cu perechea simplă. Cablurile formate din perechi torsadate nu au niciodată un conductor comun pentru mai multe circuite.

Este important ca, în cazul unui cablu ce conține mai multe perechi torsadate, fiecare circuit de comunicație să utilizeze conductoarele din aceeași pereche și nu un conductor dintr-o pereche și un conductor din altă pereche. În caz contrar, apare diafonie foarte puternică între circuite (mai mare decât la perechea simplă). De remarcat că această greșeală este ușor de comis în urma unei identificări greșite a conductoarelor dintr-un cablu și nu este pusă în evidență de dispozitivele simple de testare, care verifică doar continuitatea în curent continuu a conductoarelor cablului.

- *Perechea coaxială* are unul din conductoare în forma unui cilindru gol în interior, iar celălalt conductor este dus prin interiorul primului conductor și izolat electric față de acesta. Conductorul exterior este format de obicei dintr-o plasă formată din sârme subțiri de cupru, înfășurate elicoidal, o parte din fire fiind înfășurate într-un sens și altă parte în celălalt sens.

Cablul coaxial este și mai protejat de interferențe decât perechea torsadată. Are de obicei atenuare mai mică decât perechea simplă sau cea torsadată. Costul este însă mai ridicat.

În oricare dintre variante, pentru a reduce suplimentar interferențele cu câmpul electromagnetic înconjurător, perechea de conductoare poate fi *ecranată*, adică învelită într-un strat conductor continuu. Pentru ca ecranul să fie eficient, trebuie să aibă continuitate de jur împrejurul conductoarelor (dacă este realizat prin înfășurarea unei foițe metalice, marginile foiței trebuie să facă contact ferm între ele) și pe lungime (să aibă legătură prin conectoare

către elemente de ecranare ale echipamentelor la care este conectat cablul).

Ecranul unui cablu poate fi colectiv, îmbrăcând întreg cablul, sau individual pentru fiecare pereche de conductoare.

Pe lângă elementele cu rol electric, un cablu conține elemente cu rol de protecție. Orice cablu este înfășurat cel puțin într-o manta de protecție, care ține la un loc și protejează mecanic conductoarele. Mantaua de protecție este fabricată de obicei din PVC.

Un cablu destinat montării aeriene trebuie să fie prevăzut cu un cablu de oțel pentru susținere mecanică. Un cablu destinat montării subterane trebuie prevăzut cu un scut metalic contra rozătoarelor.

3.4.2. Proprietăți ale mediului

În cele ce urmează vom presupune că lungimea cablului este fie de același ordin de mărime fie mai mare decât raportul dintre viteza luminii în vid și frecvența maximă din spectrul semnalului. În aceste condiții, perechea de conductoare are comportament de *linie lungă*, adică semnalul se propagă din aproape, sub forma unei unde, de-a lungul perechii de conductoare.

Proprietățile electrice mai importante ale mediului sunt:

Viteza de propagare a semnalului prin mediu. Este identică cu viteza de propagare a undelor electromagnetice în materialul dielectric dintre conductoare. Se specifică de obicei prin raportare la viteza luminii în vid (notată c , $c \approx 3 \times 10^8$ m/s). În mod tipic $v \approx 0,67 \times c \approx 2 \times 10^8$ m/s

Banda de trecere a mediului. Se întinde de la zero până la o frecvență maximă de ordinul a câteva sute de megahertzi sau câțiva gigahertzi. Limitările sunt date de două fenomene independente, pierderile în dielectric (la frecvențe mari dielectricul absoarbe o parte din energia câmpului electric dintre conductoare) și efectul pelicular (la frecvențe mari curenții electrici din conductoare nu circulă uniform în toată masa acestora ci doar în vecinătatea suprafeței). Îmbătrânirea izolației cablului duce la micșorarea frecvenței maxime a benzii de trecere.

Atenuarea semnalului. Factorul de atenuare crește exponențial cu lungimea mediului. În consecință, logaritmul factorului de atenuare crește liniar cu lungimea mediului. Ca urmare, pentru un tip de cablu se specifică raportul dintre logaritmul factorului de atenuare și lungimea corespunzătoare, în decibeli pe kilometru. Cu titlu de exemplu, dăm câteva valori tipice: 17 dB/km pentru cablu coaxial „Ethernet gros“; 120 dB/km pentru cablu torsadat Ethernet.

Impedanța caracteristică a mediului. Să presupunem că atașăm la un capăt al unei bucăți infinite de cablu o sursă de tensiune alternativă. Se

observă că intensitatea curentului ce trece prin sursă și prin capătul dinspre sursă al cablului este proporțională cu tensiunea. Raportul dintre tensiune și intensitate se numește *impedanța caracteristică* a cablului.

Receptorul se caracterizează și el printr-o *impedanță de intrare*, definită ca raportul dintre tensiunea aplicată la bornele receptorului și intensitatea curentului absorbit de receptor. Emițătorul se caracterizează printr-o *impedanță de ieșire*, definită ca raportul dintre scăderea tensiunii la borne cauzată de absorbția unui curent de către un dispozitiv montat la bornele emițătorului și intensitatea curentului absorbit.

Dacă la un capăt de cablu de o anumită impedanță legăm un cablu de altă impedanță sau dacă emițătorul sau receptorul atașat are altă impedanță decât impedanța caracteristică a cablului, spunem că avem o *neadaptare de impedanță*. În acest caz, joncțiunea respectivă reflectă o parte din semnalul incident (este analog reflexiei luminii la trecerea din aer în sticlă, sau în general între medii cu indice de refracție diferit).

Reflexia produce două neajunsuri: pe de o parte scade puterea semnalului util ce ajunge la receptor, iar pe de altă parte un semnal ce suferă două reflexii succesive se poate suprapune peste semnalul util și, fiind întârziat față de acesta, îl distorsionează.

Impedanța se măsoară în *ohmi* (simbol Ω). Cablul pentru televiziune are impedanța de 75Ω . Cablul coaxial pentru rețea Ethernet are impedanța de 50Ω . Cablul torsadat Ethernet are 100Ω .

3.4.3. Legătură magistrală

La o pereche de conductoare pot fi conectate mai multe emițătoare sau receptoare. O astfel de interconectare poate avea două scopuri: pentru a realiza simplu o comunicație de tip difuziune (un emițător transmite simultan către mai multe receptoare) sau pentru a permite mai multor calculatoare să comunice fiecare cu fiecare.

O astfel de pereche de conductoare la care se leagă mai multe dispozitive se numește *magistrală*.

Realizarea mediului fizic, în acest caz, este complicată de necesitatea de a avea adaptare de impedanță în fiecare punct al mediului. În general, la simpla conectare a trei perechi de conductoare sau, echivalent, la ramificarea unei perechi apare, în punctul de ramificație, o neadaptare de impedanță.

Există dispozitive mai complicate (conținând transformatoare de semnal) care permit ramificarea unei perechi de conductoare fără a introduce o

neadaptare de impedanță, însă nu permit propagarea semnalului de la fiecare ramură spre toate celelalte.

O altă soluție de conectare a mai multor dispozitive (emițătoare sau receptoare) la un cablu constă în realizarea unei ramificații foarte scurte, astfel încât să nu aibă comportament de linie lungă (la frecvențele cu care se lucrează uzual, aceasta înseamnă cel mult câțiva centimetri), la capătul căreia se conectează emițătorul sau receptorul. Emițătorul sau receptorul astfel conectat trebuie să aibă impedanța de ieșire, respectiv de intrare, mult mai mare decât impedanța perechii de conductoare la care se conectează. O astfel de conectare se utilizează, de exemplu, în rețelele Ethernet vechi (vezi § 9.1.3 și fig. 9.1).

Dacă un capăt de pereche de conductoare este lăsat liber (neconectat), el produce reflexii. De fapt, un capăt neconectat poate fi văzut ca o joncțiune de la perechea ce are o anumită impedanță la un dispozitiv având impedanța infinită. Pentru evitarea reflexiilor, la capătul unei perechi de conductoare trebuie montat un dispozitiv numit *terminator*. Terminatorul este un simplu rezistor, având rezistența egală cu impedanța cablului. El absoarbe integral semnalul incident, neproducând nici un fel de reflexie. Notăm că terminatoarele sunt utilizate în mod normal doar pe legături magistrală; pe legăturile punct la punct, emițătorul și receptorul au, în mod obișnuit, impedanța necesară, astfel încât joacă și rol de terminator.

3.4.4. Considerente practice

Transmisia prin conductoare electrice este cea mai simplă de realizat deoarece calculatoarele însele folosesc intern semnale electrice pentru transmiterea, stocarea și prelucrarea informației. De asemenea, tăierea la dimensiune a cablurilor și montarea conectoarelor se pot realiza cu unelte relativ ieftine și fără a necesita prea multă calificare din partea lucrătorilor. Aceste motive fac ca, în majoritatea situațiilor practice, perechile de conductoare să fie încă cea mai potrivită soluție pentru comunicații pe distanțe mici.

Faptul că mediul de transmisie este conductor ridică însă probleme speciale, în situațiile în care prin conductoarele mediului de transmisie ajung să curgă curenți din alte surse.

Astfel, între carcusele, legate la pământarea rețelei electrice, a două calculatoare sau alte echipamente, poate apărea o tensiune electrică de ordinul câtorva volți; dacă echipamentele sunt conectate la rețelele electrice a două clădiri diferite, tensiunea dintre carcuse de propagare lor poate fi chiar mai mare. Pentru ca aceasta să nu perturbe semnalul util, în construcția plăcilor de rețea trebuie luate măsuri speciale de izolare. Dacă unul dintre conduc-

toare este expus atingerii cu mâna (este cazul la rețelele Ethernet cu cablu coaxial, unde conductorul exterior este legat la partea metalică exterioară a conectorilor), standardele de protecție la electrocutare cer legarea la pământ a conductorului respectiv; legarea la pământ trebuie însă făcută într-un singur punct pentru a evita suprapunerea peste semnalul util a tensiunilor dintre diverse puncte ale rețelei de pământare.

O altă sursă de tensiuni parazite între conductoarele de semnal sunt descărcările electrice din atmosferă (fulgerele și trăznetele). Deoarece în mod normal conductoarele rețelei sunt izolate față de rețeaua de pământare, fenomenele atmosferice pot induce tensiuni ridicate între conductoarele rețelei și carcasa echipamentelor, putând duce la distrugerea echipamentelor rețelei. Ca urmare, în cazul unor cabluri de rețea duse prin exteriorul clădirilor, este necesară fie ecranarea cablului și legarea ecranului la pământ, fie amplasarea unor *descărcătoare* care să limiteze tensiunea dintre conductoarele rețelei și pământ.

3.5. Transmisia prin unde radio

Undele electromagnetice sunt oscilații ale câmpului electromagnetic. Aceste oscilații se propagă din aproape în aproape.

Frecvența unei unde electromagnetice este frecvența de oscilație a câmpului electromagnetic într-un punct fixat din spațiu.

Lungimea de undă a unei unde este distanța parcursă de undă în timpul unei oscilații complete. Lungimea de undă se notează cu λ și are valoarea $\lambda = \frac{v}{f}$, unde f este frecvența și v este viteza de propagare. Viteza de propagare depinde de mediul în care se propagă unda. Ca urmare, lungimea de undă se modifică la trecerea dintr-un mediu în altul.

Lungimea de undă se utilizează adesea în locul frecvenței pentru a caracteriza unda. În acest caz lungimea de undă se calculează pentru viteza de propagare a undelor electromagnetice în vid $v = c = 3 \times 10^8$ m/s.

Viteza de propagare în aer este foarte apropiată de viteza în vid; pentru majoritatea scopurilor cele două viteze pot fi considerate egale.

Undele radio sunt unde electromagnetice având frecvențe la care pot să lucreze dispozitivele electronice; în funcție de autori, limita de jos a frecvențelor undelor radio este cuprinsă între 30 Hz ($\lambda = 10000$ km) și 3 kHz ($\lambda = 100$ km), iar limita de sus a frecvențelor este cuprinsă între 1 GHz ($\lambda = 30$ cm) și 300 GHz ($\lambda = 1$ mm), cu observația că undele electromagnetice din intervalul 1 GHz – 300 GHz se numesc *microunde* și unii autori consideră că microundele nu fac parte dintre undele radio ci sunt o categorie separată de acestea.

De interes practic în rețelele de calculatoare sunt undele radio în intervalul 300 MHz – 30 GHz, sau echivalent, cu lungimile de undă cuprinse între 1 m și 1 cm.

La transmisia prin unde radio, mărimile fizice utilizate ca semnal sunt intensitatea câmpului electric și inducția magnetică. Cele două mărimi sunt proporționale în modul și au direcții perpendiculare una pe cealaltă și pe direcția de propagare a undei.

Într-un sistem de transmisie prin unde radio, emițătorul cuprinde două blocuri distincte: un dispozitiv electronic, care produce un semnal de tip tensiune și intensitate electrică, și *antena*, care convertește semnalul din tensiune și intensitate electrică în câmp electromagnetic.

Receptorul constă de asemenea dintr-o antenă, care plasată în calea undelor electromagnetice transformă semnalul din câmp electromagnetic în tensiune și intensitate electrică, și un dispozitiv electronic, care decodifică semnalul electric.

Orice antenă poate servi atât la emisie cât și la recepție. (Singura diferență ce apare între antene este că antenele de emisie de putere mare trebuie construite astfel încât să suporte tensiunile și curenții mari ce apar în elementele lor.)

Mai multe proprietăți ale sistemului de transmisie fac ca lățimea benzii de trecere a întregului sistem să fie îngustă în raport cu frecvențele între care se încadrează banda de trecere; raportul între lățimea benzii și limita inferioară a benzii este în mod tipic de cel mult câteva procente. Din această cauză, transmisia prin unde radio este întotdeauna cu modulație, iar frecvența purtătoare este cel puțin de câteva zeci de ori mai mare decât lățimea de bandă.

De exemplu, pentru o viteză de transmisie de 10 Mbit/s avem în mod tipic nevoie de o lățime de bandă apropiată de 10 MHz, pentru care frecvența purtătoare va fi de cel puțin 200 MHz.

3.5.1. Propagarea undelor

3.5.1.1. Polarizarea

Câmpul electromagnetic se caracterizează prin două mărimi vectoriale, definite pentru fiecare punct din spațiu: *intensitatea câmpului electric*, notată cu \vec{E} , și *inducția magnetică*, notată cu \vec{B} .

Într-un fascicul de unde electromagnetice, paralel și mult mai lat decât lungimea de undă, vectorii \vec{E} și \vec{B} sunt întotdeauna perpendiculari unul pe celălalt și pe direcția de deplasare a undelor.

Dacă \vec{E} are direcție constantă și îi variază doar sensul și modulul,

fasciculul se numește *polarizat liniar*. Un fascicul polarizat liniar se caracterizează prin direcția vectorului \vec{E} , numită *direcția de polarizare*.

Dacă \vec{E} are modul constant și direcția lui se rotește uniform, în plan perpendicular pe direcția de deplasare a undei, fasciculul se numește *polarizat circular*. Se distinge *polarizare circulară stângă* dacă, privind în direcția de propagare a undelor, dinspre emițător spre receptor, direcția lui \vec{E} se rotește în sens invers acelor de ceas; și *polarizare circulară dreaptă* dacă \vec{E} se rotește în sensul acelor de ceas.

Un fascicol cu polarizare circulară rezultă de fapt prin suprapunerea a două fascicole, de amplitudine egală, polarizate perpendicular unul pe celălalt, deplasându-se în aceeași direcție și cu un decalaj de un sfert de ciclu între ele. Dacă cele două fascicole au amplitudini diferite, rezultă ceea ce se numește *polarizare eliptică*; polarizarea liniară și polarizarea circulară sunt de fapt cazuri particulare de polarizare eliptică.

3.5.1.2. Absorbția și reflexia

Absorbția undelor radio în aer este neglijabilă.

Picăturile de apă (din ploaie, nori, ceață) absorb destul de puternic undele radio, în special microundele. Apa absoarbe puternic toate undele radio; de aceea este greu de obținut legătură radio sub apă. Absorbție moderată se produce în pământ și în diferite materiale de construcție.

Scăderea puterii undelor radio datorită absorbției este exponențială cu distanța, ca și în cazul propagării semnalelor prin cabluri.

Metalele reflectă undele radio. Plasele metalice care au contact bun între firele componente și au ochiurile mult mai mici decât lungimea de undă se comportă ca o suprafață metalică compactă. Armăturile clădirilor din beton armat nu fac contact electric prea bun între ele, însă perturbă serios propagarea undelor radio.

Ionosfera reflectă undele cu lungimi de undă de ordinul metrilor; prin reflexii repetate între Pământ și ionosferă, aceste unde pot parcurge ușor multe mii de kilometri.

3.5.1.3. Difracția

Orice undă ocolește obstacolele mai mici decât o fracțiune din lungimea de undă, în vreme ce în spatele obstacolelor mai mari de câteva lungimi de undă „rămâne umbră“. De aceea, undele lungi, cu lungime de undă de ordinul kilometrilor sau sutelor de metri sunt capabile să ocolească obstacole mari, inclusiv curbura Pământului pe distanță de câteva sute sau chiar mii de kilometri. Prin contrast, undele cu lungime de undă sub câțiva metri se propagă aproape numai în linie dreaptă, dealurile sau clădirile mai mari putând provoca umbre.

3.5.1.4. Interferența undelor

Dacă într-un punct ajung unde pe mai multe căi, de exemplu o cale directă și o cale prin reflexia pe un obstacol, unda recepționată în acel punct este suma undelor ce ajung pe toate căile.

Dacă diferența de drum între două căi este un număr întreg de lungimi de undă, dar mult mai mică decât lungimea unui bit, undele se suprapun în fază și se adună, semnalul recepționat fiind mai puternic. Dacă diferența de drum este apropiată de un număr impar de lungimi de undă, undele se suprapun în antifază și se anulează reciproc, semnalul recepționat fiind slab sau nul. În aceste situații, deplasarea receptorului (sau emițătorului) pe o distanță de la un sfert din lungimea de undă și până la de câteva ori lungimea de undă poate modifica mult calitatea semnalului (reamintim că în transmisiile de date se utilizează lungimi de undă cuprinse între 1 cm și 1 m). Schimbarea lungimii de undă pe care se face transmisia poate de asemenea modifica mult efectul.

Dacă diferența de drum între semnalele recepționate pe căi diferite este comparabilă sau mai mare decât lungimea unui bit și puterile semnalului pe cele două căi sunt apropiate, semnalele propagate pe cele două căi se bruiază reciproc. Situația apare mult mai rar decât cea prezentată mai sus, însă nu poate fi corectată decât prin mutarea stațiilor față de obstacolele ce produc reflexiile.

3.5.1.5. Divergența undelor

Pe măsură ce ne depărtăm de emițător, puterea semnalului scade datorită extinderii frontului de undă. Densitatea puterii este invers proporțională cu suprafața frontului de undă, care la rândul ei este proporțională cu pătratul distanței față de emițător.

Ca urmare, puterea recepționată P_r este invers proporțională cu pătratul distanței d dintre emițător și receptor:

$$P_r = P_e \cdot \alpha \cdot \frac{1}{d^2}$$

unde α este o constantă ce depinde de construcția antenelor de emisie și de recepție, iar P_e este puterea emițătorului.

Scăderea puterii datorită extinderii frontului de undă este independentă de eventuala absorbție a undelor în mediu; aceasta din urmă duce la o scădere exponențială cu distanța a puterii semnalului.

3.5.2. Antene

O *antena* este un dispozitiv care realizează conversia între un semnal electric (tensiune și intensitate electrică) pe o pereche de conductoare și

oscilațiile electromagnetice în mediul înconjurător antenei. Orice antenă este reversibilă: dacă i se aplică un semnal electric la borne, va radia unde electromagnetice și, reciproc, dacă este plasată în calea undelor electromagnetice, va produce semnal electric la borne.

În general o antenă este optimizată pentru o anumită bandă de trecere.

O antenă are un anumit *randament*, definit ca raportul dintre puterea unei electromagnetice radiate și puterea absorbită din semnalul electric primit.

3.5.2.1. Directivitatea

O antenă nu radiază uniform de jur împrejur. Prin *câștigul* (engl. *gain*) unei antene *pe o direcție* se înțelege raportul dintre puterea radiată pe acea direcție și puterea radiată de o antenă etalon, în aceleași condiții. Ca etalon se utilizează de obicei o antenă ipotetică care ar radia egal în toate direcțiile și ar avea randamentul 100%. Deoarece energia se conservă, câștigul este pe unele direcții supraunitar și pe altele subunitar, integrala lui pe întreaga sferă fiind $4\pi\eta$ (unde η reprezintă randamentul antenei). Câștigul este dat uneori direct, alteori este dat logaritmul câștigului, exprimat în decibeli.

Câștigul antenei pe diverse direcții este reprezentat grafic prin *diagramele de câștig*. O astfel de diagramă este o reprezentare a câștigului ca funcție de unghi pe toate direcțiile dintr-un plan.

O direcție de maxim local al câștigului, împreună cu direcțiile apropiate, se numește *lob*. Lobul care cuprinde maximum global al câștigului se numește *lobul principal* al antenei. Ceilalți lobi se numesc *lobi secundari* sau *lobi laterali*. Valoarea maximă, pentru toate direcțiile posibile, a câștigului este numită *câștigul antenei*.

O antenă optimizată să aibă câștig cât mai mare pe o direcție, în detrimentul celorlalte direcții, se numește *antenă directivă*. O antenă optimizată pentru a avea câștig cât mai uniform, cel puțin în planul orizontal, se numește *antenă nedirectivă*. O antenă cu câștig perfect uniform de jur împrejur (radiator izotrop) este imposibil de realizat.

Există o legătură între dimensiunea antenei, directivitatea și lungimea de undă la care funcționează. Anume, raza unghiulară a lobului principal (măsurată în radiani) nu poate fi mai mică decât raportul dintre diametrul antenei și lungimea de undă. Ca exemplu, pentru a obține un lob principal de 3° ($\approx 0,05$ rad) la o lungime de undă de 6 cm ($f = 5$ GHz) avem nevoie de o antenă de cel puțin 1,2 m diametru. Limitarea aceasta este legată de fenomenele de difracție a undelor și nu poate fi ocolită.

O antenă de recepție plasată în calea undelor recepționează o putere proporțională cu densitatea de putere a undei incidente. Raportul dintre puterea disponibilă la bornele antenei și densitatea de putere a undei incidente se numește *aria efectivă* sau *apertura* antenei. Apertura poate fi privită ca suprafața, transversală pe direcția de propagare a undelor, de pe care antena preia întreaga energie. Apertura depinde de direcția considerată a undei incidente.

Apertura față de o anumită direcție a undei incidente este proporțională cu câștigul antenei pe acea direcție. Relația este:

$$S = G \frac{\lambda^2}{4\pi} \quad (3.6)$$

unde S este aria efectivă, G este câștigul, iar λ este lungimea de undă.

Utilizând relația (3.6), se poate calcula puterea recepționată, dacă distanța dintre emițător și receptor este mult mai mare decât dimensiunile antenelor:

$$\begin{aligned} P_r &= P_e \cdot G_e \cdot \frac{1}{4\pi d^2} \cdot S_r = \\ &= P_e \cdot G_e \cdot \frac{\lambda^2}{16\pi^2 d^2} \cdot G_r \end{aligned}$$

unde P_r este puterea disponibilă la bornele antenei receptoare, P_e este puterea aplicată la bornele antenei emițătoare, d este distanța dintre emițător și receptor, G_e este câștigul emițătorului pe direcția spre receptor, iar G_r și S_r sunt respectiv câștigul și apertura antenei receptoare pe direcția spre emițător.

EXEMPLUL 3.1: Considerăm un emițător (de exemplu, un calculator dintr-o rețea IEEE 802.11 — *wireless*) care emite un semnal cu puterea $P_e = 100$ mW (sau, echivalent, +20 dBm) și frecvența $f = 2,4$ GHz (lungimea de undă este atunci $\lambda = 0,125$ m). Mai presupunem că receptorul se găsește la o distanță $d = 100$ m față de emițător, că absorbția semnalului este neglijabilă (emițătorul și receptorul se găsesc în câmp deschis și nu plouă) și că ambele antene au un câștig $G_e = G_r = 2$ pe direcția spre partenerul de comunicație. Rezultă puterea semnalului recepționat:

$$P_r = 10^{-1} \text{ W} \cdot 2 \cdot \frac{(0,125 \text{ m})^2}{16\pi^2 (100 \text{ m})^2} \cdot 2 \approx 3,9 \cdot 10^{-9} \text{ W},$$

adică aproximativ -84 dBm.

3.5.2.2. Polarizarea

Antenele cele mai simple au polarizare liniară: unda emisă este polarizată liniar, pe o direcție stabilită prin construcția antenei. Rotirea antenei emițătorului față de cea a receptorului duce la variația semnalului recepționat între un maxim (când direcțiile polarizărilor celor două antene sunt paralele) și un minim (teoretic zero) când direcțiile sunt perpendiculare.

O antenă polarizantă liniar va recepționa întotdeauna, indiferent de direcția de polarizare, o transmisie polarizată circular; reciproc, o antenă polarizantă circular va recepționa o emisie polarizată liniar. O antenă polarizantă circular va recepționa o transmisie polarizată circular numai dacă are același sens al polarizării. Rotirea antenelor în jurul dreptei ce le unește nu are efect.

3.5.2.3. Tipuri de antene

Antenele nedirective sunt de cele mai multe ori un simplu baston metalic (de fapt, bastonul este un pol, iar carcasa aparatului sau, după caz, Pământul, este celălalt pol). O astfel de antenă are câștig maxim în planul orizontal (perpendicular pe baston) și zero pe direcție verticală (în lungul bastonului). Undele produse sunt polarizate vertical.

Antenele directive cele mai răspândite pentru comunicații de date sunt așa-numitele *antene parabolice* (denumire improprie, pentru că forma parabolică este a reflectorului antenei). O astfel de antenă este alcătuită dintr-o oglindă în formă de paraboloid de rotație, în focarul căreia este plasată antena propriu-zisă. (În alte construcții, antena propriu-zisă este plasată în altă parte, iar unda electromagnetică este adusă în focarul reflectorului parabolic printr-un tub metalic numit *ghid de undă*.)

3.5.3. Raza de acțiune a unei legături radio

Spre deosebire de legăturile prin perechi de conductoare sau prin fibre optice, legăturile prin unde radio nu pot fi delimitate net la un anumit domeniu. Dăm în continuare factorii care influențează raza de acțiune a unei legături radio. Uneori vom dori să îi contracarăm, pentru a extinde domeniul de acțiune, alții dimpotrivă, îi vom dori să ne mențină o legătură radio într-un domeniu spațial limitat pentru a nu interfera cu legături radio din apropiere. Cabluri electrice sau optice putem duce câte dorim; câmp electromagnetic este numai unul. . .

3.5.3.1. Obstacolele

Obstacolele limitează raza de acțiune a legăturii radio. Mai mult, din cauza interferenței dintre undele reflectate pe diferite căi, este dificil de analizat

exact punctele în care este posibilă recepția unei emisii radio și punctele în care emisia este obstrucționată.

3.5.3.2. Linia orizontului

Unul dintre obstacolele ce limitează raza de acțiune a undelor radio este însuși Pământul, prin curbura suprafeței sale. O stație aflată la o anumită înălțime poate comunica cu o stație aflată la nivelul solului dacă și numai dacă stația de pe sol se află mai aproape decât linia orizontului celeilalte stații. Două stații pot comunica dacă există cel puțin un punct comun orizontului celor două stații.

În câmpie, distanța până la linia orizontului este (r desemnează raza Pământului, iar h este înălțimea antenei deasupra suprafeței Pământului):

- măsurată de-a lungul curburii, de la baza turnului în care se află observatorul: $d = r \cdot \arccos \frac{r}{h+r}$;
- măsurată în linie dreaptă de la observator:

$$d = \sqrt{(r+h)^2 - r^2} = \sqrt{h(2r+h)};$$

- dacă $h \ll r$, $d \approx \sqrt{2rh}$. De remarcat că dacă exprimăm numeric $2r$ în mii de kilometri ($2r \approx 12,7 \times 10^3$ km) și h în metri, distanța d rezultă în kilometri.

Exemple:

Distanța până la linia orizontului pentru un observator aflat la 1,6 m deasupra pământului (de exemplu un radiotelefon ținut în mână) este $d = \sqrt{12,7 \cdot 1,6}$ km $\approx 4,5$ km.

Un turn cu înălțimea de 20 m (obișnuit pentru un releu GSM) are linia orizontului la 16 km. O stație aflată într-un astfel de turn poate comunica cu un radiotelefon ținut în mână la o distanță de $16 \text{ km} + 4,5 \text{ km} = 20,5 \text{ km}$ (de regulă raza de acțiune a unui releu GSM este limitată de alte considerente).

De pe un turn cu înălțimea de 50 m, distanța la linia orizontului este $d = \sqrt{12,7 \cdot 50}$ km ≈ 25 km. Două relee de telecomunicații având 50 m înălțime fiecare pot comunica direct dacă sunt la mai puțin de 50 km unul de altul.

Distanța la linia orizontului crește încet cu înălțimea; dacă se dublează înălțimea, distanța la linia orizontului crește cu un factor de $\sqrt{2} \approx 1,4$.

3.5.3.3. Utilizarea sateliților artificiali ai Pământului

Sateliții artificiali ai Pământului sunt utilizați ca echivalentul unor turnuri înalte pentru montarea unor stații radio. După altitudinea la care

sunt plasați, distingem trei categorii de sateliți:

sateliți de joasă altitudine aflați între 200...1000 km, cu perioada de rotație de 1,5...1,8 h;

sateliți de altitudine medie între 10000...15000 km (raza orbitei de 3–4 ori raza Pământului), cu perioada de rotație de 6...9 h;

sateliți geostaționari aflați la 35800 km deasupra ecuatorului, au perioada de rotație de exact o zi și ca urmare apar ficși față de Pământ.

Un satelit are o arie de acoperire incomparabil mai mare față de o stație terestră. La 200 km altitudine, un satelit acoperă o rază de 1500 km, iar un satelit de medie altitudine acoperă o rază de peste 7000 km.

Din cauza distanțelor mari, comunicația cu sateliții necesită fie puteri mari, fie antene cu directivitate foarte bună. Este de remarcat faptul că distanța de la un satelit la o stație terestră este de la câteva zeci la câteva sute de ori mai mare decât distanța de la un releu amplasat într-un turn la o stație terestră. Ca urmare, pentru aceleași antene, puterile necesare sunt de la câteva sute la câteva sute de mii de ori mai mari.

La comunicația între sateliți geostaționari și stații fixe de pe sol se pot utiliza relativ ușor antene cu directivitate bună, deoarece antenele de pe sol sunt fixe. Orbita geostaționară este însă destul de „aglomerată”: presupunând că avem antene ce dau un fascicul cu diametrul unghiular de 6° , (vezi exemplul în care rezulta, pentru $f = 5$ GHz, un diametru al antenei de peste 1,2 m) putem distinge doar între 60 de sateliți distincți.

Pentru sateliții care nu sunt geostaționari, utilizarea antenelor directive necesită un sistem foarte complicat de urmărire a satelitului.

3.5.3.4. Zgomotul

Zgomotul în transmisiile radio provine din multe surse, între altele aparatură electronică, întrerupătoare electrice (inclusiv colectoarele motoarelor de curent continuu). Transmisiile radio sunt mult mai sensibile la zgomot decât transmisiile prin conductoare electrice, deoarece la conductoare electrice undele radio pătrund accidental în semnal, din cauza ecranării imperfecte, pe câtă vreme la transmisiile radio semnalul util se amestecă direct cu zgomotul radio ambiant.

Nivelul zgomotului radio ambiant este un factor important care limitează inferior pragul de sensibilitate al receptorului și, în consecință, fixează puterea minimă pentru o anumită distanță emițător-receptor.

Nivelul de zgomot scade în general o dată cu creșterea frecvenței.

3.5.3.5. Scăderea puterii cu distanța

Densitatea de putere a undelor electromagnetice scade cu pătratul distanței de la emițător. Ca urmare, la o sensibilitate fixată a receptorului, pentru a dubla raza de acțiune a emițătorului trebuie să-i creștem puterea de 4 ori.

Pe de altă parte, dacă două emițătoare radio funcționează în aceeași regiune geografică și emit pe frecvențe identice sau foarte apropiate, atunci transmisia mai puternică „acoperă” transmisia mai slabă. Aceasta se întâmplă deoarece semnalele celor două emițătoare se suprapun. Dacă, în punctul în care este plasat receptorul, puterea unuia dintre emițătoare este mult mai mare decât puterea celuilalt, atunci receptorul va recepționa doar transmisia mai puternică, chiar dacă, singură, transmisia mai slabă ar putea fi recepționată corect. Dacă puterile sunt apropiate, receptorul nu va putea „înțelege” nici una dintre transmisii.

3.5.3.6. Emisia direcționată și polarizată

Domeniul de acțiune a unui emițător sau receptor poate fi restrâns în mod voit dotând emițătorul sau receptorul (de obicei ambele) cu antene directive. Trebuie însă calculate cu atenție divergența lobului principal, puterea emisă pe lobi secundari ai antenei și reflexiile de teren.

Polarizarea se poate utiliza pentru a separa două transmisii pe aceeași direcție și pe aceeași lungime de undă. În cazul utilizării polarizării liniare, cele două transmisii trebuie să utilizeze direcții de polarizare perpendiculare; în cazul polarizării circulare se vor folosi cele două sensuri (stânga și dreapta). Lobii secundari ai antenelor, precum și undele reflectate de diverse corpuri, au polarizări greu de controlat.

3.5.4. Spectrul radio și alocarea lui

Începem cu o precizare de terminologie: în general când este vorba de semnale, termenul de *frecvență* se utilizează cu sensul de frecvența unei componente în descompunerea Fourier a semnalului, iar termenul de *bandă* se folosește cu sensul de interval de frecvențe între care se încadrează spectrul Fourier al unui semnal.

În comunicații radio, termenul de frecvență se utilizează adesea și cu sensul de interval de frecvențe în care se încadrează o transmisie (efectiv, bandă în sensul de la semnale). Frecvențe diferite, în acest sens, înseamnă de fapt benzi disjuncte. Valoarea numerică a frecvenței, specificată în acest context, este frecvența purtătoare utilizată. Limitele efective ale benzii se determină din standardul de transmisie folosit.

Noțiunea de *bandă în care se face transmisia* specifică în acest context un interval de frecvențe alocat pentru o anumită categorie de transmisii radio. Benzile, în acest sens, se specifică fie printr-o anumită frecvență sau lungime de undă, din interiorul benzii, și având o valoare „rotundă“, fie printr-un nume. Limitele benzii se găsesc în standarde.

Două transmisii radio ce se fac pe frecvențe diferite, sau mai precis, a căror benzi de trecere sunt disjuncte, pot fi separate în general ușor. Separarea în frecvență este mult mai ușor controlabilă decât separarea spațială studiată în § 3.5.3. Două transmisii pe aceeași frecvență și în aceeași zonă geografică sunt practic imposibil de separat, dacă au puteri apropiate, sau transmisia mai slabă este imposibil de recepționat fiind „acoperită“ de cea mai puternică.

Pentru evitarea suprapunerilor între utilizatori, utilizarea diverselor benzi de frecvențe face obiectul unor reglementări legale în fiecare țară, precum și a unor acorduri internaționale. Emiterea unui semnal radio, pe o frecvență pentru care operatorul emițătorului nu este autorizat sau de o putere mai mare decât cea autorizată, poate duce la sancționarea contravențională sau chiar penală a operatorului.

În majoritatea cazurilor, un utilizator de comunicații radio care dorește să opereze un emițător trebuie să obțină o autorizație în care se specifică frecvența de lucru, puterea maximă, zona geografică în care operează, etc. Există frecvențe alocate posturilor de radio, sistemelor de comunicații radio ale diferitelor instituții (poliție, controlorii de trafic aerian, dispecerate de taxiuri, operatori de telefonie mobilă, etc.). Tot în această categorie, însă cu un statut aparte sunt radioamatorii: frecvențele sunt alocate activității de radioamator și nu unei persoane sau instituții, însă radioamatorii trebuie să se înregistreze pentru a putea emite.

Există însă benzi pentru care nu este necesară o autorizare expresă a emițătorului, cu condiția ca emițătorul să nu depășească o anumită putere. În această categorie intră frecvențele folosite de: rețelele IEEE 802.11 (Wireless Ethernet) și Bluetooth, tastaturi și mași fără fir, telefoanele fără fir, microfoanele fără fir, walkie-talkie-urile de jucărie, jucării cu telecomandă prin radio, telecomenzi pentru deschis garajul. Utilizatorul unor astfel de echipamente trebuie totuși să fie atent la eventualele diferențe între reglementările din diferite țări: un echipament poate funcționa legal fără autorizație în țara de origine, dar să necesite autorizație în altă țară.

Echipamentele care lucrează pe frecvențe pentru care nu trebuie autorizare ajung să interfereze dacă sunt plasate în apropiere. Unele dintre acestea permit selectarea frecvenței de lucru dintre 2–4 frecvențe predefinite. Utilizatorul va selecta o frecvență diferită dacă constată o funcționare proastă

și suspectează interferențe cu echipamente vecine. Altă soluție este schimbarea repetată a frecvenței de lucru, după o schemă convenită între emițător și receptor, și tolerarea unui număr de ciocniri ale transmisiilor pe perioadele în care echipamentele vecine se nimeresc aceeași frecvență. Tehnica se numește *frequency hopping* (salturi ale frecvenței).

Mai menționăm că, printre producătorii de semnale radio parazite intră și alte dispozitive, având alte scopuri decât comunicațiile. Ca fapt divers, enumerăm câteva:

- Sursele de alimentare de la aproape toate aparatele electronice moderne (așa-numitele *surse în comutație*), precum și blocul de baleiaj de linii de la televizoarele și monitoarele cu tub catodic, emit semnificativ pe frecvențe până la câteva sute de kiloherți (așa-numitele armonice, adică frecvențe care sunt multipli ai frecvenței de lucru a circuitului). Funcționarea acestora bruiază adesea posturile de radio pe unde lungi și uneori chiar medii.
- Radioemițătoarele emit și pe frecvențe ce sunt multipli ai frecvenței purtătoare (armonice). Din acest motiv, se întâmplă uneori ca un post de televiziune să apară, cu semnal foarte slab, și pe un canal superior celui pe care este transmis normal (dar atenție, uneori acest efect este datorat recepției de la un alt releu de televiziune, mai îndepărtat).

3.5.5. Particularități ale sistemelor de comunicație prin radio

3.5.5.1. Topologia legăturii

Legăturile între releele de comunicație radio, amplasate în turnuri și dotate cu antene parabolice, sunt în general punct la punct, ca în cazul legăturilor prin perechi de conductoare.

Legăturile între sateliții geostaționari și stațiile terestre sunt astfel că emisia satelitului este recepționată de mai multe stații de pe Pământ, și reciproc, satelitul recepționează emisia de la mai multe stații de pe Pământ; stațiile de pe Pământ nu comunică însă direct între ele. O astfel de comunicație poate prezenta riscul ca emisiile stațiilor de pe Pământ să se ciocnească fără ca stațiile să observe direct acest lucru.

La echipamente mobile există mai multe posibilități. Pentru distanțe mari, una din stații este fixă și se plasează într-un turn de unde poate comunica direct cu toate celelalte. Celelalte stații nu se „văd“ direct una pe alta și de cele mai multe ori nici dacă „se văd“ protocoalele folosite nu permit comunicații directe între ele (exemplu: telefoanele GSM). Stația centrală primește rol de arbitraj al transmisiilor.

Pentru distanțe mici, se poate adopta o organizare mai „democratică” (exemplu IEEE 802.11): stațiile comunică direct între ele, iar arbitrarea mediului se face prin mijloace asemănătoare cu cele utilizate pe cabluri magistrală (§ 4.2). Spre deosebire însă de cablurile magistrală, unde un pachet emis de o stație de pe cablu este recepționat de toate celelalte și, ca urmare, ciocnirea la recepție a două pachete este sesizată și de către emițătoare, la legăturile radio este posibil ca două transmisii să se ciocnească la receptor dar nici una din stațiile care le-au emis să nu recepționeze transmisia celeilalte.

3.5.5.2. Fiabilitatea

Fiabilitatea unei legături radio este în general mai scăzută decât a unei legături pe cablu:

- Rata de erori este mult mai mare. La o legătură radio, probabilitatea unei erori de un bit este în mod normal de $10^{-3} \dots 10^{-5}$. Pentru comparație, la transmisia prin perechi de conductoare, probabilitatea unei erori de un bit este de $10^{-7} \dots 10^{-10}$, iar la fibrele optice, erorile sunt și mai rare, $10^{-10} \dots 10^{-12}$.
- La frecvențe peste 10 GHz, datorită absorbției în picăturile de apă, starea legăturii poate depinde de starea vremii.
- Umbrele provocate de clădiri și relief, precum și interferențele între undele reflectate, sunt imposibil de calculat în mod practic. O stație ce ajunge în umbră va pierde legătura în mod imprevizibil.

3.5.5.3. Securitatea

La comunicațiile prin cablu pe distanță scurtă, securitatea comunicației poate fi asigurată păzind cablul. Din acest motiv rețelele locale pe cablu pot să nu prevadă măsuri contra intrușilor.

Undele radio nu pot fi păzite, analog cablului. Rețelele fără fir este esențial să aibă incorporate măsuri de securitate. Acestea presupun metode criptografice (vezi capitolul 6) ce previn ascultarea sau contrafacerea unui mesaj, și eventual schimbarea frecvenței (metoda *frequency hopping*) pentru a preveni bruiatul.

3.6. Transmisia optică

Transmisia optică este de fapt tot o transmisie prin unde electromagnetice, dar cu frecvențe mult mai mari, anume din intervalul cuprins între $1,6 \times 10^{14}$ Hz ($\lambda = 1,8 \mu\text{m}$) și $3,7 \times 10^{14}$ Hz ($\lambda = 0,8 \mu\text{m}$). Aceste unde electromagnetice fac parte din categoria undelor infraroșii. Vom folosi termenul de

lumină pentru aceste unde, deși nu se încadrează în domeniul luminii vizibile ($\lambda = 780 \text{ nm} \dots 380 \text{ nm}$).

Mărimea considerată ca semnal este puterea luminoasă. Am putea considera, în mod echivalent, că semnalul transmis de mediu este intensitatea câmpului electric sau inducția magnetică și că utilizăm modulație în amplitudine pentru a transmite semnalul util.

Emisia și recepția se realizează cu dispozitive semiconductoare capabile să emită raze infraroșii la trecerea curentului prin ele (LED-uri, asemănătoare celor de pe panourile de aparate, sau, după caz, diode laser) și, respectiv, care permit trecerea curentului doar în prezența luminii.

Pentru unele aplicații, presupunând comunicație pe distanță de cel mult câțiva metri (de exemplu, pentru telecomenzi de televizoare sau pentru dispozitive IrDA), raza de lumină se propagă direct prin aer de la emițător la receptor. Metoda este dificil de extins la distanțe mai mari.

Raza de lumină poate fi însă foarte ușor ghidată printr-o fibră optică. O fibră optică este în esență un fir dintr-un material transparent, prin interiorul căruia trece lumina. Dacă raza de lumină lovește peretele lateral al fibrei, se întoarce înapoi în fibră. În acest fel, lumina ce intră printr-un capăt al fibrei iese prin celălalt capăt chiar dacă fibra nu este perfect dreaptă.

Fibra optică se mai numește și *ghid de undă optic* (engl. *optical waveguide*), deoarece este identic ca și scop și foarte asemănător funcțional cu ghidul de undă utilizat pentru microunde.

Lungimea fibrei, între emițător și receptor, poate atinge câteva zeci de kilometri. Lucrurile care fac posibilă atingerea unor distanțe atât de mari sunt atenuarea mică (sub 1 dB/km) și imunitatea aproape perfectă la zgomot.

3.6.1. Construcția mediului

Constructiv, o fibră optică este alcătuită dintr-un *miez* (engl. *core*) din silica (bioxid de siliciu, SiO_2 , amorf), înconjurat de un *înveliș* (engl. *cladding*), tot din silica, dar cu un indice de refracție puțin mai mic. Diametrul miezului este principalul parametru dat la o fibră optică; este cuprins între $8 \mu\text{m}$ și $62,5 \mu\text{m}$. Diametrul învelișului este în mod curent de $125 \mu\text{m}$. Pentru comparație, diametrul firului de păr uman este de $20 \dots 30 \mu\text{m}$.

Între miez și înveliș poate fi o discontinuitate netă, sau se poate ca indicele de refracție să scadă gradual. Fibrele cu discontinuitate netă se numesc *fibre optice cu discontinuitate* (engl. *step index fiber*) iar fibrele cu trecere graduală de la miez la înveliș se numesc *fibre optice graduale* (engl. *grade index fiber*).

Fibra propriu-zisă fiind extrem de subțire și fragilă, ea este învelită

în mai multe straturi cu rol de protecție mecanică.

Ideea de bază a conducerii semnalului prin fibră este că o rază de lumină ce se propagă oblic prin miez și atinge suprafața de contact dintre miez și înveliș să se reflecte înapoi în miez. Reflexia trebuie să fie cu pierderi extrem de mici, deoarece o rază se va reflecta de multe ori de la un capăt la celălalt al fibrei.

3.6.1.1. Conectarea fibrelor optice

Problemele legate de conectarea fibrelor optice reprezintă principalul dezavantaj al fibrelor optice față de perechile de conductoare. Conectarea cap la cap a două tronsoane de fibră se poate face:

- *prin lipire*, încălzind fibra până la temperatura de topire a sticlei și având grijă ca să se lipească capetele dar să nu se amestece miezul cu învelișul. Conectarea prin lipire necesită echipamente mai scumpe, este nedemontabilă, dar perturbă cel mai puțin transmiterea semnalului prin fibră. O lipitură produce o atenuare a semnalului în jur de 0,1 dB, din cauza reflexiei unei părți a luminii incidente.
- *prin conectoare optice*. Fiecare capăt de fibră se șlefuieste foarte bine și se prinde într-o piesă metalică cu rol de ghidaj. Piese metalice atașate capetelor de fibră se strâng una față de cealaltă, realizând alinierea față în față a capetelor de fibră. Eventual, spațiul dintre capetele de fibră se poate umple cu un gel transparent cu indice de refracție apropiat de cel al fibrei, reducând astfel reflexia la capătul fibrei.

3.6.2. Propagarea semnalului optic

3.6.2.1. Moduri de propagare

Dacă diametrul fibrei nu este mai mare de câteva zeci de ori lungimea de undă a luminii, modelul opticii geometrice — propagarea luminii sub forma de raze — nu mai este o aproximare acceptabilă a fenomenelor ce au loc. Din studiul ecuației undelor rezultă doar un număr finit de soluții, numite *moduri de propagare*. Intuitiv, un mod este un posibil traseu al razei de lumină, traversând în mod repetat, în zig-zag, axul fibrei și păstrând un unghi fixat față de acesta; în fibre suficient de subțiri, doar anumite unghiuri sunt permise.

Dacă o fibră permite existența mai multor moduri de propagare a luminii, fibra se numește *multimod*. Modurile diferite se propagă în general cu viteze puțin diferite. Intuitiv, acest lucru se întâmplă deoarece viteza de propagare a semnalului în fibră este egală cu valoarea componentei longitudinale a vitezei de propagare a luminii, care depinde de unghiul dintre direcția

de propagare a luminii și axa fibrei. Datorită vitezelor diferite, semnalul emis de la un capăt al fibrei este distorsionat, fiind recepționat la celălalt capăt ca mai multe copii puțin decalate în timp. Acest fenomen de distorsionare a semnalului se numește *dispersie intermodală*.

Opusul fibrei multimod este fibra *monomod*, în care ecuația undelor admite o singură soluție. Existența unui singur mod elimină dispersia intermodală, îmbunătățind calitatea propagării semnalului. Pentru a admite un singur mod, fibra trebuie să fie mult mai subțire, diametrele standard fiind 10 μm sau 8 μm . Diametrul mai mic al fibrei atrage două dificultăți: pe de o parte, cerințele de aliniere mecanică a fibrei față de sursă sunt mai stricte, iar pe de altă parte densitatea de putere luminoasă emisă prin fibră trebuie să fie mai mare. Acest din urmă fapt duce la necesitatea utilizării diodelor laser ca sursă de lumină (LED-urile nu mai sunt adecvate) și, în consecință, la creșterea prețurilor echipamentelor.

3.6.2.2. Caracteristici ale mediului

Dăm în continuare caracteristicile principale ale propagării:

viteza de propagare este viteza luminii în silica, aproximativ $0,67 \times c$;

atenuarea este, așa cum am văzut, foarte mică, de ordinul câtorva decibeli pe kilometru sau chiar câteva zecimi de decibel pe kilometru.

distorsiunile apar sub forma de *dispersie*, adică lățirea impulsurilor. Sunt cauzate de mai multe fenomene, și au ca și consecință limitarea practică a produsului dintre frecvența maximă ce se poate transmite și distanța dintre emițător și receptor. Acest produs se numește (impropriu) banda de trecere și se măsoară în megahertzi kilometru (MHz km). Valorile tipice, pentru o fibră multimod, sunt de ordinul a 500 MHz km.

zgomotul în transmisia prin fibră optică apare aproape exclusiv datorită fotodiodei receptoare (zgomot termic); acesta limitează inferior sensibilitatea receptorului și, la atenuare dată a fibrei, puterea emițătorului. Captarea de paraziti de-a lungul fibrei, și în particular diafonia, sunt neglijabile.

3.6.2.3. Multiplexarea în lungimea de undă

Considerând ca semnal intensitatea câmpului electric, observăm că prin fibra optică se transmite un semnal modulat în amplitudine. Frecvența purtătoare este frecvența undelor infraroșii. Semnalul modulator este rădăcina pătrată a puterii luminoase emise.

Ca urmare, este posibilă realizarea multiplexării în frecvență a mai multor semnale pe aceeași fibră optică. Emițătoarele sunt diode laser sau LED-uri de culori diferite. Receptoarele sunt dotate cu câte un filtru de culoare corespunzătoare plasat în fața elementului fotosensibil. Această metodă de multiplexare se numește *multiplexare în lungimea de undă* (engl. *wavelength division multiplexing* — WDM). Subliniem că diferența între multiplexarea în lungime de undă și multiplexarea în frecvență este doar de terminologie, nu una principală. Diferența provine doar din faptul că, în cazul transmisiei optice, în lipsa mijloacele de-a analiza direct semnalul electromagnetic (asupra căruia operează multiplexarea în frecvență), analizăm doar puterea semnalului electromagnetic.

Este posibilă și transmisia duplex pe o singură fibră optică. Pentru aceasta se realizează o construcție cu oglinzi semitransparente care permite ca raza de lumină emisă să pătrundă în fibră, iar raza de lumină ce iese din fibră să ajungă pe elementul receptor. Pentru a preveni diafonia între cele două senzuri de propagare, este necesar ca reflexiile pe capetele fibrei să fie extrem de reduse sau să se aplice o multiplexare în lungimea de undă între cele două senzuri.

3.6.3. Considerente practice

Realizând o transmisie ghidată prin cablu, fibrele optice concurează direct cu perechile de conductoare. Fibrele optice au câteva avantaje: sunt izolatoare din punct de vedere electric, sunt foarte puțin sensibile la zgomet, este dificil de interceptat comunicația prin ele (fără a le tăia este aproape imposibil de interceptat semnalul, iar tăierea fibrei poate fi ușor detectată), au atenuare mică și, în sfârșit, sunt mult mai ușoare (conțin mult mai puțin material) decât perechile de conductoare.

Toate aceste avantaje fac fibrele optice să fie extrem de atractive pentru comunicația pe distanțe mari, precum și pentru echipamente ce lucrează în condiții mai speciale, de exemplu la tensiuni electrice mari sau în medii cu radiații electromagnetice puternice.

Principalele dificultăți la utilizarea fibrelor optice sunt legate de cablare.

Deși puterea luminii transportate prin fibra optică este foarte mică, secțiunea extrem de mică a fibrei face ca densitatea de putere să fie suficient de mare pentru a fi periculoasă. Riscul principal este ca, în cazul în care lumina de la emițătorul optic pătrunde în ochi, să producă leziuni ireparabile ale retinei. Riscul de accident este mărit prin faptul că lumina nu este vizibilă. Ca măsură de protecție, se pot utiliza ochelari speciali prevăzuți cu filtre care

lasă să treacă lumina vizibilă, dar blochează infraroșiiile transmise prin fibre.

Lipirea fibrelor sau montarea conectorilor pe fibre necesită echipamente scumpe (zeci de mii de dolari pentru un dispozitiv de lipire și în jur de o mie de dolari pentru setul de unelte necesare montării conectorilor) și personal calificat. Din acest motiv, se comercializează cabluri, de diferite lungimi, cu conectori gata atașați.

Un fir de praf ajuns pe capătul unei fibre optice obstrucționează serios trecerea luminii. De aceea, conectorii necuplate se acoperă cu capace protectoare.

Capitolul 4

Nivelul legăturii de date

Nivelul legăturii de date are ca rol realizarea unei comunicații stabile între calculatoare sau echipamente între care există o legătură directă la nivel fizic (există deci un mediu de comunicație între ele).

În general, legătura de date oferă servicii de transport de pachete.

Nivelul fizic oferă servicii de transport de pachete, însă aceste servicii suferă de următoarele lipsuri:

- Pachetele pot fi alterate sau chiar distruse complet din cauza zgomotului.
- Dacă un același mediu de transmisie este utilizat de mai multe emițătoare (ceea ce se întâmplă adesea la transmisia prin unde radio, dar uneori și la transmisia prin perechi de conductoare) și mai multe dintre aceste emițătoare transmit simultan, pachetele transmise se alterează reciproc.
- Dacă destinația nu poate prelucra datele în ritmul în care sunt transmise de către emițător, o parte din date se vor pierde.
- Construcția legăturii fizice este scumpă; mai mult, există un cost independent de capacitate. Ca urmare, este de dorit să putem construim mai multe legături logice, care să transmită fluxuri independente de pachete, partajând aceeași legătură fizică.

Ca urmare, nivelul legăturii de date are sarcina de-a realiza următoarele:

- *detectarea sau corectarea erorilor* de transmisie;
- *controlul accesului la mediu* în cazul în care există mai multe emițătoare ce partajează același mediu de transmisie;
- *retransmiterea pachetelor pierdute* din cauza erorilor de transmisie, a ciocnirilor între pachete transmise de mai multe emițătoare simultan sau a incapacității destinației de-a le prelua la timp;

- *controlul fluxului* de date, adică frânarea emițătorului în cazul în care destinația nu este capabilă să proceseze suficient de repede informația primită;
- *multiplexarea* mai multor legături logice prin aceeași legătură fizică.

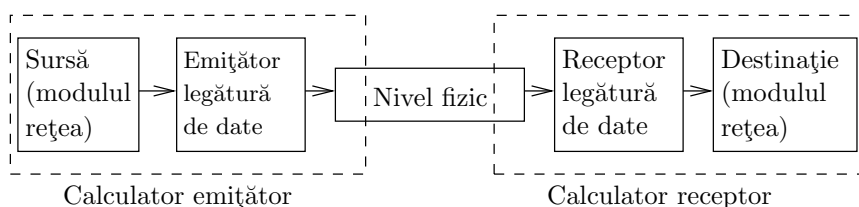


Figura 4.1: Alcătuirea nivelului legăturii de date și locul său între nivelele rețelei.

Constructiv, nivelul legăturii de date este un modul interpus între nivelul superior (în mod normal, nivelul rețea) și nivelul fizic (fig. 4.1). Pentru realizarea funcțiilor lor, modulele nivelului legăturii de date ale dispozitivelor ce comunică își transmit unul altuia, utilizând serviciile nivelului fizic, două tipuri de informații:

- *datele utile*, ce trebuie transmise de către nivelul legăturii de date în folosul nivelelor superioare;
- *informații de control*, pentru uzul strict al nivelului legăturii de date.

Informațiile de control sunt transmise fie împreună cu datele utile, în același pachet transmis prin nivelul fizic, fie separat, în pachete de sine stătătoare. În primul caz, informațiile de control sunt plasate fie în fața datelor utile, sub forma unui *antet*, fie după acestea. În cazul transmiterii datelor de control într-un pachet separat, un astfel de pachet se numește *pachet de control*.

4.1. Detectarea și corectarea erorilor

În vederea detectării sau, după caz, corectării erorilor, emițătorul de la nivelul legăturii de date adaugă, la fiecare pachet generat de nivelul superior, un număr de *biți de control*. Biții de control sunt calculați conform unui mecanism de codificare pentru canale cu perturbații (vezi § 2.4). Biții de control sunt adăugați, de regulă, la finalul pachetului.

Receptorul recalculează biții de control conform conținutului pachetului recepționat și-i compară cu cei de la finalul pachetului recepționat. În caz de nepotrivire, receptorul deduce că s-a produs o eroare de transmisie. În cazul utilizării unui cod corector de erori, receptorul reconstituie conținutul

cel mai probabil al pachetului original. În cazul unui cod detector de erori, pachetul nu poate fi recuperat; în acest caz, eventuala retransmitere a datelor cade în sarcina unui mecanism de tipul celui ce va fi studiat în § 4.3.

4.2. Controlul accesului la mediu

Problema controlului accesului la mediu se pune în situația în care pe un același mediu fizic acționează mai multe emițătoare, a căror emisie simultană interferează în așa fel încât un receptor nu poate recepționa corect oricare dintre transmisii. În aceste condiții, problema accesului la mediu constă în a elabora un protocol care să evite transmisia simultană.

În practică, problema accesului la mediu apare în următoarele ipostaze:

- la transmisia *semi-duplex*, adică în cazul comunicației bidirecționale, între două entități, utilizând același mediu fizic pentru ambele sensuri.
- la comunicația prin unde radio, dacă există mai multe stații care emit pe aceeași lungime de undă. În general, emisia unei stații este recepționată de toate stațiile pe o anumită rază. Este cazul aproape tuturor rețelelor fără fir: IEEE 802.11 (*wireless Ethernet*), Bluetooth, GSM, etc.
- dacă stațiile sunt conectate „tip magistrală“, adică mediul de comunicație — în general o pereche de conductoare — trece pe la toate stațiile. Este cazul rețelelor *Ethernet* mai vechi.

Există două strategii de control al accesului la mediu:

- *asigurarea unui interval exclusiv de emisie*, pe rând, pentru fiecare stație;
- *acceptarea posibilității coliziunilor și retransmisia pachetelor distruse* în coliziuni.

Asigurarea unui interval exclusiv de emisie permite garantarea, pentru fiecare stație, a unui debit minim cu care poate emite și a unui interval maxim de așteptare din momentul în care are ceva de transmis și până la intrarea în emisie; metoda cu coliziuni și retransmiteri este nedeterministă și ca atare asigură un anumit debit și un anumit timp de așteptare doar cu o anumită probabilitate strict mai mică decât unu. În schimb, în sistemele ce asigură un interval exclusiv de emisie, intrarea și ieșirea unei stații din rețea, precum și revenirea după o pană a unei stații, sunt complicate. În cazul asigurării unui interval exclusiv de emisie, o parte din capacitatea de transmisie a mediului este consumată de mesajele de sincronizare necesare stabilirii intervalelor fiecărei stații; în cazul acceptării coliziunilor, o parte din capacitate este pierdută datorită pachetelor distruse în coliziuni.

În general, asigurarea unui interval exclusiv de emisie este favorabilă în sistemele în timp real, cum ar fi rețelele utilizate pentru automatizări industriale transmisie audio-video. Detectarea coliziunilor și retransmiterea pachetelor distruse în coliziuni este favorabilă în sistemele interactive, cum ar fi rețelele „obișnuite“ de calculatoare.

Aproape în orice sistem în care mai multe dispozitive sunt conectate la același mediu fizic este necesar ca fiecare dispozitiv să aibă un identificator unic. Acest identificator se numește *adresă fizică* sau *adresă MAC* (de la *Media Access Control — controlul accesului la mediu*) sau, dacă nu e pericol de confuzie, *adresă*. Alocarea adresei fizice se face în mod normal prin mecanisme exterioare rețelei, adică adresele sunt alocate fie manual, de către administratorul rețelei, fie în cadrul procesului de fabricație al dispozitivului conectat la rețea.

4.2.1. Protocoale bazate pe asigurarea unui interval exclusiv de emisie

Cea mai simplă metodă din această categorie este să existe o stație desemnată ca *arbitru*, care să anunțe de fiecare dată ce stație primește dreptul de emisie. Anunțul se face printr-un pachet emis de arbitru și conținând adresa fizică a stației ce poate emite. Stația anunțată de arbitru are la dispoziție un interval de timp în care poate să emită ceea ce are de transmis. Dacă stația nu are nimic de transmis, protocolul poate prevedea fie că stația nu emite nimic, fie că emite un pachet special. Încheierea perioadei alocate unei stații se poate face fie la expirarea unei durate de timp prestabilite, fie prin anunțul explicit al stației că a încheiat transmisia. După încheierea perioadei alocate unei stații, arbitrul anunță stația următoare.

Arbitrul trebuie să aibă lista tuturor stațiilor din rețea. Ieșirea unei stații se face simplu prin anunțarea arbitrului; ieșirea arbitrului nu este posibilă (decât eventual prin desemnarea unui alt arbitru). Intrarea unei stații noi necesită un mecanism special de anunțare a arbitrului. Un astfel de mecanism este în general bazat pe coliziuni și prevede ca arbitrul să întrebe, periodic, dacă există stații ce vor să intre în rețea. Dacă o stație, alta decât arbitrul, se defectează, stația fie este văzută ca o stație ce nu are nimic de transmis, fie este detectată de către arbitru că nu răspunde și este scoasă de pe lista stațiilor din rețea. Defectarea arbitrului duce la căderea întregii rețele.

Metoda cu arbitru este utilizată, de exemplu, în cadrul fiecărei celule GSM.

O altă metodă de control al accesului este metoda cu *jeton*. În cadrul acestei metode, în loc să existe un arbitru central care deține lista completă a stațiilor, lista este distribuită, fiecare stație cunoscând adresa stației următoare. În acest fel, în intervalul de emisie alocat, fiecare stație emite datele utile, după care anunță stația următoare. Metoda cu jeton a fost utilizată în rețelele IEEE 802.4.

4.2.2. Protocoale bazate pe coliziuni și retransmitere

Cel mai simplu mecanism bazat pe coliziuni și retransmitere presupune ca o stație ce are date de transmis să le transmită imediat. În cazul unei coliziuni, stația emițătoare va repeta ulterior pachetul, până la o transmitere cu succes.

Detectarea unei coliziuni, de către fiecare dintre stațiile emițătoare, se poate face prin două metode:

- *prin ascultarea mediului* pentru a detecta o eventuală transmisie simultană.

Datorită întârzierilor de propagare diferite, este posibil ca două pachete să fie în coliziune pentru o stație receptoare și să nu fie în coliziune pentru altă stație (fig 4.2). Din acest motiv, un emițător trebuie să considere coliziune orice situație în care detectează o transmisie prea apropiată în timp de o transmisie proprie. Din același motiv, întârzierea maximă datorată propagării în rețea trebuie limitată prin standard, ceea ce impune o limită asupra întinderii geografice a rețelei.

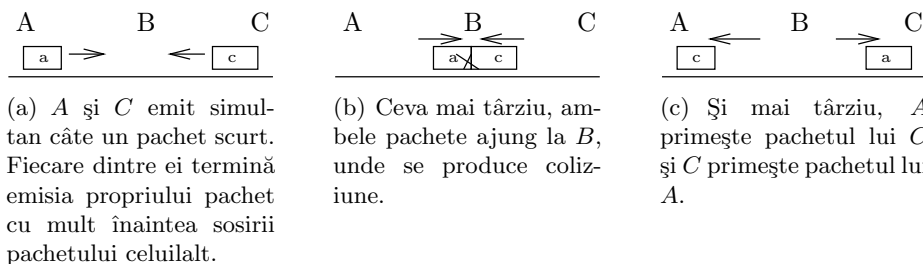


Figura 4.2: Două pachete emise simultan, în condițiile în care timpul de propagare este mai mare decât timpul de transfer. Coliziunea nu este detectată de nici unul dintre emițătoare, însă este detectată de o stație aflată la jumătatea distanței dintre acestea.

La legăturile radio poate să mai apară un fenomen, și anume, da-

torită atenuărilor diferite, este posibil ca pentru un receptor să apară coliziune între două pachete, în timp ce pentru alt receptor unul dintre pachete să fie atât de puternic atenuat încât să nu perturbe deloc recepția celui de-al doilea pachet (fig. 4.3). Din acest motiv, la transmisia radio este imposibil ca emițătorul să detecteze întotdeauna coliziunile proprii transmisii cu alte transmisii simultane.

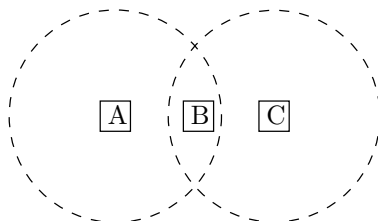


Figura 4.3: Este posibil ca două emițătoare radio, A și C , să fie situate prea departe pentru a-și recepționa una transmisia celeilalte, dar să existe o a treia stație, B , care să recepționeze transmisiile ambelor emițătoare (liniile punctate delimitează zona în care transmisia unei stații poate fi recepționată). În acest caz, A și C pot emite simultan fără a detecta coliziune, însă pentru B se produce coliziune între transmisiile lui A și C .

- *prin lipsa confirmării*, din partea receptorului, a primirii pachetului. Pentru aceasta, este necesară utilizarea unui cod detector de erori, cu ajutorul căruia receptorul să detecteze disturgerea pachetului în urma coliziunii. De asemenea, mai este necesar ca receptorul să confirme pachetele primite cu succes (astfel de mecanisme de confirmare vor fi studiate în § 4.3).

Repetarea unui pachet distrus de o coliziune se face după un interval de timp aleator. Dacă intervalul de timp până la retransmitere ar fi fix, două stații ce au emis simultan vor emite simultan și retransmiterile, ciocnindu-și la infinit pachetele. Mai mult, dacă apar frecvent coliziuni, este bine ca timpul până la următoarea retransmitere să fie mărit.

Acest protocol simplu de acces la mediu se numește *Aloha pur*.

O variantă îmbunătățită a protocolului *Aloha* este protocolul numit *Aloha cuantificat* (engl. *slotted Aloha*). În acest protocol, toate pachetele au aceeași lungime. Începerea transmisiei unui pachet nu poate avea loc oricând, ci doar la momente fixate, aflate la o durată de pachet unul de altul.

Alte îmbunătățiri ce pot fi aduse protocolului *Aloha pur* (nu însă și la *Aloha cuantificat*) sunt:

- *detectarea purtătoarei* (*CSMA* — *Carrier Sense Multiple Access*): o stație

care doreşte să emită ascultă mai întâi mediul; dacă detectează emisia altei staţii, amână emisia proprie până după finalul emisieii celeilalte staţii.

O primă posibilitate este ca staţia să înceapă emisia proprie imediat după terminarea emisieii celeilalte staţii. Dezavantajul este că, pe durata unui pachet lung, este probabil să se adune mai multe staţii care ar fi vrut să emită; ca urmare la finalul transmisiei acelui pachet toate staţiile vor emite simultan, rezultând coliziuni.

O soluţie mai bună este ca o staţie, care doreşte să emită şi constată că mediul este ocupat, să aştepte un interval de timp aleator, după care să verifice din nou dacă mediul este liber. Dacă mediul este liber, începe emisia proprie; dacă nu, aşteaptă un nou interval de timp aleator ş. a. m. d.

- *oprirea transmisiei la detectarea unei coliziuni* (numită, oarecum impropriu, *detectarea coliziunii* — *collision detection*, *CSMA/CD*): dacă o staţie, în timpul emisieii proprii, detectează o coliziune, abandonează datele rămase de transmis, transmite un semnal de o formă specială pentru a anunţa că pachetul este invalid şi apoi opreşte transmisia. În acest fel, se economiseşte timpul necesar transmisiei datelor rămase, transmisie oricum compromisă.

4.2.3. Protocoale mixte

Există şi protocoale de control al accesului la mediu care combină metode de asigurarea accesului exclusiv cu metode bazate pe coliziuni.

O posibilitate este să se negocieze, prin intermediul unor pachete de control de mici dimensiuni, accesul exclusiv la mediu în vederea transmiterii pachetelor de date, de dimensiuni mai mari.

O astfel de metodă este metoda *CSMA/CA* (*carrier sense multiple access with collision avoidance*, rom. *acces multiplu cu detectarea purtătoareii şi evitarea coliziunilor*), utilizată în reţelele IEEE 802.11. O staţie *A* care doreşte să transmită un pachet de date unei staţii *B* îi va trimite întâi un pachet de control, numit *RTS* (*request to send*, *cerere de transmisie*), în care specifică timpul necesar transmiterii pachetului (sau, echivalent, lungimea pachetului). *B* răspunde printr-un alt pachet de control, *CTS* (*clear to send*, *liber la transmisie*), destinat lui *A* dar recepţionat de toate staţiile, în care pune şi durata transmisiei copiată din pachetul *RTS*. La primirea pachetului *CTS*, staţia *A* transmite pachetul de date. O staţie care recepţionează un *CTS* adresat altei staţii nu are voie să transmită nici date, nici pachete de control, pe durata anunţată în *CTS* şi rezervată astfel destinatarului *CTS*-ului.

Această metodă este foarte favorabilă în rețele fără fir deoarece rezolvă și așa-numita *problema stației ascunse*: este posibil să existe trei stații, A , B și C , cu B situată geografic aproximativ la mijlocul distanței între A și C , cu distanța dintre A și C puțin peste raza de acțiune a transmisiei, astfel încât A nu recepționează transmisia lui C și nici reciproc, dar cu B suficient de aproape de A și de C astfel încât să poată comunica cu fiecare dintre ele. În această situație, dacă A și C emit simultan, din punctul de vedere al lui B se produce o coliziune, dar nici A nici C nu pot detecta acest lucru. Protocolul CSMA, descris în paragraful precedent, nu permite lui C să detecteze dacă A transmitea deja către B în momentul în care C dorește să transmită la rândul lui; ca urmare, CSMA se comportă exact ca Aloha pur. În protocolul CSMA/CA, în schimb, C recepționează CTS-ul adresat de B lui A și amână transmisia proprie.

Altă posibilitate de combinare a celor două strategii o constituie *protocolalele cu conflict limitat*. În cazul acestor protocoale, stațiile sunt împărțite în grupuri. Fiecărui grup i se alocă intervale exclusive de emisie (ca în cazul protocoalelor bazate pe intervale exclusive de emisie, dar cu diferența că fiecare interval se alocă unui întreg grup, nu unei stații). În cadrul fiecărui grup se aplică un protocol cu coliziuni și retransmitere. Împărțirea în grupuri poate fi făcută dinamic: dacă în cadrul unui grup apar frecvent coliziuni, grupul este scindat în două; dacă două grupuri au transmisii suficient de rare, pot fi recombinate într-unul singur.

4.3. Retransmiterea pachetelor pierdute

Dacă un pachet de date se pierde (de exemplu datorită unei erori de transmisie, eroare detectată dar nu și corectată de receptor), este necesară retransmiterea aceluia pachet.

Evident, emițătorul nu are cum să „ghicească” dacă un anumit pachet ajunge intact la destinație sau este pierdut; ca urmare, trebuie stabilită o comunicație înapoi dinspre receptor spre emițător. Principial, există două strategii:

- receptorul *confirmă* (engl. *acknowledge*, ACK) primirea corectă a pachetelor
- receptorul *infirmă* (engl. *negative acknowledge*, NAK) un pachet eronat.

Evident, confirmările sau infirmările sunt și ele pachete și deci sunt la rândul lor supuse eventualelor erori de transmisie.

Rolul unui protocol de retransmitere este să asigure că la destinație ajung *toate* pachetele emise, *în ordinea* în care sunt emise și *fără duplicate*.

Aceste trei condiții împreună formează dezideratul de *transmitere sigură* (engl. *reliable*).

4.3.1. Principiul confirmărilor pozitive și retransmiterilor

Ideea de bază a mecanismului de retransmitere este următoarea: la primirea cu succes a fiecărui pachet de date, receptorul trimite emițătorului câte un pachet cu rol de confirmare. Dacă emițătorul primește confirmarea, trece la următorul pachet. Dacă emițătorul nu primește confirmarea unui pachet în timpul dus-întors normal, repetă pachetul ce nu a fost confirmat (vezi figura 4.4).

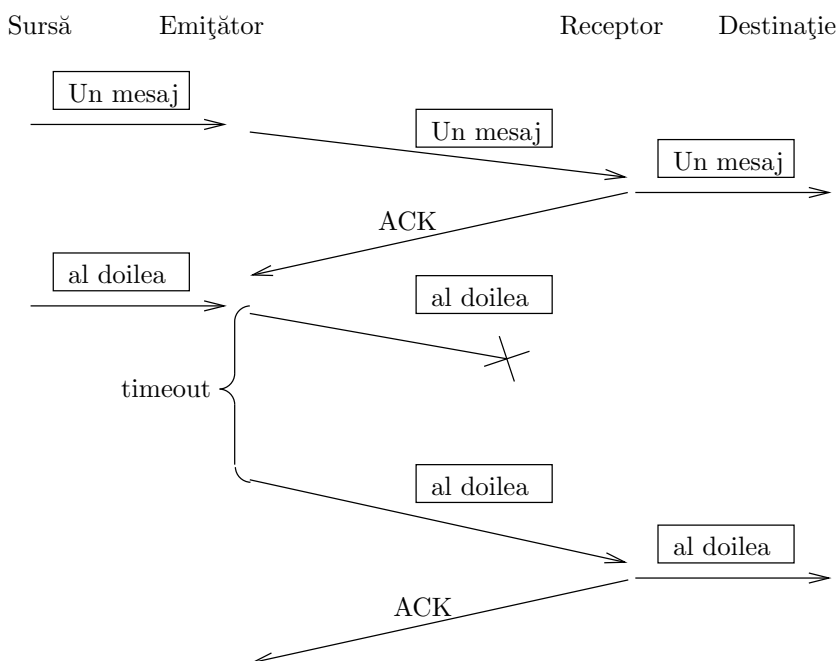


Figura 4.4: Retransmiterea pachetelor pierdute

Să examinăm acum protocolul din punctul de vedere al fiecărui participant (emițătorul și receptorul) și să nu uităm că fiecare are viziunea lui despre sistem, dată de acele informații care îi sunt accesibile.

Algoritmul emițătorului este următorul: pentru fiecare pachet al sursei, cât timp nu a primit confirmare de la receptor, trimite pachetul și așteaptă până când fie primește confirmarea, fie trece un timp egal cu durata dus-întors normală.

Algoritmul receptorului este următorul: pentru fiecare pachet primit de la emițător, trimite un pachet de confirmare spre emițător și livrează destinației pachetul primit.

Există o problemă cu algoritmul de mai sus. Întrebare pentru cititor: ce se întâmplă dacă un pachet ajunge la destinație însă confirmarea lui se pierde?

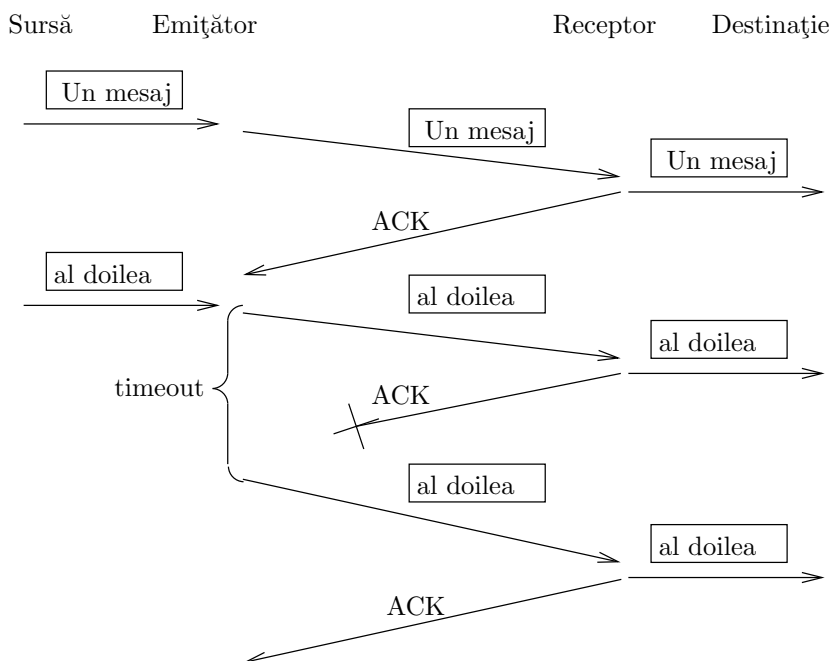


Figura 4.5: În lipsa unor măsuri adecvate, pierderea unei confirmări conduce la dublarea unui pachet.

Rezultatul se vede în figura 4.5 (pachetul ce conține textul „al doilea“ este livrat în dublu exemplar destinației). Să observăm (comparați și cu figura 4.4) că receptorul *nu are cum să distingă* între trimiterea următorului pachet de date și retransmiterea unui pachet datorită pierderii confirmării.

Pentru a obține un algoritm corect, trebuie să furnizăm receptorului suficientă informație pentru ca acesta să distingă între repetarea pachetului precedent și pachetul următor. O soluție este ca emițătorul să pună în fiecare pachet trimis numărul său de ordine în cadrul fluxului de date; acest număr de ordine se numește *numărul de secvență* al pachetului. În acest fel, receptorul va reține numărul ultimului pachet recepționat și va fi capabil să verifice dacă următorul pachet primit este repetarea acestuia sau este următorul pachet al

sursei.

Întrebare pentru cititor: dacă receptorul primește un duplicat al pachetului precedent, trebuie să-l confirme sau nu?

Să vedem raționamentul ce ne duce la răspuns: Emițătorul nu are de unde să știe dacă un pachet de date a ajuns sau nu la receptor. Dacă primește confirmarea unui pachet, poate fi sigur că pachetul confirmat a ajuns la receptor; dacă însă a trimis un pachet și nu a primit (încă) confirmarea acestuia, este posibil (conform informațiilor emițătorului) ca pachetul să fi ajuns (și eventual să se fi pierdut confirmarea) sau ca pachetul să nu fi ajuns deloc. El insistă în retransmiterea pachetului până la primirea confirmării. Prin urmare, receptorul trebuie să confirme *fiecare* pachet primit, chiar dacă este un duplicat al unui pachet anterior (vezi figura 4.6).

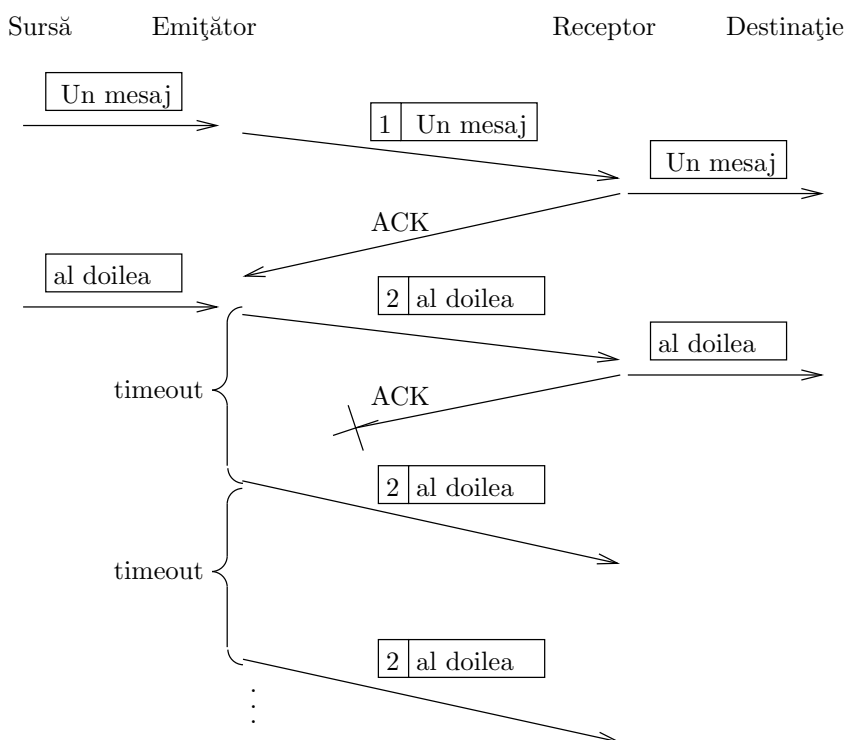


Figura 4.6: Neconfirmarea duplicatelor determină emițătorul să repete la infinit un pachet a cărui confirmare s-a pierdut.

Suntem acum în măsură să dăm algoritmi corecți pentru emițător și pentru receptor: funcționarea emițătorului este descrisă în algoritmul 4.1, iar cea a receptorului în algoritmul 4.2. Un exemplu de funcționare a acestora

este dat în figura 4.7

Algoritmul *Emitător_pas_cu_pas*

algoritmul:

$n:=0$

cât timp sursa mai are pachete de trimis execută

 fie d următorul pachet al sursei

$p:=(n, d)$

 execută

 trimite p

 recepţionează n' , fără a aştepta o durată mai mare de t

 cât timp $n' \neq n$ sau a expirat durata t

 sfârşit cât timp

sfârşit algoritm

Algoritmul 4.1: Algoritmul emiţătorului în protocolul simplu cu confirmări şi retransmiteri. Parametrul t este ales puţin mai mare decât durata dus-întors a nivelului fizic.

Algoritmul *Receptor_pas_cu_pas*

algoritmul:

$n:=0$

cât timp mai există pachete de primit execută

 recepţionează (n', d) de la nivelul fizic

 dacă $n' = n$ atunci

$n:=n + 1$

 livrează d destinaţiei

 sfârşit dacă

 trimite n' spre nivelul fizic

 sfârşit cât timp

sfârşit algoritm

Algoritmul 4.2: Algoritmul receptorului în protocolul simplu cu confirmări şi retransmiteri.

În aceşti algoritmi am presupus că numărul de secvenţă n poate creşte oricât de mult. În practică, n se reprezintă pe un număr fix de biţi şi ca urmare are o valoare maximă după care revine la 0. Vom vedea în § 4.3.3 în ce condiţii acest lucru afectează corectitudinea algoritmului.

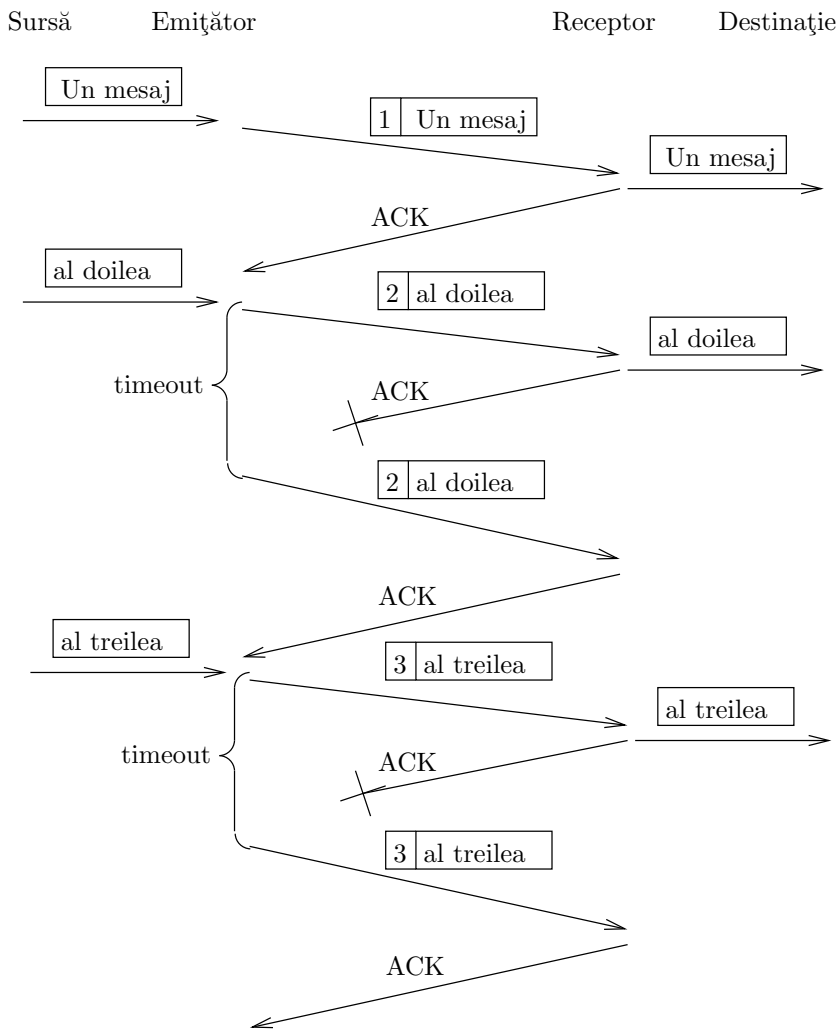


Figura 4.7: Funcționarea corectă a unui algoritm de retransmisie.

4.3.2. Trimiterea în avans a mai multor pachete

Deși corect, algoritmul pas cu pas din paragraful precedent este ineficient în situația în care timpul dus-întors între emițător și receptor este mult mai mare decât timpul necesar emiterii unui pachet. Acest lucru se întâmplă în cazul transmiterii unor pachete mici prin legături de debit mare și la distanță mare. În această situație, emițătorul emite repede un pachet, după care așteaptă (cea mai mare parte a timpului) propagarea pachetului spre destinatar și întoarcerea confirmării.

Pentru creșterea eficienței, ar fi util ca emițătorul să poată trimite unul după altul mai multe pachete, fără a aștepta confirmarea primului pentru a-l trimite pe următorul. Timpul maxim de așteptare pentru confirmarea unui pachet rămâne neschimbat, însă permitem trimiterea mai multor pachete în acest timp.

Trimiterea mai multor pachete în avans schimbă câteva lucruri față de cazul trimiterii unui singur pachet:

- Deoarece la un anumit moment pot exista mai multe pachete trimise de emițător și neconfirmate, pentru a putea corela confirmările cu pachetele de date, este necesar ca și confirmările să fie numerotate. Există două strategii posibile:
 - Fiecare pachet de date este confirmat printr-un pachet de confirmare distinct, conținând numărul de secvență al pachetului confirmat.
 - Un pachet de confirmare conține numărul de secvență până la care receptorul a primit toate pachetele. Cu alte cuvinte, un pachet de confirmare confirmă toate pachetele de date cu număr de secvență mai mic sau egal cu valoarea din pachetul de confirmare. Această strategie permite trimiterea unui singur pachet de confirmare pentru o serie de câteva pachete de date sosite imediat unul după altul.

De menționat că alegerea între aceste variante trebuie consemnată ca parte a protocolului stabilit între emițător și receptor.

- În urma pierderii unui pachet, receptorul poate ajunge în situația de-a primi un pachet fără să fi primit mai întâi pachetul anterior. În această situație, receptorul nu poate livra imediat acel pachet destinației. Există două acțiuni posibile pentru receptor:
 - Receptorul ignoră complet pachetul (în consecință, nici nu trimite confirmare).
 - Receptorul memorează pachetul, urmând să-l livreze destinației după primirea pachetului (sau pachetelor) anterioare.

Alegerea uneia sau a celeilalte variante privește doar receptorul, nu este parte a protocolului de comunicație. Mai mult, decizia de-a memora sau de-a ignora pachetul poate fi luată independent pentru fiecare pachet primit de receptor, în funcție de memoria disponibilă în acel moment.

O metodă utilizată frecvent este ca receptorul să fixeze o *fereastră de recepție*, adică un interval de numere de secvență acceptabile. Pentru aceasta, receptorul fixează la început un număr k . Dacă la un moment dat toate pachetele până la numărul de secvență n inclusiv au fost recepționate și livrate destinației, receptorul acceptă doar pachetele cu numere de secvență situate între $n + 1$ și $n + k$; acest interval constituie fereastra de recepție. Un pachet cu număr de secvență mai mic sau egal cu n este sigur un duplicat, un pachet între $n + 1$ și $n + k$ este memorat (dacă nu a fost primit deja) și confirmat, iar un pachet cu număr de secvență strict mai mare decât $n + k$ este ignorat. La primirea pachetului $n + 1$, fereastra este avansată până la primul număr de secvență ce nu a fost încă primit.

Este esențial ca un pachet, ce nu este nici memorat de receptor, nici transmis destinației, să nu fie confirmat. În cazul confirmării unui astfel de pachet, este probabil ca emițătorul să nu-l mai retransmită niciodată, ca urmare receptorul nu va putea să-l furnizeze destinației.

Emițătorul trebuie să țină și el evidența unei *ferestre de emisie*, conținând pachete, primite de la sursă în vederea trimiterii spre receptor, dar încă neconfirmate de către receptor. Notând cu n primul număr de secvență neconfirmat și cu k dimensiunea ferestrei emițătorului, fereastra emițătorului poate conține pachetele cu numere de la n la $n + k - 1$. Emițătorul trimite, în mod repetat, pachetele din fereastră, până ce primește confirmări pentru ele. În momentul în care pachetul n este confirmat, fereastra emițătorului avansează până la următorul pachet neconfirmat.

Din cauza ferestrelor emițătorului și receptorului, protocolul acesta se numește *protocolul ferestrei glisante*. Dacă fereastra emițătorului este de dimensiune 1, protocolul ferestrei glisante funcționează exact ca și protocolul pas cu pas din paragraful precedent.

Funcționarea protocolului ferestrei glisante, în diverse variante, este ilustrată în figurile 4.8–4.10.

4.3.3. Spațiul numerelor de confirmare

Evident, în practică, numerele de secvență nu pot crește la infinit. În general, numerele de secvență sunt reprezentate pe un număr fixat de biți și, ca urmare, au o valoare maximă după care se reiau de la 0.

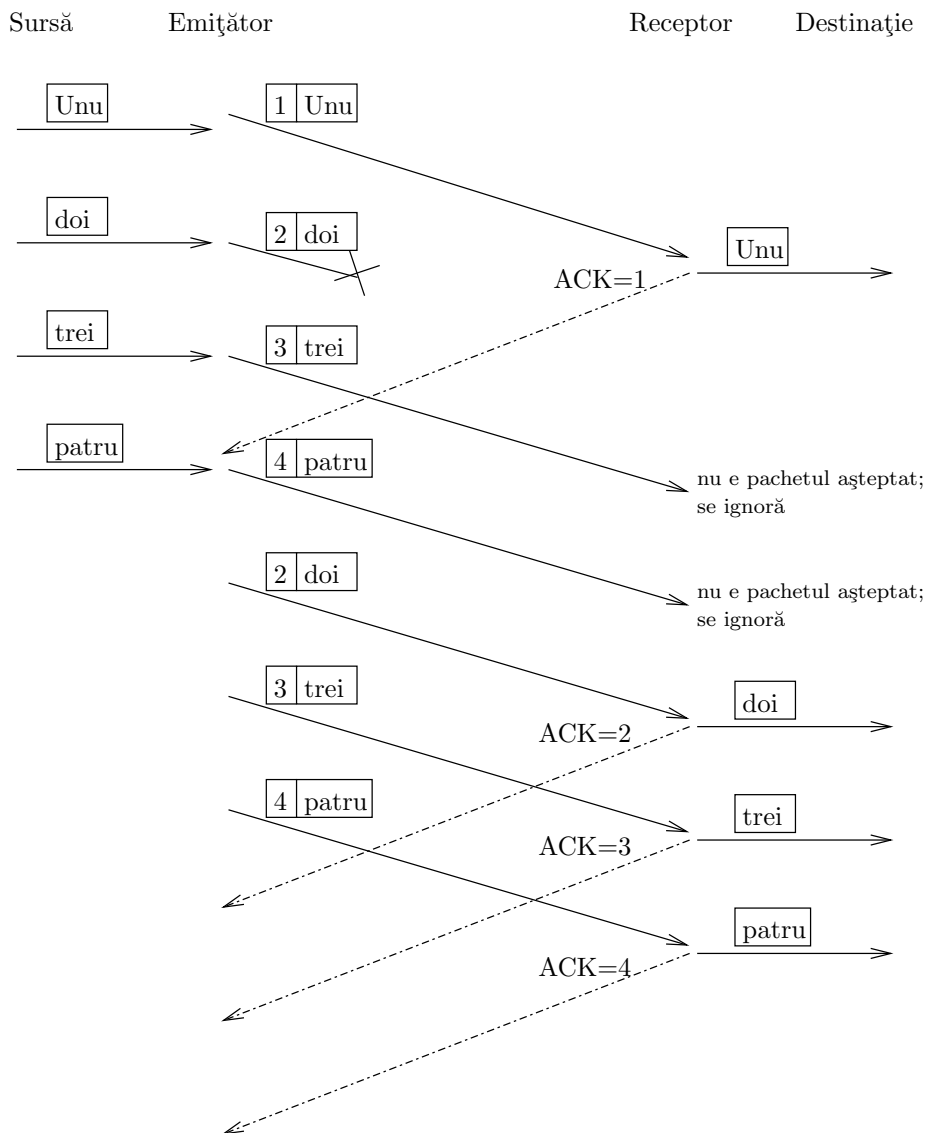


Figura 4.8: Funcționarea ferestrei glisante în cazul în care dimensiunea ferestrei de recepție este 1 și dimensiunea ferestrei de emisie este cel puțin 3.

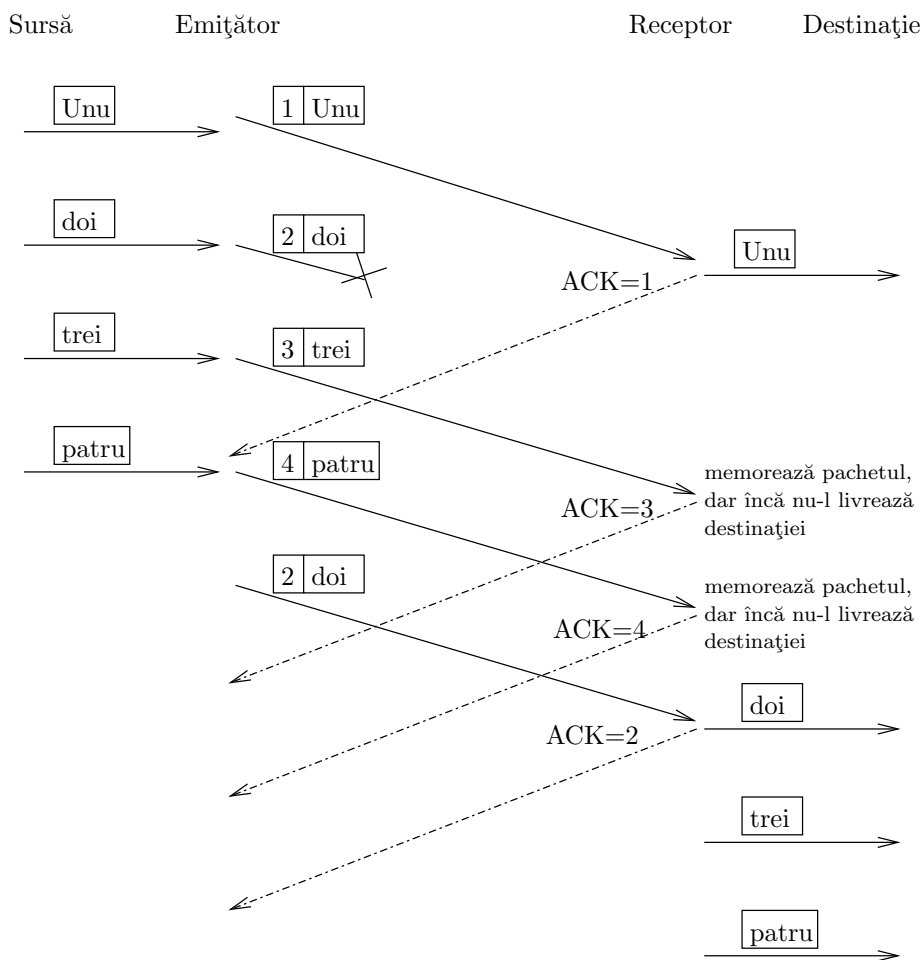


Figura 4.9: Funcționarea ferestrei glisante în cazul în care dimensiunea ferestrei de recepție este cel puțin 3 și protocolul prevede confirmarea individuală a pachetelor.

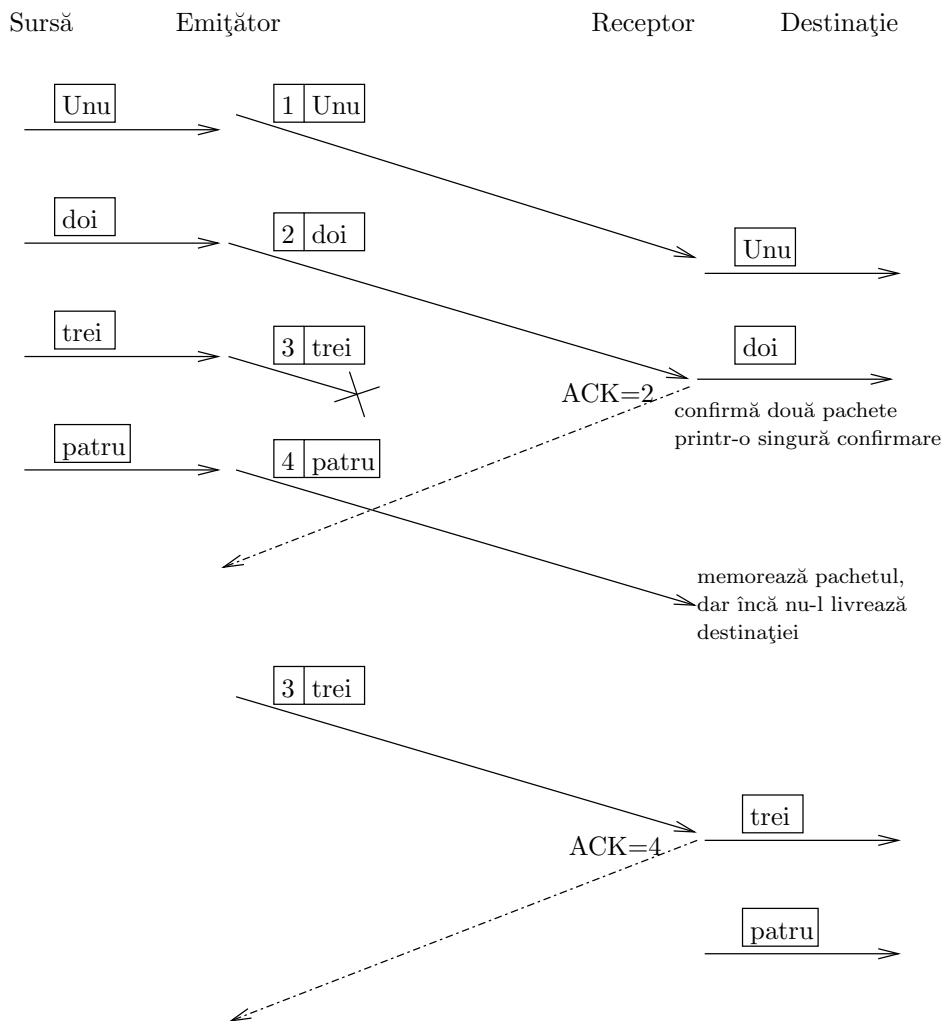


Figura 4.10: Funcționarea ferestrei glisante în cazul în care dimensiunea ferestrei de recepție este cel puțin 2 și protocolul prevede că un pachet de confirmare confirmă toate pachetele de date până la numărul de secvență conținut în pachet. De remarcă posibilitatea de optimizare a numărului de pachete de confirmare prin combinarea mai multor confirmări într-un singur pachet (confirmarea cu numărul 2).

Pentru a preciza lucrurile, vom numi *număr de secvență teoretic* numărul de secvență pe care l-ar avea un pachet dacă numerele de secvență nu ar fi limitate și *număr de secvență transmis* numărul transmis efectiv. Numărul de secvență transmis are ca valoare numărul de secvență teoretic modulo n , unde n este numărul de numere de secvență distincte disponibile.

Pentru ca mecanismele de confirmare și retransmitere, descrise în § 4.3.1 și § 4.3.2, să funcționeze corect, ele trebuie modificate în așa fel încât să compare efectiv numerele de secvență teoretice. Pentru aceasta, este necesar ca, în orice moment, atât receptorul cât și emițătorul să poată, pe baza informațiilor pe care le au, să deducă univoc numărul de secvență teoretic din numărul de secvență transmis.

Vom analiza în continuare ce relație trebuie să existe între numărul n de valori distincte pe care le poate lua numărul de secvență transmis și numărul de pachete trimise în avans pentru a nu exista ambiguități privitor la numărul de secvență teoretic al unui pachet de date sau de confirmare.

Propoziția 4.1 *Dacă dimensiunea ferestrei emițătorului este k și dacă pachetele se pot doar pierde, fără să-și poată schimba ordinea, atunci sunt necesare și suficiente $2k$ numere de secvență distincte pentru identificarea univocă a pachetelor.*

Demonstrație. Trebuie să arătăm trei lucruri: că există întotdeauna un interval de lungime $2k$, calculabil cu datele receptorului, în care se încadrează numărul de secvență al următorului pachet primit de receptor, că există un interval de lungime $2k$ în care se încadrează următoarea confirmare primită de emițător și, în final, că dacă utilizăm doar $2k - 1$ numere de secvență distincte putem da un exemplu în care apare o ambiguitate.

Presupunem că cel mai mare număr de secvență primit de către receptor este n . Deoarece emițătorul a trimis deja pachetul n , rezultă că pachetele până la $n - k$ inclusiv au fost deja confirmate și deci nu vor mai fi trimise. Pe de altă parte, deoarece pachetul $n + 1$ încă nu a ajuns la receptor, rezultă că acest pachet nu a fost confirmat și deci receptorul nu poate trimite pachete cu numere de secvență strict mai mari decât $n + k$. Ca urmare, dacă la un moment dat cel mai mare număr de secvență primit de receptor este n , următorul număr de secvență primit va fi în intervalul $[n - k - 1, n + k]$.

Să privim acum din perspectiva emițătorului. Fie n cel mai mare număr de secvență trimis. Deoarece n a fost deja trimis, rezultă că toate pachetele până la $n - k$ inclusiv au fost deja confirmate. În momentul primei trimiteri a pachetului $n - k$, pachetele până la $n - 2k$ inclusiv erau deja confirmate. Ca urmare, nici unul dintre pachetele cu numere de secvență mai mici sau egale cu $n - 2k$ nu a mai fost trimis ulterior primei trimiteri a

pachetului $n - k$. Ca urmare, după primirea confirmării pachetului $n - k$ nu mai pot sosi la emițător confirmări ale pachetelor cu numere mai mici sau egale cu $n - 2k$. Prin urmare, numărul următoarei confirmări se va încadra în intervalul $[n - 2k + 1, n]$.

Să arătăm acum că $2k$ numere de secvență distincte sunt într-adevăr necesare. Considerăm două scenarii:

1. Emițătorul transmite pachetele de la 1 la k , toate acestea ajung la receptor, dar toate confirmările se pierd. Emițătorul retransmite pachetul 1, care ajunge la receptor.
2. Emițătorul transmite pachetele de la 1 la k , acestea ajung la receptor, sunt confirmate și confirmările ajung înapoi la emițător. În continuare, emițătorul transmite pachetele de la $k + 1$ la $2k$, dar toate se pierd cu excepția pachetului $2k$.

Considerând doar informațiile receptorului, observăm că în ambele cazuri acesta primește pachetele de la 1 la k , după care, în primul caz primește pachetul 1, iar în al doilea caz primește pachetul $2k$. Pentru ca receptorul să poată distinge aceste pachete, este necesar ca acestea să aibă numere de secvență transmise distincte. Ca urmare, trebuie să existe cel puțin $2k$ valori distincte pentru numărul de secvență transmis. \diamond

4.4. Controlul fluxului

Prin *controlul fluxului* (engl. *flow control*) se înțelege procesul (și mecanismul ce-l realizează) prin care o sursă de date este frânată astfel încât să nu transmită date cu debit mai mare decât este capabilă destinația să le prelucreze.

În lipsa controlului fluxului, dacă sursa emite date mai rapid decât este capabilă destinația să le prelucreze, o parte din date se pierd. De remarcat că stocarea datelor într-o memorie tampon a destinației nu rezolvă problema, ci doar permite destinației să preia, pe durată scurtă de timp (până la umplerea memoriei tampon), un debit mai ridicat de date.

Vom presupune în cele ce urmează că transmisia între emițător și receptor este sigură (fără erori și fără pierderi, duplicări sau inversiuni de pachete).

Forma cea mai simplă de control al fluxului este standardizarea unui debit fix de transmitere a datelor și proiectarea tuturor componentelor sistemului de comunicație în așa fel încât să poată opera la acel debit. O astfel de abordare poate fi adecvată în sisteme în timp real, cum ar fi de exemplu telefonia digitală. În astfel de sisteme, capacitatea de prelucrare a informației,

necesară sistemului, poate fi anticipată, iar surplusul de capacitate nu poate fi valorificat.

Dacă soluția unui debit fix de transmisie nu este satisfăcătoare, este necesar un mecanism prin care receptorul să informeze emițătorul asupra posibilității sale de preluare a datelor. Pentru aceasta este necesar un al doilea canal de comunicație, înapoi, dinspre receptor spre emițător.

4.4.1. Cereri de suspendare și de continuare

Un mecanism primitiv de control al fluxului prevede ca receptorul să poată trimite emițătorului cereri de suspendare a transmisiei și cereri de continuare a transmisiei.

Astfel, receptorul este prevăzut cu o memorie tampon. Dacă memoria tampon a receptorului este aproape plină, receptorul trimite emițătorului un mesaj prin care cere acestuia să suspende transmisia de date. Ulterior, când destinația consumă datele din memoria tampon a receptorului, receptorul cere emițătorului să continue transmisia.

Acest mecanism este utilizat la transmisia prin linie serială, sub numele de *software flow control* sau de *xon/xoff*. Cererea de suspendare a transmisiei se face prin trimiterea unui caracter, numit uneori *xoff*, având codul ASCII 19. Reluarea transmisiei se cere prin transmiterea unui caracter, numit uneori *xon*, având codul 17. De la un terminal text, clasic, caracterul *xoff* se transmite tastând combinația *ctrl-S*, iar *xon* se transmite tastând *ctrl-Q*. Astfel, un utilizator lucrând la un terminal text poate tasta *ctrl-S* pentru a cere calculatorului oprirea trimiterii de date spre afișare și, după ce citește datele afișate, va tasta *ctrl-Q* pentru continuarea transmisiei. Evident, cu acest mecanism de control al fluxului, caracterele cu codurile 17 și 19 nu pot fi utilizate pentru a transmite informație utilă.

Același principiu, implementat puțin diferit, este mecanismul numit *hardware flow control*. În acest caz, semnalizarea de suspendare și reluare a transmisiei se face printr-o pereche de conductoare separată de cea utilizată pentru transmiterea datelor.

Deoarece din momentul în care receptorul cere suspendarea transmisiei și până în momentul în care receptorul nu mai primește pachete trece o anumită durată de timp — egală cu durata dus-întors pe legătură — este necesar ca receptorul să aibă o memorie tampon suficient de mare pentru primirea pachetelor trimise în acest interval de timp.

4.4.2. Mecanismul pas cu pas

Un alt mecanism de control al fluxului presupune ca receptorul să semnalizeze emițătorului când este pregătit să accepte următorul pachet. E-

emiţătorul trimite un singur pachet, apoi aşteaptă semnalizarea receptorului că este pregătit să primească următorul pachet, apoi trimite următorul pachet ş. a. m. d. Mecanismul este asemănător cu mecanismul de retransmitere a pachetelor pierdute (§ 4.3), însă cu diferenţa că emiţătorul aşteaptă primirea „confirmării“ fără a retransmite pachetul de date dacă această aşteptare depăşeşte o anumită durată.

Ca şi la mecanismul de retransmitere a pachetelor pierdute, trimiterea a câte unui singur pachet urmată de aşteptarea permisiunii de a-l trimite pe următorul conduce la ineficienţă dacă durata dus-întors este semnificativ mai mare decât durata de transfer a unui pachet. În acest caz, se poate stabili ca receptorul să comunice periodic emiţătorului numărul de pachete pentru care mai are spaţiu în memoria tampon. Emiţătorul poate trimite cel mult numărul de pachete anunţat de receptor înainte de-a primi un nou anunţ de disponibilitate de la acesta. Deoarece anunţul de disponibilitate al receptorului ajunge la emiţător cu o anumită întârziere, timp în care emiţătorul a putut trimite un număr de pachete, este necesar ca emiţătorul să scadă din disponibilitatea anunţată de receptor numărul de pachete trimise între timp. Pentru aceasta este necesar ca pachetele să fie numerotate şi anunţul de disponibilitate să conţină şi numărul de ordine al ultimului pachet de date primit. În acest fel, dacă emiţătorul primeşte un anunţ de disponibilitate prin care este informat că receptorul tocmai a primit pachetul n şi are memorie pentru încă k pachete, atunci emiţătorul poate trimite cel mult pachetul $n + k$ înainte de-a primi un nou anunţ de la receptor.

4.4.3. Mecanism combinat cu retransmiterea pachetelor pierdute

Să observăm acum că orice mecanism de retransmitere a pachetelor pierdute poate fi folosit, fără modificări, şi cu rolul de mecanism de control al fluxului. Într-adevăr, receptorul nu trebuie decât să ignore complet orice pachet pe care nu îl poate prelua (în particular, să nu-i confirme primirea). În acest fel, la umplerea memoriei receptorului, pachetele trimise în continuare de emiţător nu vor fi confirmate. În consecinţă, ele vor fi retransmise până când destinaţia va consuma o parte dintre datele sosite la receptor, receptorul va putea prelua noi pachete de la emiţător şi va confirma emiţătorului primirea acestor pachete. Mecanismul este însă destul de ineficient, deoarece emiţătorul repetă pachete care ajung corect la receptor.

Este posibilă combinarea controlului fluxului cu retransmiterea pachetelor pierdute, combinând în acelaşi pachet confirmarea unui pachet de date cu anunţul de disponibilitate şi utilizând acelaşi număr de secvenţă pen-

tru ambele mecanisme. Un exemplu clasic de astfel de mecanism combinat este protocolul TCP, descris pe larg în § 10.3.1.

4.5. Multiplexarea în timp

În general, prin multiplexare se înțelege un procedeu prin care printr-un același canal fizic de comunicație se stabilesc mai multe comunicații care decurg relativ independent una de alta. Serviciul oferit fiecărei comunicații este numit *canal logic*; fiecare comunicație ocupă deci câte un canal logic și toate canalele logice sunt construite pe același canal fizic.

În § 3.3.3 și § 3.6.2.3 am văzut mecanisme de multiplexare (în frecvență, respectiv în lungime de undă) construite la nivelul fizic. La nivelul legăturii de date se poate construi un al treilea mecanism de multiplexare: *multiplexarea în timp*.

Ideea multiplexării în timp este de-a transmite intercalat, prin canalul fizic, pe rând, pachete sau șiruri de biți aparținând fiecărui canal logic. Evident, intercalarea trebuie făcută în așa fel încât receptorul să poată separa datele corespunzătoare fiecărui canal logic. De asemenea, emițătorul trebuie să asigure o împărțire echitabilă a capacității canalului fizic între canalele logice.

Separarea datelor corespunzătoare canalelor logice se poate face prin două metode:

- Fiecare canal logic are asociat un identificator unic. Fiecare pachet are, în antet, identificatorul canalului logic căruia îi aparțin datele utile (fig. 4.11(b)).
- Se stabilește o ordine de succesiune între canalele logice. Prin canalul fizic se transmite, pe rând, câte un pachet aparținând fiecărui canal logic (fig. 4.11(c)). De notat că, dacă sursa unui canal logic nu transmite pachete o perioadă mai lungă de timp, trebuie ca emițătorul de la nivelul legăturii de date să trimită pachete vide în contul aceluși canal (pentru a permite celorlalte canale logice să transmită pachete fără a încurca evidențele receptorului).

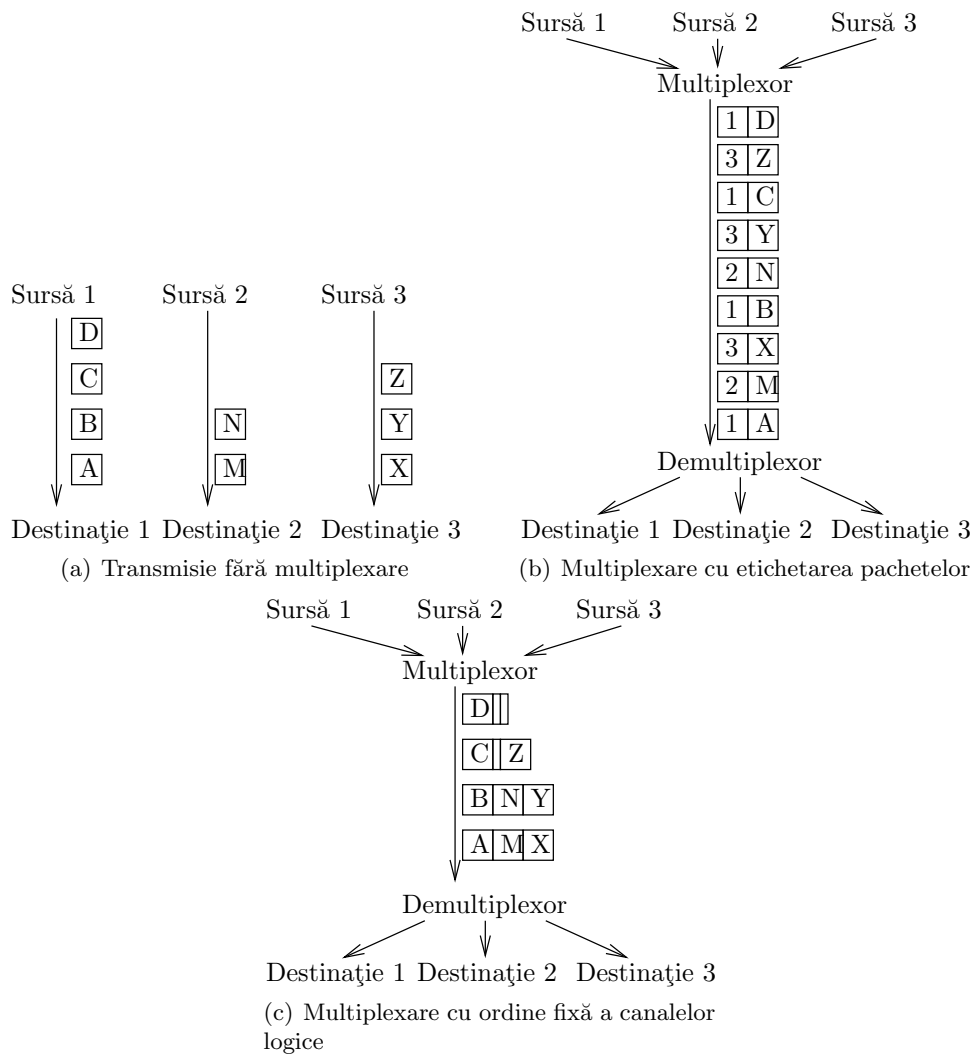


Figura 4.11: Funcţionarea mecanismelor de multiplexare în timp

Capitolul 5

Nivelul rețea și nivelul transport

Dacă niște dispozitive, relativ numeroase sau întinse pe distanțe mari, trebuie să poată comunica fiecare cu fiecare, este adesea prea costisitor să se construiască câte o legătură fizică între fiecare două dispozitive. Este necesar în acest caz să se poată stabili comunicații între dispozitive între care nu există o legătură fizică directă dar există legături *indirecte* prin intermediul unui șir de dispozitive legate fizic fiecare cu următorul.

O *rețea de comunicație* este un ansamblu de dispozitive care permit stabilirea de comunicații indirecte.

Într-o rețea de comunicație numim:

- *nod*: orice dispozitiv ce participă activ în rețea.
- *legătură directă*: orice legătură între noduri, utilizabilă de către nivelul rețea; două noduri între care există o legătură directă se numesc *vecini*.
- *nod final* sau *stație* (engl. *host*): un nod care este sursă sau destinație pentru date;
- *nod intermediar* sau *ruter* (engl. *router*): un nod ce poate fi tranzitat de trafic ce nu are ca sursă sau destinație acel nod; uneori este numit, în mod incorect, *server*.
- *adresă de rețea* sau, simplu, *adresă*: un identificator (un șir de simboluri) ce identifică unic un nod al rețelei. Fiecare nod terminal trebuie să aibă cel puțin o adresă; nodurile intermediare nu au întotdeauna adrese.
- *drum* sau *rută*: o secvență de noduri, fiecare vecin cu următorul, împreună cu legăturile directe dintre ele.

Notăm că în unele rețele există o distincție netă între nodurile finale și nodurile intermediare: de exemplu în rețeaua telefonică, aparatele telefonice

sunt noduri finale iar centralele telefonice sunt noduri intermediare. În alte rețele, unele sau toate nodurile sunt simultan noduri finale și noduri intermediare.

Unei rețele i se asociază un graf, construit astfel: fiecărui nod al rețelei i se asociază un vârf al grafului, iar fiecărei legături directe i se asociază o muchie (sau un arc, dacă legăturile sunt asimetrice). Rețeaua trebuie să fie astfel construită încât graful asociat ei să fie conex (respectiv tare conex), altfel, evident, vor exista perechi de noduri ce nu vor putea comunica.

Funcția principală a nodurilor rețelei este aceea de-a retransmite datele, asigurând continuitatea transportului lor de la nodul sursă la nodul destinație. Realizarea acestei funcții va fi studiată în § 5.1. Pentru retransmiterea datelor spre destinație, fiecare nod trebuie să decidă cărui vecin să retransmită datele; problema luării acestei decizii se numește *problema dirijării* (engl. *routing*) și va fi studiată în § 5.2. În final, în § 5.3 vom studia problemele ce apar atunci când solicitarea rețelei este ridicată (nu este neglijabilă față de capacitatea nodurilor și legăturilor utilizate).

5.1. Retransmiterea datelor de către nodurile intermediare

Vom studia în cele ce urmează, pe scurt, activitatea nodurilor într-o rețea. Problema determinării următorului nod de pe drumul spre o anumită destinație (problema dirijării) va fi studiată mai târziu, în § 5.2.

Constructiv, într-un nod al unei rețele trebuie să existe următoarele componente (vezi figura 5.1):

- *Adaptarea spre legătura fizică*, pentru fiecare legătură fizică ce pleacă din nod, este o componentă care realizează transmisia și recepția datelor prin acea legătură. Aceasta este formată din modulul nivelului legăturii de date și din modulul nivelului fizic.
- *Adaptarea spre aplicație*, pentru nodurile terminale, este o componentă ce realizează intermedierea între serviciile oferite direct de nivelul rețea și nevoile aplicațiilor ce se execută pe acel nod. Aceasta este, de principiu, modulul nivelului transport.
- *Modulul de rețea* este componenta care dirijează fluxul de date prin nod, fiind responsabil de alegerea vecinului căruia trebuie să-i fie transmise datele, precum și de transmiterea efectivă a acestora către modulul de adaptare spre mediul fizic (în nodurile intermediare) sau, respectiv, către modulul de adaptare spre aplicație (în nodul destinație).

Nivelul rețea este ansamblul modulelor de rețea ale nodurilor rețelei.

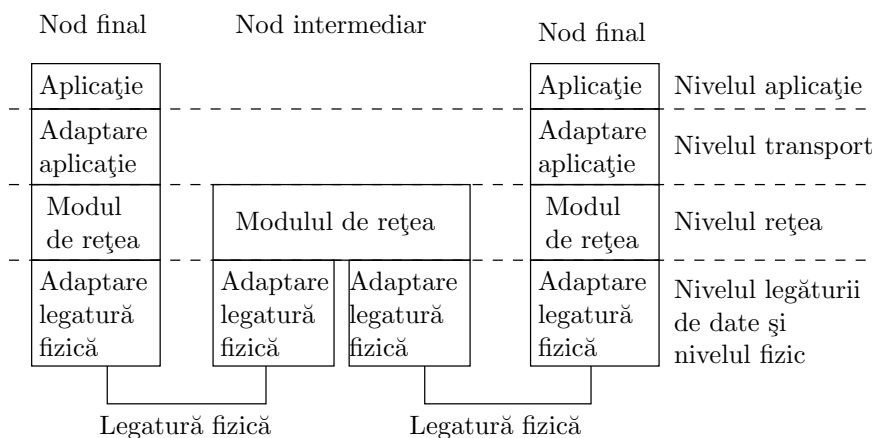


Figura 5.1: Modulele nodurilor unei rețele. Sunt figurate doar modulele din trei noduri, de-a lungul traseului datelor între două aplicații.

Un ansamblu de calculatoare constituie o rețea dacă și numai dacă graful nodurilor și legăturilor directe este conex (tare conex, dacă legăturile pot fi asimetrice), și în plus modulele de rețea ale tuturor nodurilor pot comunica printr-un protocol comun.

În lipsa unui protocol comun între modulele de rețea nu se poate stabili comunicația între oricare două noduri finale într-un mod uniform, fără ca aplicația client să trebuiască să aibe cunoștințe despre nodurile intermediare. Din acest punct de vedere spunem că nivelul rețea, și în special protocolul utilizat de nivelul rețea, este liantul întregii rețele.

După serviciul oferit, o rețea poate fi cu *datagrame* (numite uneori *pachete*) sau cu *conexiune*:

- *datagrame*: Într-o rețea ce oferă serviciu tip datagrame, aplicația sursă crează o datagramă conținând datele de transmis și o paseze modulului rețea, specificând totodată adresa nodului destinație. Datagrama este transmisă din aproape în aproape până la nodul destinație, unde este pasată aplicației (vezi § 5.1.1). De remarcat că două datagrame distincte generate de același nod sursă și adresate aceluiași nod destinație sunt prelucrate, de către rețea, complet independent una de alta. Funcționarea rețelelor ce oferă servicii de tip datagrame este similară sistemului de poștă (poșta obișnuită).
- *conexiune*: Într-o rețea ce oferă serviciu de tip conexiune, o aplicație ce dorește să comunice cu o aplicație dintr-un alt nod începe prin a so-

licita modulului rețea deschiderea unei conexiuni către acel nod. Nivelul rețea informează nodul destinație despre cererea de deschidere a conexiunii și, dacă aplicația destinație acceptă, conexiunea este deschisă și nodul inițiator este informat de acest lucru. După deschiderea conexiunii, unul sau ambele noduri (în funcție de tipul conexiunii deschise, unidirecțională sau bidirecțională) poate transmite celuilalt pachete de date prin conexiunea deschisă. La terminarea comunicației, una dintre aplicații solicită nivelului rețea închiderea conexiunii. Ca efect, nivelul rețea informează nodul partener cu privire la închiderea conexiunii și eliberează resursele alocate conexiunii. Funcționarea rețelelor ce oferă serviciu de tip conexiune este descrisă în § 5.1.2. Un model tipic de rețea ce oferă conexiuni este sistemul telefonic.

5.1.1. Retransmiterea în rețele bazate pe datagrame

Vom studia în cele ce urmează activitatea unui nod într-o rețea ce oferă transport de datagrame.

O datagramă este format dintr-un *antet* și *datele utile*. Antetul cuprinde mai multe informații utile în vederea dirijării. Informația ce nu poate lipsi din antet este adresa destinatarului.

Modulul de rețea al nodului primește o datagramă fie de la nivelul superior (dinspre aplicație), fie de la nivelul inferior (de pe o legătură directă). Modulul de rețea memorează temporar datagrama primită. În continuare, el are de făcut două lucruri:

- să determine dacă datagrama este destinată nodului curent, iar dacă nu, care este următorul vecin direct pe ruta spre destinație;
- să inițieze efectiv transmisia datagramei.

Dacă legătura prin care trebuie trimisă datagrama este încă ocupată cu transmiterea unei datagrame anterioare, datagrama trebuie pus într-o coadă de așteptare. Se poate întâmpla ca memoria utilizabilă pentru coada de așteptare să se epuizeze, caz în care este necesară sacrificarea unora dintre datagramele din coadă sau refuzul primirii unor datagrame noi. Detalii cu privire la operarea rețelei în acest caz sunt date în § 5.3.

5.1.2. Retransmiterea în rețele bazate pe conexiuni

Într-o rețea bazată pe conexiuni, activitatea este împărțită în două sarcini: stabilirea și desfacerea conexiunilor, pe de o parte, și transmiterea efectivă a datelor pe conexiuni, pe de altă parte.

Deschiderea conexiunii începe print trimiterea, de către nodul terminal ce dorește inițierea conexiunii, a unei cereri către primul nod intermediar.

Fiecare nod intermediar, pe rând, determină nodul următor prin care trebuie să treacă conexiunea și-i trimite mai departe cererea de deschidere a conexiunii. Determinarea nodului următor se face la fel ca și în cazul rețelelor bazate pe datagrame (vezi § 5.2). După determinarea nodului vecin, nodul curent memorează în tabela conexiunilor deschise nodul astfel ales. Conexiunea este deschisă în momentul în care cererea de deschidere a conexiunii ajunge și este acceptată de nodul destinație. Odată conexiunea deschisă, drumul corespunzător între cele două noduri finale este fixat pe toată durata conexiunii.

În faza de comunicare propriu-zisă, există două metode prin care se poate realiza tranzitarea traficului prin fiecare nod intermediar:

- *Comutare de circuite fizice:* În acest caz, mediul prin care intră datele în nod este conectat fizic (de exemplu, cu ajutorul unui întrerupător electric) la mediul prin care trebuie trimise mai departe datele (vezi fig. 5.2). Aceasta metodă, amintită aici doar pentru completudine, nu se mai utilizează în prezent (a fost utilizată în rețelele telefonice vechi, analogice).

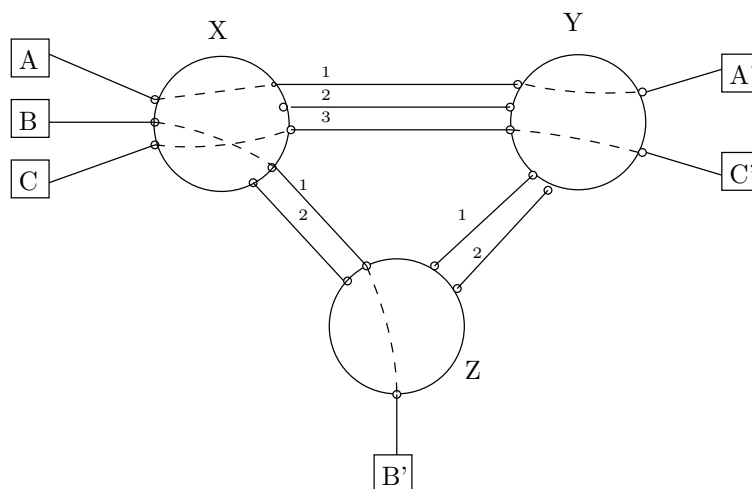


Figura 5.2: O rețea cu comutare de circuite fizice. Cercurile mari reprezintă nodurile intermediare, iar liniile punctate reprezintă interconectările mediilor fizice. De remarcat necesitatea mai multor legături fizice între câte două noduri.

- *Comutare de circuite virtuale:* Fiecare pachet ce sosește printr-o legătură de date este memorat temporar și apoi retransmis prin legătura spre următorul nod.

Să remarcăm că, în ambele cazuri, o legătură între două noduri este

asociată unei singure conexiuni; două conexiuni nu pot utiliza (direct) o aceeași legătură. (Din acest motiv, în sistemul telefonic vechi, între două centrale telefonice erau duse în paralel mai multe perechi de conductoare, numărul de convorbiri simultane utilizând o rută trecând prin cele două centrale fiind limitat la numărul de perechi de conductoare.) Deoarece, în special pe distanțe mari, mediul fizic este scump, se utilizează mecanisme de multiplexare. Acestea pot lucra fie la nivel fizic (multiplexare în frecvență — § 3.3.3 — sau în lungimea de undă — § 3.6.2.3), fie la nivelul legăturii de date (multiplexare în timp, § 4.5). Mai remarcăm că multiplexarea în timp poate fi utilizată doar în sistemele cu comutare de circuite virtuale.

La utilizarea comutării de circuite virtuale împreună cu multiplexarea în timp, un nod care a primit un pachet trebuie să-l memoreze până când îi vine rândul să fie transmis mai departe prin legătura de ieșire, adică până când, în legătura fizică de ieșire, vine rândul la transmisie canalului logic prin care trebuie trimis pachetul.

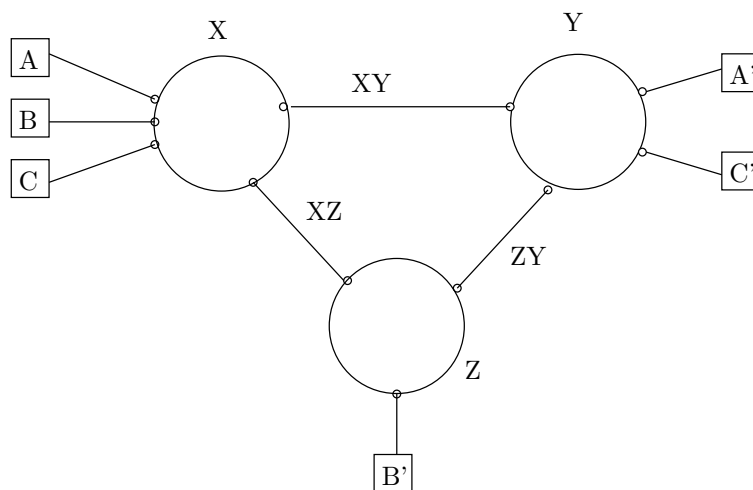


Figura 5.3: O rețea cu comutare de circuite virtuale. Desfășurarea în timp a recepției și transmitere mai departe a pachetelor, pentru nodul X , este prezentată în figura 5.4. Legăturile directe între nodurile intermediare utilizează multiplexare în timp.

Închiderea conexiunii se face prin transmiterea unui pachet special de cerere a închiderii conexiunii. Acest pachet urmează aceeași rută ca și pachetele normale de date. Fiecare nod de pe traseu, la primirea pachetului, șterge conexiunea respectivă din tabelul conexiunilor și eliberează resursele alocate.

Comutarea de circuite virtuale seamănă la prima vedere cu transmisia

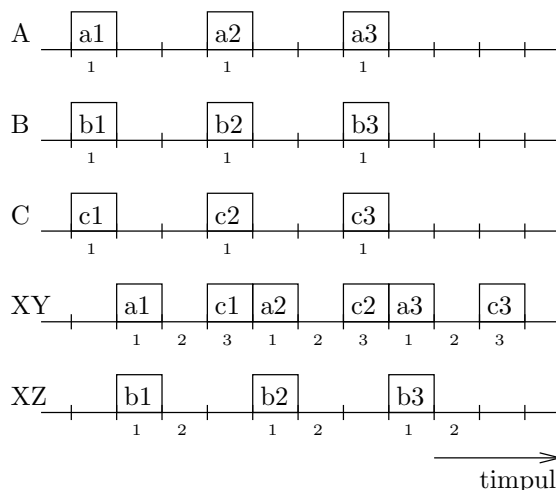


Figura 5.4: Desfășurarea în timp a recepției și a transmiterii mai departe a pachetelor, pentru nodul X din rețeaua din figura 5.3. XY și XZ desemnează legăturile fizice între nodurile X și Y , respectiv X și Z . Numerele de sub axe marchează perioadele de timp alocate canalelor logice corespunzătoare. Legăturile dintre canalele virtuale de intrare și de ieșire sunt identice cu legăturile fizice din figura 5.2

de datagrame. Diferența vine din felul în care un nod, care primește un pachet și trebuie să-l trimită mai departe, ia decizia privind legătura prin care să-l trimită. În cazul comutării de circuite virtuale, decizia este luată în funcție de circuitul virtual căruia îi aparține pachetul, informație dedusă din legătura de date prin care a intrat pachetul. Decizia se ia pe baza tabelii de circuite și este identică pentru toate pachetele aparținând aceluiași circuit. O urmare a acestui fapt este că defectarea oricărui nod sau oricărei legături de-a lungul unei conexiuni duce la închiderea forțată a conexiunii. În cazul rețelei bazate pe datagrame, decizia de dirijare se ia în funcție de adresa destinație, conținută în datagramă. Două datagrame între aceleași două stații pot fi dirijate pe rute diferite.

5.2. Algoritmi de dirijare

Ne vom ocupa în continuare de modul în care un nod decide spre care dintre vecini să trimită o datagramă (în cazul rețelelor bazate pe datagrame), respectiv spre care dintre vecini să transmită cererea de inițiere a unei conexiuni (în cazul rețelelor bazate pe conexiuni). Problema determinării acestui nod vecin se numește *problema dirijării*.

Rezolvarea problemei dirijării se bazează pe determinarea unui drum de cost minim, de la nodul sursă la nodul destinație al datagramei sau al conexiunii, în graful asociat rețelei de calculatoare.

Graful asociat rețelei de calculatoare este un graf ce are câte un vârf asociat fiecărui nod al rețelei și câte o muchie asociată fiecărei legături directe între două noduri. Fiecărei muchii i se asociază un cost, existând următoarele posibilități pentru definirea costului:

- toate costurile egale;
- în funcție de lungimea fizică a legăturii (cu cât o legătură este mai lungă, cu atât costul asociat este mai mare);
- în funcție de capacitatea legăturii;
- în funcție de încărcarea legăturii.

Remintim, din teoria grafelor, o proprietate importantă a drumurilor de cost minim: Dacă $v_0, v_1, \dots, v_{j-1}, v_j, v_{j+1}, \dots, v_k$ este un drum de cost minim de la v_0 la v_k , atunci $v_0, v_1, \dots, v_{j-1}, v_j$ este un drum de cost minim de la v_0 la v_j și v_j, v_{j+1}, \dots, v_k este un drum de cost minim de la v_j la v_k . De asemenea, dacă există cel puțin un drum de cost minim de la v_0 la v_k ce trece prin v_j , dacă $v_0, v_1, \dots, v_{j-1}, v_j$ este un drum de cost minim de la v_0 la v_j și v_j, v_{j+1}, \dots, v_k este un drum de cost minim de la v_j la v_k , atunci $v_0, v_1, \dots, v_{j-1}, v_j, v_{j+1}, \dots, v_k$ este drum de cost minim de la v_0 la v_k . Această proprietate stă la baza algoritmilor de determinare a drumului minim într-un graf.

În consecință, dacă un pachet de la un nod v_0 spre un nod v_k ajunge la un nod v_j , nodul următor, după v_j , de pe drumul de cost minim de la v_0 spre v_k depinde doar de v_k , nu și de v_0 . Ca urmare, pentru a efectua retransmiterea datelor, fiecare nod v_j trebuie să cunoască doar, pentru fiecare destinație posibilă v_k , următorul vârf v_{j+1} de pe drumul optim spre acea destinație. Corespondența, pentru fiecare v_j , între destinația v_k și nodul următor v_{j+1} poartă denumirea de *tabelă de dirijare*.

Pentru a putea aplica direct un algoritm clasic de determinare a drumurilor de cost minim, este necesară centralizarea datelor despre nodurile și legăturile din rețea, în vederea obținerii efective a grafului rețelei. După calculul drumurilor, este necesară distribuirea tabelelor de dirijare către toate nodurile rețelei.

Într-o rețea mică, centralizarea informațiilor despre legături și apoi distribuirea informațiilor de dirijare către noduri se poate face manual, de către administratorul rețelei.

În rețelele mai mari, acest proces trebuie automatizat (total sau parțial). Deoarece nu este de dorit oprirea completă a rețelei oricâteori se modifică vreo legătură, trebuie luate măsuri ca timpul scurs de la modificarea legăturilor până la actualizarea a regulilor de dirijare pe toate nodurile să fie scurt și funcționarea rețelei în acest timp să fie acceptabilă. Metodele principale de calcul pentru tabelele de dirijare sunt descrise în § 5.2.1 și § 5.2.2.

În rețelele foarte mari, cum ar fi Internet-ul, centralizarea completă a datelor nu este rezonabilă; trebuie utilizați algoritmi de dirijare care să permită fiecărui nod efectuarea dirijării fără a necesita decât puține informații și doar despre o mică parte a rețelei. De asemenea, tabela de dirijare trebuie să aibă o reprezentare mai compactă decât câte un rând pentru fiecare nod destinație posibil. În astfel de cazuri se utilizează dirijarea ierarhică (§ 5.2.3).

Există și metode ad-hoc de dirijare, utilizate în diverse situații mai deosebite, de exemplu dacă graful asociat rețelei de calculatoare are anumite particularități. Acestea vor fi studiate în § 5.2.4.

5.2.1. Calculul drumurilor cu informații complete despre graful rețelei

În cadrul acestei metode, fiecare nod al rețelei adună toate informațiile despre graful asociat rețelei, după care calculează drumurile de la el la toate celelalte noduri.

Pentru ca fiecare nod să dispună în permanență de graful asociat rețelei de calculatoare, fiecare modificare a rețelei trebuie anunțată tuturor nodurilor. Pentru aceasta, fiecare nod testează periodic legăturile cu vecinii săi și, oricâteori constată o modificare, transmite o înștiințare în toată rețeaua. Transmitia informației respective se face astfel:

- Fiecare nod crează, periodic, un pachet ce conține numele nodului, starea legăturilor cu vecinii (costurile actuale ale legăturilor), precum și un *număr de secvență* (număr care tot crește de la un astfel de pachet la următorul). Apoi transmite acest pachet tuturor vecinilor, printr-un protocol sigur (cu confirmare și retransmitere).
- Fiecare nod ce primește un pachet descriind starea legăturilor verifică dacă este sau nu mai recent (adică cu număr de secvență mai mare) decât ultimul astfel de pachet primit de la acel nod. Dacă este mai recent, îl trimite tuturor vecinilor (mai puțin celui dinspre care a venit pachetul) și actualizează reprezentarea proprie a grafului rețelei. Dacă pachetul este mai vechi, înseamnă că este o copie ce a sosit pe altă cale și este ignorat.

Calculul drumurilor de cost minim de la un vârf la toate celelalte este o problemă clasică în teoria grafelor. Dacă toate costurile sunt pozitive — condiție îndeplinită de graful asociat unei rețele de calculatoare — algoritmul cel mai eficient este algoritmul lui Dijkstra (algoritmul 5.1). Notând cu n numărul de vârfuri (noduri ale rețelei) și cu m numărul de muchii (legături directe), complexitatea algoritmului lui Dijkstra este timp $O(m + n \log n)$ și spațiu $O(m + n)$. Calculul trebuie refăcut complet la fiecare modificare a grafului asociat rețelei de calculatoare.

5.2.2. Calculul drumurilor optime prin schimb de informații de distanță

Această metodă (vezi algoritmul 5.2) este inspirată din algoritmul Bellman-Ford de determinare a drumurilor de cost minim într-un graf, însă calculele sunt repartizate între nodurile rețelei de calculatoare în așa fel încât nici un nod să nu aibă nevoie de informații complete despre graf. Metoda se numește *cu vectori distanță* deoarece prevede transmiterea, de la fiecare nod la vecinii săi direcți, a unor vectori reprezentând distanțele de la nodul curent la toate celelalte noduri.

Algoritmul prevede că fiecare nod deține o tabelă conținând, pentru fiecare destinație posibilă, distanța până la ea și primul nod de pe drumul optim spre acea destinație. Inițial, tabelul este inițializat astfel: pentru vecinii direcți, costul drumului este pus ca fiind costul legăturii directe spre acel nod, iar primul nod spre acea destinație este fixat chiar acel nod; pentru nodurile ce nu sunt vecini direcți, costul este inițializat cu infinit.

După initializare, nodurile recalculează periodic tabelele de distanțe. Pentru fiecare nod, calculul se face astfel: mai întâi, nodul cere vecinilor direcți tabelele acestora. Apoi, pentru fiecare destinație posibilă, drumul optim este calculat ca fiind cel mai puțin costisitor dintre legătura directă și drumurile prin fiecare dintre vecinii direcți. Costul drumului printr-un vecin direct este calculat ca fiind costul legăturii dintre nodul curent și vecinul considerat adunat cu costul, conform tabelii vecinului, al drumului de la vecinul respectiv la nodul destinație. De remarcat că, în calculul tabelii de distanțe a unui nod, nu se utilizează deloc tabela de distanțe a acelui nod de la iterația precedentă.

După câteva iterații ale buclei principale, algoritmul se stabilizează (converge), în sensul că tabelele calculate la fiecare iterație sunt identice cu cele calculate la iterația precedentă. Numărul de iterații până la stabilizare este egal cu numărul cel mai mare de muchii de-a lungul vreunui drum optim.

După stabilizare, algoritmul este lăsat în continuare să se execute

Algoritmul *Dijkstra*

intrarea: $G = (V, E)$ graf orientat ($E \subseteq V \times V$)

$c : E \rightarrow [0, \infty)$ costurile asociate arcelor

$x_0 \in V$ vârful curent

ieşirea: $t : V \rightarrow \{(x_0, y) \in E\}$ tabela de dirijare; $t(x)$ este legătura directă prin care x_0 trebuie să trimită pachetele destinate lui x .

algoritmul:

pentru $i \in V$ execută

$d[i] := \infty$

sfârşit pentru

$d[x_0] := 0$

$Q := V$

cât timp $Q \neq \emptyset$ execută

fie $v \in Q$ elementul din Q pentru care $d[v]$ este minim

$Q := Q \setminus \{v\}$

pentru $y \in Q : (v, y) \in E$ execută

dacă $d[v] + c(v, y) < d[y]$ atunci

$d[y] := d[v] + c(v, y)$

dacă $v = x_0$ atunci

$t(y) := (x_0, y)$

altfel

$t(y) := t(v)$

sfârşit dacă

sfârşit dacă

$Q := Q \cup \{y\}$

sfârşit pentru

sfârşit cât timp

sfârşit algoritm

Algoritmul 5.1: Algoritmul lui Dijkstra cu adăugirea pentru calculul tablei de dirijare.

Algoritmul *Vector_dist*

intrarea: V mulţimea de noduri a reţelei;

x nodul curent;

$N^{\text{out}}(i)$ mulţimea vecinilor direcţi ai lui i ;

$(c_{i,j})_{i,j \in V}$ costurile legăturilor directe; $c_{i,j} = \infty$ dacă $i \notin N^{\text{out}}(i)$; fiecare nod x cunoaşte doar $(c_{x,j})_{j \in V}$.

ieşirea: $(d_{i,j})_{i,j \in V}$ costurile drumurilor optime; fiecare nod va calcula doar

$(d_{x,j})_{j \in V}$;

$(p_{i,j})_{i,j \in V}$ primul nod, după i , pe drumul optim de la i la j .

algoritmul:

pentru $i \in V$ execută

$d_{x,i} := c_{x,i}$

sfârşit pentru

cât timp adevărat execută

obţine de la vecinii direcţi $d_{i,j}$, pentru $i \in N^{\text{out}}(i)$

pentru $j \in V$ execută

$d_{x,j} := \min(c_{x,j}, \min_{i \in N^{\text{out}}(x)} d_{x,i} + c_{i,j})$

$p_{x,i} :=$ vecinul pentru care s-a obţinut minimul

sfârşit pentru

sfârşit cât timp

sfârşit algoritm

Algoritmul 5.2: Algoritmul de dirijare cu vectori distanţă

pentru ca, dacă ulterior se modifică legăturile directe, să actualizeze în continuare tabelele de dirijare. Dealtfel, este destul de dificil de determinat, în interiorul algoritmului, momentul în care s-a produs stabilizarea. Dacă apare o legătură directă nouă sau dacă scade costul unei legături directe existente, tabelele de dirijare se stabilizează din nou după un număr de iterații cel mult egal cu numărul maxim de muchii de-a lungul unui drum optim. Dacă se elimină o legătură directă sau crește costul unei legături directe, tabelele de dirijare se stabilizează mult mai încet, așa cum se vede în exemplul 5.2.

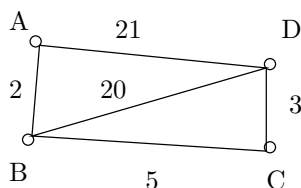


Figura 5.5: Rețeaua pentru exemplele 5.1 și 5.2. Numerele reprezintă costurile asociate legăturilor directe.

EXEMPLUL 5.1: Fie rețeaua din figura 5.5. Calculul tabelelor de dirijare, conform algoritmului 5.2, de la inițializare până la stabilizare, duce la următoarele tabele:

- *Inițializarea:* În această fază, sunt luate în considerare doar legăturile directe; dacă un nod nu este accesibil direct, ruta până la acesta este marcată ca având cost infinit.

Nodul A:			Nodul B:			Nodul C:		
dest.	via	cost	dest.	via	cost	dest.	via	cost
B	B	2	A	A	2	A	–	∞
C	–	∞	C	C	5	B	B	5
D	D	21	D	D	20	D	D	3
Nodul D:								
dest.	via	cost						
A	A	21						
B	B	20						
C	C	3						

- *Iterația 1:* Pentru fiecare destinație posibilă, se ia în considerare legătura directă (dacă există) și rutele prin fiecare din vecinii direcți. Costul legăturii directe este cunoscut, iar costul rutei printr-un vecin este costul legăturii spre acel vecin plus costul raportat de acel vecin. De exemplu, nodul B ia în considerare ca rute spre D: legătura directă de cost 20,

legătura prin A de cost $2+21=23$ și legătura prin C de cost $5+3=8$; cea mai bună este cea prin C. Ca alt exemplu, nodul A are următoarele rute spre D: legătura directă de cost 21 și legătura prin B de cost $2+20=22$; de notat că pentru legătura prin B se ia costul $B \rightarrow D$ raportat de B, calculat de către B la inițializare.

Nodul A:			Nodul B:			Nodul C:		
dest.	via	cost	dest.	via	cost	dest.	via	cost
B	B	2	A	A	2	A	B	7
C	B	7	C	C	5	B	B	5
D	D	21	D	C	8	D	D	3

Nodul D:		
dest.	via	cost
A	A	21
B	C	8
C	C	3

- *Iterația 2:* Să urmărim ruta calculată de A către D. Sunt luate în considerare legătura directă de cost 21 și legătura prin B a cărui cost este acum $2+8=10$ întrucât se bazează pe costul legăturii $B \rightarrow D$ calculat de către B la iterația 1.

Nodul A:			Nodul B:			Nodul C:		
dest.	via	cost	dest.	via	cost	dest.	via	cost
B	B	2	A	A	2	A	B	7
C	B	7	C	C	5	B	B	5
D	B	10	D	C	8	D	D	3

Nodul D:		
dest.	via	cost
A	C	10
B	C	8
C	C	3

- Începând cu iterația 3, tabelele calculate sunt identice cu cele de la iterația 2.

EXEMPLUL 5.2: Fie rețeaua din figura 5.5 și fie tabelele de dirijare rezultate după stabilizarea algoritmului cu vectori distanță (vezi exemplul 5.1). Să presupunem că legătura $B-C$ cade, rezultând rețeaua din figura 5.6. Să urmărim evoluția, în continuare, a tabelor de dirijare.

La prima iterație, la recalcularea rutelor nodului B spre C și spre D, nodul B ia în calcul rute prin A sau prin D. Rutele optime găsite sunt cele prin A, bazate pe vechile tabele ale lui A; nodul B nu are cum să determine

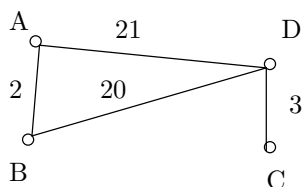


Figura 5.6: Rețeaua rezultată prin căderea legăturii B–C din rețeaua din figura 5.6.

că aceste rute nu mai sunt valide deoarece se bazează pe legătura B–C. La fel procedează și nodul C, găsind că rutele optime spre A și B trec prin D.

Nodul A:			Nodul B:			Nodul C:		
dest.	via	cost	dest.	via	cost	dest.	via	cost
B	B	2	A	A	2	A	D	13
C	B	7	C	A	9	B	D	11
D	B	10	D	A	12	D	D	3
Nodul D:								
dest.	via	cost						
A	C	10						
B	C	8						
C	C	3						

La următoarea iterație se vor modifica costurile rutelor din A spre C și D și din D spre A și B:

Nodul A:			Nodul B:			Nodul C:		
dest.	via	cost	dest.	via	cost	dest.	via	cost
B	B	2	A	A	2	A	D	13
C	B	11	C	A	9	B	D	11
D	B	14	D	A	10	D	D	3
Nodul D:								
dest.	via	cost						
A	C	16						
B	C	14						
C	C	3						

În continuare, costurile aparente ale rutelor cresc de la o iterație la alta, până când ajung la valorile rutelor reale optime. La a 3-a iterație de la căderea legăturii B–C, tabelele ajung în forma următoare:

Nodul A:			Nodul B:			Nodul C:		
dest.	via	cost	dest.	via	cost	dest.	via	cost
B	B	2	A	A	2	A	D	19
C	B	11	C	A	13	B	D	17
D	B	14	D	A	14	D	D	3
Nodul D:								
dest.	via	cost						
A	C	16						
B	C	14						
C	C	3						

Urmează, la a 4-a iterație, descoperirea de către D a rutelor reale spre A și spre B:

Nodul A:			Nodul B:			Nodul C:		
dest.	via	cost	dest.	via	cost	dest.	via	cost
B	B	2	A	A	2	A	D	19
C	B	15	C	A	13	B	D	17
D	B	18	D	A	14	D	D	3
Nodul D:								
dest.	via	cost						
A	A	21						
B	B	20						
C	C	3						

Restul rutelor reale sunt descoperite și mai târziu, stabilizarea tabelor survenind abia la a 8-a iterație.

În general, numărul de iterații după care se stabilizează tabelele după căderea sau creșterea costului unei legături poate fi cel mult egal cu raportul dintre cea mai mare creștere de cost între două noduri și cel mai mic cost al unei legături directe. În cazul exemplului 5.2, costul drumului optim de la B la C crește, prin căderea legăturii directe B–C, de la 5 la 23, o creștere de 18 unități. Costul cel mai mic al unei legături directe este 2 (legătura A–B). Ca urmare, stabilizarea tabelor poate lua cel mult $\frac{18}{2} = 9$ iterații. În cazul în care căderea unei legături duce la deconectarea rețelei, acest lucru nu va fi detectat niciodată, numărul de iterații necesar fiind infinit.

Pentru a îmbunătăți comportamentul în cazul căderii sau creșterii costului legăturilor, se poate modifica algoritmul astfel: tabelele vor ține ruta completă spre destinație, iar la recalcularea rutelor, rutele ce trec de două ori prin același nod nu sunt luate în considerare.

EXEMPLUL 5.3: Să reluăm rețeaua din exemplul 5.2, cu memorarea întregului

drum în tabela de distanţe. După stabilizarea tabelelor pe reţeaua din figura 5.5, se obţin următoarele tabele:

Nodul A:			Nodul B:			Nodul C:		
dest.	ruta	cost	dest.	ruta	cost	dest.	ruta	cost
B	B	2	A	A	2	A	B,A	7
C	B,C	7	C	C	5	B	B	5
D	B,C,D	10	D	C,D	8	D	D	3
Nodul D:								
dest.	ruta	cost						
A	C,B,A	10						
B	C,B	8						
C	C	3						

După căderea legăturii B–C (fig. 5.6), evoluţia tabelelor de dirijare are loc după cum urmează:

- *Iteraţia 1:* Să considerăm drumurile posibile de la nodul B spre nodul C. Legătură directă nu există. Drumul prin A începe cu muchia AB şi continuă cu ruta din tabela, de la iteraţia anterioară, a lui A, adică drumul ABC. Prin urmare, drumul prin A este BABC şi este respins datorită repetării vârfului B. De menţionat că nu se face vreo verificare în urma căreia să se observe că drumul BABC conţine muchia inexistentă BC; din lipsa unor informaţii globale, este imposibil de prins toate cazurile de utilizare a unor muchii inexistente. Drumul de la B la C prin D este BDC, de cost $20+3=23$; acesta este singurul candidat, ca urmare este ales ca rută optimă de la B la C.

Analog, în calculul rutei de la B la D, ruta prin A, anume BABCD, este respinsă şi, ca urmare, rămâne să fie aleasă doar legătura directă BD. La calculul rutei de la C la A, ar exista o singură posibilitate, prin nodul D, însă aceasta conduce la drumul CDCBA care este respins din cauza repetării nodului C. Ca urmare, nodul C marchează lipsa rutei punând costul ∞ . Analog, se determină inexistenţa vreunei rute valide de la C la B.

Nodul A:			Nodul B:			Nodul C:		
dest.	ruta	cost	dest.	ruta	cost	dest.	ruta	cost
B	B	2	A	A	2	A	–	∞
C	B,C	7	C	D,C	23	B	–	∞
D	B,C,D	10	D	D	20	D	D	3

Nodul D:

dest.	ruta	cost
A	C,B,A	10
B	C,B	8
C	C	3

- *Iterația 2:*

Nodul A:

dest.	ruta	cost
B	B	2
C	D,C	24
D	D	21

Nodul B:

dest.	ruta	cost
A	A	2
C	D,C	23
D	D	20

Nodul C:

dest.	ruta	cost
A	–	∞
B	–	∞
D	D	3

Nodul D:

dest.	ruta	cost
A	A	21
B	B	20
C	C	3

- *Iterația 3:* Se stabilizează tabelele.

Nodul A:

dest.	ruta	cost
B	B	2
C	D,C	24
D	D	21

Nodul B:

dest.	ruta	cost
A	A	2
C	D,C	23
D	D	20

Nodul C:

dest.	ruta	cost
A	D,A	24
B	D,B	23
D	D	3

Nodul D:

dest.	ruta	cost
A	A	21
B	B	20
C	C	3

5.2.3. Dirijarea ierarhică

Dirijarea ierarhică se aplică cu precădere în rețelele foarte mari, unde este imposibil ca fiecare nod să aibă informații despre toate celelalte noduri. Exemple clasice de astfel de rețele sunt Internet-ul și rețeaua telefonică.

Ideea dirijării ierarhice este ca rețeaua să fie împărțită în *subrețele*. Subrețelele alcătuiesc o ierarhie arborescentă: o subrețea rădăcină (considerată pe nivelul 0), câteva subrețele subordonate ei (nivelul 1), subrețele subordonate câte unei subrețele de pe nivelul 1 (alcătuind nivelul 2), ș. a. m. d. Fiecare nod are informații de dirijare:

- către nodurile din subrețeaua proprie, individual pentru fiecare nod;

- către subreţeaua imediat superioară ierarhic: o singură rută, comună, pentru toate nodurile din acea subreţea, ruta conducând spre cel mai apropiat
- către fiecare din subreţelele imediat inferioare ierarhic, câte o rută pentru fiecare subreţea.

Ruta de la un nod iniţial către o subreţea vecină subreţelei nodului iniţial este ruta de la nodul iniţial către cel mai apropiat nod de la graniţa dintre cele două subreţele.

Fiecare subreţea este suficient de mică, astfel încât, în interiorul fiecărei subreţele, calculul rutelor se face prin metode de dirijare „obişnuite“.

Pentru ca orice nod să poată determina din ce subreţea face parte nodul destinaţie a unui pachet, precum şi localizarea subreţelei respective în ierarhie, adresa fiecărui nod este astfel construită încât să descrie poziţia nodului în ierarhia de reţele. Astfel, adresele sunt formate din componente, prima componentă identificând subreţeaua de nivel 1 din care face parte sau căreia îi este subordonat nodul, urmând identificatorul subreţelei de nivel 2, ş. a. m. d., încheind cu identificatorul nodului în cadrul subreţelei din care face parte.

De remarcat că, în general, dirijarea ierarhică nu conduce la drumul optim către destinaţie. Aceasta deoarece în dirijarea ierarhică se caută optimul local în fiecare subreţea şi, ca urmare, este posibil să se rateze optimul global (a se vedea exemplul 5.4).

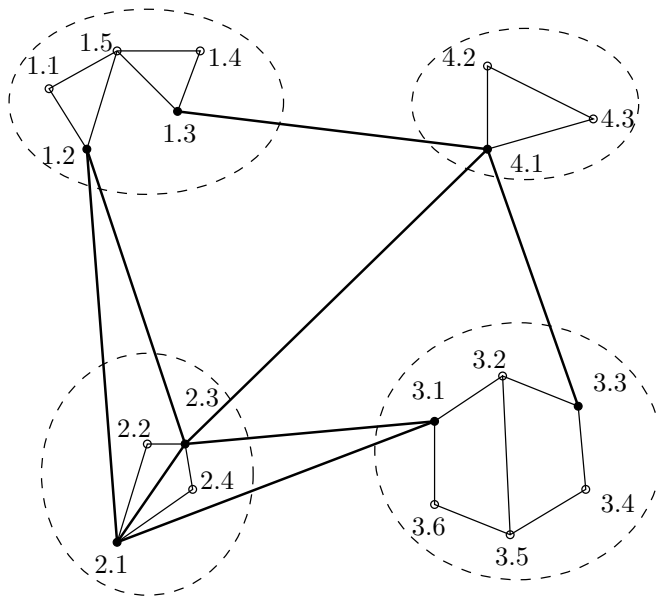
EXEMPLUL 5.4: În figura 5.7 este reprezentată o reţea cu dirijare ierarhică pe două nivele. Reţeaua este formată dintr-o subreţea rădăcină şi patru subreţele subordonate ei.

Adresa fiecărui nod este formată din două componente, prima identificând subreţeaua de nivel 1 din care face parte şi a doua identificând nodul în cadrul subreţelei respective.

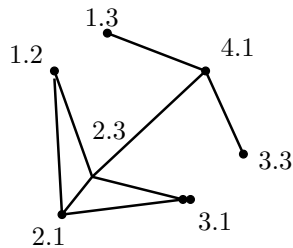
Să presupunem că nodul 1.1 are de trimis un pachet către nodul 3.4. Dirijarea decurge astfel:

- Nodul 1.1 determină că destinaţia 3.4 face parte din altă subreţea decât el însuşi; ca urmare, caută drumul spre cel mai apropiat nod ce are legătură cu reţeaua ierarhic superioară. Nodul acesta este 1.2.
- Nodul 1.2 caută drumul spre cel mai apropiat nod din subreţeaua 3. Nodul găsit este 3.1 şi drumul până la el este 1.2, 2.3, 3.1 (echivalent, se poate lua drumul 1.2, 2.1, 3.1).
- Nodul 3.1 trimite pachetul spre destinaţia 3.4 pe drumul cel mai scurt, anume 3.1, 3.2, 3.3, 3.4 (alt drum, echivalent, este 3.1, 3.6, 3.5, 3.4).

Să observăm că drumul pe care îl urmează pachetul, 1.1, 1.2, 2.3, 3.1, 3.2, 3.3,



(a) Toată reţeaua. Subreţelele de pe nivelul 1 sunt încercuite cu linie punctată.



(b) Reprezentarea (micşorată) doar a subreţelei rădăcină.

Figura 5.7: O reţea cu dirijare ierarhică pe două nivele. Reţeaua de pe nivelul rădăcină are nodurile reprezentate prin cercuri pline (mici) şi legăturile reprezentate cu linii îngroşate.

3.4 nu este optim, întrucât lungimea lui este 6, iar drumul 1.1, 1.5, 1.3, 4.1, 3.3, 3.4 are lungimea 5.

5.2.4. Metode particulare de dirijare

5.2.4.1. Inundarea

Inundarea este o metodă aplicabilă în rețele bazate pe datagrame. Inundarea constă în a trimite copii ale unei datagrame prin toate legăturile directe, cu excepția celei prin care a intrat datagrama.

Inundarea garantează că, dacă destinația este accesibilă și nodurile nu sunt prea încărcate (astfel încât să se sacrifice datagrame din lipsă de spațiu de memorare), datagrama ajunge la destinație. Ca avantaj față de alte metode, inundarea nu necesită ca nodurile să adune nici un fel de informație despre rețea.

Pe de altă parte, inundarea face ca fiecare datagramă să ajungă la fiecare nod al rețelei, nu doar la destinatarul dorit. Ca urmare, la fiecare nod ajung toate pachetele care circulă prin rețea. La un număr de noduri mai mare de câteva zeci, metoda inundării generează prea mult trafic pentru a fi în general acceptabilă.

Dacă graful rețelei este un arbore, atunci, considerând nodul sursă a datagramei ca rădăcină, copiile datagramei circulă în arbore de la fiecare nod la fii săi; transmisia se oprește la frunze. De notat însă că o rețea al cărei graf atașat este un arbore este extrem de vulnerabilă la pene: defectarea oricărui nod intern duce la deconectarea rețelei.

Dacă însă graful rețelei conține cicluri, atunci o datagramă, o dată ajunsă într-un ciclu, ciclează la infinit. Pentru ca inundarea să fie utilizabilă în rețele cu cicluri, trebuie făcută o modificare pentru prevenirea ciclării infinite.

O posibilă soluție — utilizată și pentru alte metode de dirijare — este aceea de-a asocia fiecărei datagrame un *contor de salturi* care marchează prin câte noduri a trecut datagrama. La atingerea unei anumite valori prestabilite, datagrama nu mai este trimisă mai departe. Cu această modificare, inundarea transmite datagramele pe toate drumurile (nu neapărat simple) de la sursa datagramei și de lungime dată.

O altă soluție, cu avantajul suplimentar că asigură ca fiecare pachet să ajungă într-un singur exemplar la destinație, este ca fiecare nod al rețelei să identifice (de exemplu, prin menținerea unor numere de secvență) duplicatele unui pachet și să trimită mai departe un pachet doar la prima lui sosire.

Inundarea se utilizează în rețelele Ethernet. Graful unei rețelele Ethernet trebuie să fie întotdeauna un arbore.

5.2.4.2. Învățarea rutelor din adresele sursă ale pachetelor

O metodă simplă de construcție a tabelelor de dirijare este ca, la primirea unui pachet de la un nod sursă S dinspre un nod vecin V , să se introducă sau să se actualizeze în tabela de dirijare regula pentru destinația S prevăzând ca următor nod pe V . Regulile astfel introduse trebuie să aibă valabilitate limitată în timp — altfel apar probleme la modificarea legăturilor din rețea. De asemenea, mai trebuie un mecanism pentru dirijarea pachetelor pentru care încă nu există reguli de dirijare — de exemplu, se poate folosi inundarea.

Metoda este utilizată în rețelele Ethernet.

5.2.5. Metode de difuziune

Ne vom ocupa în continuare de metodele de dirijare aplicabile în vederea trimiterii copiilor unei datagrame spre mai multe destinații. Distingem două posibile cerințe, *difuziune completă* (engl. *broadcast*) — trimiterea spre toate nodurile unei rețele — și *difuziune selectivă* (engl. *multicast*) — trimiterea datagramei spre o submulțime dată a multimei nodurilor.

Desigur, întotdeauna este posibilă difuzarea prin transmiterea separată a unei datagrame spre fiecare nod. O astfel de metodă este însă neeconomică.

O posibilitate simplă de realizare a difuziunii complete este inundarea. (§ 5.2.4.1).

O altă posibilitate este să se construiască întâi un arbore parțial (preferabil de cost minim) de acoperire a vârfurilor destinație, iar apoi să se aplice metoda inundării în acest arbore. Această metodă este utilizabilă atât pentru difuzare completă cât și pentru difuzare parțială. Datorită necesității calculului arborelui parțial, este favorabilă în cazul în care trebuie trimise multe datagrame aceleiași mulțimi de destinatari.

Descriem și o a treia posibilitate, utilă în special în situația în care destinatarii sunt puțini și nu se trimit multe datagrame aceleiași mulțimi de destinatari. Metoda constă în a trimite în datagramă multimea adreselor destinație. Fiecare nod determină legătura de ieșire pentru fiecare destinație din lista din datagramă. Apoi trimite câte o datagramă pe fiecare legătură directă ce apare pe ruta spre cel puțin una dintre destinații. Datagrama trimisă prin fiecare legătură directă va avea în lista de destinații doar acele noduri către care ruta trece prin acea legătură directă. Intuitiv, metoda ar putea fi privită astfel: se trimite câte o datagramă către fiecare nod destinație, însă, cât timp drumul a două sau mai multe datagrame este comun, datagramele călătoresc reunite într-o singură datagramă cu mai multe adrese destinație.

5.3. Funcționarea la trafic ridicat

Până aici am studiat comportamentul unei rețele doar pentru cazul în care debitul fluxului de date care intră într-un nod nu depășește niciodată nici capacitatea legăturilor prin care trebuie trimis mai departe, nici capacitatea modulului de rețea de-a efectua prelucrările necesare. Dacă debitul cu care intră pachete într-un nod depășește fie capacitatea de prelucrare a nodului, fie capacitatea legăturii prin care pachetele trebuie să iasă, nodul memorează pachetele într-o structură de coadă, de unde le extrage pe măsură ce pot fi transmise prin legătura de ieșire. Un efect imediat este creșterea timpului de propagare, din cauza staționării pachetelor în coada de așteptare. Dacă excesul de debit de intrare se păstrează mai mult timp, coada crește până când memoria alocabilă cozii de așteptare este epuizată; în acel moment, nodul va trebui fie să sacrifice pachete, fie să solicite, prin intermediul mecanismului de control al fluxului de la nivelul legăturii de date, micșorarea debitului de intrare. De notat că reducerea și, în extremis, blocarea fluxului de date la intrarea într-un nod poate duce la acumularea de date de transmis în nodului vecin dinspre care vine acel flux, ducând mai departe la blocarea reciprocă a unui grup de noduri.

Principalele probleme ce apar în cazul în care capacitatea nodurilor sau legăturilor directe este depășită sunt următoarele:

- Într-o rețea aglomerată, pachetele sau datagramele întârzie mult sau chiar se pierd, lucru care poate declanșa retrimiteri intempestive de pachete, ducând la aglomerare și mai mare a rețelei și la performanțe și mai scăzute. O astfel de situație, de degradare suplimentară a performanțelor în urma creșterii încărcării, se numește *congestie* și trebuie evitată sau, cel puțin, ținută sub control.
- Capacitatea disponibilă a rețelei trebuie împărțită în mod echitabil între utilizatori.
- Diferite aplicații au diferite priorități cu privire la caracteristicile necesare ale serviciului oferit de rețea. Reacția unei rețele aglomerate trebuie să țină cont de aceste priorități. De exemplu, un nod supraaglomerat poate să fie nevoit să arunce o parte dintre datagramele aflate în tranzit. Dacă datagramele aparțin unei aplicații de transfer de fișiere, este preferabil să fie aruncate cele mai recente (acestea fiind retransmise mai târziu; dacă se aruncă datagramele mai vechi, este posibil ca destinatarul să nu aibă ce face cu cele mai noi și să trebuiască retransmise toate). Dimpotrivă, dacă datagramele aparțin unei aplicații de tip videoconferință,

este preferabil să fie aruncate datagramele mai vechi.

De notat că, adesea, rezolvarea problemelor de mai sus necesită o colaborare între nivelul rețea și nivelele superioare.

5.3.1. Alegerea pachetelor de transmis

Considerăm un ruter ale cărui linii de ieșire sunt utilizate la maximul capacității. Vom analiza în continuare modul în care el poate alege, dintre pachetele primite, care va fi următorul pachet pe care să-l retransmită.

O posibilitate simplă este de a menține o singură coadă și de a accepta un pachet proaspăt sosit dacă are loc în coadă și de a-l distruge dacă nu are loc. Atunci când debitul de intrare este mare, soluția duce la a accepta primul pachet ce sosește după eliberarea unei poziții în coadă. Ca urmare, un emițător care produce multe pachete este avantajat față de un emițător care produce puține pachete.

O distribuire mai echitabilă a capacității este de-a construi câte o coadă pentru fiecare nod sursă, legătură de intrare sau cicruit virtual. Nodul extrage, în vederea retransmiterii, pe rând, câte un pachet din fiecare coadă. În acest fel, fiecare sursă fiecăre linie de intrare sau, după caz, circuit virtual obține trimiterea aceluiași număr de pachete în unitatea de timp. Metoda se numește *așteptare echitabilă* (engl. *fair queueing*).

O variantă a metodei anterioare este de a oferi fiecărei intrări nu un număr egal de pachete preluate ci un număr egal de biți preluați. Pentru aceasta, se poate asocia fiecărei cozi numărul total de biți ai pachetelor preluate din acea coadă și retransmise mai departe. De fiecare dată nodul intermediar extrage următorul pachet din coada cu cel mai mic număr de biți transmiși.

Pe lângă posibilitatea de a oferi intrărilor transmiterea aceluiași număr de biți sau de pachete, se poate oferi numere de biți sau pachete transmise proportionale cu anumite valori. De exemplu, se poate oferi unei legături mai importante, dinspre un grup mai mare de calculatoare, un număr dublu de biți preluați și retransmiși față de o legătură secundară. Metoda se numește *așteptare echitabilă ponderată* (engl. *weighted fair queueing*).

În afară de metodele de așteptare echitabilă — eventual, împreună cu ele — se poate pune în aplicare și un sistem de priorități. Astfel, fiecărui pachet i se poate asocia un nivel de prioritate: pachetele pentru aplicații în timp real și pachetelor asociate sesiunilor interactive li se asociază nivele de prioritate mai ridicate, iar aplicațiilor care transferă fișiere mari li se asociază nivele de prioritate coborâtă. Într-un ruter, fiecărui nivel de prioritate i se asociază o coadă separată, cu spațiu de memorare rezervat. Atunci când linia de ieșire este liberă și ruterul trebuie să decidă care este următorul pachet, examinează

cozile în ordine descrescătoare a nivelelor de prioritate până găsește o coadă nevidă. Pachetul următor ce va fi transmis este extras din prima coadă nevidă.

Dacă metoda priorităților este combinată cu așteptarea echitabilă, fiecărui nivel i se asociază un set de cozi, în interiorul setului funcționând regulile de la așteptarea echitabilă. Următorul pachet trimis este extras din setul de cozi cel mai prioritar în care există cel puțin o coadă nevidă.

5.3.2. Controlul congestiei

Prin *congestie* se înțelege scăderea debitului traficului util în rețea în situația în care cererea de trafic printr-o legătură sau printr-un nod depășește capacitatea acesteia. Scăderea debitului util, în opoziție cu limitarea traficului la capacitatea legăturii sau nodului respectiv, este datorată unei funcționări defectuoase a rețelei, în special datorită pierderii și retransmiterii unui număr mare de pachete.

Ca principiu general, este bine ca, dacă rețeaua este foarte încărcată, nodurile terminale să încerce să reducă frecvența și mărimea pachetelor transmise. Evident, pentru acest lucru, este necesar un mecanism care să semnaleze nodurilor finale asupra prezenței sau iminenței congestiei.

Descriem în continuare, pe scurt, mecanisme utilizate pentru semnalarea congestiei, precum și mecanismele prin care nodurile finale pot reacționa la astfel de semnale.

Prima posibilitate de semnalizare a congestiei este ca, atunci când un nod intermediar este încărcat la limita capacității sale, pentru fiecare pachet de date primit spre livrare să trimită sursei pachetului de date un pachet de control prin care să-i ceară să reducă traficul. Cusurul metodei constă în faptul că pachetele de cerere de reducere a traficului încarcă suplimentar o rețea deja încărcată. Metoda este utilizabilă în Internet, existând un tip de pachete ICMP pentru acest scop (vezi § 10.2.5.4).

A doua posibilitate este ca nodul încărcat să semnalizeze destinației fiecărui pachet de date faptul că rețeaua este încărcată. Această semnalizare este mai ușor de făcut întrucât poate fi transmisă odată cu pachetul de date, sub forma unui bit din antetul fiecărui pachet. Dezavantajul, față de metoda precedentă, este că nu semnalizează sursei traficului, ci destinației; rămâne deci necesar de elaborat un protocol de informare a sursei. Informarea sursei poate fi făcută simplu dacă între sursă și destinație se utilizează un protocol de control al fluxului: în cazul în care destinației îi este semnalizat că rețeaua este congestionată, destinația cere sursei, prin intermediul protocolului de control al fluxului, să reducă debitul transmisiei. Metoda semnalizării destinației este utilizată în Internet, sub numele de *explicit congestion notification*; pentru

informarea, mai departe, a sursei se poate utiliza dimensiunea ferestrei TCP (§ 10.3.1.8).

O semnalizare implicită a faptului că rețeaua este încărcată constă în însăși pierderea pachetelor. Pierderea poate fi observată de nodul sursă prin aceea că nu primește confirmări (în cazul utilizării unui protocol cu confirmare și retransmitere, § 4.3) sau răspuns la mesajele trimise (în cazul unei aplicații care trimite o datagramă de cerere și așteaptă o datagramă care să răspundă la cerere). Pentru ca pierderea pachetelor să poată fi utilizată ca semnalizare a congestiei, mai este necesar ca pierderea unui pachet din alte cauze decât congestia să fie puțin probabilă. Rezultă, pentru legăturile directe cu rată a erorilor ridicată (în principal, legături radio), necesitatea utilizării, la nivelul legăturii de date, fie a unui cod corector de erori, fie a unui protocol de confirmare și retransmitere .

Indiferent de metoda de semnalizare utilizată, o implementare simplă riscă să ducă la oscilații: dacă un nod intermediar ajunge congestionat, semnalizează tuturor nodurilor terminale ale legăturilor stabilite prin el despre congestie. Reacția este diminuarea traficului prin toate legăturile și ca urmare scăderea traficului mult sub maximul admis. După un timp, nodurile terminale vor crește din nou traficul, până la congestionarea, din nou, a nodului intermediar considerat. Soluționarea problemei oscilațiilor se face punând nodul intermediar să trimită semnale că este supraîncărcat cu puțin înainte de-a ajunge la limita capacității sale și de-a alege aleator legăturile cărora li se semnalizează încărcarea.

5.3.3. Formarea (limitarea) traficului

Prin *formarea traficului* se înțeleg metode de uniformizare a debitului unui flux de date. Mecanismele de limitare pot fi plasate în nodul sursă sau într-un ruter și pot acționa asupra fluxului de pachete provenit de la un anumit nod sursă, asupra fluxului între două stații date, asupra fluxului printr-un circuit virtual sau asupra fluxului ce intră sau iese printr-o anumită legătură directă.

Cel mai simplu mecanism de formare a traficului este limitarea debitului de date la o anumită valoare fixată. Mecanismul se numește *găleată găurită*, prin analogie cu următorul mecanism fizic: într-o găleată (reprezentând coada de așteptare a ruterului) se toarnă apă (reprezentând pachetele unui flux). Găleata are o gaură prin care curge apă (pachete ce sunt preluate din coadă și retransmise de către ruter). Debitul apei care curge (debitul fluxului de ieșire) este constant atâ timp cât găleata nu este goală. De asemenea, dacă găleata este plină, o parte din apa ce intră se revarsă în afară (surplusul

de pachete se pierd).

Un mecanism mai elaborat permite scurte rafale. Ca idee, ruterul ține evidența capacității nefolosite (diferența dintre debitul maxim acceptat și debitul fluxului) și permite, în contul acesteia, un exces de debit. Mai în detaliu, ruterul asociază cozii un număr de *jetoane*. Periodic, numărul de jetoane este crescut cu o unitate, fără însă a depăși o valoare maximă. Dacă există un pachet în coadă și numărul de jetoane este mai mare sau egal cu numărul de biți ai pachetului, pachetul este preluat din coadă și retransmis, iar numărul de jetoane asociat cozii este scăzut cu o valoare egală cu numărul de biți ai pachetului. Mecanismul se numește *găleata cu jeton*.

5.3.4. Rezervarea resurselor

Pentru a avea, în mod garantat, o anumită capacitate și un anumit timp de propagare oferite unui flux de date, este necesar să fie rezervate fluxului resursele necesare — capacitatea de prelucrare în noduri și capacitatea de transfer prin legăturile directe.

Rezervarea resurselor se poate face doar în rețele ce oferă servicii de tip conexiune — utilizarea rezervării în rețele cu datagrame duce la necesitatea implementării unui mecanism de ținerea evidenței fluxurilor de datagrame similar celui din rețelele cu circuite virtuale.

La deschiderea conexiunii, în timpul stabilirii rutei conexiunii se stabilește și capacitatea pe care o va garanta conexiunea și fiecare nod se asigură că dispune de capacitățile necesare (că suma capacităților alocate conexiunilor ce partajează o legătură directă nu depășește capacitatea legăturii directe). Dacă resursele necesare nu sunt disponibile, conexiunea este refuzată sau se negociază o capacitate mai mică.

Asociat conexiunii se plasează, la nodul sursă, un mecanism de limitare a debitului de date, astfel încât operarea conexiunii să se încadreze în resursele alocate.

Rezervarea resurselor este utilă în special aplicațiilor în timp real. Rețeaua telefonică utilizează astfel de mecanisme.

Avantajul unui debit garantat se plătește prin limitarea drastică a debitului permis. Dacă debitul efectiv utilizat de un flux de date este adesea sub debitul alocat fluxului sau dacă o parte din capacitatea unei legături directe rămâne adesea nealocată complet conexiunilor ce trec prin ea, rețeaua nu este folosită la maximum de capacitate.

În acest caz, valorificarea capacității rămase, cu păstrarea capacității garantate prin rezervarea resurselor, se poate face astfel: Datelor ce aparțin conexiunilor cu trafic garantat li se asociază un nivel de prioritate ridicat.

Sunt permise și alte date (de exemplu, prin conexiuni fără trafic garantat), însă acestora li se asociază un nivel de prioritate scăzut. În fiecare ruter se utilizează un mecanism bazat pe priorități. În acest fel, fluxurile ce au rezervat resurse au capacitate garantată, iar restul datelor sunt transmise, fără vreo garanție, dacă mai rămân resurse și pentru ele.

5.4. Nivelul transport

Rolul nivelului transport este de-a face o adaptare între serviciile oferite de nivelul rețea și nevoile aplicațiilor. Funcțiile îndeplinite de nivelul transport sunt similare cu unele dintre funcțiile nivelului legăturii de date:

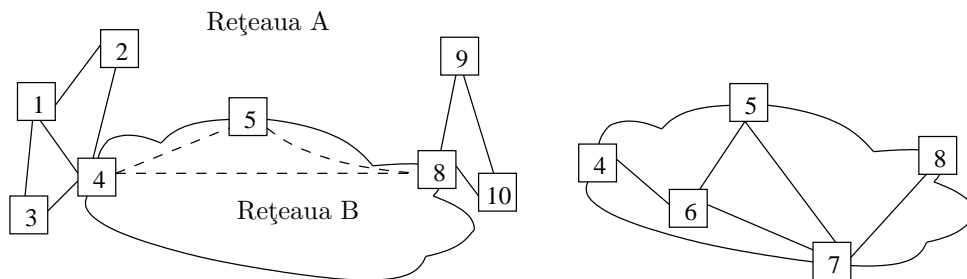
- *Transport sigur*: O rețea congestionată se poate să distrugă pachete din lipsă de spațiu de memorare. În plus, într-o rețea ce oferă transport de datagrame este posibil ca două datagrame să ajungă la destinație în ordine inversă față de cea în care au fost emise, iar în anumite cazuri este posibil ca o datagramă să ajungă în mai multe exemplare la destinație (de exemplu dacă se utilizează dirijare prin inundare și rețeaua nu este un arbore). Metodele bazate pe confirmări și retransmiteri (vezi § 4.3) pot fi utilizate, cu mici modificări, pentru a asigura transport sigur la nivelul transport. O diferență importantă față de transportul sigur la nivelul legăturii de date este că nivelul rețea poate inversa ordinea unor datagrame, în vreme ce nivelul fizic nu inversează pachete. O altă diferență este că la nivelul rețea timpul de propagare al unei datagrame variază în limite foarte largi. De aceea, pe de o parte trebuie luate măsuri pentru a fixa o durată maximă de viață a unei datagrame în rețea, iar pe de altă parte algoritmul de confirmare și retransmitere trebuie să se aștepte să primească astfel de datagrame întârziate și să nu le confunde cu datagrame noi — aceasta din urmă înseamnă că spațiul numerelor de secvență trebuie să fie suficient de mare.
- *Controlul fluxului*: Controlul fluxului are același rol și se implementează în același mod la nivelul transport ca și la nivelul legăturii de date.
- *Multiplexarea*: Multiplexarea poate fi utilă în mai multe scopuri: pentru a permite mai multor aplicații care se execută pe același calculator să comunice independent una de alta și pentru a permite unei perechi de aplicații care comunică să-și transmită independent mai multe fluxuri de date.

5.5. Interconectarea rețelelor

Probleme privind interconectarea rețelelor se pun în situația în care se dorește ca două sau mai multe rețele ce nu pot funcționa unitar ca o singură rețea să ofere totuși servicii de comunicare similare unei rețele unitare.

Motivele de existență a rețelelor distincte pot fi: rețele ce utilizează protocoale diferite la nivel rețea, rețele ce utilizează același protocol, dar există suprapuneri între adresele alocate în aceste rețele, dorința unor administratori de-a nu divulga informații despre legăturile din rețea sau de-a filtra pachetele care intră sau ies și altele.

Problema interconectării este complexă și necesită soluții ad-hoc, adaptate nevoilor de interoperabilitate și particularităților rețelelor de interconectat. Există însă câteva metode generale, dintre care vom analiza aici metoda tunelării.



(a) Rețeaua A, având legături directe proprii (reprezentate prin linii continue) și legături realizate ca tunele prin rețeaua B

(b) Rețeaua B

Figura 5.8: Legături prin tunel. O parte dintre legăturile directe din rețeaua A, figurate cu linie punctată, sunt obținute apelând la serviciile rețelei B. Legătura 4-5 apare ca legătură directă pentru rețeaua A, dar este construită de rețeaua B prin intermediul nodului 6.

Un *tunnel* este o legătură, realizată prin intermediul unei rețele, care este utilizată de o altă rețea ca și când ar fi o legătură directă (vezi fig. 5.8). Pachetele celei de-a doua rețele, incluzând antetele specifice acestuia, sunt transportate ca date utile printr-o conexiune sau, după caz, prin datagrame ale primei rețele.

Capitolul 6

Metode și protocoale criptografice

Vom studia în acest capitol cum se poate proteja comunicația dintre două entități contra acțiunilor unui terț, numit *adversar* sau *intrus*, care interceptează sau alterează comunicația între ele. Protecția comunicației împotriva acțiunilor unui adversar se numește *securizarea* comunicației.

Adversarul poate fi:

- *adversar pasiv*, care doar interceptează mesajele transmise;
- *adversar activ*, care și interceptează și modifică mesajele.

Protecția comunicației față de acțiunile adversarului cuprinde:

Asigurarea confidențialității are ca obiectiv să împiedice un adversar pasiv să înțeleagă un mesaj interceptat sau să extragă vreo informație din el.

Verificarea autenticității mesajelor, numită și *autentificarea mesajelor*, are ca obiectiv detectarea, de către receptor, a *falsurilor*, adică a mesajelor create sau modificate de un adversar activ. Verificarea autenticității mesajelor se aseamănă cu detectarea erorilor. Spre deosebire însă de detectarea erorilor, unde modificările produse de mediul de transmisie sunt aleatoare, la verificarea autenticității mesajelor avem un adversar care încearcă în mod deliberat să producă modificări nedetectabile.

Asigurarea non-repudiabilității mesajelor are ca obiectiv să permită receptorului să dovedească autenticitatea unui mesaj în fața unui terț, altfel spus, emițătorul să nu poată nega faptul că a transmis un anumit mesaj. Asigurarea non-repudiabilității este similară cu autentificarea mesajelor, dar în plus trebuie să nu permită nici măcar receptorului să creeze un mesaj care să pară autentic.

Verificarea prospețimii are ca obiectiv detectarea, de către receptor, a eventualelor copii ale unui mesaj (autentic) mai vechi. Este posibil ca un adversar să intercepteze, de exemplu, un ordin de transfer de bani în favoarea sa și apoi să transmită băncii multiple copii ale ordinului de transfer de bani. În lipsa verificării prospețimii, banca va efectua de mai multe ori transferul de bani. Verificarea autenticității mesajelor, singură, nu rezolvă problema, deoarece fiecare copie este identică cu originalul și, ca atare, este autentică.

Autentificarea entităților are ca obiectiv verificarea, de către o entitate, a identității entității cu care comunică. Mai exact, există un server și unul sau mai mulți clienți legitimi care deschid conexiuni către server. Modelul adversarului, în acest caz, este puțin diferit: adversarul poate să deschidă o conexiune spre server și să încerce să se dea drept un client legitim. Eventual, adversarul poate să intercepteze comunicațiile clienților legitimi, pentru a obține informații în vederea păcălirii serverului, dar nu poate altera comunicația printr-o conexiune deschisă de altcineva. În prezența unui adversar activ, autentificarea entităților nu este prea utilă, deoarece adversarul poate să lase protocolul de autentificare să se desfășoare normal și apoi să trimită orice în numele clientului. În prezența unui adversar activ, este mai degrabă necesar un mecanism de *stabilirea cheii* (vezi mai jos).

Stabilirea cheii are ca obiectiv obținerea, de către partenerii de comunicație legitimi, a unui șir de biți, numit *cheie*, ce urmează a fi utilizată la asigurarea confidențialității și la verificarea autenticității mesajelor. Cheia obținută trebuie să fie cunoscută doar de către cei doi parteneri care doresc să comunice.

În multe lucrări, în loc de autentificarea mesajelor se pune problema *verificării integrității mesajelor* — verificarea de către receptor că mesajul este identic cu cel emis de emițător (că nu a fost modificat pe traseu) — și a *autentificării sursei* mesajului — verificarea de către receptor a identității autorului unui mesajului. Cele două operații — verificarea integrității și autentificarea sursei — nu au sens decât împreună. Aceasta deoarece, dacă un mesaj a fost alterat de către adversar (lucru care se constată cu ocazia verificării integrității), mesajul poate fi văzut ca un mesaj produs de adversar și pretinzând că provine de la autorul mesajului original; acest din urmă mesaj nu a fost modificat în timpul transportului (de la adversar spre destinatar), dar sursa sa nu este autentică (mesajul provine de la altcineva decât autorul indicat în mesaj).

6.1. Asigurarea confidențialității

6.1.1. Introducere

Problema asigurării confidențialității unui mesaj constă în a transmite informații în așa fel încât doar destinatarul dorit să le poată obține; un adversar care ar intercepta comunicația nu trebuie să fie capabil să obțină informația transmisă.

Formal, presupunem că emițătorul are un mesaj de transmis, numit *text clar* (engl. *plaintext*). Emițătorul va genera, printr-un algoritm, plecând de la textul clar, un așa-zis *text cifrat* (engl. *ciphertext*). Receptorul autorizat trebuie să poată recupera textul clar aplicând un algoritm asupra textului cifrat. Adversarul, care dispune de textul cifrat dar nu cunoaște anumite detalii ale algoritmului aplicat de emițător, trebuie să nu fie capabil să reconstituie textul clar. Operația prin care emițătorul transformă textul clar în text cifrat se numește *criptare* sau, uneori, *cifrare* (engl. *encryption*). Operația prin care receptorul obține textul clar din textul cifrat se numește *decriptare* sau *descifrare* (engl. *decryption*). Împreună, algoritmii de criptare și decriptare constituie un *cifru*.

Pentru a formaliza notațiile, vom nota cu T mulțimea mesajelor posibile de transmis; fiecare text clar posibil este un element $t \in T$. Criptarea este o funcție $c : T \rightarrow M$, unde M este mulțimea textelor cifrate posibile. $m = c(t)$ este textul cifrat corespunzător textului clar t . Textul cifrat este trimis pe canalul nesigur și este presupus accesibil adversarului. Decriptarea o vom nota cu d , unde $d : M \rightarrow T$.

Spunem că (c, d) formează o pereche criptare-decriptare dacă îndeplinesc simultan condițiile:

- orice text cifrat poate fi decriptat corect prin d , adică $d \circ c = 1_T$;
- un adversar care cunoaște textul cifrat $m = c(t)$ dar nu cunoaște c sau d nu poate deduce t sau afla informații despre t .

În practică, este necesar ca producerea unei perechi de funcții (c, d) să fie ușor de făcut, inclusiv de către persoane fără pregătire deosebită. Acest lucru este necesar deoarece dacă perechea (c, d) utilizată de două entități care comunică este aflată, sau se bănuiește că a fost aflată, de către cineva din afară, ea trebuie schimbată repede. De asemenea, este bine ca persoanele ce nu au pregătire de matematică și informatică să poată utiliza singure metode criptografice.

Pentru acest scop, algoritmii de criptare și decriptare sunt făcuți să primească, pe lângă textul clar și respectiv textul cifrat, încă un argument

numit *cheie*. Fiecare valoare a cheii produce o pereche criptare-decriptare distinctă. Cheia se presupune a fi ușor de generat la nevoie.

Mulțimea cheilor posibile se numește *spațiul cheilor* și o vom nota în continuare cu K . Funcțiile de criptare și decriptare sunt de forma $c : T \times K \rightarrow M$ și respectiv $d : M \times K \rightarrow T$. Cheia este scrisă ca parametru. Pentru o valoare fixată a cheii $k \in K$, criptarea devine $c_k : T \rightarrow M$, iar decriptarea $d_k : M \rightarrow T$, cu $d_k \circ c_k = 1_T$. Pentru fiecare $k \in K$, (c_k, d_k) formează o pereche criptare-decriptare.

Algoritmii propriu-ziși, adică funcțiile $c : T \times K \rightarrow M$ și $d : M \times K \rightarrow T$, se presupune că sunt cunoscuți adversarului; dacă merită să puteți schimba cheia, înseamnă că deja aveți îndoieli privitoare la cât de secret puteți ține algoritmul față de adversar... Ca urmare, pentru o aplicație, un algoritm public nu este mai nesigur decât un algoritm „secret“, necunoscut publicului dar posibil cunoscut adversarului. Un algoritm foarte cunoscut, dar fără vulnerabilități cunoscute, este preferabil față de un algoritm „secret“ deoarece există șanse mult mai mari ca autorul și utilizatorul aplicației să afle despre vulnerabilități înainte ca vulnerabilitățile să fie exploatare împotriva lor.

Adesea avem $T = M$; în acest caz c_k este o funcție bijectivă (o permutare pe T). În aceste condiții, uneori rolurile funcțiilor c și d pot fi interschimbate, adică d_k să se folosească ca funcție de criptare și c_k pentru decriptare.

EXEMPLUL 6.1 (*Substituția monoalfabetică*): Considerăm un alfabet (finit) S și notăm cu n numărul de litere ($n = |S|$). De exemplu,

$$S = \{a, b, c, \dots, z\};$$

în acest caz $n = 26$. Textele clare sunt șiruri de litere din alfabet: $T = S^*$. Mulțimea textelor cifrate este identică cu mulțimea textelor clare: $M = T$. Cheile posibile sunt permutările lui S ; $|K| = n!$. Pentru un text clar $p = (s_1, s_2, \dots, s_l)$, textul cifrat este

$$c_k(p) = (k(s_1), k(s_2), \dots, k(s_l)).$$

Decriptarea se calculează

$$d_k((m_1, m_2, \dots, m_l)) = (k^{-1}(m_1), k^{-1}(m_2), \dots, k^{-1}(m_l)).$$

Criptarea și decriptarea sunt simplu de executat, chiar și manual. Cheile sunt ușor de reprezentat. Dacă alfabetul are o ordine cunoscută,

reprezentarea cheii poate consta în înșiruirea literelor în ordinea dată de permutare. De exemplu

qwertyuiopasdfghjklzxcvbnm

înseamnă $k(a) = q$, $k(b) = w$, etc. Cu această cheie, cuvântul „criptic“ devine, prin criptare, „ekohzoe“.

Să examinăm puțin siguranța. Să presupunem că un adversar încearcă decriptarea textului cifrat cu fiecare cheie posibilă. O astfel de încercare se numește *atac prin forță brută*. Să mai presupunem că adversarul reușește să verifice un miliard de chei în fiecare secundă. Deoarece numărul de chei este $26! \approx 4 \cdot 10^{26}$, adversarul ar avea nevoie în medie de 6,5 miliarde de ani pentru a găsi cheia corectă.

Pe de altă parte, într-un text în limba română, anumite litere (de exemplu e , a , t , s) apar mai frecvent decât altele. Ca urmare, permutările primelor prin funcția k vor apare în textul cifrat cu frecvență mai mare decât permutările celorlalte. Un adversar, care dispune de suficient text cifrat, va încerca doar acele chei care fac să corespundă unei litere din textul cifrat doar litere a căror frecvență normală de apariție este apropiată de frecvența de apariție a literei considerate în textul cifrat. În acest fel, numărul de încercări se reduce considerabil, astfel încât un astfel de cifru poate fi spart ușor în câteva minute.

EXEMPLUL 6.2 (*Cifrul Vernam, numit și cheia acoperitoare, engl. One time pad*): La acest cifru, $T = \{t \in \{0, 1\}^* : |t| \leq n\}$ (mulțimea șirurilor de biți de lungime mai mică sau egală cu un $n \in \mathbb{N}$ fixat), $M = T$ și $K = \{0, 1\}^n$. Funcția de criptare este

$$c_k(t_1, t_2, \dots, t_l) = (t_1 \oplus k_1, t_2 \oplus k_2, \dots, t_l \oplus k_l),$$

unde \oplus este operația *sau exclusiv*.

Decriptarea coincide cu criptarea, $d_k = c_k$.

Din punctul de vedere al siguranței, criptarea cu cheie acoperitoare este un mecanism perfect de criptare: adversarul nu poate deduce nimic din mesajul criptat (în afară de lungimea textului clar), deoarece orice text clar putea fi, cu egală probabilitate, originea textului cifrat recepționat.

Criptarea cu cheie acoperitoare este dificil de utilizat practic deoarece necesită o cheie la fel de lungă ca și mesajul de transmis și, în plus, cheia nu poate fi refolosită (dacă se transmit două mesaje folosind aceeași cheie, se pierde siguranța metodei).

6.1.2. Refolosirea cheilor

Până aici am considerat problema criptării unui singur mesaj. Utilizarea aceleiași chei pentru mai multe mesaje aduce adversarului noi posibilități de acțiune:

1. Două mesaje identice vor fi criptate identic; adversarul poate detecta astfel repetarea unui mesaj.
2. Anumite informații transmise criptat la un moment dat pot deveni publice ulterior. Adversarul poate obține astfel perechi (t_i, m_i) cu $m_i = c_k(t_i)$. Încercările de determinare a cheii de criptare sau de decriptare a unui text cifrat, pe baza informațiilor aduse de astfel de perechi text clar, text cifrat, se numește *atac cu text clar cunoscut*.
3. În anumite cazuri, adversarul poate determina emițătorul să trimită mesaje conținând părți generate de adversar. Acest lucru poate ajuta mult tentativelor de spargere de la punctul precedent. Atacul se numește *cu text clar ales*. De asemenea, ținând cont și de posibilitățile de la punctul 1, dacă adversarul bănuiește textul clar al unui mesaj, poate să încerce să-și confirme sau infirme bănuiala.
4. Anumite cifruri, de exemplu cifrul cu cheie acoperitoare, sunt ușor de atacat de un adversar dispunând de două texte cifrate cu aceeași cheie.

Punctele 2 și 3 pot fi contracarate prin anumite proprietăți ale cifrului (vezi § 6.1.3).

Pentru punctele 1 și 4, orice cifru, în forma în care este folosit în practică, mai primește în funcția de criptare un argument aleator. O parte din acest argument, numită *vector de inițializare*, are rolul de-a face ca același text clar să fie cifrat în mod diferit în mesaje diferite.

În acest caz, criptarea are forma $c : T \times K \times R \rightarrow M$ și decriptarea $d : M \times K \rightarrow T$, cu:

$$d_k(c_k(t, r)) = t, \forall t \in T, k \in K, r \in R.$$

Evident, pentru ca decriptarea să fie posibilă, informația corespunzătoare argumentului aleator trebuie să se regăsească în textul cifrat. Ca urmare, lungimea textului cifrat trebuie să fie cel puțin egală cu lungimea textului clar plus lungimea argumentului aleator.

Adesea, argumentul aleator nu este secret; ca urmare, poate fi transmis în clar. Trebuie însă ca adversarul să nu poată să controleze generarea argumentului aleator utilizat de emițător. De asemenea, nu este permis ca adversarul să mai aibă vreun control asupra conținutului textului clar după ce obține informații despre argumentul aleator ce urmează a fi folosit.

6.1.3. Problema spargerii unui cifru

Un cifru este *complet spart* dacă un adversar care nu cunoaște dinainte cheia poate decripta orice text cifrat. Dacă adversarul obține cheia, înseamnă că cifrul este complet spart.

Un cifru este *parțial spart* dacă un adversar care nu cunoaște inițial cheia poate dobândi informații despre textul clar prin observarea textului cifrat. Dacă adversarul poate decripta o parte din textul clar sau poate să verifice dacă un anumit șir apare în textul clar, înseamnă că cifrul este parțial spart.

Se poate presupune că un adversar poate estima textele clare ce ar putea fi transmise și eventual probabilitățile lor; există cazuri în care textul clar transmis este dintr-o mulțime mică, de exemplu poate fi doar *da* sau *nu*. Dacă un adversar ce a interceptat textul cifrat poate elimina anumite texte clare, sau, estimând probabilitățile diverselor chei de cifrare, poate estima probabilități, pentru textele clare, diferite față de estimările sale inițiale, înseamnă de asemenea că adversarul a extras informație din textul cifrat și în consecință cifrul este parțial spart.

EXEMPLUL 6.3: Considerăm că, din informațiile adversarului, textul clar este este cu probabilitate de 30% ION, cu probabilitate de 40% ANA și cu probabilitate de 30% DAN. De asemenea, presupunem că adversarul știe că se utilizează substituție monoalfabetică.

În momentul în care adversarul interceptează textul cifrat **AZF**, el calculează probabilitățile diverselor texte clare cunoscând textul cifrat și găsește 50% ION, 0% ANA și 50% DAN (exclue ANA deoarece ar da aceeași literă pe prima și pe ultima poziție în textul cifrat). Adversarul a dobândit o informație asupra textului clar, ceea ce înseamnă că cifrul a fost spart parțial.

Cu privire la informațiile de care dispune adversarul ce încearcă spargerea cifrului, există trei nivele posibile:

atac cu text cifrat: adversarul dispune doar de o anumită cantitate de text cifrat;

atac cu text clar cunoscut: adversarul dispune, pe lângă textul cifrat de spart, de un număr de perechi (t_i, m_i) , cu $m_i = c_k(t_i)$;

atac cu text clar ales: adversarul dispune de perechi (t_i, m_i) în care t_i este la alegerea adversarului.

Afară de cazul în care cheia se schimbă la fiecare mesaj, este necesar ca cifrul să nu poată fi spart printr-un atac cu text clar ales.

Dificultatea spargerii unui cifru este de două feluri:

- *dificultatea probabilistică sau informațională*,
- *dificultate computațională*.

Dificultatea informațională constă în faptul că pot exista mai multe perechi text clar, cheie, care ar fi putut produce textul cifrat interceptat m .

Presupunând $|T| = |M|$ și că orice bijecție $c : T \rightarrow M$ putea fi aleasă, cu egală probabilitate, ca funcție de criptare, adversarul care recepționează un text cifrat m nu poate deduce nimic cu privire la textul clar t — orice text clar avea aceeași probabilitate de a genera m . Un astfel de cifru este perfect — textul cifrat nu aduce nici o informație adversarului.

Deoarece există $(|T|)!$ bijecții posibile, lungimea necesară a cheii este $\log_2((|T|)!)$. Presupunând că T este mulțimea șirurilor de n biți, avem $|T| = 2^n$ și lungimea cheii este $\log_2((2^n)!) \text{ biți}$, lungime a cărei comportament asimptotic este de forma $\Theta(n2^n)$. Pentru $n = 20$, cheia are câțiva megabiți.

De notat că un algoritm de criptare secret nu este un cifru perfect, deoarece nu toate cele $(2^n)!$ funcții de criptare posibile au aceeași probabilitate de a fi alese.

Cifrul Vernam (vezi exemplul 6.2) este de asemenea un cifru perfect, câtă vreme cheia nu este refolosită.

În exemplul 6.3, dificultatea informațională constă în imposibilitatea adversarului de a distinge între *DAN* și *ION*

Incertitudinea adversarului asupra textului clar este cel mult egală cu incertitudinea asupra cheii. De aici rezultă că, pentru a obținerea unui cifru perfect, numărul de biți ai cheii trebuie să fie mai mare sau egal cu numărul de biți de informație din mesaj. Cifrul Vernam este în același timp perfect (sub aspectul dificultății informaționale a spargerii) și optim din punctul de vedere al lungimii cheii.

Dificultatea computațională constă în imposibilitatea adversarului de a deduce informații asupra textului clar cu un efort computațional rezonabil.

Un prim lucru care se cere de la un cifru este ca, dându-se un număr de perechi text clar – text cifrat, să nu existe o metodă rapidă de a determina cheia.

Un *atac prin forță brută* (engl. *brute force attack*) constă în a decripta textul cifrat folosind toate cheile posibile și a verifica dacă se obține textul clar (sau un text clar inteligibil, dacă textul clar adevărat nu este cunoscut dinainte). Fezabilitatea unui atac prin forță brută depinde direct de lungimea cheii (de fapt, de numărul de chei posibile). Pentru o cheie de 56 de biți (exemplu cifrul DES), un atac prin forță brută este perfect posibil, la viteza

actuală necesitând un efort în jur de un an-calculator. Un atac prin forță brută este nefezabil deocamdată de la 80 de biți în sus; se consideră că va fi fezabil în jurul anului 2015. De la 128 de biți în sus atacul prin forță brută necesită, din cauza unor limitări fizice teoretice, o cantitate de energie comparabilă cu producția mondială pe câteva luni; o astfel de cheie este puțin probabil că va putea fi spartă vreodată prin forță brută.

Un cifru se consideră a fi vulnerabil în momentul în care se descoperă o metodă de decriptare a unui mesaj semnificativ mai eficientă decât un atac prin forță brută. Inexistența unei metode eficiente de spargere nu este niciodată demonstrată; în cel mai bun caz se demonstrează că spargerea unui cifru este cel puțin la fel de dificilă ca rezolvarea unei anumite probleme de matematică, problemă cunoscută de multă vreme dar fără rezolvare eficientă cunoscută. Acest din urmă tip de demonstrație se aplică mai mult la cifrurile asimetrice din § 6.1.5; problemele de matematică sunt de exemplu descompunerea în factori primi a unui număr mare — de ordinul sutelor de cifre — sau logaritmul discret — rezolvarea în $x \in \{0, \dots, p-1\}$ a ecuației $a^x = b \pmod{p}$, cu p număr prim mare.

Pentru un cifru bloc (un cifru care criptează independent blocuri de text clar de o anumită lungime fixă), dimensiunea blocului trebuie să fie mare pentru a face repetările blocurilor suficient de rare. Dacă dimensiunea blocului este de n biți, există 2^n posibilități pentru conținutul unui bloc. Considerând o distribuție uniformă a conținutului fiecărui bloc, un șir de $2^{n/2}$ blocuri are probabilitate cam $1/2$ să aibă cel puțin două blocuri cu conținut identic. (Acest fapt este cunoscut ca *paradoxul zilei de naștere*: într-un grup de 23 de persoane, probabilitatea să existe două dintre ele născute în aceeași zi din an este peste 50%; în general, într-un grup de \sqrt{k} numere aleatoare având k valori posibile, probabilitatea ca cel puțin două să fie egale este în jur de $1/2$).

Ca o consecință, dimensiunea n a blocului trebuie să fie suficient de mare și cheia să fie schimbată suficient de des, astfel încât numărul de blocuri criptate cu o cheie dată să fie mult mai mic decât $2^{n/2}$. În majoritatea cazurilor, valoarea minimă rezonabilă pentru n este 64 de biți. La această lungime, repetarea unui bloc de text cifrat este probabil să apară începând de la 2^{32} blocuri, adică 32 GiB.

6.1.4. Algoritmi de criptare utilizați în practică

Cifrurile mai cunoscute și utilizate pe scară mai largă în practică sunt date în tabela 6.1. Există două tipuri de cifruri: *cifru bloc* (engl. *block cipher*), care criptează câte un bloc de date de lungime fixată (de obicei 64, 128 sau, eventual, 256 de biți), și *cifru flux* (engl. *stream cipher*), care criptează mesaje

de lungime arbitrară și produc biții textului cifrat pe măsură ce primesc biții corespunzători din textul clar.

Pentru a cripta un text de lungime arbitrară, cu ajutorul unui cifru bloc, există câteva metode standard, pe care le vom descrie în continuare. În cele ce urmează, notăm cu n lungimea blocului, în biți.

ECB — Electronic Code Book: Textul clar se împarte în blocuri de lungime n . Ultimul bloc se completează la lungimea n ; biții adăugați pot fi zerouri, biți aleatori sau se pot utiliza alte scheme de completare. Fiecare bloc se criptează apoi independent de celelalte (vezi fig. 6.1).

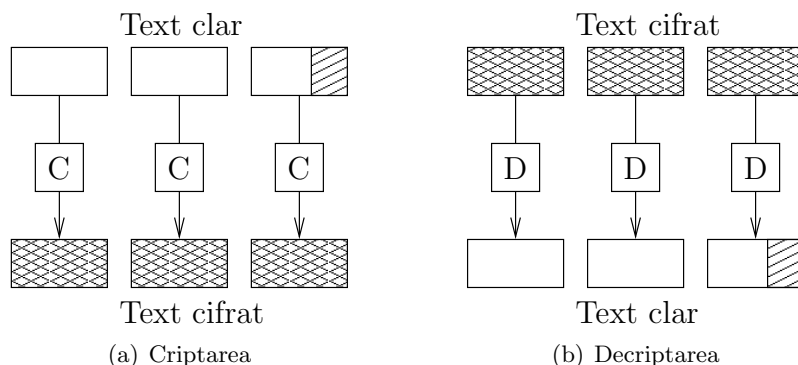


Figura 6.1: Criptarea în mod ECB

Metoda ECB nu se recomandă deoarece pentru o cheie fixă același text clar se transformă în același text cifrat.

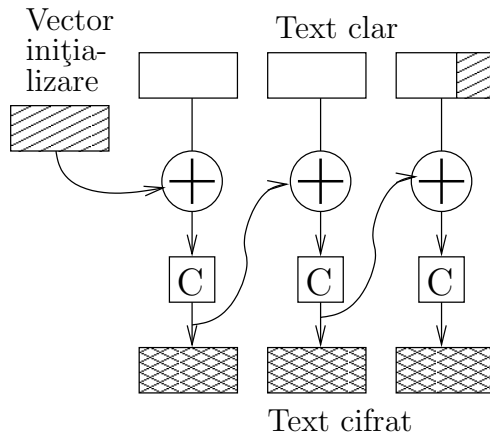
O altă critică citată frecvent este că un adversar care permută blocurile de text cifrat va obține permutarea blocurilor corespunzătoare de text clar, chiar dacă nu înțelege nimic din textul cifrat. Deși afirmația este adevărată, critica este nefondată întrucât un cifru nu are ca scop protejarea integrității mesajelor.

CBC — Cipher Block Chaining: (Vezi fig. 6.2.) Ca și la ECB, textul clar se împarte în blocuri și ultimul bloc se completează cu biți aleatori. În plus, se alege un șir de n biți aleatori; acesta se numește *vector de inițializare*. Vectorul de inițializare se transmite de obicei separat. Se efectuează *xor* pe biți între vectorul de inițializare și primul bloc de text clar; rezultatul se cifrează și se trimite. Apoi, se face *xor* între fiecare bloc de text clar și blocul precedent de text cifrat, și rezultatul se cifrează și se transmite destinatarului.

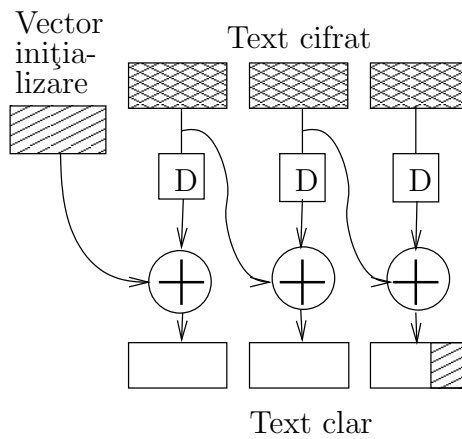
Față de ECB, metoda CBC face ca un același bloc de text clar să se cripteze diferit, în funcție de vectorul de inițializare utilizat. Dacă

Nume	lungime bloc	lungime cheie	observații
DES	64	56	A fost utilizat pe scară destul de largă, fiind susținut ca standard de către guvernul Statelor Unite ale Americii. În prezent este nesigur datorită lungimii mici a cheii (a fost deja spart prin forță brută). Au existat și speculații cum că ar fi fost proiectat astfel încât să fie ușor de spart de către cei care ar cunoaște niște detalii de proiectare.
3DES	64	112 sau 168	Constă în aplicarea de 3 ori succesiv a cifrului DES, cu 2 sau toate 3 cheile distincte. A fost creat pentru a nu inventa un cifru total nou, dar făcând imposibilă spargerea prin forță brută.
AES	128	128, 192 sau 256	Desemnat, în urma unui concurs, ca nou standard utilizat de guvernul american. Proiectat de doi belgieni, Joan Daemen și Vincent Rijmen și publicat sub numele <i>rijndael</i> .
CAST-128	64	între 40 și 128 biți	Creat de Carlisle Adams și Stafford Tavares în 1996.
Blowfish	64	până la 448 biți	Creat de Bruce Schneier în 1993.
Twofish	128	până la 256 biți	Creat de Bruce Schneier și alții și a participat la concursul pentru AES.
Serpent	128	128, 192 sau 256	Creat de Ross Anderson, Eli Biham și Lars Knudsen; candidat pentru AES.
RC6	128	128, 192 sau 256	Creat de Ronald Rivest; candidat pentru AES; patentat în favoarea firmei RSA Security.
RC4	flux	până la 256 biți	Creat de Ronald Rivest în 1987; foarte rapid; are câteva slăbiciuni, care pot fi contracarate prin artificii legate de modul de utilizare.

Tabelul 6.1: Cifruri mai cunoscute.



(a) Criptarea



(b) Decriptarea

Figura 6.2: Criptarea în mod CBC

vectorul de inițializare este ales aleator, repetările unui bloc de text cifrat vor fi extrem de rare (imposibil de exploatat de adversar).

Vectorul de inițializare trebuie ales satisfăcând :

- să fie distribuit uniform și necorelat cu textul clar sau alți vectori de inițializare;
- să nu poată fi controlat de adversar;
- să nu poată fi aflat de adversar cât timp adversarul ar putea influența textul clar, pentru a împiedica un atac cu text clar ales.

Vectorul de inițializare se construiește utilizând una din următoarele variante:

- se generează cu un generator de numere aleatoare criptografic (vezi § 6.4) și se transmite în clar înaintea mesajului;
- se stabilește prin metode asemănătoare cu stabilirea cheii (vezi § 6.3);
- dacă se transmit mai multe mesaje unul după altul, vectorul de inițializare pentru un mesaj se ia ca fiind ultimul bloc al mesajului precedent (ca la înlănțuirea blocurilor în cadrul aceluiași mesaj). Pentru a împiedica un atac cu text clar ales, dacă textul clar al unui mesaj este format după expedierea mesajului precedent este necesar trimiterea unui mesaj care să fie ignorat de destinatar și cu conținut aleator.

CFB — Cipher Feedback: CFB și următorul mod, OFB, sunt utilizate în special acolo unde mesajele sunt mult mai scurte decât dimensiunea blocului, însă emițătorul transmite aceluiași receptor o secvență mai lungă de mesaje (presupunem că un mesaj nu poate fi grupat împreună cu următorul deoarece, de exemplu, un mesaj depinde de răspunsul receptorului la mesajul precedent; prin urmare, un mesaj nu este disponibil pentru criptare înainte ca mesajul precedent să fie criptat în totalitate și trimis).

CFB criptează fragmente de text clar de dimensiune fixă m . Dimensiunea m a fragmentului trebuie să îndeplinească o singură restricție, și anume să fie un divizor al dimensiunii n a blocului cifrului. Se poate lua $m = 8$ și atunci cifrul criptează câte un caracter. CFB funcționează astfel (vezi fig. 6.3): Emițătorul generează aleator un vector de inițializare de n biți, pe care îl transmite receptorului și îl încarcă totodată într-un registru de deplasare. Apoi, pentru fiecare caracter de

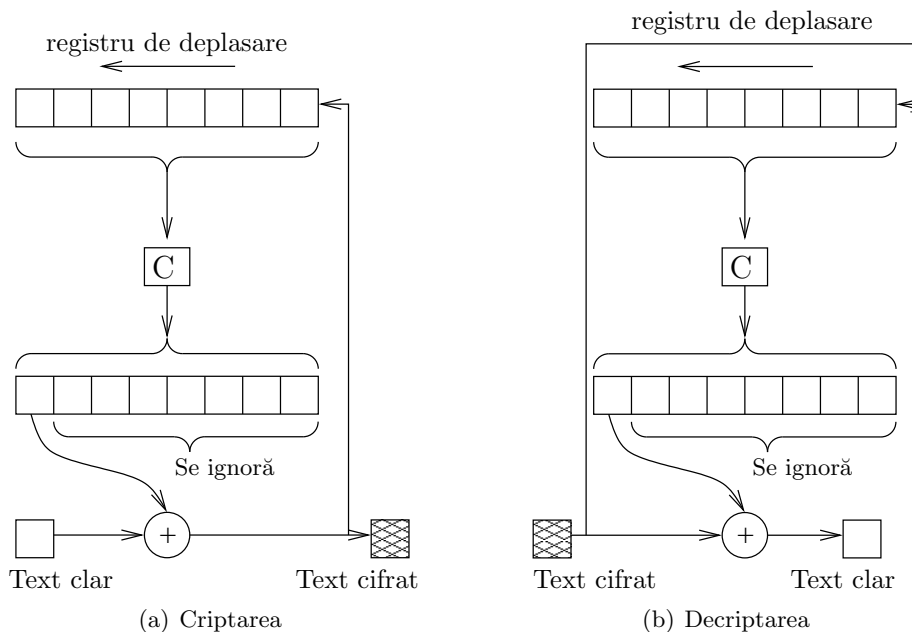


Figura 6.3: Criptarea în mod CFB

criptat, emițătorul:

- criptează conținutul registrului de deplasare utilizând cheia secretă,
- execută xor pe biți între următorii m biți din textul clar și primii m biți din rezultatul criptării,
- transmite ca text cifrat rezultatul pasului precedent,
- deplasează conținutul registrului de deplasare cu m biți spre stânga,
- introduce, pe pozițiile cele mai din dreapta ale registrului de deplasare, m biți de text cifrat produs.

CFB are o proprietate interesantă de *autosincronizare*: dacă la un moment dat, din cauza unor erori de transmisie, se pierde sincronismul dintre emițător și receptor, sincronismul se reface automat după n biți.

De remarcat, de asemenea, că decriptarea CFB utilizează tot funcția de criptare a cifrului bloc.

OFB — Output Feedback: OFB este un mecanism asemănător cu cifrul Vernam (cu cheie acoperitoare) (exemplul 6.2), însă cheia este un șir pseudoaleator generat cu un algoritm de criptare. Primii n biți din șirul pseudoaleator se obțin criptând vectorul de inițializare, următorii n biți

se obțin criptând precedenții n biți pseudoaleatori, ș. a. m. d.

La OFB utilizarea unui vector de inițializare aleator este chiar mai importantă decât la celelalte moduri de criptare, întrucât re folosirea unui vector de inițializare conduce la repetarea șirului pseudoaleator (cheia Vernam), rezultând un cifru relativ ușor de spart.

CTR — Counter: Se construiește similar cu OFB, însă șirul pseudoaleator se obține criptând numerele v , $v + 1$, $v + 2$, etc., reprezentate pe n biți, v fiind vectorul de inițializare. Modul CTR este foarte asemănător cu OFB, însă are avantajul că destinatarul poate decripta un fragment de mesaj fără a decripta tot mesajul până la fragmentul dorit. Acest fapt îl face potrivit pentru criptarea fișierelor pe disc. Totuși, deoarece cifrul Vernam aflat la bază este vulnerabil unui adversar având la dispoziție două texte clare criptate cu aceeași cheie, metoda este vulnerabilă dacă adversarul are posibilitatea de-a obține două variante ale unui fișier.

6.1.5. Criptografie asimetrică (cu cheie publică)

Intuitiv, s-ar putea crede că, dacă funcția de criptare este complet cunoscută (inclusiv cheia), funcția inversă — decriptarea — este de asemenea calculabilă în mod rezonabil. În realitate, există funcții de criptare (injective) a căror cunoaștere nu permite decriptarea în timp rezonabil. În esență, ideea este că, deși $m = c(t)$ este rezonabil de ușor de calculat, determinarea lui t din ecuația $c(t) = m$ nu se poate face mult mai rapid decât prin încercarea tuturor valorilor posibile pentru t .

Pentru ca o astfel de metodă de criptare să fie utilă, trebuie ca destinatarul autorizat al mesajului să-l poată totuși decripta în timp rezonabil. Pentru aceasta, se cere ca funcția de criptare c să poată fi inversată ușor de către cineva care cunoaște o anumită informație, dificil de dedus din c . Această informație va fi făcută cunoscută doar destinatarului mesajului criptat. De fapt, în cazul criptografiei asimetrice, destinatarul mesajului este chiar autorul acestei informații secrete și a funcției de criptare.

Ca și în cazul criptografiei simetrice, funcția de criptare c primește un parametru, cheia k_c , numită *cheie de criptare* sau *cheie publică*. Astfel, criptarea se calculează $m = c_{k_c}(t)$.

Pentru decriptare, vom nota cu d algoritmul general și cu k_d informația care permite decriptarea în timp rezonabil. Decriptarea o scriem $t = d_{k_d}(m)$. k_d o numim *cheie de decriptare* sau *cheie secretă*. Fiecare cheie secretă k_d este asociată unei anumite chei publice k_c , putând servi la decriptarea mesajelor criptate doar cu cheia publică pereche.

Evident, cunoscând cheia publică k_c este posibil, însă trebuie să fie dificil computațional, să se calculeze cheia secretă k_d corespunzătoare. Pentru ca sistemul de criptare să fie util mai este necesar să existe un procedeu eficient (computațional) de generare a unei perechi de chei (k_c, k_d) aleatoare.

Un *sistem criptografic asimetric* (sau *cifru asimetric* sau *cifru cu cheie publică*) este un ansamblu format din algoritmi de criptare c și decriptare d și un algoritm de generare aleatoare a perechilor de chei (k_c, k_d) .

Pentru ca sistemul criptografic să fie sigur trebuie ca rezolvarea ecuației $c_{k_c}(t) = m$ cu necunoscuta t să fie dificilă computațional. Implicit, determinarea cheii secrete k_d corespunzătoare unei chei publice k_c trebuie de asemenea să fie dificilă computațional.

EXEMPLUL 6.4 (*Cifrul RSA*): Generarea cheilor se face astfel:

- se generează numerele prime p și q (de ordinul a 500 cifre zecimale fiecare);
- se calculează $n = pq$ și $\phi = (p - 1)(q - 1)$;
- se generează aleator un număr $e \in \{2, 3, \dots, \phi - 1\}$, relativ prim cu ϕ ;
- se calculează d cu proprietatea că $ed \equiv 1 \pmod{\phi}$ (utilizând algoritmul lui Euclid).

Cheia publică este $k_c = (n, e)$, iar cheia secretă este $k_d = (n, d)$.

Spațiul textelor clare și spațiul textelor cifrate sunt

$$P = M = \{0, 1, \dots, n - 1\}.$$

Criptarea și decriptarea sunt:

$$c_{k_c}(p) = p^e \pmod{n} \quad (6.1)$$

$$d_{k_d}(m) = m^d \pmod{n} \quad (6.2)$$

Cifrul a fost inventat de Rivest, Shamir și Adelman în 1977. Numele RSA vine de la inițialele autorilor.

6.1.5.1. Utilizarea criptografiei asimetrice

Pentru pregătirea comunicației, receptorul generează o pereche de chei (k_c, k_d) și face publică k_c . Emițătorul poate cripta un mesaj, folosind k_c , și numai posesorul lui k_d îl va putea decripta. Notăm că odată criptat un mesaj, acesta nu mai poate fi decriptat nici măcar de autorul său (deși autorul — și de altfel oricine — poate verifica dacă un text cifrat dat corespunde sau nu unui text clar dat).

Dacă se dorește comunicație bidirecțională, se utilizează câte o pereche de chei (k_c, k_d) distinctă pentru fiecare sens.

O aceeași pereche de chei poate fi utilizată de o entitate pentru toate mesajele pe care le primește, indiferent cu câți parteneri comunică. Astfel, fiecare entitate își stabilește o pereche de chei din care cheia publică o transmite tuturor partenerilor de comunicație și cheia secretă o folosește pentru a decripta mesajele trimise de toți către ea. Pentru comparație, în criptografia simetrică, fiecare pereche de parteneri ce comunică trebuie să aibă propria cheie secretă.

Un sistem criptografic asimetric este în esență un cifru bloc. Pentru a cripta o cantitate arbitrară de text clar, se pot utiliza modurile ECB sau CBC. Modurile CFB, OFB și CTR nu sunt utilizabile deoarece utilizează doar funcția de criptare, care în cazul criptografiei asimetrice este publică.

Algoritmii criptografici asimetrici sunt mult mai lenți decât cei simetrici. Din acest motiv, datele propriu-zise se criptează de obicei cu algoritmi simetrici, iar cheia de criptare pentru date se transmite utilizând criptografie asimetrică (vezi și § 6.3).

6.2. Autentificarea mesajelor

Autentificarea mesajelor este un mecanism prin care destinatarul unui mesaj poate verifica faptul că autorul mesajului este o anumită entitate și că mesajul nu a fost modificat de altcineva.

Verificarea autenticității unui mesaj constă în aplicarea de către receptor a unui *test de autenticitate* asupra mesajului primit. Un test de autenticitate trebuie să îndeplinească două proprietăți:

- orice mesaj autentic să treacă testul;
- pentru un mesaj neautentic, produs cu un efort computațional rezonabil, probabilitatea ca mesajul să treacă testul să fie extrem de mică.

Există două nivele distincte de „spargere” a unui test de autenticitate:

- *fals existent* (engl. *existential forgery*): posibilitatea ca un adversar să genereze un mesaj neautentic care să treacă testul de autenticitate. Mesajul astfel produs nu trebuie să aibă un conținut inteligibil; trebuie doar să fie acceptat de algoritmul de autentificare.
- *fals ales* (engl. *chosen forgery*): în plus față de falsul existent, conținutul mesajului neautentic este (total sau în mare parte) la alegerea adversarului.

Evident, un mesaj, acceptat ca autentic o dată, va fi acceptat ca autentic și în cazul unei repetări ulterioare. Prevenirea unor atacuri bazate pe

repetarea unor mesaje anterioare este o problemă separată și va fi studiată în § 6.2.4.

În studiul metodelor de autentificare a mesajelor, presupunem că mesajul de transmis nu este secret. Aceasta deoarece în practică apare frecvent necesitatea ca un mesaj public să poată fi testat de oricine în privința autenticității. De exemplu, textul unei legi este o informație publică, dar un cetățean ar trebui să poată verifica dacă textul ce i-a parvenit este textul autentic emis de autoritatea abilitată.

Remarcăm de asemenea că faptul că un mesaj criptat utilizând un algoritm simetric poate fi decriptat de către receptor (utilizând cheia secretă) și este inteligibil nu e o garanție privind autenticitatea mesajului. Într-adevăr, pentru unele metode de criptare, cum ar fi modul OFB al oricărui cifru bloc, un adversar poate opera modificări asupra textului cifrat cu efecte previzibile asupra textului clar, chiar dacă nu cunoaște efectiv textul clar. Din acest motiv, metodele de asigurare a confidențialității se separă de metodele de control a autenticității.

6.2.1. Funcții de dispersie criptografice

În general (nu neapărat în criptografie), prin *funcție de dispersie* (engl. *hash function*) se înțelege o funcție h care asociază unui șir de biți t , de lungime oricât de mare, o valoare întreagă într-un interval de forma $[0, 2^n)$ cu n fixat (sau echivalent, un șir de biți de lungime n), satisfăcând condiția că, pentru șirurile care apar în problema unde se folosește funcția de dispersie, două șiruri distincte să nu aibă aceeași valoare a funcției de dispersie cu probabilitate semnificativ mai mare de 2^{-n} . Valoarea funcției de dispersie aplicată unui șir se numește *dispersia* aceluși șir.

O *funcție de dispersie criptografică* este o funcție de dispersie care are anumite proprietăți suplimentare, dintre cele enumerate în continuare:

1. *rezistența la preimage* (engl. *preimage resistance*): dându-se $h(t)$, să fie dificil de regăsit t . Eventual, dificultatea să se păstreze chiar în cazul cunoașterii unei părți din t .
2. *rezistența la a doua preimage* (engl. *second preimage resistance*): dându-se un șir t , să fie dificil de găsit un al doilea șir t' , cu $t' \neq t$, astfel încât $h(t) = h(t')$.
3. *rezistența la coliziuni* (engl. *collision resistance*): să fie dificil de găsit două șiruri distincte t_1 și t_2 ($t_1 \neq t_2$), astfel încât $h(t_1) = h(t_2)$.

De remarcat că cele trei condiții sunt diferite. Totuși, condiția de rezistență la coliziuni implică rezistența la a doua preimage. De asemenea,

majoritatea funcțiilor rezistente la coliziuni satisfac și condiția de rezistență la preimagine.

Numărul de biți n ai dispersiei trebuie să fie suficient de mare pentru a împiedica căutarea unei coliziuni prin forță brută. Conform paradoxului zilei de naștere, există șanse mari de găsire a unei coliziuni într-o mulțime de $2^{n/2}$ intrări. Pentru a face impractic un atac prin forță brută, trebuie ca $n/2 \geq 64$ (și mai bine $n/2 \geq 80$), de unde $n \geq 128$ sau mai bine $n \geq 160$.

Funcțiile de dispersie mai cunoscute sunt descrise în tabelul 6.2.

Nume	lungime	observații
MD5	128	Creată de Ronald Rivest în 1991. Este extrem de răspândită, însă câteva slăbiciuni descoperite recent o fac destul de nesigură.
SHA1	160	Dezvoltată de NSA (National Security Agency, SUA). Deocamdată este mai sigură decât MD5, dar are deja câteva slăbiciuni.
RIPMD-160	160	Dezvoltată la Katholieke Universiteit Leuven în 1996.

Tabelul 6.2: Funcții de dispersie criptografice.

6.2.1.1. Utilizarea funcțiilor de dispersie

Presupunem existența între emițător și receptor a două canale de transmitere a informației: un canal principal nesigur și un canal sigur dar cu capacitate foarte redusă. Ca exemplu practic, canalul nesigur este Internet-ul, iar canalul sigur este un bilet scris sau o convorbire telefonică.

Presupunem de asemenea că h este o funcție de dispersie rezistentă la a doua preimagine și preferabil rezistentă la coliziuni.

Emițătorul unui mesaj t calculează $s = h(t)$. Apoi, transmite t prin canalul principal și transmite s prin canalul sigur. Receptorul testează dacă $h(t) = s$. Un adversar care ar modifica t în t' ar trebui să găsească un t' cu $h(t') = h(t)$ pentru a păcăli receptorul; acest lucru este nefezabil în virtutea proprietății de rezistență la a doua preimagine a funcției de dispersie h .

Există situații practice în care t este (parțial) la dispoziția adversarului. De exemplu, presupunem că secretara redactează un mesaj t la cererea șefului, secretara putând alege formularea exactă a mesajului t . Șeful își exprimă acordul asupra mesajului calculând și trimițând destinatarului $s = h(t)$. Dacă adversarul este secretara, ea nu se găsește în situația de-a crea un t' satisfăcând $h(t') = h(t)$ pentru t fixat (adică de-a crea a doua preimagine) ci

este în situația de-a crea t și t' distincte cu $h(t) = h(t')$ (adică de-a găsi o coliziune). Din acest motiv, o funcție de dispersie utilizată pentru controlul autenticității mesajelor se cere să fie rezistentă la coliziuni.

Există pe sistemele Linux comenzile `md5sum` și `sha1sum` care calculează și afișează dispersia `md5` respectiv `sha1` a conținutului unui fișier. Dispersia este afișată în hexa. Dacă notăm într-un loc sigur dispersia unui fișier, putem controla ulterior dacă fișierul a fost sau nu modificat între timp.

6.2.2. Funcții de dispersie cu cheie

O funcție de dispersie cu cheie (engl. *keyed hash function*), numită și *MAC* (*message authentication code*), este o funcție de dispersie $h_k(t)$, parametrizată cu o cheie k , având proprietatea că, pentru cineva care nu cunoaște dinainte cheia k , este nefezabil computațional să obțină o (nouă) pereche (s, t) în care $s = h_k(t)$, chiar dacă cunoaște un număr de perechi (s_i, t_i) cu $s_i = h_k(t_i)$.

O funcție de dispersie cu cheie se utilizează astfel: Mai întâi, emițătorul și receptorul se înțeleg asupra unei chei secrete k (de exemplu conform metodelor din § 6.3). La trimiterea unui mesaj t , emițătorul calculează $s = h_k(t)$ și trimite împreună perechea (s, t) . Receptorul testează dacă $s = h_k(t)$.

Orice autentificare prin dispersie cu cheie este *a priori* vulnerabilă la un atac numit *atac prin reflexie*, descris în continuare. Notăm cu A și B cele două părți care comunică și cu k cheia de dispersie utilizată pentru autentificarea mesajelor. Un adversar activ poate intercepta un mesaj trimis de A către B și să-l trimită înapoi lui A . Dacă aceeași cheie k este utilizată pentru autentificarea ambelor sensuri de comunicație (și de la A la B , și de la B la A) și dacă mesajele au același format, atunci A acceptă mesajul ca venind de la B . Pentru a preveni un atac prin reflexie, există două soluții:

- Se utilizează chei distincte pentru cele două sensuri.
- Fiecare mesaj conține numele entității emițătoare. Eventual, numele entității nu apare efectiv în mesajul transmis, dar participă la calculul dispersiei: $s = h_k(t \cdot A)$ și A trimite spre B perechea (t, s) .

Argumente similare cu cele privind dimensiunea blocurilor la cifrurile bloc și dimensiunea cheii de cifrare conduc la cerințe pentru împiedicarea atacurilor prin forță brută: dimensiunea cheii și dimensiunea dispersiei de minim 64 de biți, preferabil 80 de biți.

O construcție uzuală pentru funcții de dispersie cu cheie pornind de la funcții de dispersie rezistente la coliziuni și rezistente la preimagine este

(conform [RFC 2104, 1997]):

$$h_k(m) = \text{hash}(K \oplus \text{opad} \cdot \text{hash}(K \oplus \text{ipad} \cdot m))$$

unde:

- \cdot reprezintă concatenarea,
- \oplus este operația *sau exclusiv*,
- hash este funcția de dispersie criptografică (de exemplu *md5* sau *sha1*),
- K este cheia k completată la o lungime B aleasă în funcție de anumite particularități ale funcției de dispersie de la bază; pentru *md5* și *sha1*, B se ia de 64 de octeți.
- *ipad* și *opad* sunt șiruri obținute prin repetarea de B ori a octetului cu valoarea (hexa) 36, respectiv 5C.

Rezultatul funcției hash se poate trunchia la lungime mai mică (notăm că funcția de dispersie hash dă 128–160 biți, iar pentru o dispersie cu cheie sunt suficienți 64–80 de biți). Trunchierea are ca avantaj micșorarea cantității de informație pusă la dispoziția adversarului.

O construcție uzuală pentru funcții de dispersie cu cheie pornind de la un cifru bloc este următoarea:

- se completează mesajul la un număr întreg de blocuri;
- se execută o criptare în mod CBC cu un vector de inițializare zero (sau inițializat cu dispersia mesajului precedent);
- rezultatul criptării ultimului bloc se criptează utilizând o a doua cheie (cheia funcției de dispersie este considerată ca fiind concatenarea celor două chei de criptare), rezultând valoarea dispersiei.

6.2.3. Semnătura digitală

Semnătura digitală este o construcție similară dispersiei cu cheie, studiată în paragraful precedent. Construcția este însă asimetrică, utilizând chei diferite pentru crearea dispersiei (numită, în acest caz, semnătură) și, respectiv, pentru verificare dispersiei. Astfel, relația dintre semnătura digitală și dispersia cu cheie este similară cu cea dintre criptografia asimetrică și criptografia simetrică.

O schemă de semnătură digitală are următoarele elemente:

- un algoritm prin care se poate genera aleator o pereche de chei (k_s, k_v) , unde k_s este *cheia secretă* sau *cheia de semnătură*, iar k_v este *cheia publică* sau *cheia de verificare*.

- o funcție de semnare h ;
- o funcție de verificare v .

În faza pregătitoare, autorul de mesaje semnate generează o pereche de chei (k_s, k_v) și transmite cheia publică k_v receptorului sau receptoarelor. La transmiterea cheii publice, trebuie utilizat un canal sigur, astfel încât cheia să nu poată fi modificată în timpul transmisiei.

Autorul mesajului t crează *semnătura* $s = h_{k_s}(t)$ și transmite perechea (s, t) . Receptorul verifică dacă $v_{k_v}(t, s) = \text{true}$.

Așa cum se vede, semnătura s depinde și de mesajul de semnat t și de semnatarul acestuia (mai exact de cheia k_s). Ca urmare, o semnătură nu poate fi tăiată de pe un mesaj și plasată pe alt mesaj.

Unii algoritmi de semnătură digitală necesită un al treilea argument pentru funcția de semnătură; acest argument trebuie să fie un număr aleator. În acest caz există mai multe semnături valide pentru un același mesaj.

O posibilitate de construcție pentru semnătură este pe baza unui mecanism de criptare asimetric în care criptarea este bijectivă; de exemplu RSA are această proprietate. Construcția simplificată este:

$$\begin{aligned} h_{k_s}(t) &= d_{k_s}(t) \\ v_{k_v}(t, s) &= (c_{k_v}(s) = t) \end{aligned}$$

Construcția de mai sus se bazează pe ne fezabilitatea calculului lui $s = d_{k_s}(t)$ fără cunoașterea lui k_s . Totuși, construcția aceasta permite adversarului să producă un fals existent: un adversar poate alege aleator un s și calcula $t = c_{k_v}(s)$.

O îmbunătățire a semnăturii electronice de mai sus este să nu se aplice d_{k_s} direct asupra lui t ci asupra unei dispersii rezistente la preimagine și la coliziuni a lui t . Metoda are două avantaje, pe de o parte că algoritmul de criptare asimetric, lent, se aplică asupra dispersiei și nu asupra întregului mesaj (dispersia se calculează mai repede decât criptarea asimetrică), iar pe de altă parte se împiedică falsul existent deoarece chiar dacă adversarul calculează $c_{k_v}(s)$ nu poate găsi un mesaj t cu dispersia astfel fixată.

Semnătura digitală asigură nu doar autentificarea mesajelor, ci și nonrepudiabilitatea mesajelor. Acest lucru se întâmplă deoarece cheia de semnătură este cunoscută doar de către emițător. Ca urmare, doar emițătorul poate genera semnătura. Prin urmare, prezența unei semnături verificabile atestă faptul că documentul a fost produs de emițător. Funcțiile de dispersie cu cheie nu realizează (direct) mesaje nerepudiabile deoarece receptorul poate produce mesaje semnate la fel de bine ca și emițătorul.

6.2.4. Verificarea prospețimii mesajelor

Este adesea necesar ca receptorul să poată distinge între un mesaj (autentic) „nou“ și o copie a unui mesaj mai vechi. De exemplu, dacă mesajul cere destinatarului să execute o operație neidempotentă, cum ar fi să transfere o sumă de bani dintr-un cont în altul, este necesar ca destinatarul să accepte mesajul doar o singură dată.

O copie a unui mesaj „vechi“ este identică cu originalul din momentul când acesta era „nou“. Ca urmare, un test de autenticitate nu detectează niciodată vechimea mesajului.

Notăm că testul de prospețime nu poate consta în simpla verificare dacă un mesaj este identic cu vreunul dintre mesajele anterioare. Aceasta deoarece, pe de o parte, producerea unui mesaj identic cu un mesaj anterior este perfect legitimă (de exemplu, se poate cere un nou transfer, constând în aceeași sumă de bani către același destinatar), iar pe de altă parte, memorarea tuturor mesajelor deja primite nu este fezabilă.

Soluțiile problemei verificării prospețimii sunt similare cu metodele de transmisie sigură (§ 4.3), cu diferența că trebuie să reziste la atacuri voite, nu numai la disfuncționalități întâmplătoare.

Ideea este să introducem în mesajul autentificat un „identificator de mesaj“ care să fie diferit de la un mesaj la altul și asupra căruia să se execute de fapt testul de prospețime. Un astfel de element se numește *număr unic* (engl. *nonce*, de la *number (used) once*).

Numărul unic poate fi:

un număr de ordine: Emițătorul ține evidența unui număr curent de ordine. Pentru fiecare mesaj, emițătorul scrie numărul curent în mesaj și incrementează apoi numărul curent. Receptorul ține de asemenea evidența numărului curent de ordine. La fiecare mesaj primit, verifică dacă numărul din mesaj coincide cu numărul curent de ordine; în caz contrar mesajul nu este acceptat. După acceptarea unui mesaj, numărul curent de ordine este incrementat. Remarcăm că numărul de ordine poate fi omis din mesajul transmis efectiv; el trebuie doar să participe la calculul semnăturii mesajului.

Metoda are două neajunsuri: necesită menținerea pe termen lung a numerelor de ordine curente și necesită un contor separat de număr de ordine pentru fiecare partener de comunicație.

ora curentă: Emițătorul scrie, în mesajul autentificat, ora curentă. Receptorul consideră mesajul proaspăt dacă ora din mesaj coincide cu ora curentă a receptorului. Din păcate, receptorul este nevoit să accepte un decalaj de cel puțin câteva zecimi de secundă, deoarece ceasurile nu sunt

perfect sincronizate și deoarece transportul mesajului nu este instantaneu. În interiorul acestui decalaj admis, adversarul poate trimite copii ale mesajului, copii ce vor fi acceptate ca proaspete de destinatar.

Pentru corectarea acestei probleme, mecanismul se face astfel: Emițătorul se asigură că două mesaje distincte vor avea numărul unic distinct. Pentru aceasta, fie rezoluția marcajului de timp se face mai fină decât timpul necesar emiterii unui mesaj, fie emițătorul mai ține un contor care se incrementează la fiecare mesaj trimis și făcut astfel încât să nu se reseteze înainte ca marcajul de timp să treacă la următoarea valoare. Receptorul memorează numerele unice ale mesajelor primite, pe durata cât marcajul de timp din mesaj este acceptabil de către testul de prospețime (de exemplu, dacă mesajul este trimis la ora 08:12:45 și testul de prospețime acceptă un decalaj de 10 secunde, numărul unic al mesajului va fi păstrat până la ora 08:12:55). La primirea unui mesaj, receptorul verifică dacă marcajul de timp este actual (în interiorul intervalului acceptabil) și dacă numărul unic nu este identic cu cel al unuia dintre mesajele memorate.

Utilizarea orei la verificarea prospețimii necesită un mecanism sigur care să mențină sincronismul ceasurilor dispozitivelor care comunică.

un număr (aleator) ales de receptor: Receptorul care așteaptă un mesaj trimite emițătorului un număr aleator proaspăt generat. Numărul aleator trebuie să aibă cel puțin 64–80 biți, pentru ca să nu se repete (decât cu probabilitate neglijabil de mică) și să nu poată fi prezis de către adversar. Emițătorul adaugă numărul la mesajul trimis. Receptorul acceptă un mesaj ca proaspăt doar dacă numărul aleator din mesaj coincide cu numărul aleator tocmai trimis. Ca și pentru varianta cu număr de ordine, numărul aleator nu este necesar să fie inclus în mesaj; este suficient să participe la calculul semnăturii mesajului.

Neajunsul principal al metodei este necesitatea de-a transfera numărul aleator dinspre receptor spre emițător. În plus, este necesar ca receptorul să știe că urmează să primească un mesaj, ceea ce necesită adesea încă un mesaj (emițătorul spune *vezi că vreau să-ți spun ceva*, receptorul răspunde trimițând numărul aleator și, în final, emițătorul trimite mesajul propriu-zis). Acest lucru face aplicarea metodei scumpă și în anumite cazuri imposibilă — de exemplu metoda nu este aplicabilă pentru securizarea poștei electronice sau pentru protocoale de difuziune (broadcast).

În cazul unui schimb de mai multe mesaje, de exemplu o sesiune

ssh, se poate combina numărul aleator cu un număr de ordine. La deschiderea conexiunii, receptorul trimite numărul aleator. Emițătorul include în fiecare pachet al conexiunii numărul aleator primit la deschiderea conexiunii și numărul de ordine al pachetului.

6.2.5. Combinarea criptării, autentificării și verificării prospețimii

Verificarea prospețimii unui mesaj se face pe baza unui număr unic inclus în mesaj, număr unic pe care receptorul îl verifică la primirea mesajului. Dacă numărul unic are o singură valoare validă, se poate conveni că numărul nu se transmite efectiv, însă dispersia sau semnătura se calculează ca și când numărul ar fi parte a mesajului.

Combinarea criptării cu autentificarea se poate face în trei moduri:

- Se calculează întâi semnătura sau dispersia, iar apoi se criptează rezultatul concatenării datelor utile cu dispersia sau semnătura. Deși nu există o slăbiciune cunoscută, unii criptografi susțin că prezența dispersiei în mesajul criptat ar putea oferi unui adversar o posibilitate de-a sparge criptarea [Rogaway 1995].
- Se criptează mai întâi mesajul, iar apoi se calculează dispersia sau semnătura mesajului criptat. Pentru această metodă, este necesar ca o anumită cheie pentru semnătură sau dispersie să nu se utilizeze decât cu o singură cheie de criptare. În caz contrar, este posibil ca un mesaj criptat cu o cheie să fie copiat de adversar și trimis destinatarului după schimbarea cheii de criptare. Mesajul trece testul de autenticitate, însă, în urma decriptării cu noua cheie, rezultă un alt text clar decât cel original (vulnerabilitate de tip fals existent).
- Se criptează doar textul clar, iar apoi la textul cifrat rezultat se adaugă semnătura sau dispersia textului clar. Dacă autentificarea se face prin dispersie cu cheie, metoda nu prezintă vulnerabilități (în caz contrar, se poate arăta că funcția de dispersie este vulnerabilă la fals existent). Acest mod de combinare a criptării cu dispersia cu cheie este utilizat de protocolul *ssh* versiunea 2. Dacă autentificarea se face prin semnătură digitală, metoda nu este corectă deoarece permite unui adversar care bănuiește textul clar să verifice dacă textul clar este cel bănuțit de el.

6.3. Stabilirea cheilor

În paragrafele precedente, am presupus că partenerii de comunicație dispun deja de cheile necesare criptării și autentificării mesajelor transmise. În

cele ce urmează, vom studia cum se poate face ca aceste chei să fie disponibile partenerilor. Cheile respective pot fi chei pentru criptografie simetrică sau pentru autentificare prin dispersie cu cheie, caz în care cheile le vom numi *chei simetrice*, sau pot fi chei publice pentru criptografie asimetrică sau pentru semnătură digitală. Transmiterea celor două tipuri de chei au cerințe distincte.

În cazul unei chei simetrice, acțiunea prin care cheia este generată și făcută disponibilă partenerilor de comunicație se numește *stabilirea cheii*. Un protocol de stabilire a cheii trebuie să îndeplinească următoarele cerințe:

- Cheia stabilită să nu poată fi cunoscută de altcineva decât de entitățile ce doresc să comunice. Acțiunea de satisfacere a acestei cerințe poartă denumirea de *autentificarea cheilor* și este obligatorie în orice proces de stabilire a cheilor.
- Cheia stabilită să fie proaspătă și, eventual, să nu poată fi impusă unilateral de vreuna dintre părți ci să fie calculată din elemente generate de fiecare dintre parteneri. Această cerință este utilă pentru a preîntâmpina situația în care un adversar, care reușește să obțină o cheie mai veche, ar putea determina entitățile care comunică să refolosească acea cheie.
- Fiecare entitate ce comunică să aibă confirmarea că partenerul de comunicație a primit efectiv cheia. Acțiunea care verifică satisfacerea acestei cerințe poartă denumirea de *confirmarea cheii*, iar confirmarea cheii împreună cu autentificarea cheii poartă denumirea de *autentificarea explicită a cheii*. Este posibil să nu se prevadă confirmarea cheii ca parte a protocolului de stabilire a cheii; în acest caz, începerea comunicației propriu-zise cu ajutorul cheii respective constituie confirmarea cheii.

Notăm că, în anumite cazuri, autentificarea cheii stabilite se face unilateral (adică doar una dintre entități știe cu certitudine ce entitate mai cunoaște cheia negociată). În astfel de cazuri, a doua entitate fie nu are nevoie să o autentifice pe prima (de exemplu, a doua entitate este un server public), fie utilizează un mecanism de autentificare a utilizatorilor (§ 6.5).

În cazul unei chei publice, acțiunea prin care cheia este făcută disponibilă partenerilor se numește *certificarea cheii*. Spre deosebire de cazul cheilor simetrice, unde transmisia unei chei între partenerii de comunicație se face doar pentru o cheie proaspăt generată, în cazul cheilor publice se întâmplă frecvent ca o aceeași cheie publică să fie transmisă de mai multe ori către o aceeași entitate. Certificarea unei chei are următoarele cerințe:

- Receptorul unei chei publice să poată verifica dacă cheia primită este într-adevăr cheia publică a partenerului de comunicație.
- Receptorul cheii să poată verifica dacă cheia mai este validă. O cheie

publică trebuie să poată fi invalidată dacă se suspectează că a fost compromisă cheia secretă corespunzătoare.

În prezența unui adversar, stabilirea cheilor simetrice și certificarea cheilor publice necesită mecanisme de securizare a comunicației (criptare și autentificare).

Stabilirea cheilor sau certificarea cheilor între două entități care nu au deja chei care să le permită o comunicație securizată între ele se poate face prin două metode:

- *cu ajutorul unui terț de încredere*: Această metodă necesită existența unei a treia entități care să poată deja comunica securizat cu fiecare din primele două și care să prezinte încredere acestora.
- *prin intermediul unui utilizator uman* (§ 6.3.5).

După rolul lor față de un mecanism de stabilire a cheilor, cheile se numesc:

- *chei efemere* (engl. *ephemeral key*) sau *chei de sesiune* (engl. *session key*), utilizate pentru comunicația propriu-zisă între două entități. O cheie efemeră se utilizează pe durată scurtă, de exemplu pe durata unei conexiuni sau pentru a cripta un singur mesaj. Ea este creată special pentru o anumită ocazie și este distrusă imediat după aceea. Deoarece, din motive de viteză, comunicația propriu-zisă este protejată prin criptografie simetrică și, uneori, prin dispersie cu cheie, cheile efemere sunt chei simetrice.
- *chei de lungă durată* (engl. *long-term key*), utilizate în cadrul mecanismelor de stabilire sau certificare a cheilor. Cheile de lungă durată pot fi chei simetrice sau chei asimetrice.

Mai dăm câteva considerente practice legate de stabilirea sau certificarea cheilor:

- Numărul de chei pe care trebuie să le posedă o entitate pentru a putea comunica trebuie să fie cât mai mic. Ideal, fiecare entitate dispune de o singură cheie de lungă durată, permițând o comunicație securizată cu un terț de încredere. Atunci când entitatea are nevoie să comunice cu o altă entitate, obține, prin intermediul terțului de încredere, cheia necesară comunicării cu partenerul dorit. După utilizare, cheia respectivă poate fi ștearsă.
- Intervenția manuală în stabilirea cheilor trebuie să fie minimă. Totuși, un prim transport manual al unei chei este întotdeauna necesar.

- Deoarece cheile de lungă durată pot fi aflate de adversari, cheile trebuie să poată fi schimbate periodic. De asemenea, dacă un adversar înregistrează comunicația legată de stabilirea unei chei de sesiune și, ulterior, obține o cheie de lungă durată utilizată în stabilirea acelei chei de sesiune, este bine să nu poată să obțină cheia de sesiune, pentru a nu putea mai departe decripta sesiunea.

6.3.1. Stabilirea cheilor în prezența unui adversar pasiv

Ne vom ocupa în continuare de stabilirea unei chei de sesiune între două entități, A și B , în prezența unui adversar pasiv și în lipsa vreunei chei deja disponibile. Cu alte cuvinte, problema este ca A și B să ajungă la un secret partajat printr-o comunicație integral la vedere.

În acest paragraf nu ne vom ocupa de situația în care un adversar ar putea participa activ în schimbul de mesaje. Aplicarea metodelor din acest paragraf în prezența unui adversar activ permite *atacul omului din mijloc* descris în § 6.3.1.3.

Există două metode utilizabile în acest scop:

- utilizarea criptografiei asimetrice,
- alte metode, dintre care cea mai cunoscută este metoda Diffie-Hellman.

6.3.1.1. Stabilirea cheilor prin criptografie asimetrică

Protocolul este următorul:

- Pregătirea: A generează o pereche de chei pentru criptografie asimetrică; cheia secretă este cheia sa de lungă durată.
- Stabilirea cheii de sesiune:
 - a) A trimite lui B cheia publică a lui A .
 - b) B generează aleator o cheie de sesiune k
 - c) B trimite lui A cheia de sesiune k criptată cu cheia publică primită de la A
 - d) A decriptează cheia transmisă de B

O variantă îmbunătățită este ca A să aibă, pe lângă cheia (mai bine zis, perechea de chei) de lungă durată, o a doua pereche de chei pentru criptografie asimetrică care să fie regenerată periodic. A transmite lui B ambele chei publice (cheia de lungă durată și cheia publică proaspătă), iar B criptează cheia de sesiune cu cheia proaspătă iar rezultatul îl criptează cu cheia de lungă

durată. Avantajul obținut este că dacă un adversar obține cheia secretă de lungă durată a lui A , dar între timp cheia proaspătă a expirat și a fost distrusă, adversarul nu mai poate decripta comunicațiile vechi.

Soluția în varianta simplă este utilizată de PGP/GPG. Aici emițătorul mesajului are rolul lui B și receptorul are rolul lui A . Emițătorul obține cheia publică a receptorului, iar la trimiterea unui mesaj generează aleator o cheie de sesiune, criptează mesajul folosind cheia de sesiune și criptează cheia de sesiune folosind cheia publică a destinatarului. Mesajul transmis conține cele două elemente criptate.

Soluția în varianta îmbunătățită este aplicată de protocolul *ssh* versiunea 1. Serverul are rolul lui A , generând perechile de chei și transmițând clientului cele două chei publice. Clientul generează cheia de sesiune și i-o trimite serverului criptată pe rând cu cele două chei.

6.3.1.2. Stabilirea cheii prin metoda Diffie-Hellman

Protocolul este următorul:

1. Se generează (pe o cale oarecare) un număr prim p mare și un număr g ;
2. A alege un număr aleator x și calculează și-i transmite lui B numărul

$$n_A = g^x \bmod p;$$

3. B alege un număr aleator y și calculează și-i transmite lui A numărul

$$n_B = g^y \bmod p;$$

4. A și B calculează cheia de sesiune k astfel: A calculează

$$k = (n_B)^x \bmod p,$$

iar B calculează

$$k = (n_A)^y \bmod p.$$

Cheia de sesiune calculată de cei doi este aceeași:

$$(g^x \bmod p)^y \bmod p = (g^y \bmod p)^x \bmod p = g^{xy} \bmod p,$$

iar calcularea lui $g^{xy} \bmod p$ cunoscând doar g , n , $g^x \bmod p$ și $g^y \bmod p$ este foarte dificilă.

Ca avantaj față de soluția din paragraful precedent, nu este necesară o cheie de lungă durată. De asemenea, aplicarea metodei Diffie-Hellman este

mai rapidă decât regenerarea la fiecare mesaj a unei perechi de chei pentru un cifru asimetric. Ca dezavantaj, este necesară o comunicație interactivă; protocolul Diffie-Hellman nu este aplicabil transmiterii mesajelor prin poștă electronică.

6.3.1.3. Atacul man-in-the-middle

Protocoalele descrise în paragrafele 6.3.1.1 și 6.3.1.2 sunt aplicabile doar în absența unui adversar activ. Un adversar activ I se poate interpune între A și B astfel:

- comunică cu A jucând rolul lui B pentru a stabili o cheie de sesiune k_1 ;
- comunică cu B jucând rolul lui A pentru a stabili o cheie de sesiune k_2 ;
- decriptează mesajele trimise de A lui B (acestea fiind criptate cu k_1), le citește, eventual le modifică, după care le criptează (și eventual le autentifică) utilizând cheia k_2 .

Rezultatul atacului este că A și B cred că comunică fiecare cu celălalt când de fapt ei comunică cu I .

Există metode, bazate pe transmiterea în fragmente a mesajelor, care împiedică forma pură a atacului man-in-the-middle; totuși, în general, pentru ca A să-l distingă pe B de un adversar este necesar ca B să aibă o caracteristică distinctivă inimitabilă; o astfel de caracteristică este cunoașterea unei chei secrete.

Ca urmare, în orice comunicație sigură trebuie să existe cel puțin un pas din inițializare în care să se transfere o cheie între B și A , sau între B și un terț de încredere și între terț și A .

6.3.2. Stabilirea cheilor în prezența unui adversar activ

Vom studia în continuare problema stabilirii unei chei de sesiune între două entități, A și B , în prezența unui adversar activ. În urma protocolului, cheia de sesiune trebuie să fie cunoscută doar de A și de B și să fie proaspătă (un adversar să nu poată impune alegerea unei chei stabilite la o execuție anterioară a protocolului). În acest scop, presupunem că A și B și-au stabilit chei de lungă durată, simetrice sau asimetrice, pentru criptare și autentificare.

Există multe protocoale de stabilire a cheilor în această situație. În general, aceste protocoale se construiesc astfel:

1. Fie una dintre părți generează aleator o cheie și o transmite criptat partenerului, fie se aplică un protocol de tipul Diffie-Hellman.
2. Fiecare parte trimite celelalte un *nonce* (de exemplu un număr aleator).

3. Fiecare parte trimite celeilalte o semnătură sau o dispersie calculată din informațiile de la punctele 1 și 2.

Primul punct are ca scop obținerea unei chei pe care să o cunoască doar părțile între care a avut loc schimbul de mesaje. Al doilea punct are rolul de-a asigura că negocierea și, în consecință, cheia rezultată este proaspătă. Al treilea punct are rolul de-a asigura fiecare parte că mesajele de la punctele 1 și 2 au fost schimbate cu partenerul dorit. Dacă verificarea semnăturii sau dispersiei de la punctul 3 eșuează, înseamnă că protocolul a fost influențat de un adversar activ și, în consecință, cheia negociată este compromisă. Este esențial deci ca, până în momentul în care verificarea autenticității mesajelor transmise a fost efectuată cu succes, cheia negociată să nu fie utilizată.

EXEMPLUL 6.5: Transmiterea unei chei prin criptare simetrică se poate face după cum urmează. Notăm cu A și B cele două entități care comunică, cu K_c o cheie pentru criptare simetrică, cunoscută doar de A și de B și cu K_h o cheie pentru dispersie, de asemenea cunoscută doar de A și de B .

1. A alege un număr aleator r_A și-l transmite lui B .
2. B alege cheia de sesiune k . Apoi calculează și transmite $x = c_{K_c}(k)$ și $s = h_{K_h}(r_A \cdot k)$.
3. A decriptează x obținând k și verifică dispersia s .

De remarcat că, dacă transmisia conținând cheia de sesiune k criptată cu cheia de lungă durată K_c este interceptată de un adversar, iar adversarul obține ulterior cheia K_c , atunci adversarul poate decripta cheia de sesiune și întreaga sesiune protejată cu această cheie.

EXEMPLUL 6.6: Stabilirea cheii de sesiune prin schimb Diffie-Hellman și autentificare prin semnătură digitală se face după cum urmează. Protocolul este, cu mici modificări, cel utilizat de *ssh* versiunea 2. În cele ce urmează, notăm cu A și B cele două entități, cu g și p baza și modulul din schimbul Diffie-Hellman, cu K_{sA} și K_{sB} cheile (secrete) de semnătură ale lui A și B și cu K_{vA} și K_{vB} cheile (publice) de verificare corespunzătoare.

1. A alege două numere aleatoare, r_A , utilizat pentru asigurarea prospețimii schimbului de chei, și x_A utilizat pentru derivarea cheii prin metoda Diffie-Hellman. Apoi A transmite lui B numerele r_A și $n_A = g^{x_A} \bmod p$ (unde g și p sunt parametrii pentru Diffie-Hellman).
2. B procedează similar, alegând r_B și x_B și transmițând r_B și $n_B = g^{x_B} \bmod p$.
3. A transmite lui B semnătura $s_A = h_{K_{sA}}(r_A \cdot n_A \cdot r_B \cdot n_B)$.

4. analog, B transmite lui A semnătura $s_B = h_{K_{s_B}}(r_B \cdot n_B \cdot r_A \cdot n_A)$.
5. A verifică semnătura s_B și, dacă se potrivește, calculează cheia de sesiune $k = n_B^{x_A} \bmod p$.
6. B verifică semnătura s_A și, dacă se potrivește, calculează cheia de sesiune $k = n_A^{x_B} \bmod p$.

6.3.3. Stabilirea cheilor cu ajutorul unui terț de încredere

Vom studia în continuare problema stabilirii unei chei între două entități A și B care nu partajează în prealabil nici un fel de cheie, în prezența unui adversar activ. Presupunem, în schimb, existența unei a treia entități (un *terț de încredere*, engl. *trusted third party*), T , satisfăcând condițiile:

- A și T partajează chei pentru criptare și autentificare (K_{cA} , respectiv K_{hA});
- B și T partajează chei pentru criptare și autentificare; (K_{cB} , respectiv K_{hB});
- A și B au încredere în serviciile oferite de T .

T se mai numește *server de distribuire a cheilor* (engl. *Key Distribution Center* — KDC).

Mai presupunem că protocolul de stabilire a cheii este inițiat de către A .

Ideea soluției este următoarea: serverul T generează aleator o cheie de sesiune, pe care o transmite lui A și lui B . Transmisia către A este criptată și autentificată cu cheia partajată între A și T , iar transmisia către B este criptată și autentificată cu cheia partajată între B și T .

Deoarece este de presupus că T face acest serviciu pentru mai multe entități, pachetele transmise către A și B trebuie să conțină și numele lor, astfel încât, la primirea pachetului, A și B să știe cine este partenerul cu care comunică.

Simplificat, protocolul ar fi următorul:

1. A transmite spre T o cerere în clar prin care cere o cheie de sesiune pentru B ;
2. T generează o cheie de sesiune k ;
3. T transmite spre A un pachet

$$(A, B, c_{K_{cA}}(k), h_{K_{cA}}(k \cdot A \cdot B))$$

4. T transmite spre B un pachet

$$(A, B, c_{K_{cB}}(k), h_{K_{cB}}(k \cdot A \cdot B))$$

5. A decriptează cheia de sesiune k și verifică dispersia, precum și numele A și B ;

6. B procedează la fel ca și A .

Protocolul de mai sus oferă doar autentificarea cheii; nu verifică și prospețimea acesteia. Vulnerabilitatea introdusă este dată de faptul că un adversar poate trimite mesajele de la pașii 3 și 4 în locul serverului T pentru a forța stabilirea unei chei vechi. Dacă adversarul reușește să obțină o cheie de sesiune, el poate forța astfel stabilirea aceleiași chei, compromise, în sesiunile următoare.

Pentru verificarea prospețimii, trebuie introduse în mesaje niște elemente de control al prospețimii.

O primă posibilitate, exploatată de protocolul Kerberos, este adăugarea unei perioade de valabilitate. Perioada de valabilitate este adăugată în mesajele transmise în pașii 1, 3 și 4. A și B resping, în cadrul pașilor 5 și 6, cheia de sesiune dacă timpul curent este în afara perioadei de valabilitate înscrise în pachete lângă cheie.

Protocolul Kerberos mai aduce câteva modificări față de protocolul descris mai sus, una dintre ele fiind aceea că mesajul de la pasul 4 este trimis de T lui A împreună cu mesajul de la pasul 3, urmând ca A să-l transmită către B la deschiderea conexiunii către acesta.

O a doua posibilitate, exploatată de protocolul Otway-Rees, se bazează pe numere aleatoare. Acesta prevede că A și B transmit către T câte un număr aleator, iar T include numărul aleator transmis de A în mesajul de la pasul 3 și numărul aleator transmis de B în mesajul de la pasul 4. A și B verifică, în cadrul pașilor 5, respectiv 6, că numerele aleatoare incluse în mesajul autentificat de la T sunt într-adevăr numerele trimise de ele.

Utilizarea practică a stabilirii cheilor cu ajutorul unui terț de încredere se face construind un server T care crează chei de sesiune, la cerere, pentru toate entitățile din rețea. Astfel, T partajează o cheie simetrică (de fapt, două: una pentru criptare, cealaltă pentru autentificare) cu fiecare dintre entitățile din rețea.

Într-o rețea mare, nu mai este posibil să se lucreze cu un singur server T , deoarece aceasta ar cere tuturor să aibă încredere în administratorul serverului T . Pentru stabilirea de chei între orice două entități în aceste

condiții, se construiește un sistem astfel:

- Se configurează mai multe servere de distribuire a cheilor, fiecare entitate având o cheie partajată cu unul dintre aceste servere.
- Se configurează chei partajate între câte două servere de distribuire a cheilor. Aceste chei partajate formează legături securizate între servere. Graful format de serverele de distribuire a cheilor și de legăturile securizate trebuie să fie conex.

Atunci când o entitate A dorește să comunice cu o entitate B , se caută un lanț de servere, T_1, \dots, T_n , astfel încât A să aibă cheie partajată cu T_1 , T_1 să aibă cheie partajată cu T_2 , ș. a. m. d. Apoi, se stabilește, cu ajutorul lui T_1 , o cheie între A și T_2 ; cu ajutorul lui T_2 se stabilește o cheie între A și T_3 ș. a. m. d. până la obținerea unei chei între A și B .

În acest sistem, în loc să aibă toată lumea încredere într-un singur server, fiecare pereche de entități care doresc să comunice trebuie să aibă încredere în lanțul de servere ce participă la stabilirea cheii.

6.3.4. Certificarea cheilor publice

Presupunând că fiecare entitate are o pereche cheie secretă – cheie publică, o entitate A care dorește să comunice cu o entitate B își pune problema să dobândească cheia publică a lui B . Cheia publică nu este un secret, însă trebuie ca autenticitatea ei să fie verificabilă.

Soluția imediată este transportul cheii lui B de către un utilizator uman (vezi § 6.3.5 pentru detalii). Acest lucru nu este însă fezabil decât pentru un număr mic de chei. O soluție aplicabilă în rețele mari constă în utilizarea certificatelor.

Un *certificat* este un ansamblu cuprinzând o cheie publică, un nume de entitate și o semnătură a unui terț pe ansamblul celor două. Terțul semnatar se numește *autoritate de certificare*. Prin semnătură, autoritatea de certificare atestă că cheia publică din certificat aparține entității al cărei nume figurează în certificat.

Utilizarea certificatelor se face astfel. Dacă A nu are cheia publică a lui B , dar:

- are cheia publică a lui C dintr-o sursă sigură,
- are un certificat pentru B semnat de C , dintr-o sursă nesigură,
- are încredere în C că a verificat corect identitatea lui B înainte de a-i semna certificatul,

atunci A poate verifica semnătura lui C pe certificatul lui B și, dacă este corectă, poate prelua cheia lui B de pe certificat.

Schema de mai sus poate cuprinde mai multe autorități de certificare, din care o primă autoritate a cărei cheie se obține prin transport de către om și un lanț de autorități din care fiecare semnează certificatul următoarea.

Pentru schimbarea cheilor, în cazul compromiterii unei chei secrete, se prevăd două mecanisme:

expirarea cheilor. Un certificat are durată de valabilitate limitată; data creerii și data expirării sunt de asemenea înscrise în certificat și semnate de autoritatea de certificare. O entitate a cărui certificat se apropie de expirare va crea o nouă pereche de chei și va solicita autorității de certificare eliberarea unui nou certificat pentru noua cheie publică. După expirare, un certificat nu mai este recunoscut de nimeni ca valid și nu mai este folosit.

revocarea certificatelor. Se țin, pe servere accesibile public, așa-zise *certificate de revocare* ale certificatelor ale căror chei secrete se consideră compromise. Un certificat de revocare al unui certificat este un ansamblu cuprinzând datele de identificare ale certificatului revocat și semnătura autorității de certificare a certificatului revocat asupra acelor date de identificare.

Publicarea unui certificat de revocare pentru un certificat anunță invalidării certificatului original. O entitate care utilizează un certificat va verifica înainte, de fiecare utilizare, dacă nu există un certificat de revocare al certificatului respectiv.

Certificatul de revocare poate fi produs doar de către autoritatea de certificare emitentă sau de posesorul certificatului.

6.3.5. Transportul prin utilizatori umani

Orice protocol sigur de stabilire a cheilor are nevoie cel puțin o dată de un canal sigur bazat pe alte mijloace decât cele criptografice. Acest canal este necesar pentru transportul unei prime chei criptografice, utilizabilă pentru transportul sigur al altor chei. Acest canal sigur implică întotdeauna intervenția omului, și se prezintă sub una din următoarele forme:

1. Omul transportă, pe un suport amovibil (dischetă, CD, DVD, memorie flash) sau printr-o legătură ad-hoc securizată (cablu direct), o cheie de pe sistemul sursă pe sistemul destinație. Eventual această cheie poate fi cheia publică a unei autorități recunoscute, cheie înglobată într-un produs soft (de regulă browserele web au înglobate astfel de chei);
2. Omul citește o informație de pe sistemul sursă și o introduce sau o verifică pe sistemul destinație. Cantitatea de informație astfel trans-

portabilă este mică, de ordinul câtorva sute de biți cel mult. Metoda este greu de aplicat pentru informații secrete, deoarece informația este afișată pe ecranul sistemului sursă și ca urmare este vizibilă persoanelor din apropiere. Informațiile astfel transportate sunt de obicei dispersiile criptografice ale unor chei publice.

3. Omul inventează o parolă și o introduce pe ambele sisteme. Parola este utilizată pe post de cheie secretă. Omul este utilizat atât cu rolul de canal sigur de transport, cât și ca generator de „numere“ aleatoare.

Ne vom ocupa în continuare de aspecte privind implementarea metodelor 2 și 3.

Metoda 2 necesită o scriere a unui șir de biți într-o formă ușor de citit de către un om. Trebuie ținut cont de faptul că omul nu poate fi atent simultan la un ansamblu prea mare de obiecte diferite (simboluri, grupuri de simboluri, cuvinte). Între citirile unor astfel de grupuri, omul trebuie să-și poată lua puncte de reper pentru a ști unde a rămas. Ca urmare, un grup de cifre sau simboluri fără legătură între ele (adică care nu constituie un cuvânt din vocabularul utilizatorului) nu trebuie să fie mai mare de 6–8 simboluri. O dispersie *md5* scrisă direct în hexa cuprinde 32 simboluri (cifre hexa); utilizatorul va avea nevoie să pună degetul pe ecran pentru a o citi. Dacă aceeași dispersie *md5* este scrisă ca 8 grupuri de câte 4 cifre, cu spațiu sau alt separator între grupuri, devine mult mai ușor de citit. Pentru șiruri mai lungi, devine necesar un al doilea nivel de grupare.

O altă metodă, mai dificil de pus în practică, este transformarea șirului de biți într-un șir de cuvinte dintr-un dicționar standard sau într-un șir de silabe ce alcătuiesc cuvinte, probabil fără sens, dar pronunțabile. Pentru un dicționar de 4096 cuvinte, un cuvânt codifică 12 biți. O dispersie *md5* poate fi scrisă ca un șir de 11 cuvinte. Un astfel de șir de cuvinte poate fi memorat mai ușor decât secvența de 32 cifre hexa echivalentă.

Pentru a aplica metoda 3, remarcăm pentru început că o parolă inventată și memorată de om nu poate fi utilizată direct pe post de cheie. Ideal, dacă caracterele utilizate pentru parolă sunt cele 94 caractere ASCII imprimabile, parola ar avea o entropie de 6,44 biți pe caracter. Se estimează că un text în limbaj natural nu are mai mult de 1–2 biți pe caracter. O parolă ce poate fi memorată rezonabil va avea o entropie cuprinsă undeva între aceste două limite. Pe de altă parte, securitatea unui sistem criptografic se bazează pe faptul că entropia cheii este de 1 bit pe cifră binară (vezi § 6.1.3). Rezultă de aici două lucruri:

- Parola trebuie trecută printr-un „concentrator de entropie“ înainte de-a fi

utilizată ca și cheie. Acest „concentrator de entropie“ poate fi o funcție de dispersie (nu neapărat criptografică).

- Parola trebuie să fie suficient de lungă (și de aleator aleasă) ca să furnizeze efectiv biții de entropie necesari. O frază în limbaj natural, utilizată pentru derivarea unei chei de 128 biți, trebuie să aibă cam 85 litere (în jur de 20 de cuvinte).

De obicei este nerezonabil să cerem unui om să utilizeze o parolă cu entropie suficient de mare. În acest caz, micșorarea lungimii efective a cheii derivate din parolă trebuie compensată prin îngreunarea încercării cheilor de către adversar. Mai exact, protocolul ce utilizează cheia derivată din parolă trebuie modificat în așa fel încât să nu permită unui adversar să facă încercarea exhaustivă a cheilor pe mașina lui (unde poate dispune de putere mare de calcul și nu lasă urme) ci să trebuiască să contacteze una din mașinile care cunosc cheia (mașini care înregistrează tentativa și pot refuza un număr prea mare de tentative în timp scurt).

Ca terminologie, distingem *tentative de spargere off-line*, în care adversarul poate verifica dacă o parolă este corectă utilizând doar echipamentele proprii, și *tentative de spargere on-line*, în care adversarul contactează serverul atacat, încercând să deschidă o conexiune cu parola supusă verificării. Dacă o cheie nu poate fi spartă off-line, o lungime efectivă (entropie) de 30–40 biți este suficientă. O parolă bună, în acest context, trebuie să aibă doar 5–6 cuvinte, sau, dacă conține cifre și punctuație, 10–12 caractere.

6.4. Numere aleatoare

Un sistem criptografic utilizează numere aleatoare, în diverse scopuri:

- generarea cheilor,
- generarea vectorilor de inițializare sau a *salt*-urilor,
- generarea numerelor unice (*nonce*).

Numerele aleatoare generate trebuie să fie uniform distribuite și independente unul de altul. În plus față de alte aplicații, numerele aleatoare pentru scopuri criptografice trebuie să nu poată fi prevăzute sau influențate de un eventual adversar. (Condiții similare trebuie îndeplinite de numerele aleatoare utilizate pentru jocuri de noroc cu miză serioasă.)

Există două metode utilizabile pentru generarea numerelor aleatoare:

- *generatoare fizice*, care funcționează pe baza unor fenomene fizice suficient de aleatoare;

- *generatoare de numere pseudoaleatoare*, care produc numerele prin calcule (prin urmare, numerele generate sunt deterministe), dar algoritmul de generare „amestecă“ suficient de bine numerele pentru ca șirul de numere rezultat să pară aleator.

6.4.1. Generatoare fizice

Generatoarele fizice se împart mai departe în generatoare fizice dedicate, pe de o parte, și dispozitive având alte scopuri, dar utilizabile pentru producerea de numere aleatoare, pe de altă parte.

Dispozitivele dedicate se pot baza pe zgomotul termic al unui rezistor, dezintegrarea unei substanțe radioactive sau curenții de aer din interiorul carcasei unui harddisc. Dispozitivele dedicate sunt cele mai sigure și relativ rapide. Pe de altă parte, utilizatorul este nevoit să aibă încredere că producătorul dispozitivului l-a construit corect și nu a instalat o „ușă dosnică“, permițându-i acestuia să prevadă numerele generate. În plus, fiind produse în serie mai mică, dispozitivele sunt relativ scumpe.

Orice dispozitiv periferic prezintă un anumit grad de nedeterminism în comportamentul său. De exemplu, să măsurăm timpul scurs între momentele în care sunt apășate două taste consecutive și să exprimăm printr-un număr durata respectivă. Vom constata că cifrele cele mai puțin semnificative ale numărului respectiv sunt destul de aleatoare. Prin anumite prelucrări, se poate extrage de aici un șir de numere aleatoare de calitate bună. Alte dispozitive utilizabile în acest scop sunt: mausul, o videocameră (eventual îndreptată spre o flacără sau spre o „lampă cu lavă“), placa de rețea (însă aici trebuie multă grijă, deoarece pachetele primite de placa de rețea pot fi supravegheate de adversar), un ceas (de fapt, două ceasuri independente, exploatănd fluctuația derivei relative a celor două ceasuri).

Dispozitivele nededicate au ca avantaj prețul mai redus și riscul mai mic de-a fi „măsluite“ de către fabricant. Debitul de biți aleatori produși variază însă în funcție de utilizarea sistemului; în particular, pentru un server care nu admite utilizatori locali este dificil de obținut un debit satisfăcător de biți aleatori.

6.4.2. Generatoare de numere pseudoaleatoare

Un generator de numere pseudoaleatoare funcționează astfel:

- În fiecare moment t , generatorul are asociată o *stare internă* s_t . Starea internă este o valoare dintr-o mulțime arbitrară suficient de mare.
- Atunci când este nevoie de un număr aleator, acest număr este obținut aplicând o funcție asupra stării interne. Numărul produs este deci $x_t =$

$f(s_t)$, unde f este o funcție fixată.

- După producerea unui număr aleator, starea internă este modificată, pentru ca următorul număr să nu mai aibă aceeași valoare. Actualizarea stării interne se face pe baza unei a doua funcții, g . Avem deci $s_{t+1} = g(s_t)$.

Dacă funcțiile f și g „amestecă” suficient de bine biții, șirul pseudoaleator produs, $(x_t)_{t \in \mathbb{N}}$ are proprietăți statistice suficient de apropiate de numerele cu adevărat aleatoare.

De remarcat că întregul șir depinde de valoarea stării inițiale s_0 a generatorului; pentru aplicații non-criptografice și pentru teste, acest lucru este bun deoarece face comportamentul programelor reproductibil.

De asemenea, trebuie remarcat că, deoarece, în practică, mulțimea stărilor interne posibile este finită, mai devreme sau mai târziu starea internă se repetă și, ca urmare, întregul șir pseudoaleator este periodic. Perioada șirului pseudoaleator este cel mult egală cu numărul de stări interne posibile. Pentru o funcție g cu comportament apropiat de o funcție aleatoare sau de o permutare aleatoare, perioada șirului este de ordinul rădăcinii pătrate din numărul de stări interne posibile.

Pentru scopuri criptografice, sunt necesare câteva proprietăți suplimentare:

- Un adversar care cunoaște funcțiile f și g utilizate și cunoaște o parte dintre elementele șirului pseudoaleator (x_t) să nu poată calcula alte elemente ale șirului pseudoaleator. O condiție necesară pentru aceasta este ca funcția f să fie rezistentă la preimagine. O altă condiție necesară este ca mulțimea stărilor interne posibile să fie suficient de mare pentru ca explorarea ei prin forță brută să nu fie posibilă practic.
- Starea inițială s_0 trebuie să fie creată pornind de la un generator fizic. În caz contrar, starea inițială este previzibilă pentru un adversar și, ca urmare, întregul șir este previzibil.
- Este bine ca, dacă un adversar reușește să obțină starea internă s_t , să nu poată calcula numerele pseudoaleatoare deja generate (adică $x_0 \dots x_{t-1}$). Acest lucru este util pentru ca, dacă un adversar reușește să spargă un calculator, să nu poată decodifica comunicațiile anterioare. Pentru a îndeplini această cerință, este necesar ca și g să fie rezistentă la preimagine.

Generatoarele de numere pseudoaleatoare utilizate în practică nu se bazează pe generatoare fizice doar pentru starea inițială s_0 , ci modifică starea

internă și în timpul funcționării generatorului, pe măsură ce obțin biți aleatori de la generatorul fizic. Acest lucru are scopul ca, dacă un adversar reușește să obțină starea internă la un moment dat sau să afle starea internă inițială, să nu poată prevedea decât o mică parte dintre numerele aleatoare produse ulterior. De asemenea, starea inițială nu este generată la pornirea calculatorului, întrucât, de obicei, în acel moment nu sunt disponibili prea mulți biți aleatori de la generatoarele fizice. Starea internă este salvată la oprirea calculatorului, într-un fișier ce nu poate fi citit de utilizatorii obișnuiți, și reîncărcată la pornirea calculatorului.

6.4.3. Generatoare utilizate în practică

Sistemele de operare moderne oferă utilizatorului generatoare criptografice de numere aleatoare gata implementate. Acestea sunt disponibile astfel:

- Pe unele sisteme de tip unix, fișierul special `/dev/random` oferă numere aleatoare de la un generator fizic bazat pe acțiunile utilizatorului. De asemenea, fișierul special `/dev/urandom` oferă numere pseudoaleatoare utilizabile pentru aplicații criptografice.
- Sistemul Windows oferă funcția `CryptGenRandom()`, declarată în fișierul antet `wincrypt.h`.

De remarcat că utilizarea unor funcții non-criptografice pentru generarea numerelor aleatoare, cum ar fi `rand()` sau `random()` din biblioteca C standard este extrem de riscantă.

6.5. Autentificarea utilizatorilor

Prezentăm în continuare câteva utilizări ale metodelor criptografice în autentificarea utilizatorilor.

6.5.1. Stocarea parolelor

Un sistem de operare are nevoie să verifice parolele utilizatorilor ce doresc să se conecteze. Soluția trivială este ținerea unei baze de date cu corespondența nume, parolă. Această soluție (stocarea parolelor în clar) are un dezavantaj: un adversar care reușește să spargă sistemul sau un administrator indiscret poate obține direct parolele tuturor utilizatorilor. „Recolta“ este valoroasă dacă utilizatorii folosesc aceleași parole și pe alte sisteme.

O îmbunătățire a securității se face prin „criptarea“ parolelor. De fapt, nu este vorba de criptare, ci de transformarea parolei printr-o dispersie

criptografică h rezistentă la preimagine. În baza de date este stocată în locul parolei p transformata parolei $s = h(p)$.

La conectarea unui utilizator, sistemul cere parola utilizatorului și verifică, pentru parola introdusă p' , dacă $h(p') = s$. Deoarece h este rezistentă la preimagine, probabilitatea ca un adversar, care nu cunoaște parola p , să găsească o parolă p' satisfăcând $h(p') = s$ este neglijabil de mică.

Soluția are în continuare o slăbiciune, legată de faptul că un adversar care obține s poate obține p printr-un dicționar creat off-line: adversarul ia o mulțime de parole și, pentru fiecare parolă, calculează și memorează dispersia, obținând astfel un dicționar care asociază dispersiei parola corespunzătoare. Înarmat cu un astfel de dicționar, un adversar care obține s poate regăsi foarte rapid p dacă p era în dicționarul încercat.

Metoda de mai sus poate fi îmbunătățită, împiedicând crearea unui dicționar de dispersii și obligând adversarul să facă toată munca de spargere a parolelor după obținerea lui s . Conform noii metode, la inițializarea parolei, sistemul generează un număr aleator r și scrie în fișierul de parole perechea (r, s) unde $s = h(p \cdot r)$. Verificarea parolei p' dată de utilizator se face testând dacă $s = h(p' \cdot r)$. Un adversar care ar dori să facă un dicționar ar avea nevoie de un număr de dispersii egal cu produsul dintre numărul de parole de încercat și numărul de valori posibile pentru r .

Notăm că, indiferent de mecanismul folosit la stocarea parolelor, un adversar ce a spart un sistem, sau un administrator indiscret, va putea întotdeauna să obțină parola cu care se conectează un utilizator la acel sistem. De exemplu, adversarul poate înlocui programul obișnuit de login cu un program care scrie undeva parola în clar. Schemele de mai sus protejează doar parolele neutilizate după momentul spargerii sistemului.

Mai notăm că, în vederea transmiterii parolei prin rețea, transformările descrise mai sus nu pot înlocui criptarea „adevărată“. Dacă, în vederea autentificării utilizatorului, serverul cere $h(p)$ (în loc de p), atunci $h(p)$ devine efectiv parola de conectare prin rețea și orice adversar care cunoaște $h(p)$ poate impersona utilizatorul, fără a avea nevoie de p .

6.5.2. Parole de unică folosință

O *parolă de unică folosință* (engl. *One Time Password*) este o parolă care este acceptată de sistem ca fiind parolă validă cel mult o dată.

Una din aplicațiile parolelor de unică folosință este conectarea la un sistem, în prezența unui adversar pasiv, fără a recurge la criptare. Notăm că aplicativitatea este foarte limitată:

- comunicația nefiind criptată, metoda este inaplicabilă dacă datele trans-

mise trebuie să rămână secrete;

- un adversar activ poate „deturna“ conexiunea după deschidere și poate da orice comenzi în numele utilizatorului conectat.

Unul dintre sistemele de parole de unică folosință este descris în continuare. În cele ce urmează, h este o dispersie rezistentă la preimagine, iar $h^n(x) = h(h(\dots h(x)\dots))$.

1. Utilizatorul alege o *parolă primară*, de lungă durată, p .
2. La inițializarea parolei pe sistem, sistemul generează și afișează un număr aleator, nesecret, r . De asemenea, sistemul afișează un număr de iterații, n , preconfigurat (uzual $n = 10000$).
3. Utilizatorul calculează, cu ajutorul unui dispozitiv de calcul de încredere, $p_n = h^n(p \cdot r)$. Apoi transmite p_n sistemului pe care dorește să-și configureze autentificarea.
4. Sistemul memorează în baza de date ansamblul (p_n, n, r) .
5. La prima conectare, sistemul afișează r și $n - 1$ și cere utilizatorului să calculeze și să introducă parola de unică folosință $p_{n-1} = h^{n-1}(p \cdot r)$. Sistemul verifică parola de unică folosință testând dacă $h(p_{n-1}) = p_n$. Apoi sistemul înlocuiește în baza de date (p_n, n, r) cu $(p_{n-1}, n - 1, r)$.

Un avantaj al sistemului este faptul că parola primară p este cunoscută doar de către dispozitivul utilizat pentru calculul parolelor de unică folosință. În particular, calculatorul care autentifică utilizatorul nu obține niciodată și nici nu poate deduce parola primară. Administratorul unui astfel de calculator nu poate impersona utilizatorul pe un alt calculator pe care utilizatorul utilizează aceeași parolă primară.

Dezavantajul sistemului, față de parola clasică, este că utilizatorul are nevoie de un dispozitiv de calcul în vederea calculării parolelor de unică folosință. Proceduri de lucru pentru utilizator pot fi:

- Utilizatorul rulează local un program pentru calculul parolelor de unică folosință. Această metodă este aplicabilă doar dacă utilizatorul are deplină încredere în calculatorul local.
- Utilizatorul calculează, cu ajutorul unui calculator de încredere, următoarele 4-5 parole de unică folosință și le notează pe hârtie. Ca de obicei, scrierea parolelor pe hârtie implică un risc, însă aflarea parolelor de către un adversar nu permite decât deschiderea unui număr mic de sesiuni și mai ales nu permite aflarea, de către adversar, a parolei primare.
- Utilizatorul folosește un dispozitiv de calcul dedicat. Aceasta este metoda cea mai sigură, dar și cea mai scumpă.

Rețele de calculatoare

Procoale

Radu-Lucian Lupșa

Aceasta este ediția electronică a cărții *Rețele de calculatoare*, publicată la Casa Cărții de Știință, în 2008, ISBN: 978-973-133-377-9.

Drepturile de autor aparțin subsemnatului, Radu-Lucian Lupşa.

Subsemnatul, Radu-Lucian Lupşa, acord oricui dorește dreptul de a copia conținutul acestei cărți, integral sau parțial, cu condiția atribuirii corecte autorului și a păstrării acestei notițe.

Cartea poate fi descărcată gratuit de la adresa
<http://www.cs.ubbcluj.ro/~rlupsa/works/retele.pdf>

Cuprins

Principii

Cuprins	5
Prefață	13
1 Introducere	15
1.1 Serviciile oferite de rețea	15
1.2 Principalele elemente ale unei rețele de calculatoare	20
1.3 Premise generale în elaborarea și implementarea protocoalelor în rețele	22
2 Noțiuni de teoria informației	25
2.1 Problema codificării informației pentru un canal discret	26
2.2 Coduri cu proprietatea de prefix	29
2.2.1 Reprezentarea arborescentă a codurilor prefix	29
2.2.2 Decodificarea în cazul codurilor prefix	31
2.2.3 Lungimile cuvintelor unui cod prefix	33
2.3 Coduri optime	39
2.3.1 Cantitatea de informație	40
2.3.2 Lungimea medie a cuvintelor de cod	41
2.3.3 Generarea codului optim prin algoritmul lui Huffman	44
2.3.4 Compresia fișierelor	50
2.4 Coduri detectoare și corectoare de erori	51
2.4.1 Modelul erorilor	52
2.4.2 Principiile codurilor detectoare și corectoare de erori	53
2.4.3 Câteva coduri detectoare sau corectoare de erori	55
2.4.3.1 Bitul de paritate	55
2.4.3.2 Paritate pe linii și coloane	55
2.4.3.3 Coduri polinomiale	56
2.4.4 Coduri detectoare și corectoare de erori în alte domenii	57

3 Nivelul fizic	59
3.1 Problema transmisiei informației la nivelul fizic	59
3.2 Transmiterea semnalelor	60
3.2.1 Modificările suferite de semnale	60
3.2.2 Analiza transmiterii semnalelor cu ajutorul transformatei Fourier	62
3.3 Codificarea informației prin semnale continue	65
3.3.1 Scheme de codificare	65
3.3.2 Modulația	68
3.3.3 Multiplexarea în frecvență	71
3.3.4 Capacitatea maximă a unui canal de comunicație	71
3.4 Transmisia prin perechi de conductoare	72
3.4.1 Construcția cablului	72
3.4.2 Proprietăți ale mediului	74
3.4.3 Legătură magistrală	75
3.4.4 Considerente practice	76
3.5 Transmisia prin unde radio	77
3.5.1 Propagarea undelor	78
3.5.1.1 Polarizarea	78
3.5.1.2 Absorbția și reflexia	79
3.5.1.3 Difrakția	79
3.5.1.4 Interferența undelor	80
3.5.1.5 Divergența undelor	80
3.5.2 Antene	80
3.5.2.1 Directivitatea	81
3.5.2.2 Polarizarea	83
3.5.2.3 Tipuri de antene	83
3.5.3 Raza de acțiune a unei legături radio	83
3.5.3.1 Obstacolele	83
3.5.3.2 Linia orizontului	84
3.5.3.3 Utilizarea sateliților artificiali ai Pământului	84
3.5.3.4 Zgomotul	85
3.5.3.5 Scăderea puterii cu distanța	86
3.5.3.6 Emisia direcționată și polarizată	86
3.5.4 Spectrul radio și alocarea lui	86
3.5.5 Particularități ale sistemelor de comunicație prin radio	88
3.5.5.1 Topologia legăturii	88
3.5.5.2 Fiabilitatea	89
3.5.5.3 Securitatea	89
3.6 Transmisia optică	89
3.6.1 Construcția mediului	90
3.6.1.1 Conectarea fibrelor optice	91
3.6.2 Propagarea semnalului optic	91
3.6.2.1 Moduri de propagare	91

3.6.2.2	Caracteristici ale mediului	92
3.6.2.3	Multiplexarea în lungimea de undă	92
3.6.3	Considerente practice	93
4	Nivelul legăturii de date	95
4.1	Detectarea și corectarea erorilor	96
4.2	Controlul accesului la mediu	97
4.2.1	Protocoale bazate pe asigurarea unui interval exclusiv de emisie	98
4.2.2	Protocoale bazate pe coliziuni și retransmitere	99
4.2.3	Protocoale mixte	101
4.3	Retransmiterea pachetelor pierdute	102
4.3.1	Principiul confirmărilor pozitive și retransmiterilor	103
4.3.2	Trimiterea în avans a mai multor pachete	108
4.3.3	Spațiul numerelor de confirmare	109
4.4	Controlul fluxului	114
4.4.1	Cereri de suspendare și de continuare	115
4.4.2	Mecanismul pas cu pas	115
4.4.3	Mecanism combinat cu retransmiterea pachetelor pierdute	116
4.5	Multiplexarea în timp	117
5	Nivelul rețea și nivelul transport	119
5.1	Retransmiterea datelor de către nodurile intermediare	120
5.1.1	Retransmiterea în rețele bazate pe datagrame	122
5.1.2	Retransmiterea în rețele bazate pe conexiuni	122
5.2	Algoritmi de dirijare	125
5.2.1	Calculul drumurilor cu informații complete despre graful rețelei	127
5.2.2	Calculul drumurilor optime prin schimb de informații de distanță	128
5.2.3	Dirijarea ierarhică	136
5.2.4	Metode particulare de dirijare	139
5.2.4.1	Inundarea	139
5.2.4.2	Învățarea rutelor din adresele sursă ale pachetelor	140
5.2.5	Metode de difuziune	140
5.3	Funcționarea la trafic ridicat	141
5.3.1	Alegerea pachetelor de transmis	142
5.3.2	Controlul congestiei	143
5.3.3	Formarea (limitarea) traficului	144
5.3.4	Rezervarea resurselor	145
5.4	Nivelul transport	146
5.5	Interconectarea rețelelor	147
6	Metode și protocoale criptografice	149
6.1	Asigurarea confidențialității	151
6.1.1	Introducere	151
6.1.2	Refolosirea cheilor	154
6.1.3	Problema spargerii unui cifru	155

6.1.4	Algoritmi de criptare utilizați în practică	157
6.1.5	Criptografie asimetrică (cu cheie publică)	163
6.1.5.1	Utilizarea criptografiei asimetrice	164
6.2	Autentificarea mesajelor	165
6.2.1	Funcții de dispersie criptografice	166
6.2.1.1	Utilizarea funcțiilor de dispersie	167
6.2.2	Funcții de dispersie cu cheie	168
6.2.3	Semnătura digitală	169
6.2.4	Verificarea prospețimii mesajelor	171
6.2.5	Combinarea criptării, autentificării și verificării prospețimii . . .	173
6.3	Stabilirea cheilor	173
6.3.1	Stabilirea cheilor în prezența unui adversar pasiv	176
6.3.1.1	Stabilirea cheilor prin criptografie asimetrică	176
6.3.1.2	Stabilirea cheii prin metoda Diffie-Hellman	177
6.3.1.3	Atacul man-in-the-middle	178
6.3.2	Stabilirea cheilor în prezența unui adversar activ	178
6.3.3	Stabilirea cheilor cu ajutorul unui terț de încredere	180
6.3.4	Certificarea cheilor publice	182
6.3.5	Transportul prin utilizatori umani	183
6.4	Numere aleatoare	185
6.4.1	Generatoare fizice	186
6.4.2	Generatoare de numere pseudoaleatoare	186
6.4.3	Generatoare utilizate în practică	188
6.5	Autentificarea utilizatorilor	188
6.5.1	Stocarea parolelor	188
6.5.2	Parole de unică folosință	189

Protocoale

Cuprins	195
----------------	------------

7 Codificări de interes practic	203
7.1 Probleme privind reprezentarea numerelor întregi	203
7.1.1 Reprezentări pe biți	203
7.1.1.1 Bitul	204
7.1.1.2 Șiruri de biți	204
7.1.1.3 Reprezentarea pe biți a numerelor întregi	205
7.1.2 Reprezentări pe octeți	206
7.1.2.1 Octeți	206
7.1.2.2 Șiruri de octeți	208
7.1.2.3 Reprezentarea numerelor pe un număr întreg de octeți . . .	208
7.1.2.4 Reprezentarea numerelor pe un șir arbitrar de biți	210
7.1.3 Probleme privind reprezentarea lungimii șirurilor	212
7.1.4 Alte metode de reprezentare a numerelor întregi	214

7.2	Codificarea textelor	215
7.2.1	Codificarea ASCII	216
7.2.2	Codificările ISO-8859	217
7.2.3	Codificările Unicode	218
7.2.3.1	Codificarea UTF-8	220
7.2.3.2	Codificările UTF-16	220
7.2.3.3	Codificările UTF-32	221
7.3	Reprezentarea datei și orei	221
7.3.1	Măsurarea timpului	222
7.3.2	Obiectivele în alegerea reprezentării timpului în calculator	224
7.3.3	Formate utilizate în practică	225
7.3.3.1	Formatul utilizat de poșta electronică	225
7.3.3.2	ISO-8601 și RFC-3339	226
7.3.3.3	Timpul POSIX	227
7.3.3.4	TAI 64	227
7.4	Recodificări	228
7.4.1	Codificarea hexazecimală	228
7.4.2	Codificarea în baza 64	229
7.4.3	Codificări bazate pe secvențe de evitare	229
8	Programarea în rețea — introducere	231
8.1	Interfața de programare <i>socket BSD</i>	231
8.1.1	Comunicația prin conexiuni	232
8.1.1.1	Deschiderea conexiunii de către client	233
8.1.1.2	Deschiderea conexiunii de către server	233
8.1.1.3	Comunicația propriu-zisă	234
8.1.1.4	Închiderea conexiunii	234
8.1.2	Comunicația prin datagrame	235
8.1.3	Principalele apeluri sistem	237
8.1.3.1	Funcția <code>socket()</code>	237
8.1.3.2	Funcția <code>connect()</code>	237
8.1.3.3	Funcția <code>bind()</code>	238
8.1.3.4	Funcția <code>listen()</code>	239
8.1.3.5	Funcția <code>accept()</code>	239
8.1.3.6	Formatul adreselor	240
8.1.3.7	Interacțiunea dintre <code>connect()</code> , <code>listen()</code> și <code>accept()</code>	242
8.1.3.8	Funcțiile <code>getsockname()</code> și <code>getpeername()</code>	242
8.1.3.9	Funcțiile <code>send()</code> și <code>recv()</code>	243
8.1.3.10	Funcțiile <code>shutdown()</code> și <code>close()</code>	245
8.1.3.11	Funcțiile <code>sendto()</code> și <code>recvfrom()</code>	245
8.1.4	Exemple	246
8.1.4.1	Comunicare prin conexiune	246
8.1.4.2	Comunicare prin datagrame	249
8.2	Formatarea datelor	252

8.2.1	Formate binare	252
8.2.1.1	Tipuri întregi	252
8.2.1.2	Şiruri de caractere şi tablouri	254
8.2.1.3	Variabile compuse (struct -uri)	255
8.2.1.4	Pointeri	257
8.2.2	Formate text	257
8.2.3	Probleme de robusteţe şi securitate	257
8.2.4	Probleme privind costul apelurilor sistem	258
8.3	Probleme de concurenţă în comunicaţie	260
9	Reţele IEEE 802	263
9.1	Reţele IEEE 802.3 (Ethernet)	263
9.1.1	Legături punct la punct prin perechi de conductoare	266
9.1.2	Legături prin fibre optice	272
9.1.3	Legături prin cablu magistrală	274
9.1.4	Repetoarele şi comutatoarele	277
9.1.5	Dirijarea efectuată de comutatoare (switch-uri)	279
9.1.6	Facilităţi avansate ale switch-urilor	279
9.1.6.1	Switch-uri configurabile	279
9.1.6.2	Filtrare pe bază de adrese MAC	280
9.1.6.3	Trunking	280
9.1.6.4	Legături redundante	281
9.1.6.5	Reţele virtuale (VLAN)	281
9.1.7	Considerente privind proiectarea unei reţele	282
9.2	Reţele IEEE 802.11 (Wireless)	283
9.2.1	Arhitectura reţelei	283
9.2.2	Accesul la mediu	285
9.2.3	Generarea pachetelor <i>beacon</i>	286
9.2.4	Securitatea reţelelor 802.11	286
10	Internetul	291
10.1	Arhitectura reţelei	291
10.2	Protocolul IP	292
10.2.1	Structura pachetului IP	293
10.2.2	Bazele dirijării pachetelor IP	294
10.2.2.1	Subreţele şi interfeţe	294
10.2.2.2	Prefixul de reţea	295
10.2.2.3	Tabela de dirijare	296
10.2.3	Scrierea ca text a adreselor şi prefixelor	298
10.2.3.1	Scrierea adreselor IP	298
10.2.3.2	Scrierea prefixelor de reţea	300
10.2.4	Alocarea adreselor IP şi prefixelor de reţea	300
10.2.4.1	Alocarea pe utilizări	301
10.2.4.2	Alocarea adreselor şi dirijarea ierarhică	301

10.2.5	Erori la dirijare și protocolul ICMP	302
10.2.5.1	Pachete nelivrabile	303
10.2.5.2	Diagnosticarea funcționării rutelor	305
10.2.5.3	Ciclarea pachetelor IP	305
10.2.5.4	Congestia	306
10.2.5.5	Redirecționarea	306
10.2.6	Alte chestiuni privind dirijarea pachetelor	307
10.2.6.1	Dimensiunea maximă a pachetelor și fragmentarea	307
10.2.6.2	Calitatea serviciului	308
10.2.7	Configurarea și testarea unei rețele IP locale	309
10.2.7.1	Alegerea parametrilor	309
10.2.7.2	Configurarea parametrilor de rețea pe diverse sisteme de operare	312
10.2.7.3	Testarea și depanarea rețelelor	313
10.3	Nivelul transport	314
10.3.1	Conexiuni cu livrare garantată: protocolul TCP	314
10.3.1.1	Principiul conexiunii TCP	315
10.3.1.2	Comunicația bidirecțională	320
10.3.1.3	Deschiderea și închiderea conexiunii	320
10.3.1.4	Alegerea numărului inițial de secvență	323
10.3.1.5	Închiderea forțată a conexiunii	324
10.3.1.6	Identificarea aplicației destinație	325
10.3.1.7	Correspondența între funcțiile <code>socket()</code> și acțiunile modului TCP	326
10.3.1.8	Controlul fluxului	327
10.3.1.9	Stabilirea time-out-ului pentru retransmiterea pachetelor	327
10.3.1.10	Algoritmul lui Nagle și optimizarea numărului de pachete	328
10.3.1.11	Trimiterea datelor speciale (out of band)	328
10.3.2	Datagrame nesigure: UDP	329
10.4	Identificarea nodurilor după nume: sistemul DNS	330
10.4.1	Numele de domeniu	330
10.4.2	Structura logică a bazei de date DNS	332
10.4.3	Împărțirea în domenii de autoritate	333
10.4.4	Mecanismul de interogare a serverelor	334
10.4.5	Sincronizarea serverelor pentru un domeniu	335
10.4.6	Căutarea numelui după IP	336
10.5	Legăturile directe între nodurile IP	337
10.5.1	Rezolvarea adresei — ARP	337
10.6	Configurarea automată a stațiilor — DHCP	339
10.7	Situații speciale în dirijarea pachetelor	341
10.7.1	Filtre de pachete (firewall)	341
10.7.2	Rețele private	346
10.7.3	Translația adreselor (NAT)	347
10.7.3.1	Translația adresei sursă	347

10.7.3.2	Translația adresei destinație	350
10.7.4	Tunelarea	351
11	Aplicații în rețele	353
11.1	Poșta electronică	353
11.1.1	Formatul mesajelor	354
11.1.1.1	Antetul mesajelor	355
11.1.1.2	Extensii MIME	358
11.1.1.3	Atașarea fișierelor și mesaje din mai multe părți	359
11.1.1.4	Codificarea corpului mesajului și a atașamentelor	360
11.1.2	Transmiterea mesajelor	362
11.1.2.1	Protocolul SMTP	362
11.1.2.2	Determinarea următorului MTA	365
11.1.2.3	Configurarea unui MTA	366
11.1.3	Securitatea poștei electronice	368
11.2	Sesiuni interactive la distanță	371
11.2.1	Protocolul <i>ssh</i>	373
11.2.1.1	Conexiunea <i>ssh</i> protejată criptografic	373
11.2.1.2	Metode de autentificare în <i>ssh</i>	376
11.2.1.3	Multiplexarea conexiunii, tunelarea și aplicații	379
11.2.2	Sistemul X-Window	379
11.3	Transferul fișierelor în rețea	380
11.3.1	Protocolul <i>ftp</i>	381
11.3.2	Protocolul HTTP	382
11.3.2.1	Structura cererilor și a răspunsurilor	383
11.3.2.2	URL-urile	384
11.3.2.3	Alte facilități HTTP	385
11.3.2.4	Proxy HTTP	386
11.3.2.5	Conexiuni securizate: SSL/TLS	387
11.3.2.6	Utilizarea TLS pentru web	389
11.4	PGP/GPG	390
11.4.1	Structura cheilor GnuPG	390
11.4.1.1	Chei primare și subchei	391
11.4.1.2	Utilizatori și identități	392
11.4.1.3	Generarea și modificarea cheilor	392
11.4.1.4	Controlul perioadei de valabilitate a cheilor	393
11.4.1.5	Gestiunea cheilor secrete	395
11.4.2	Transmiterea și certificarea cheilor publice	395
11.4.2.1	Transmiterea cheilor publice	395
11.4.2.2	Verificarea autenticității cheilor	397
11.4.3	Transmiterea mesajelor criptate sau semnate	399
	Bibliografie	401
	Index	405

Capitolul 7

Codificări de interes practic

7.1. Probleme privind reprezentarea numerelor întregi

Până aici am privit un mesaj transmis printr-o rețea ca un șir de simboluri dintr-un alfabet finit. Între simbolurile ce alcătuiesc șirul se stabilește o *ordine*, existând un prim element al șirului, un al doilea, etc. Transmisia elementelor se face în ordinea în care apar ele în șir, primul simbol transmis fiind cel care ocupă prima poziție din șir. Până aici am considerat că transmisia unui șir de la un dispozitiv la altul conservă ordinea între elemente.

Așa cum vom vedea însă în paragraful de față, din rațiuni legate de standardizarea reprezentării numerelor întregi, transmiterea unui șir nu conservă întotdeauna ordinea elementelor. În cele ce urmează, vom examina interferențele între reprezentarea numerelor în calculator și ordinea simbolurilor ce alcătuiesc un mesaj. Vom oferi cititorului o perspectivă, inspirată din [Cohen 1980] și mai puțin întâlnită în alte lucrări, asupra relațiilor dintre biți, octeți și reprezentarea numerelor.

7.1.1. Reprezentări pe biți

În paragraful de față vom face abstracție de anumite complicații constructive ale sistemelor de calcul reale. Vom considera reprezentări pe biți, ignorând deocamdată aspectele legate de gruparea biților în octeți și de faptul că, în memoria calculatorului, adresele identifică octeți și nu biți.

Ca urmare, rugăm cititorul să uite, pentru moment, noțiunea de *octet* (*byte*).

7.1.1.1. Bitul

Pentru reprezentarea diverselor date, alegerea unui alfabet cu două simboluri este avantajoasă din două motive. Pe de o parte, este cel mai mic alfabet posibil, ca urmare alegerea unui alfabet cu două elemente aduce o anumită simplitate și naturalețe construcției matematice. Pe de altă parte, din punct de vedere practic, al construcției echipamentelor fizice, dispozitive cu două stări stabile sunt mult mai ușor de construit decât dispozitive cu mai multe stări.

În scris, cele două simboluri sunt notate în mod obișnuit cu 0 și 1. Atragem atenția că:

- Alegerea celor două simboluri utilizate, precum și a corespondenței dintre starea dispozitivului fizic și simbolul asociat, poate fi făcută oricum. Odată însă făcută o alegere, aceasta trebuie respectată de toate entitățile implicate în comunicație.
- Numai în unele cazuri simbolurile au rol de cifră, adică au asociate valori numerice. Valoarea numerică a unui simbol este importantă doar dacă simbolul este interpretat ca număr sau intră în reprezentarea unui număr. În restul cazurilor, este important doar să existe simboluri distincte.

Un simbol dintr-un alfabet cu două elemente se numește *bit*, de la *binary digit* (rom. *cifră binară*).

7.1.1.2. Șiruri de biți

În cadrul unui șir de biți, avem nevoie să identificăm fiecare bit al șirului. Pentru aceasta, se stabilește o *ordine* a biților: avem un prim bit, un al doilea bit, etc. Trebuie remarcat însă că ordinea este o convenție: nu există o legătură directă între ordinea convențională a unui șir de biți și amplasamentul dispozitivelor fizice care memorează acei biți. Numărul de ordine al unui bit, în cadrul acestei ordini convenționale, se numește în mod obișnuit *poziția* (sau, eventual, *adresa* sau *deplasamentul*) bitului. Numerotarea pozițiilor se face de obicei începând de la 0 sau de la 1; în cele ce urmează vom utiliza numerotarea de la 0.

La transmiterea unui șir de biți, este natural ca primul bit transmis, considerând ordinea cronologică, să fie primul bit al șirului (poziția 0). La memorarea unui șir într-o memorie cu acces direct (memorie RAM sau fișier pe disc), este natural ca în celula cu adresa cea mai mică, dintre celulele alocate șirului, să fie plasat primul bit al șirului (bitul de pe poziția 0). În acest fel, *primul bit* al unui șir înseamnă, simultan, bitul de pe poziția (convențională) 0, bitul transmis primul (cronologic) și bitul memorat la adresa cea mai mică.

7.1.1.3. Reprezentarea pe biți a numerelor întregi

Reprezentarea numerelor naturale prin șiruri de biți se bazează pe ceea ce matematicienii numesc *reprezentare pozițională în baza 2*, pe care o presupunem cunoscută. În reprezentarea într-o bază de numerație, distingem cifra unităților, având ponderea $2^0 = 1$, cifra de pondere $2^1 = 2$ (în baza zece s-ar numi cifra zecilor; pentru baza 2 nu avem un nume), cifra de pondere $2^2 = 4$, etc.

Există două alegeri posibile cu privire la legătura dintre ponderile cifrelor în număr și pozițiile lor în șir:

1. *Primul bit al șirului este cifra de pondere maximă.* Aceasta alegere este identică celei utilizate în scrierea numerelor în limbile „obișnuite“ (indo-europene), cu scriere de la stânga spre dreapta. Se mai numește *big endian*.

În această schemă de reprezentare, un șir de biți $b_0b_1\dots b_{n-1}$ reprezintă numărul

$$b_0 \cdot 2^{n-1} + b_1 \cdot 2^{n-2} + \dots + b_{n-1} \cdot 2^0.$$

2. *Primul bit al șirului este cifra de pondere 1.* Această reprezentare este asemănătoare scrierii numerelor în limbile semite (araba și ebraica, cu scriere de la dreapta spre stânga și unde numerele sunt scrise tot cu cifra unităților în dreapta). Se mai numește *little endian*.

Această alegere are avantajul unei scrieri mai simple a relației dintre valoarea numărului și șirul de biți: valoarea unui număr reprezentat pe n biți este

$$b_0 \cdot 2^0 + b_1 \cdot 2^1 + \dots + b_{n-1} \cdot 2^{n-1}.$$

Fiind două scheme de reprezentare distincte, dacă un sistem transmite un număr în reprezentare *little endian*, iar celălalt sistem interpretează șirul de biți primit ca fiind într-o reprezentare *big endian*, receptorul „înțelege“ alt număr decât cel transmis de emițător. Ca urmare, orice protocol care specifică transmitere binară a numerelor trebuie să precizeze dacă se utilizează o reprezentare *little endian* sau una *big endian*.

EXEMPLUL 7.1: Fie șirul de biți 11001, în care am scris primul bit (în sensul din § 7.1.1.2) pe poziția cea mai din stânga.

Dacă acest șir este reprezentarea *big endian* a unui număr, numărul respectiv este 25. Dacă reprezentarea a fost făcută în format *little endian*, numărul este 19.

Este important de remarcat că distincția dintre schema de reprezentare *big endian* și schema *little endian* există numai acolo unde pe de o parte avem o ordine a biților dată de adresele lor în memorie sau de ordinea cronologică la transmiterea lor prin mediul fizic, iar pe de altă parte fiecare bit are o anumită pondere în reprezentarea unui număr întreg.

7.1.2. Reprezentări pe octeți

În paragraful precedent, am ignorat în mod deliberat noțiunea de *octet (byte)* și am presupus că, în memorie, fiecare bit ar avea o adresă individuală. Vom studia, în continuare, problemele legate de gruparea, în cadrul sistemelor de calcul reale, a biților în octeți și de faptul că, în general, ordinea biților în octeți nu este vizibilă programatorului.

7.1.2.1. Octeți

În memoria calculatoarelor, biții sunt grupați în grupuri de dimensiune fixă, în mod obișnuit câte 8 biți. Un astfel de grup se numește *octet* (denumire intrată pe filieră franceză) sau *bait* (adaptare a englezescului *byte*).

Un octet poate fi privit în două moduri distincte:

- ca un șir de 8 biți,
- ca un număr întreg cuprins între 0 și 255.

Echivalența între aceste două moduri de-a privi un octet este o problemă ce necesită multă atenție. Dacă identificăm biții după ponderile lor, există o corespondență biunivocă între un astfel de grup de 8 biți și un număr întreg între 0 și 255. Dacă însă identificăm biții după poziția lor în șirul de 8 biți (bitul 0, bitul 1, ..., bitul 7), atunci corespondența biunivocă între număr și șir de biți există doar după ce am stabilit o corespondență între poziția unui bit în șir și ponderea sa (*big endian*, *little endian* sau eventual o corespondență mai complicată).

După modul în care se identifică biții unui octet în definiția operațiilor efectuate de diverse componente ale unui sistem de calcul, operațiile se pot împărți în trei categorii:

1. *Operații pentru care biții se identifică după pondere* sau, echivalent, octetul este privit ca număr. Aici se încadrează operațiile aritmetice și operațiile de deplasare pe biți. De remarcat că operațiile *deplasare la stânga* (engl. *shift left*), respectiv *deplasare la dreapta* (engl. *shift right*) pot fi descrise în termeni de operații aritmetice: deplasarea la stânga este o înmulțire cu 2, iar deplasarea la dreapta este o împărțire la 2. În acest context, „spre stânga” și „spre dreapta” înseamnă spre pozițiile cu pondere mai mare, respectiv mai mică, neavând nici o legătură cu

primele sau cu ultimele poziții. Aceste operații sunt efectuate de unitatea aritmetică din microprocesorul calculatorului.

2. *Operații pentru care biții sunt identificați după numărul lor de ordine* sau, echivalent, octetul este privit ca un șir arbitrar de biți, fără a avea asociată o valoare numerică. Aici intră transmiterea bit cu bit (transmitere serială) a unui octet. Această operație este efectuată de placa de rețea și de alte adaptoare seriale (de exemplu, adaptoarele USB). Tot aici s-ar încadra, dacă ar exista, o operație de obținere sau de modificare a unui bit (al octetului) identificat prin numărul său de ordine. O asemenea operație nu este oferită, în mod normal, într-un sistem de calcul — nu există o instrucțiune care să extragă, de exemplu, bitul numărul 5 dintr-un octet.
3. *Operații care pot fi definite fie identificând biții după ponderea lor, fie identificând biții după numărul lor de ordine.* În această categorie se încadrează transmiterea unui octet ca un tot unitar, verificarea egalității a doi octeți și operațiile logice pe bit (*și, sau, sau exclusiv și negația*).

Pentru oricare dintre aceste operații, dacă definim operația identificând biții după numărul lor de ordine, efectul ei asupra valorii numerice a octetului nu depinde de corespondența aleasă între pozițiile biților și ponderile lor.

În aceste condiții, în interiorul unui calculator, biții din cadrul unui octet sunt identificați după ponderea lor. În construcția calculatorului, proiectantul are grijă ca atunci când un bit având, într-un modul al calculatorului, o anumită pondere este transferat către alt modul al calculatorului, să ajungă acolo pe o poziție cu aceeași pondere.

La transmisia unui octet între două sisteme de calcul, mecanismele de transmisie sunt astfel construite încât să transmită valoarea octetului. Întrucât, prin mediul fizic al rețelei, biții sunt transmiși secvențial, biții unui octet sunt aici identificați prin numărul lor de ordine în cadrul transmisiei. Pentru a păstra valoarea octetului în timpul transmisiei prin mediul rețelei, corespondența dintre numărul de ordine al unui bit și ponderea sa (*little endian* sau *big endian*) trebuie să facă parte din specificațiile protocolului de nivel fizic al rețelei.

Numerotarea biților unui octet intervine, de asemenea, în descrierea unor scheme de reprezentare a datelor unde un număr este reprezentat pe un grup de biți ce nu formează un număr întreg de octeți. Este cazul schemelor de reprezentare pentru structuri de date ce conțin câmpuri de 1 bit, 2 biți, 12 biți, etc. Și aici este necesar să se specifice dacă numerotarea biților este *little*

endian sau *big endian*. Mai multe detalii despre astfel de reprezentări vor fi studiate în § 7.1.2.4.

7.1.2.2. Șiruri de octeți

Ca și în cazul biților (vezi § 7.1.1.2), și cu octeții putem construi șiruri. În cadrul unui șir de octeți, octeții sunt așezați într-o ordine, existând un prim octet (numerotat ca octetul 0), al doilea octet (poziția 1), etc. La transmisia printr-o legătură în rețea, primul octet al șirului este, cronologic, primul octet transmis. La memorare, primul octet este cel memorat la adresa cea mai mică.

În virtutea celor două moduri de-a privi un octet, un șir de n octeți poate fi, la rândul lui, privit ca:

- un șir de $8n$ biți,
- un șir de n numere, fiecare cuprins între 0 și 255.

Pentru a putea privi un șir de n octeți ca un șir de $8n$ biți, este necesar să avem o numerotare, bine definită, a biților în cadrul unui octet. Rezultă un șir de biți în care între poziția p_B a unui bit în șirul de $8n$ biți, poziția p_{BO} a bitului în cadrul octetului în care se găsește și poziția p_O a celui octet în șirul de octeți are loc relația:

$$p_B = 8 \cdot p_O + p_{BO}. \quad (7.1)$$

Relația de mai sus are această formă simplă dacă se utilizează numerotare de la 0; pentru numerotarea de la 1, forma relației e mai complicată.

Transmiterea unui șir de octeți printr-o conexiune, precum și memorarea șirului într-un fișier pe disc urmată de citirea lui înapoi în memorie, păstrează ordinea și valorile octeților din șir. Valorile octeților sunt păstrate dacă privim octeții ca numere între 0 și 255; dacă privim octeții ca șiruri de 8 biți, valorile octeților se păstrează numai dacă pe ambele sisteme utilizăm aceeași corespondență între numerele de ordine și ponderile biților.

7.1.2.3. Reprezentarea numerelor pe un număr întreg de octeți

Cel mai mare număr ce poate fi reprezentat pe un octet este 255, ceea ce este mult prea puțin pentru majoritatea aplicațiilor. Pentru a putea reprezenta numere din intervale mai largi, sunt necesare scheme de reprezentare pe mai mult de 8 biți. Schemele cele mai simple sunt cele care utilizează un număr întreg de octeți; acestea vor fi prezentate în continuare. Schemele de reprezentare ce utilizează șiruri de biți ce nu formează neapărat un număr întreg de octeți vor fi studiate în § 7.1.2.4.

Deoarece un octet are valoarea între 0 și 255, putem considera fiecare octet ca fiind o cifră în baza 256. Reprezentarea unui număr printr-un șir de octeți se face ca reprezentare pozițională în baza 256. Există două reprezentări posibile:

- *little endian*: primul octet are ponderea 1, al doilea octet are ponderea 256, al treilea octet are ponderea $256^2 = 65536$, etc.
- *big endian*: primul octet are ponderea 256^{n-1} (unde n este numărul de octeți ai reprezentării), al doilea octet are ponderea 256^{n-2} ș. a. m. d., penultimul octet are ponderea 256, iar ultimul octet are ponderea 1.

Reamintim că prin *primul octet* înțelegem octetul care este transmis primul, în ordine cronologică, de la un dispozitiv la altul și, totodată, octetul memorat la adresa cea mai mică.

EXEMPLUL 7.2: Descriem mai jos reprezentarea numărului 300 în schemele de reprezentare *little endian* și *big endian*, pe 2 și pe 4 octeți. Pentru fiecare dintre aceste patru scheme de codificare, este dat șirul de octeți ce reprezintă numărul 300.

poziție octet (nr. ordine)	Valorile octeților pentru diverse reprezentări			
	2 octeți big endian	2 octeți little endian	4 octeți big endian	4 octeți little endian
0	1	44	0	44
1	44	1	0	1
2	—	—	1	0
3	—	—	44	0

De exemplu, în cadrul reprezentării pe 2 octeți în format *big endian*, valoarea numărului reprezentat se regăsește ca valoarea octetului 0 înmulțită cu 256 plus valoarea octetului 1, anume: $1 \cdot 256 + 44 = 300$. În cadrul reprezentării pe 4 octeți în format *little endian*, octetul 0 are ponderea $256^0 = 1$, octetul 1 are ponderea $256^1 = 256$, octetul 2 are ponderea $256^2 = 65536$, iar octetul 3 are ponderea $256^3 = 16218368$. Valoarea numărului reprezentat este

$$44 \cdot 1 + 1 \cdot 256 + 0 \cdot 256^2 + 0 \cdot 256^3 = 300$$

Unitatea aritmetică a calculatorului poate efectua operații aritmetice asupra numerelor reprezentate pe 2 octeți sau, pentru unele calculatoare, pe 4 sau 8 octeți. Pe unele calculatoare, unitatea aritmetică lucrează cu numere reprezentate după sistemul *big endian*; pe alte calculatoare, unitatea

aritmetică cere reprezentare *little endian*. După acest criteriu, calculatoarele ale căror unități aritmetice lucrează cu numere reprezentate pe mai mult de un octet se împart în calculatoare *little endian* și calculatoare *big endian*.

Variabilele de tip întreg, în majoritatea limbajelor de programare, sunt reprezentate pe 2, 4 sau 8 octeți, în ordinea fixată de unitatea aritmetică.

Este posibilă utilizarea, pentru anumite variabile întregi, a unei reprezentări diferite de cea a unității aritmetice. De asemenea, se pot utiliza reprezentări pe mai mulți octeți decât permite unitatea aritmetică. La manipularea acestor variabile, programatorul trebuie să aibă în vedere că operațiile aritmetice „normale“ fie nu pot fi executate deloc, fie nu se efectuează corect asupra lor. Astfel de numere se manipulează, de obicei, prelucrând separat fiecare octet.

La memorarea pe disc sau la transmiterea printr-o conexiune, trebuie stabilit printr-un standard dacă se utilizează un format *little endian* sau *big endian*, precum și numărul de octeți pe care se reprezintă fiecare număr memorat sau, respectiv, transmis. Majoritatea protocoalelor pentru Internet prevăd formate *big endian* pentru numerele întregi transmise. Multe dintre formatele de fișiere prevăd formate *little endian*. Există și formate (de exemplu, formatul TIFF pentru imagini, formatele UTF-16 și UTF-32 pentru texte) care permit emițătorului să aleagă formatul dorit și prevăd un mecanism prin care emițătorul informează receptorului despre alegerea făcută.

Dacă formatul de pe disc sau de pe conexiune coincide cu formatul unității aritmetice locale, un program poate transfera direct șiruri de octeți între o variabilă întregă locală și fișierul de pe disc sau, respectiv, conexiunea spre celălalt calculator. Dacă formatul de pe disc sau de pe conexiune este invers față de cel local, un program care transferă date trebuie să inverseze ordinea octeților imediat înainte de scrierea pe disc sau de trimiterea pe conexiune, precum și imediat după citirea de pe disc sau recepționarea de pe conexiune.

7.1.2.4. Reprezentarea numerelor pe un șir arbitrar de biți

Ne vom ocupa în continuare de metode de reprezentare, pentru numere întregi, în care biții ce intră în reprezentarea unui număr nu formează neapărat un număr întreg de octeți. O astfel de schemă este o generalizare a schemei prezentate în paragraful precedent.

O astfel de metodă de reprezentare se bazează pe reprezentarea numerelor în baza 2 (vezi și § 7.1.1.3). Pentru ca o astfel de schemă să fie complet definită, este necesar să fie stabilită (standardizată) corespondența dintre poziția fiecărui bit din reprezentare și ponderea asociată. În descrierea

unei astfel de scheme de reprezentare, trebuie precizate trei lucruri:

- dacă reprezentarea numărului prin șirul de biți se face după metoda *big endian* sau *little endian*;
- dacă numerotarea biților în cadrul fiecărui octet se face începând de la bitul de pondere 1 (adică valoarea octetului este reprezentată după schema *little endian*) sau începând de la bitul de pondere 128 (adică valoarea octetului este reprezentată după schema *big endian*).
- dacă numerotarea biților în cadrul șirului de biți se face după relația (7.1) sau după o altă metodă.

Într-o schemă de reprezentare „rațională“, la primele două puncte se utilizează fie formatul *big endian* pentru amândouă, fie formatul *little endian* pentru amândouă, iar la punctul al treilea se utilizează relația (7.1).

Rezultă astfel două metode coerente:

- *big endian*. În această metodă, numerotarea biților începe cu cel mai semnificativ bit al primului octet, iar după cel mai puțin semnificativ bit al primului octet urmează cel mai semnificativ bit al celui de-al doilea octet. Orice număr, indiferent pe câți biți s-ar reprezenta, se reprezintă în format *big endian*.
- *little endian*. În această metodă, numerotarea biților începe cu cel mai puțin semnificativ bit al primului octet, iar după cel mai semnificativ bit al primului octet urmează cel mai puțin semnificativ bit al celui de-al doilea octet. Orice număr, indiferent pe câți biți s-ar reprezenta, se reprezintă în format *little endian*.

În cadrul acestor două metode, dacă reprezentăm un număr folosind un șir de biți ce formează un număr întreg de octeți, formatele de reprezentare rezultate coincid cu formatele de reprezentare pe octeți studiate în paragraful precedent.

EXEMPLUL 7.3: Considerăm o schemă de reprezentare pentru două numere întregi, a și b , în care a se reprezintă pe 4 biți și b se reprezintă pe 12 biți. În total avem $4 + 12 = 16$ biți, adică 2 octeți.

Dacă alegem metoda *big endian*, schema de reprezentare va utiliza cei mai semnificativi 4 biți ai primului octet pentru a-l reprezenta pe a , ceilalți 4 biți ai primului octet vor fi cei mai semnificativi 4 biți ai lui b , iar cel de-al doilea octet va conține cei mai puțin semnificativi 8 biți din b . Această schemă de reprezentare este ilustrată în figura 7.1, cu valori concrete $a = 11$ și $b = 300$.

Dacă alegem metoda *little endian*, schema de reprezentare va utiliza cei mai puțin semnificativi 4 biți ai primului octet pentru a-l reprezenta pe a , ceilalți 4 biți ai primului octet vor fi cei mai puțin semnificativi 4 biți ai lui b ,

iar cel de-al doilea octet va conține cei mai semnificativi 8 biți din b . Această schemă de reprezentare este ilustrată în figura 7.2, cu valori concrete $a = 11$ și $b = 300$.

b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
1	0	1	1	0	0	0	1	0	0	1	0	1	1	0	0

(a) Reprezentarea privită ca șir de biți

Nr. octet	Valoare (binar)								Valoare (zecimal)
	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	
0	1	0	1	1	0	0	0	1	177
1	0	0	1	0	1	1	0	0	44

(b) Valorile octeților. La scrierea valorilor biților în octet (coloana din mijloc) s-a utilizat convenția obișnuită, de-a scrie cifrele mai semnificative în stânga.

Figura 7.1: Reprezentare *big endian* pentru numărul 11 pe 4 biți urmat de numărul 300 reprezentat pe 12 biți (exemplul 7.3).

b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
1	1	0	1	0	0	1	1	0	1	0	0	1	0	0	0

(a) Reprezentarea privită ca șir de biți

Nr. octet	Valoare (binar)								Valoare (zecimal)
	c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0	
0	1	1	0	0	1	0	1	1	203
1	0	0	0	1	0	0	1	0	18

(b) Valorile octeților. La scrierea valorilor biților în octet (coloana din mijloc) s-a utilizat convenția obișnuită, de-a scrie cifrele mai semnificative în stânga.

Figura 7.2: Reprezentare *little endian* pentru numărul 11 pe 4 biți urmat de numărul 300 reprezentat pe 12 biți (exemplul 7.3).

Un alt exemplu în care avem de-a face cu numere reprezentate pe șiruri arbitrare de biți este legat de așa-zisa *codificare în baza 64*, descrisă în § 7.4.2.

7.1.3. Probleme privind reprezentarea lungimii șirurilor

Oridecâteori se transmite un șir de obiecte, este necesar ca receptorul să poată determina numărul de obiecte transmise. Acest lucru este valabil

indiferent de natura obiectelor: biți, octeți, cifre zecimale, caractere ale unui text, numere în cadrul unui șir de numere.

Există trei metode de a face ca receptorul să poată determina numărul de obiecte ce-i sunt transmise:

- *Numărul de obiecte este fixat.* În acest caz, indiferent de valorile datelor ce se transmit, numărul de obiecte transmise este același. Metoda este utilizată frecvent la memorarea unui șir în memoria RAM sau pe disc, deoarece permite alocarea de la început a memoriei pentru reprezentarea lui și permite accesul direct la datele memorate după șirul în discuție. Dezavantajul principal al metodei este acela că dimensiunea fixată trebuie astfel aleasă încât să fie suficientă în orice caz ce poate să apară la execuție. De asemenea, trebuie să existe o valoare potrivită pentru pozițiile „neutilizate“ din șir; de exemplu, în reprezentarea numerelor, pozițiile cele mai semnificative se completează cu zerouri.

Deoarece la transmisia printr-o conexiune nu se poate pune problema accesului direct (adică altfel decât secvențial) la date, metoda este puțin utilizată în transmisia datelor prin rețea. Șiruri de lungime fixă se utilizează la reprezentarea binară a numerelor.

- *Numărul de obiecte este transmis separat, în fața șirului.* Această metodă ușurează munca receptorului, care știe exact ce să aștepte și poate alocă memorie pentru recepționarea datelor. În schimb, munca emițătorului este complicată prin faptul că acesta trebuie să cunoască de la început numărul de obiecte din șir. Acest fapt face metoda inaplicabilă în anumite situații.

Transmiterea de la început a numărului de obiecte este utilizată, de exemplu, de protocolul *HTTP* (§ 11.3.2) la transmiterea, de către server, a conținutului paginii cerute de client. Serverul transmite întâi numărul de octeți ai paginii și apoi șirul de octeți ce formează pagina.

- *După ultimul obiect din șirul propriu-zis, se transmite o valoare specială, cu rol de terminator.* Această metodă ușurează munca emițătorului, care poate să înceapă transmiterea șirului înainte de-a ști câte elemente are, în schimb îngreunează munca receptorului, care trebuie să citească elementele șirului unu câte unu și să verifice dacă nu a întâlnit terminatorul. De asemenea, trebuie fixată valoarea terminatorului, care trebuie să fie o valoare reprezentabilă în formatul pentru element, dar care nu apare niciodată ca valoare a unui element valid.

Metoda este utilizată frecvent în transmiterea unui șir de caractere. Rolul de terminator poate fi acordat caracterului *null* (caracterul cu codul ASCII zero), caracterului *newline* (sfârșit de rând), caracterului

spațiu, etc. Orice alegere s-ar face, caracterul sau caracterele astfel alese pentru a marca sfârșitul unui șir nu pot să apară în șirul propriu-zis. Ca urmare, transmiterea unui fișier cu conținut arbitrar (șir de octeți cu valori arbitrare) nu se poate face prin metoda cu terminator (decât dacă un octet al fișierului se codifică pe mai mult de 8 biți).

7.1.4. Alte metode de reprezentare a numerelor întregi

Schemele de reprezentare (formatele) pentru numere întregi, studiate în § 7.1.2.3 și § 7.1.2.4, se numesc formate binare. Pe lângă formatele binare, pentru reprezentarea numerelor întregi se mai utilizează următoarele tipuri de formate:

- Formatul *text*. În cadrul acestui format, reprezentarea numărului este un text format din caracterele corespunzătoare cifrelor reprezentării zecimale (obișnuite) a numărului.
- Formatul *binar-zecimal*, numit și *BCD* din engl. *binary coded decimal*. În cadrul acestui format, numărul este reprezentat mai întâi în baza 10, iar apoi fiecare cifră zecimală este reprezentată pe 4 biți conform metodelor din § 7.1.2.4.

Descriem puțin mai pe larg reprezentarea numerelor în format text, deoarece o astfel de reprezentare se utilizează frecvent în protocoalele în rețea. Motivul principal al utilizării formatului text este ușurința depanării aplicațiilor ce utilizează astfel de protocoale: comunicația poate fi înregistrată într-un fișier și examinată cu un program obișnuit pentru vizualizarea fișierelor text.

În format text, se utilizează convențiile de reprezentare a numerelor în textele scrise: se începe cu cifra cea mai semnificativă, numărul de cifre este variabil (depinde de valoarea numărului) și prima cifră (cea mai semnificativă) scrisă nu este zero, cu excepția cazului numărului 0 care se reprezintă ca o singură cifră zero.

Fiecare cifră se reprezintă ca un caracter, fiind necesară mai departe o schemă de reprezentare a textelor (vezi § 7.2). În cazul codificării ASCII, reprezentarea fiecărei cifre ocupă exact un octet. De remarcat însă că, în acest caz, cifra 0 nu se reprezintă ca un octet cu valoarea 0, ci ca un octet având ca valoare codul ASCII pentru caracterul „0”; acesta este 48 (sau, echivalent, 30₁₆).

Deoarece lungimea reprezentării este variabilă, este necesar să fie transmisă sub o formă sau alta informația privind lungimea reprezentării numărului (numărul de cifre). În acest scop, în reprezentările text, numerele sunt separate de obicei prin spații, caractere *tab*, caractere *newline* etc.

EXEMPLUL 7.4: Redăm mai jos reprezentările text ASCII terminat cu spațiu, BCD *big endian* pe 4 octeți și BCD *little endian* pe 4 octeți, pentru numărul 300. Valorile octeților sunt scrise în baza 2, bitul cel mai semnificativ fiind scris în stânga.

poziție octet (nr. ordine)	Valorile octeților		
	text ASCII	BCD <i>big endian</i>	BCD <i>little endian</i>
0	00110011	00000000	00000000
1	00110000	00000000	00000011
2	00110000	00000011	00000000
3	00100000	00000000	00000000

7.2. Codificarea textelor

Prin text înțelegem aici un text scris în limbaj natural sau într-un limbaj de programare, fără formatare avansată.

Un text este văzut în general ca o succesiune de *caractere*. Caracterele sunt în principal literele din alfabetul limbii în care este scris textul, semne de punctuație, cifre și diferite alte semne grafice. Nu se face distincție între diferitele variante de-a scrie o aceeași literă (litere „normale“, cursive („italice“), adline („bold“), etc).

Pe lângă caracterele grafice, descrise mai sus, sunt definite *caractere de control*, având rolul de a marca puncte (locuri) în cadrul textului sau fragmente din text. Utilizări ale caracterelor de control sunt, de exemplu, trecerea la rând nou sau interzicerea trecerii la rând nou (ruperea rândului) într-un anumit punct.

Un aspect discutabil legat de alegerea setului de caractere este dacă o literă cu un semn diacritic este cazul să fie caracter distinct față de litera simplă din care provine sau să fie format din caracterul corespunzător literei respective fără semne diacritice și un caracter de control care să marcheze semnul diacritic adăugat. În aceeași idee, s-ar putea face și distincția dintre literele mari (majuscule) și literele mici (minuscule) corespunzătoare tot pe baza unor caractere de control cu rol de modificador.

Operațiile efectuate asupra textelor, care trebuie să fie permise de codificarea aleasă, sunt în principal următoarele:

- afișarea textului;
- concatenarea unor texte sau alte operații de sinteză;

- căutarea unui cuvânt, extragerea unor cuvinte sau unor fragmente de text și diverse alte operații de analiză a textului;
- sortarea alfabetică.

De notat că regulile de sortare alfabetică sunt complexe și depind de limbă. De exemplu, în limba română, literele cu diacritice sunt considerate imediat după literele fără diacritice. Următoarele cuvinte sunt sortate alfabetic: *sac*, *suc*, *șiret*; de notat că orice cuvânt ce începe cu *ș* este sortat după toate cuvintele ce încep cu *s*. În franceză, însă, literele cu diacritice sunt considerate, în prima fază, echivalente cu cele fără diacritice, intervenind în ordinea alfabetică doar pentru cuvinte care diferă doar prin diacritice. Exemplu: *été*, *être*, *étude*; de notat că apar amestecate cuvinte ce încep cu *é* și *ê*.

Majoritatea codificărilor sunt bazate pe reprezentarea una după alta a literelor (caracterelor) ce formează cuvintele textului.

Codificările caracterelor sunt de obicei descrise în două etape. În prima etapă, fiecărui caracter îi este asociat un număr întreg pozitiv, numit *codul caracterului*. În a doua etapă, fiecărui cod de caracter îi este asociată o codificare ca șir de biți sau ca șir de octeți.

Pentru o schemă de codificare trebuie așadar specificate trei elemente:

- setul de caractere;
- numerotarea (codificarea) caracterelor;
- reprezentarea pe biți sau pe octeți a codurilor caracterelor.

7.2.1. Codificarea ASCII

Codificarea (codul) ASCII este codificarea cea mai des întâlnită pentru texte.

Setul de caractere cuprinde 128 caractere dintre care:

- 33 de caractere de control;
- caracterul *spațiu* (considerat de unii ca fiind caracter imprimabil și de alții ca fiind caracter de control);
- 94 de caractere imprimabile, cuprinzând: 52 de litere (cele 26 litere ale alfabetului latin, cu cele două forme, majuscule și minuscule) cele 10 cifre zecimale și un număr de 32 de semne de punctuație și alte simboluri.

Codurile asociate caracterelor ASCII sunt cuprinse între 0 și 127, caracterele de control primind codurile 0–31 și 127, spațiul are codul 32, iar

(celelalte) caractere imprimabile au codurile cuprinse între 33 și 126. Pentru a ușura sortarea alfabetică, codurile sunt grupate astfel:

- literele mari de la 65 (41 hexa) pentru A la 90 (5A hexa) pentru Z;
- literele mici de la 97 (61 hexa) pentru a la 122 (7A hexa) pentru z;
- cifrele de la 48 (30 hexa) pentru 0 la 57 (39 hexa) pentru 9.

De remarcat și că diferența dintre codul oricărei litere mici și codul literei mari corespunzătoare este 32 (20 hexa).

Pentru reprezentarea unui caracter ASCII sunt necesari doar 7 biți, însă cel mai adesea un caracter ASCII se reprezintă pe un octet, al cărui cel mai semnificativ bit este întotdeauna 0.

Datorită faptului că pe de o parte caracterele ASCII se reprezintă pe un octet, iar pe de altă parte că dintre caracterele de control multe nu sunt utilizate deloc în majoritatea aplicațiilor, rămân multe coduri reprezentabile (cca. 140) care nu sunt utilizate. Se poate extinde setul de caractere, asociind noilor caractere coduri între 128 și 255 sau coduri între 0 și 31 a căror caractere corespunzătoare nu sunt folosite efectiv. Toate aceste codificări rezultate se numesc generic *seturi ASCII extinse*.

7.2.2. Codificările ISO-8859

ISO-8859 este o familie de coduri, construite toate ca extensii (alternative) ale codificării ASCII.

Fiecare cod din familie cuprinde câte 256 caractere: cele 128 caractere ASCII, plus 128 de caractere alese pentru a acoperi alfabetul câte unui grup de limbi. Limbile acoperite de câteva dintre codificările ISO-8859 sunt:

- ISO-8859-1, alfabetul latin pentru limbile din vestul Europei;
- ISO-8859-2, alfabetul latin pentru limbile din estul Europei;
- ISO-8859-5, alfabetul chirilic;
- ISO-8859-6, alfabetul arab;
- ISO-8859-7, alfabetul grecesc;
- ISO-8859-8, alfabetul ebraic.

Codurile asociate caracterelor sunt codurile din codificarea ASCII pentru cele 128 de caractere din setul ASCII și numere de la 128 la 255 pentru caracterele suplimentare.

Reprezentarea pe octeți pentru un text ISO-8859-*n* se face cu câte un octet pentru fiecare caracter, octetul conținând codul caracterului.

Fiecare cod din familie este extensie a codului ASCII în sensul că mulțimea caracterelor din fiecare astfel de cod include mulțimea caracterelor

Caracter	Cod (hexa)	Caracter	Cod (hexa)
Ă	C3	ă	E3
Â	C2	â	E2
Î	CE	î	EE
Ș	AA	ș	BA
Ț	DE	ț	FE

Tabelul 7.1: Caracterele cu diacritice din alfabetul limbii române și codificările ISO-8859-2 corespunzătoare

ASCII și codurile asociate caracterelor comune cu setul ASCII coincid cu codurile ASCII. Ca urmare, un text ASCII este întotdeauna interpretat corect ca text ISO-8859- n . Pe de altă parte, un text scris în ISO-8859- n și interpretat ca ISO-8859- m va fi evident interpretat greșit.

Ordinea numerică a codurilor din oricare dintre codificările ISO-8859 este diferită de ordinea alfabetică. În general, în ordinea alfabetică, literele cu diacritice își au locul imediat lângă literele similare fără diacritice; în codificările ISO-8859-1 sau ISO-8859-2, de exemplu, literele cu diacritice au codurile mai mari de 128 în vreme ce literele fără diacritice au coduri între 65 și 123.

7.2.3. Codificările Unicode

Unicode este un set de caractere ce se dorește să cuprindă litere din toate scrierile de pe Pământ. Numărul de caractere din unicode este limitat, datorită modurilor de codificare definite, la aproximativ un milion (mai exact, la $110000_{16} = 1114112$). Nu toate aceste coduri sunt definite în prezent, codurile încă nedefinite putând fi definite în versiuni următoare ale standardului.

Codurile unicode sunt numere de la 0 la $2^{20} + 2^{16} - 1$. Codurile de la 0 la 127 corespund aceluiași caractere ca și în codificarea ASCII.

Reprezentarea codurilor unicode ca șiruri de octeți poate fi făcută în mai multe moduri. Cele mai răspândite codificări sunt:

- *UTF-8* este o codificare de lungime variabilă, între 1 și 4 octeți pentru un caracter;
- *UTF-16*, *UTF-16LE*, *UTF-16BE* sunt codificări de lungime variabilă, 2 sau 4 octeți pentru un caracter;
- *UTF-32*, *UTF-32LE*, *UTF-32BE* sunt codificări de lungime fixă, de 4 octeți pentru fiecare caracter.

Carac- ter	Cod <i>unicode</i> (hexa)	Cod <i>unicode</i> (zecimal)	UTF-8 (hexa)
Ă	102	258	C4 82
ă	103	259	C4 83
Â	C2	194	C3 82
â	E2	226	C3 A2
Î	CE	206	C3 8E
î	EE	238	C3 AE
Ș	218	536	C8 98
ș	219	537	C8 99
Ț	21A	538	C8 9A
ț	21B	539	C8 9B
Ş	15E	350	C5 9E
ş	15F	351	C5 9F
Ț	162	354	C5 A2
ț	163	355	C5 A3

Tabelul 7.2: Caracterele cu diacritice din alfabetul limbii române și codificările *unicode* corespunzătoare. Notă: caracterele Ș, ș, Ț și ț au câte două forme utilizate: una cu virgulă dedesupt, cealaltă cu sedilă. Conform normelor stabilite de Academia Română, forma corectă este cea cu virgulă. Codificarea formei cu virgulă a fost standardizată mai recent, motiv pentru care multe documente utilizează încă forma cu sedilă.

7.2.3.1. Codificarea UTF-8

Correspondența de la codul caracterului la șirul de octeți este dată în tabelul 7.3.

Valorile lui c (în baza 16)	reprezentarea UTF-8 (în baza 2)
0–7F	$0c_7c_6c_5c_4c_3c_2c_1c_0$
80–7FF	$110c_{10}c_9c_8c_7c_6$ $10c_5c_4c_3c_2c_1c_0$
800–FFFF	$1110c_{15}c_{14}c_{13}c_{12}$ $10c_{11}c_{10}c_9c_8c_7c_6$ $10c_5c_4c_3c_2c_1c_0$
10000–1FFFFF	$11110c_{20}c_{19}c_{18}$ $10c_{17}c_{16}c_{15}c_{14}c_{13}c_{12}$ $10c_{11}c_{10}c_9c_8c_7c_6$ $10c_5c_4c_3c_2c_1c_0$

Tabelul 7.3: Codificarea UTF-8. c reprezintă codul *unicode* al caracterului; $c_{20} \dots c_0$ reprezintă cifrele reprezentării binare a lui c , cu c_{20} reprezentând cifra cea mai semnificativă și c_0 cea mai puțin semnificativă. Codificarea există doar pentru $0 \leq c < 2^{21}$.

De remarcat că schema pentru coduri mari (de exemplu, schema pentru c între 80_{16} și $7FF_{16}$) poate fi principial aplicată și la coduri mai mici (de exemplu, pentru $c = 41_{16}$, rezultând doi octeți, $C1_{16}$ urmat de 81_{16}). O astfel de codificare este însă interzisă de standard pentru a asigura unicitatea codificării UTF-8.

Codificarea UTF-8 permite recuperarea sincronismului (dacă receptorul pierde câțiva octeți poate regăsi unde începe un caracter nou), deoarece fiecare caracter nou începe cu un octet cuprins între 0 și 127 sau între 192 și 255, iar ceilalți octeți din codificarea unui caracter sunt cuprinși între 128 și 191. O altă proprietate este că lungimea codificării UTF-8 a unui caracter poate fi determinată după citirea primului octet.

7.2.3.2. Codificările UTF-16

Codificarea UTF-16 este descrisă în două etape: într-o primă etapă, codul *unicode* este transformat într-unul sau două numere de câte 16 biți, iar în a doua etapă fiecare astfel de număr este scris ca 2 octeți consecutivi.

Caracterele cu codul *unicode* între 0 și $D7FF_{16}$ sau între $E000_{16}$ și $FFFF_{16}$ se scriu ca un singur întreg pe 16 biți.

Caracterele cu codul *unicode* între 10000_{16} și $10FFFF_{16}$ se scriu ca doi întregi de câte 16 biți astfel: Mai întâi, din codul *unicode* se scade 10000_{16} , rezultând o valoare între 0 și $FFFF_{16}$ (20 biți). Primul întreg de 16 biți se formează punând cifrele 110110 urmate de primii 10 din cei 20 de biți. Al doilea întreg se formează punând cifrele 110111 urmate de ultimii 10 din cei 20 de biți. De exemplu, codul *unicode* 10302_{16} se scrie ca doi întregi astfel: $D83C_{16}$ $DF02_{16}$.

Într-o a doua etapă este definită scrierea fiecărui întreg de 16 biți ca un șir de doi octeți. Există două modalități de a reprezenta fiecare astfel de întreg, începând de la octetul mai semnificativ (de rang mai mare) sau începând de la octetul mai puțin semnificativ. Pentru a reflecta aceste variante diferite de alegere, există trei codificări distincte numite generic *UTF-16*:

- *UTF-16LE*: Primul octet este cel mai puțin semnificativ (*little endian*);
- *UTF-16BE*: Primul octet este cel mai semnificativ (*big endian*);
- *UTF-16*: Ordinea octeților poate fi fie *big endian*, fie *little endian*, la alegerea emițătorului. Primul caracter codificat trebuie să fie caracterul cu codul $FEFF_{16}$ (definit inițial ca fiind un caracter de control ce interzice ruperea în rânduri în acel punct, dar este utilizat în prezent doar ca marcaj pentru identificarea ordinii octeților). Ordinea octeților este dedusă de receptor prin examinarea primilor doi octeți: dacă aceștia sunt FE_{16} urmat de FF_{16} , înseamnă că ordinea octeților este *big endian*; dacă este FF_{16} urmat de FE_{16} , înseamnă că ordinea octeților este *little endian*.

7.2.3.3. Codificările UTF-32

Codificarea UTF-32 constă în codificarea fiecărui caracter ca un întreg pe 32 de biți, reprezentat la rândul lui ca un șir de 4 octeți. Ca și în cazul codificărilor *UTF-16*, există trei codificări *UTF-32*:

- *UTF-32LE*: Primul octet este cel mai puțin semnificativ (*little endian*);
- *UTF-32BE*: Primul octet este cel mai semnificativ (*big endian*);
- *UTF-32*: Ordinea octeților poate fi fie *big endian*, fie *little endian*, la alegerea emițătorului. Primul caracter codificat trebuie să fie caracterul cu codul $FEFF_{16}$. Ordinea octeților este dedusă de receptor prin examinarea primilor patru octeți: dacă aceștia sunt 0, 0, FE_{16} , FF_{16} , înseamnă că ordinea octeților este *big endian*; dacă este FF_{16} , FE_{16} , 0, 0, înseamnă că ordinea octeților este *little endian*.

7.3. Reprezentarea datei și orei

Determinarea datei și orei producerii unui eveniment, precum și memorarea sau transmiterea acestora, sunt operații frecvente într-o rețea de calculatoare.

Problema reprezentării datei și orei este mult mai dificilă decât pare la prima vedere. Din acest motiv, vom începe prin a studia ce se poate înțelege

prin „ora curentă“, iar apoi vom vedea ce scheme de reprezentare ale datei și orei există și ce avantaje și dezavantaje aduce fiecare dintre ele.

7.3.1. Măsurarea timpului

Există două metode utilizate pentru indicarea timpului curent (datei și orei curente):

- pe baza unor fenomene astronomice, anume alternanța zi-noapte (în termeni astronomici, *ziua solară mijlocie*), alternanța anotimpurilor (*anul tropic*) și, eventual, fazele lunii (*luna sinodică*);
- pe baza unui fenomen fizic repetabil, de exemplu oscilația unui pendul sau vibrația unui cristal de cuarț.

Prima variantă este de interes practic imediat pentru sincronizarea activităților umane. Are însă complicații inerente legate de următoarele fapte:

- alternanța zi-noapte nu este simultană pe tot Pământul ci este decalată pe longitudine;
- anul, luna și ziua sunt incommensurabile (rapoartele duratelor lor sunt numere iraționale);
- anul, luna și ziua nu au durate constante (fenomenele corespunzătoare nu sunt perfect periodice) și nici măcar previzibile exact (în special rotația Pământului are neuniformități imprevizibile datorate redistribuirii masei în interiorul Pământului).

A doua variantă măsoară direct timpul ca mărime fizică și oferă avantaje atunci când avem de determinat ordinea cronologică a unor evenimente sau de calculat duratele de timp dintre ele. Timpul (fizic) a ajuns să poată fi definit independentă de mișcarea Pământului doar după dezvoltarea, începând cu anii 1950, a ceasurilor atomice, mai precise decât mișcările Pământului. Măsurarea timpului se face pe baza *secunde* definite în Sistemul Internațional de unități (SI) ca *9192631770 de perioade ale oscilației corespunzătoare tranziției între cele două nivele hiperfine ale stării fundamentale a atomului de cesiu 133*.

Ca urmare a acestor complicații, există mai multe standarde de măsurare și reprezentare a timpului:

Timpul atomic internațional (TAI) este dat de numărul de secunde SI scurse de la un anumit moment ales ca reper. Secundele TAI se grupează în minute, ore, zile, etc.

Timpul universal UT1 este de fapt măsura unui anumit unghi, legat de rotația Pământului, exprimată în unități de timp (24 h în loc de 360°). (Unghiul respectiv este unghiul orar, pentru un observator aflat

pe meridianul 0° , al soarelui mijlociu.) Curge neuniform datorită neuniformității mișcării de rotație a Pământului; după media ultimilor câțiva ani, 24 h UT1 este aproximativ 86400,002 s SI.

Timpul universal coordonat (UTC) este bazat pe secunda SI, dar gruparea secundelor în zile este modificată pentru a menține diferența dintre UT1 și UTC la sub o secundă.

Astfel, o zi UTC normală are 24 ore a 60 minute a 60 secunde SI fiecare, adică 86400 s. Dacă UT1–UTC se apropie de -1 s, se adaugă o secundă de corecție (engl. *leap second*) la o zi, astfel încât acea zi UTC are 86401 s, proces aproape echivalent cu a muta UTC cu o secundă înapoi. În acest scop, ultimul minut al zilei are 61 de secunde în loc de 60, după ora 23:59:59 urmează, la o secundă, 23:59:60 și abia după încă o secundă ora 0:00:00 a zilei următoare. Dacă UT1–UTC se apropie de 1 s, se elimină o secundă din ultimul minut al unei zile, astfel că la o secundă după 23:59:58 urmează ora 0:00:00 a zilei următoare. Din anul 1972 (de la introducerea UTC în forma actuală) până în 2008 au fost adăugate 23 de secunde de corecție și nu a fost eliminată nici una. A 24-a secundă de corecție se va adăuga la sfârșitul anului 2008, astfel încât ziua de 31 decembrie 2008 va avea 86401 secunde. Datorită unei diferențe inițiale de 10 s între TAI și UTC, diferența TAI–UTC este în prezent de 33 s.

Timpul legal în fiecare țară este definit fie pe baza UT1, fie pe baza UTC (diferența este neglijabilă pentru uzul practic), ca fiind UTC (sau UT1) plus sau minus un anumit număr de ore și uneori și fracțiuni de oră (exemplu, India are ora legală UTC+5h30min).

În țările în care există oră de vară, la trecerea de la ora de iarnă la cea de vară și invers, diferența dintre ora legală și UTC crește, respectiv scade, cu o oră (de notat că UTC nu are oră de vară). De exemplu, ora legală în România este UTC+2 h în timpul iernii (din ultima duminică din octombrie până în ultima duminică din martie) și UTC+3 h în timpul verii.

Ora suplimentară introdusă la trecerea de la ora de vară la cea de iarnă nu are notație distinctă, de tipul secundelor de corecție din UTC. Ca urmare, la trecerea de la ora de vară la ora de iarnă, există perechi de momente de timp care sunt notate la fel și ca urmare ora legală este ambiguă. Exemplu: dacă prin trecerea de la ora de vară la cea de iarnă ora 4:00:00 devine 3:00:00, atunci notația 3:30:00 poate corespunde la două momente de timp, ora de vară 3:30:00 (la 30 min înaintea schimbării orei) și ora de iarnă 3:30:00 (la 30 min după schimbarea orei).

Pentru gruparea zilelor în unități mai mari, în special în ani, sunt utilizate mai multe sisteme (calendare):

Calendarul gregorian, introdus în anul 1582 și în vigoare în România din anul 1924, este calendarul actual. Anii bisecți (de 366 de zile) sunt anii cu numărul anului divizibil cu 4, cu excepția celor divizibili cu 100 fără a fi divizibili cu 400. Ani bisecți sunt 1600, 2000, 2400 etc; ani nebisecți divizibili cu 100 sunt 1700, 1800, 1900, 2100, 2200 etc. Durata medie a anului gregorian este 365,2425 zile, ceva mai lung decât anul tropic de aproximativ 265,2422 zile.

Calendarul iulian, predecesorul calendarului gregorian, introdus în anul 45 î.e.n. și având regula mai simplă cum că sunt bisecți toți anii cu număr divizibil cu 4. Este utilizat adesea de istorici pentru a data și evenimente dinainte de anul 45 î.e.n., caz în care el este numit *calendar iulian proleptic*. Cu o durată medie a anului de 365,25 zile, calendarul iulian rămâne în urmă cu 1 zi la aproximativ 128 de ani.

Ziua iuliană este un simplu număr ce arată numărul de zile scurse de la o dată de referință. Acest sistem este utilizat frecvent în astronomie deoarece permite ușor calculul duratelor dintre două date; din același motiv reprezintă o schemă potrivită pentru reprezentarea datei în calculator. Există în două variante. Prima, JD (*julian day*), are ca referința data de 1 ianuarie 4713 î.e.n. conform calendarului iulian proleptic, la amiaza UT1. Momentul respectiv este JD 0,0, miezul nopții următoare este JD 0,5, etc. Cealaltă, MJD (*modified julian day*), are ca referință 17 noiembrie 1858 ora 0, adică este JD-2400000,5.

7.3.2. Obiectivele în alegerea reprezentării timpului în calculator

De obicei, operațiile ce trebuiesc efectuate asupra reprezentării timpului sunt:

1. Citirea sau scrierea timpului ca oră legală conform calendarului gregorian în formatul obișnuit, precum și efectuarea de operații aritmetice de genul adunării sau scăderii unei durate formale (exemplu: mâine la aceeași oră; aceasta înseamnă în mod obișnuit peste 24 de ore, dar poate însemna peste 23 sau 25 de ore dacă intervine trecerea de la ora de iarnă la cea de vară sau invers).
2. determinarea ordinii cronologice a două momente de timp;
3. determinarea exactă, ca timp fizic, a duratei între două momente de timp,

4. pentru aplicații speciale, citirea sau afișarea timpului în alte formate (timpul legal al altui fus orar, UTC, TAI, JD, etc).

Punctul 1 este cerut de toate sistemele. Punctul 2 este important pentru foarte multe aplicații și rezolvarea lui corectă interzice mutarea ceasului înapoi. Punctul 3 este important în aplicațiile în timp real; de asemenea, funcționarea ceasului sistem presupune, în mod repetat, adunarea unei durate de timp la un moment de timp.

Reprezentarea directă a orei legale, sub forma an, lună, zi, oră, minut, secundă, fracțiuni de secundă, rezolvă simplu punctul 1. Ea ridică însă probleme la punctul 2 dacă sunt implicate calculatoare aflate pe fusuri orare distincte sau dacă se efectuează operații în intervalul de o oră în jurul trecerii de la ora de vară la cea de iarnă; pentru tratarea corectă a acestor cazuri este necesar să se știe, despre fiecare oră, pe ce fus orar este considerată și care sunt regulile privind ora de vară. Punctul 3 ridică, pe lângă problemele comune cu cele legate de punctul 2, complicații legate de saltul cu o oră înainte la trecerea de la ora de iarnă la ora de vară și calculele legate de calendar; de asemenea, pentru calcule exacte ale duratelor, sunt necesare informații cu privire la secunde de corecție.

Reprezentarea orei UTC permite determinarea ordinii cronologice și a duratelor fără a necesita date despre fusurile orare sau regulile privind ora de vară, în schimb aceste date sunt necesare la conversia între reprezentarea UTC și timpul legal.

Reprezentarea TAI ca număr de unități de timp scurse de la un anumit moment fixat rezolvă extrem de simplu punctele 2 și 3 în schimb mută dificultățile la rezolvarea punctului 1.

7.3.3. Formate utilizate în practică

Deoarece într-o rețea pot fi prezente calculatoare situate pe fusuri orare distincte, aproape orice format util în rețea fie transmite direct ora UTC sau TAI, fie transmite suficientă informație pentru ca receptorul să poată calcula ușor ora UTC.

7.3.3.1. Formatul utilizat de poșta electronică

Pentru poșta electronică (§ 11.1), reprezentarea datei se face ca text și conține, în ordine:

- opțional ziua din săptămână, ca prescurtare de 3 litere din limba engleză),
- ziua, ca număr între 1 și 31,
- luna, ca șir de trei litere, prescurtare din engleză,

- anul, ca şir de 4 cifre,
- ora, totdeauna ca 2 cifre, între 00 şi 23,
- minutul, ca două cifre, între 00 şi 59,
- opţional, secunda, ca două cifre între 00 şi 60,
- diferenţa dintre ora legală conform căreia a fost scrisă data şi ora UTC; aceasta este dată ca 4 cifre, 2 pentru numărul de ore şi 2 pentru numărul de minute, cele patru cifre fiind precedate de semnul + sau -. Componentele datei sunt separate printr-un amestec de virgule, spaţii şi caractere *două puncte*.

De exemplu, data:

Thu, 25 Oct 2007, 17:22:19 +0300

înseamnă că la momentul scrierii mesajului ora locală a expeditorului era joi, 25 octombrie 2007, ora 17:22:19 şi că ora respectivă este cu 3 ore în avans faţă de UTC. Ca urmare, ora UTC la acel moment era 14:22:19.

Data considerată în acest exemplu este plauzibilă conform orei legale a României, în 25 octombrie 2007 fiind încă în vigoare ora de vară care este cu 3 ore în avans faţă de UTC.

Orele astfel specificate sunt uşor de comparat şi nu există ambiguităţi legate de trecerea de la ora de vară la cea de iarnă. De exemplu, un mesaj trimis înainte de trecerea la ora de iarnă ar fi datat

Sun, 28 Oct 2007, 03:40 +0300

urmat la jumătate de oră, după trecerea la ora de iarnă, de

Sun, 28 Oct 2007, 03:10 +0200

7.3.3.2. ISO-8601 şi RFC-3339

ISO-8601 este o standardizare a modului de scriere ca text a datei şi orei. Standardul fiind foarte complex, a apărut RFC-3339 care cuprinde cazurile mai utile şi mai frecvent folosite din ISO-8601.

RFC-3339 prevede reprezentarea datelor astfel:

- anul, ca patru cifre (nu sunt admise prescurtări de genul 07 pentru 2007),
- luna, ca 2 cifre (01 pentru ianuarie, 12 pentru decembrie),
- ziua, ca 2 cifre (de la 01 până la 31 sau mai puţin, în funcţie de lună).

Cele trei componente sunt separate prin liniuţe (ISO-8601 permite şi alipirea lor):

2007-10-28

Ora se reprezintă prin 2 cifre pentru oră (00–23), 2 cifre pentru minut (00–59), două cifre pentru secundă (00–60, în funcție și de prezența unei secunde de corecție), eventual fracțiunile de secundă și eventual specificarea fusului orar. Ora, minutul și secunda sunt separate prin *două puncte*, fracțiunile de secundă se separă de câmpul pentru secunde prin *punct*, iar specificarea fusului orar se face printr-un semn plus sau minus urmat de două cifre pentru numărul de ore diferență urmat de caracterul *două puncte* și încă două cifre pentru numărul de minute diferență. Exemplu:

21:12:58.342+02:00

reprezintă același moment de timp, dar pe alt fus orar, cu

14:12:58.342-05:00

Data și ora se specifică împreună punând litera T între ele:

2007-10-28T14:12:58.342-05:00

7.3.3.3. Timpul POSIX

În sistemele de tip UNIX (conforme standardului POSIX), reprezentarea timpului este făcută printr-un număr întreg considerat de obicei că reprezintă numărul de secunde scurse de la 1 ianuarie 1970 ora 0:00 UTC. Ora UTC în formatul obișnuit se obține grupând secunde în minute, ore, zile, luni și ani conform regulilor obișnuite. De fapt, numărul dat ca dată nu este exact numărul de secunde scurse de la 1 ianuarie 1970, ci diferă de acesta prin numărul de secunde de corecție adăugate pentru menținerea în sincronism a UTC cu rotația Pământului. De aceea, „timpul unix“ are salturi înapoi de câte o secundă la fiecare introducere a unei astfel de secunde de corecție. Timpul POSIX este reprezentat în mod obișnuit ca întreg cu semn pe 32 de biți, și ca urmare valoarea cea mai mare ce poate fi reprezentată corespunde datei de 19 ianuarie 2038, ora 3:14:07 UTC.

7.3.3.4. TAI 64

TAI 64 este un standard ce presupune reprezentarea timpului ca număr de secunde, incluzând secunde de corecție. Numărul de secunde este reprezentat pe 63 de biți (plus un bit rezervat), cu valoarea 2^{62} corespunzând datei de 1 ianuarie 1970 ora 0 TAI. Intervalul de timp reprezentabil este imens, de ordinul a 10^{11} ani.

Pentru aplicații ce au nevoie de rezoluție mai bună de o secundă, standardul prevede încă două câmpuri de câte 32 de biți (total 128 de biți), reprezentând respectiv numărul de nanosecunde și de attosecunde (valori între 0 și $10^9 - 1$).

7.4. Recodificări

Este necesar uneori să codificăm un șir mai mult sau mai puțin arbitrar de octeți sub forma unui șir de caractere supus unor restricții. Astfel de situații apar:

- La trimiterea fișierelor atașate la mesajele de poștă electronică, întregul mesaj, inclusiv partea ce cuprinde fișierele atașate, trebuie să îndeplinească anumite restricții, între altele, să nu conțină caractere ASCII de control cu excepția perechilor *carriage return–line feed* de la finalul fiecărui rând, să nu aibă rânduri prea lungi, etc. Pe de altă parte, fișierele atașate pot fi fișiere binare cu conținut arbitrar.
- La stocarea în fișiere text a unor informații reprezentate natural ca șir arbitrar de octeți, de exemplu la stocarea în fișiere text a unor chei de criptare, semnături electronice, etc.
- În limbaje de programare, la scrierea în șirurile de caractere a unor caractere cu rol special, ca de exemplu a ghilimelelor (care în mod normal sunt interpretate ca terminatorul șirului de caractere).

În astfel de situații, este necesar să se codifice un șir arbitrar de octeți (fiecare octet poate lua orice valori între 0 și 255) pe un alfabet constând în litere, cifre și câteva simboluri speciale. Deoarece numărul de simboluri disponibile este mai mic decât numărul de valori posibile ale octeților, un octet nu se va putea codifica numai pe un singur caracter.

7.4.1. Codificarea hexazecimală

Codificarea hexazecimală prevede ca fiecare octet să se reprezinte pe două caractere, fiecare caracter putând fi din mulțimea cuprinzând cifrele zecimale (0–9) și literele A–F.

Exemplu: șirul de octeți 120, 0, 23, 45, 20 se scrie: 7800172D14.

Uneori, în șirul de cifre hexa se inserează spații sau caractere *newline* pentru lizibilitate sau pentru evitarea rândurilor foarte lungi.

Deoarece un caracter se reprezintă de obicei pe un octet, prin această recodificare rezultă o dublare a lungimii șirului.

7.4.2. Codificarea în baza 64

Codificarea *în baza 64* codifică un șir de 3 octeți cu valori arbitrare ca un șir de 4 caractere din mulțimea cuprinzând literele mari, literele mici, cifrele și caracterele +, / și =.

Codificarea se face în modul următor:

1. Șirul inițial de octeți se completează la un multiplu de 3 octeți prin adăugarea a 0, 1 sau 2 octeți cu valoarea 0.
2. Se formează un șir de 24 de biți punând unul după altul cei câte 8 biți din fiecare octet; din fiecare octet se începe cu bitul cel mai semnificativ.
3. Șirul de 24 de biți se împarte în 4 grupuri de câte 6 biți.
4. Fiecare șir de 6 biți se consideră ca fiind un număr cuprins între 0 și 63, considerând primul bit din șir ca fiind cel mai semnificativ.
5. Fiecare număr obținut la pasul anterior se reprezintă printr-un caracter. Cele 64 de valori posibile, de la 0 la 63, se reprezintă ca: litere mari (0→A, 25→Z), litere mici (26→a, 51→z), cifre (52→0, 61→9) și caracterele + și / (62→+, 63→/). Dacă o valoare 0 provine din biți 0 proveniți integral dintr-un octet completat la pasul 1, în loc de A se scrie =.

De exemplu, șirul de octeți 120, 0, 23, 45, 20 devine șirul de biți

```
01111000 00000000 00010111 00101101 00010100 00000000
```

ultimul octet fiind adăugat la pasul 1. Șirul de biți se regrupează

```
011110 000000 000000 010111 001011 010001 010000 000000
```

rezultând șirul de numere „în baza 64”: 30, 0, 0, 23, 11, 17, 16, 0, care se codifică eAAXLRQ=

7.4.3. Codificări bazate pe secvențe de evitare

Recodificările prin secvențe de evitare sunt utilizate în situația în care majoritatea octeților sau caracterelor din textul de recodificat sunt codurile unor caractere ce se regăsesc în alfabetul în care se face recodificarea. În această situație, este favorabil ca, pe cât posibil, octeții sau caracterele din textul de recodificat să fie codificate prin ele însele, în special pentru ca textul să poată fi înțeles (parțial, cel puțin) de către un utilizator uman direct în forma recodificată.

Recodificarea se face astfel. Se distinge un caracter în alfabetul destinație, caracter ce este denumit *caracter de evitare* (enlg. *escape character*).

- Orice caracter din alfabetul sursă care se regăsește în alfabetul destinație și este diferit de caracterul de evitare se recodifică ca el însuși.

- Caracterul de evitare (dacă face parte din alfabetul sursă), precum și caracterele din alfabetul sursă ce nu se găsesc în alfabetul destinație, se codifică ca secvențe de caractere ce încep cu un caracter de evitare.

Exemple:

- Pentru poșta electronică, un caracter ce nu pot fi transmise direct este codificat ca o secvență de trei caractere, un caracter *egal* (=) și două cifre hexa reprezentând codul caracterului de transmis. Această recodificare se numește *quoted printables*. Caracterele ASCII imprimabile, cu excepția caracterului *egal*, se transmit direct (fără recodificare). Ca urmare, un text în care caracterele ce trebuie recodificate sunt rare poate fi înțeles de către un utilizator uman fără prea mari dificultăți. Ca exemplu, fraza precedentă se scrie (presupunând o codificare ISO-8859-2 pentru caractere și apoi o recodificare *quoted printables*):

Ca urmare, un text =EEn care caracterele ce trebuie
recodificate sunt rare poate fi =EEn=FEeles de c=E3tre
un utilizator uman f=E3r=E3 prea mari difficult=E3=FEi.

- În *URL*-uri, caracterele ce au rol sintactic (*spațiu*, /, etc) se codifică prin caracterul *procent* (%) urmat de două cifre hexa reprezentând valoarea caracterului respectiv. De notat că aceste coduri sunt în cadrul codificării UTF-8; ca urmare, o pagină cu numele *șir* ar avea un *URL* de forma

`http://example.com/%C8%98ir`

- În limbajul C și în alte limbaje derivate din el, ghilimelele (") servesc la delimitarea șirurilor de caractere. Dacă se dorește introducerea unor astfel de caractere într-un șir, sau a caracterelor ASCII de control, se introduc secvențe *escape* cum ar fi: \" pentru ghilimele ("), \\ pentru *backslash* (\), \n pentru *newline* (caracterul ASCII cu codul 10).
- În HTML, caracterele *unicode* sunt scrise prin secvențe *&#cod;*. De exemplu, litera ț se scrie *ț*.

Capitolul 8

Programarea în rețea — introducere

8.1. Interfața de programare *socket BSD*

Interfața *socket* este un ansamblu de funcții sistem utilizate de programe (de fapt, de procese) pentru a comunica cu alte procese, aflate în execuție pe alte calculatoare. Interfața *socket* a fost dezvoltată în cadrul sistemului de operare BSD (sistem de tip UNIX, dezvoltat la Universitatea Berkley) — de aici denumirea de *socket BSD*. Interfața *socket* este disponibilă în aproape toate sistemele de operare actuale.

Termenul *socket* se utilizează atât pentru a numi ansamblul funcțiilor sistem legate de comunicația prin rețea, cât și pentru a desemna fiecare capăt al unei conexiuni deschise în cadrul rețelei.

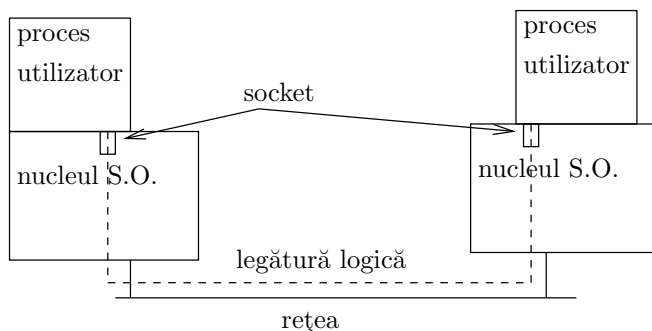


Figura 8.1: Comunicația între două procese prin rețea

Prezentăm în continuare principiile de bază ale interfeței *socket* (vezi

și figura 8.1):

- Pe fiecare calculator rulează mai multe procese și fiecare proces poate avea mai multe căi de comunicație deschise. Prin urmare, pe un calculator trebuie să poată exista la un moment dat mai multe legături (conexiuni) active.
- Realizarea comunicării este intermediată de sistemele de operare de pe calculatoarele pe care rulează cele două procese. Deschiderea unei conexiuni, închiderea ei, transmiterea sau recepționarea de date pe o conexiune și configurarea parametrilor unei conexiuni se fac de către sistemul de operare, la cererea procesului. Cererile procesului se fac prin apelarea funcțiilor sistem din familia *socket*.
- În cadrul comunicației dintre procesul utilizator și sistemul de operare local (prin intermediul apelurilor din familia *socket*), capetele locale ale conexiunilor deschise sunt numite *socket*-uri și sunt identificate prin numere întregi, unice în cadrul unui proces la fiecare moment de timp.
- Fiecare entitate care comunică în cadrul rețelei este identificat printr-o *adresă* unică. O adresă este asociată de fapt unui socket. Adresa este formată conform regulilor protocolului de rețea utilizat.
- Interfața socket conține funcții pentru comunicației atât conform modelului conexiune cât și conform modelului cu datagrame.
- Funcțiile sistem oferite permit stabilirea comunicației prin diferite protocoale (de exemplu, IPv4, IPv6, IPX), dar au aceeași sintaxă de apel independent de protocolul dorit.

8.1.1. Comunicația prin conexiuni

În cele ce urmează, prin *client* desemnăm procesul care solicită în mod activ deschiderea conexiunii către un partener de comunicație specificat printr-o *adresă*, iar prin *server* înțelegem procesul care așteaptă în mod pasiv conectarea unui *client*.

Vom da în cele ce urmează o scurtă descriere a operațiilor pe care trebuie să le efectueze un proces pentru a deschide o conexiune și a comunica prin ea. Descrierea este împărțită în patru părți: deschiderea conexiunii de către client, deschiderea conexiunii de către server, comunicația propriu-zisă și închiderea conexiunii.

O descriere mai amănunțită a funcțiilor sistem apelate și a parametrilor mai des utilizați este făcută separat (§ 8.1.3), iar pentru detalii suplimentare se recomandă citirea paginilor corespunzătoare din documentația on-line.

8.1.1.1. Deschiderea conexiunii de către client

Procesul client trebuie să ceară mai întâi sistemului de operare local crearea unui socket. Trebuie specificat protocolul de rețea utilizat (TCP/IPv4, TCP/IPv6, etc), dar încă nu se specifică partenerul de comunicație. Socket-ul proaspăt creat este în starea *neconectat*.

După crearea socket-ului, clientul cere sistemului de operare conectarea socket-ului la un anumit server, specificat prin adresa *socket*-ului serverului. De exemplu, pentru protocolul TCP/IPv4, adresa partenerului se specifică prin adresa IP (vezi § 10.1) și numărul portului (vezi § 10.3.1).

Funcțiile sistem apelate sunt: `socket()` pentru crearea socket-ului și `connect()` pentru deschiderea efectivă a conexiunii.

8.1.1.2. Deschiderea conexiunii de către server

Procesul server începe tot prin a cere sistemului de operare crearea unui socket de tip conexiune pentru protocolul dorit. Acest socket nu va servi pentru conexiunea propriu-zisă cu un client, ci doar pentru așteptarea conectării clienților; ca urmare este numit uneori *socket de așteptare*. După crearea acestui socket, serverul trebuie să ceară sistemului de operare stabilirea adresei la care serverul așteaptă cereri de conectare (desigur, acea parte din adresă care identifică mașina serverului nu este la alegerea procesului server) și apoi cere efectiv începerea așteptării clienților. Funcțiile apelate în această fază sunt, în ordinea în care trebuie apelate: `socket()` pentru crearea socket-ului, `bind()` pentru stabilirea adresei și `listen()` pentru începerea așteptării clienților.

Preluarea efectivă a unui client conectat se face prin apelarea unei funcții sistem numită `accept()`. La apelul funcției `accept()`, sistemul de operare execută următoarele:

- așteaptă cererea de conectare a unui client și deschide conexiunea către acesta;
- crează un nou socket, numit *socket de conexiune*, care reprezintă capătul dinspre server al conexiunii proaspăt deschise;
- returnează apelantului (procesului server) identificatorul socket-ului de conexiune creat.

După un apel `accept()`, socket-ul de așteptare poate fi utilizat pentru a aștepta noi clienți, iar socket-ul de conexiune nou creat se utilizează pentru a comunica efectiv cu acel client.

8.1.1.3. Comunicația propriu-zisă

O dată deschisă conexiunea, clientul poate trimite șiruri de octeți către server și invers, serverul poate trimite șiruri de octeți către client. Cele două sensuri de comunicație funcționează identic (nu se mai distinge cine a fost client și cine a fost server) și complet independent (trimiterea datelor pe un sens nu este condiționată de recepționarea datelor pe celălalt sens).

Pe fiecare sens al conexiunii, se poate transmite un șir arbitrar de octeți. Octeții trimiși de către unul dintre procese spre celălalt sunt plasați într-o coadă, transferați prin rețea la celălalt capăt și citați de către procesul de acolo. Comportamentul acesta este similar cu cel al unui *pipe* UNIX.

Trimiterea datelor se face prin apelul funcției `send()` (sau, cu funcționalitate mai redusă, `write()`). Apelul acestor funcții plasează datele în coadă spre a fi transmise, dar nu așteaptă transmiterea lor efectivă (returnează, de principiu, imediat controlul către procesul apelant). Dacă dimensiunea datelor din coadă este mai mare decât o anumită valoare prag, (aleasă de sistemele de operare de pe cele două mașini), apelul `send()` se blochează, returnând controlul procesului apelant abia după ce partenerul de comunicație citește date din coadă, ducând la scăderea dimensiunii datelor din coadă sub valoarea prag.

Recepționarea datelor trimise de către partenerul de comunicație se face prin intermediul apelului sistem `recv()` (cu funcționalitate mai redusă se poate utiliza `read()`). Aceste funcții returnează procesului apelant datele deja sosite pe calculatorul receptor și le elimină din coadă. În cazul în care nu sunt încă date disponibile, ele așteaptă sosirea a cel puțin un octet.

Sistemul garantează sosirea la destinație a tuturor octeților trimiși (sau înștiințarea receptorului, printr-un cod de eroare, asupra căderii conexiunii), în ordinea în care au fost trimiși. Nu se păstrează însă demarcarea între secvențele de octeți trimise în apeluri `send()` distincte. De exemplu, este posibil ca emițătorul să trimită, în două apeluri succesive, șirurile `abc` și `def`, iar receptorul să primească, în apeluri `recv()` succesive, șirurile `ab`, `cde` și `f`.

8.1.1.4. Închiderea conexiunii

Închiderea conexiunii se face separat pentru fiecare sens și pentru fiecare capăt. Există două funcții:

- `shutdown()` închide, la capătul local al conexiunii, sensul de comunicație cerut de procesul apelant;
- `close()` închide la capătul local ambele sensuri de comunicație și în plus distruge socket-ul, eliberând resursele alocate (identificatorul de socket

și memoria alocată în spațiul nucleului).

Terminarea unui proces are efect identic cu un apel `close()` pentru toate socket-urile existente în acel moment în posesia aceluși proces.

Dacă capătul de emisie al unui sens de comunicație a fost închis, receptorul poate citi în continuare datele existente în acel moment în coadă, după care un eventual apel `recv()` va semnaliza apelantului faptul că a fost închisă conexiunea.

Dacă capătul de recepție al unui sens a fost închis, o scriere ulterioară de la celălalt capăt este posibil să returneze un cod de eroare (pe sistemele de tip UNIX, scrierea poate duce și la primirea, de către procesul emițător, a unui semnal SIGPIPE).

8.1.2. Comunicația prin datagrame

În comunicația prin datagrame, datagramele sunt transmise independent una de cealaltă și fiecare datagramă are o adresă sursă, o adresă destinație și niște date utile. Un proces ce dorește să trimită sau să primească datagrame trebuie mai întâi să creeze un *socket* de tip *dgram*; un astfel de socket conține în principal adresa de rețea a procesului posesor al socket-ului. După crearea unui socket, se poate cere sistemului de operare să asocieze socket-ului o anumită adresă sau se poate lăsa ca sistemul de operare să-i atribuie o adresă liberă arbitrară. Crearea unui socket se face prin apelul funcției `socket()`, iar atribuirea unei adrese se face prin apelul `bind()`.

O dată creat un socket, procesul poate trimite datagrame de pe acel socket, prin apelul funcției `sendto()`. Datagramele trimise vor avea ca adresă sursă adresa socket-ului și ca adresă destinație și conținut util valorile date ca parametri funcției `sendto()`. De pe un socket se pot trimite, succesiv, oricâte datagrame și oricâtor destinatari.

Datagramele emise sunt transmise către sistemul de operare al destinatarului, unde sunt memorate în buffer-ele sistemului. Destinatarul poate citi o datagramă apelând funcția `recvfrom()`. Această funcție ia următoarea datagramă adresată socket-ului dat ca parametru la `recvfrom()` și o transferă din buffer-ele sistemului local în memoria procesului apelant. Funcția oferă apelantului conținutul datagramei (datele utile) și, separat, adresa expeditorului datagramei. În ciuda numelui, `recvfrom()` nu poate fi instruită să ia în considerare doar datagramele expediate de la o anumită adresă.

Sistemul nu garantează livrarea tuturor datagramelor (este posibilă pierderea unor datagrame) și nici nu oferă vreun mecanism de informare a expeditorului în cazul unei pierderi. Mai mult, există posibilitatea (e drept, rară) ca o datagramă să fie duplicată (să ajungă două copii la destinatar) și

este posibil ca două sau mai multe datagrame adresate aceluiași destinatar să ajungă la destinație în altă ordine decât cea în care au fost emise. Dacă astfel de situații sunt inadmisibile pentru aplicație, atunci protocolul de comunicație trebuie să prevadă confirmări de primire și repetarea datagramelor pierdute, precum și numere de secvență sau alte informații pentru identificarea ordinii corecte a datagramelor și a duplicatelor. Implementarea acestor mecanisme cade în sarcina proceselor.

La terminarea utilizării unui socket, procesul posesor poate cere distrugerea socket-ului și eliberarea resurselor asociate (identificatorul de socket, memoria ocupată în sistemul de operare pentru datele asociate socket-ului, portul asociat socket-ului). Distrugerea socket-ului se face prin apelul funcției `close()`.

În mod curent, într-o comunicație prin datagrame, unul dintre procese are rol de *client*, în sensul că trimite cereri, iar celălalt acționează ca *server*, în sensul că prelucrează cererile clientului și trimite înapoi clientului răspunsurile la cereri. Într-un astfel de scenariu, serverul crează un socket căruia îi asociază o adresă prestabilită, după care așteaptă cereri, apelând în mod repetat `recvfrom()`. Clientul crează un socket, căruia nu-i asociază o adresă (nu execută `bind()`). Clientul trimite apoi cererea sub forma unei datagrame de pe socket-ul creat. La trimiterea primei datagrame, sistemul de operare dă o adresă socket-ului; datagrama emisă poartă ca adresă sursă această adresă. La primirea unei datagrame, serverul recuperează datele utile și adresa sursă, procesează cererea și trimite răspunsul către adresa sursă a cererii. În acest fel, răspunsul este adresat exact socket-ului de pe care clientul a trimis cererea. Clientul obține răspunsul executând `recvfrom()` asupra socket-ului de pe care a expediat cererea.

Cu privire la tratarea datagramelor pierdute, un caz simplu este acela în care clientul pune doar întrebări (interogări) serverului, iar procesarea interogării nu modifică în nici un fel starea serverului. Un exemplu tipic în acest sens este protocolul DNS (§ 10.4). În acest caz, datagrama cerere conține interogarea și datagrama răspuns conține atât cererea la care se răspunde cât și răspunsul la interogare. Serverul ia (în mod repetat) câte o cerere, calculează răspunsul și trimite o înapoi o datagramă cu cererea primită și răspunsul la cerere. Clientul trimite cererile sale și așteaptă răspunsurile. Deoarece fiecare răspuns conține în el și cererea, clientul poate identifica fiecare răspuns la ce cerere îi corespunde, chiar și în cazul inversării ordinii datagramelor. Dacă la o cerere nu primește răspuns într-un anumit interval de timp, clientul repetă cererea; deoarece procesarea unei cereri nu modifică starea serverului, duplicarea cererii de către rețea sau repetarea cererii de către client ca urmare a pierderii

răspunsului nu au efecte nocive. Clientul trebuie să ignore răspunsurile duplicate la o aceeași interogare.

8.1.3. Principalele apeluri sistem

8.1.3.1. Funcția `socket()`

Funcția are sintaxa:

```
int socket(int proto_family, int type, int protocol)
```

Funcția creează un socket și returnează identificatorul său. Parametrii sunt:

- **type**: desemnează tipul de servicii dorite:
 - SOCK_STREAM**:conexiune punct la punct, flux de date bidirecțional la nivel de octet, asigurând livrare sigura, cu pastrarea ordinii octeților și transmisie fara erori.
 - SOCK_DGRAM**:datagrama, atât punct la punct cât și difuziune; transmisia este garantată a fi fără erori, dar livrarea nu este sigura și nici ordinea datagramelor garantata.
 - SOCK_RAW**:acces la protocoale de nivel coborât; este de exemplu utilizat de către comanda `ping` pentru comunicație prin protocolul ICMP.
- **proto_family** identifică tipul de rețea cu care se lucrează (IP, IPX, etc). Valori posibile:
 - PF_INET**:protocol Internet, versiunea 4 (IPv4)
 - PF_INET6**:protocol Internet, versiunea 6 (IPv6)
 - PF_UNIX**:comunicație locală pe o mașină UNIX.
- **protocol** selectează protocolul particular de utilizat. Acest parametru este util dacă pentru un tip de rețea dat și pentru un tip de serviciu fixat există mai multe protocoale utilizabile. Valoarea 0 desemnează protocolul implicit pentru tipul de rețea și tipul de serviciu alese.

8.1.3.2. Funcția `connect()`

Funcția are sintaxa:

```
int connect(int sock_id, struct sockaddr* addr, int addr_len)
```

Funcția are ca efect conectarea socketului identificat de primul parametru — care trebuie să fie un socket de tip conexiune proaspăt creat (încă neconectat)

— la serverul identificat prin adresă dată prin parametrii `addr` și `addr_len`. La adresa respectivă trebuie să existe deja un server care să aștepte conexiuni (să fi fost deja executat apelul `listen()` asupra socket-ului serverului).

Adresa trebuie plasată, înainte de apelul `connect()`, într-o structură având un anumit format; conținutul acestei structuri va fi descris în § 8.1.3.6. Adresa în memorie a acestei structuri trebuie dată ca parametrul `addr`, iar lungimea structurii de adresă trebuie dată ca parametrul `addr_len`. Motivul acestei complicații este legat de faptul că funcția `connect()` trebuie să poată lucra cu formate diferite de adresă, pentru diferite protocoale, iar unele protocoale au adrese de lungime variabilă.

Funcția `connect()` returnează 0 în caz de succes și `-1` în caz de eroare. Eroarea survenită poate fi constatată fie verificând valoarea variabilei globale `errno`, fie apelând funcția `perror()` imediat după funcția sistem ce a întâmpinat probleme. Eroarea cea mai frecventă este lipsa unui server care să asculte la adresa specificată.

8.1.3.3. Funcția `bind()`

```
int bind(int sd, struct sockaddr* addr, socklen_t len)
```

Funcția are ca efect atribuirea adresei specificate în parametrul `addr` socket-ului identificat prin identificatorul `sd`. Această funcție se apelează în mod normal dintr-un proces server, pentru a pregăti un socket *stream* de așteptare sau un socket *dgram* pe care se așteaptă cereri de la clienți.

Partea, din adresa de atribuit socket-ului, ce conține adresa mașinii poate fi fie una dintre adresele mașinii locale, fie valoarea specială `INADDR_ANY` (pentru IPv4) sau `IN6ADDR_ANY_INIT` (pentru IPv6). În primul caz, socket-ul va primi doar cereri de conexiune (sau, respectiv, pachete) adresate adresei IP date socket-ului, și nu și cele adresate altora dintre adresele mașinii server.

EXEMPLUL 8.1: Să presupunem că mașina server are adresele 193.226.40.130 și 127.0.0.1. Dacă la apelul funcției `bind()` se dă adresa IP 127.0.0.1, atunci socket-ul respectiv va primi doar cereri de conectare destinate adresei IP 127.0.0.1, nu și adresei 193.226.40.130. Dimpotrivă, dacă adresa acordată prin `bind()` este `INADDR_ANY`, atunci socket-ul respectiv va accepta cereri de conectare adresate oricăreia dintre adresele mașinii locale, adică atât adresei 193.226.40.130 cât și adresei 127.0.0.1.

Adresa atribuită prin funcția `bind()` trebuie să fie liberă în acel moment. Dacă în momentul apelului `bind()` există un alt socket de același tip având aceeași adresă, apelul `bind()` eșuează.

Pe sistemele de tip UNIX, pentru atribuirea unui număr de port mai mic decât 1024 este necesar ca procesul apelant să ruleze din cont de administrator.

Funcția `bind()` poate fi apelată doar pentru un socket proaspăt creat, căruia nu i s-a atribuit încă o adresă. Aceasta înseamnă că funcția `bind()` nu poate fi apelată de două ori pentru același socket. De asemenea, funcția `bind()` nu poate fi apelată pentru un socket de conexiune creat prin funcția `accept()` și nici pentru un socket asupra căruia s-a apelat în prealabil vreuna dintre funcțiile `connect()`, `listen()` sau `sendto()` — aceste funcții având ca efect atribuirea unei adrese libere aleatoare.

Funcția returnează 0 în caz de succes și -1 în caz de eroare. Eroarea cea mai frecventă este că adresa dorită este deja ocupată.

8.1.3.4. Funcția `listen()`

```
int listen(int sd, int backlog)
```

Funcția cere sistemului de operare să accepte, din acel moment, cererile de conexiune pe adresa socket-ului `sd`. Dacă socketului respectiv nu i s-a atribuit încă o adresă (printr-un apel `bind()` anterior), funcția `listen()` îi atribuie o adresă aleasă aleator.

Parametrul `backlog` fixează dimensiunea cozii de așteptare în acceptarea conexiunilor. Anume, vor putea exista `backlog` clienți care au executat `connect()` fără ca serverul să fi creat încă pentru ei socket-uri de conexiune prin apeluri `accept()`. De notat că nu există nici o limitare a numărului de clienți conectați, preluați deja prin apelul `accept()`.

8.1.3.5. Funcția `accept()`

```
int accept(int sd, struct sockaddr *addr, socklen_t *addrlen)
```

Apelul funcției `accept()` are ca efect crearea unui socket de conexiune, asociat unui client conectat (prin apelul `connect()`) la socket-ul de așteptare `sd`. Dacă nu există încă nici un client conectat și pentru care să nu se fi creat socket de conexiune, funcția `accept()` așteaptă până la conectarea următorului client.

Funcția returnează identificatorul socket-ului de conexiune creat.

Dacă procesul server nu dorește să afle adresa clientului, va da valori NULL parametrilor `addr` și `addrlen`. Dacă procesul server dorește să afle adresa clientului, atunci va trebui să aloce spațiu pentru o structură pentru memorarea adresei clientului, să pună adresa structurii respective în parametrul

`addr`, să plaseze într-o variabilă de tip întreg dimensiunea memoriei alocate pentru adresa clientului și să pună adresa acestui întreg în parametrul `adrilen`. În acest caz, la revenirea din apelul `accept()`, procesul server va găsi în structura de adresă adresa socket-ului client și în variabila întreagă a cărui adresă a fost dată în parametrul `adrilen` va găsi dimensiunea efectiv utilizată de sistemul de operare pentru a scrie adresa clientului.

8.1.3.6. Formatul adreselor

Pentru funcțiile socket ce primesc de la apelant (ca parametru) o adresă din rețea (`bind()`, `connect()` și `sendto()`), precum și pentru cele ce returnează apelantului adrese de rețea (`accept()`, `recvfrom()`, `getsockname()` și `getpeername()`), sunt definite structuri de date (`struct`) în care se plasează adresele socket-urilor.

Pentru ca funcțiile de mai sus să poată avea aceeași sintaxă de apel independent de tipul de rețea (și, în consecință, de structura adresei), funcțiile primesc adresa printr-un pointer la zona de memorie ce conține adresa de rețea. Structura zonei de memorie respective depinde de tipul rețelei utilizate. În toate cazurile, aceasta începe cu un întreg pe 16 biți reprezentând tipul de rețea.

Dimensiunea structurii de date ce conține adresa de rețea depinde de tipul de rețea și, în plus, pentru anumite tipuri de rețea, dimensiunea este variabilă. Din acest motiv:

- funcțiile care primesc de la apelant o adresă (`connect()`, `bind()` și `sendto()`) au doi parametri: un pointer către structura de adresă și un întreg reprezentând dimensiunea acestei structuri;
- funcțiile care furnizează apelantului o adresă (`accept()`, `recvfrom()`, `getsockname()` și `getpeername()`) primesc doi parametri: un pointer către structura de adresă și un pointer către o variabilă de tip întreg pe care apelantul trebuie s-o inițializeze, înaintea apelului, cu dimensiunea pe care a alocat-o pentru structura de adresă și în care funcția pune, în timpul apelului, dimensiunea utilizată efectiv de structura de adresă.

În ambele cazuri, parametrul pointer către structura de adresă este declarat ca fiind de tip `struct sockaddr*`. La apelul acestor funcții este necesară conversia a pointer-ului către structura de adresă de la pointer-ul specific tipului de rețea la `struct sockaddr*`.

O adresă a unui capăt al unei conexiuni TCP sau a unei legături prin datagrame UDP este formată din adresa IP a mașinii și numărul de port (vezi § 10.2.3.1, § 10.3.1.6 și § 10.3.2). Pentru nevoile funcțiilor de mai sus, adresele socket-urilor TCP și UDP se pun, în funcție de protocolul de nivel

rețea (IPv4 sau IPv6), într-o structură de tip `sockaddr_in` pentru IPv4 sau `sockaddr_in6` pentru IPv6.

Pentru adrese IPv4 este definită structura `sockaddr_in` având următorii membrii:

- `sin_family`:trebuie să conțină constanta `AF_INET`;
- `sin_port`:de tip întreg de 16 biți (2 octeți), fără semn, în ordine rețea (cel mai semnificativ octet este primul), reprezentând numărul portului;
- `sin_addr`:conține adresa IP. Are tipul `struct in_addr`, având un singur câmp, `s_addr`, de tip întreg pe 4 octeți în ordine rețea.

Adresa IPv4 poate fi convertită de la notația obișnuită (notația zecimală cu puncte) la `struct in_addr` cu ajutorul funcției

```
int inet_aton(const char *cp, struct in_addr *inp);
```

Conversia inversă, de la structura `in_addr` la string în notație zecimală cu punct se face cu ajutorul funcției

```
char *inet_ntoa(struct in_addr in);
```

care returnează rezultatul într-un buffer static, apelantul trebuind să copieze rezultatul înainte de un nou apel al funcției.

Pentru adrese IPv6 este definită structura `sockaddr_in6` având următorii membrii:

- `sin6_family`:trebuie să conțină constanta `AF_INET6`;
- `sin6_port`:de tip întreg de 16 biți (2 octeți), fără semn, în ordine rețea (cel mai semnificativ octet este primul), reprezentând numărul portului;
- `sin6_flow`:eticheta de flux.
- `sin6_addr`:conține adresa IP. Are tipul `struct in6_addr`, având un singur câmp, `s6_addr`, de tip tablou de 16 octeți.

Obținerea unei adrese IPv4 sau IPv6 cunoscând numele de domeniu (vezi § 10.4) se face cu ajutorul funcției

```
struct hostent *gethostbyname(const char *name);
```

care returnează un pointer la o structură ce conține mai multe câmpuri dintre care cele mai importante sunt:

- `int h_addrtype`:tipul adresei, `AF_INET` sau `AF_INET6`;

```
char **h_addr_list:pointer la un șir de pointeri către adresele IPv4 sau
    IPv6 ale mașinii cu numele name, în formatul in_addr sau respectiv
    in6_addr;
```

```
int h_length:lungimea șirului h_addr_list.
```

8.1.3.7. Interacțiunea dintre `connect()`, `listen()` și `accept()`

La apelul `connect()`, sistemul de operare de pe mașina client trimite mașinii server o cerere de conectare. La primirea cererii de conectare, sistemul de operare de pe mașina server acționează astfel:

- dacă adresa din cerere nu corespunde unui socket pentru care s-a efectuat deja apelul `listen()`, refuză conectarea;
- dacă adresa corespunde unui socket pentru care s-a efectuat `listen()`, încearcă plasarea clientului într-o coadă de clienți conectați și nepreluți încă prin `accept()`. Dacă plasarea reușește (coada fiind mai mică decât valoarea parametrului `backlog` din apelul `listen()`), sistemul de operare trimite sistemului de operare de pe mașina client un mesaj de acceptare; în caz contrar trimite un mesaj de refuz.

Apelul `connect()` revine în procesul client în momentul sosirii acceptului sau refuzului de la sistemul de operare de pe mașina server. Revenirea din apelul `connect()` nu este deci condiționată de apelul `accept()` al procesului server.

Apelul `accept()` preia un client din coada descrisă mai sus. Dacă coada este vidă în momentul apelului, funcția așteaptă sosirea unui client. Dacă coada nu este vidă, apelul `accept()` returnează imediat.

Parametrul `backlog` al apelului `listen()` se referă la dimensiunea cozii de clienți conectați (prin `connect()`) și încă nepreluți prin `accept()`, și nu la clienții deja preluți prin `accept()`.

8.1.3.8. Funcțiile `getsockname()` și `getpeername()`

```
int getsockname(int sd, struct sockaddr *name, socklen_t *namelen);
int getpeername(int sd, struct sockaddr *name, socklen_t *namelen);
```

Funcția `getsockname()` furnizează apelantului adresa socket-ului `sd`. Funcția `getpeername()`, apelată pentru un socket de tip conexiune deja conectat, furnizează adresa partenerului de comunicație.

Funcția `getsockname()` este utilă dacă un proces acționează ca server, creînd în acest scop un socket de așteptare, dar numărul portul pe care așteaptă conexiunile nu este prestabilit ci este transmis, pe altă cale, viitorilor client. În acest caz, procesul server crează un socket (apelând `socket()`),

cere primirea cererilor de conexiune (apelând `listen()`), dar fără a fi apelat `bind()`) după care determină, prin apelul `getsockname()`, adresa atribuită la `listen()` socket-ului respectiv și transmite această adresă viitorilor clienți.

8.1.3.9. Funcțiile `send()` și `recv()`

Apelurile sistem `send()` și `recv()` sunt utilizate în faza de comunicație pentru socket-uri de tip conexiune. Descriem în continuare utilizarea acestor funcții considerând un singur sens de comunicație și ca urmare ne vom referi la un *proces emițător* și un *proces receptor* în raport cu sensul considerat. De notat însă că o conexiune *socket stream* este bidirecțională și comunicarea în cele două sensuri se desfășoară independent și prin aceleași mecanisme.

Sintaxa funcțiilor este:

```
ssize_t send(int sd, const void *buf, size_t len, int flags);
ssize_t recv(int sd, void *buf, size_t len, int flags);
```

Funcția `send()` trimite pe conexiunea identificată prin socket-ul `sd` un număr de `len` octeți din variabila a cărei adresă este indicată de pointer-ul `buf`. Funcția returnează controlul după plasarea datelor de transmis în buffer-ele sistemului de operare al mașinii locale. Valoarea returnată de funcția `send()` este numărul de octeți scriși efectiv, sau `-1` în caz de eroare. Datele plasate în buffer-e prin apelul `send()` urmează a fi trimise spre receptor fără alte acțiuni din partea emițătorului.

În modul normal de lucru, dacă nu există spațiu suficient în buffer-ele sistemului de operare, funcția `send()` așteaptă ca aceste buffer-e să se elibereze (prin transmiterea efectivă a datelor către sistemul de operare al receptorului și citirea lor de către procesul receptor). Această așteptare are ca rol frânarea procesului emițător dacă acesta produce date la un debit mai mare decât cel cu care este capabilă rețeaua să le transmită sau procesul receptor să le preia. Prin plasarea valorii `MSG_DONTWAIT` în parametrul `flags`, acest comportament este modificat. Astfel, în acest caz, dacă nu există suficient spațiu în buffer-ele sistemului de operare, funcția `send()` scrie doar o parte din datele furnizate și returnează imediat controlul procesului apelant. În cazul în care funcția `send()` nu scrie nimic, ea returnează valoarea `-1` și setează variabila globală `errno` la valoarea `EAGAIN`. În cazul în care funcția `send()` scrie cel puțin un octet, ea returnează numărul de octeți scriși efectiv. În ambele cazuri, este sarcina procesului emițător să apeleze din nou, la un moment ulterior, funcția `send()` în vederea scrierii octeților rămași.

Deoarece funcția `send()` returnează înainte de transmiterea efectivă a datelor, eventualele erori legate de transmiterea datelor nu pot fi raportate

apelantului prin valoarea returnată de `send()`. Pot să apară două tipuri de erori: căderea rețelei și închiderea conexiunii de către receptor. Aceste erori vor fi raportate de către sistemul de operare al emițătorului procesului emițător prin aceea că apeluri `send()` ulterioare pentru același socket vor returna `-1`. În plus, pe sistemele de tip UNIX, apelul `send()` pentru o conexiune al cărui capăt destinație este închis duce la primirea de către procesul emițător a unui semnal SIGPIPE, care are ca efect implicit terminarea imediată a procesului emițător.

Funcția `recv()` extrage date sosite pe conexiune și aflate în buffer-ul sistemului de operare local. Funcția primește ca argumente identificatorul socket-ului corespunzător conexiunii, adresa unei zone de memorie unde să plaseze datele citite și numărul de octeți de citit.

Numărul de octeți de citit reprezintă numărul maxim de octeți pe care funcția îi va transfera din buffer-ul sistemului de operare în zona procesului apelant. Dacă numărul de octeți disponibili în buffer-ele sistemului de operare este mai mic, doar octeții disponibili în acel moment vor fi transferați. Dacă în momentul apelului nu există nici un octet disponibil în buffer-ele sistemului de operare local, funcția `recv()` așteaptă sosirea a cel puțin un octet. Funcția returnează numărul de octeți transferați (citiți de pe conexiune).

Comportamentul descris mai sus poate fi modificat prin plasarea uneia din următoarele valori în parametrul `flags`:

`MSG_DONTWAIT`: în cazul în care nu este nici un octet disponibil, funcția `recv()` returnează valoarea `-1` și setează variabila globală `errno` la valoarea `EAGAIN`;

`MSG_WAITALL`: funcția `recv()` așteaptă să fie disponibili cel puțin `len` octeți și citește exact `len` octeți.

Este important de notat că datele sunt transmise de la sistemul de operare emițător spre cel receptor în fragmente (pachete), că împărțirea datelor în fragmente este independentă de modul în care au fost furnizate prin apeluri `send()` succesive și că, în final, fragmentele ce vor fi disponibile succesiv pentru receptor sunt independente de fragmentele furnizate în apelurile `send()`. Ca urmare, este posibil ca emițătorul să trimită, prin două apeluri `send()` consecutive, șirurile de octeți `abc` și `def`, iar receptorul, apelând repetat `recv()` cu `len=3` și `flags=0`, să primească `ab`, `cd` și `ef`. Singurul lucru garantat este că prin concatenarea tuturor fragmentelor trimise de emițător se obține același șir de octeți ca și prin concatenarea tuturor fragmentelor primite de receptor.

În cazul închiderii conexiunii de către emițător, apelurile `recv()` efectuate de procesul receptor vor citi mai întâi datele rămase în buffer-e, iar

după epuizarea acestora vor returna valoarea 0. Prin urmare, funcţia `recv()` returnează valoarea 0 dacă şi numai dacă emiţătorul a închis conexiunea şi toate datele trimise înainte de închiderea conexiunii au fost deja citite. Dealtfel, valoarea 0 returnată de `recv()` sau `read()` este semnalizarea uzuală a terminării datelor de citit şi se utilizează şi pentru a semnaliza sfârşitul unui fişier sau închiderii capătului de scriere într-un *pipe* UNIX.

8.1.3.10. Funcţiile `shutdown()` şi `close()`

```
int shutdown(int sd, int how);
int close(int sd);
```

Funcţia `shutdown()` închide sensul de emisie, sensul de recepţie sau ambele sensuri de comunicaţie ale conexiunii identificate de identificatorul de socket `sd`, conform valorii parametrului `how`: `SHUT_WR`, `SHUT_RD` sau respectiv `SHUT_RDWR`. Utilitatea principală a funcţiei este închiderea sensului de emisie pentru a semnaliza celuilalt capăt terminarea datelor transmise (apelurile `recv()` din procesul de la celălalt capăt al conexiunii vor returna 0). Funcţia `shutdown()` poate fi apelată doar pe un socket conectat şi nu distruge socket-ul.

Funcţia `close()` distruge socket-ul `sd`. Dacă socket-ul era un socket conectat în acel moment, închide ambele sensuri de comunicaţie. După apelul `close()`, identificatorul de socket este eliberat şi poate fi utilizat ulterior de către sistemul de operare pentru a identifica socket-uri sau alte obiecte create ulterior. Apelul `close()` este necesar pentru a elibera resursele ocupate de socket. Poate fi efectuat oricând asupra oricărui tip de socket.

Terminarea unui proces, indiferent de modul de terminare, are ca efect şi distrugerea tuturor socket-urilor existente în acel moment, printr-un mecanism identic cu câte un apel `close()` pentru fiecare socket.

8.1.3.11. Funcţiile `sendto()` şi `recvfrom()`

```
ssize_t sendto(int sd, const void *buf, size_t len, int flags,
               const struct sockaddr *to, socklen_t tolen);
ssize_t recvfrom(int sd, void *buf, size_t len, int flags,
                 struct sockaddr *from, socklen_t *fromlen);
```

Funcţia `sendto()` trimite o datagramă de pe un socket *dgram*. Parametrii reprezintă :

- `sd`: socket-ul de pe care se transmite datagrama, adică a cărui adresă va fi utilizată ca adresă sursă a datagramei;

- **to**: pointer spre structura ce conține adresa de rețea a destinatarului; **tolen** reprezintă lungimea structurii pointate de **to**;
- **buf**: pointer spre o zonă de memorie ce conține datele utile; **len** reprezintă lungimea datelor utile. Datele utile sunt un șir arbitrar de octeți.

Funcția returnează numărul de octeți ai datagramei trimise (adică valoarea lui **len**) în caz de succes și -1 în caz de eroare. Funcția returnează controlul apelantului înainte ca pachetul să fie livrat destinatarului și ca urmare eventuala pierdere a pachetului nu poate fi raportată apelantului.

Funcția `recvfrom()` citește din bufferele sistemului de operare local următoarea datagramă adresată socket-ului `dat` ca parametru. Dacă nu există nici o datagramă, funcția așteaptă sosirea următoarei datagrame, cu excepția cazului în care `flags` conține valoarea `MSG_DONTWAIT`, caz în care `recvfrom()` returnează imediat valoarea -1 și setează `errno` la valoarea `EAGAIN`.

Datagrama este citită în zona de memorie pointată de parametrul `buf` și a cărei dimensiune este dată în variabila `len`. Funcția `recvfrom()` returnează dimensiunea datagramei. Dacă datagrama este mai mare decât valoarea parametrului `len`, finalul datagramei este pierdut; funcția `recvfrom()` nu scrie niciodată dincolo de `len` octeți în memoria procesului.

Adresa emițătorului datagramei este plasată de funcția `recvfrom()` în variabila pointată de `from`. Parametrul `fromlen` trebuie să poarte la o variabilă de tip întreg a cărei valoare, înainte de apelul `recvfrom()`, trebuie să fie egală cu dimensiunea, în octeți, a zonei de memorie alocate pentru adresa emițătorului. Funcția `recvfrom()` modifică această variabilă, punând în ea dimensiunea utilizată efectiv pentru scrierea adresei emițătorului.

8.1.4. Exemple

8.1.4.1. Comunicare prin conexiune

Dăm mai jos textul sursă (în C pentru Linux) pentru un client care se conectează la un server TCP/IPv4 specificat prin numele mașinii și numărul portului TCP (date ca argumente în linia de comandă), îi trimite un șir de caractere fixat (`abcd`), după care citește și afișează tot ce trimite server-ul.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main(int argc, char* argv[])
{
```

```
int port, sd, r;
struct hostent* hh;
struct sockaddr_in adr;
char buf[100];
if(argc!=3){
    fprintf(stderr, "Utilizare: cli adresa port\n");
    return 1;
}
memset(&adr, 0, sizeof(adr));
adr.sin_family = AF_INET;
if(1!=sscanf(argv[2], "%d", &port)){
    fprintf(stderr, "numarul de port trebuie sa fie un numar\n");
    return 1;
}
adr.sin_port = htons(port);
hh=gethostbyname(argv[1]);
if(hh==0 || hh->h_addrtype!=AF_INET || hh->h_length<=0){
    fprintf(stderr, "Nu se poate determina adresa serverului\n");
    return 1;
}
memcpy(&adr.sin_addr, hh->h_addr_list[0], 4);
sd=socket(PF_INET, SOCK_STREAM, 0);
if(-1==connect(sd, (struct sockaddr*)&adr, sizeof(adr)) )
{
    perror("connect()");
    return 1;
}
send(sd, "abcd", 4, 0);
shutdown(sd, SHUT_WR);
while((r=recv(sd, buf, 100, 0))>0){
    write(1,buf,r);
}
if(r==-1){
    perror("recv()");
    return 1;
}
close(sd);
return 0;
}
```

Dăm în continuare textul sursă pentru un server care aşteaptă conectarea unui client pe portul specificat în linia de comandă, afişează adresa de la care s-a conectat clientul (adresa IP şi numărul de port), citeşte de pe conexiune

și afișează pe ecran tot ce transmite clientul (până la închiderea sensului de conexiune de la client la server) și apoi trimite înapoi textul xyz.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main(int argc, char* argv[])
{
    int sd, sd_c, port, r;
    struct sockaddr_in my_addr, cli_addr;
    socklen_t cli_addr_size;
    char buf[100];
    if(argc!=2){
        fprintf(stderr, "Utilizare: srv port\n");
        return 1;
    }
    memset(&my_addr, 0, sizeof(my_addr));
    my_addr.sin_family = AF_INET;
    if(1!=sscanf(argv[1], "%d", &port)){
        fprintf(stderr, "numarul de port trebuie sa fie un numar\n");
        return 1;
    }
    my_addr.sin_port=htons(port);
    my_addr.sin_addr.s_addr=htonl(INADDR_ANY);
    sd=socket(PF_INET, SOCK_STREAM, 0);
    if(-1==bind(sd, (struct sockaddr*)&my_addr,
        sizeof(my_addr)) )
    {
        perror("bind()");
        return 1;
    }
    listen(sd, 1);
    cli_addr_size=sizeof(cli_addr);
    sd_c = accept(sd, (struct sockaddr*)&cli_addr,
        &cli_addr_size);
    printf("client conectat de la %s:%d\n",
        inet_ntoa(cli_addr.sin_addr),
        ntohs(cli_addr.sin_port)
    );
    close(sd);
    while((r=recv(sd_c, buf, 100, 0))>0){
```



```

    write(1,buf,r);
}
if(r==-1){
    perror("recv()");
    return 1;
}
send(sd_c, "xyz", 3, 0);
close(sd_c);
return 0;
}

```

8.1.4.2. Comunicare prin datagrame

Mai jos este descris un client UDP/IPv4 care se conectează la un server specificat prin numele mașinii sau adresa IP și numărul de port. Clientul trimite serverului o datagramă de 4 octeți conținând textul `abcd` și așteaptă o datagramă ca răspuns, a cărei conținut îl afișează.

```

#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
int main(int argc, char* argv[])
{
    int port, sd, r;
    struct hostent* hh;
    struct sockaddr_in adr;
    socklen_t adr_size;
    char buf[100];
    if(argc!=3){
        fprintf(stderr, "Utilizare: cli adresa port\n");
        return 1;
    }
    memset(&adr, 0, sizeof(adr));
    adr.sin_family = AF_INET;
    if(1!=sscanf(argv[2], "%d", &port)){
        fprintf(stderr, "numarul de port trebuie sa fie un numar\n");
        return 1;
    }
    adr.sin_port = htons(port);
    hh=gethostbyname(argv[1]);
    if(hh==0 || hh->h_addrtype!=AF_INET || hh->h_length<=0){

```

```

    fprintf(stderr, "Nu se poate determina adresa serverului\n");
    return 1;
}
memcpy(&adr.sin_addr, hh->h_addr_list[0], 4);
sd=socket(PF_INET, SOCK_DGRAM, 0);
if(sd==-1){
    perror("socket()");
    return 1;
}
if(-1==sendto(sd, "abcd", 4, 0,
    (struct sockaddr*)&adr, sizeof(adr)) )
{
    perror("sendto()");
    return 1;
}
adr_size=sizeof(adr);
r=recvfrom(sd, buf, 100, 0,
    (struct sockaddr*)&adr, &adr_size);
if(r==-1){
    perror("recvfrom()");
    return 1;
}
printf("datagrama primita de la de la %s:%d\n",
    inet_ntoa(adr.sin_addr),
    ntohs(adr.sin_port)
);
buf[r]=0;
printf("continut: \"%s\"\n", buf);
close(sd);
return 0;
}

```

În continuare descriem un server UDP/IPv4. Acesta așteaptă o datagramă de la un client, afișează adresa de la care a fost trimisă datagrama precum și conținutul datagramii primite. Apoi trimite înapoi, la adresa de la care a sosit datagrama de la client, o datagramă conținând șirul de 3 octeți xyz.

```

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>

```

```
int main(int argc, char* argv[])
{
    int sd, port, r;
    struct sockaddr_in my_addr, cli_addr;
    socklen_t cli_addr_size;
    char buf[101];
    if(argc!=2){
        fprintf(stderr, "Utilizare: srv port\n");
        return 1;
    }
    memset(&my_addr, 0, sizeof(my_addr));
    my_addr.sin_family = AF_INET;
    if(1!=sscanf(argv[1], "%d", &port)){
        fprintf(stderr, "numarul de port trebuie sa fie un numar\n");
        return 1;
    }
    my_addr.sin_port=htons(port);
    my_addr.sin_addr.s_addr=htonl(INADDR_ANY);
    sd=socket(PF_INET, SOCK_DGRAM, 0);
    if(-1==bind(sd, (struct sockaddr*)&my_addr,
        sizeof(my_addr)) )
    {
        perror("bind()");
        return 1;
    }
    cli_addr_size=sizeof(cli_addr);
    r=recvfrom(sd, buf, 100, 0,
        (struct sockaddr*)&cli_addr, &cli_addr_size);
    if(r==-1){
        perror("recvfrom()");
        return 1;
    }
    printf("datagrama primita de la de la %s:%d\n",
        inet_ntoa(cli_addr.sin_addr),
        ntohs(cli_addr.sin_port)
    );
    buf[r]=0;
    printf("continut: \"%s\"\n", buf);
    sendto(sd, "xyz", 3, 0,
        (struct sockaddr*)&cli_addr, cli_addr_size);
    close(sd);
    return 0;
}
```

8.2. Formatarea datelor

Diferite formate de reprezentare a datelor pe conexiune au fost descrise în capitolul 7. În acest paragraf ne vom ocupa de problemele privind transmiterea și recepția datelor în astfel de formate.

8.2.1. Formate binare

Formatele binare sunt asemănătoare cu formatele utilizate de programele compilate pentru stocarea datelor în variabilele locale. Până la un punct, este rezonabilă transmiterea unei informații prin instrucțiuni de forma

```
Tip msg;
...
send(sd, &msg, sizeof(msg), 0);
```

și recepția prin

```
Tip msg;
...
recv(sd, &msg, sizeof(msg), MSG_WAITALL);
```

unde `Tip` este un tip de date oarecare declarat identic în ambele programe (emițător și receptor).

Există însă câteva motive pentru care o astfel de abordare nu este, în general, acceptabilă. Vom descrie în continuare problemele legate de fiecare tip de date în parte, precum și câteva idei privind rezolvarea lor.

8.2.1.1. Tipuri întregi

La transmiterea variabilelor întregi apar două probleme de portabilitate:

- dimensiunea unui întreg nu este, în general, standardizată exact (în C/C++ un `int` poate avea 16, 32 sau 64 de biți);
- ordinea octeților în memorie (*big endian* sau *little endian*) depinde de arhitectura calculatorului.

Dacă scriem un program pentru un anumit tip de calculatoare și pentru un anumit compilator, pentru care știm exact dimensiunea unui `int` și ordinea octeților, putem transmite și recepționa date prin secvențe de tipul:

```
int a;
...
send(sd, &a, sizeof(a), 0);
```

pentru emițător și

```
int a;
...
recv(sd, &a, sizeof(a), MSG_WAITALL);
```

pentru receptor. Dacă însă emițătorul este compilat pe o platformă pe care `int` are 16 biți și este reprezentat *big endian*, iar receptorul este compilat pe o platformă pe care `int` are 32 de biți și este *little endian*, cele două programe nu vor comunica corect.

Pentru a putea scrie programe portabile, biblioteca C standard pe sisteme de tip UNIX conține, în header-ul `arpa/inet.h`:

- `typedef`-uri pentru tipuri întregi de lungime standardizată (independentă de compilator): `uint16_t` de 16 biți și `uint32_t` de 32 de biți;
- funcții de conversie între formatul local (*little endian* sau *big endian*, după caz) și formatul *big endian*, utilizat cel mai adesea pentru datele transmise în Internet. Aceste funcții sunt: `htons()` și `htonl()` (de la *host to network, short*, respectiv *host to network, long*), pentru conversia de la format local la format *big endian* (numit și *format rețea*), și `ntohs()` și `ntohl()` pentru conversia în sens invers. Variantele cu `s` (`htons()` și `ntohs()`) convertesc întregi de 16 biți (de tip `uint16_t`, iar cele cu `l` convertesc întregi de 32 de biți (`uint32_t`).

Implementarea acestor `typedef`-uri și funcții depinde de platformă (de arhitectură și de compilator). Utilizarea lor permite ca restul sursei programului să nu depindă de platformă.

Transmiterea unui întreg, într-un mod portabil, se face astfel:

```
uint32_t a;
...
a=htonl(a);
send(sd, &a, sizeof(a), 0);
```

```
uint32_t a;
...
recv(sd, &a, sizeof(a), MSG_WAITALL);
a=ntohl(a);
```

Indiferent pe ce platformă sunt compilate, fragmentele de mai sus emit, respectiv recepționează, un întreg reprezentat pe 32 de biți în format *big endian*.

8.2.1.2. Șiruri de caractere și tablouri

Transmiterea sau memorarea unui tablou necesită transmiterea (respectiv memorarea), într-un fel sau altul, a numărului de elemente din tablou. Două metode sunt frecvent utilizate în acest scop: transmiterea în prealabil a numărului de elemente și transmiterea unui element cu valoare speciale (terminator) după ultimul element.

Pe lângă numărul de elemente efective ale tabloului este necesară cunoașterea numărului de elemente alocate. La reprezentarea în memorie sau în fișiere pe disc, sunt utilizate frecvent tablouri de dimensiune fixată la compilare. Avantajul dimensiunii fixe este că variabilele situate după tabloul respectiv se pot plasa la adrese fixe și pot fi accesate direct; dezavantajul este un consum sporit de memorie și o limită mai mică a numărului de obiecte ce pot fi puse în tablou.

La transmiterea tablourilor prin conexiuni în rețea, de regulă numărul de elemente transmise este egal cu numărul de elemente existente în mod real, plus elementul terminator (dacă este adoptată varianta cu terminator). Nu sunt utilizate tablouri de lungime fixă deoarece datele situate după tablou oricum nu pot fi accesate direct.

În cazul reprezentării cu număr de elemente, receptorul citește întâi numărul de elemente, după care alocă spațiu (sau verifică dacă spațiul alocat este suficient) și citește elementele. În cazul reprezentării cu terminator, receptorul citește pe rând fiecare element și-i verifică valoarea; la întâlnirea terminatorului se oprește. Înainte de-a citi fiecare element, receptorul trebuie să verifice dacă mai are spațiu alocat pentru acesta, iar în caz contrar fie să re-aloce spațiu pentru tablou și să copieze în spațiul nou alocat elementele citite, fie să renunțe și să semnaleze eroare.

EXEMPLUL 8.2: Se cere transmiterea unui șir de caractere. Reprezentarea șirului pe conexiune este: un întreg pe 16 biți *big endian* reprezentând lungimea șirului, urmat de șirul propriu-zis (reprezentare diferită deci de reprezentarea uzuală în memorie, unde șirul se termină cu un caracter nul). Descriem în continuare emițătorul:

```
char* s;  
uint16_t l;  
...  
l=htons(strlen(s));  
send(sd, &l, 2, 0);  
send(sd, s, strlen(s), 0);
```

și receptorul:

```

char* s;
uint16_t l;
if(2==recv(sd, &l, 2, MSG_WAITALL) &&
    0!=(s=new char[l=ntohs(l)+1]) &&
    l==recv(sd, s, l, MSG_WAITALL)){
    s[l]=0;
    // sir citit cu succes
} else {
    // tratare eroare
}

```

De remarcat la receptor necesitatea de-a reface terminatorul nul, netransmis prin rețea.

EXEMPLUL 8.3: Se cere transmiterea unui șir de caractere. Reprezentarea pe conexiune va fi ca un șir de caractere urmat de un caracter nul (adică reprezentare identică celei din memorie, dar pe lungime variabilă, egală cu minimul necesar). Emițătorul este:

```

char* s;
...
send(sd, s, strlen(s)+1, 0);

```

Receptorul:

```

char s[500];
int dim_alloc=500, pos, ret;
while(pos<dim_alloc-1 &&
    1==(ret=recv(sd, s+pos, 1, 0)) &&
    s[pos++]!=0) {}
if(ret==1 && s[pos-1]==0){
    // sir citit cu succes
} else {
    // tratare eroare
}

```

8.2.1.3. Variabile compuse (struct-uri)

La prima vedere, variabilele compuse (**struct**-urile) sunt reprezentate la fel și în memorie și pe conexiune — se reprezintă câmpurile unul după altul — și ca urmare transmiterea lor nu ridică probleme deosebite.

Din păcate însă, reprezentarea unei structuri în memorie depinde de platformă (arhitectura calculatorului și compilator), datorită problemelor

privind alinierea întregilor. Din considerente legate de arhitectura magistralei de date a calculatorului (detalii ce ies din cadrul cursului de față), accesarea de către procesor a unei variabile de tip întreg sau real a cărei adresă în memorie nu este multiplu de un anumit număr de octeți este pentru unele procesoare imposibilă iar pentru celelalte inefficientă. Numărul ce trebuie să dividă adresa se numește *aliniere* și este de obicei minimul dintre dimensiunea variabilei și lățimea magistralei. Astfel, dacă magistrala de date este de 4 octeți, întregii de 2 octați trebuie să fie plasați la adrese pare, iar întregii de 4 sau 8 octeți trebuie să fie la adrese multiplu de 4; nu există restricții cu privire la caractere (întregi pe 1 octet). Compilatorul, împreună cu funcțiile de alocare dinamică a memoriei, asigură alinierea recurgând la următoarele metode:

- adaugă octeți nefolosiți între variabile,
- adaugă octeți nefolosiți între câmpurile unei structuri,
- adaugă octeți nefolosiți la finalul unei structuri ce face parte dintr-un tablou,
- alocă variabilele de tip structură la adrese multiplu de o lățimea magistralei.

Ca urmare, reprezentarea în memorie a unei structuri depinde de platformă și, în consecință, un fragment de cod de forma:

```
struct Msg {
    char c;
    uint32_t i;
};
Msg m;
...
m.i=htonl(m.i);
send(sd, &m, sizeof(m), 0);
```

este neportabil. În funcție de lățimea magistralei, de faptul că alinierea incorectă duce la imposibilitatea accesării variabilei sau doar la inefficientă și, în acest din urmă caz, de opțiunile de compilare, dimensiunea structurii `Msg` de mai sus poate fi 5, 6 sau 8 octeți, între cele două câmpuri fiind respectiv 0, 1 sau 3 octeți neutilizați.

Rezolvarea problemei de portabilitate se face transmițând separat fiecare câmp:

```
struct Msg {
    char c;
    uint32_t i;
```



```
};  
Msg m;  
...  
m.i=htonl(m.i);  
send(sd, &m.c, 1, 0);  
send(sd, &m.i, 4, 0);
```

8.2.1.4. Pointeri

Deoarece un pointer este o adresă în cadrul unui proces, transmiterea unui pointer către un alt proces este complet inutilă.

8.2.2. Formate text

Într-un format text, fiecare câmp este în esență un șir de caractere terminat cu spațiu, tab, *newline* sau un alt caracter specificat prin standard. Metodele descrise pentru transmiterea și recepționarea unui șir de caractere se aplică și la obiectele transmise în formate de tip text.

8.2.3. Probleme de robustețe și securitate

Orice apel de funcție sistem poate eșua din multe motive; ca urmare, la fiecare apel `send()` sau `recv()` programul trebuie să verifice valoarea returnată.

Un receptor robust trebuie să se comporte rezonabil la orice fel de date trimise de partenerul de comunicație, inclusiv în cazul încălcării de către acesta a standardului de reprezentare a datelor și inclusiv în cazul în care emițătorul închide conexiunea în mijlocul transmiterii unei variabile.

Validitatea datelor trebuie verificată întotdeauna după citire. Dacă receptorul așteaptă un întreg pozitiv, este necesar să se verifice prin program că numărul primit este într-adevăr pozitiv. Este de asemenea necesar să se stabilească și să se impună explicit niște limite maxime. Astfel, să presupunem că programul receptor primește un șir de întregi reprezentat prin lungimea, ca număr de elemente, pe 32 de biți, urmată de elementele propriu-zise. Chiar dacă de principiu numărul de elemente ne așteptăm să fie între 1 și câteva sute, trebuie să ne asigurăm că programul se comportă rezonabil dacă numărul de elemente anunțat de emițător este 0, 2147483647 (adică $2^{31} - 1$) sau alte asemenea valori. Comportament rezonabil înseamnă fie să fie capabil să proceseze corect datele, fie să declare eroare și să încheie curat operațiile începute.

Orice program care nu este robust este un risc de securitate.

8.2.4. Probleme privind costul apelurilor sistem

Apelul funcțiilor `send()` și `recv()` este scump, în termeni de timp de procesor, deoarece, fiind funcții sistem, necesită o comutare de drepturi în procesor, salvarea și restaurarea contextului apelului și o serie de verificări din partea nucleului sistemului de operare; în total, echivalentul câtorva sute de instrucțiuni. Acest cost este independent de numărul de octeți transferați.

Este, prin urmare, eficient ca fiecare apel `send()` sau `recv()` să transfere cât de mulți octeți se poate. Un program care trimite date este bine să pregătească datele într-o zonă tampon locală și să trimită totul printr-un singur apel `send()`. Un program care primește date este bine să ceară (prin `recv()`) câte un bloc mai mare de date și apoi să analizeze datele primite. Acest mod de lucru se realizează cel mai bine prin intermediul unor funcții de bibliotecă adecvate.

Descriem în continuare funcțiile din biblioteca standard C utilizabile în acest scop. Biblioteca poate fi utilizată atât pentru emisie și recepție printr-o conexiune *socket stream* cât și pentru citire sau scriere într-un fișier sau pentru comunicare prin *pipe* sau *fifo*.

Elementul principal al bibliotecii este structura `FILE`, ce conține:

- un identificator de fișier deschis, conexiune *socket stream*, *pipe* sau *fifo*;
- o zonă de memorie tampon, împreună cu variabilele necesare gestionării ei.

Funcțiile de citire ale bibliotecii sunt `fread()`, `fscanf()`, `fgets()` și `fgetc()`. Fiecare dintre aceste funcții extrage datele din zona tampon a structurii `FILE` dată ca parametru. Dacă în zona tampon nu sunt suficienți octeți pentru a satisface cererea, aceste funcții apelează funcția sistem `read()` asupra identificatorului de fișier din structura `FILE` pentru a obține octeții următori. Fiecare astfel de apel `read()` încearcă să citească câțiva kiloocteți. Pentru fiecare din funcțiile de mai sus, dacă datele de returnat aplicației se găsesc deja în zona tampon, costul execuției este de câteva instrucțiuni pentru fiecare octet transferat. Ca urmare, la citirea a câte un caracter o dată, utilizarea funcțiilor de mai sus poate reduce timpul de execuție de câteva zeci de ori.

EXEMPLUL 8.4: Fie următoarele fragmente de cod:

```
int sd;
char s[512];
int i,r;
...
while( ((r=recv(sd, s+i, 1, 0))==1 && s[i++]!=0 ) {}
```

și

```
FILE* f;
char s[512];
int i,r;
...
while( (r=fgetc(f))!=EOF && (s[i++]=r)!=0) {}
```

Ambele fragmente de cod citesc de pe un *socket stream* un șir de octeți terminat cu un caracter nul. Primul fragment de cod apelează `recv()` o dată pentru fiecare caracter al șirului. Al doilea fragment apelează `fgetc()` o dată pentru fiecare caracter al șirului. La o mică parte dintre aceste apeluri, funcția `fgetc()` va apela în spate funcția sistem `read()` pentru a citi efectiv datele de pe conexiune; la restul apelurilor, `fgetc()` returnează apelantului câte un caracter aflat deja în zona tampon. Ca rezultat global, al doilea fragment de cod se va executa de câteva zeci de ori mai repede decât primul.

Testarea închiderii conexiunii și terminării datelor se poate face apelând funcția `foef()`. De notat că această funcție poate să returneze `false` chiar dacă s-a ajuns la finalul datelor; rezultatul `true` este garantat doar după o tentativă nereușită de-a citi dincolo de finalul datelor transmise.

Pentru scriere, funcțiile de bibliotecă corespunzătoare sunt `fwrite()`, `fprintf()`, `fputs()` și `fputc()`. Aceste funcții scriu datele în zona tampon din structura `FILE`. Transmiterea efectivă pe conexiune (sau scrierea în fișier) se face automat la umplerea zonei tampon. Dacă este necesar să ne asigurăm că datele au fost transmise efectiv (sau scrise în fișier), funcția `fflush()` efectuează acest lucru. Funcția `fclose()` de asemenea trimite sau scrie ultimele date rămase în zona tampon.

Asocierea unei zone tampon unei conexiuni deja deschise se face apelând funcția `fdopen()`. Funcția `fdopen()` primește doi parametri. Primul parametru este identificatorul de socket căruia trebuie să-i asocieze zona tampon (identificatorul returnat de funcția `socket()` sau `accept()`). Al doilea parametru specifică funcției `fdopen()` dacă trebuie să asocieze zona tampon pentru citire sau pentru scriere; este de tip șir de caractere și poate avea valoarea `"r"` pentru citire sau `"w"` pentru scriere. Pentru un *socket stream*, cele două sensuri functionând complet independent, aceluiași socket i se pot asocia două zone tampon (două structuri `FILE`), câte una pentru fiecare sens.

Funcția `fclose()` scrie informațiile rămase în zona tampon (dacă zona tampon a fost creată pentru sensul de scriere), eliberează memoria alocată zonei tampon și închide conexiunea.

8.3. Probleme de concurență în comunicație

O particularitate a majorității programelor ce comunică în rețea este aceea că trebuie să răspundă prompt la mesaje provenind din surse diferite și într-o ordine necunoscută dinainte.

Să luăm de exemplu un server *ssh* (§ 11.2.1). Serverul poate avea mai mulți clienți conectați simultan. La fiecare moment, este imposibil de prezis care dintre clienți va trimite primul o comandă.

Dacă serverul execută un apel `recv()` blocant de pe socket-ul unui client, serverul va fi pus în așteptare până ce acel client va trimite date. Este posibil ca utilizatorul ce comandă acel client să stea 10 minute să se gândească. Dacă în acest timp un alt client trimite o comandă, serverul nu o va putea „vedea” cât timp este blocat în așteptarea datelor de la primul client. Ca urmare, datele de la al doilea client vor aștepta cel puțin 10 minute pentru a fi procesate.

Există mai multe soluții la problema de mai sus:

- Serverul citește de la clienți, pe rând, prin apeluri `recv()` neblocante (cu flagul `MSG_DONTWAIT`):

```
for(i=0 ; true ; i=(i+1)%nr_clienti){
    r=recv(sd[i], buf, dim, MSG_DONTWAIT);
    if(r>=0 || errno!=EAGAIN){
        /* prelucreaza mesajul primit
        sau eroarea aparuta */
    }
}
```

În acest fel, serverul nu este pus în așteptare dacă un client nu i-a trimis nimic. Dezavantajul soluției este acela că, dacă o perioadă de timp nici un client nu trimite nimic, atunci bucla se execută în mod repetat, consumând inutil timp de procesor.

- Pentru evitarea inconvenientului soluției anterioare, sistemele de operare de tip UNIX oferă o funcție sistem, numită `select()`, care primește o listă de identificatori de *socket* și, opțional, o durată, și pune procesul în așteptare până când fie există date disponibile pe vreunul din *socket*-ii dați, fie expiră durata de timp specificată.
- O abordare complet diferită este aceea de-a crea mai multe procese — sau, în sistemele de operare moderne, mai multe fire de execuție (thread-uri) în cardul procesului server — fiecare proces sau fir de execuție urmărind un singur client. În acest caz, procesul sau firul de execuție poate executa `recv()` blocant asupra socket-ului corespunzător clientului său. În lipsa

activității clienților, fiecare proces sau fir de execuție al serverului este blocat în apelul `recv()` asupra socket-ului corespunzător. În momentul în care un client trimite date, nucleul sistemului de operare trezește procesul sau firul ce execută `recv()` pe socket-ul corespunzător; procesul sau firul execută prelucrările necesare după care probabil execută un nou `recv()` blocant.

Cazul unui server cu mai mulți clienți nu este singura situație în care este nevoie de a urmări simultan evenimente pe mai multe canale. Alte situații sunt:

- un client care trebuie să urmărească simultan acțiunile utilizatorului și mesajele sosite de la server;
- un server care poate trimite date cu debit mai mare decât capacitatea rețelei sau capacitatea de prelucrare a clientului; în acest caz serverul are de urmărit simultan, pe de o parte noi cereri ale clienților, iar pe de altă parte posibilitatea de-a trimite noi date spre clienți în urma faptului că vechile date au fost prelucrate de aceștia.
- un server care trebuie să preia mesaje de la clienții conectați și, în același timp, să poată accepta clienți noi.

Un aspect important ce trebuie urmărit în proiectarea unui server concurent este servirea echitabilă a clienților, independent de comportamentul acestora. Dacă un client trimite cereri mai repede decât este capabil serverul să-l servească, serverul trebuie să execute o parte din cereri, apoi să servească cereri ale celorlalți clienți, apoi să revină la primul și să mai proceseze o parte din cereri și așa mai departe. Nu este permis ca un client care inundă serverul cu cereri să acapareze întreaga putere de calcul a serverului și ceilalți clienți să aștepte la infinit.

Capitolul 9

Rețele IEEE 802

Vom studia în continuare standardul utilizat de cele mai multe rețele locale. IEEE 802 definește de fapt o familie de tipuri de rețele locale, dintre care cele mai des întâlnite sunt:

- rețele *Ethernet* (de 10, 100 sau 1000 Mbit/s), construite conform standardului IEEE 802.3;
- rețele numite *Wireless Ethernet*, conform standardului IEEE 802.11.

9.1. Rețele IEEE 802.3 (Ethernet)

Cele mai multe rețele locale (rețele cu întinderi geografice reduse, de până la câțiva kilometri) sunt construite pe baza standardului IEEE 802.3 [IEEE 802.3, 2005]. Astfel de rețele mai sunt numite, în mod curent, rețele *Ethernet* (denumirea standardului original, din care a fost dezvoltat standardul IEEE 802.3) sau rețele *UTP 10/100/1000* (UTP vine de la *unshielded twisted pairs* — *perechi torsadate neecranate* — și desemnează tipul de cablu utilizat cel mai frecvent în instalațiile actuale, iar 10/100/1000 sunt capacitățile posibile ale legăturilor, măsurate în megabiți pe secundă).

Standardul este complex (are peste 1500 de pagini) și a rezultat în urma unei evoluții întinse pe mai mult de 20 de ani. În cele ce urmează vom trece în revistă aspectele mai importante.

Componentele din care se realizează o rețea Ethernet sunt:

Interfața de rețea sau *placa de rețea* (engl. *Network Interface Card* — *NIC*) este dispozitivul prin care se conectează un calculator la rețea.

Cablul magistrală. O rețea constrită cu cablu magistrală consta într-un cablu, format din două conductoare izolate între ele, la care sunt

conectate, în paralel, interfețele de rețea ale calculatoarelor (fig. 9.1). În acest sistem, semnalul emis de orice interfață de rețea este recepționat de toate celelalte interfețe de rețea conectate la acel cablu.

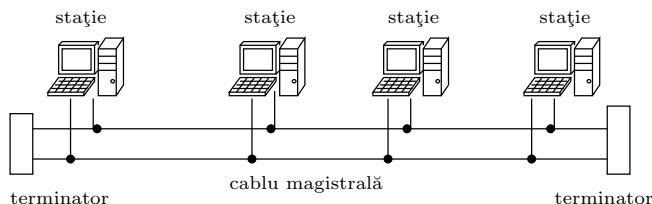


Figura 9.1: O rețea Ethernet construită cu un cablu magistrală

Comunicația se face prin pachete de dimensiune variabilă.

Două interfețe care emit simultan își bruiază reciproc semnalele emise; este necesar deci un mecanism de control al accesului la mediu (vezi § 4.2). IEEE 802.3 alege soluția cu detectarea coliziunilor și retransmiterea pachetelor distruse de coliziuni.

Deoarece fiecare interfață de rețea „aude” toate pachetele emise în rețea, este prevăzut un mecanism prin care interfața să identifice și să livreze sistemului de operare numai pachetele ce îi sunt destinate. Anume, fiecare interfață de rețea are o adresă unică, numită *adresă fizică* sau *adresă MAC* și fiecare pachet poartă adresa sursei și adresa destinației.

Repetorul (engl. *repeater*) este un dispozitiv care este conectat la mai multe cabluri de rețea și copiază pachetele de date de pe fiecare cablu pe celelalte.

Repetorul este conectat la fiecare cablu de rețea întocmai ca o interfață de rețea a unui calculator. Interfața repetorului către cablul de rețea se numește *port*.

Oridecâteori repetorul recepționează un pachet printr-unul dintre porturile sale (printr-unul din cablurile de rețea conectate la repetor), îl retransmite (repetă) pe toate celelalte cabluri de rețea conectate (toate cu excepția celui prin care a intrat pachetul). Retransmiterea se face cu întârziere de ordinul duratei câtorva biți, în orice caz mai puțin decât durata unui pachet. Dacă repetorul recepționează simultan pachete prin două sau mai multe porturi, consideră că are loc o *coliziune* și semnalizează acest lucru emițând, prin toate porturile, un semnal special de anunțare a coliziunii. Acest semnal de coliziune se propagă în toată rețeaua.

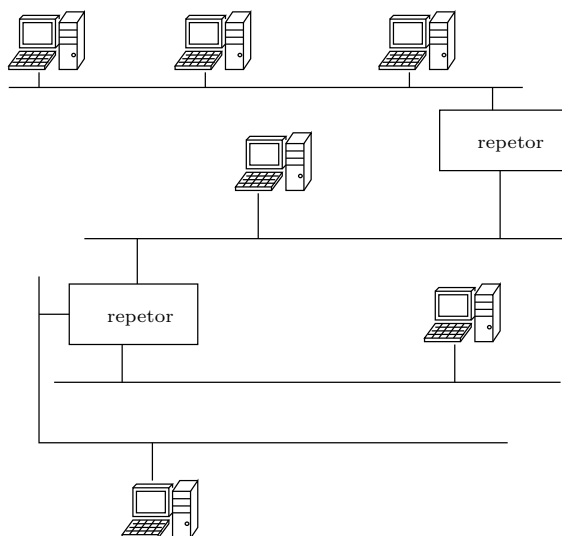


Figura 9.2: O rețea construită din mai multe cabluri magistrală interconectate prin repetitoare.

Repetoarele permit construirea unei rețele întinse pe o distanță mai mare decât lungimea maximă a unui singur cablu (lungime limitată de atenuarea semnalului pe cablu). O rețea construită cu repetitoare este desenată în figura 9.2.

Odată cu ieftinirea repetitoarelor, a devenit curentă utilizarea câte unui cablu pentru conectarea fiecărui calculator la un repetitor. În acest fel, un cablu de rețea va avea legate la el doar două echipamente: fie o interfață de rețea și un repetitor, fie două repetitoare, fie două interfețe de rețea (în acest din urmă caz rezultă o rețea formată doar din două calculatoare). De regulă, cablul de legătură folosit în aceste cazuri este o legătură duplex (vezi mai jos) și poate conecta doar două echipamente. Repetitoarele utilizate în această situație se mai numesc *hub-uri* (eng. hub = butuc de roată).

Comutatoarele. Un *comutator* (eng. *switch*) este un dispozitiv asemănător cu un repetitor, dar cu următoarele modificări:

- este capabil să memoreze câte un pachet întreg pentru fiecare port;
- dacă primește simultan două sau mai multe pachete, le memorează și le retransmite pe rând;
- dacă este posibil, în loc să retransmită un pachet prin toate porturile comutatorului, îl retrimite doar pe calea către interfața de rețea

căreia îi este destinat pachetul (a se vedea § 9.1.5 pentru detalii).

Legăturile duplex. Cablurile de legătură între două echipamente pot fi făcute cu căi independente pentru cele două sensuri. Dacă și echipamentele conectate sunt capabile să emită și să recepționeze simultan, este posibilă realizarea unei comunicații duplex între cele două echipamente.

Există în cadrul IEEE 802.3 mai multe sub-standarde legate de nivelul fizic, privitoare la cablurile de legătură între echipamente. Cu excepția debitului de comunicație și a existenței sau absenței posibilității comunicației duplex, tipul cablului de legătură ales nu afectează restul rețelei.

Pentru echipamente capabile să funcționeze după mai multe standarde privind nivelul fizic (debite diferite și mod semi-duplex sau duplex), există un protocol de negociere al modului de transmisie la nivel fizic folosit; acesta va fi studiat în § 9.1.1 cu ocazia prezentării standardului 10 Base T.

9.1.1. Legături punct la punct prin perechi de conductoare

Grupăm la un loc studiul legăturilor punct la punct prin perechi de conductoare datorită multiplelor aspecte comune între toate tipurile de astfel de legături admise de standard.

Toate aceste variante de legături sunt gândite pentru a realiza un cablaj ieftin și fiabil în interiorul unei singure clădiri.

În toate cazurile, mediul de transmisie este format din două sau patru perechi de conductoare torsadate. Cu cablurile recomandate de standard, lungimea maximă a unei legături este de 100 m.

Condițiile de izolare electrică și de pământare nu permit utilizarea sigură pentru legături aeriene prin exteriorul clădirilor. Pentru astfel de scopuri standardul recomandă utilizarea fibrelor optice.

Descriem în continuare particularitățile tuturor standardelor privitoare la nivelul fizic construit pe perechi torsadate.

10 Base T . Este o legătură duplex la 10 Mbit/s utilizând două perechi de conductoare torsadate, câte o perechie pentru fiecare sens (4 conductoare în total). Denumirea standardului vine de la viteza de comunicație (10 Mbit/s), codificarea (în banda de bază) și tipul mediului (*Twisted pairs — perechi torsadate*).

Cablul de legătură constă, așa cum am văzut, din 4 conductoare izolate. Conductoarele sunt împerecheate 2 câte 2 (formând deci 2 perechi). În cadrul fiecărei perechi, conductoarele sunt răsucite unul în jurul celuilalt

pentru reducerea interferențelor cu câmpurile electromagnetice din jur. Caracteristicile electrice ale cablurilor, specificate prin standard, sunt în general îndeplinite de către tronsoanele de până la 100 m construite din cablurile folosite în mod curent pentru rețeaua telefonică și clasificate, în sistemul american de telefonie, *UTP Cat 3* (UTP de la *Unshielded Twisted Pairs* — *perechi torsadate neecranate*, iar *Cat 3*, de la *Category 3*).

Dăm în continuare, cu titlu informativ, câteva caracteristici:

- impedanța caracteristică: 100 Ω;
- atenuare: maxim 11,5 dB pentru tot tronsonul de cablu (de fapt acesta este parametrul care limitează lungimea unui tronson de cablu; dacă folosim un cablu cu atenuarea pe 200 m mai mică de 11,5 dB, putem cabla un tronson de 200 m cu astfel de cablu fără probleme);
- timpul de propagare al semnalului: maxim 1000 ns. Standardul cere, în plus, ca viteza de propagare să fie cel puțin $0,585 \cdot c$ (adică cel puțin de 0,585 de ori viteza luminii în vid).

Conectarea cablului la interfața de rețea sau la repetoare se realizează prin intermediul unui conector cu 8 pini, asemănător cu cel de telefon, standardizat sub numele RJ45.

Utilizarea pinilor este următoarea: emisia între pinii 1 și 2 și recepția între pinii 3 și 6. Pinii 4, 5, 7 și 8 sunt neutilizați.

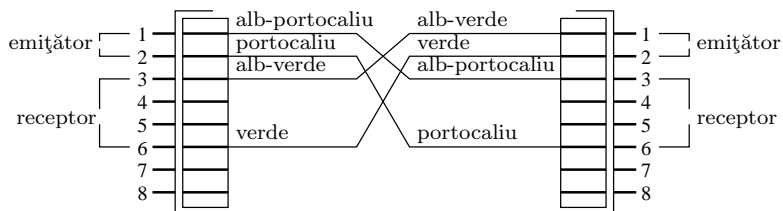
Conductoarele legate la emițător la un capăt trebuie legate la receptor la celălalt capăt (fig. 9.3). Acest lucru se poate realiza în două moduri:

1. Legarea cablului la conectoare se face „în X”: pinul 1 de pe un conector se leagă la pinul 3 de pe celălalt conector, 2 cu 6, 3 cu 1 și respectiv 6 cu 2, conform fig. 9.3(a). Un astfel de cablu se numește *cablu inversor* sau *cablu X*.
2. Cablul este „unu-la-unu” (adică pinul 1 de pe un conector este legat cu pinul 1 de pe celălalt conector, 2 cu 2, 3 cu 3 și 6 cu 6), iar inversarea se face în dispozitivul de la un capăt al cablului, prin legarea inversată a conectorului la circuite: pinii 1 și 2 la receptor și 3 și 6 la emițător, ca în fig. 9.3(b).

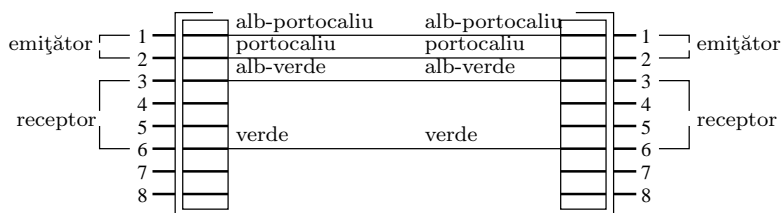
Standardul recomandă inversarea legăturilor în conectoarele repetoarelor și cere marcarea conectorilor cu inversare printr-un simbol „X”.

Conectarea a două echipamente prevăzute cu inversare în conector se face cu ajutorul unui cablu inversor, ca în figura 9.3(c).

Există și dispozitive care detectează automat pinii folosiți la emisie și recepție. Aceste dispozitive sunt desemnate *auto MDI/MDIX*. Dacă la unul



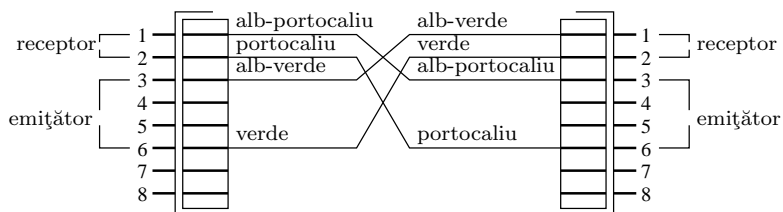
(a) Inversare realizată în cablul de legătură



conector
normal

conector
cu inversare

(b) Inversare realizată de unul dintre dispozitivele de la capete



conector
cu inversare

conector
cu inversare

(c) Conectarea a două echipamente prevăzute cu inversare în conector

Figura 9.3: Conectarea a două echipamente 10 Base T. Sunt date și culorile standard pentru izolațiile conductoarelor din cablu.

dintre capete se găsește un astfel de dispozitiv, se poate utiliza atât cablu unu-la-unu cât și cablu inversor, fără nici un fel de restricții. Mecanismul de detectare a pinilor utilizați se bazează pe pulsurile pentru verificarea mediului, descrise mai jos.

Transmiterea biților se face în codificare Manchester. Cele două nivele de tensiune, la emițător, sunt unul între 2,2 V și 2,8 V și celălalt între -2,2 V și -2,8 V.

Pe lângă transmiterea informației utile, standardul prevede emiterea periodică, de către fiecare echipament, a unui puls de testare a cablului. O interfață de rețea sau un repetor care nu primește periodic pulsuri de test de la celălalt capăt va „deduce“ că legătura nu este validă. Starea legăturii este semnalată printr-un led; de asemenea, plăcile de rețea semnalează starea legăturii printr-un bit de control ce poate fi citit de driver-ul plăcii de rețea.

O adăugire ulterioară la standard prevede ca în secvența de pulsuri de testare a cablului să se codifice disponibilitatea echipamentului ce le emite de a funcționa în regim duplex sau la o viteză mai mare de 10 Mbit/s (adică conform unuia din standardele descrise mai jos). Un echipament capabil de comunicație duplex și care este informat că echipamentul de la celălalt capăt este capabil de asemenea de comunicație duplex va intra automat în mod duplex.

Un echipament vechi, datând dinaintea acestei adăugiri la standard, va funcționa numai în regim semiduplex. Păstrarea compatibilității este asigurată de faptul că echipamentul vechi va înțelege pulsurile doar ca testarea liniei, iar pulsurile generate de el este puțin probabil să coincidă întâmplător cu pulsurile de negociere a modului de transmisie.

100 Base Tx. Este foarte asemănător cu 10 Base T, dar obține o viteză de transmisie de 100 Mbit/s.

Cablul constă tot din două perechi de conductoare torsadate, însă cu proprietăți mai bune de transmitere a semnalului (obține aceleași caracteristici de atenuare până la frecvențe de 10 ori mai mari). Cablurile utilizate sunt cele desemnate *UTP Cat 5*. Lungimea maximă a unui tronson este de 100 m.

Conectoarele și utilizarea pinilor sunt identice cu 10 Base T. Din acest motiv un cablu pentru 100 Base Tx poate fi întotdeauna utilizat la o legătură 10 Base T.

În general, echipamentele capabile să opereze conform standardului 100 Base Tx sunt capabile să lucreze și cu 10 Base T. Stabilirea vitezei se face printr-un mecanism similar cu cel utilizat la 10 Base T pentru negocierea modului semiduplex sau duplex. Trebuie însă spus că mecanismul de negociere

nu testează și calitatea cablului; din acest motiv, dacă legăm o placă de rețea de 100 Mbit/s la un hub sau switch de 100 Mbit/s printr-un cablu ce nu permite 100 Mbit/s (de exemplu *Cat 3* în loc de *Cat 5*), este necesar să configurăm manual viteza de 10 Mbit/s.

100 Base T4. Transmite 100 Mbit/s semi-duplex, utilizând cabluri *Cat 3*. Sunt necesare 4 perechi de conductoare (8 conductoare în total).

Câte o pereche de conductoare este rezervată pentru fiecare sens. Celelalte două perechi se utilizează în sensul în care are loc efectiv transmiterea informației (adică, întotdeauna trei perechi sunt utilizate pentru transmiterea informației și a patra este temporar nefolosită).

Codificarea informației este mai specială, utilizând 3 nivele de semnalizare în loc de obișnuitele 2 și transmițând simultan pe trei canale, pentru a obține un semnal ce se încadrează în banda de trecere a unui cablu *Cat 3*.

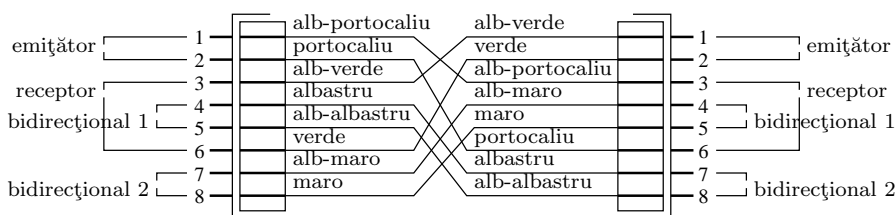


Figura 9.4: Utilizarea pinilor și conectarea între echipamente la 100 Base T4.

Conectoarele sunt tot RJ45, cu următoarea utilizare a pinilor: emisie: pinii 1 și 2; recepție: 3 și 6, bidirecțional 1: pinii 4 și 5; bidirecțional 2: pinii 7 și 8. Ca și la celelalte cabluri, descrise mai sus, este necesară o încrucișare, realizată fie în cablu, fie în conectorul unuia dintre echipamente. Standardul cere inversarea atât a emisie cu recepția cât și a celor două legături bidirecționale (fig. 9.4).

Întrucât în majoritatea instalațiilor sunt disponibile cabluri *Cat 5*, utilizarea standardului 100 Base T4 este extrem de rară.

100 Base T2. Transmite 100 Mbit/s duplex, utilizând două perechi *Cat 3*. Ca și la 100 Base T4, se utilizează o codificare complicată, însă obține performanțele lui 100 Base Tx pe cabluri identice cu cele folosite de 10 Base T.

Utilizarea lui este rară, datorită răspândirii cablului *Cat 5*.

1000 Base T. Transmite 1 Gbit/s duplex, utilizând 4 perechi *Cat 5*.

Legătura constă în patru perechi torsadate (8 conductoare) conforme *Cat 5*, de lungime maxim 100 m.

Se utilizează o schemă de codificare mai complicată, ce utilizează fiecare pereche de conductoare în regim duplex.

Conectoarele folosite sunt tot RJ45. Din rațiuni de compatibilitate, legăturile trebuie să realizeze aceeași inversare a unor perechi de fire ca și la 100 Base T4 (fig. 9.4).

Majoritatea plăcilor de rețea și celorlalte echipamente pentru rețele IEEE 802.3, produse recent și desemnate *Ethernet gigabit*, implementează standardele 10 Base T, 100 Base Tx și 1000 Base T.

1000 Base Cx. Transmite 1 Gbit/s duplex utilizând 2 perechi de conductoare de construcție specială.

Se utilizează câte o pereche pentru fiecare sens.

Standardul permite două tipuri de conectoare: conectoare trapezoidale cu 9 pini (identice cu cele utilizate pentru porturile seriale) sau niște conectoare cu 8 pini asemănătoare, dar incompatibile, cu RJ45.

Datorită incompatibilității cu 10 Base T și 100 Base Tx, puține echipamente utilizează 1000 Base Cx.

Realizarea practică a cablajelor

Cablurile *UTP Cat 5* folosite au de obicei 4 perechi de fire torsadate (8 fire în total), învelite toate într-o teacă protectoare. Doar 2 perechi (4 fire) sunt utilizate efectiv de legăturile 10 Base T și 100 Base Tx.

În cadrul fiecărei perechi, unul din fire are izolația într-o culoare plină iar celălalt este combinat, alb alternând cu culoarea firului pereche. Culoarele folosite pentru perechi sunt portocaliu, verde, albastru și maro. Menționăm că se comercializează, din păcate, și cabluri în care firele ce ar trebui să fie colorate cu alb plus o culoare sunt doar albe și, ca urmare, pentru a le identifica este necesar să se desfacă teaca protectoare pe o lungime suficient de mare pentru a vedea cum sunt torsadate firele (care cu care este împerecheat prin răsucire).

Schema de conectare standardizată este dată în tabela 9.1. Varianta „normal“ este utilizată la cablurile unu-la unu, precum și la unul din capetele cablurilor inversoare. Varianta „inversat“ este utilizată la celălalt capăt al cablurilor inversoare. Varianta „semi-inversat“ a fost utilizată frecvent pentru al doilea capăt al cablurilor inversoare, dar nu funcționează decât pentru rețele 10 Base T și 100 Base Tx, care nu utilizează deloc perechile albastru și maro. Pentru 1000 Base T, varianta „semi-inversat“ nu este prevăzută de standard; ca urmare unele echipamente funcționează cu astfel de cabluri, iar alte echipamente nu funcționează.

Nr. pin	Culoare		
	normal	inversat	semi-inversat
1	alb-portocaliu	alb-verde	alb-verde
2	portocaliu	verde	verde
3	alb-verde	alb-portocaliu	alb-portocaliu
4	albastru	alb-marou	albastru
5	alb-albastru	marou	alb-albastru
6	verde	portocaliu	portocaliu
7	alb-marou	albastru	alb-marou
8	marou	alb-albastru	marou

Tabelul 9.1: Atașarea conecătoarelor RJ45

Atragem atenția că este foarte important să se respecte schema de conectare din următoarele motive:

- Răsucirea firelor afectează transmiterea semnalului și sensibilitatea la parazii. Nerespectarea perechilor, adică utilizarea pentru un circuit a două fire care nu sunt împerecheate prin răsucire, duce la pierderi aleatoare de pachete, cu atât mai multe cu cât cablul este mai lung.
- Este necesar un efort inutil de mare pentru atașarea corectă a conectorului la al doilea capăt, dacă atașarea primului s-a făcut nestandard. Amatorii să se gândească la cazul când capătul cablat nestandard se găsește într-un dulap înghesuit, iar capătul unde trebuie atașat celălalt conector este câteva etaje mai sus sau mai jos. . .

Toate sistemele IEEE 802.3 ce utilizează perechi torsadate sunt proiectate pentru legături în interiorul unei singure clădiri. La cablurile trase prin exterior, descărcările electrice din atmosferă riscă să inducă în cablul de rețea tensiuni suficient de mari pentru a distruge plăcile de rețea sau hub-urile sau switch-urile atașate. Pentru legături exterioare se recomandă utilizarea fibrelor optice.

9.1.2. Legături prin fibre optice

IEEE 802.3 standardizează mai multe tipuri de legături prin fibre optice. Toate acestea sunt foarte similare din punctul de vedere al logicii funcționării; diferențele sunt aproape în totalitate aspecte minore legate de realizarea nivelului fizic.

Toate legăturile pe fibră optică sunt punct-la-punct (nu magistrală). Există totuși un echipament, numit *stea pasivă*, la care se pot conecta mai

multe plăci de rețea și care distribuie semnalul transmis de o placă spre toate celelalte. Astfel, o stea pasivă se comportă întrucâtva similar cu un cablu magistrală.

O legătură punct la punct constă din două fibre optice, câte una pentru fiecare sens; astfel, fiecare legătură este capabile de transmisie duplex.

10 Base F: standardizează transmisia prin fibră optică la un debit de transmisie de 10 Mbit/s. Rata erorilor, obținută cu o astfel de legătură, este în jur de 10^{-9} (1 bit eronat la 10^9 biți transmiși).

Grupează trei variante, cu diferențe foarte mici între ele (în general, echipamentele corespunzătoare pot fi interconectate fără probleme):

- 10 Base FP, destinat utilizării în configurații cu stea pasivă, ca atare funcționând în mod semi-duplex.
- 10 Base FB, destinat utilizării în conjuncție cu multe repetoare în cascadă și funcționând în mod semi-duplex.
- 10 Base FL, funcționând în mod duplex.

Se utilizează o fibră optică cu diametrul miezului de 62,5 μm (și diametrul exterior, al învelișului, de 125 μm), având o viteză de propagare de minim 0,67c, o atenuare de 3,75 dB/km (dacă atenuarea fibrei utilizate este mai mare, lungimea maximă a unei legături trebuie micșorată corespunzător) și o bandă de 160 MHzkm. Lungimea unui tronson de cablu este maxim 2 km (pentru 10 Base FP, lungimea fiecărei conexiuni dintre placa de rețea și steaua pasivă este de cel mult 1 km).

Conectarea cablului la echipamente se face cu ajutorul a două conec-toare (câte unul pentru fiecare fibră; reamintim că se utilizează o fibră pentru fiecare sens). Conec-toarele sunt descrise de standardul IEC 60874-10:1992 sub numele BFOC/2.5. Atenuarea introdusă de conector nu trebuie să depășească 1 dB, iar puterea undei reflectate nu trebuie să depășească -25 dB din puterea undei incidente.

Pentru semnalizare se folosesc unde infraroșii cu lungimea de undă cuprinsă între 800 nm și 910 nm. Nivelul semnalului emițătorului este între -15 dBm și -11 dBm pentru 10 Base FP și între -12 dBm și -11 dBm pentru 10 Base FB și 10 Base FL. Nivelul acceptabil pentru semnalul recepționat este între -41 dBm și -27 dBm pentru 10 Base FP și între $-32,5$ dBm și -12 dBm pentru 10 Base FB și 10 Base FL.

Semnalizarea utilizează codificarea Manchester, întocmai ca în cazul lui 10 Base T.

Spre deosebire de 10 Base T, nu se utilizează pulsuri de testare a legăturii care să permită negocierea modului semiduplex sau duplex sau a vitezei de transmisie; ca urmare, acești parametri trebuie configurați manual.

100 Base FX: oferă o viteză de transfer de 100 Mbit/s. Pe lângă viteza mai mare, 100 Base FX aduce câteva modificări față de 10 Base F, și anume utilizarea unor conectoare duble (conectând ambele fibre simultan) și un mecanism de negociere a vitezei de transfer.

1000 Base SX și 1000 Base LX. Aceste standarde oferă viteză de transfer de 1 Gbit/s.

Varianta 1000 Base SX transmite pe lungimea de undă de 850 nm, prin fibre cu diametrul miezului de 62,5 μm sau de 50 μm . Lungimea maximă de cablu între două echipamente este cuprinsă între 220 m pentru fibră cu dispersie intermodală de 160 MHzkm și 550 m pentru fibră cu dispersie intermodală de 500 MHzkm.

Varianta 1000 Base LX transmite pe lungimea de undă de 1310 nm, prin fibre multimod de 62,5 μm sau de 50 μm sau monomod de 10 μm . Lungimea maximă de cablu între două echipamente este de 550 m pentru fibra multimod și 5 km pentru fibra monomod.

9.1.3. Legături prin cablu magistrală

Există două variante de legături prin cablu magistrală standardizate prin IEEE 802.3; ambele variante realizează o viteză de transmisie de 10 Mbit/s.

Cele două variante sunt foarte asemănătoare, motiv pentru care le vom studia împreună.

Mediul de comunicație este un cablu format dintr-o pereche de conductoare coaxiale. Impedanța caracteristică a cablului este de 50 Ω (este deci incompatibil cu cablul utilizat pentru televiziune, care are impedanța de 75 Ω). Cablul nu are voie să aibă ramificații și trebuie încheiat la ambele capete prin terminatoare. Ramificațiile sau lipsa terminatoarelor duc la reflexia semnalului la ramificație sau la capătul fără terminator, rezultând bruierea semnalului de către reflexia lui.

Pe cablu se leagă, în paralel, interfețele de rețea și eventual repetoarele. Derivația pentru legătura la interfața de rețea este construită special pentru a reduce la minim reflexiile produse (impedanța emițătorului și receptorului este mult mai mare decât impedanța cablului, anume de cel puțin 100 k Ω); din acest motiv circuitele emițătorului și receptorului trebuie plasate la cel mult câțiva centimetri de cablu.

Semnalul este produs în codificare Manchester, cu durata unui bit de 100 ns; de aici viteza brută de transmisie de 10 Mbit/s.

Ca modificare față de codificarea Manchester clasică, peste semnal este suprapusă o componentă continuă, în scopul simplificării detectării coliziunilor. Pe cablu în repaus, tensiunea între conductoare este 0 V. Dacă o interfață emite date, apare o tensiune continuă între conductorul central și tresă. Dacă două sau mai multe interfețe emit simultan, tensiunea continuă crește peste un anumit prag la care se declară coliziune. La detectarea unei coliziuni, interfețele de rețea conectate la cablu opresc transmisia, conform metodei CSMA/CD (§ 4.2.2).

Există două sub-standarde privitoare la caracteristicile mecanice și electrice ale cablului de conectare: *10 Base 5* și *10 Base 2*.

10 Base 5, numit și *cablu galben* sau *cablu gros*, prevede utilizarea unui cablu coaxial având aproximativ 10 mm grosime totală, preferabil colorat în galben pentru o mai bună vizibilitate. Lungimea totală maximă a unui cablu este de 500 m. Standardul este gândit pentru cablare prin exteriorul clădirilor.

Denumirea sub-standardului vine de la viteza (10Mbit/s), codificarea (în banda de bază — *Base*) și lungimea maximă a cablului, în sute de metri.

Cu titlu informativ, dăm câteva detalii, specificate prin standard, cu privire la caracteristicile cablului:

- impedanța caracteristică: $50 \Omega \pm 2 \Omega$;
- viteza de propagare a semnalului: minim $0,77 \cdot c$;
- atenuarea: maxim 17 dB/km (8,5 dB pe tot tronsonul de cablu) la 10 MHz și maxim 12 dB/km (6 dB pe tot cablul) la 5 MHz;
- se acceptă maxim 100 interfețe de rețea pe un tronson de cablu.

Cablul trebuie conectat la pământ (la instalația de pământare a clădirii) *într-un singur punct*. Se specifică explicit prin standard că atât cablul cât și elementele legate de el trebuie să fie izolate față de pământ sau față de alte conductoare (cu excepția sus-menționatei unice legături de pământare). De asemenea, interfețele de rețea trebuie să realizeze o izolare electrică între cablul de rețea și circuitele calculatorului care să reziste la o tensiune de 1500 V. La efectuarea lucrărilor asupra rețelei, persoanele care lucrează trebuie să aibă grijă să nu atingă simultan cablul de rețea și un conductor legat la pământ, iar în cazul în care conectează sau deconectează două tronsoane de rețea să aibă grijă să nu închidă contactul electric între cele două tronsoane prin corpul lor. Toate aceste măsuri sunt luate deoarece este posibil să apară tensiuni electrice între legăturile de pământarea ale instalațiilor electrice în clădiri diferite. De

asemenea, este posibil ca într-un cablu, în special dacă este dus prin exterior, să se inducă, inductiv sau capacitiv, tensiuni parazite importante din cauza rețelelor de alimentare electrică din apropiere sau din cauza fulgerelor.

Circuitele electronice ale interfețelor de rețea sunt împărțite în două module: un modul, conținând emițătorul și receptorul propriu-zise, se atașează direct pe cablu; al doilea modul cuprinde logică de comandă și este construit sub forma unei plăci ce se introduce în calculator.

Atașarea modului de semnal la cablul de rețea se poate realiza în două moduri:

- prin conectarea celor două segmente de cablu de-o parte și de alta prin conectoare standardizate (conectoare coaxiale, numite *conectoare N*, cu prindere cu filet);
- prin realizarea unei *prize vampir*: se dă o gaură în cablu fără a-i intrerupe conductoarele, prin gaură se introduce o clemă ce va face contact cu firul central, iar legătura cu tresa se face printr-o altă clemă ce se strânge pe o zonă de pe care s-a îndepărtat mantaua exterioară a cablului.

Legătura dintre modulul emițător-receptor (engl. *transceiver*) și modulul de logică al plăcii de rețea sau al repetorului este de asemenea standardizată, sub numele de interfață *AUI*. Cablul de legătură dintre cele două module constă din 5 perechi de conductoare torsadate și ecranate individual, utilizează conectoare trapezoidale cu 15 pini și poate avea lungime maximă de aproximativ 50m.

10 Base 2 se mai numește *cablu subțire*, *cablu negru* sau *cablu BNC* (oarecum incorect, BNC fiind numele conectoarelor prevăzute a fi utilizate pentru acest tip de legătură). Este foarte asemănător cu 10 Base 5, însă folosește un cablu mai potrivit pentru cablaje în interior. Lungimea maximă a unui tronson este de 185 m.

Cablul este tot coaxial, dar este mai subțire (≈ 5 mm) pentru a putea fi îndoit mai ușor (standardul cere să poată fi îndoit la raza de 5 cm), în schimb este admis să aibă atenuare mai mare și, ca urmare, tronsoanele sunt limitate la lungime mai mică.

Dăm din nou câteva caracteristici ale cablului:

- viteza de propagare: $0,65 \cdot c$;
- atenuarea, pentru 185 m: maxim 8,5 dB la 10 MHz și maxim 6 dB la 5 MHz;
- maxim 30 interfețe atașate pe un tronson.

Conectarea interfețelor de rețea și a terminatoarelor se face prin conectori standardizați sub numele *BNC* (conectorii BNC sunt standardizați pentru aparatură electronică în general, nu se folosesc doar la rețele Ethernet), astfel (fig. 9.5): Fiecare bucată de cablu trebuie să aibă montate pe capete conectori BNC mamă. Există elemente numite *joncțiuni T* care conțin o ramificație și sunt prevăzute cu un conector BNC mamă (pe mijlocul T-ului) și două conectori BNC tată. La conectorii tată se atașează bucățile de cablu de-o parte și de alta, iar la conectorul mamă al T-ului se atașează conectorul tată de pe placa de rețea. Terminatoarele sunt prevăzute cu conector BNC mamă și se atașează pe T-urile de la plăcile de rețea extreme.

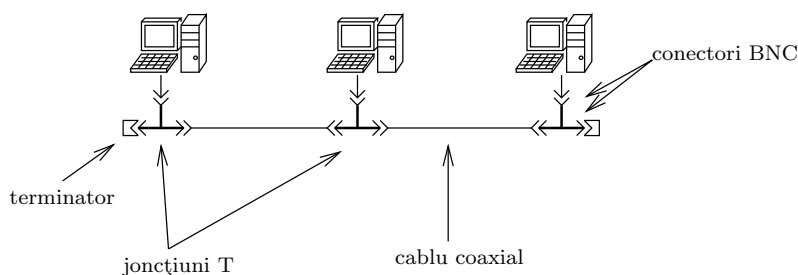


Figura 9.5: Conectarea unei rețele 10 Base 2

Pana cea mai frecventă ce apare la o rețea, afectând conexiunile directe, este întreruperea unui fir (de obicei o pană de contact între sârmă și conector sau între contactele unui conector). O întrerupere a cablului magistrală duce de regulă la oprirea funcționării întregii rețele, nu doar „ruperea” în două a rețelei. Aceasta se întâmplă deoarece capătul de cablu unde s-a produs întreruperea reflectă semnalul (este ca un cablu fără terminator) și, ca urmare, orice pachet emis pe acel cablu se ciocnește cu reflexia lui. Soluția cea mai eficientă de găsim a penei este o căutare binară prin izolarea — neapărat cu terminatoare — a unor porțiuni din ce în ce mai lungi din cablul rețelei.

9.1.4. Repetoarele și comutatoarele

Ca funcție în cadrul unei rețele, atât repetoarul cât și comutatorul este un dispozitiv la care sunt conectate mai multe cabluri de rețea și care, la primirea unui pachet pe un cablu, retransmite pachetul pe toate *celelalte* cabluri conectate. Interfața repetoarelor sau comutatorului către fiecare dintre cabluri se numește *port*.

Excepție făcând cazul comutatoarelor mai evoluate (vezi § 9.1.6.4), o rețea construită cu repetoare sau comutatoare trebuie să aibă o *topologie*

arborescentă, adică între orice două interfețe de rețea trebuie să existe *un drum și numai unul* format din cabluri directe și repetoare sau comutatoare.

Într-o rețea construită corect, arborescent, un pachet emis de o placă de rețea se propagă prin cabluri și repetoare sau comutatoare din ce în ce mai departe de interfața de origine, sfârșind prin a ajunge la toate plăcile din rețea.

În cazul în care rețeaua, în loc să fie arborescentă, conține circuite, se va întâmpla ca două copii ale aceluiași pachet să ajungă pe două căi distincte la un anumit repetoar sau comutator. Dacă este un repetoar, cele două copii, ajungând aproximativ simultan, vor produce o coliziune. Cum acest lucru se întâmplă cu orice pachet trimis în rețea, fiecare pachet va suferi o coliziune cu el însuși și va fi repetat la infinit cu același succes, rezultând astfel trafic util nul. În cazul comutatoarelor, dacă două copii ale unui pachet ajung pe două căi diferite la un comutator, acesta le va considera ca fiind pachete distincte. În consecință, le va memora și retransmite, fiecare copie fiind retransmisă inclusiv pe calea prin care a intrat cealaltă copie. În acest fel, copiile ciclează la infinit prin rețea, rezultând o „furtună de pachete“, adică o multiplicare incontrollabilă a pachetelor.

În cazul utilizării repetoarelor, pe lângă topologia în arbore mai trebuie respectate niște condiții, și anume:

- toate componentele legate la repetoare trebuie să lucreze la aceeași viteză (fie 10 Mbit/s, fie 100 Mbit/s, fie 1 Gbit/s). Aceasta deoarece un repetoar nu memorează pachetul de retransmis și, ca urmare, nu-l poate retransmite la altă cadență a biților decât cea cu care îl recepționează.
- să nu existe mai mult de 4 repetoare de-a lungul nici unui drum între două interfețe de rețea. Această restricție este impusă pentru ca diferențele de viteză de transmisie a repetoarelor și variația întârzierii introduse de repetoare să nu ducă la micșorarea sub o anumită limită a timpului dintre două pachete consecutive.
- întârzierea cea mai mare a transmisiei între două interfețe de rețea (întârzierea pe cablu plus întârzierea introdusă de repetoare) să nu fie mai mare decât jumătate din durata necesară emiterii unui pachet. Pentru o rețea de 10 Mbit/s, aceasta înseamnă o lungime maximă totală de 2500 m între oricare două interfețe de rețea.

În cazul switch-urilor, nu apare nici una din limitările expuse mai sus, cu excepția faptului că pe eventualele legături semi-duplex întârzierea trebuie să fie de cel mult jumătate din durata minimă a pachetului.

La rețele ce utilizează atât switch-uri cât și repetoare, restricțiile de la repetoare se aplică, separat, pe fiecare subrețea formată din repetoare

interconectate și legăturile acestora spre interfețele de rețea și switch-uri.

9.1.5. Dirijarea efectuată de comutatoare (switch-uri)

Comutatoarele (switch-urile) sunt capabile să realizeze o dirijare primitivă a pachetelor primite. Anume, un comutator ține o tabelă cu asocierea între adresa fizică (adresa MAC) a unei interfețe de rețea și *portul* (conectorul) la care este conectată, direct sau indirect, acea interfață.

Dacă un comutator primește un pachet cu o anumită adresă MAC sursă, comutatorul va asocia acea adresă cu portul prin care a intrat pachetul. Ulterior, dacă comutatorul primește un pachet având ca destinație acea adresă MAC, îl va trimite doar prin portul asociat acelei adrese.

Asocierea între adresa MAC și port este menținută doar un timp scurt (de ordinul secunde) pentru ca să se asigure actualizarea asocierii în cazul mutării interfeței de rețea de la un port la altul (prin mutarea fizică a cablului).

Mai multe adrese MAC pot avea asociat același port; această situație apare dacă la acel port este conectat un repetor sau un alt comutator. Unei adrese îi poate fi asociat cel mult un port.

La primirea unui pachet, comutatorul caută portul asociat adresei. Dacă există, va trimite pachetul doar prin acel port. Dacă nu există asociere, pachetul este trimis prin toate porturile cu excepția celui prin care a intrat. Evident, pachetele de broadcast se încadrează în această din urmă categorie.

9.1.6. Facilități avansate ale switch-urilor

9.1.6.1. Switch-uri configurabile

În mod obișnuit, switch-urile nu au parametri configurabili și nu au adresă; ele sunt transparente față de traficul ce trece prin ele.

Switch-urile mai avansate au parametri configurabili. Pentru configurare, este necesar să poată fi accesate de pe un calculator. Accesul se poate realiza în următoarele moduri:

- *Prin intermediul unui cablu serial:* La switch se conectează, prin intermediul unui cablu serial, un teleterminal (sau un calculator care execută un simulator de terminal, de genul *HyperTerm*). Switch-ul oferă o interfață text — oferă câteva comenzi de configurare. De cele mai multe ori, există o comandă *help* sau *?* care listează comenzile disponibile.
- *Prin telnet:* Switch-ul se prezintă ca și cum ar mai avea intern o interfață de rețea conectată la el însuși. Această interfață de rețea are o adresă MAC (adesea scrisă pe eticheta aplicată pe carcasă) și o adresă IP

configurabilă (adresa IP inițială se configurează prin intermediul conexiunii seriale descrise mai sus). Conectarea prin *telnet* la adresa IP a switch-ului (pe portul standard al protocolului telnet, anume 23) oferă acces la interfața de configurare prezentată mai sus. Evident, pentru împiedicarea configurării switch-ului de către persoane neautorizate, switch-ul permite configurarea unei parole, care este cerută la conectare.

- *Prin interfață web*: Ca și la conectarea prin telnet, switch-ul prezintă o adresă IP. Administratorul se poate conecta cu orice navigator web la această adresă și va primi pagini ce conțin parametrii actuali și formulare pentru modificarea parametrilor. Ca și în cazul configurării prin telnet, accesul poate și este recomandabil să fie restricționat prin parolă.

Pentru cazul uitării parolei, există o procedură de revenire la configurarea implicită. Aceasta constă de obicei în apăsarea unui buton de reset timp de 10–15 secunde sau punerea sub tensiune a switch-ului în timp ce se ține apăsat butonul de reset.

9.1.6.2. Filtrare pe bază de adrese MAC

Unele switch-uri pot fi configurate să nu accepte, pe un anumit port, decât pachete ce provin de la o anumită adresă MAC sau de la o adresă dintr-o anumită listă. De asemenea, un pachet destinat unei adrese MAC dintr-o astfel de listă nu va fi trimis decât prin portul pe a cărui listă se găsește adresa.

Această facilitate este introdusă pentru a împiedica eventuali intruși să intre în rețea racordându-se pur și simplu la prizele de rețea accesibile.

Deși îmbunătățește securitatea unei rețele, soluția are câteva limitări:

- lista adreselor asociabile unui port este limitată (de multe ori la 8 sau 16 adrese);
- multe plăci de rețea permit schimbarea (prin soft) a adresei MAC.

9.1.6.3. Trunking

Prin *trunking* se înțelege utilizarea mai multor cabluri în paralel ca legătură între două switch-uri. În acest fel, traficul ce se poate stabili între acele două switch-uri este suma capacităților legăturilor configurate în trunking.

Porturile utilizate în regim trunking trebuie configurate pe ambele switch-uri. Este de asemenea posibil ca legarea în trunking să utilizeze o extensie a protocolului IEEE 802.3 care este proprietatea firmei producătoare a switch-ului; în acest caz este posibil ca două switch-uri realizate de firme diferite să nu se poată lega în trunking.

9.1.6.4. Legături redundante

IEEE 802.1D [IEEE 802.1D, 2004] prevede un protocol pentru descoperirea și dezactivarea ciclurilor (în sensul teoriei grafelor) formate de legăturile dintre switch-uri.

Majoritatea switch-urilor nu implementează însă acest algoritm și, ca urmare, în majoritatea cazurilor existența ciclurilor duce la „furtuni de pachete“ (multiplicarea incontrollabilă a pachetelor în rețea).

Dacă toate switch-urile de pe traseul unui ciclu implementează protocolul de descoperire a ciclurilor, ele colaborează automat pentru dezactivarea uneia dintre legături și utilizarea doar a unui arbore parțial al grafului inițial al legăturilor. La căderea unei legături, switch-urile vor colabora pentru reactivarea unei legături dezactivate, în vederea păstrării conexității rețelei.

Menționăm că, în cazul existenței unui ciclu, nu este posibilă împărțirea traficului între drumurile alternative. Unul din drumuri va fi obligatoriu dezactivat complet, cât timp celălalt este funcțional.

9.1.6.5. Rețele virtuale (VLAN)

Mecanismul de *rețele virtuale* (*Virtual Local Area Network*) constă în împărțirea unei rețele fizice în mai multe rețele virtuale disjuncte. Fiecare rețea virtuală se comportă exact ca o rețea IEEE 802.3 independentă. Constructiv, rețelele virtuale partajează aceleași echipamente (comutatoare, cabluri sau chiar plăci de rețea).

A nu se confunda VLAN cu VPN (*Virtual Private Network* — *rețea privată virtuală* — descrisă în § 10.7.4).

Partiționarea în VLAN-uri poate fi dezirabilă din mai multe motive, cum ar fi: limitarea traficului de broadcast sau separarea traficului din motive de securitate.

Există două posibilități de construcție a VLAN-urilor.

O primă posibilitate constă în partiționarea porturilor unui switch. În acest fel, un switch se comportă ca mai multe switch-uri (virtuale) independente, fiecare având doar o parte a porturilor switch-ului fizic. Un port al unui switch poate să aparțină doar unui singur VLAN.

O a doua posibilitate este cea definită în [IEEE 802.1Q, 2003]. Fiecare VLAN constă dintr-o parte din echipamentele (interfețe de rețea, cabluri și switch-uri) rețelei fizice; VLAN-uri distincte pot partaja în voie echipamente fizice. Astfel, fiecare interfață de rețea aparține unuia sau mai multor VLAN-uri, fiecare cablu aparține unuia sau mai multor VLAN-uri și fiecare port al fiecărui switch aparține unuia sau mai multor VLAN-uri. Fiecare switch, la primirea unui pachet de broadcast sau pentru a cărui destinație nu are asociere,

va trimite pachetul prin toate porturile aparținând VLAN-ului pachetului, cu excepția portului prin care a intrat pachetul.

Pentru ca mecanismul descris mai sus să poată funcționa, este necesar ca, pe cablurile ce aparțin mai multor VLAN-uri, pentru fiecare pachet să se poată deduce cărui VLAN aparține. Pentru aceasta, fiecare pachet este etichetat cu un *identificator de VLAN (VLAN-ID)*; acest VLAN-ID este un număr reprezentabil pe 12 biți.

Pentru păstrarea compatibilității cu echipamentele ce nu suportă VLAN-uri 802.1Q, un segment de rețea care aparține doar unui singur VLAN poate fi configurat să utilizeze pachete neetichetate; switch-ul ce realizează legătura dintre un astfel de segment și restul rețelei fizice realizează adăugarea și eliminarea etichetei de VLAN pe pachetele ce tranzitează spre, respectiv dinspre, restul rețelei. Echipamentele incompatibile 802.1Q pot fi montate doar pe cabluri prin care trec pachete neetichetate.

O placă de rețea compatibilă 802.1Q poate fi configurată să facă parte din mai multe VLAN-uri. Pentru aceasta, ea se montează pe un cablu prin care trec pachete etichetate. Placa de rețea se comportă ca și când ar fi de fapt mai multe plăci de rețea, virtuale, câte una în fiecare VLAN. Fiecare placă virtuală are asociat un VLAN-ID, primește doar pachetele ce poartă acel VLAN-ID și marchează cu VLAN-ID-ul său toate pachetele emise.

Pe fiecare switch trebuie configurate porturile care aparțin fiecărui VLAN. De asemenea, pentru fiecare port trebuie stabilit dacă utilizează pachete etichetate (cu VLAN ID-ul) sau pachete neetichetate. Un port ce utilizează pachete neetichetate poate aparține unui singur VLAN.

9.1.7. Considerente privind proiectarea unei rețele

Proiectarea și construcția unei rețele Ethernet este în general extrem de ușoară; acesta este și unul din motivele popularității Ethernet-ului.

De obicei este necesar să se construiască o rețea în care toate calculatoarele să se „vadă” între ele (la nivel de rețea Ethernet; controlul accesului la diversele resurse oferite de sisteme se face prin soft, la nivel superior). Tot ce este necesar în acest caz este să se amplaseze un număr de switch-uri și cabluri astfel încât fiecare calculator să fie legat la un switch și switch-urile să fie legate între ele într-o rețea conexă și fără „bucle”.

Se utilizează de obicei o structurare ierarhică a legăturilor (așa-numita *cablare structurată*): de la calculatoarele dintr-o încăpere sau eventual 2-3 încăperi vecine se adună cablurile într-un switch, iar de la aceste switch-uri se adună cabluri către un switch central. Pentru rețelele mai mari, între switch-ul central și switch-urile asociate încăperilor se mai adaugă un nivel.

În instalațiile mici și fără pretenții, cablurile se duc aparent și se fixează de pereți sau pe mobilă numai acolo unde este strict necesar. Ușurința realizării și reconfigurării este plătită prin faptul că apar dificultăți la curățenie, cablurile se degradează ușor dacă se calcă pe ele și, în sfârșit, se mai întâmplă ca cineva să se împiedice de un cablu, rezultând echipamente trase pe jos sau cabluri smulse din conectoare.

Pentru evitarea neajunsurilor expuse mai sus, se preferă să se tragă cablurile prin paturi de cablu, tuburi îngropate (ca la instalațiile electrice) sau prin tavane false. Deoarece astfel de cablaje se modifică mai dificil, este bine să se aibă în vedere posibilele modificări ce ar putea fi de dorit în viitor. Asta înseamnă:

- Să se prevadă mai multe cabluri de la posibilele amplasamente de calculatoare la amplasamentul switch-ului asociat încăperii. Cablurile neutilizate nu e necesar să aibă toate loc în switch; se vor conecta sau deconecta după necesități.
- Să se prevadă 2–3 cabluri de la switch-urile corespunzătoare unei încăperi la switch-ul central. Astfel, dacă va fi nevoie să se construiască două rețele distincte, o parte din calculatoarele din încăperea conectându-se la o rețea și altele la altă rețea, se vor pune două switch-uri în încăperea, fiecare conectat prin câte un cablu la switch-ul central.

9.2. Rețele IEEE 802.11 (Wireless)

9.2.1. Arhitectura rețelei

Elementul de bază într-o rețea wireless [IEEE 802.11, 1999] este *celula wireless* (termenul original conform standardului este *Basic Service Set* — BSS). O celulă wireless este formată din mai multe stații (STA) situate într-o zonă geografică destul de restrânsă (de ordinul câtorva zeci de metri) pentru ca semnalul emis de fiecare stație să fie recepționat de toate stațiile din celulă.

Fiecare celulă are asociat un identificator de 48 de biți, unic, numit *Basic Service Set ID* — BSSID. Acest identificator este înscris în fiecare pachet de date vehiculat în rețea, astfel încât pentru orice pachet de date recepționat prin antenă se poate determina celula wireless căreia îi aparține. Mai multe celule wireless pot coexista în aceeași zonă, traficul din cadrul fiecărei celule putând fi distins pe baza BSSID-ului de traficul celorlalte celule.

Fiecare stație aparține (este asociată) la un anumit moment cel mult unei celule. Asocierile sunt dinamice — o stație poate să intre sau să iasă

oricând dintr-o celulă. Fiecare stație se identifică printr-o *adresă* unică de 48 de biți, numită în mod curent *adresa MAC* a stației.

Accesul la mediu este controlat în principal prin metode bazate pe urmărirea traficului pe mediu, detectarea coliziunilor și, într-o anumită măsură, metode de rezervare în prealabil a accesului la mediu. Acestea vor fi descrise în detaliu în § 9.2.2.

Prezența unei celule wireless organizate într-o anumită zonă este manifestată prin emiterea periodică de către una dintre stații a unui pachet special, numit *beacon*. Pe lângă BSSID-ul celulei, pachetele *beacon* mai conțin un șir de caractere numit SSID sau uneori *numele rețelei* (engl. *network name*). Acest șir este fixat de administratorul rețelei și servește la identificarea rețelei pentru utilizatorii umani.

O stație poate obține lista celulelor active în zona sa ascultând pachetele *beacon*. Lista afișată utilizatorului va conține SSID-urile rețelelor.

Există două moduri de lucru în care poate funcționa o rețea 802.11:

- Rețea formată dintr-o singură celulă independentă, neconectată prin mijloace IEEE 802 de alte echipamente. În terminologia standardului, o astfel de celulă se numește *Independent BSS* — IBSS; în mod curent rețeaua astfel formată se numește *ad-hoc*.
- Rețea formată din una sau mai multe celule, operând împreună și posibil conectate la o infrastructură IEEE 802 (de exemplu la o rețea Ethernet — 802.3). Un astfel de mod de lucru se numește *mod infrastructură* sau *managed*.

În mod infrastructură, în cadrul fiecărei celule există o stație care are rolul legării celulei la infrastructură (altfel spus, la restul rețelei IEEE 802.11). O astfel de stație poartă denumirea de *Access Point* — AP. Un AP este o stație, și ca atare are o adresă MAC. Într-o celulă a unei rețele de tip infrastructură, o stație ce intră sau iese dintr-o celulă trebuie să anunțe AP-ul responsabil de celula respectivă.

AP-ul este responsabil de generarea pachetelor *beacon* și BSSID-ul celulei este adresa MAC a AP-ului.

AP-urile unei aceleiași rețele 802.11 trebuie să fie interconectate, formând așa-numitul *Distribution System* (DS). DS-ul poate fi conectat la alte rețele din familia IEEE 802 prin intermediul unor dispozitive numite *portal*-uri. Celulele din aceeași rețea vor avea același SSID.

Standardul original nu prevede nimic în legătură cu modul de conectare a AP-urilor și deci de realizare a DS-ului. Ca urmare, fiecare fabricant de AP-uri și-a construit propriul protocol de comunicare inter-AP. Ulterior IEEE a

emis un standard, [IEEE 802.11F, 2003], care fixează un protocol de comunicare între AP-uri.

De obicei un dispozitiv vândut sub numele de *access point* conține un AP și un portal către rețele Ethernet. Un astfel de dispozitiv prezintă un modul radio prin intermediul căruia se comportă ca o stație cu rol de AP și un conector Ethernet. Într-o primă aproximație, un astfel de dispozitiv poate fi privit ca un *switch* conectat pe de o parte la fiecare dintre stațiile membre ale celulei și pe de altă parte la un dispozitiv Ethernet.

Unele *access point*-uri ce se găsesc în comerț oferă funcționalități suplimentare față de un AP combinat cu un portal. Aceste funcții sunt oferite prin extensii ale protocolului și ca urmare pot fi utilizate de regulă doar împreună cu echipamente produse de aceeași firmă. Funcționalitățile sunt:

- funcție de *switch* (punte) între o rețea Ethernet (fixă) și o celulă *wireless*, acționând însă ca și stație oarecare (nu AP). Această funcție se numește *wireless bridge* sau *AP client* (uneori există funcții cu ambele nume, cu diferențe minore între ele);
- funcție de AP, dar utilizând tot rețeaua *wireless* pentru partea de infrastructură. În acest mod, dispozitivul este în același timp AP pentru o celulă și stație oarecare în altă celulă, iar a doua legătură este utilizată pentru dirijarea spre rețeaua fixă a datelor din celula în care dispozitivul este AP.

9.2.2. Accesul la mediu

Deoarece într-o rețea 802.11 avem un mediu partajat între mai mulți emițători, este necesar să avem un mecanism de control al accesului la mediu. Metoda de control al accesului la mediu în IEEE 802.11 se numește *Carrier-Sense Multiple Access with Collision Avoidance* (CSMA/CA; rom: acces multiplu cu detectarea semnalului purtător și evitarea coliziunilor).

În principal, strategia de control al accesului la mediu se bazează pe detectarea coliziunilor și repetarea pachetelor ce au suferit coliziuni, adică aceeași strategie ca și pentru *Ethernet*-ul pe cablu coaxial.

Datorită condițiilor specifice rețelelor fără fir, sunt aduse câteva modificări. În principal, la transmisia radio nu există o delimitare comună între zonele de acțiune pentru diverse stații din aceeași celulă: este posibil ca o stație B să recepționeze bine transmisia stației A, stația C să recepționeze transmisia lui B, dar stația C să nu recepționeze transmisia lui A. Într-un astfel de caz, dacă A și C transmit simultan, pachetele emise se ciocnesc la B, dar deoarece nici

una din stațiile A și C nu recepționează transmisia celeilalte ele nu au cum să detecteze coliziunea.

În rețelele IEEE 802.11, o stație care dorește să trimită un pachet va trimite întâi un pachet de control, numit *Request To Send* (RTS; rom: *cerere de transmisie*), în care specifică destinatarul și durata de timp necesară transmiterii pachetului. Dacă destinatarul a primit pachetul RTS și este liber, va trimite înapoi un pachet de control *Clear To Send* (CTS; rom: *accept transmisie*). La primirea pachetului CTS, emițătorul trimite pachetul de date.

O stație care recepționează un pachet CTS destinat altei stații nu are voie să trimită nimic pe durata rezervată de pachetul CTS, pentru a nu interfera cu transmisia acceptată prin acel CTS. Această restricție trebuie respectată și în cazul recepției unui pachet CTS destinat altei rețele din aceeași zonă (adică purtând un BSS-ID diferit).

Utilizarea pachetelor RTS și CTS nu este obligatorie. Pentru pachetele mici este preferabilă trimiterea direct a pachetului de date și repetarea acestuia în cazul unei coliziuni. Pentru pachetele de *broadcast*, utilizarea RTS și CTS este imposibilă; ca urmare un pachet de broadcast este trimis direct.

9.2.3. Generarea pachetelor *beacon*

În modul infrastructură, pachetele *beacon* ale unei celule sunt generate exclusiv de către AP-ul celulei.

În modul ad-hoc, generarea pachetelor *beacon* este făcută distribuit, de către toate stațiile membre ale celulei IBSS. Simplificat, o stație care nu recepționează un *beacon* într-un anumit interval de timp predefinit generează ea însăși pachetul *beacon*.

9.2.4. Securitatea rețelelor 802.11

Deoarece la rețelele 802.11 comunicația este prin unde radio, a căror domeniu de acțiune nu poate fi net limitat, utilizarea unor metode care să asigure confidențialitatea și integritatea datelor transportate este esențială.

Există mai multe mecanisme de securitate ce pot fi utilizate. În cadrul unei celule se poate utiliza, la alegere, unul singur dintre acestea:

- *Open system*: înseamnă, de fapt, lipsa oricărui mecanism de securitate. Se utilizează acolo unde se dorește să se ofere acces public la Internet. De remarcat însă că, datorită lipsei oricărui mecanism de confidențialitate sau asigurarea integrității mesajelor, oricine poate asculta sau modifica comunicația oricui în cadrul celulei.
- *Wired Equivalent Privacy* — *WEP* (rom. *securitate echivalentă cu rețeaua cablată*): oferă confidențialitate și autentificarea și verificarea integrității

mesajelor. În acest scop, toți membrii celulei trebuie să cunoască o anumită cheie de lungă durată, numită *pre-shared key* (rom. *cheie partajată în prealabil*); această cheie trebuie dată de utilizator la inițierea celulei sau, după caz, la introducerea stației în celulă. Criptarea se face utilizând cifrul RC4, cu o cheie construită din secretul partajat și dintr-un *vector de inițializare* ales aleator, pentru fiecare pachet, de către emițător și transmis în antetul pachetului. Controlul integrității pachetului este făcut tot pe baza secretului partajat. WEP are două slăbiciuni: pe de o parte, datorită existenței unei slăbiciuni a cifrului RC4 (există câteva chei slabe, foarte ușor de spart), WEP poate fi spart destul de ușor; pe de altă parte, modelul de securitate oferit este destul de neflexibil.

- *WiFi Protected Access* — *WPA*: corectează problemele WEP, păstrând compatibilitatea cu plăcile de rețea existente. În privința criptării, WPA păstrează cifrul RC4 din motive de compatibilitate, dar vine cu o schemă diferită de gestiune a cheilor de criptare, capabilă să evite cheile slabe.

În privința obținerii unui model de securitate mai flexibil, WPA are două moduri de lucru:

- *WPA-Personal*, numit și *WPA-PSK* (de la *Pre-Shared Key*), în care se utilizează un secret partajat între toți membrii celulei, fiind similar cu WEP (dar mult mai sigur).
 - *WPA-Enterprise*, în care cheile se obțin pe baza unor chei individuale ale utilizatorilor. Controlul accesului și obținerea cheilor se face printr-un mecanism numit *Extensible Authentication Protocol (EAP)*, descris mai jos.
- *IEEE 802.11i* [IEEE 802.11i, 2004], numit și *WPA2*, extinde WPA adăugând, între altele, posibilitatea utilizării cifrului AES. Ca și în cazul WPA, există două moduri de lucru, cu cheie partajată în prealabil sau utilizând *EAP*.

Protocolul de autentificare extensibil, *EAP* [RFC 3748, 2004], este un protocol generic, ce permite utilizarea mai multor scheme de autentificare. *EAP* este utilizat și de alte protocoale în afară de WPA și WPA2, și anume poate fi utilizat în cadrul legăturilor PPP [RFC 1661, 1994], precum și pentru autentificarea conectărilor la o rețea cablată IEEE 802.3, conform [IEEE 802.1X, 2001].

Arhitectura *EAP* conține următoarele componente:

- *clientul* ce trebuie să-și dovedească identitatea în scopul obținerii accesului

la rețea. Rolul clientului îl are placa de rețea 802.11 (sau placa de rețea 802.3 sau clientul PPP). În terminologia *EAP*, acesta este numit *supplicant*.

- *punctul de acces* este entitatea care trebuie să autentifice clientul pentru a-i oferi acces la serviciile rețelei. Rolul de punct de acces îl are AP-ul 802.11 (sau switch-ul 802.3 sau serverul PPP). În terminologia *EAP*, acesta se numește *authenticator*.
- *serverul de autentificare* este entitatea care deține baza de date cu cheile clienților și realizează efectiv autentificarea.

Protocolul *EAP* prevede un schimb de mesaje între client și serverul de autentificare. Dacă serverul de autentificare este distinct față de punctul de acces, comunicația dintre client și serverul de autentificare trece prin punctul de acces, iar porțiunea din calea de comunicație dintre punctul de acces și serverul de autentificare este protejată criptografic pe baza unui secret partajat între punctul de acces și serverul de autentificare. Serverul de autentificare este de obicei un server RADIUS.

Unele dintre mecanismele efective de autentificare utilizabile în cadrul *EAP* sunt:

- *EAP-MD5* prevede că serverul de autentificare trimite clientului un număr aleator, iar clientul răspunde cu dispersia MD5 a concatenării numărului aleator cu parola clientului. Funcționarea mecanismului necesită ca serverul să aibă în baza de date, în clar, parola clientului. *EAP-MD5* permite doar autentificarea clientului, nu și stabilirea unor chei pentru criptarea sau autentificarea mesajelor.
- *EAP-TLS* necesită ca atât clientul cât și serverul de autentificare să aibă prestabilite chei secrete SSL/TLS, iar fiecare dintre ei să aibă certificatul TLS al celuilalt (vezi și § 11.3.2.5). Se stabilește o conexiune TLS între client și serverul de autentificare, utilizând certificatele acestora, iar în cadrul acestei conexiuni stabilesc cheile pentru comunicația ulterioară între client și punctul de acces.
- *PEAP* (de la *Protected EAP*) prevede utilizarea TLS pentru deschiderea unei conexiuni securizate între client și serverul de autentificare, însă doar serverul are o cheie TLS, clientul autentificând serverul pe baza certificatului corespunzător. După deschiderea conexiunii TLS, urmează autentificarea clientului de către server, iar în caz de succes are loc negocierea cheilor pentru securizarea comunicației între client și punctul de acces. În terminologia *PEAP*, conexiunea *TLS* se numește *mecanismul exterior de autentificare*, iar mecanismul de autentificare a clientului

se numeşte *mecanismul interior*. Mecanismul interior cel mai răspândit este *MSCHAP*, care este un mecanism similar cu *EAP-MD5*.

Capitolul 10

Internetul

Denumirea *Internet* desemnează două lucruri: pe de o parte un protocol de nivel rețea (*Internet Protocol, IP, protocolul Internet*), iar pe de altă parte rețeaua Internet, care este o rețea la scară mondială bazată pe protocolul Internet.

Capitolul de față prezintă:

- protocolul Internet (IP), împreună cu celelalte protocoale de bază ale rețelelor de tip Internet (TCP, DNS, ARP, etc.);
- câteva aspecte administrative legate de rețeaua mondială Internet.

10.1. Arhitectura rețelei

Facem în continuare o scurtă trecere în revistă a conceptelor de bază ale unei rețele bazate pe protocolul Internet. Aceste concepte vor fi detaliate în paragrafele care urmează.

Serviciul de comunicație oferit de o rețea Internet este de tip *data-gram*; în terminologia Internet acestea se numesc *pachete*.

Ca orice rețea (vezi capitolul 5), o rețea Internet este alcătuită din *noduri*, interconectate între ele. Într-o rețea Internet, toate nodurile pot acționa ca noduri finale (adică să fie sursă sau destinație pentru comunicație). Sunt numite *stații* (engl. *hosts*) nodurile ce nu pot acționa ca noduri intermediare și *rutere* nodurile ce pot acționa ca noduri intermediare.

Stațiile sunt în mod uzual calculatoare (PC-uri, mainframe-uri), dispozitive mobile (PDA-uri), imprimantele de rețea sau alte dispozitive. Remarcăm că switch-urile Ethernet sunt noduri IP numai dacă sunt configurabile. În acest caz, ele au doar rol de stație și doar în scopul de-a putea fi contactate în vederea configurării.

Nodurile intermediare sunt fie PC-uri, fie dispozitive dedicate (*rutere* dedicate).

Legăturile directe pot fi realizate prin linii seriale, linii telefonice cu modemuri, rețele locale IEEE 802, cablu TV, etc. Modul de utilizare a fiecărui tip de legătură directă de către o rețea Internet este standardizat prin standarde auxiliare (§ 10.5). Există chiar un standard [RFC 1149, 1990] de utilizare ca legături directe a porumbeilor călători; deși standardul a fost publicat ca o glumă de 1 aprilie, el ilustrează foarte bine independența între nivele într-o rețea.

Din punctul de vedere al unei rețele Internet, o legătură directă este orice fel de canal de comunicație pe care rețeaua de tip Internet o poate folosi.

Fiecare nod este identificat prin una sau mai multe *adrese IP*. Cu excepția unor adrese cu rol special, o adresă IP identifică unic un nod. Unele noduri, în special cele intermediare, au mai multe adrese IP asociate.

Adresele IP sunt arareori folosite direct de utilizatorii umani. În locul lor se utilizează *numele de domeniu*. Corespondența între un nume de domeniu și adresa IP se realizează cu ajutorul sistemului DNS (*Domain Name Service*), descris în § 10.4.

Protocolul Internet a fost proiectat pentru a asigura o toleranță deosebit de mare la pene. După căderea unor noduri sau a unor legături, dacă mai există totuși un drum între două noduri el va fi găsit și utilizat în cele din urmă. Această toleranță la pene vine cu un preț: nu există garanții cu privire la întârzierea maximă în livrarea unui pachet sau debit minim garantat; ba chiar este posibil ca un pachet să fie pierdut complet (acest lucru se poate întâmpla cu pachetele surprinse pe drum de o pană, precum și în caz de încărcare mare a rețelei), să ajungă în dublu exemplar sau două pachete să ajungă la destinație în ordine inversă a trimiterii.

Este sarcina nivelelor superioare să se descurce în aceste condiții. În acest scop, între aplicație și nivelul rețea este plasat un nivel intermediar, *nivelul transport*, cu rolul de-a furniza aplicației un serviciu mai potrivit.

10.2. Protocolul IP

Protocolul Internet (engl. *Internet Protocol — IP*) descrie formatul pachetelor și câteva aspecte privind activitatea nodurilor rețelei.

Protocolul IP are două versiuni aflate curent în uz: versiunea 4 (cea mai utilizată în prezent, numită prescurtat *IPv4*) standardizată prin [RFC 791, 1981] și versiunea 6 (care se răspândește relativ încet, numită prescurtat *IPv6*) standardizată prin [RFC 2460, 1998].

10.2.1. Structura pachetului IP

Un pachet IP este alcătuit dintr-un antet fix, un număr variabil de opțiuni și, în final, datele utile.

Antetul fix conține datele necesare pentru dirijarea pachetului. Conținutul antetului fix este dat în tabelele 10.1 (pentru versiunea 4) și 10.2 (pentru versiunea 6). Semnificația diferitelor câmpuri va fi descrisă în paragrafele care urmează.

Nume câmp	Lungime (biți)	Rol
Versiune	4	Versiunea protocolului; valoarea este fixă: 4.
IHL	4	Lungimea antetului, inclusiv opțiunile, în grupuri de 32 biți (valoarea minimă este 5, adică 160 biți).
TOS	8	Tip serviciu (vezi § 10.2.6.2).
Lungime totală	16	Lungimea totală, antet plus date utile, în octeți.
Identificare	16	Identificator pentru reasamblarea fragmentelor (vezi § 10.2.6.1).
Rezervat	1	Rezervat pentru extinderi ulterioare; are valoarea 0.
Nu fragmenta	1	vezi § 10.2.6.1.
Ultimul fragment	1	Marchează ultimul fragment sau un pachet nefragmentat (vezi § 10.2.6.1).
Deplasament	13	Deplasament pentru reasamblarea fragmentelor.
Timp de viață	8	Timpul rămas până la distrugerea pachetului (vezi § 10.2.5.3).
Protocol	8	Identificarea protocolului de nivel superior căruia îi aparțin datele utile.
Suma de control	16	Suma de control a antetului.
Adresă sursă	32	Adresa nodului ce a creat pachetul.
Adresă destinație	32	Adresa destinatarului final al pachetului.

Tabelul 10.1: Antetul IP versiunea 4

Opțiunile sunt informații pentru dirijarea pachetului pentru cazuri mai speciale; deoarece aceste informații nu sunt necesare decât pentru anumite tipuri de pachete, ele sunt prezente doar în pachetele în care este nevoie de

Nume câmp	Lungime (biți)	Rol
Versiune	4	Versiunea protocolului IP. Valoarea este fixă: 6.
Clasă trafic	8	tip serviciu (vezi § 10.2.6.2).
Etichetă flux	20	vezi § 10.3.1.8.
Lungime rest	16	Lungimea pachetului minus antetul fix, în octeți.
Tip antet următor	8	Dacă există opțiuni, tipul primului antet opțional; altfel, protocolul căruia îi aparțin datele utile.
Limită salturi	8	Numărul maxim de salturi până la distrugerea pachetului (vezi § 10.2.5.3).
Adresa sursă	128	Adresa nodului ce a emis pachetul.
Adresa destinație	128	Adresa destinatarului final al pachetului.

Tabelul 10.2: Antetul IP versiunea 6

ele.

Datele utile sunt un șir de octeți asupra căruia protocolul IP nu impune nici o restricție, cu excepția lungimii. Lungimea maximă admisă de protocol este de 65515 octeți (65535 octeți pachetul întreg) pentru IPv4 și 65535 octeți, inclusiv antetele opționale, pentru IPv6. Este permis ca unele noduri să nu poată procesa pachete în care datele utile sunt mai lungi de 556 octeți (576 octeți tot pachetul) pentru IPv4 și 1240 octeți (1280 octeți tot pachetul) pentru IPv6 (a se vedea și § 10.2.6.1).

10.2.2. Bazele dirijării pachetelor IP

10.2.2.1. Subrețele și interfețe

O *subrețea* este o mulțime de noduri legate direct fiecare cu fiecare. De exemplu, o rețea Ethernet construită cu cabluri magistrală este o subrețea IP. O rețea Ethernet cu hub-uri sau switch-uri este de asemenea o subrețea IP întrucât, din punctul de vedere al calculatorului la care este atașată o placă de rețea, o rețea Ethernet construită cu cablu magistrală se comportă identic cu o rețea construită cu hub-uri sau switch-uri. Ca alt exemplu, o linie serială construiește o subrețea cu două calculatoare.

Interfața de rețea este un concept abstract care desemnează legătura dintre un nod și o subrețea. În cazul în care legătura directă este realizată de

o rețea IEEE 802, interfața de rețea este placa de rețea împreună cu driver-ul ei.

Fiecare interfață de rețea are propria adresă IP. Ca urmare, un nod ce are n plăci de rețea va avea n adrese IP distincte.

Are sens să vorbim despre *interfețele membre ale unei subrețele*, ca fiind interfețele prin care nodurile din subrețea sunt conectate la acea subrețea. Adresele IP dintr-o subrețea sunt adresele IP ale interfețelor din acea subrețea.

10.2.2.2. Prefixul de rețea

Fiecare subrețea are asociat un *prefix de rețea*, adică un anumit șir de biți de lungime mai mică decât lungimea unei adrese IP. Toate adresele IP ale interfețelor din acea subrețea trebuie să înceapă cu acel prefix de rețea. Prefixul unei subrețele nu este permis să fie prefix al unei adrese IP din afara acelei subrețele. Ca urmare, un prefix identifică unic o subrețea.

Sufixul unei adrese, adică șirul de biți din adresă care nu fac parte din prefixul subrețelei, îl vom numi *adresa în cadrul subrețelei*. Numărul de biți ai sufixului determină numărul de noduri ce pot fi membre ale subrețelei.

Adresele în care sufixul este format numai din biți 0 sau numai din biți 1 (așadar două adrese pentru fiecare subrețea) sunt rezervate și nu pot fi asignate nodurilor rețelei (a se vedea și [RFC 1700, 1994]).

EXEMPLUL 10.1: Pentru simplificarea exemplului vom presupune că adresele IP sunt doar de 8 biți. Presupunem că o subrețea ar avea prefixul de rețea 10110. Adresa 10110010, dacă există, trebuie să desemneze o interfață din acea rețea.

Adresa 10111010 nu poate face parte din acea subrețea, deoarece nu începe cu prefixul subrețelei. Notăm că un nod care are o interfață în subrețeaua 10110 și o interfață în altă rețea ar putea avea adresa 10111010 pe cea de-a doua interfață.

Din subrețeaua considerată, cu prefixul 10110, pot face parte adresele, în număr de $2^3 = 8$, din intervalul 10110000–10110111. Adresele 10110000 și 10110111 sunt rezervate; rămân deci 6 adrese ce pot fi asignate nodurilor subrețelei.

Un exemplu de asignare a adreselor este prezentat în figura 10.1. Pătrățelele numerotate reprezintă calculatoarele, iar liniile reprezintă legăturile directe, figurate aici ca și când ar fi realizate prin cabluri magistrală. De remarcat că nodul cu numărul 3 are două adrese, 10110001 și 10111010, câte una pentru fiecare interfață.

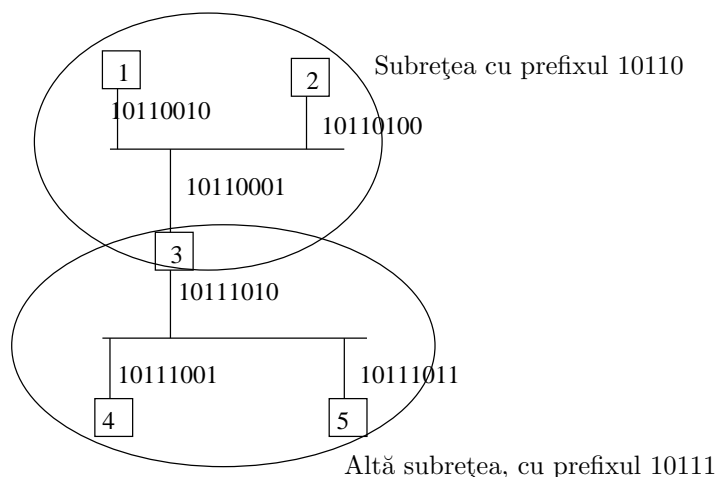


Figura 10.1: O rețea formată din două subrețele. Vezi exemplul 10.1

10.2.2.3. Tabela de dirijare

La primirea unui pachet IP, un nod execută următorul algoritm:

1. dacă adresa destinație este una din adresele nodului curent, pachetul este livrat local (nivelului superior);
2. altfel, dacă adresa destinație este adresa unui vecin direct, pachetul este livrat direct aceluia vecin;
3. altfel, se determină vecinul direct cel mai apropiat de destinatarul pachetului și i se dă pachetul, urmând ca acesta să-l trimită mai departe.

Pentru pasul 2, este necesar ca nodul să determine dacă adresa destinație corespunde unui vecin direct și care este interfața prin care se realizează legătura. Livrarea efectivă este realizată de interfața de rețea; acesteia i se dă pachetul și adresa IP a vecinului.

Pentru pasul 3, trebuie determinat în primul rând vecinul direct căruia i se va trimite pachetul. Dacă acesta are mai multe interfețe, trebuie utilizată interfața prin intermediul căruia el este vecin nodului curent. O dată determinată adresa interfeței, trimiterea pachetului se face ca la pasul 2.

Deciziile de la pașii 2 și 3 se iau pe baza *tabelei de dirijare* a nodului curent. O tabelă de dirijare constă dintr-o mulțime de *reguli de dirijare*. Fiecare regulă asociază o *țintă* unui *grup de adrese* destinație.

Grupul de adrese este specificat printr-un prefix de rețea. Pentru un pachet dat se aplică acea regulă de dirijare în care prefixul ce specifică grupul este prefix al adresei destinație a pachetului. Dacă există mai multe astfel de

reguli de dirijare, se aplică regula cu prefixul cel mai lung — adică cea mai specifică dintre regulile de dirijare aplicabile.

Ținta poate fi fie o interfață, fie o adresă IP.

Dacă ținta este o interfață, destinația trebuie să fie un vecin direct, accesibil prin acea interfață; în acest caz pachetul de dirijat este livrat direct destinatarului prin interfața dată în regulă, conform pasului 2.

Dacă ținta este o adresă, aceasta trebuie să fie adresa unei interfețe vecine. În acest caz pachetul de dirijat este trimis nodului vecin a cărui adresă este specificată în tabela de dirijare. Nodul vecin respectiv poartă denumirea de *gateway* și trebuie să fie configurat să acționeze ca nod intermediar.

De notat că adresa sursă și adresa destinație din antetul IP nu se modifică în cursul acestei proceduri. Sursa rămâne nodul care a emis pachetul, iar destinația rămâne nodul căruia trebuie să-i fie livrat în cele din urmă pachetul. Atunci când modulul de rețea pasează unei interfețe de rețea un pachet în vederea transmiterii pachetului către un nod vecin, modulul de rețea va comunica interfeței două lucruri: pachetul, în care adresa destinație reprezintă destinatarul final, și adresa vecinului direct căruia interfața îi va livra pachetul. Acesta din urmă poate fi diferit față de destinatarul final dacă este doar un intermediar pe drumul către destinatarul final.

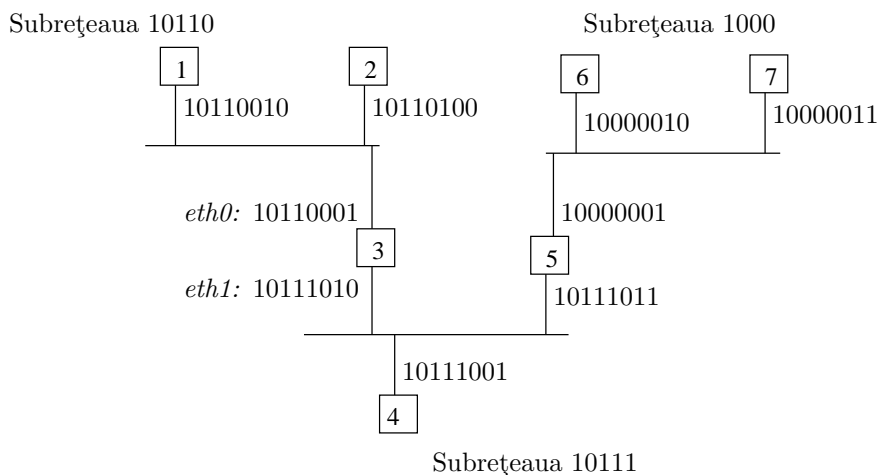


Figura 10.2: O rețea pentru exemplul 10.2

EXEMPLUL 10.2: Fie rețeaua din figura 10.2, formată din trei subrețele. Pentru nodul nr. 3, au fost figurate și numele interfețelor de rețea: *eth0* către subrețea de sus și *eth1* către subrețea de jos. Tabela de dirijare a nodului 3 este cea ilustrată în tabelul 10.3.

Nr. crt.	Grup de adrese (prefix)	Ținta
1.	10110	interfața <i>eth0</i>
2.	10111	interfața <i>eth1</i>
3.	1000	nodul 10111011

Tabelul 10.3: Tabela de dirijare pentru nodul 3 din figura 10.2 (exemplul 10.2).

Considerăm că nodul 3 primește un pachet cu destinația 10110010 (nodul 1). Singura regulă aplicabilă este regula 1, deoarece 10110 este prefix pentru 10110010. Conform acestei reguli, pachetul poate fi livrat direct prin interfața *eth0*.

Fie acum un pachet cu destinația 10000010 (nodul 6). Regula aplicabilă este regula 3. Conform acesteia, pachetul trebuie trimis nodului cu adresa 1011101, urmând ca acesta să-l trimită mai departe. Modulul IP caută în continuare regula aplicabilă pentru destinația 10111011 și găsește regula 2, conform căreia pachetul se trimite prin interfața *eth1*. Prin urmare, pachetul destinat lui 10000010 va fi trimis lui 10111011 prin interfața *eth1* (urmând ca nodul 5 să-l livreze mai departe nodului 6).

De remarcat că nodurile ce apar ca țintă în regulile tabelelor de dirijare trebuie specificate prin adresele interfețelor accesibile direct din nodul curent. În exemplul 10.2, este esențial ca, în ultima regulă a tabelii de dirijare a nodului 3, nodul 5 să fie specificat prin adresa 10111011 și nu prin adresa 10000001. Nodul cu adresa 10111011 este accesibil prin interfața *eth1*, conform regulii 2; dacă ar fi fost specificat prin adresa 10000001, nodul 3 nu ar fi putut determina cum să-i trimită pachetul.

În majoritatea cazurilor, tabela de dirijare are o regulă numită *implicită*, corespunzătoare prefixului vid și, ca urmare, aplicată pentru pachetele pentru care nu este aplicabilă nici o altă regulă. Această regulă, dacă există, are totdeauna ca țintă o adresă IP a unui nod vecin al nodului curent. Acest nod (de fapt, această adresă IP) poartă denumirea de *default gateway*.

10.2.3. Scrierea ca text a adreselor și prefixelor

10.2.3.1. Scrierea adreselor IP

Atunci când o adresă IP este scrisă pe hârtie sau într-un fișier text, se afișează pe ecran sau se citește de la tastatură, adresa este scrisă într-un format standard descris în continuare.

Adresele IP versiunea 4, care sunt șiruri de 32 de biți, se scriu ca șir de 4 numere, scrise în baza 10, separate prin puncte. Fiecare număr este de fapt valoarea câte unui grup de 8 biți, văzut ca număr. Această scriere se numește *notație zecimală cu punct*.

Pe lângă notația zecimală cu punct, adresele IP versiunea 4 pot fi scrise în *notația zecimală simplă*: se scrie direct valoarea adresei ca număr, scris în baza 10.

EXEMPLUL 10.3: Fie adresa 1100-0000-1010-1000-0000-0000-0010-0010 (liniuțele au fost scrise numai pentru ușurarea citirii). Notația zecimală cu punct este 192.168.0.34. Notația zecimală simplă este 3232235554.

Adresele IP versiunea 6 sunt șiruri de 128 de biți. Scrierea lor obișnuită se face ca un șir de 32 cifre hexa, fiecare reprezentând câte 4 biți din adresă. Cifrele hexa sunt grupate câte 4, iar grupurile succesive sunt separate prin câte un caracter *două puncte*. Pentru a scurta scrierea, se permit următoarele optimizări:

- zerourile de la începutul unui grup pot să nu fie scrise;
- un grup cu valoarea 0 sau mai multe astfel de grupuri consecutive se pot elimina, împreună cu separatorii *două puncte* dintre ei, rămânând doar două caractere *două puncte* succesive. Acest lucru se poate face într-un singur loc al adresei, altfel s-ar crea evident o ambiguitate.

EXEMPLUL 10.4: O posibilă adresă IPv6 este

```
fe80:0000:0000:0000:0213:8fff:fe4e:fbf4
```

Posibile scrieri prescurtate sunt

```
fe80:0:0:0:213:8fff:fe4e:fbf4
```

sau

```
fe80::213:8fff:fe4e:fbf4
```

Adresa 0:0:0:0:0:0:0:1 se scrie compact ::1.

Pentru adrese IPv6 alocate în vederea compatibilității cu IPv4, este acceptată scrierea în care primii 96 biți sunt scriși în format IPv6, iar ultimii 32 de biți sunt scriși în format IPv4, separați de primii printr-un caracter *două puncte*.

EXEMPLUL 10.5: Adresa 0:0:0:0:0:0:c100:e122 se poate scrie și
0:0:0:0:0:0:193.0.225.34
sau, mai compact,
::193.0.225.34

10.2.3.2. Scrierea prefixelor de rețea

Prefixele de rețea fiind de lungime variabilă, trebuie precizată atât valoarea efectivă a prefixului cât și lungimea acestuia. Există două notații: notația cu adresa subrețelei și lungimea prefixului și notația cu adresa subrețelei și masca de rețea.

În notația cu adresă și lungime, prefixul se completează cu zerouri la lungimea unei adrese IP (adică la 32 de biți pentru versiunea 4 și la 128 de biți pentru versiunea 6); rezultatul se numește *adresa de rețea*. Adresa de rețea se scrie ca și când ar fi o adresă IP normală, după care se scrie (fără spațiu) un caracter *slash* (/) urmat de lungimea prefixului scrisă ca număr în baza 10.

EXEMPLUL 10.6: Prefixul IPv4 1100-0000-1010-1000-110 se scrie, în notația cu adresă de rețea și lungime (notație cu slash) 192.168.192.0/19. Prefixul 1100-0000-1010-1000-1100-0000 se scrie 192.168.192.0/24. De remarcat importanța specificării lungimii.

În notația cu adresă de rețea și mască de rețea, se scrie mai întâi adresa de rețea, ca și în cazul scrierii cu adresă și lungime, după care se scrie (cu un *slash* între ele sau în rubrici separate) așa-numita *mască de rețea*. Mască de rețea constă dintr-un șir de biți 1 de lungimea prefixului de rețea urmat de un șir de biți 0, având în total lungimea unei adrese IP. Mască de rețea, se scrie ca și când ar fi o adresă IP.

Notația cu adresă și mască se utilizează numai pentru IP versiunea 4.

EXEMPLUL 10.7: Prefixul 1100-0000-1010-1000-110 se scrie, în notația cu adresă de rețea și mască, 192.168.192.0/255.255.224.0. Prefixul 1100-0000-1010-1000-1100-0000 se scrie 192.168.192.0/255.255.255.0.

10.2.4. Alocarea adreselor IP și prefixelor de rețea

Alocarea adreselor IP pentru rețeaua mondială Internet se realizează de către *Internet Assigned Numbers Authority (IANA)*. Mai multe despre alocare se găsește la [IANA,]. Deși nu este actualizat, este instructiv de citit și [RFC 1700, 1994].

10.2.4.1. Alocarea pe utilizări

Adresele IPv4 sunt împărțite după cum urmează:

- Adresele cu prefixele 0.0.0.0/8 și 127.0.0.0/8 sunt rezervate. Adresa 127.0.0.1, pentru fiecare nod, desemnează acel nod, cu alte cuvinte un pachet destinat adresei 127.0.0.1 este totdeauna livrat nodului curent. Adresa 0.0.0.0 înseamnă adresă necunoscută; poate fi folosită doar ca adresă sursă în pachete emise de un nod care încearcă să își afle propria adresă.
- Adresele cu prefixul 224.0.0.0/4 sunt utilizate ca adrese de multicast (așa-numita *clasă D*).
- Adresele cu prefixul 240.0.0.0/4 sunt rezervate (așa-numita *clasă E*).
- Adresele cu prefixele 10.0.0.0/8, 172.16.0.0/12 și 192.168.0.0/16 sunt numite *adrese private* [RFC 1918, 1996]. Aceste adrese pot fi utilizate intern de oricine, fără să fie necesar alocarea la IANA, însă cu condiția ca pachetele purtând astfel de adrese ca sursă sau destinație să nu ajungă în afara nodurilor gestionate de acea persoană sau instituție. Aceste adrese se utilizează pentru acele noduri, din rețeaua proprie a unei instituții, care nu au nevoie de acces direct la Internet. Mai multe detalii despre utilizarea acestor adrese vor fi date în § 10.7.2
- Restul adreselor se alocă normal nodurilor din Internet.

10.2.4.2. Alocarea adreselor și dirijarea ierarhică

În lipsa oricăror grupări ale adreselor, majoritatea nodurilor din Internet ar trebui să aibă în tabela de dirijare câte o regulă pentru fiecare nod. O asemenea soluție nu este realizabilă practic la scară mondială. Din această cauză, adresele se alocă instituțiilor doritoare în blocuri de adrese, fiecare bloc având un prefix unic, întocmai ca în cazul subrețelelor.

Un bloc de adrese se alocă unei subrețele sau grup de subrețele interconectate care apar, din restul Internetului, ca o singură subrețea. Din afara subrețelei corespunzătoare unui bloc, toate pachetele destinate adreselor din bloc sunt dirijate identic, conform unei reguli care are ca prefix prefixul blocului. În tabela de dirijare a unui nod oarecare din Internet va fi necesar astfel câte o regulă pentru fiecare bloc, și nu câte o regulă pentru fiecare nod.

Pentru stabilirea dimensiunilor blocurilor, inițial adresele IP versiunea 4 au fost împărțite în *clase*:

- A:** Adresele cu prefixul 0.0.0.0/1 au fost împărțite în 128 blocuri alocabile, fiecare bloc având câte 2^{24} adrese. Lungimea prefixului unui bloc este de 8 biți.

B: Adresele cu prefixul 128.0.0.0/2 au fost împărţite în 16384 blocuri de câte 65536 adrese. Prefixul unui bloc este de 16 biţi.

C: Adresele cu prefixul 192.0.0.0/3 au fost împărţite în 2^{21} blocuri de câte 256 adrese. Lungimea prefixului unui bloc este de 24 de biţi.

Ideea împărţirii între clasele A, B şi C era aceea ca, dându-se o adresă IP, să se poată determina lungimea prefixului blocului din care face parte. Acest lucru simplifică mult calcularea tabelelor de dirijare şi chiar căutarea în tabela de dirijare a regulii aplicabile.

Împărţirea în clase s-a dovedit prea inflexibilă. Pe de o parte, împărţirea este inefficientă, ducând la alocarea câte unui bloc de clasă B (adică 65536 adrese) pentru instituţii care nu aveau nevoie de mai mult de câteva sute de adrese. Pe de altă parte, nu există nici o corelaţie între blocurile de adrese alocate unor instituţii diferite dar din aceeaşi zonă geografică; în consecinţă, pentru majoritatea rutelor din Internet este nevoie de câte o regulă de dirijare pentru fiecare instituţie căreia i s-a alocat un bloc de adrese.

Ca urmare s-a decis o nouă schemă de alocare a blocurilor de adrese. Noua schemă se numeşte CIDR (engl. *Classless InterDomain Routing*) şi este descrisă în [RFC 1518, 1993].

În schema CIDR, un prefix de bloc poate avea orice lungime. O instituţie ce doreşte acces Internet poate solicita alocarea unui bloc de adrese, cu un număr de adrese egal cu o putere a lui 2.

O instituţie care furnizează acces Internet altor instituţii este încurajată să aloce mai departe, din blocul alocat ei, sub-blocuri pentru instituţiile cărora le oferă acces Internet. Astfel, din afara reţelei furnizorului de acces Internet, reţeaua furnizorului împreună cu toţi clienţii lui se vede ca o singură subreţea în care toate adresele au acelaşi prefix.

CIDR mai prevede o grupare a blocurilor pe continente, astfel încât pentru un nod aflat pe un continent toate (sau majoritatea) adreselor de pe un alt continent să se dirijeze conform unei singure reguli. Această grupare este aplicabilă doar adreselor care nu erau deja alocate la momentul introducerii CIDR; CIDR nu şi-a pus problema realocării adreselor deja alocate.

Pentru adresele IP versiunea 6 se foloseşte numai schema CIDR.

10.2.5. Erori la dirijare şi protocolul ICMP

Protocolul ICMP (*Internet Control Message Protocol*) are scop diagnosticarea diverselor probleme legate de dirijarea pachetelor IP. Fiind strâns legat de protocolul IP, ICMP are două versiuni, ICMP pentru IPv4, descris în [RFC 792, 1981] şi numit uneori ICMPv4, şi ICMP pentru IPv6, descris în [RFC 2463, 1998] şi numit şi ICMPv6.

Protocolul constă în transmiterea, în anumite situații, a unor *pachete ICMP*. Un pachet ICMP este un pachet IP în care câmpul *protocol* are valoarea 1 pentru ICMPv4, respectiv 58 pentru ICMPv6, iar zona de date utile este structurată conform standardului ICMP.

Pachetele ICMP sunt de obicei generate de modulul de rețea al unui nod, ca urmare a unei erori apărute în livrarea unui pachet IP. Pachete ICMP mai pot fi generate și de programe utilizator, prin intermediul *socket*-urilor de tip `SOCK_RAW`. Astfel de aplicații servesc la testarea funcționării rețelei.

O dată generat, un pachet ICMP este transmis prin rețea ca orice alt pachet IP. Ajuns la destinație, modulul de rețea (IP) al nodului destinație examinează câmpul *protocol* și, constatând că este vorba de un pachet ICMP, îl livrează modulului ICMP al nodului destinație. Modulul ICMP trebuie să fie prezent și funcțional în orice nod IP; în implementările obișnuite este parte a nucleului sistemului de operare al calculatorului ce constituie nodul.

Datele utile sunt formate conform standardului ICMP și încep cu doi întregi pe câte 8 biți reprezentând tipul și subtipul mesajului ICMP (vezi tabelul 10.4). Formatul restului pachetului depinde de tipul mesajului ICMP; în majoritatea cazurilor, este prezentă o copie a primilor câteva zeci de octeți din pachetului IP care a dus la generarea pachetului ICMP.

Situațiile ce duc la generarea pachetelor ICMP, precum și acțiunile întreprinse de un nod la primirea unui pachet ICMP, sunt descrise în paragrafele următoare.

10.2.5.1. Pachete nelivrabile

Un nod declară un pachet nelivrabil dacă:

- nici o regulă din tabela de dirijare a nodului nu este aplicabilă destinației pachetului; sau
- interfața de rețea prin care trebuie trimis pachetul nu este funcțională sau nu poate livra pachetul destinatarului (destinatarul nu răspunde).

În aceste cazuri, nodul curent trimite un pachet ICMP, având:

- *adresa sursa*: adresa nodului curent,
- *adresa destinație*: adresa sursă a pachetului nelivrabil,
- *tip*: *destination unreachable*.

Pachetul ICMP mai cuprinde antetul pachetului ce nu a putut fi livrat. Destinatarul pachetului ICMP, care este de fapt sursa pachetului nelivrabil, trebuie să analizeze antetul pachetului returnat și să informeze nivelul superior (probabil TCP sau UDP) asupra problemei.

Tip	Subtip	Ce semnalizează
3 — Destination unreachable	0 — network unreachable 1 — host unreachable 3 — protocol unreachable	pachet nelivrabil, conform § 10.2.5.1
	4 — fragmentation needed	pachet prea mare și flagul <i>nu fragmenta</i> setat; vezi § 10.2.6.1
	5 — source route failed	pachetul a avut opțiunea <i>dirijare de către sursă</i> și ruta specificată este invalidă.
11 — time exceeded	0 — TTL exceeded	pachetul se află de prea mult timp în rețea (probabil ciclează), § 10.2.5.3
	1 — fragment reassembly time exceeded	probabil fragment pierdut, § 10.2.6.1
12 — parameter problem		pachet neconform cu standardul
4 — source quench		cerere încetinire sursă, § 10.2.5.4
5 — redirect	0 — network	redirecționare, § 10.2.5.5
	1 — host	
	2 — TOS & network	
	3 — TOS & host	
8 — echo request		cerere ecou, § 10.2.5.2
9 — echo reply		răspuns ecou, § 10.2.5.2

Tabelul 10.4: Tipuri și subtipuri mai importante de pachete ICMPv4

10.2.5.2. Diagnosticarea funcționării rutelor

Testarea funcționării comunicației la nivel rețea este un test simplu și extrem de util în găsirea penelor dintr-o rețea.

În acest scop, pe majoritatea sistemelor există o comandă utilizator, numită `ping`, care testează legătura dintre nodul curent și nodul specificat.

Comanda `ping` funcționează prin trimiterea unui pachet ICMP cu tipul *echo request* (rom. cerere ecou) către nodul specificat. Nodul destinație al unui pachet *echo request* răspunde prin trimiterea înapoi (către sursa pachetului *echo request*) a unui pachet ICMP cu tipul *echo reply* (rom. răspuns ecou). Pachetul *echo reply* este livrat comenzii `ping`.

Pachetele *echo request* și *echo reply* se mai numesc uneori *ping* și *pong*. Pachetele cu tipurile *ping* și *pong* au prevăzute, conform standardului, două câmpuri, *identificare* și *nr. secvență*, pe baza cărora nucleul sistemului și comanda `ping` identifică corespondențele între pachetele *ping* trimise și pachetele *pong* recepționate. Pachetele *ping* și *pong* au prevăzut și un câmp, de dimensiune arbitrară, de date utile; scopul acestui câmp este testarea transmiterii pachetelor mari.

Pe lângă comanda `ping` care testează funcționarea unei legături, există o comandă, `traceroute` (pe sisteme de tip Unix) sau `tracert` (pe Windows), care afișează adresele ruterelor prin care trece un pachet pentru o anumită destinație.

Există mai multe metode pentru a afla drumul spre un anumit nod. Metoda utilizată de comanda `traceroute` se bazează pe trimiterea, spre acel nod, a unor pachete *ping* cu valori mici pentru timpul de viață (vezi § 10.2.5.3). Un astfel de pachet parcurge începutul drumului spre nodul destinație, însă, după parcurgerea unui număr de noduri intermediare egal cu valoarea inițială a timpului de viață, provoacă trimiterea înapoi a unui pachet ICMP de tip *TTL exceeded*. Trimițând pachete *ping* cu diferite valori pentru timpul de viață, se primesc pachete *TTL exceeded* de la diferitele noduri de pe traseul spre destinație.

O altă posibilitate de-a afla ruta spre un anumit nod este furnizată de un antet opțional, standardizat și în IPv4 și în IPv6, care cere ruterelor să-și scrie fiecare adresa în acest antet opțional.

10.2.5.3. Ciclarea pachetelor IP

Este posibil să existe (temporar) inconsistențe în tabelele de dirijare. De exemplu, se poate ca tabela de dirijare a nodului A să indice nodul B ca nod următor pe ruta către C, iar tabela nodului B să indice ca nod următor pe ruta către C nodul A. În acest caz, dacă A primește un pachet destinat lui

C i-l va trimite lui B, B va pasa pachetul înapoi lui A, ș. a. m. d.

Pentru a preveni ciclarea nelimitată a pachetelor în astfel de cazuri, în antetul IP este prevăzut un câmp numit *timp de viață*. Valoarea acestui câmp este inițializată de către nodul sursă al pachetului (valoarea inițială este de ordinul zecilor) și este scăzută cel puțin cu 1 de către fiecare nod prin care trece pachetul. Dacă valoarea ajunge la 0, nodul nu mai trimite mai departe pachetul ci îl ignoră sau trimite înapoi un pachet ICMP cu tipul *time exceeded*, subtipul *time to live (TTL) exceeded* (rom. *depășire timp de viață*) pentru a semnala situația.

10.2.5.4. Congestia

În general, prin *congestie* se înțelege situația în care într-un nod intră pachete într-un ritm mai rapid decât poate nodul să retrimită pachetele, rezultând de aici o funcționare proastă a rețelei (vezi § 5.3).

În cazul congestiei, nodul congestionat poate cere sursei să reducă traficul prin trimiterea către aceasta a unui pachet ICMP cu tipul *source quench*.

10.2.5.5. Redirecționarea

Un nod, care primește un pachet și constată că trebuie trimis mai departe în aceeași subrețea din care a sosit pachetul, poate informa sursa pachetului cu privire la faptul că pachetul a mers pe o rută neoptimă. Informarea se face printr-un pachet ICMP cu tipul *redirect*.

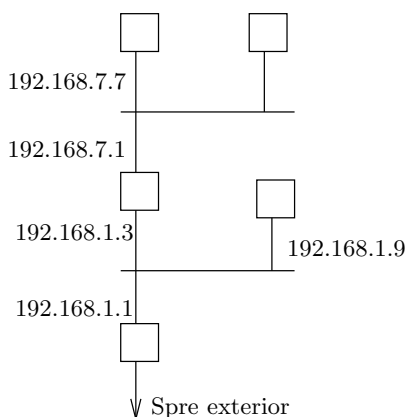


Figura 10.3: O rețea pentru ilustrarea redirecționării pachetelor (vezi exemplul 10.8)

EXEMPLUL 10.8: Considerăm rețeaua din figura 10.3. Pentru nodurile din

subrețeaua 192.168.1.0/24, ar trebui să existe în tabela de dirijare:

- o regulă care să asocieze prefixului 192.168.7.0/24 gateway-ul 192.168.1.3;
- o regulă indicând ca *default gateway* adresa 192.168.1.1.

În practică, pentru simplificarea administrării, se omite configurarea primei reguli pe toate nodurile cu excepția nodului 192.168.1.1. În consecință, o stație din subrețeaua 192.168.1.0/24, de exemplu 192.168.1.9, care are de trimis un pachet către un nod din subrețeaua 192.168.7.0/24, de exemplu către 192.168.7.7, va trimite pachetul lui 192.168.1.1 în loc de 192.168.1.3. Nodul 192.168.1.1 trimite mai departe pachetul către 192.168.1.3 și, totodată, trimite un pachet *ICMP redirect* către 192.168.1.9; aceasta din urmă își poate actualiza tabela de dirijare pentru a trimite direct la 192.168.1.3 următoarele pachete destinate nodurilor din subrețeaua 192.168.7.0/24.

10.2.6. Alte chestiuni privind dirijarea pachetelor

10.2.6.1. Dimensiunea maximă a pachetelor și fragmentarea

Dimensiunea maximă a unui pachet IP este de 64 KiB.

Pe de altă parte, legătura directă între două noduri, dacă are noțiunea de pachet, are o dimensiune maximă a pachetului, care poate fi mai mică decât dimensiunea maximă a pachetului IP: de exemplu, un pachet Ethernet are o dimensiune maximă de 1500 octeți.

Dacă un pachet IP de transmis este mai mare decât dimensiunea maximă a pachetelor admise de legătura directă între două noduri de pe traseu, există următoarele acțiuni posibile:

- se face o fragmentare și reasamblare la nivelul legăturii directe, în mod invizibil față de nivelul rețea;
- se face o fragmentare și reasamblare la nivelul rețea (IP);
- se refuză livrarea pachetelor IP și se lasă în sarcina nivelului superior să se descurce, eventual furnizându-i acestuia dimensiunea maximă acceptabilă a pachetului.

Trebuie remarcat că, în 1981, când s-a standardizat protocolul Internet, era mult prea mult să se ceară fiecărui nod să dispună de câte 64 KiB de memorie pentru memorarea fiecărui pachet.

Fragmentarea la nivelul legăturii directe nu privește protocolul IP. Protocolul IP versiunea 6 cere ca nivelul legăturii directe să permită transmiterea pachetelor IP de până la 1280 B, recomandabil până la 1500 B.

Pentru a permite producerea, de către nivelul superior, a unor pachete IP suficient de mici, există un protocol pentru aflarea dimensiunii maxime a pachetelor ce pot trece prin legăturile directe. Protocolul este descris în [RFC 1981, 1996].

Protocolul Internet permite și fragmentarea la nivel rețea a pachetelor.

Pentru IP versiunea 4, câmpurile necesare pentru fragmentarea și reasamblarea pachetelor sunt prevăzute în antetul standard. De asemenea, există un flag, *nu fragmenta*, care cere rutelor de pe traseu să nu încerce fragmentarea ci în schimb să abandoneze transmiterea pachetului.

Pentru IP versiunea 6, câmpurile privind fragmentarea au fost mutate într-un antet opțional, deoarece este probabil să nu fie utilizate frecvent. Fragmentarea poate fi făcută doar de emițătorul pachetului, ruterele de pe traseu fiind obligate să abandoneze transmiterea în cazul în care pachetul este prea mare.

Fragmentele sunt pachete IP obișnuite, care se transmit independent unul de altul din punctul în care s-a efectuat fragmentarea.

Nodul destinație efectuează reasamblarea pachetelor. În acest scop se utilizează câmpurile *identificare* și *deplasament* și flagul *mai urmează fragmente*. Astfel, un pachet se va asambla din fragmente având toate aceeași valoare în câmpurile *identificare*, *adresă sursă*, *adresă destinație* și *protocol*. Pachetul asamblat va avea antetul identic cu al fragmentelor (mai puțin câmpurile ce controlează fragmentarea). Datele utile vor fi reconstituite din datele utile ale fragmentelor. Câmpul *deplasament* al unui fragment arată locul datelor utile din fragment în cadrul pachetului (reamintim că fragmentele, ca orice pachete IP, se pot pierde, pot fi duplicate și ordinea lor de sosire poate fi inversată). Lungimea pachetului este determinată din faptul că, în ultimul fragment, flagul *mai urmează fragmente* are valoarea 0.

Destinația încearcă reasamblarea unui pachet din momentul în care a primit primul fragment al pachetului. Dacă celelalte fragmente nu sosesc într-un interval de timp suficient de scurt, nodul abandonează reasamblarea și trimite înapoi un pachet ICMP cu tipul *time exceeded*, subtipul *fragment reassembly time exceeded*.

10.2.6.2. Calitatea serviciului

Dacă un nod este relativ aglomerat, acesta trebuie să ia decizii privind prioritatea pachetelor:

- dacă unele pachete trebuie trimise cât mai repede, față de altele care pot fi ținute mai mult în coada de așteptare;
- la umplerea memoriei ruterului, care pachete pot fi aruncate (distruse).

Câmpul *tip serviciu* din antetul IP conține informații despre nivelul de calitate a serviciului cerut de emițătorul pachetului; în funcție de acesta, modulul de rețea ia deciziile privind ordinea de prioritate a pachetelor.

10.2.7. Configurarea și testarea unei rețele IP locale

10.2.7.1. Alegerea parametrilor

În majoritatea cazurilor, într-o rețea locală, subrețelele IP, adică legăturile directe între nodurile IP, se realizează prin intermediul unor rețele din familia IEEE 802 (Ethernet sau 802.11). Primul lucru ce trebuie stabilit este alcătuirea subrețelelor.

În continuare se stabilește, pentru fiecare subrețea IP, prefixul de rețea corespunzător. Prefixul fiecărei subrețele trebuie, pe de o parte, să permită alocarea unui număr suficient de adrese nodurilor din subrețea și, pe de altă parte, să ducă la alocarea de adrese dintre adresele alocate de furnizorul de acces Internet sau dintre adresele rezervate pentru rețele private (vezi și § 10.7.2 pentru alte considerente privind utilizarea adreselor private).

Pentru fiecare subrețea IP, nodurile componente trebuie să facă parte din același VLAN 802.1Q (dacă se definesc VLAN-uri) și ca urmare trebuie să facă parte din aceeași rețea 802 fizică. Această cerință este determinată de faptul că, în cadrul unei subrețele IP, fiecare nod trebuie să poată comunica cu orice alt nod al subrețelei fără a implica dirijare la nivel IP; comunicarea trebuie să fie realizată de nivelul inferior, adică de rețeaua 802.

De notat însă că în cadrul aceleiași rețele IEEE 802, și chiar în cadrul aceluiași VLAN 802.1Q, se pot defini mai multe subrețele IP. Astfel de subrețele lucrează independent una de cealaltă și necesită rutere pentru a fi legate logic între ele. Pentru ca un nod să fie membru în subrețelele IP stabilite în aceeași rețea fizică este suficient să definească mai multe adrese IP pe aceeași placă de rețea (vezi § 10.5, în special § 10.5.1 pentru detalii).

Notă: independența subrețelelor IP de pe același VLAN este limitată de faptul că subrețelele partajează debitul maxim de transmisie și că un intrus care ar sparge un calculator ar putea avea acces la toate subrețelele IP stabilite pe VLAN-ul sau rețeaua fizică din care face parte calculatorul spart.

Configurarea tabelului de dirijare trebuie să fie făcută astfel încât, pentru orice nod sursă și pentru orice nod destinație, fiecare nod de pe traseul unui pachet să găsească corect următorul nod. În rețelele cu structură arborescentă (în care între oricare două subrețele există un singur drum posibil), acest lucru se realizează de regulă astfel:

- Pentru fiecare subrețea se alege, dintre nodurile ce acționează ca rutere către alte subrețele, un *default gateway*. Acesta se alege de regulă ca

fiind nodul din subrețea cel mai apropiat de ieșirea spre restul Internet-ului. Se obișnuiește ca nodul ales ca *default gateway* să primească adresa IP cea mai mică din subrețea (adică adresa în care sufixul are valoarea 1).

- Pe toate nodurile subrețelei se configurează ca *default gateway* nodul ales ca *default gateway* al subrețelei. Pentru nodurile care fac parte din mai multe subrețele, se ia *default gateway*-ul din subrețeaua cea mai apropiată de exterior (astfel un nod nu va avea ca *default gateway* pe el însuși).
- Pe nodul ales ca *default gateway* pentru o subrețea se vor configura rutele către subrețelele „din subordine“ — subrețelele mai depărtate de legătura spre exterior decât subrețeaua considerată.

Mai notăm că într-o tabelă de dirijare statică nu se pot configura, pentru toleranță la pene, mai multe căi spre o aceeași destinație. Dacă se dorește așa ceva este necesară instalarea unui program de calcul automat al tabelii de dirijare.

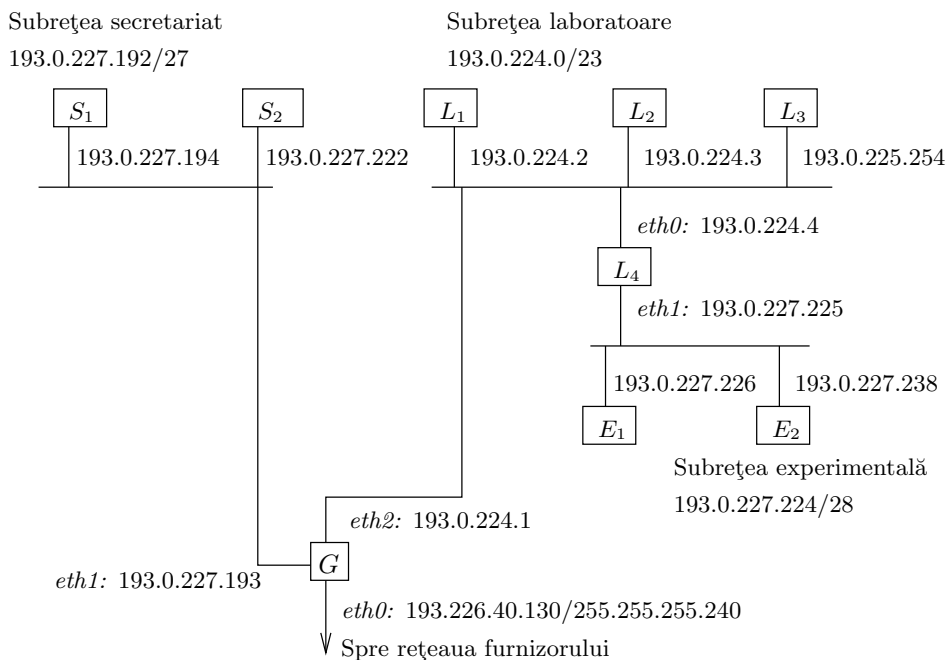


Figura 10.4: Rețea pentru exemplul 10.9

EXEMPLUL 10.9: Să considerăm că avem de construit o rețea într-o școală. Presupunem că am obținut alocarea blocului de adrese 193.0.224.0/22 pentru utilizare în rețeaua proprie și că ruterul ce asigură legătura cu rețeaua furnizorului de acces Internet a primit adresa (în rețeaua furnizorului) 193.226.40.130 cu masca 255.255.255.240 (prefix de 28 de biți).

Să presupunem, de asemenea, că s-a decis împărțirea rețelei interne în trei subrețele (fig. 10.4), respectiv pentru secretariat, laboratorul de informatică și o rețea specială pentru experimente. Împărțirea luată ca exemplu este o împărțire tipică din considerente de trafic și de securitate: rețeaua experimentală trebuie să poată fi izolată ușor de restul rețelei, iar secretariatul este separat față de traficul și eventual atacurile dinspre laboratoarele de informatică.

Fiecare subrețea este construită dintr-un număr de switch-uri Ethernet, *access point*-uri 802.11 și calculatoarele respective. Switch-urile și *access point*-urile nu au fost figurate explicit deoarece ele nu sunt vizibile din punctul de vedere al nivelului IP.

Pentru conectarea celor trei subrețele împreună și la Internet configurăm două rutere: G , dotat cu trei plăci de rețea, care leagă rețeaua secretariatului, rețeaua din laboratoare și rețeaua furnizorului de acces Internet și L_4 (probabil amplasat în laborator, pentru a fi la îndemână în timpul experimentelor), dotat cu două plăci de rețea, care leagă rețeaua experimentală de rețeaua din laborator.

Odată stabilite subrețelele, să alocăm adresele. Blocul de adrese disponibile este 193.0.224.0/22, conținând 1024 de adrese. Putem crea blocuri având ca dimensiuni puteri ale lui 2: 512, 256, 128, 64, 32, 16, 8 sau 4 adrese. Începem prin a alocă laboratoarelor un bloc cât mai mare, de 512 adrese (510 utilizabile efectiv), anume 193.0.224.0/23. Din blocul de 512 adrese rămas (193.0.226.0/23), să alocăm 32 adrese secretariatului și 16 adrese rețelei experimentale. Este bine să le alocăm cât mai compact, pentru ca dintre adresele nealocate să păstrăm posibilitatea de-a alocă blocuri cât mai mari. Vom alocă cele două blocuri de 32 și 16 adrese din ultimul bloc de 64 de adrese din cele 512 libere: 193.0.227.192/27 pentru secretariat și 193.0.227.224/28 pentru rețeaua experimentală.

Pentru fiecare din cele trei subrețele, există o alegere naturală pentru *default gateway*: G pentru rețeaua secretariatului și pentru rețeaua din laboratoare și, respectiv, L_4 pentru rețeaua experimentală. În fiecare caz, *default gateway*-ul este nodul cel mai apropiat de exterior. În fiecare subrețea, adresa dată ruterului cu rol de *default gateway* este cea mai mică adresă din acea subrețea.

Să vedem acum cum trebuie configurate tabelele de dirijare. Pentru stații, tabelele sunt formate din câte două reguli: o regulă pentru livrarea directă, care asociază prefixului subrețelei unica interfață de rețea, și o regulă implicită, care asociază prefixului vid adresa *default gateway*-ului.

Pentru nodul L_4 , tabela de dirijare are trei reguli, două fiind pentru livrarea directă prin cele două interfețe, iar a treia este regula implicită:

- 193.0.224.0/23 → *eth0*;
- 193.0.227.224/28 → *eth1*;
- 0.0.0.0/0 → 193.0.224.1 (prin *eth0*).

Pentru nodul G avem 5 reguli: trei reguli de livrare directă prin cele trei interfețe, o regulă implicită indicând *default gateway*-ul rețelei furnizorului de acces Internet și o regulă pentru dirijarea spre subrețeaua „subordonată“ 193.0.227.224/28:

- 193.226.40.128/28 → *eth0*;
- 193.0.227.192/27 → *eth1*;
- 193.0.224.0/23 → *eth2*;
- 0.0.0.0/0 → 193.226.40.129 (prin *eth0*).
- 193.0.227.224/28 → 193.0.224.1 (prin *eth1*).

10.2.7.2. Configurarea parametrilor de rețea pe diverse sisteme de operare

Pe sistemele Windows, există două posibilități de configurare: comanda `ipconfig` (în mod text) și seria de casete de dialog din Start/ Control panel/ Network/ *interfață*. Prin ambele interfețe se realizează atât modificarea parametrilor din modulul IP din nucleul sistemului de operare cât și scrierea lor în *Windows registry* pentru reîncărcarea lor la repornirea sistemului. Comportamentul de ruter, dacă este dorit, trebuie activat explicit.

Pe sistemele de tip Linux configurarea este puțin mai complicată, dar și mult mai flexibilă. Comanda `ifconfig` setează parametrii legați de interfețele de rețea, anume adresa IP și masca de rețea. Tot comanda `ifconfig` introduce în tabela de dirijare regulile de livrare directă (de fapt acesta este motivul pentru care are nevoie de masca de rețea). Tabela de dirijare se afișează și se modifică cu comenzile `route` sau `ip` (prima este mai simplă și se găsește pe toate sistemele, a doua este mai complexă și servește și altor scopuri). De remarcat că oprirea unei interfețe de rețea cu `ifconfig` duce automat la eliminarea din tabela de dirijare a tuturor regulilor ce au ca țintă

adrese accesibile prin acea interfață. Activarea comportamentului de ruter se face cu o comandă `sysctl`.

Comenzile `ifconfig`, `route`, `ip` și `sysctl` au efect imediat asupra modulului IP din nucleu, dar configurările respective se pierd la repornirea sistemului. Parametrii de rețea utilizați la pornirea sistemului sunt luați de script-urile de inițializare din niște fișiere text; din păcate, fiecare distribuție de Linux are propriile fișiere de configurare. De exemplu, Fedora plasează datele în fișiere în directorul `/etc/sysconfig/network-scripts`.

10.2.7.3. Testarea și depanarea rețelelor

Cel mai util instrument de depanare pentru problemele de conectivitate este comanda `ping`. Pentru buna funcționare a acesteia și a comenzii `traceroute`, este necesar ca, dacă este instalat un filtru de pachete (firewall), acesta să permită trecerea pachetelor ICMP cu tipurile *echo-request*, *echo-reply*, *destination unreachable* și *time exceeded*.

Primul lucru ce trebuie testat, în cazul unei probleme de conectivitate sau după o lucrare mai amplă la rețea, este dacă legăturile directe funcționează. Acest lucru se face dând comanda `ping` pentru câte un vecin din fiecare subrețea din care face parte calculatorul de pe care se efectuează testul. Dacă `ping`-ul merge, înseamnă că legătura funcționează (plăcile de rețea, cablurile, switch-urile și *access point*-urile de pe traseu) și că adresele IP și măștile de rețea sunt „suficient de bune“ pentru ca nodurile să fie văzute ca făcând parte din aceeași subrețea. Dacă `ping`-ul nu merge și pachetele *ping* și *pong* nu sunt filtrate, pana trebuie căutată printre lucrurile enumerate până aici.

Dacă `ping`-ul merge pe legăturile directe, se trece la verificarea legăturilor între subrețele diferite. Dacă o legătură indirectă nu funcționează deși toate legăturile directe ce ar trebui să fie utilizate funcționează, problema este de la regulile de dirijare (sau de la un filtru de pachete; de aceea este bine ca pachetele *ping* și *pong* să nu fie filtrate). Există două lucruri ușor de scăpat din vedere aici: faptul că pentru ca `ping`-ul să meargă este necesar ca dirijarea să funcționeze corect în ambele sensuri și faptul că între regulile de dirijare intră inclusiv regulile de *default gateway* de pe stații.

De exemplu, o stație care nu are configurat *default gateway* va putea comunica cu vecinii direcți, dar nu și cu alte calculatoare — nici măcar cu *default gateway*-ul, dacă esre specificat prin altă adresă decât cea din aceeași subrețea cu stația configurată. Alt exemplu: la rețeaua din exemplul 10.9, dacă pe nodul *G* nu se pune regula care asociază prefixului `193.0.227.224/28` *gateway*-ul `193.0.224.4`, atunci pachetele dinspre subrețeaua `193.0.227.224/28`

pot să iasă spre Internet, însă pachetele dinspre Internet nu trec de nodul G . Un `ping` executat de pe E_1 către L_4 merge, însă către L_2 nu merge. În acest din urmă caz, pachetele `ping` ajung la L_2 , dar pachetele `pong` sunt trimise de L_2 către G (conform regulii implicite). G , neavând altă regulă, aplică și el regula implicită și le trimite către 193.226.40.129 (*default gateway*-ul din rețeaua furnizorului, nefigurat pe desen). De aici pachetele se întorc înapoi spre G , deoarece furnizorul știe că toată rețeaua 193.0.224.0/22 este în spatele lui G . Astfel, pachetele `pong` ciclează între G și 193.226.40.129.

10.3. Nivelul transport

Aplicațiile nu folosesc direct protocolul IP din mai multe motive:

- dacă două aplicații se execută pe același calculator, este necesar ca nucleul sistemului de operare să determine cărei aplicații îi este destinat fiecare pachet sosit;
- serviciul oferit direct de nivelul rețea (pachete ce se pot pierde, pot sosi în altă ordine și, în anumite cazuri, pot fi duplicate) este de obicei inadecvat.

Adaptarea între nevoile aplicațiilor și serviciile oferite de nivelul rețea IP cade în sarcina *nivelului transport*. Nivelul transport constă dintr-o componentă a nucleului sistemului de operare, la care se adaugă uneori niște funcții de bibliotecă.

Componenta nivelului transport situată în nucleul din sistemului de operare furnizează aplicației funcțiile sistem din familia `socket()`. Serviciile oferite aplicației prin `socket`-uri de tip *stream* sunt implementate utilizând protocolul TCP. Serviciile oferite prin `socket`-uri de tip *dgram* sunt implementate prin protocolul UDP. Modulele rețelei IP și locul modulelor TCP și UDP sunt arătate în figura 10.5.

10.3.1. Conexiuni cu livrare garantată: protocolul TCP

Scopul protocolului TCP (*Transmission Control Protocol*) este acela de a realiza o conexiune de tip flux de octeți, bidirecțională, cu livrare garantată. Protocolul este descris în [RFC 793, 1981].

Mai în detaliu, TCP oferă:

- serviciu de tip *conexiune*, cu cele trei faze, de deschidere conexiune, comunicație și închidere conexiune;
- transportă *flux de octeți*, adică emițătorul trimite un șir de octeți, negrupați în mesaje, de lungime arbitrară;

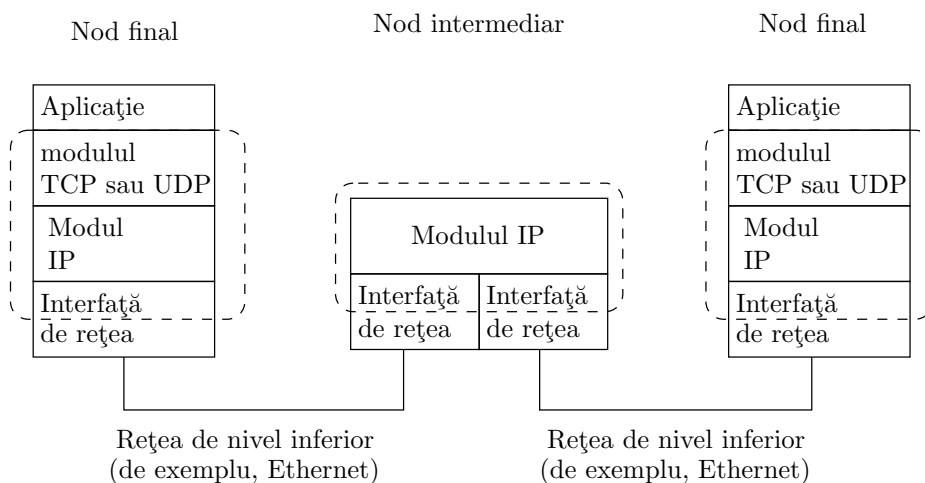


Figura 10.5: Modulele unei rețele IP. Partea inclusă în sistemul de operare este delimitată cu linie punctată.

- legătură *bidirecțională*, adică fiecare din cei doi parteneri de comunicație poate trimite date celuilalt;
- *livrare sigură*, adică octeții trimiși de emițător ajung la receptor sigur, fără duplicate și în aceeași ordine în care au fost trimiși.

Modulul TCP are la dispoziție, pentru realizarea conexiunilor TCP, facilitățile oferite de rețeaua IP.

10.3.1.1. Principiul conexiunii TCP

Livrarea sigură este obținută pe baza unui mecanism de numerotare și confirmare a pachetelor, așa cum am văzut în § 4.3. Mecanismul este implementat după cum urmează. Pentru început considerăm transmisia unidirecțională și conexiunea deja deschisă.

Zona de date utile a unui pachet IP ce transportă date pentru protocolul TCP conține un *antet TCP* și *datele utile TCP*. Antetul TCP este descris în tabelul 10.5.

Fiecărui octet al fluxului de date utile (de transportat de către TCP) i se asociază un *număr de secvență*; octeții consecutivi au asociate numere de secvență consecutive.

Fiecare pachet TCP conține, în zona de date utile, un șir de octeți utili consecutivi. Câmpul *număr de secvență* din antetul TCP conține numărul de secvență al primului octet din datele utile.

Nume câmp	Dim. (biți)	Observații
Port sursă	16	
Port destinație	16	
Nr. secvență	32	
Nr. confirmare	32	
Deplasament date	4	poziția datelor utile în pachet, dependent de dimensiunea opțiunilor
Rezervat	6	valoarea 0
Urgent	1	vezi § 10.3.1.11
<i>Acknowledge</i>	1	indică faptul că numărul de confirmare este valid
<i>Push</i>	1	arată că pachetul trebuie trimis urgent, fără a fi ținut în diverse zone tampon
<i>Reset</i>	1	cere închiderea forțată a conexiunii
<i>Synchronize</i>	1	cere deschiderea conexiunii
<i>Finalize</i>	1	cere închiderea conexiunii
Dim. fereastră	16	vezi § 10.3.1.8
Suma de control	16	suma de control a antetului
Poziție date urgente	16	vezi § 10.3.1.11
Opțiuni	variabil	

Tabelul 10.5: Antetul TCP

Modulul TCP receptor ține evidența numărului de secvență al ultimului octet primit. La primirea unui pachet TCP, modulul receptor determină dacă datele utile sunt octeți deja primiți (duplicați), octeți ce vin imediat în continuarea celor primiți până în acel moment sau octeți până la care există octeți lipsă (nerecepționați încă din cauza unui pachet pierdut sau întârziat). Octeții primiți în continuarea celor deja primiți sunt puși într-o coadă pentru a fi livrați aplicației la cererea acesteia.

La primirea unui pachet TCP, receptorul trimite înapoi un pachet TCP de confirmare. Un pachet TCP de confirmare are în antetul TCP flagul *acknowledge* setat și în câmpul *număr de confirmare* numărul de secvență al următorului octet așteptat de la emițător. Un pachet cu număr de confirmare n informează emițătorul că toți octeții cu numere de secvență mai mici sau egale cu $n - 1$ au fost primiți de receptor și nu mai trebuie retransmiși.

Emițătorul retransmite octeții neconfirmați. Datele utile, furnizate de aplicație emițătorului, sunt păstrate într-o zonă tampon și ținute acolo până la confirmarea primirii lor de către receptor. Datele trimise și neconfirmate într-un anumit interval de timp se retransmit. Datele noi intrate în zona tampon sunt trimise cu un nou pachet. Dacă un pachet nu este confirmat și între prima trimitere și momentul retrimiterii au mai sosit date de la aplicație, emițătorul poate la retrimiteri să concateneze datele vechi neconfirmate cu cele noi.

EXEMPLUL 10.10: În figura 10.6 este prezentată (simplificat) o parte dintr-un schimb de pachete corespunzător unei conexiuni TCP. Presupunem că aplicația sursă are de trimis șirul *abcdefghi*, și că acesta este pasat modulului TCP emițător în etape, întâi șirul *abcd*, apoi *efg*, *h* și în final *i*. Mai presupunem conexiunea TCP deja deschisă și numărul curent de secvență 10. Să analizăm puțin schimbul de pachete:

- Emițătorul trimite un prim pachet, cu număr de secvență 10 și date utile șirul de 4 octeți *abcd*. Acești 4 octeți au numere de secvență respectiv 10, 11, 12 și 13; primul dintre acestea este scris în câmpul *număr de secvență* al antetului TCP.
- La primirea acestui pachet, receptorul răspunde cu un pachet de confirmare, cu numărul de confirmare 14 (acesta este următorul număr de secvență așteptat).
- Emițătorul trimite acum următorul pachet de date, cu numărul de secvență 14 și date utile *efg*. Presupunem că acest pachet se pierde.
- Ca urmare a primirii de la aplicația sursă a următorului octet, *h*, emițătorul TCP trimite imediat următorul pachet, cu numărul de secvență

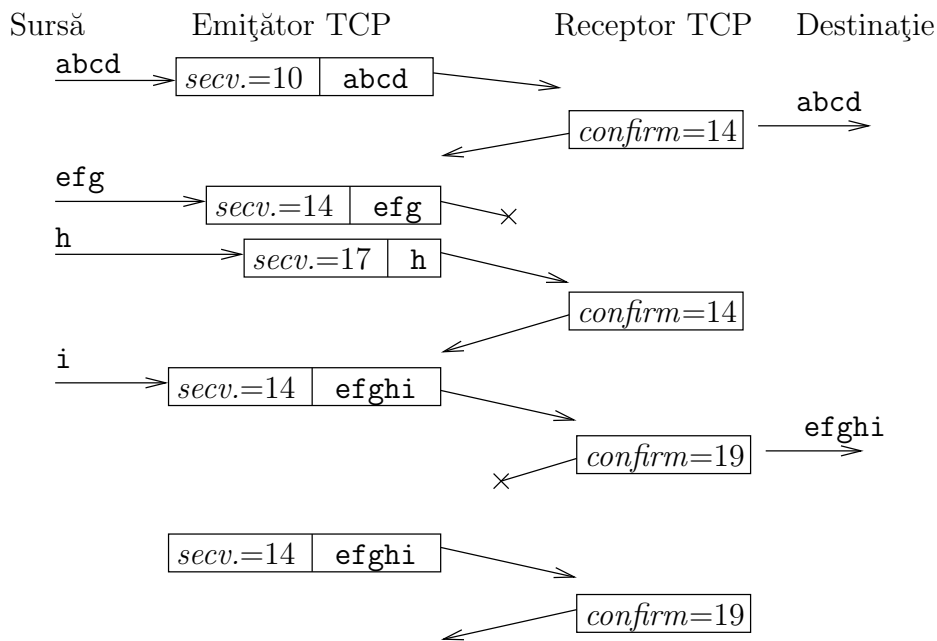


Figura 10.6: O secvență de pachete TCP schimbate între emițător (stânga) și receptor (dreapta); vezi exemplul 10.10.

17 și date utile `h` (presupunem că emițătorul nu utilizează algoritmul Nagle, § 10.3.1.10).

- La primirea acestui pachet (cu număr de secvență 17), receptorul nu îl poate confirma deoarece nu a primit numerele de secvență 14–16; dacă în acest moment receptorul ar trimite un pachet cu număr de confirmare 18, emițătorul ar crede că toate numerele de secvență până la 17 inclusiv au fost primite și nu ar mai retrimite niciodată numerele de secvență 14–16. Ca urmare, receptorul are două posibilități: să ignore pachetul primit (adică să nu trimită nici un pachet înapoi) sau să retrimită numărul de confirmare 14; în exemplul de față am considerat a doua variantă.
- Presupunem acum că pe de o parte a expirat time-out-ul emițătorului pentru numerele de secvență 14–17 și pe de altă parte că a primit de la aplicația sursă următorul octet (`i`). Emițătorul împachetează acum tot ce are de (re)trimis într-un singur pachet și trimite numărul de secvența 14 și datele utile `efghi`.
- Receptorul confirmă pachetul primit, trimițând numărul de confirmare 19. Presupunem că pachetul de confirmare respectiv se pierde.
- Emițătorul retrimite pachetul anterior, după expirarea time-out-ului de la trimiterea lui.
- Receptorul constată că a primit un duplicat al datelor precedente, însă retrimite confirmarea cu numărul 19. Netrimiteria în acest moment a confirmării ar duce la repetarea la infinit de către emițător a pachetului precedent. De notat că, deși receptorul primește un duplicat, emițătorul încă nu știe de primirea datelor de către receptor.

În continuarea schimbului de pachete ilustrat, nu se vor mai trimite pachete până ce aplicația sursă nu va da emițătorului TCP noi date de transmis.

Am presupus pâna aici că numerele de secvență sunt numere naturale care pot crește nedeterminat de mult. În realitate, numerele de secvență sunt reprezentate (vezi tabelul 10.5) pe 32 de biți. Cum un număr de secvență este asociat unui octet transmis, rezultă că există numere de secvență distincte doar pentru 4 GiB de date; după aceea numerele de secvență încep să se repete. Aceasta nu era o problemă în anii '80, deoarece la 10 Mbit/s repetarea unui număr de secvență apare cel mai repede după aproape o oră, timp în care este puțin probabil să mai existe în rețea un pachet vechi cu același număr de secvență. Într-o rețea cu debit de 1 Gbit/s, se pot transmite 4 GiB în 34 secunde, ceea ce face foarte posibilă o confuzie între un pachet care s-a „încurcat“ prin rețea timp de 34 s și un pachet nou care poartă informație situată, în cadrul conexiunii, 4 GiB mai încolo, și are același număr de secvență.

Eliminarea riscului de confuzie între pachete la debite mari de transmisie a datelor se realizează, conform [RFC 1323, 1992], prin introducerea în antetul TCP a unui câmp opțional cuprinzând un marcaj de timp. Metoda se aplică doar la comunicația între module TCP care implementează RFC 1323. În cazul în care unul din modulele TCP nu implementează RFC 1323, modulele TCP vor avea grijă să nu repete un număr de secvență mai devreme de câteva minute, oprind efectiv transmisia la nevoie.

10.3.1.2. Comunicația bidirecțională

Cele două sensuri de comunicație ale unei conexiuni TCP funcționează (aproape) complet independent. Numerele de secvență utilizate pe cele două sensuri evoluează independent.

Totuși, datele utile pentru un sens sunt plasate în același pachet TCP cu confirmarea pentru celălalt sens. Astfel, fiecare pachet TCP conține întotdeauna date pentru un sens și confirmare pentru celălalt sens.

Dacă este necesară transmiterea unor date, dar nu și a unei confirmări, în câmpul *număr de confirmare* se repetă ultimul număr de confirmare transmis.

Dacă este necesar să se transmită doar o confirmare, fără date utile pentru celălalt sens, atunci zona de date utile se face de dimensiune 0 iar în câmpul *număr de secvență* se pune numărul de secvență al următorului octet ce va fi transmis.

10.3.1.3. Deschiderea și închiderea conexiunii

Pentru fiecare sens de comunicație se consideră câte doi octeți fictivi: un *octet de pornire* aflat înaintea primului octet util al datelor transmise și un *octet de încheiere* aflat după ultimul octet al datelor utile. Acești octeți fictivi au asociate numere de secvență ca și când ar fi octeți obișnuiți. Ei nu sunt transmiși în zona de date utile; prezența lor într-un pachet este semnalizată prin setarea flagurilor *synchronize* și respectiv *finalize* din antetul TCP.

Astfel, dacă un pachet TCP are flagul *synchronize* setat, numărul de secvență n și zona de date utile conținând k octeți, înseamnă că pachetul transmite $k+1$ octeți dintre care primul, cu numărul de secvență n , este octetul fictiv de pornire, urmat de cei k octeți din zona de date utile, cu numere de la $n+1$ la $n+k$.

Inițiatorul unei comunicații TCP trimite un pachet TCP conținând un octet fictiv de pornire, fără a seta flagul *acknowledge* (acesta este singurul pachet ce nu are flagul *acknowledge* setat) și punând o valoare arbitrară (care va fi ignorată la destinație) în câmpul *număr de confirmare*.

Numărul de secvență al octetului fictiv de pornire este la alegerea inițiatorului comunicației.

Receptorul pachetului inițial răspunde, dacă dorește să accepte conexiunea, printr-un pachet TCP care, pe de o parte, confirmă pachetul de inițiere primit, iar pe de altă parte conține la rândul lui un octet fictiv de pornire.

Fiecare parte consideră conexiunea deschisă în momentul în care sunt satisfăcute simultan condițiile:

- octetul fictiv de pornire trimis de ea a fost confirmat;
- a primit un octet fictiv de pornire de la partener.

Fiecare parte poate trimite date înainte de a dispune de o conexiune deschisă; datele primite înainte de momentul în care conexiunea este deschisă nu pot fi livrate aplicației până la deschiderea completă a conexiunii.

Închiderea conexiunii se face separat pe fiecare sens de comunicație. Marcarea închiderii unui sens se face trimițând un octet fictiv de încheiere. După trimiterea octetului de încheiere este interzis să se mai trimită date noi. Ca urmare, orice pachete (de confirmare) trimise de partea care a închis conexiunea vor avea ca număr de secvență numărul imediat următor octetului fictiv de încheiere și date utile vide. Octetul fictiv de încheiere se confirmă normal. O conexiune poate funcționa oricât de mult timp cu un sens închis.

Nr. pachet	Sens	Nr. secv.	Nr. confirm.	Flaguri	Date utile
1	$A \rightarrow B$	123	0	syn	—
2	$A \leftarrow B$	25	124	syn,ack	—
3	$A \rightarrow B$	124	26	ack	abc
4	$A \leftarrow B$	26	127	ack	—
5	$A \rightarrow B$	127	26	ack,fin	de
6	$A \leftarrow B$	26	130	ack	—
7	$A \leftarrow B$	26	130	ack	xyz
8	$A \rightarrow B$	130	29	ack	—
9	$A \leftarrow B$	29	130	ack,fin	—
10	$A \rightarrow B$	130	30	ack	—

Tabelul 10.6: Un exemplu de schimb de pachete în cadrul unei conexiuni. Vezi exemplul 10.11.

EXEMPLUL 10.11: Un exemplu de schimb de pachete în cadrul unei conexiuni este dat în tabelul 10.6.

Pentru a urmări ușor schimbul de pachete, să remarcăm că A trimite lui B un număr de 7 octeți din care 2 fictivi, „ $\langle \text{syn} \rangle abcde \langle \text{fin} \rangle$ “, cu numerele de secvență de la 123 la 129 inclusiv, iar B trimite lui A un număr de 5 octeți din care 2 fictivi, „ $\langle \text{syn} \rangle xyz \langle \text{fin} \rangle$ “, cu numerele de secvență de la 25 la 29 inclusiv. Fiecare pachet care transportă octeți numerotați, indiferent dacă aceștia sunt date utile sau octeți fictivi $\langle \text{syn} \rangle$ sau $\langle \text{fin} \rangle$, trebuie confirmat. Pachetele ce conțin doar numărul de confirmare nu se confirmă.

Din punctul de vedere a lui A , conexiunea este complet deschisă la primirea pachetului nr. 2; din punctul de vedere al lui B conexiunea este complet deschisă la primirea pachetului 3. După trimiterea pachetului 5, A nu mai are voie să trimită date noi; poate doar să repete datele vechi (dacă nu ar fi primit pachetul 6 ar fi trebuit să retrimite pachetul 5) și să trimită confirmări. B consideră conexiunea complet închisă după primirea pachetului 10 (a primit un $\langle \text{fin} \rangle$ de la A și a trimis și i s-a confirmat $\langle \text{fin} \rangle$ -ul propriu). A poate considera de asemenea conexiunea complet închisă după trimiterea pachetului 10, însă mai trebuie să păstreze un timp datele despre conexiune pentru cazul în care pachetul 10 s-ar pierde și B ar repeta pachetul 9.

O problemă specială legată de închiderea conexiunii este problema determinării duratei de la închiderea conexiunii până la momentul în care datele asociate conexiunii nu mai sunt necesare și memoria asociată poate fi eliberată.

Din punctul de vedere al unui modul TCP, conexiunea este închisă în momentul în care sunt îndeplinite condițiile:

- modulul TCP a trimis octetul special de încheiere,
- modulul TCP a primit confirmarea propriului octet de încheiere,
- modulul TCP a primit un octet special de încheiere de la partener.

După închiderea conexiunii din punctul de vedere al modulului TCP local, există încă posibilitatea ca modulul TCP partener să nu primească confirmarea modulului TCP local pentru octetul special de închidere trimis de modulul TCP partener). Urmarea este că modulul TCP partener nu consideră închisă conexiunea (conform regulilor de mai sus) și retrimite octetul special de încheiere până la confirmarea acestuia. Modulul TCP local ar trebui să păstreze informațiile despre conexiune până când modulul TCP partener primește confirmarea octetului său de încheiere. Din păcate, determinarea acestui moment este imposibilă, deoarece din acel moment modulul TCP partener nu va mai trimite nici un pachet (conexiunea fiind închisă). Ca soluție de compromis, protocolul TCP prevede păstrarea datelor despre conexiune un anumit interval de timp (de ordinul câtorva minute) după închiderea

conexiunii.

10.3.1.4. Alegerea numărului inițial de secvență

Numărul de secvență al octetului fictiv de pornire, numit și *număr inițial de secvență* (engl. *initial sequence number, ISN*), trebuie ales în așa fel încât să nu poată exista confuzie între numere de secvență dintr-o conexiune veche și cele din conexiunea curentă între aceleași două părți (aceleași adrese IP și numere de port).

Ideal, modulul TCP ar trebui să păstreze datele despre o conexiune atât timp cât mai pot exista în rețea pachete aparținând conexiunii. În aceste condiții, la redeschiderea unei conexiuni între aceleași două părți, fiecare parte poate atribui octetului fictiv de pornire numărul de secvență imediat următor numărului de secvență asociat octetului fictiv de încheiere al conexiunii precedente. În acest caz, putem privi conexiunile succesive ca fiind o singură conexiune în care transmisiile sunt delimitate prin secvențe de octeți fictivi $\langle \text{fin} \rangle \langle \text{syn} \rangle$.

Dacă de la precedentă conexiune a trecut destul timp pentru ca pachetele corespunzătoare să nu mai existe în rețea (fie au ajuns la destinație, fie au fost distruse ca urmare a depășirii timpului de viață), alegerea numărului inițial de secvență poate fi făcută oricum. Există câteva considerente, enumerate mai jos, care duc la utilitatea unor alegeri particulare.

Un prim considerent este îngreunarea unor atacuri de tip *IP spoofing*.

Un atac *IP spoofing* (numiu uneori simplu *spoofing*) constă în a trimite pachete IP în care se falsifică valoarea câmpului *adresă sursă*. Scopul unui astfel de atac este de-a înșela un mecanism de autentificare bazat pe adresa IP a partenerului de comunicație sau de-a deturna o conexiune TCP deja autentificată. În acest din urmă caz, adversarul lasă un client legitim să se conecteze la server și, după efectuarea autentificării, adversarul injectează un mesaj destinat serverului și având ca adresă sursă adresa clientului autentificat. Mesajul este interpretat de server ca provenind de la clientul autentificat și, dacă conține o comandă autorizată pentru client, comanda este executată, deși în realitate provine de la adversar.

De multe ori, într-un atac de tip *spoofing* adversarul nu are și posibilitatea de-a determina ruterele de pe traseu să-i permită recuperarea pachetelor de răspuns. Un atac în astfel de condiții se numește *blind spoofing*.

Un atac *blind spoofing* se contracarează foarte simplu generând aleator numărul inițial de secvență, adică făcând ca valoarea lui să fie imprevizibilă pentru adversar. Cum adversarul trebuie să emită pachete cu numere de secvență și de confirmare valide, în cazul acestor măsuri adversarul trebuie

efectiv să ghicească numărul de secvență.

Un al doilea considerent este prevenirea atacului *syn flooding*. Într-un astfel de atac, adversarul trimite multe pachete TCP de deschidere de conexiune (cu flagul *synchronize* pe 1 și *acknowledge* pe 0), cu diferite valori pentru câmpurile *port sursă* și uneori *adresă sursă* (este posibil ca adversarul să execute și *IP spoofing*). Mașina atacată trebuie să aloce o structură de date în care să memoreze datele despre conexiune până la terminarea conexiunii sau la expirarea unui time-out. Adversarul însă nu mai trimite nimic pe nici una dintre conexiuni, mulțumindu-se să țină ocupată memorie pe mașina atacată; este vorba deci de un atac *denial of service*.

Contramăsura, numită *syn cookie*, se realizează astfel. O mașină care așteaptă cereri de conectare generează aleator un șir de biți, pe care îl ține secret. La primirea unei cereri de conectare, mașina alege, ca număr de secvență inițial (pentru sensul dinspre ea spre inițiatorul conexiunii), valoarea unei funcție de dispersie criptografică, aplicată asupra numărului de secvență al pachetului primit concatenat cu un șirul secret. Apoi, mașina țintă a conexiunii trimite pachetul de răspuns, acceptând numărul de secvență propus de inițiatorul conexiunii și conținând numărul de secvență astfel generat. Mașina țintă nu înregistrează încă, în tabela de conexiuni, conexiunea astfel deschisă. În cazul unei conexiuni reale, la trimiterea următorului pachet de către inițiatorul conexiunii, mașina țintă verifică numărul de secvență folosind aceeași funcție de dispersie, după care memorează conexiunea în tabelul de conexiuni. În acest fel, o conexiune creată dintr-un atac *syn flooding* nu ocupă memorie, în schimb o conexiune legitimă se poate deschide.

10.3.1.5. Închiderea forțată a conexiunii

Refuzul cererii de deschidere a unei conexiuni se face trimițând inițiatorului un pachet TCP cu flagul *reset* setat.

La primirea unui pachet care nu corespunde unei conexiuni deschise (adică pachetul este primit într-un moment în care conexiunea este închisă și pachetul nu are flagul *synchronize* setat), modulul TCP trebuie să trimită înapoi un pachet cu flagul *reset* setat. Ideea este ca, dacă nodul curent a căzut și a fost repornit, pierzând evidența conexiunilor deschise, să informeze toți partenerii de comunicație care încearcă să continue comunicația cu el că datele despre comunicație au fost pierdute.

Un nod care a primit un pachet cu flagul *reset* ca răspuns la un pachet TCP pentru o conexiune trebuie să abandoneze forțat conexiunea. Aplicația ce utilizează acea conexiune este informată, printr-un cod de eroare, la următoarea operație privind conexiunea.

Alte situații care conduc la abandonarea unei conexiuni sunt:

- depășirea unui număr de încercări de trimitere de date fără a primi confirmare;
- primirea unui pachet ICMP cu tipul *destination unreachable*.

Timpul cât emițătorul încearcă retransmiterea pachetelor neconfirmate, până în momentul în care declară legătura căzută, este de ordinul zecilor de secunde până la 2–3 minute. Ca urmare, întreruperea pe termen scurt a unui cablu de rețea nu duce la întreruperea conexiunilor TCP ce utilizează acel cablu. De asemenea, dacă un nod sau o legătură a rețelei IP cade, dar rămâne un drum alternativ între capetele conexiunii TCP, iar nivelul rețea reface tabelele de dirijare pentru a folosi acel drum alternativ și acest lucru se întâmplă suficient de repede pentru ca modulul TCP să nu declare conexiunea căzută, atunci conexiunea TCP continuă să funcționeze normal, cu pachetele IP circulând pe noua rută.

Dacă, pe o conexiune TCP deschisă, toate datele vechi au fost trimise și confirmate, atunci nu se transmit pachete IP câtă vreme nu sunt date utile noi de transmis. Ca urmare, dacă aplicațiile nu comunică vreme îndelungată pe o conexiune TCP deschisă, căderea legăturii sau a unuia din capete nu este detectată de celălalt capăt decât în momentul în care acesta are de transmis date. Dacă acel capăt nu are de transmis date vreme îndelungată, de exemplu câteva ore sau chiar zile, conexiunea rămâne deschisă din punctul de vedere al modulului TCP local. Pentru a evita o astfel de situație, modulul TCP poate fi instruit — via un apel `fcntl()` asupra socket-ului corespunzător conexiunii — să trimită din când în când câte un pachet fără date utile, doar pentru a solicita o confirmare de la celălalt capăt. Opțiunea aceasta se numește *keep alive* (rom. *menține în viață*), deși mai degrabă ar trebui numită *testează că mai este în viață*. Utilizarea opțiunii *keep alive* are și un dezavantaj: căderea temporară a rețelei are șanse mai mari să ducă la abandonarea conexiunii.

10.3.1.6. Identificarea aplicației destinație

Pentru a identifica aplicația căreia îi sunt destinate datele primite, conexiunile TCP sunt identificate printr-un cvadrupelet format din adresele IP ale celor două capete și *porturile* celor două capete. Porturile sunt numere în intervalul 1–65535 utilizate pentru a deosebi conexiunile stabilite între aceleași adrese IP.

Sistemul de operare ține evidența conexiunilor deschise. Pentru fiecare conexiune, sistemul reține adresa IP locală, portul local, adresa IP a celuilalt capăt și portul de la celălalt capăt. De asemenea, sistemul reține aplicația care a deschis conexiunea.

La primirea unui pachet TCP, sistemul ia adresele sursă și destinație din antetul IP și portul sursă și destinație din antetul TCP. Pe baza acestora sistemul identifică conexiunea căreia îi aparține pachetul. De aici sistemul regăsește pe de o parte informațiile necesare gestionării conexiunii (zone tampon, numere de secvență, etc) și pe de altă parte aplicația căreia îi sunt destinate datele.

10.3.1.7. Corespondența între funcțiile `socket()` și acțiunile modului TCP

Funcționalitatea modului TCP este oferită programului utilizator prin intermediul funcțiilor sistem din familia `socket()`. Funcțiile `socket()` și modul lor de utilizare într-o aplicație au fost studiate în § 8.1. Aici vom arăta legătura dintre apelarea de către o aplicație a funcțiilor *socket* și acțiunile modului TCP de expediere și de recepție a unor pachete.

Apelul `socket(PF_INET, SOCK_STREAM, 0)` are ca efect doar crearea unei structuri de date pentru apelurile ulterioare.

Pe partea de server, apelurile `bind()` și `listen()` au, de asemenea, ca efect doar completarea unor informații în structurile de date. Dacă aplicația a apelat direct `listen()` fără a fi apelat înainte `bind()`, sistemul (modulul TCP) îi alocă un port liber.

La primirea unui pachet de deschidere conexiune (cu flagul *synchronize* setat) pentru o adresă și un număr de port pentru care se așteaptă deschiderea unei conexiuni (a fost deja apelat `listen()`), modulul TCP execută schimbul de pachete de deschidere a conexiunii. După aceea, modulul TCP așteaptă ca aplicația să apeleze `accept()` pentru a-i putea semnaliza deschiderea unei noi conexiuni. În cazul în care nu se așteaptă deschiderea unei conexiuni, modulul TCP trimite un pachet cu flagul *reset*.

Dacă, în adresa locală dată funcției `bind()`, s-a specificat constanta `INADDR_ANY`, este acceptat un pachet de deschidere de conexiune ce specifică ca adresă destinație oricare dintre adresele nodului curent. Dacă în apelul `bind()` s-a specificat o anumită adresă a nodului curent, este acceptată deschiderea conexiunii doar dacă adresa destinație a pachetului de deschidere este adresa specificată în apelul `bind()`.

Pe partea de client, apelul `connect()` este cel care determină trimiterea unui pachet cu flagul *synchronize*. Funcția `connect()` așteaptă fie deschiderea completă a conexiunii (confirmarea pachetului *syn* plus un pachet *syn* de la server), fie o semnalizare de eroare (un pachet TCP cu flagul *reset* sau un pachet ICMP). În caz favorabil, funcția `connect()` returnează 0, în caz defavorabil semnalizează eroarea survenită.

10.3.1.8. Controlul fluxului

TCP are și rol de control al fluxului, adică de-a încetini emițătorul în cazul în care receptorul nu este capabil să proceseze datele suficient de repede.

O metodă extremă de control al fluxului este neconfirmarea de către receptor a octeților ce nu pot fi procesați. Metoda aceasta are însă dezavantajul că determină emițătorul să retrimită octeții respectivi, generând trafic inutil.

Metoda utilizată de TCP este ca receptorul să semnalizeze emițătorului, prin valoarea câmpului *dimensiune fereastră* din antetul TCP, numărul de octeți pe care receptorul este capabil să-i recepționeze în acel moment. Emițătorul nu va trimite mai mulți octeți decât dimensiunea ferestrei trimisă de receptor. Există o singură excepție: dacă receptorul a semnalizat dimensiunea ferestrei 0, emițătorul poate trimite un singur octet; aceasta se face pentru cazul în care receptorul a anunțat o fereastră de dimensiune 0 și apoi a anunțat o fereastră mai mare dar pachetul ce anunța mărirea ferestrei s-a pierdut.

Pe lângă mecanismul sus-menționat, emițătorul TCP reduce debitul de date emise și în cazul în care constată pierderi de pachete. Ideea este că pachetele IP se pot pierde fie ca urmare a erorilor la nivel fizic, fie ca urmare a congestiei nodurilor intermediare. Cu excepția transmisiei radio, erorile la nivel fizic sunt rare, astfel încât pierderile de pachete IP se datorează în majoritatea cazurilor congestiei nodurilor.

10.3.1.9. Stabilirea time-out-ului pentru retransmiterea pachetelor

Așa cum am văzut, pachetele TCP neconfirmate într-un anumit interval de timp sunt retransmise. Valoarea aleasă a timpului după care se face retransmiterea influențează performanțele transferului de date. O valoare prea mică duce la repetarea inutilă a unor pachete ce ajung la destinație însă într-un timp mai lung și, ca urmare, la generarea unui trafic inutil. O valoare prea mare duce la detectarea cu întârziere a pachetelor pierdute.

Modulul TCP încearcă să estimeze durata de timp necesară unui pachet emis să ajungă la destinație, să fie procesat și să se întoarcă și să se recepționeze confirmarea. Acest timp se numește *timp dus-întors* (engl. *round-trip time*, *RTT*). Timpul dus-întors nu este fix, ci depinde de perechea emițător-receptor considerată și de încărcarea rețelei în momentul considerat. Modulul TCP estimează statistic media și dispersia timpului dus-întors pentru fiecare conexiune deschisă și fixează timpul după care se retrimite pachetele neconfirmate la o valoare ceva mai mare decât media timpului dus-întors.

10.3.1.10. Algoritmul lui Nagle și optimizarea numărului de pachete

La primirea datelor de la programul aplicație, prin apelul sistem `send()`, modulul TCP poate trimite imediat un pachet sau poate aștepta; așteptarea este utilă dacă astfel se pot plasa mai multe date în același pachet IP.

Algoritmul lui Nagle prevede că, la primirea unor date de la utilizator prin `send()`:

- dacă nu sunt date trimise și neconfirmate, sau dacă datele noi sunt suficient de mari pentru a umple un pachet, datele se trimit imediat;
- altfel, modulul TCP așteaptă până la primirea confirmării sau expirarea timpului de retransmitere, și abia atunci trimite datele primite între timp de la aplicație.

O altă optimizare constă în a întârzia câteva fracțiuni de secundă trimiterea confirmării pentru un pachet TCP primit. Ideea este că, dacă aplicația care recepționează datele din acel pachet are de trimis date ca răspuns la datele primite, datele ce constituie răspunsul să fie trimise în același pachet cu confirmarea datelor primite.

10.3.1.11. Trimiterea datelor speciale (out of band)

TCP prevede un mecanism de transmitere, în cadrul fluxului normal de date, a unor date cu un marcaj special. Mecanismul este întrucâtva echivalent cu a avea două fluxuri de date atașate conexiunii, unul pentru datele „obișnuite“ și celălalt pentru date „speciale“. Datele speciale poartă denumirea, în terminologia angloamericană, *out of band data (OOB)*.

O posibilă utilizare este următoarea: presupunem o aplicație care transferă fișiere. Presupunem că emițătorul trimite mai întâi lungimea fișierului și apoi conținutul. Dacă utilizatorul lansează trimiterea unui fișier mare (să zicem câțiva gigaocți) și apoi se răzgândește și dorește să abandoneze operația, partea de emițător a aplicației nu poate semnaliza receptorului în nici un fel abandonarea transiterii. Aceasta deoarece octeții transmiși sunt interpretați de receptor ca fiind conținutul fișierului. Aici intervin datele cu marcajul *special (out of band)*: emițătorul trimite conținutul fișierului ca date normale, iar o eventuală semnalizare de abandonare a transferului este trimisă ca date speciale.

Datele speciale se consideră că trebuie să fie livrate cât mai repede cu putință aplicației destinație. În terminologia TCP, ele sunt denumite *date urgente* (engl. *urgent data*). Transmiterea lor se face astfel:

- datele speciale se plasează într-un pachet TCP fără a fi precedate de date normale;

- pachetul respectiv are flagul *urgent* setat;
- câmpul *poziție date urgente* conține dimensiunea datelor speciale rămase de transmis (în pachetul curent și în următoarele).

De principiu un pachet conținând date speciale va avea setat și flagul *push*, care indică faptul că modulul TCP receptor ar trebui să livreze datele către aplicația destinație cât mai repede.

10.3.2. Datagrama nesigure: UDP

Există aplicații care se descurcă acceptabil cu datagrama nesigure. Pentru acestea, nivelul transport trebuie să rezolve doar problema găsirii aplicației căreia îi este destinată datagrama. Această problemă este rezolvată de protocolul UDP.

Fiecărei aplicații ce utilizează UDP i se acordă, de către modulul UDP al sistemului de operare local, un *port UDP*. Un port UDP alocat unei aplicații nu va fi acordat și altei aplicații decât după ce este eliberat de prima.

O datagramă UDP poartă ca adrese sursă și destinație perechi formate din câte o adresă IP și un număr de port. Adresa IP este adresa nodului pe care rulează aplicația sursă, respectiv destinație, iar portul este portul alocat de sistem aplicației.

O datagramă UDP este construită dintr-un pachet IP cu identificatorul protocolului UDP în *protocol* și cu zona de date utile conținând un *antet UDP* și datele datagramii UDP. Antetul UDP conține portul sursă și portul destinație. Adresele IP sursă și destinație sunt cele plasate în câmpurile corespunzătoare ale pachetului IP.

Deoarece o datagramă UDP trebuie să fie plasată într-un singur pachet IP, dimensiunea datelor utile ale unei datagrami UDP este limitată de dimensiunea maximă a pachetului IP.

Programul utilizator cere trimiterea unei datagrami UDP prin intermediul apelului `sendto()` sau `sendmsg()`. Programul trebuie să specifice adresa destinație — adresa IP și portul. Adresa sursă este adresa asociată socket-ului de pe care se emite pachetul. Dacă nu s-a asociat în prealabil o adresă (prin apelul `bind()`), sistemul alocă automat un număr de port liber.

Deoarece, conform protocolului UDP, recepția datagramii trimise nu este confirmată în nici un fel, funcțiile `sendto()` sau `sendmsg()` nu au cum să returneze programului apelant informații despre livrarea pachetului. Dealtfel, ambele funcții se termină (returnează controlul apelantului) înainte ca pachetul să fie efectiv livrat.

Funcțiile sistem `recvfrom()` și `recvmsg()` așteaptă recepția unei datagrami având ca adresă destinație adresa asociată socket-ului dat ca argu-

ment. Funcția returnează datele utile din datagramă precum și adresa sursă a datagramei.

10.4. Identificarea nodurilor după nume: sistemul DNS

Utilizarea adreselor IP direct de către utilizatorii umani este dificilă deoarece:

- adresele IP sunt greu de ținut minte de către oameni;
- adresele IP sunt legate de topologia rețelei și se schimbă odată cu aceasta; spre exemplu, dacă o firmă schimbă furnizorul de Internet folosit, adresele stațiilor firmei este probabil să se schimbe.

De fapt, adresele IP sunt similare numerelor de telefon. Ca urmare, este utilă o „carte de telefon a Internetului“.

Serviciul numelor de domeniu — DNS (engl. *Domain Name Service*) — este un mecanism ce permite identificarea unui nod de rețea printr-un nume ușor de memorat de către om și care să fie independent de topologia rețelei.

DNS este construit ca o bază de date ce cuprinde înregistrări ce asociază unui nume o adresă IP. Această bază de date este împărțită între mai multe *servere de nume*, care pot fi interogate de oricine. O aplicație care dorește să comunice și să folosească nume în loc de adrese IP interoghează întâi niște servere de nume, până ce află adresa IP corespunzătoare numelui dat, după care lucrează cu adresa astfel obținută.

Baza de date DNS este stocată distribuit (pe mai multe servere, pentru a nu avea un server supraaglomerat) și redundant (există mai multe servere capabile să răspundă la o cerere dată).

DNS este proiectat în ideea că informațiile se citesc frecvent și se modifică rar. Cu ocazia modificărilor se admite să existe temporar incoerențe — un utilizator să obțină informația nouă în timp ce altul deține informația veche.

10.4.1. Numele de domeniu

Numele de domeniu [RFC 1034, 1987] sunt numele ce pot fi date nodurilor și altor obiecte, în cadrul DNS. Un nume de domeniu este compus dintr-un șir de componente. Fiecare componentă este un șir de caractere; RFC 1034 nu impune restricții, însă recomandă ca fiecare componentă să fie formată din cel mult 63 de caractere, ce pot fi litere, cifre sau caracterul *minus* (-), cu restricția ca primul caracter să fie literă și ultimul caracter să

nu fie minus. Literele mari sunt echivalente cu literele mici corespunzătoare. Componentele au asociată o ordine ierarhică.

Scrierea în text a unui nume de domeniu se face scriind componentele, începând cu cea mai de jos, din punct de vedere al ierarhiei, și terminând cu cea mai de sus. După fiecare componentă se scrie un caracter *punct* (.). În particular, numele vid (format din zero componente) se scrie „.” (un caracter punct).

EXEMPLUL 10.12: În adresa `nessie.cs.ubbcluj.ro`, componentele sunt, în ordine descrescătoare ierarhic:

- `ro` — România,
- `ubbcluj` — Universitatea Babeș-Bolyai Cluj-Napoca,
- `cs` — Departamentul de Informatică (din engl. *Computer Science*),
- `nessie` — numele stației.

Această scriere este inspirată din scrierea adreselor poștale, care încep cu numele destinatarului și se termină cu țara.

În majoritatea cazurilor, aplicațiile acceptă specificarea numelor de domenii fără punctul final. În lipsa punctului final, interpretarea este însă diferită. Anume, dacă numele nu este terminat cu punct, aplicația va încerca să adauge la nume șiruri de componente superioare ierarhic dintr-o listă configurată de administratorul sistemului. Primul nume de domeniu, astfel construit, care există în DNS este considerat ca fiind semnificația numelui dat de utilizator.

EXEMPLUL 10.13: Presupunem că lista de căutare configurată de administrator pentru un sistem conține, în ordine, `scs.ubbcluj.ro`, `cs.ubbcluj.ro` și `ubbcluj.ro`. O căutare pentru numele `www` va duce la căutarea numelui `www.scs.ubbcluj.ro`, care va fi găsit și vor fi returnate informațiile despre acest nume. O căutare pentru numele `www.cs` va duce la căutarea, în ordine, a numelor `www.cs.scs.ubbcluj.ro`, `www.cs.cs.ubbcluj.ro` și `www.cs.ubbcluj.ro`; acesta din urmă este găsit și căutarea este încheiată.

Structurarea numelui în mai multe componente servește la administrarea ierarhică a spațiului de nume. O organizație care dobândește un nume de domeniu poate crea și administra după voie numele formate prin adăugare de componente ierarhic inferioare. De exemplu, Universitatea Babeș-Bolyai din Cluj-Napoca a obținut numele `ubbcluj.ro`. Crearea numelui `nessie.cs.ubbcluj.ro` este decizia exclusivă a Universității Babeș-Bolyai.

O instituție care dorește un nume de domeniu trebuie să contacteze instituția care administrează domeniul părinte și să ceară alocarea unui nume.

Alocarea numelui se plătește — fie o taxă plătită o singură dată, fie o taxă anuală. Instituția ce a obținut numele este responsabilă de întreținerea unui server de nume (vezi § 10.4.3 și § 10.4.4) pentru domeniul ce i-a fost alocat.

Adesea firmele care oferă acces Internet oferă gratuit clienților nume de domeniu ca subdomenii ale domeniului firmei. Exemplu ipotetic, firma XYNet, posesoarea domeniului `xynet.example`, oferă clientului ABC s.r.l. domeniul `abc.xynet.example`. Din păcate, numele mașinilor firmei ABC sunt legate în acest caz de furnizorul de acces Internet, iar dacă firma ABC s.r.l. va schimba furnizorul de Internet, va fi nevoită să-și schimbe numele de domeniu ale serverelor sale, ceea ce poate duce la pierderea clienților ce nu află noul nume al site-ului firmei.

De remarcat că, deși organizarea ierarhică a numelor de domeniu seamănă cu organizarea numelor de fișiere și directoare, nu există o restricție similară cu aceea că într-un director nu e permis să aibă același nume un fișier și un subdirector. Anume, ca exemplu, numele `ubbcluj.ro` și `cs.ubbcluj.ro` pot desemna simultan noduri în rețea.

10.4.2. Structura logică a bazei de date DNS

Să ignorăm deocamdată problemele legate de implementarea DNS.

DNS se prezintă ca un tabel cu cinci coloane: *numele de domeniu*, *tipul*, *clasa*, *valoarea* și *termenul de valabilitate*. Tipul și clasa se utilizează pentru a putea pune în DNS și alte informații în afară de adrese IP. Înregistrările corespunzătoare adreselor IP au tipul *A* (de la *address*=adresă) și clasa *IN* (de la *Internet*). Câmpul *valoare* a unei înregistrări cu tipul *A* și clasa *IN* conține adresa IP a nodului cu numele de domeniu dat.

DNS permite căutarea unei înregistrări pentru care s-au specificat numele de domeniu, clasa și tipul.

Este permis să existe mai multe înregistrări pentru același nume, clasă și tip, dacă au valori diferite. Cineva care obține prin interogarea DNS mai multe adrese IP pentru un nume dat poate folosi oricare din adrese; acestea se presupune că sunt ale aceluiași calculator sau ale unor calculatoare ce furnizează servicii echivalente.

O listă a tipurilor de înregistrări DNS mai des utilizate este dată în tabelul 10.7.

Remarcăm în mod deosebit tipul *CNAME* (nume canonic). O înregistrare având numele *nume1*, tipul *CNAME* și valoarea *nume2* definește *nume1* ca pseudonim pentru *nume2*. În acest caz, *nume2* este considerat numele canonic al aceluși obiect. Dacă pentru un nume există o înregistrare *CNAME*, nu este permis să mai existe vreo altă înregistrare pentru acel nume.

Tip	Valoare	Observații
A	adresă IPv4	adresa corespunzătoare numelui solicitat
AAAA	adresă IPv6	adresa IPv6 corespunzătoare numelui solicitat ([RFC 3596, 2003])
CNAME	nume de domeniu	numele canonic corespunzător numelui solicitat
PTR	nume de domeniu	numele canonic al nodului cu adresa IP solicitată, vezi § 10.4.6
SOA	date de identificare ale informațiilor despre zonă	vezi § 10.4.5
NS	nume de domeniu	numele canonic al serverului de domeniu pentru zona având ca rădăcină numele solicitat
MX	nume de domeniu și prioritate	serverele de poștă electronică pentru domeniul solicitat, § 11.1

Tabloul 10.7: Tipuri de înregistrări DNS mai des folosite

Mai mult, numele canonic din câmpul *valoare* al unei înregistrări *CNAME* nu este permis să apară ca nume de domeniu în altă înregistrare *CNAME*.

O aplicație care caută o înregistrare de un anumit tip pentru un nume trebuie să caute și o înregistrare *CNAME* pentru acel nume. Dacă găsește o înregistrare *CNAME*, trebuie să caute o înregistrare cu numele canonic găsit și având tipul căutat inițial. Valoarea astfel găsită trebuie utilizată ca și când ar fi fost găsită pentru numele original.

10.4.3. Împărțirea în domenii de autoritate

Mulțimea numelor de domeniu este împărțită în *zone*. Pentru fiecare zonă există unul sau mai multe *servere de nume* sau *server DNS* care dețin toate înregistrările corespunzătoare numelor din acea zonă.

Privim spațiul de nume ca un arbore în care rădăcina este domeniul rădăcină și fiecare nume are ca părinte numele obținut din el prin înlăturarea celei mai din stânga componente. O zonă este o submulțime de nume care, împreună cu legăturile dintre ele, formează un arbore. De remarcat că rădăcina zonei face parte din zonă.

Un server care este responsabil de o zonă trebuie să dețină toate înregistrările corespunzătoare numelor din zonă. Faptul că un nume care ar

aparține zonei nu figurează în tabela ținută de acel server înseamnă că numele respectiv nu există.

Din motive de toleranță la pene, pentru fiecare zonă există de regulă cel puțin două servere responsabile. De principiu, tabelele despre o zonă dată ale serverelor responsabile de acea zonă trebuie să fie identice; pot exista însă temporar incoerențe între tabele cu ocazia modificărilor unor informații.

Pe lângă înregistrările despre zonele pentru care este responsabil, un server de nume trebuie să mai dețină înregistrări care să permită regăsirea serverelor de nume ale zonelor adiacente — zona imediat superioară ierarhic și zonele imediat subordonate, dacă există — și a serverelor de nume pentru zona rădăcină. Aceste înregistrări se numesc *glue records* — înregistrări lipici — deoarece crează legătura cu zonele învecinate.

Pentru numele rădăcină al fiecărei zone trebuie să existe următoarele înregistrări:

- O înregistrare *SOA* (*Start Of Authority*), care conține niște date administrative despre zonă (vezi § 10.4.5);
- Una sau mai multe înregistrări *NS* (*Name Server*) care conțin numele serverelor de nume responsabile de zonă. De remarcat că serverele de nume nu este obligatoriu să fie ele însele membre ale zonei.

Înregistrările *NS* ale rădăcinii unei zone împreună cu înregistrările *A* ale acelor nume sunt „înregistrările lipici“ ce trebuie ținute de un server cu privire la zonele vecine.

Un server poate ține și alte înregistrări, în afară de înregistrările din zonele pentru care este responsabil și de înregistrările de legătură.

10.4.4. Mecanismul de interogare a serverelor

Protocolul utilizat pentru interogarea serverelor de nume este descris în [RFC 1035, 1987].

Un server de nume așteaptă cereri prin datagrame UDP trimise pe portul 53 și prin conexiuni TCP pe portul 53. Clientul trimite cererea întâi ca datagramă UDP. Dacă răspunsul este prea lung pentru a încapa într-o datagramă UDP atunci clientul repune întrebarea printr-o conexiune TCP.

Interogarea cuprinde numele de domeniu căutat, tipul și clasa. Răspunsul conține interogarea și un șir de înregistrări, împărțite în trei categorii: înregistrări din zonele pentru care este responsabil, înregistrări de legătură și alte înregistrări.

Dacă numele din interogare este dintr-o zonă pentru care serverul este responsabil, serverul va răspunde cu înregistrarea sau înregistrările care constituie răspunsul la interogare — eventual va spune că nu există înregistrări.

Dacă numele căutat este din afara zonei de responsabilitate, există două comportamente posibile pentru server:

- *iterativ*. Serverul răspunde cu înregistrările de legătură către zona căutată de client. Clientul urmează să interogheze alte servere.
- *recursiv*. Serverul interoghează (el însuși) un server pentru zona vecină mai apropiată de zona numelui căutat de client, eventual interoghează în continuare servere din înregistrările returnate, până ce află răspunsul la întrebarea clientului. Înregistrările găsite sunt plasate în categoria a treia — alte înregistrări.

Un server recursiv poate fi configurat să rețină înregistrările astfel obținute, astfel încât la o interogare ulterioară să poată răspunde direct, fără a mai căuta un server responsabil pentru numele cerut de client. O înregistrare astfel memorată este ținută un timp cel mult egal cu termenul de valabilitate al înregistrării, după care se consideră că informația s-ar fi putut modifica și ca urmare înregistrarea este aruncată.

Căutarea adresei corespunzătoare unui nume se face de către programul utilizator care are de contactat nodul cu numele dat. Interogarea DNS se face de regulă prin intermediul unor funcții de bibliotecă, cum ar fi `gethostbyname()`. Aceste funcții de bibliotecă trebuie să găsească un prim server de nume pe care să-l interogheze. Pentru aceasta, în sistemul de operare există un loc standardizat unde administratorul scrie adresele IP ale unuia sau mai multor servere de nume. În sistemele de tip Unix, locul este fișierul `/etc/resolv.conf`, iar în Windows este în registry.

10.4.5. Sincronizarea serverelor pentru un domeniu

Protocolul de interogare DNS prevede posibilitatea de-a cere toate înregistrările dintr-o anumită zonă ([RFC 1035, 1987], [RFC 1995, 1996]). Acest lucru se utilizează pentru a putea întreține ușor mai multe servere responsabile de o anumită zonă. Toate înregistrările privind zona se scriu în baza de date a unuia dintre servere, denumit *master*. Celelalte servere, numite *slave*, sunt configurate să copieze periodic informațiile de pe *master*.

De notat că, într-o astfel de configurație, atât serverul *master* cât și serverele *slave* sunt considerate responsabile de zonă; mecanismul este invizibil pentru cineva care face o interogare DNS pentru un nume din zonă.

Momentele la care un server *slave* copiază datele de pe serverul *master* depind de următorii parametri din înregistrarea *SOA* a zonei:

- *serial* este numărul de ordine al datelor; administratorul zonei trebuie să crească numărul serial oricâteori modifică vreo înregistrare din zonă.

Un *slave* nu execută copierea dacă numărul serial curent al *master*-ului coincide cu numărul serial propriu.

- *refresh* este timpul după care un server *slave* trebuie să interogheze serverul *master* pentru a vedea dacă s-a modificat vreo înregistrare;
- *retry* este timpul de așteptare după o încercare eșuată de-a contacta serverul *master* înainte de-a încerca din nou;
- *expire* este timpul după care, în cazul în care nu a reușit să contacteze serverul *master*, serverul *slave* trebuie să nu se mai considere responsabil de zonă.

Există și un protocol ([RFC 1996, 1996]) prin care serverul *master* cere explicit unui server *slave* să copieze datele despre zonă.

10.4.6. Căutarea numelui după IP

DNS nu permite căutarea unei înregistrări după valoare, deoarece găsirea serverului ce deține înregistrarea ar necesita interogarea tuturor serverelor DNS din Internet.

Pentru a putea răspunde la întrebări de tipul *cine are o adresă IP dată*, se utilizează următorul mecanism:

Fiecărei adrese IP versiunea 4 i se asociază un nume de domeniu astfel: se scrie adresa în formă zecimală cu punct, cu componentele în ordine inversă, și se adaugă `in-addr.arpa.`. Astfel, adresei IP 193.0.225.34 îi corespunde numele `34.225.0.193.in-addr.arpa.`

Pentru aceste nume se pun în DNS înregistrări cu tipul *PTR* și având ca valoare numele de domeniu al nodului respectiv. Interogarea și administrarea acestor nume de domeniu se fac întocmai ca și pentru numele obișnuite.

De principiu, un subdomeniu din `in-addr.arpa.` corespunde unui bloc de adrese IP alocat unei instituții; subdomeniul corespunzător din domeniul `in-addr.arpa.` este administrat de aceeași instituție.

În situația în care alocarea blocurilor de adrese IP se face după schema CIDR, granițele blocurilor nu coincid cu granițele subdomeniilor lui `in-addr.arpa.`. De exemplu, dacă firma X are alocat blocul de adrese 193.226.40.128/28, ea nu va putea primi în administrare întregul domeniu `40.226.193.in-addr.arpa.`, deoarece acesta conține și alte adrese decât cele ale firmei X. Administrarea numelor din `in-addr.arpa.` se face, în acest caz, conform [RFC 2317, 1998]: numele corespunzătoare IP-urilor se definesc ca pseudonime pentru niște nume din domenii create special pentru blocurile alocate. Pentru exemplul dat, construcția este următoarea:

- se crează domeniul `128/28.40.226.193.in-addr.arpa.`, asociat blocului

193.226.40.128/28, a cărui administrare este delegată firmei X;

- numele de domeniu de la `128.40.226.193.in-addr.arpa`. până la `143.40.226.193.in-addr.arpa`. se definesc ca pseudonime (*CNAME*) pentru numele de la `128.128/28.40.226.193.in-addr.arpa`. până la `143.128/28.40.226.193.in-addr.arpa`.

Pentru adresele IPv6 se folosește un mecanism asemănător, definit în [RFC 3596, 2003]. Anume, fiecărei adrese IPv6 i se asociază un nume în domeniul `ip6.arpa`. Numele se construiește astfel:

- Se iau grupuri de câte 4 biți din adresa IPv6 și se scrie cifra hexa corespunzătoare ca o componentă separată.
- Ordinea ierarhică a componentelor astfel obținute este aceea în care componentele corespunzătoare biților mai semnificativi din adresa IP sunt superioare ierarhic componentelor corespunzătoare biților mai puțin semnificativi.

EXEMPLUL 10.14: Pentru adresa `4321:0:1:2:3:4:567:89ab`, numele de domeniu asociat este

```
b.a.9.8.7.6.5.0.4.0.0.0.3.0.0.0.2.0.0.0.1.0.0.0.0.0.0.0.1.2.3.4.
    ip6.arpa
```

10.5. Legăturile directe între nodurile IP

10.5.1. Rezolvarea adresei — ARP

În multe cazuri, ceea ce, din punctul de vedere al unei rețele Internet este o legătură directă, este de fapt o legătură între două noduri într-o rețea de alt tip, de exemplu o rețea IEEE 802. O astfel de rețea joacă rolul nivelului legăturii de date în contextul protocolului Internet și ca urmare o vom numi aici *rețea de nivel inferior*.

Transmiterea unui pachet IP de la un nod *A* către un nod *B*, în cadrul aceleiași rețele de nivel inferior, se face astfel:

- mai întâi, nodul *A* determină adresa, în cadrul rețelei de nivel inferior (adică adresa MAC, pentru IEEE 802), a nodului *B*;
- apoi *A* trimite pachetul IP nodului *B*, sub formă de date utile în cadrul unui pachet al rețelei de nivel inferior, pachet destinat adresei găsite anterior.

Determinarea adresei MAC a nodului B se face cu ajutorul unui mecanism numit *ARP* (engl. *Address Resolution Protocol*, rom. *protocol de determinare a adresei*). Mecanismul funcţionează astfel:

- A trimite în reţeaua de nivel inferior un pachet de difuziune (broadcast) conţinând adresa IP a lui B . Acest pachet este un fel de întrebare „cine are adresa IP cutare?”.
- La un astfel de pachet răspunde doar nodul cu adresa IP specificată în pachet — adică doar nodul B în cazul de faţă. Pachetul de răspuns este adresat direct lui A (prin adresa MAC a lui A recuperată din interogare) şi conţine adresa IP şi adresa MAC ale lui B . Pachetul are semnificaţia „eu am adresa IP cutare şi am adresa MAC cutare”.

După determinarea corespondenţei $IP \rightarrow MAC$ prin mecanismul ARP, nodul A păstrează corespondenţa în memorie un anumit timp (de ordinul minutelor), astfel încât, dacă nodurile A şi B schimbă mai multe pachete în timp scurt, mecanismul ARP este invocat doar o singură dată.

Dacă nodul A primeşte mai multe răspunsuri ARP cu adrese MAC diferite pentru aceeaşi adresă IP, înseamnă că există mai multe noduri cărora li s-a dat din greşeală aceeaşi adresă IP. În această situaţie A ar trebui să semnalizeze situaţia:

- printr-un pachet ICMP cu tipul *destination unreachable* destinat sursei pachetului destinat lui B , şi
- printr-un mesaj afişat pe ecran şi înregistrat în fişierele jurnal ale sistemului de operare.

Mecanismul ARP este utilizat de asemenea la configurarea adresei unei interfeţe de reţea ca verificare că adresa configurată este unică în subreţea. Mai exact, dacă administratorul configurează o anumită adresă IP pentru o interfaţă, sistemul emite mai întâi o cerere ARP pentru adresa ce urmează a fi setată. Dacă cererea primeşte răspuns înseamnă că mai există un nod ce are adresa respectivă, caz în care sistemul tipăreşte un mesaj de avertisment şi eventual refuză configurarea adresei respective. Dacă cererea nu primeşte răspuns este probabil ca adresa să fie unică şi ca urmare sistemul o poate accepta ca adresă proprie.

Informaţiile despre asocierile $IP \rightarrow MAC$ cunoscute nodului curent se determină, pe sistemele de tip UNIX, cu ajutorul comenzii `arp`. Trimiterea, în vederea testării, a unei cereri ARP către o staţie se poate face cu ajutorul comenzii `arping`.

10.6. Configurarea automată a stațiilor — DHCP

Sunt situații în care este util ca un nod să-și determine propria adresă IP, împreună cu alți câțiva parametri (masca de rețea, *default gateway*, servere DNS) prin interogări în rețea, în loc ca acești parametri să fie stocați într-o memorie nevolatilă (disc sau memorie flash) a nodului. Situații în care acest lucru este util sunt:

- pentru un calculator fără harddisc;
- pentru un laptop, PDA sau alt dispozitiv mobil, care este mutat frecvent dintr-o rețea în alta, unde parametrii trebuie configurați de fiecare dată în funcție de rețeaua la care este conectat;
- într-o rețea mare în care este de dorit ca parametrii de rețea ai stațiilor să poată fi schimbați ușor de pe un calculator central.

Există trei protocoale ce au fost utilizate de-a lungul timpului pentru determinarea parametrilor de rețea: RARP, BOOTP și DHCP. Vom studia mai în detaliu protocolul DHCP, celelalte două nemaifiind utilizate în prezent. De notat însă că protocolul DHCP este conceput ca o extensie a protocolului BOOTP.

Un nod care dorește să-și determine parametrii de rețea (adresa IP proprie, masca de rețea, *default gateway*-uri, numele propriu, adresele serverelor DNS) se numește *client DHCP*. Clientul DHCP trimite o cerere, la care răspunde un *server DHCP* stabilit pentru rețeaua respectivă. Răspunsul conține parametrii solicitați. Vom studia în continuare:

- cum se transmit cererea și răspunsul DHCP, în condițiile în care modulul IP al clientului nu este configurat și ca urmare nu este complet funcțional;
- cum determină serverul parametri clientului.

Transmiterea cererii și a răspunsului DHCP. Presupunem în continuare că nodul client este conectat la o singură rețea de tip IEEE 802.

Cererea DHCP este transmisă ca un pachet UDP. Adresa IP destinație a pachetului este adresa de broadcast locală (255.255.255.255), iar portul destinație este portul standard pe care ascultă serverul DHCP, anume portul 67. Adresa IP sursă este 0.0.0.0 (valoarea standard pentru adresă necunoscută). Pachetul IP este încapsulat într-un pachet Ethernet destinat adresei de broadcast (FF:FF:FF:FF:FF:FF) și purtând ca adresă sursă adresa plăcii de rețea locale. Serverul DHCP trebuie să fie în aceeași subrețea cu clientul

(sau să existe în aceeași subrețea un server *proxy DHCP* care să preia cererea clientului și s-o retrimite serverului).

Răspunsul DHCP este plasat la rândul lui într-un pachet UDP purtând ca adresă sursă adresa serverului DHCP și ca adresă destinație adresa alocată clientului DHCP, cu portul destinație 68. Pachetul este încapsulat într-un pachet Ethernet destinat adresei MAC a clientului. Asocierea IP → MAC pentru transmiterea pachetului de către server nu se face prin mecanismul ARP (care ar eșua deoarece clientul DHCP nu poate încă răspunde la cererea ARP) ci este setată de către serverul DHCP prin intermediul unui apel sistem.

Determinarea parametrilor de către server. Pentru fiecare subrețea pentru care acționează, un server DHCP trebuie să aibă două categorii de date: parametrii comuni tuturor nodurilor din subrețea (masca de rețea, gateway-uri, servere DNS) și parametrii specifici fiecărui nod în parte (adresa IP a nodului).

Parametrii comuni sunt setați de administratorul serverului DHCP într-un fișier de configurare al serverului.

Pentru adresele IP ale nodurilor din rețea există două strategii ce pot fi folosite:

Alocare manuală: Adresa fiecărui nod este fixată manual de către administratorul serverului DHCP. Nodul client este identificat prin adresa MAC sau prin nume. În primul caz administratorul trebuie să scrie într-un fișier de configurare al serverului toate adresele MAC ale plăcilor de rețea și adresele IP corespunzătoare. În al doilea caz, pe serverul DHCP se scriu numele stațiilor și adresele corespunzătoare iar pe fiecare stație se setează numele stației (a doua soluție nu este aplicabilă pe calculatoare fără harddisc).

Alocare dinamică: Serverul dispune de o mulțime de adrese pe care le alocă nodurilor. Serverul păstrează corespondența dintre adresele MAC ale clienților și adresele IP ce le-au fost alocate. Adresele alocate pot fi eliberate la cererea explicită a clientului sau la expirarea perioadei de alocare (vezi mai jos).

Adresele se atribuie de regulă pe o durată determinată. Perioada de alocare poate fi prelungită la solicitarea clientului printr-o cerere DHCP de prelungire. După expirarea perioadei de alocare, clientul nu mai are voie să utilizeze adresa.

Atribuirea adreselor pe perioadă determinată are două avantaje (față de atribuirea pe durată nedeterminată):

- la alocarea dinamică a adreselor, dacă clientul este scos din rețea fără

a elibera explicit adresa, adresa este eliberată automat la expirarea perioadei de atribuire;

- dacă este necesară modificarea strategiei de atribuire a adreselor se pot opera modificările necesare în configurarea serverului DHCP iar clienții vor primi noile adrese cu ocazia încercării de prelungire a atribuirii adresei.

10.7. Situații speciale în dirijarea pachetelor

Vom studia în paragraful de față anumite procedee mai deosebite utilizate în dirijarea pachetelor. Aceste procedee se aplică îndeosebi în rețelele interne ale unor instituții.

10.7.1. Filtre de pachete (firewall)

Un *filtru de pachete* (engl. *firewall*) este un nod IP care nu transmite toate pachetele conform regulilor normale de funcționare ale unui nod IP ci, în funcție de anumite reguli, ignoră complet sau trimite pachete ICMP de eroare pentru anumite pachete.

Scopul unui filtru de pachete este de-a interzice anumite acțiuni în rețea, în special pentru a contracara anumite încercări de spargere a unui calculator.

Configurarea unui filtru de pachete constă în stabilirea unui ansamblu de *reguli de filtrare*. Prezentăm în continuare posibilitățile oferite de mecanismul *iptables* din sistemul Linux, cu mențiunea că facilitățile de bază se regăsesc în toate sistemele.

O *regulă de filtrare* este o pereche formată dintr-o *condiție* și o *acțiune*. Regulile sunt grupate în șiruri numite *lanțuri*. Există trei lanțuri predefinite:

- *INPUT* aplicat pachetelor destinate nodului curent,
- *OUTPUT* aplicat pachetelor generate de nodul curent,
- *FORWARD* aplicat pachetelor generate de alt nod și având ca destinație alt nod (pentru care nodul curent acționează ca ruter).

Pentru fiecare pachet ajuns la modulul IP, acesta aplică prima regulă, din lanțul corespunzător traseului pachetului, pentru care pachetul îndeplinește condiția specificată în regulă. Aplicarea regulii înseamnă executarea acțiunii specificate de regulă.

Principalele *acțiuni* ce pot fi specificate sunt:

- *ACCEPT* — pachetul este livrat normal,
- *DROP* — pachetul este ignorat (ca și când nu ar fi fost primit),

- *REJECT* — se trimite înapoi un pachet semnalând o eroare — implicit *ICMP destination unreachable*.

Condițiile specificate într-o regulă pot privi:

- interfața prin care a intrat pachetul (cu excepția lanțului *OUTPUT*),
- interfață prin care ar ieși pachetul (cu excepția lanțului *INPUT*),
- adresa IP sursă și adresa IP destinație (se poate specifica și un prefix de rețea, condiția fiind satisfăcută de pachetele ce au adresă începând cu acel prefix sau, eventual, pachetele ce au adresă ce nu începe cu acel prefix),
- adresa MAC sursă sau destinație (pentru pachete ce intră, respectiv ies, prin interfețe ce au conceptul de adresă MAC),
- protocolul (TCP, UDP, ICMP),
- portul sursă sau destinație (pentru protocoale care au noțiunea de port),
- tipul și subtipul ICMP (pentru pachete ICMP),
- flag-uri ale diverselor protocoale,
- dimensiunea pachetului,
- starea conexiunii TCP căreia îi aparține pachetul (vezi mai jos).

Un nod (intermediar) prin care trec toate pachetele asociate unei conexiuni TCP poate, examinând antetul TCP al fiecărui pachet, să țină evidență stării conexiunii. Ca urmare, nodul poate stabili dacă un pachet deschide o conexiune nouă, aparține unei conexiuni deschise sau este un pachet invalid.

Este adevărat, acest lucru înseamnă o încălcare a principiului separării nivelelor: TCP este un protocol de nivel transport, deasupra nivelului rețea. Ca urmare, modulele de rețea nu ar trebui să interpreteze protocolul TCP (antetele TCP ar trebui considerate pur și simplu date utile). Ca urmare, nodurile intermediare, din care nu acționează asupra pachetelor în tranzit decât modulul rețea și modulele inferioare, nu ar trebui să „înțeleagă” protocolul TCP.

Regulile de filtrare se configurează de către administratorul sistemului. Ca și în cazul parametrilor IP:

- Pe sistemele Linux, regulile aplicate de nucleul sistemului de operare se examinează și se modifică cu ajutorul unei comenzi — `iptables`. Regulile valabile la inițializarea sistemului sunt configurate de *script*-urile invocate la pornire, fiind încărcate dintr-un fișier text.

- Pe sistemele Windows există o interfață grafică cu care se configurează simultan regulile curente aplicate de nucleu și în același timp acele reguli sunt scrise în *registry* pentru a fi încărcate la repornirea sistemului.

Prin regulile de filtrare se urmăresc de obicei următoarele lucruri:

- Să fie blocate pachetele pentru care se poate determina că adresa sursă a fost falsificată. Aici intră:
 - pachete ce intră pe interfață către Internet și au ca adresă sursă o adresă din rețeaua internă,
 - pachete ce au ca sursă o adresă de broadcast (clasa D) sau de clasă E,
 - pachete ce intră dinspre o anumită subrețea au ca sursă o adresă ce nu face parte din subrețeaua respectivă și nici din alte subrețele din direcția respectivă.
- Să fie interzise conexiunile din afara rețelei locale către servicii care sunt oferite doar pentru rețeaua locală. De exemplu, accesul la un *share* Windows este adesea de dorit să nu fie posibil din afara rețelei locale. Pentru aceasta există două strategii:
 - se blochează pachetele destinate porturilor pe care așteaptă conexiuni serviciile respective;
 - se permit conexiunile către serviciile ce se doresc a fi accesibile din afară (web, mail, eventual ssh), se permit conexiunile inițiate din interior și se interzic toate celelalte pachete.

Prima metodă este mai simplă însă necesită lista completă a serviciilor ce trebuie blocate. A doua metodă este mai sigură, întrucât serviciile sunt inaccesibile dacă nu s-a specificat explicit contrariul, însă este dificil de permis intrarea pachetelor de răspuns pentru conexiunile inițiate din interior. Aceasta se întâmplă deoarece o conexiune inițiată din interior are alocat un port local cu număr impvizibil; ca urmare nu se poate stabili o regulă simplă pentru permiterea intrării pachetelor destinate aceluși port.

Soluția uzuală este:

- pentru conexiun TCP, se permit pachetele asociate unei conexiuni deja deschise, se permit pachetele către exterior, se permit pachetele destinate serviciilor publice și se interzic toate celelalte pachete.

- pentru UDP, unde nu se poate ține evidența unor conexiuni, se interzic pachetele destinate unor servicii private, se permit pachetele spre exterior, se permit pachetele provenite de la serviciile ce se dorește a fi accesate în exterior (serverele DNS sau NTP utilizate) și se interzic toate celelalte pachete.
- Să fie interzise diferite alte pachete „dubioase“, cum ar fi:
 - pachete destinate adresei de broadcast a rețelei locale sau adresei de broadcast generale (255.255.255.255),
 - pachete având ca adresă sursă sau destinație adresa mașinii locale (127.0.0.1) sau o adresă privată.

EXEMPLUL 10.15: Fie un ruter având în spate o rețea internă având adrese cu prefixul 193.226.40.128/28. Ruterul are interfața *eth0* cu adresa 193.0.225.20 către exterior și interfața *eth1* cu adresa 193.226.40.129 către subrețeaua locală.

Din rețeaua locală dorim să se poată deschide orice fel de conexiuni TCP către exterior; din exterior dorim să nu se poată deschide alte conexiuni decât către serverul *http* și *https* de pe 193.226.40.130 și către serverele *ssh* de pe toate mașinile din rețeaua locală.

Din rețeaua locală dorim să putem accesa servicii DNS și NTP. Acestea le furnizăm astfel:

- pe ruter instalăm un server DNS și un server NTP, accesibile din rețeaua locală; acestea furnizează serviciile respective pentru rețeaua locală
- permitem cererile emise de serverele DNS și NTP de pe ruter, precum și răspunsurile corespunzătoare. Cererile NTP provin de pe portul UDP 123 al ruterului și sunt adresate portului UDP 123 al unui nod din exterior, iar cererile DNS sunt emise de pe un port UDP mai mare sau egal cu 1024 și sunt adresate portului DNS 53 de pe un nod extern.

Pentru diagnosticarea funcționării rețelei vom mai permite trecerea pachetelor ICMP *ping*, *pong*, *destination unreachable* și *time exceeded*.

Traficul în interiorul rețelei locale îl permitem fără restricții.

Blocăm toate încercările de *spoofing* detectabile.

```
# blocare spoofing detectabil si alte pachete dubioase
iptables -A FORWARD -i eth0 -s 193.226.40.128/28 -j DROP
iptables -A FORWARD -i eth0 -s 193.0.225.20 -j DROP
iptables -A FORWARD -i eth0 -s 127.0.0.0/8 -j DROP
```



```
iptables -A FORWARD -i eth0 -s 0.0.0.0/8 -j DROP
iptables -A FORWARD -i eth0 -s 224.0.0.0/3 -j DROP
iptables -A FORWARD -i eth0 -s 10.0.0.0/8 -j DROP
iptables -A FORWARD -i eth0 -s 172.30.16.0/12 -j DROP
iptables -A FORWARD -i eth0 -s 192.168.0.0/16 -j DROP
iptables -A FORWARD -i eth1 -s ! 193.226.40.128/28 -j DROP
iptables -A FORWARD -d 255.255.255.255 -j DROP
iptables -A FORWARD -i eth0 -d 193.226.40.159 -j DROP
iptables -A INPUT -i eth0 -s 193.226.40.128/28 -j DROP
iptables -A INPUT -i eth0 -s 193.0.225.20 -j DROP
iptables -A INPUT -i eth0 -s 127.0.0.0/8 -j DROP
iptables -A INPUT -i eth0 -s 0.0.0.0/8 -j DROP
iptables -A INPUT -i eth0 -s 224.0.0.0/3 -j DROP
iptables -A INPUT -i eth0 -s 10.0.0.0/8 -j DROP
iptables -A INPUT -i eth0 -s 172.30.16.0/12 -j DROP
iptables -A INPUT -i eth0 -s 192.168.0.0/16 -j DROP
iptables -A INPUT -i eth1 -s ! 193.226.40.128/28 -j DROP
iptables -A INPUT -d 255.255.255.255 -j DROP
iptables -A INPUT -i eth0 -d 193.226.40.159 -j DROP
# celelalte restrictii
iptables -A INPUT -i eth1 -j ACCEPT
iptables -A FORWARD -i eth1 -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED -j ACCEPT
iptables -A INPUT -i eth0 -p udp --dport 1-1023 -j DROP
iptables -A INPUT -i eth0 -p udp --sport 53 -j ACCEPT
iptables -A INPUT -i eth0 -p udp --sport 123 --dport 123 -j ACCEPT
iptables -A FORWARD -d 193.226.40.130 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -d 193.226.40.130 -p tcp --dport 443 -j ACCEPT
iptables -A FORWARD -d 193.226.40.128/28 -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT
iptables -A INPUT -p icmp --icmp-type destination-unreachable \
-j ACCEPT
iptables -A INPUT -p icmp --icmp-type time-exceeded -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type echo-request -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type echo-reply -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type destination-unreachable \
-j ACCEPT
iptables -A FORWARD -p icmp --icmp-type time-exceeded -j ACCEPT
iptables -A INPUT -j DROP
iptables -A FORWARD -j DROP
```

10.7.2. Rețele private

O *rețea privată* este o rețea, conectată sau nu la Internet, a cărei calculatoare nu pot comunica direct cu calculatoarele din Internet.

O utilizare tipică este cea prezentată în figura 10.7. O instituție *A* are o rețea proprie de calculatoare. Din această rețea proprie, o parte dintre calculatoare — să le numim *publice* — trebuie să comunice nerestricționat cu alte calculatoare din Internet, în vreme ce restul calculatoarelor — le vom numi *private* — este acceptabil să poată comunica doar cu alte calculatoare din rețeaua internă.

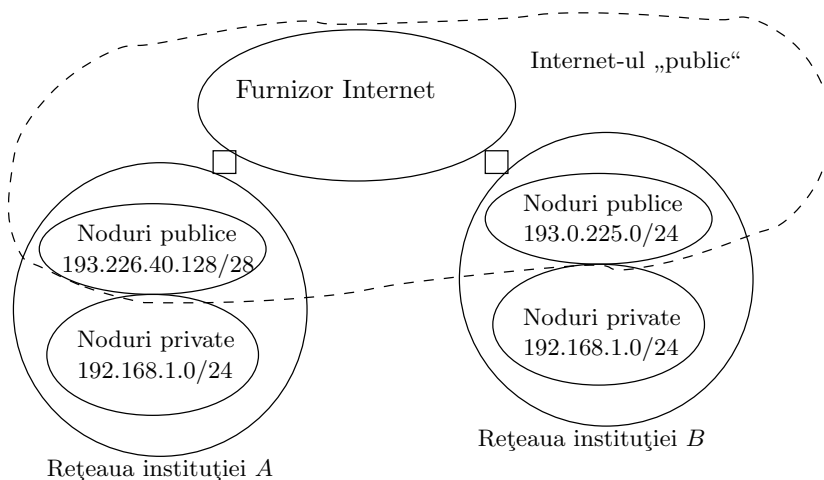


Figura 10.7: Două rețele locale având fiecare aceleași adrese private pentru o parte din calculatoare.

Calculatoarele private nu este necesar să aibă adrese unice în Internet. Adresele calculatoarelor private pot fi refolosite de către calculatoare private din alte astfel de rețele interne, de exemplu de cele ale instituției *B* din figură. De fapt, în general, o adresă trebuie să fie unică doar în mulțimea nodurilor cu care un anumit nod ar putea dori să comunice.

În situația descrisă, este necesar ca adresele din Internet-ul „public“ să fie unice, adresele din rețeaua locală să fie unice și să nu existe suprapuneri între adresele din Internet și adresele din rețeaua locală. Cerința din urmă este determinată de cerința ca nodurile cu adrese publice din rețeaua proprie să poată comunica și cu nodurile private și cu nodurile din Internet.

Un pachet a cărui adresă destinație este o adresă privată este dirijat de către rutere astfel:

- dacă ruterul face parte dintr-o rețea locală în care există acea adresă,

pachetul este dirijat către unicul nod din rețeaua locală purtând adresa respectivă;

- altfel, ruter-ul declară pachetul nelivrabil.

Așa cum am văzut în § 10.2.4.1, următoarele blocuri de adrese IP sunt alocate pentru astfel de utilizări în rețele private: 10.0.0.0/8, 172.16.0.0/12 și 192.168.0.0/16.

De cele mai multe ori, calculatoarele private li se oferă posibilități limitate de-a comunica cu calculatoare din Internet, prin intermediul unor mecanisme descrise în paragrafele următoare.

10.7.3. Translația adreselor (NAT)

Translația adreselor este un mecanism prin care un ruter modifică adresa sursă sau adresa destinație a unor pachete.

10.7.3.1. Translația adresei sursă

Presupunem că avem o rețea privată și dorim ca de pe calculatoarele cu adrese private să se poată deschide conexiuni către calculatoare din Internet — spre exemplu, pentru a putea accesa pagini web. Fără vreun mecanism special, acest lucru nu este posibil, din următorul motiv: Un calculator C cu adresă privată care dorește să deschidă o conexiune către un calculator S din Internet trimite un pachet IP având ca adresă sursă adresa proprie (privată) C , ca adresă destinație adresa serverului S (adresă care este publică) și conținând o cerere de deschidere de conexiune TCP. Pachetul ajunge la destinație, iar serverul S vom presupune că acceptă conexiunea. Serverul S trimite înapoi un pachet IP având ca adresă sursă adresa proprie S și ca adresă destinație adresa, privată, a clientului C . Deoarece adresa clientului nu este unică la nivelul Internet-ului (ci doar la nivelul propriei rețele interne), pachetul de răspuns nu poate fi livrat.

Translația adresei sursă rezolvă problema de mai sus în modul următor: În primul rând, trebuie să existe un nod (ruter) G în rețeaua internă, având adresă publică și prin care să tranziteze toate pachetele de la C către S (vezi fig. 10.8). Un pachet provenind de la C către S este modificat de către G , acesta punând adresa proprie G în locul adresei lui C . Serverul S primește pachetul ca provenind de la G și ca urmare răspunde cu un pachet (de acceptarea conexiunii) destinat lui G . Deoarece adresa lui G este publică, pachetul de răspuns ajunge la G . În acel moment, G trebuie să determine faptul că pachetul este răspuns la o cerere adresată de C . Ca urmare, G modifică adresa destinație a pachetului, punând adresa lui C în locul propriei adrese,

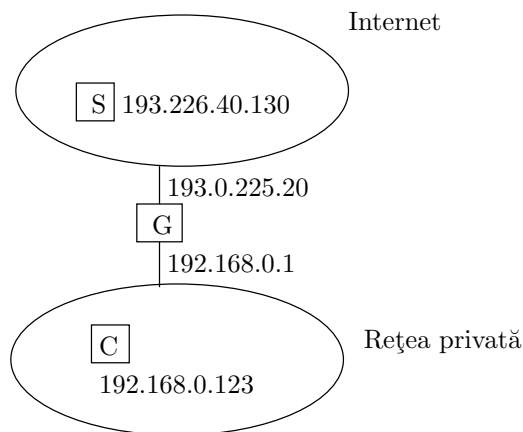


Figura 10.8: Rețea privată pentru exemplificarea mecanismului de translație a adresei sursă

după care trimite mai departe pachetul către C — acest lucru este acum posibil deoarece G este în rețeaua internă, și ca urmare adresa lui C indică singurul nod din rețeaua proprie având acea adresă.

Pentru ca mecanismul de mai sus să fie realizabil, este necesar ca ruterul G să poată determina cărui nod privat îi este destinat în mod real fiecare pachet având ca adresă destinație adresa G . De regulă, acest lucru necesită identificarea fiecărui pachet venit din Internet ca răspuns la un pachet dinspre un nod privat spre Internet.

Acest lucru este cel mai simplu de făcut pentru conexiunile TCP. Pentru o conexiune TCP, ruterul G urmărește pachetele de deschidere a conexiunii. La primirea unui pachet de deschidere a conexiunii dinspre un nod privat C , de pe un port p_c , către un server S , ruterul G alocă un port TCP local p_g (de preferință $p_g = p_c$, însă dacă p_c nu este liber se poate alocă un p_g diferit). Pachetul de deschidere a conexiunii este modificat de G astfel încât adresa sursă să fie G și portul sursă să fie p_g . G păstrează asocierea (C, p_c, p_g) . La sosirea unui pachet cu adresa destinație G și portul destinație p_g , ruterul G pune adresa destinație C și portul destinație p_c ; la primirea unui pachet cu adresa sursă C și portul sursă p_c ruterul G pune adresa sursă G și portul sursă p_g . Asocierea (C, p_c, p_g) este păstrată până la închiderea conexiunii, determinată prin schimbul corespunzător de pachete.

EXEMPLUL 10.16: Pentru rețeaua ilustrată în figura 10.8, presupunem că clientul C având adresa (privată) 192.168.0.123 deschide o conexiune TCP de pe portul efemer 3456 către serverul S , având adresa 193.226.40.130, pe

Sens	Între C și G		Între G și S	
	sursă	destinație	sursă	destinație
$C \rightarrow S$	192.168.0.123 port 3456 (p_c)	193.226.40.130 port 80	193.0.225.20 port 7890 (p_g)	193.226.40.130 port 80
$C \leftarrow S$	193.226.40.130 port 80	192.168.0.123 port 3456 (p_c)	193.226.40.130 port 80	193.0.225.20 port 7890 (p_g)

Tabelul 10.8: Adresele sursă și destinație ale pachetelor schimbate între clientul C și serverul S în exemplul 10.16

portul 80 (pentru a aduce o pagină web). Ruterul G având adresa publică 193.0.225.20 efectuează translația adresei sursă. Adresele pachetelor transmise între C și S sunt date în tabelul 10.8.

Pentru alte protocoale, asocierea este mai dificil de făcut. Pentru UDP, există noțiunea de port și ca urmare identificarea pachetelor primite de G pe baza portului destinație p_g este posibilă. Este însă dificil de determinat durata valabilității asocierii (C, p_c, p_g), întorcând nu există pachete de „închiderea conexiunii UDP“. Se poate însă utiliza un timp de expirare, de ordinul câtorva minute, de exemplu, în care dacă nu trec pachete asocierea este desfăcută. Pentru ICMP *ping* și *pong* există un număr de identificare care poate fi folosit cu același rol ca și un număr de port. Translația adreselor pentru astfel de pachete funcționează întocmai ca și în cazul UDP.

Mecanismul de translație a adresei sursă, descris mai sus, permite deschiderea unei conexiuni de la un nod cu adresă privată către un nod public din Internet. Nodul privat „are impresia“ că comunică direct cu serverul din Internet. Serverul din Internet „are impresia“ că comunică cu ruterul G pe portul alocat de acesta.

Față de utilizarea adreselor publice, utilizarea adreselor private și a translației adresei sursă are două limitări majore:

- nu permite deschiderea conexiunilor în sens invers, dinspre Internet către un nod privat;
- dacă pe conexiune sunt trimise, sub forma de date utile pentru protocolul TCP, informații privind adresa IP și portul de pe client, vor fi constatate incoerențe între IP-ul și portul clientului văzute de către server (acestea fiind G și respectiv p_g) și IP-ul și portul clientului văzute de client (acestea fiind C și respectiv p_c).

Cea de-a doua limitare poate fi eliminată dacă G cunoaște protocolul de nivel aplicație dintre C și S și modifică datele despre conexiune schimbate

între C și S . Prima limitare poate fi eliminată în măsura în care este vorba de conexiuni inițiate în urma unor negocieri pe o conexiune anterioară (de exemplu, conexiunile de date din protocolul FTP); pentru aceasta, G trebuie, din nou, să urmărească comunicația dintre C și S și să modifice datele privitoare la adresa și portul pe care clientul așteaptă conexiune dinspre server.

10.7.3.2. Translația adresei destinație

Presupunem că avem o rețea privată, un server S cu adresă privată, un ruter G situat în rețeaua proprie dar având adresă publică și un client C din Internet. Clientul C dorește să deschidă o conexiune către serverul S . Desigur, comunicarea „normală“ nu este posibilă deoarece în contextul lui C adresa privată a lui S nu este unică.

Este posibil însă ca adresa publicată (în DNS-ul public) pentru serverul S să fie adresa lui G . În acest caz, C deschide conexiunea către G . Printr-un mecanism similar cu cel din paragraful precedent, G modifică de data aceasta adresa destinație, punând, în locul propriei adrese, adresa privată a lui S . Pachetul de răspuns de la S este de asemenea modificat de către G , care pune ca adresă sursă adresa proprie G în loc de S . Astfel, S „are impresia“ că comunică direct cu C , iar C „are impresia“ că comunică cu G .

Translația adresei destinație poate fi utilă în următoarele situații:

- dacă avem o singură adresă publică și dorim să avem mai multe servere accesibile din exterior, servere ce nu pot funcționa pe aceeași mașină. Putem furniza astfel un server *HTTP* și un server *SMTP* care apar din Internet ca fiind la aceeași adresă IP dar sunt găzduite în realitate pe calculatoare distincte. Necesitatea utilizării calculatoarelor distincte pentru aceste servicii poate rezulta din motive de securitate sau din motive legate de performanțele calculatoarelor utilizate.
- dacă dorim totuși acces din afară către calculatoarele din rețeaua internă. De exemplu, pe fiecare calculator rulează un server *SSH*. Fiecărui calculator îi vom asocia un port pe ruterul G . O conexiune din afară, prin protocolul *SSH*, către un anumit port de pe G va fi redirectionată către serverul *SSH* de pe calculatorul privat corespunzător.
- pentru a distribui cererile către un server foarte solicitat. În acest caz, serverul va avea ca adresă publicată în DNS adresa lui G , însă vor exista de fapt mai multe servere pe calculatoare S_1, S_2, \dots, S_n în rețeaua internă. Conexiunile deschise din Internet către G vor fi redirectionate echilibrat către serverele S_1, S_2, \dots, S_n . Mai exact, la deschiderea unei conexiuni către G , G alege un server S_k către care redirectionează acea conexiune. Orice pachet ulterior de pe acea conexiune va fi redirectionat

către S_k . Conexiuni noi pot fi redirecționate către alte servere.

10.7.4. Tunelarea

Prin *tunelare* se înțelege, în general, transmiterea unor date aparținând unui anumit protocol ca date utile în cadrul unui protocol de același nivel sau de nivel superior.

Ne vom ocupa în cele ce urmează de tunelarea pachetelor IP, adică de transmiterea pachetelor IP prin protocoale de nivel rețea sau de nivel aplicație.

O situație în care este necesară tunelarea este aceea în care există două rețele private și se dorește ca nodurile din cele două rețele să poată comunica nerestricționat între ele. De exemplu, avem o instituție care are două sedii și are o rețea privată în fiecare sediu. Există mai multe soluții pentru a realiza legătura:

- *Translația adreselor* realizează o legătură supusă unor restricții, studiate în § 10.7.3.
- *Unificarea fizică* a celor două rețele private, ducând o legătură fizică între ele (fig. 10.9), oferă conectivitate completă, însă ducerea unui cablu special pentru aceasta poate fi extrem de costisitor.

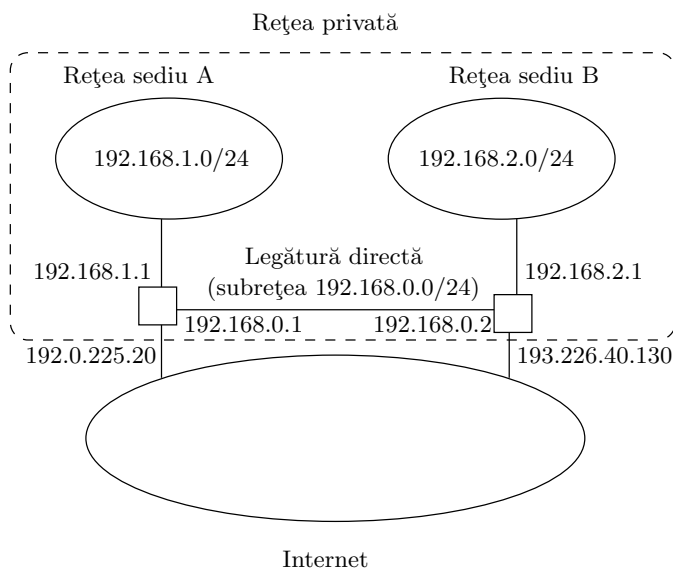


Figura 10.9: Unificarea rețelelor private printr-o legătură fizică directă.

- *Tunelarea* oferă conectivitate completă ca și legătura fizică folosind legăturile la Internet existente. Construcția constă în realizarea unei conexiuni (de exemplu TCP) între două rutere cu adrese publice din cele două rețele interne (fig. 10.10). Conexiunea dintre rutere este un „cablu virtual“ ce preia rolul conexiunii fizice.

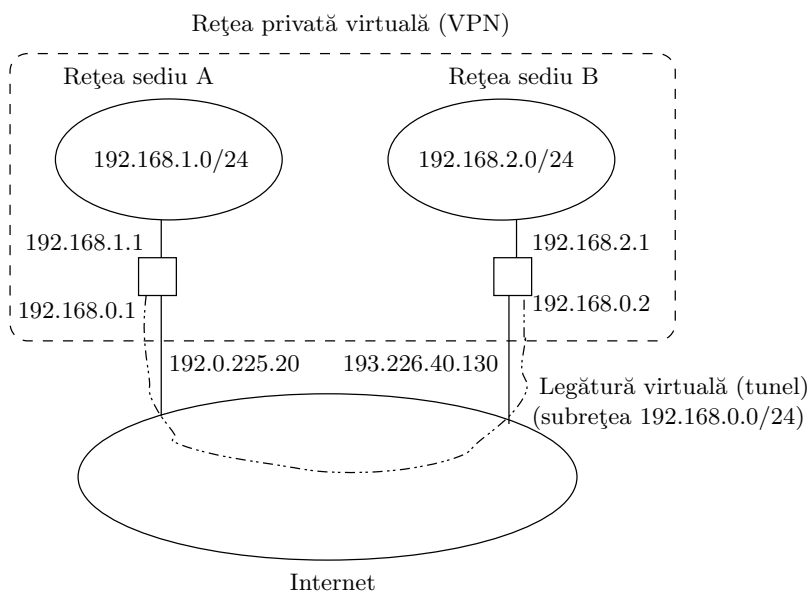


Figura 10.10: Unificarea rețelelor private printr-un tunel

Tunelul se prezintă față de nivelul rețea ca și când ar fi o legătură fizică. Ca urmare, fiecare capăt al tunelului este o interfață de rețea, având o adresă IP și o mască de rețea.

Pentru tunelarea propriu-zisă există mai multe protocoale. Unele dintre protocoale criptează pachetele tunelate; astfel de protocoale oferă securitatea unui cablu direct bine păzit.

Un tunel poate avea mai mult de două capete. Un tunel cu mai multe capete se comportă ca o subrețea cu mai multe interfețe conectate la ea — de exemplu o rețea Ethernet.

Capitolul 11

Aplicații în rețele

11.1. Poșta electronică

Poșta electronică servește la transferul de mesaje electronice între utilizatori aflați pe sisteme diferite. Protocolul a fost creat inițial pentru mesaje text și a fost modificat ulterior pentru a permite transferul de fișiere cu conținut arbitrar. Sistemul este conceput în ideea că este acceptabil ca transferul mesajului să dureze câteva ore, pentru a putea funcționa pe sisteme ce nu dispun de o legătură permanentă în rețea.

Poșta electronică este una dintre primele aplicații ale rețelelor de calculatoare și a fost dezvoltată în aceeași perioadă cu Internet-ul. Ca urmare, protocolul cuprinde prevederi create în ideea transferului prin alte mijloace decât o legătură prin protocol Internet.

Arhitectura sistemului cuprinde următoarele elemente (vezi fig. 11.1):

- Un proces de tip *mail user agent* (*MUA*), controlat de utilizatorul ce dorește expedierea mesajului. Acesta interacționează cu utilizatorul pentru a-l asista în compunerea mesajului și stabilirea adresei destinatarului. La final, *mail user agent*-ul trimite mesajul unui proces de tip *mail transfer agent* (*MTA*). Transferul este inițiat de *MUA*-ul utilizatorului expeditor și utilizează protocolul SMTP (§ 11.1.2.1).
- O serie de procese de tip *mail transfer agent* care trimit fiecare următorului mesajul. Transferul este inițiat de *MTA*-ul emițător și utilizează tot protocolul SMTP.
- De fiecare adresă destinație este răspunzător un *mail transfer agent* care memorează mesajele destinate acelei adrese. Odată mesajul ajuns la

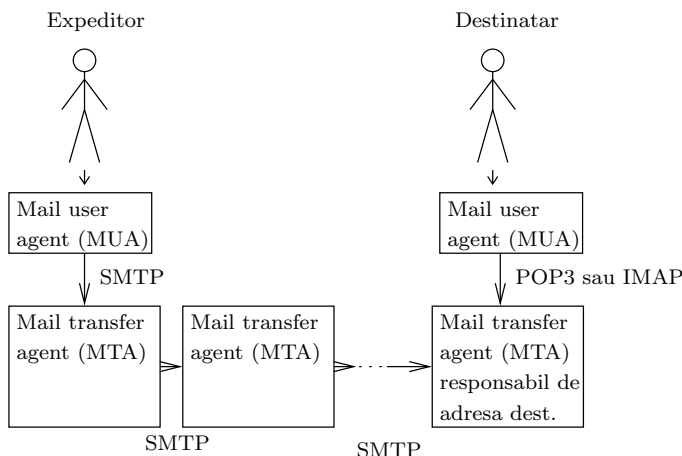


Figura 11.1: Elementele sistemului de transmitere a poştei electronice. Săgeţile arată sensul în care se iniţiază comunicaţia (de la client spre server), nu sensul în care se transferă mesajul de poştă electronică.

mail transfer agent-ul răspunzător de adresa destinaţie, el este memorat local (afară de cazul în care are loc aici o rescriere de adresă, vezi § 11.1.2.3).

- Utilizatorul destinaţie citeşte mesajul cu ajutorul unui proces de tip *mail user agent*. Acesta contactează *mail transfer agent*-ul responsabil de adresa utilizatorului destinaţie şi recuperează mesajul de la el. Transferul este iniţiat de *MUA* (adică de receptor). Există două protocoale utilizate pentru transfer: POP3 şi IMAP.

11.1.1.1. Formatul mesajelor

Formatul mesajelor este definit în [RFC 2822, 2001] (care înlocuieşte „clasicul“ [RFC 822, 1982]). Fiecare mesaj începe cu un antet cuprinzând adresa expeditorului, adresa destinatarului, data şi alte câteva informaţii. După antet urmează corpul mesajului, care conţine mesajul propriu-zis.

Întreg mesajul este de tip text ASCII. Rândurile sunt delimitate prin secvenţe formate din caracterul *carriage return* (cod 13) urmat de *line feed* (cod 10). Rândurile nu au voie să aibă lungime mai mare de 998 caractere şi se recomandă să nu aibă mai mult de 78 de caractere. Caracterele de control (cu codul între 0 şi 31 sau egal cu 127) nu sunt permise, cu excepţia secvenţelor *carriage return* – *line feed* care separă rândurile. În particular, caractere *carriage return* izolate sau caractere *line feed* izolate nu sunt permise.

Pentru a permite transmiterea mesajelor prin linii seriale incapabile să transmită caractere de 8 biți (ci doar caractere de 7 biți), este recomandabil ca mesajele să nu conțină caractere cu codul între 128 și 255.

Datorită unor protocoale folosite pentru transmiterea și pentru stocarea mesajelor, se impun următoarele restricții suplimentare asupra conținutului mesajelor:

- nici un rând să nu constea doar dintr-un caracter punct;
- un rând ce urmează după un rând vid să nu înceapă cu cuvântul **From**.

De menționat că în protocolul de comunicație dintre două *mail transfer agent*-uri sunt transferate informații privind adresa expeditorului și adresa destinatarului, independente de cele plasate în antetul mesajului. Aceste informații formează așa-numitul *plic* (engl. *envelope*) al mesajului. Expeditorul și destinatarul date în antetul mesajului sunt informații pentru utilizatorul uman; informațiile de pe *plic* sunt cele utilizate efectiv în transmiterea mesajului.

11.1.1.1. Antetul mesajelor

Antetul mesajelor este constituit dintr-un număr de *câmpuri*, fiecare câmp având un *nume* și o *valoare*. De principiu, fiecare câmp este un rând separat conținând numele, caracterul *două puncte* (:) și valoarea; secvența *carriage return* urmat de *line feed* acționează ca separator între câmpuri. Antetul se termină cu două secvențe *carriage return – line feed* consecutive.

Dacă un câmp este prea lung pentru a încapa într-un rând (standardul recomandă ca rândurile să nu depășească 78 de caractere și interzice rândurile mai lungi de 998 caractere), câmpul poate fi continuat pe rândurile următoare, care trebuie să înceapă cu spațiu sau *tab*; spațiile și *tab*-urile de la începutul rândurilor următoare se consideră ca făcând parte din câmp.

EXEMPLUL 11.1: Un posibil document (vezi mai jos explicațiile privind semnificațiile diverselor câmpuri):

```
From: Radu Lupsa <rlupsa@cs.ubbcluj.ro>
To: Test User <test@example.com>
Date: Sat, 1 Sep 2007 10:12:20 +0300
Message-ID: my-emailer.20070901101220.53462@nessie.cs.ubbcluj.ro
Subject: Un mesaj dat ca
        exemplu
```

Salut,
Mesajul acesta este doar un exemplu.

Principalele câmpuri ce pot apare într-un mesaj sunt:

- **To**, **Cc** și **Bcc** reprezintă adresele la care trebuie livrat mesajul.

Adresele din câmpul **To** reprezintă persoanele cărora le este adresat mesajul.

Adresele din câmpul **Cc** reprezintă persoane ce trebuie informate de trimiterea mesajului; de exemplu, în corespondența oficială între un angajat al unei firme și un client al firmei, angajatul va pune adresa clientului în câmpul **To** (deoarece acestuia îi este destinat mesajul), iar în câmpul **Cc** va pune adresa șefului (care trebuie informat cu privire la comunicație).

Adresele din câmpul **Bcc** reprezintă persoane cărora le va fi livrat mesajul, fără ca ceilalți destinatari să fie informați despre aceasta. Câmpul **Bcc** este completat de către *mail user agent* și este eliminat de către primul *mail transfer agent* de pe traseu.

- **From**, **Sender** și **Reply-to** reprezintă adresa expeditorului și adresa la care trebuie răspuns.

În condiții obișnuite, un mesaj conține doar câmpul **From**, reprezentând adresa expeditorului mesajului și totodată adresa la care trebuie trimis un eventual răspuns.

Câmpul **Sender** este utilizat atunci când o persoană trimite un mesaj în numele altei persoane sau în numele unei organizații pe care o reprezintă. Există două situații practice care conduc la această situație: Dacă mesajul provine de la șef, dar este scris și trimis efectiv de secretara șefului, atunci adresa șefului este pusă în câmpul **From**, iar adresa secretarei în câmpul **Sender**. Dacă o persoană trimite un mesaj în numele unei organizații, atunci adresa organizației va fi trecută în câmpul **From** și adresa persoanei ce scrie mesajul va fi plasată în câmpul **Sender**.

În fine, câmpul **Reply-to** reprezintă adresa la care trebuie trimis un eventual răspuns la mesaj, dacă această adresă este diferită de adresa din câmpul **From**. Dacă destinatarul răspunde la un mesaj primit, utilizând funcționalitatea de *reply* a *mail user agent*-ului său, *MUA*-ul oferă implicit, ca adresă destinație a mesajului de răspuns, adresa preluată din câmpul **Reply-to** a mesajului original. În lipsa unui câmp **Reply-to**, *MUA*-ul oferă adresa din câmpul **From**. De asemenea, chiar în prezența unui câmp **Reply-to**, este bine ca *MUA*-ul să ofere posibilitatea de-a trimite răspunsul la adresa din câmpul **From**.

Un exemplu de utilizare a câmpului **Reply-to** este următorul: un mesaj adresat unei liste de discuții are ca **From** adresa autorului mesajului și ca **To** adresa listei. Mesajul se transmite centrului de

distribuție al listei (un *mail transfer agent*), care retransmite mesajul către abonații listei. Mesajul retransmis către abonați va avea adăugat un câmp **Reply-to** indicând adresa listei. Astfel, mesajul primit de abonat are **From** autorul mesajului, **To** adresa listei și **Reply-to** tot adresa listei. Abonatul listei va răspunde foarte ușor pe adresa listei deoarece *mail user agent*-ul său va prelua adresa listei din **Reply-to** și o va pune ca adresă destinație în mesajul de răspuns. Totuși, e bine ca utilizatorul să nu folosească orbește această facilitate, întrucât uneori răspunsul este bine să ajungă doar la autorul mesajului original, nu la toată lista...

- **Received** și **Return-path** au ca rol diagnosticarea sistemului de livrare a mesajelor. Fiecare *mail transfer agent* de pe traseul mesajului adaugă în fața mesajului un câmp **Received** în care scrie numele mașinii sale, numele și adresa IP a *mail transfer agent*-ului care i-a trimis mesajul, data și ora curentă și emițătorul și destinatarul mesajului conform cererii *mail transfer agent*-ului care îi transmite mesajul (cei indicați pe „plicul” mesajului, nu cei din câmpurile **To** sau **From**). Astfel, un mesaj începe cu un șir de antete **Received** conținând, în ordine inversă, nodurile prin care a trecut mesajul.

Ultimul *mail transfer agent* adaugă un câmp **Return-path**, conținând adresa sursă a mesajului conform *plicului* (datele furnizate de MTA-ul sursă).

Câmpurile **Received** și **Return-path** sunt utilizate pentru a returna un mesaj la autorul său în cazul în care apar probleme la livrarea mesajului. De notat diferența între **Reply-to**, care reprezintă destinatarul unui eventual răspuns dat de utilizator la mesaj, și **Return-path**, care reprezintă destinatarul unui eventual mesaj de eroare.

- **Date** reprezintă data generării mesajului. Este în mod normal completat de *mail user agent*-ul expeditorului mesajului. Are un format standard, cuprinzând data și ora locală a expeditorului precum și indicativul fusului orar pe care se găsește expeditorul. Formatul cuprinde: ziua din săptămână (prescurtare de trei litere din limba engleză), un caracter virgulă, ziua din lună, luna (prescurtarea de trei litere din limba engleză), anul, ora, un caracter *două puncte*, minutul, opțional încă un caracter *două puncte* urmat de secundă și în final indicativul fusului orar. Indicativul fusului orar cuprinde diferența, în ore și minute, între ora locală și ora universală coordonată (*UTC*; vezi § 7.3.1 pentru detalii). Faptul că ora locală este oră de vară sau nu apare în indicativul de fus orar. România are în timpul iernii ora locală egală cu *UTC+2h*,

iar în timpul verii UTC+3h.

- **Subject** reprezintă o scurtă descriere (cât mai sugestivă) a mesajului, dată de autorul mesajului.
- **Message-ID**, **In-reply-to**, **Reference** servesc la identificarea mesajelor. Valoarea câmpului **Message-ID** este un șir de caractere care identifică unic mesajul. El este construit de către *mail user agent*-ul expeditorului pornind de la numele calculatorului, de la ora curentă și de la niște numere aleatoare, astfel încât să fie extrem de improbabil ca două mesaje să aibă același **Message-ID**.

În cazul răspunsului la un mesaj prin funcția *reply* a *mail user agent*-ului, mesajul de răspuns conține un câmp **In-reply-to** având ca valoare **Message-ID**-ul mesajului la care se răspunde. Valoarea câmpului **Reference** se crează din câmpurile **Reference** și **Message-ID** ale mesajului la care se răspunde. Câmpurile **Reference** și **Message-ID** pot fi folosite de exemplu pentru afișarea, de către *mail user agent*-ul destinație, a succesiunilor de mesaje legate de o anumită problemă și date fiecare ca replică la precedentul.

- **Resent-from**, **Resent-sender**, **Resent-to**, **Resent-cc**, **Resent-bcc**, **Resent-date** și **Resent-msg-id** sunt utilizate dacă destinatarul unui mesaj dorește să retrimită mesajul, fără modificări, către altcineva. O astfel de retrimiteră se poate efectua printr-o comandă *bounce* sau *re-send* a *MUA*-ului. În acest caz, câmpurile din antetul mesajului original (inclusiv **From**, **To** sau **Date**) sunt păstrate, reflectând expeditorul, destinatarul și data trimiterii mesajului original. Pentru informațiile privind retransmiterea (adresa utilizatorului ce efectuează retransmiterea, destinatarul mesajului retransmis, data retransmiterii, etc.), se utilizează câmpurile **Resent-...** enumerate mai sus. Semnificația fiecăruia dintre aceste câmpuri **Resent-...** este identică cu semnificația câmpului fără **Resent-** corespunzător, dar se referă la retransmitere, nu la mesajul original.

11.1.1.2. Extensii MIME

Standardul original pentru mesaje de poștă electronică (rfc 822) a suferit o serie de extensii. Acestea sunt cunoscute sub numele *Multipurpose Internet Mail Extension (MIME)* și sunt descrise în [RFC 2045, 1996], [RFC 2046, 1996] și [RFC 2047, 1996].

Extensiile MIME servesc în principal pentru a putea transmite fișiere atașate unui mesaj de poștă electronică.

Un mesaj conform standardului *MIME* trebuie să aibă un câmp în antet cu numele `Mime-version`. Valoarea câmpului este versiunea standardului *MIME* în conformitate cu care a fost creat mesajul.

11.1.1.3. Atașarea fișierelor și mesaje din mai multe părți

Standardul *MIME* oferă posibilitatea atașării unor fișiere la un mesaj de poștă electronică. Mecanismul se încadrează într-unul mai general, care permite formarea unui mesaj din mai multe părți. Un mesaj cu atașamente este dat în exemplul 11.2.

Fiecare mesaj are în antet un câmp, `Content-type`, care arată ce tip de date conține și în ce format sunt ele reprezentate. Exemple de tipuri sunt: `text/plain` (text normal), `text/html` (document HTML), `image/jpeg` (imagine în format *JPEG*), etc.

De regulă, partea din fața caracterului *slash* (/) arată tipul de document, iar partea a doua arată formatul (codificarea) utilizată. Astfel, o imagine va avea `Content-type` de forma `image/format`; de exemplu, `image/gif`, `image/jpeg`, etc.

Mesajele ce conțin doar text obișnuit trebuie să aibă `Content-type: text/plain`. Acesta este de altfel tipul implicit în cazul absenței câmpului `Content-type`.

Mesajele cu atașamente au `Content-type: multipart/mixed`. În general, un mesaj de tip `multipart/subtip` este format de fapt din mai multe „fișiere“ (oarecum ca un fișier arhivă *zip*). Într-un mesaj `multipart/mixed`, de obicei una dintre părți este mesajul propriu-zis, iar fiecare dintre celelalte părți este câte un fișier atașat.

Fiecare parte a unui mesaj `multipart` are un antet și un corp, similar cu un mesaj de sine stătător. Antetul părții poate conține doar câmpuri specifice *MIME*. Astfel, fiecare parte are propriul tip, care poate fi în particular chiar un `multipart`.

Cele mai importante subtipuri ale tipului `multipart` sunt:

- `multipart/mixed` înseamnă pur și simplu mai multe componente puse împreună, ca un fișier arhivă.
- `multipart/alternative` arată că părțile sunt variante echivalente ale aceluiași document, în formate diferite.

Separarea părților unui mesaj de tip `multipart` se face printr-un rând ce conține un anumit text, ce nu apare în nici una dintre părțile mesajului. Textul utilizat ca separator este plasat în câmpul `Content-type` după `multipart/subtip`. Este scris sub forma (utilizabilă și în alte câmpuri și pen-

tru alte informații) unui șir `boundary=șir` situat după șirul `multipart/subtip` și separat prin punct și virgulă față de aceasta. Exemplu:

```
Content-type: multipart/mixed; boundary="abcdxxxx"
```

Corpul mesajului `multipart` este separat după cum urmează:

- în fața primei părți precum și între fiecare două părți consecutive se găsește un rând format doar din textul de după `boundary=` precedat de două caractere minus (`--`);
- după ultima parte se găsește un rând format doar din textul de după `boundary=`.

Fiecare parte a unui mesaj de tip `multipart/mixed` are un câmp `Content-disposition` [RFC 2183, 1997]. Valoarea acestui câmp arată ce trebuie să facă *mail user agent*-ul destinație cu partea de mesaj în care se găsește acest antet. Valori posibile sunt:

- `inline` arată că mesajul sau partea de mesaj trebuie să fie afișată utilizatorului;
- `attachment` arată că mesajul sau partea de mesaj nu trebuie afișată decât la cerere. Un astfel de câmp poate avea în continuare o informație suplimentară, `filename=nume`, arată numele sugerat pentru salvarea părții respective. De notat că, din motive de securitate, *mail user agent*-ul destinație trebuie să nu salveze orbește partea de mesaj sub numele extras din mesaj, ci să ceară mai întâi permisiunea utilizatorului. În caz contrar, un adversar poate să trimită un fișier având atașat un anumit fișier, cu care să suprascrie un fișier al destinatarului.

11.1.1.4. Codificarea corpului mesajului și a atașamentelor

Standardul original al formatului mesajelor prevede că mesajele conțin doar text ASCII, cu utilizare restricționată a caracterelor de control. Singurele caractere de control (cele cu codurile între 0 și 31) admise sunt *carriage return* (cod 13) și *line feed* (cod 10), utilizate pentru separarea rândurilor din mesaj. De asemenea, se recomandă să nu se utilizeze caracterele cu coduri între 128 și 255, datorită imposibilității transmisiei lor în unele sisteme.

Ca urmare, transmiterea unui fișier arbitrar (inclusiv a unui text ISO-8859) nu este posibilă direct.

Transmiterea unui conținut arbitrar se face printr-o recodificare a acestuia utilizând doar caracterele permise în corpul mesajului. Ca urmare,

mesajele apar, față de *mail transfer agent*-uri, identice cu cele conforme formatului original. Un *mail user agent* vechi, conform standardului original, nu va fi capabil să transmită un mesaj cu extensii MIME, iar în cazul primirii unui astfel de mesaj îl va afișa într-un mod mai puțin inteligibil pentru utilizator.

Recodificarea este aplicată doar asupra corpului mesajului, nu și asupra antetului. Antetul conține un câmp, **Content-transfer-encoding**, a cărui valoare arată dacă și ce recodificare s-a aplicat asupra conținutului. Codificările definite de [RFC 2045, 1996] sunt:

- **7bit** — înseamnă de fapt lipsa oricărei recodificări. În plus, arată că mesajul nu conține decât caractere ASCII (cu codurile cuprinse între 0 și 127).
- **8bit** — arată că mesajul nu a fost recodificat, dar poate conține orice caractere (cu coduri între 0 și 255).
- **quoted-printables** — arată că fiecare caracter de control și fiecare caracter egal (=) a fost recodificat ca o secvență de trei caractere, formată dintr-un caracter egal (=) urmat de două cifre hexa; acestea din urmă reprezintă codul caracterului original. De exemplu, caracterul egal se recodifică =3D, iar caracterul *escape* (cod 27) se recodifică =1B. Restul caracterelor pot fi scrise direct sau pot fi recodificate ca și caracterele speciale; de exemplu litera *a* poate fi scrisă *a* sau =61.
- **base64** — corpul mesajului a fost recodificat în baza 64 (vezi § 7.4.2).

În lipsa vreunui antet **Content-transfer-encoding**, se presupune codificarea **7bit**.

Pentru un mesaj (sau o parte de mesaj) de tip **multipart**, mesajul (respectiv partea) nu este permis să fie codificat decât **7bit** sau **8bit**, însă fiecare parte a unui **multipart** poate fi codificată independent de restul. În mod curent, un mesaj cu atașamente are corpul mesajului codificat **7bit**, partea corespunzătoare mesajului propriu-zis este codificată **7bit** sau **quoted-printables**, iar părțile corespunzătoare atașamentelor sunt codificate **base64**.

EXEMPLUL 11.2: Un mesaj cu fișiere atașate este dat în continuare:

```
From: Radu Lupsa <rlupsa@cs.ubbcluj.ro>
To: Test User <test@cs.ubbcluj.ro>
Date: Sat, 1 Sep 2007 10:12:20 +0300
Message-ID: my-emailer.20070901101220.53462@nessie.cs.ubbcluj.ro
Subject: Un mesaj cu fișiere atasate
MIME-Version: 1.0
Content-transfer-encoding: 7bit
Content-type: multipart/mixed; boundary="qwertyuiop"
```

```
--qwertyuiop
Content-type: text/plain
Content-transfer-encoding: 7bit
Content-disposition: inline

Acesta este mesajul propriu-zis.
--qwertyuiop
Content-type: application/octet-stream
Content-disposition: attachment; filename="test.dat"
Content-transfer-encoding: base64

eAAXLRQ=
--qwertyuiop
Content-type: text/plain; charset=utf-8
Content-disposition: attachment; filename="test.txt"
Content-transfer-encoding: quoted-printables

=C8=98i =C3=AEnc=C4=83 un text.
qwertyuiop
```

11.1.2. Transmiterea mesajelor

Așa cum am văzut, mesajele sunt transmise din aproape în aproape, fiecare mesaj parcurgând un lanț de *MTA*-uri. Fiecare *MTA*, cu excepția ultimului, determină următorul *MTA* și-i pasează mesajul.

11.1.2.1. Protocolul SMTP

Protocolul utilizat pentru transmiterea mesajelor de la un *MTA* la următorul este protocolul *Simple Mail Transfer Protocol* — *SMTP*. Este un protocol de tip text, construit de obicei peste o conexiune TCP.

Rolul de client SMTP îl are *MTA*-ul ce are mesajul de trimis mai departe; rolul de server aparține *MTA*-ului ce primește mesajul. Clientul deschide o conexiune TCP către portul 25 al serverului.

După deschiderea conexiunii, serverul trimite un mesaj conținând un cod de răspuns urmat de numele serverului. În continuare, clientul trimite câte o cerere, la care serverul răspunde cu un cod ce arată dacă cererea a fost executată cu succes sau nu, urmat de un text explicativ. Cererile sunt formate de regulă dintr-un cuvânt-cheie urmat de eventuali parametri și se încheie printr-o secvență *carriage return – line feed*. Răspunsurile sunt formate dintr-un număr, transmis ca secvență de cifre, urmat de un text explicativ. Numărul

arată, într-o formă ușor de procesat de către calculator, dacă cererea s-a executat cu succes sau nu și cauza erorii. Textul cuprinde informații suplimentare pentru diagnosticarea manuală a transmiției mesajelor.

Clientul trebuie să înceapă printr-o cerere **HELO** care are ca parametru numele mașinii clientului. Prin aceasta clientul se identifică față de server. De notat că nu există posibilitatea autentificării clientului de către server; serverul este nevoit să „ia de bun“ numele transmis de client.

După identificarea prin cererea **HELO**, clientul poate transmite serverului mai multe mesaje de poștă electronică.

Transmiterea fiecărui mesaj va începe printr-o cerere **MAIL FROM:**, având ca parametru adresa expeditorului mesajului. După acceptarea de către server a comenzii **MAIL FROM:**, clientul va trimite una sau mai multe cereri **RCPT TO:**; fiecare cerere are ca parametru o adresă destinație. Fiecare cerere **RCPT TO:** poate fi acceptată sau refuzată de către server independent de celelalte. Serverul va transmite mesajul fiecăreia dintre adresele destinație acceptate. În final, clientul trimite o cerere **DATA** fără parametrii. Serverul răspunde, în mod normal cu un cod de succes. În caz de succes, clientul trimite corpul mesajului, încheiat cu un rând pe care se găsește doar caracterul punct. După primirea corpului mesajului, serverul trimite încă un răspuns.

Mesajele primite de MTA sunt plasate într-o coadă de așteptare, stocată de obicei în fișiere pe disc. MTA-ul receptor încearcă imediat să trimită mai departe fiecare mesaj primit. Dacă trimiterea mai departe nu este posibilă imediat, mesajul este păstrat în coadă și MTA-ul reîncearcă periodic să-l trimită. După un număr de încercări eșuate sau în cazul unei erori netempore (de exemplu, dacă nu există adresa destinație), MTA-ul abandonează și încearcă trimiterea unui mesaj (e-mail) de eroare înapoi către expeditor.

Dacă adresa destinație este locală, MTA-ul plasează mesajul în fișierul corespunzător cutiei poștale a destinatarului.

De notat că în trimiterea mai departe, livrarea locală sau trimiterea unui mesaj de eroare, MTA-ul utilizează doar informațiile de pe *plic*, adică parametrii comenzilor **MAIL FROM:** și **RCPT TO:**, și nu valorile câmpurilor **From** sau **To** din antetul mesajului.

EXEMPLUL 11.3: Fie mesajul din exemplul 11.1. Transmiterea lui de la MTA-ul de pe `cs.ubbcluj.ro` către `example.com` decurge astfel (rândurile transmise de la `cs.ubbcluj.ro` la `example.com` sunt precedate de o săgeată la dreapta, iar rândurile transmise în sens invers de o săgeată la stânga):

← 220 `example.com`

→ **HELO** `nessie.cs.ubbcluj.ro`

```

← 250 example.com
→ MAIL FROM: <rlupsa@cs.ubbcluj.ro>
← 250 Ok
→ RCPT TO: <test@example.com>
← 250 Ok
→ DATA
← 354 End data with <CR><LF>.<CR><LF>
→ From: Radu Lupsa <rlupsa@cs.ubbcluj.ro>
→ To: Test User <test@example.com>
→ Date: Sat, 1 Sep 2007 10:12:20 +0300
→ Message-ID: my-emailer.20070901101220.53462@nessie.cs.ubbcluj.ro
→ Subject: Un mesaj dat ca
→ exemplu
→ Salut,
→ Mesajul acesta este doar un exemplu.
→ .
← 250 Ok: queued
→ QUIT
← 221 Bye

```

EXEMPLUL 11.4: Următorul mesaj ilustrează utilizarea câmpurilor în cazul unui mesaj transmis unei liste de utilizatori. Mesajul este reprodus așa cum ajunge la un abonat al listei, având adresa `test@example.com`. Mesajul ilustrează, de asemenea, utilizarea câmpului `Sender` de către cineva care transmite un mesaj în numele altcuiva și a câmpului `Cc`.

```

Return-path: errors-26345@comunitati-online.example
Received: from server27.comunitati-online.example
  by example.com for test@example.com; 31 Aug 2007 22:09:23 -0700
Reply-to: Forumul OZN <ozn@comunitati-online.example>
Received: from roswell.greenmen.example
  by server27.comunitati-online.example
  for ozn@comunitati-online.example; 1 Sep 2007 05:09:21 +0000
Received: from localhost by roswell.greenmen.example
  for ozn@comunitati-online.example; 1 Sep 2007 08:09:20 +0300
From: Organizatia omuletilor verzi <office@greenmen.example>
Sender: Ion Ionescu <ion@greenmen.example>
To: Forumul OZN <ozn@comunitati-online.example>
Cc: Organizatia omuletilor verzi <office@greenmen.example>
Date: Sat, 1 Sep 2007 10:12:20 +0300
Message-ID: my-emailer.20070901101220.534@roswell.greenmen.example
In-reply-to: my-emailer.20070830222222.321@ufo.example

```

Subject: Re: Incident

Organizatiei omuletilor verzi anunta ca nu a avut nici un amestec in incidentul de la balul anual E.T.

Ion Ionescu,
Presedintele Organizatiei omuletilor verzi

11.1.2.2. Determinarea următorului MTA

Un MTA care are un mesaj de transmis către o anumită adresă determină următorul MTA căruia trebuie să-i transmită mesajul astfel:

1. MTA-ul caută mai întâi, printre informațiile sale de configurare (vezi § 11.1.2.3), dacă are vreo regulă privind adresa destinație. Dacă MTA-ul este responsabil de adresa destinație a mesajului, îl memorează local. Dacă MTA-ul este poartă de intrare a mesajelor pentru MTA-urile din rețeaua locală, transmite mesajul către MTA-ul determinat conform configurării.
2. Dacă nu există informații de configurare pentru adresa destinație, MTA-ul caută în DNS o înregistrare cu tipul MX pentru numele de domeniu din adresa destinație. O astfel de înregistrare conține una sau mai multe nume de servere SMTP capabile să preia mesajele destinate unei adrese din acel domeniu. Dacă găsește o astfel de înregistrare, MTA-ul contactează una din mașinile specificate în înregistrările MX găsite și-i transmite mesajul.
3. Dacă nu există nici înregistrări MX, MTA-ul contactează mașina cu numele de domeniu din adresa destinație și-i transmite mesajul.
4. Dacă nu există nici un server cu numele de domeniu din adresa destinație, adică dacă toate cele trei variante de mai sus eșuează, atunci MTA-ul declară că mesajul este nelivrabil și transmite înapoi spre emițător un mesaj de eroare.

Un MUA lucrează, de obicei, mult mai simplu. Acest lucru duce la simplificarea MUA-ului prin separarea clară a rolurilor: MUA-ul trebuie să ofere facilități de editare și să prezinte utilizatorului o interfață prietenoasă, iar MTA-ul are toate complicațiile legate de livrarea mesajelor. Pentru transmiterea oricărui mesaj, un MUA contactează un același MTA, a cărui adresă este configurată în opțiunile MUA-ului. Pe sistemele de tip UNIX, MUA-urile contactează implicit MTA-ul de pe mașina locală (`localhost`).

11.1.2.3. Configurarea unui MTA

De cele mai multe ori, un MTA este responsabil de adresele utilizatorilor calculatoarelor dintr-o reţea locală. În acest caz, MTA-ul memorează local mesajele adresate acestor utilizatori şi le oferă acestora posibilitatea de a le citi prin IMAP sau POP3. De asemenea, MTA-ul preia şi transmite spre exterior mesajele utilizatorilor, generate de MUA-urile ce rulează pe calculatoarele din reţeaua locală.

În configuraţii mai complicate, un MTA acţionează ca punct de trecere pentru mesajele care pleacă sau sosesc la un grup de MTA-uri dintr-o reţea locală. El preia mesajele de la toate MTA-urile din reţeaua locală în scopul retransmiterii lor către exterior. De asemenea, preia mesajele din exterior destinate tuturor adreselor din reţeaua locală şi le retrimite MTA-urilor, din reţeaua locală, responsabile. Un astfel de MTA este numit *mail gateway*. Un *mail gateway* poate fi util ca unic filtru contra viruşilor şi spam-urilor pentru o întregă reţea locală (în opoziţie cu a configura fiecare MTA din reţeaua locală ca filtru).

Un MTA trebuie configurat cu privire la următoarele aspecte:

- care sunt adresele locale şi cum se livrează mesajele destinate acestor adrese;
- care sunt maşinile (în principiu, doar din reţeaua locală) de la care se acceptă mesaje spre a fi trimise mai departe spre Internet;
- ce transformări trebuie aplicate mesajelor.

Implicit, un MTA consideră ca fiind adrese locale acele adrese care au numele de domeniu identic cu numele maşinii pe care rulează MTA-ul şi având partea de utilizator identică cu un nume de utilizator al maşinii locale. Pe sistemele de operare de tip UNIX, un mesaj adresat unui utilizator local este adăugat la finalul unui fişier având ca nume numele utilizatorului şi situat în directorul `/var/mail`.

MTA-ul responsabil de o anumită adresă destinaţie poate fi configurat de către utilizatorul destinatar să execute anumite prelucrări asupra mesajului primit. Pe sistemele de tip UNIX, această configurare se face prin directive plasate în fişierele `.forward` şi `.procmailrc` din directorul personal al utilizatorului.

Fişierul `.forward` conţine un şir de adrese la care trebuie retrimis mesajul *în loc* să fie livrat local în `/var/mail/user`.

Fişierul `.procmailrc` cuprinde instrucţiuni mai complexe de procesare a mesajelor primite: în funcţie de apariţia unor şiruri, descrise prin expresii regulate, în mesajul primit, mesajul poate fi plasat în fişiere specificate

în `.procmailrc` sau poate fi pasat unor comenzi care să-l proceseze.

Pentru cazuri mai complicate, un MTA poate fi configurat de către administrator să execute lucruri mai complicate:

- Este posibil să se configureze adrese de poștă, situate în domeniul numelui mașinii locale, care să fie considerate valide chiar dacă nu există utilizatori cu acele nume. Pentru fiecare astfel de adresă trebuie definită o listă de adrese, de regulă locale, la care se va distribui fiecare mesaj primit. Ca exemplu, adresa `root@cs.ubbcluj.ro` este configurată în acest fel; un mesaj trimis la această adresă nu este plasat în `/var/mail/root` ci este retrimis utilizatorilor (configurați în fișierul `/etc/aliases`) care se ocupă de administrarea sistemului.
- Un MTA poate fi configurat să se considere responsabil de mai multe domenii, ale căror nume nu au nimic comun cu numele mașinii MTA-ului. De exemplu, se poate configura MTA-ul de pe `nessie.cs.ubbcluj.ro` să accepte mesajele destinate lui `ion@example.com` și să le livreze utilizatorului local `gheorghe`. Desigur, pentru ca un mesaj destinat lui `ion@example.com` să poată fi livrat, mai trebuie ca mesajul să ajungă până la `nessie.cs.ubbcluj.ro`. Pentru aceasta, trebuie ca MUA-ul expeditor sau un MTA de pe traseu să determine ca următor MTA mașina `nessie.cs.ubbcluj.ro`, lucru care se întâmplă dacă în DNS se pune o înregistrare MX pentru numele de domeniu `example.com` indicând către `nessie.cs.ubbcluj.ro`. Un astfel de mecanism este utilizat în mod curent de furnizorii de servicii Internet pentru a găzdui poșta electronică a unor clienți care au nume propriu de domeniu dar nu dețin servere de poștă electronică care să preia poșta adresată adreselor din domeniul respectiv.
- Dacă utilizatorii expediază mesaje de pe mai multe mașini dintr-o rețea locală, nu este de dorit ca numele mașinii interne să apară în adresa de poștă electronică. De exemplu, dacă un același utilizator are cont pe mai multe mașini, nu este de dorit ca adresa expeditorului să depindă de mașina de pe care utilizatorul scrie efectiv mesajul. De exemplu, nu este de dorit ca dacă scrie un mesaj pe mașina `linux.scs.ubbcluj.ro` mesajul să apară având adresa sursă `From: ion@linux.scs.ubbcluj.ro`, iar dacă îl scrie de pe `freebsd.scs.ubbcluj.ro` să apară cu adresa `From: ion@freebsd.scs.ubbcluj.ro`. Pentru aceasta, MTA-ul care se ocupă de livrarea spre exterior a mesajelor generate în rețeaua internă va schimba, atât în antetul mesajului (valoarea câmpului `From`) cât și pe „plic“ (valoarea parametrului comenzii `SMTP MAIL FROM:`), adresa expeditorului, eliminând numele mașinii locale și păstrând doar

restul componentelor numelui de domeniu. În exemplul de mai sus, din adresă rămâne doar `ion@scs.ubbcluj.ro`. Modificarea poate fi mai complexă: astfel, dacă `nessie.cs.ubbcluj.ro` este configurat să accepte mesajele destinate lui `ion@example.com` și să le livreze utilizatorului local `gheorghe`, este probabil de dorit ca pentru mesajele compuse de utilizatorul local `gheorghe` să facă transformarea adresei sursă din `gheorghe@nessie.cs.ubbcluj.ro` în `ion@example.com`.

- Un alt element important de configurare privește decizia unui MTA de a accepta sau nu spre livrare un mesaj. În mod normal, un MTA trebuie să accepte mesajele generate de calculatoarele din rețeaua locală și mesajele destinate adreselor locale, dar să nu accepte trimiterea mai departe a mesajelor provenite din exterior și destinate în exterior. Un MTA care acceptă spre livrare orice mesaje este numit în mod curent *open relay*. Un *open relay* este privit de obicei ca un risc pentru securitate, deoarece este adesea utilizat de utilizatori rău-voitori pentru a trimite mesaje ascunzându-și identitatea.

11.1.3. Securitatea poștei electronice

Principalele probleme privind securitatea sunt:

- *spoofing*-ul — falsificarea adresei sursă (**From** sau **Sender**);
- *spam*-urile — mesaje, de obicei publicitare, trimise unui număr mare de persoane și fără a fi legate de informații pe care destinatarii le-ar dori;
- virușii — programe executabile sau documente, atașate unui mesaj electronic, a căror execuție sau respectiv vizualizare duce la trimiterea de mesaje electronice către alți destinatari.

Falsificarea adresei sursă este extrem de simplă deoarece, în transmiterea obișnuită a mesajelor, nu este luată nici o măsură de autentificare a expeditorului.

Falsificarea adresei sursă (spoofing) poate fi depistată în anumite cazuri examinând câmpurile **Received** din antetul mesajului și verificând dacă există neconcordanțe între numele declarat prin **HELO** a unui client SMTP și adresa sa IP sau neconcordanțe între numele primului MTA și partea de domeniu din adresa expeditorului. De notat că anumite neconcordanțe pot fi legitime, în cazul în care căsuțele poștale dintr-un domeniu sunt ținute de un calculator al cărui nume nu face parte din acel domeniu (vezi § 11.1.2.3).

O metodă mai eficientă pentru depistarea falsurilor este utilizarea semnăturii electronice. Există două standarde de semnătură electronică utilizate, OpenPGP [RFC 2440, 2007, RFC 3156, 2001] și S-MIME [S/MIME,].

Verificarea semnăturii necesită însă ca destinatarul să dispună de cheia publică autentică a expeditorului. Până la generalizarea utilizării mesajelor semnate, sistemul de poștă electronică trebuie să asigure livrarea mesajelor nesemnate și în consecință cu risc de a fi falsificate.

Ușurința falsificării adresei sursă și ușurința păstrării anonimatului autorului unui mesaj a dus la proliferarea excrocheriilor. Adesea excrocheriile constau în trimiterea de mesaje unui număr mare de utilizatori (acest fapt în sine este *spam*) în speranța de-a găsi printre aceștia unii care să se lase păcăliți.

Spam-urile dăunează deoarece consumă în mod inutil timpul destinatarului. În plus, există riscul ca un mesaj legitim, „îngropat“ între multe spam-uri, să fie șters din greșeală.

Există detectoare automate de spam-uri, bazate pe diferite metode din domeniul inteligenței artificiale. Astfel de detectoare se instalează pe MTA-uri și resping sau marchează prin antete speciale mesajele detectate ca spam-uri. Un MTA care detectează și respinge sau marchează spam-urile se numește *filtru anti-spam*.

De menționat filtrele anti-spam nu pot fi făcute 100% sigure deoarece nu există un criteriu clar de diferențiere. Ca urmare, orice filtru anti-spam va lăsa să treacă un anumit număr de spam-uri și există și riscul de-a respinge mesaje legitime.

Majoritatea furnizorilor de servicii Internet nu permit, prin contract, clienților să trimită spam-uri și depun eforturi pentru depistarea și penalizarea autorilor. În acest scop, ei primesc sesizări și întrețin liste cu adresele de la care provin spam-urile (*liste negre — blacklist*).

Trimiterea spam-urilor necesită recoltarea, de către autorul spam-urilor, a unui număr mare de adrese valide de poștă electronică. Acest lucru se realizează cel mai ușor prin căutarea, în paginile web, a tot ceea ce arată a adresă de poștă electronică. Contramăsura la această recoltare este scrierea adreselor, din paginile web, doar în forme dificil de procesat automat — de exemplu, ca imagine (într-un fișier *jpeg*, *gif* sau *png*).

Termenul de *virus* poate desemna mai multe lucruri, înrudite dar distincte. În general, un virus informatic este un fragment de program a cărui execuție duce la inserarea unor copii ale sale în alte programe de pe calculatorul pe care se execută virusul. Improprriu, prin *virus* se mai desemnează un fragment, inserat într-un program util, care execută acțiuni nocive utilizatorului în contul căruia se execută acel program. Denumirea corectă pentru un astfel de program este aceea de *cal troian*. Denumirea de *virus* poate fi dată, corect, doar fragmentelor de program capabile să se reproducă (să-și insereze

cópii în alte programe).

În contextul poștei electronice, un *virus* este un fragment dintr-un program plasat ca fișier atașat la un mesaj electronic. Virusul se poate reproduce fie prin mijloace independente de poșta electronică, fie prin expedierea, către alți utilizatori, a unor cópii ale mesajului. În acest al doilea caz, virusul utilizează, de obicei, adrese extrase din lista, ținută de MUA, a adreselor partenerilor de corespondență ai utilizatorului care primește mesajul virusat.

Indiferent de forma de propagare (infectarea fișierelor locale sau transmiterea de mesaje spre alți utilizatori), pentru a-și realiza scopul, un virus trebuie să ajungă să determine *execuția, cu drepturile utilizatorului victimă, a unei secvențe de instrucțiuni aleasă de autorul virusului*. Acest lucru se poate întâmpla în două moduri:

- Virusul se găsește într-un program executabil, pe care utilizatorul îl execută.
- Virusul este un document astfel construit încât, exploatănd o eroare din programul utilizat pentru vizualizarea documentului, să determine programul de vizualizare să execute acțiunea dorită de autorul virusului.

Pentru a păcăli destinatarul și a-l determina să execute sau să vizualizeze fișierul atașat, corpul mesajului este construit astfel încât să câștige încrederea utilizatorului. Astfel, mesajul este adesea construit ca și când ar proveni de la administratorul de sistem sau de la un prieten al destinatarului.

Metodele de prevenire a virușilor de poștă electronică sunt aceleași ca și metodele de prevenire a virușilor în general. Pentru programele executabile, dacă utilizatorul are încredere în autorul declarat al programului (de exemplu, autorul este o firmă de soft de încredere), atunci programul poate fi semnat electronic, iar utilizatorul poate verifica această semnătură pentru a se convinge de autenticitatea programului. Pentru programe provenite din surse ce nu sunt de încredere, execuția lor se poate face într-un mediu controlat, de exemplu dintr-un cont separat, cu drepturi minime, sau prin intermediul unui interpretor care să nu execute instrucțiunile potențial nocive. Această din urmă abordare este utilizată de *applet*-urile Java.

Pe lângă aceste metode de prevenire, există câteva acțiuni care micșorează riscul sau consecințele execuției unui virus. Una dintre ele este reducerea la minimum necesar a lucrului din cont de administrator. Altă măsură preventivă este aceea de a nu vizualiza sau executa fișierele atașate unui mesaj suspect; pentru aceasta, este bine ca expeditorul unui mesaj să nu trimită niciodată un mesaj numai cu fișiere atașate, ci să scrie un mic text explicativ, care să-i permită destinatarului să identifice autorul și scopul fișierelor atașate.

11.2. Sesiuni interactive la distanță

O sesiune interactivă *locală* a unui utilizator (vezi fig. 11.2) presupune că tastatura, ecranul și eventual alte dispozitive cu ajutorul cărora utilizatorul interacționează cu sistemul de calcul (mouse, difuzoare, etc.) sunt conectate direct la calculatorul pe care se desfășoară sesiunea utilizatorului. Conectarea este realizată printr-o interfață hardware de conectare a dispozitivelor periferice (RS232, PS/2, VGA, USB, DVI), care permite legături pe distanțe de cel mult câteva zeci de metri. Un dispozitiv (tastatură, ecran, etc.) este conectat la un singur calculator, mutarea lui de la un calculator la altul putându-se face fie prin mutarea fizică a conectorului, fie prin comutatoare speciale (KVM switch).

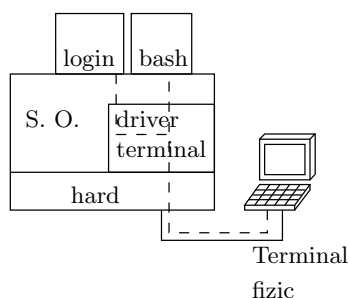


Figura 11.2: Componentele implicate într-o sesiune interactivă locală

În general ne gândim că pe un astfel de sistem lucrează un singur utilizator la un moment dat. Totuși, există și posibilitatea de-a conecta mai multe ansambluri tastatură–ecran la un același calculator, în felul acesta lucrând simultan mai mulți utilizatori. Acest mecanism s-a utilizat masiv în anii 1970, sistemele fiind numite *cu time-sharing*. PC-urile au repetat, până la un punct, istoria calculatoarelor mari: au început ca sisteme monoutilizator, monotasking (sistemul DOS), au continuat cu un multitasking primitiv, bazat pe soluții ad-hoc (deturnarea întreruperilor în DOS, sistemul Windows până la versiunile 3.x), sisteme multitasking fără protecție între utilizatori (Windows 9x și ME) și în final sisteme multitasking propriu-zise (Windows NT/2000/XP și sistemele de tip UNIX — Linux și porturile FreeBSD, Solaris, etc).

Linux (prin mecanismul consolelor virtuale) și Windows XP (prin mecanismul *switch user*) permit deschiderea simultană a mai multor sesiuni locale de la același ansamblu tastatură–ecran, pentru același utilizator sau pentru utilizatori distincți. O singură sesiune poate fi activă la un moment dat, celelalte fiind „înghețate“. Sistemul permite comutarea între sesiuni.

În cazul unei *sesiuni la distanță*, în locul unui terminal, conectat printr-o interfață specializată la calculatorul pe care se desfășoară sesiunea, se utilizează un calculator, conectat prin rețea la calculatorul pe care se desfășoară sesiunea. În felul acesta, un utilizator aflat în fața unui calculator conectat la Internet poate deschide o sesiune la distanță către orice alt calculator din Internet (bineînțeles, cu condiția să aibă un cont acolo). Principial, numărul de sesiuni deschise simultan către un calculator este limitat doar de resursele calculatorului (memorie și viteză de procesare).

Deschiderea unei sesiuni prin mecanismul de sesiune la distanță se poate face și către calculatorul local. Acest mecanism poate fi utilizat pentru deschiderea unei sesiuni ca alt utilizator, fără a închide prima sesiune.

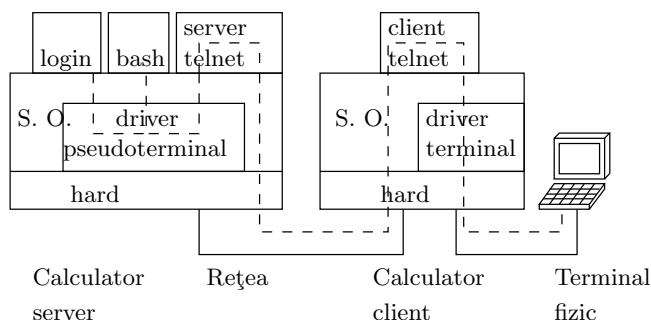


Figura 11.3: Componentele implicate într-o sesiune interactivă la distanță.

Sistemul pentru deschiderea sesiunilor la distanță (vezi fig. 11.3) constă din două componente majore:

- Pe sistemul la care este conectat fizic utilizatorul rulează o aplicație care trimite prin rețea ceea ce utilizatorul introduce de la tastatură și afișează pe ecran ceea ce trimite sistemul de la distanță. Afișarea se poate face pe tot ecranul sau într-o fereastră. Această aplicație deschide în mod activ conexiunea la deschiderea sesiunii, motiv pentru care este un *client*.
- Pe sistemul de la distanță, pe care are loc sesiunea, rulează o aplicație care primește prin rețea datele trimise de aplicația client și le livrează proceselor ce rulează în cadrul sesiunii. De asemenea, preia datele de ieșire ale acestor procese — datele care în cazul unei sesiuni locale s-ar afișa pe ecran — și le trimite prin rețea clientului. Această aplicație este lansată la pornirea sistemului și așteaptă conexiuni — fiind în acest sens un server. La conectarea unui client, aplicația server autentifică clientul după care (în cazul unei autentificări cu succes) lansează procesele care sunt lansate în mod normal la deschiderea unei sesiuni. De exemplu,

în cazul unui sistem de tip UNIX, serverul lansează în execuție un *shell* rulând în contul utilizatorului.

Pentru ca serverul să comunice cu procesele din cadrul sesiunii, este necesar ca sistemul de operare să ofere un mecanism adecvat de comunicație între procese. Mecanismul de comunicație trebuie să apară față de procesele din sesiunea utilizatorului ca și când ar fi tastatura și ecranul adevărate. În cazul sistemelor de tip UNIX, acest mecanism este mecanismul *pseudoterminalelor*. De notat că mecanismul *pipe* nu este adecvat deoarece un *pipe* nu apare procesului ca un terminal și nu permite, de exemplu, unui editor de texte, ce ar rula în sesiunea utilizatorului, să solicite primirea fiecărui caracter tastat în parte. De notat că, în mod normal, un proces primește câte o linie în momentul în care utilizatorul apasă *enter*; până atunci nucleul sistemului permite utilizatorului editarea liniei.

11.2.1. Protocolul *ssh*

Protocolul *ssh* a fost dezvoltat ca o alternativă, protejată criptografic, la *telnet*. Protocolul *ssh* este însă extensibil, permițând tunelarea protejată criptografic a conexiunilor TCP pentru alte aplicații.

Protocolul *ssh* este construit pe mai multe nivele. Nivelul cel mai de jos [RFC 4253, 2006] realizează protejarea criptografică a conexiunii și se bazează pe serviciile de conexiune TCP oferite de sistemul de operare. Nivelul următor realizează multiplexarea conexiunii protejate criptografic. Nivelul cel mai de deasupra cuprinde aplicația propriu-zisă și permite sesiuni de lucru, interactive sau nu, în mod text, către sisteme de tip UNIX, tunelarea unor conexiuni TCP arbitrare, transferul de fișiere și transmiterea informațiilor de autentificare criptografică pentru alte sesiuni *ssh*.

11.2.1.1. Conexiunea *ssh* protejată criptografic

Descriem în continuare modul în care *ssh* realizează protejarea criptografică a conexiunii. Protocolul este un exemplu instructiv de utilizare practică a primitivelor criptografice.

În secvența de inițializare a conexiunii — care va fi descrisă mai jos — clientul și serverul stabilesc un *identificator de sesiune* și, pentru fiecare sens, o cheie de criptare și o cheie de autentificare. De asemenea, se stabilesc algoritmi de criptare simetrică, compresie și dispersie cu cheie utilizați pentru fiecare sens al comunicației. Comunicația decurge complet independent în cele două sensuri.

Pentru fiecare sens, datele de transmis sunt grupate în pachete, de dimensiune variabilă. Pentru fiecare pachet de date utile, se construiește și se transmite pe conexiune un pachet generat astfel:

1. Datele utile sunt comprimate utilizând algoritmul de compresie curent pentru sensul de comunicație curent.
2. Se adaugă, după datele comprimate, un șir de octeți aleatori, iar în fața lor se adaugă un octet reprezentând lungimea șirului aleator. Apoi, în fața șirului astfel obținut, se adaugă lungimea totală a șirului, reprezentată pe patru octeți. Numărul de octeți aleatori adăugați trebuie astfel ales încât să rezulte în urma concatenării un șir de lungime multiplu de lungimea blocului cifrului utilizat.
3. Rezultatul pasului precedent se criptează.
4. În fața blocului (necriptat) rezultat din pasul 2 se adaugă numărul de ordine al pachetului curent, după care din rezultatul concatenării se calculează dispersia cu cheia de autentificare curentă. Numărul de ordine începe de la 0 și crește cu 1 la fiecare pachet independent de eventuala schimbare a cheilor sau algoritmilor criptografici utilizați.
5. Pachetul transmis efectiv este rezultatul concatenării pachetului criptat (rezultat din pasul 3) cu dispersia cu cheie (rezultată din pasul 4).

Rolul acestor transformări este următorul. Pe de o parte, compresia crește entropia datelor de criptat, făcând mai dificilă spargerea cifrului. Octeții adăugați la finalul blocului fac ca în cazul repetării aceluiași bloc de date utile să rezulte blocuri criptate diferite. Lungimea completării aleatoare este și ea criptată, făcând dificilă determinarea lungimii datelor utile din blocul criptat. Pe de altă parte, dispersia criptografică cu cheie se calculează dintr-un bloc conținând datele utile și numărul de ordine al blocului, fapt ce permite receptorului să verifice că datele sunt autentice și că sunt proaspete — numărul de ordine al blocului primit este cel așteptat. Numărul de ordine al blocului fiind cunoscut receptorului, nu este nevoie să fie trimis efectiv.

În cazul vreunei nepotriviri privitoare la dispersia criptografică cu cheie a unui bloc, conexiunea este abandonată. Remarcăm faptul că o astfel de nepotrivire poate fi cauzată doar de o tentativă de modificare a datelor de către un adversar activ, nivelul TCP și nivelele inferioare corectând erorile de transmisie la nivel fizic și eventualele pierderi de pachete IP datorate unei congestii în rețea.

La deschiderea conexiunii *ssh*, compresia, criptarea și dispersia cu cheie sunt dezactivate. Negocierea primului set de chei și a algoritmilor de compresie, criptare și dispersie cu cheie se face în clar. O dată alese cheile

și algoritmi, acestea sunt activate și se poate începe comunicația în folosul aplicațiilor. Algoritmii și cheile pot fi renegociate ulterior oricâteori una dintre părți (clientul sau serverul) o solicită.

Negocierea cheilor și algoritmilor se face după cum urmează. Fiecare parte trimite liste, în ordinea descrescătoare a preferinței, cu algoritmi de criptare, compresie, dispersie cu cheie, semnătură digitală și schimb de chei suportate. Algoritmii utilizați, pentru fiecare categorie, este primul algoritm de pe lista clientului care se regăsește și în lista serverului (adică cel mai favorabil clientului, dintre cei acceptați de server). Urmează schimbul de mesaje conform protocolului Diffie-Hellman (*ssh* nu are definite deocamdată alte metode de schimb de chei). Din informația secretă construită prin schimbul Diffie-Hellman se construiesc (pe baza unor funcții de dispersie fără cheie) cheile secrete pentru criptare și pentru autentificare pentru fiecare sens.

Mai rămâne de autentificat schimbul Diffie-Hellman, despre care am văzut că, singur, este vulnerabil la atacul unui adversar activ. Autentificarea cheii față de client (adică autentificarea, față de client, a serverului cu care comunică acesta) se face după cum urmează. Serverul are o pereche de chei pentru semnătură electronică. Clientul trebuie să aibă cheia publică a serverului. După realizarea schimbului Diffie-Hellman, serverul trimite clientului o semnătură, calculată cu cheia sa secretă, asupra întregului schimb de informație de până atunci — adică listele de algoritmi suportați și pachetele corespunzătoare protocolului Diffie-Hellman, emise de ambele părți. Prin verificarea semnăturii, clientul se asigură că negocierea a avut loc într-adevăr cu serverul autentic. Autentificarea clientului față de server se face ulterior, existând în acest scop mai multe mecanisme posibile (vezi § 11.2.1.2).

Pentru facilitarea răspândirii utilizării protocolului *ssh*, serverul transmite la deschiderea conexiunii cheia sa publică către client. Notăm că, deoarece transmisia cheii publice a serverului nu poate fi încă autentificată, utilizarea de către client a cheii publice transmise de server prezintă riscul ca un adversar activ să se dea drept serverul autentic. Dacă însă adversarul n-a modificat cheia publică transmisă de server, restul comunicației este sigur. Mai mult, la prima conectare, clientul stochează local cheia primită de la server. La următoarele conectări, clientul compară cheia primită de la server cu cea stocată local; dacă sunt diferite, avertizează utilizatorul. În acest fel, dacă la prima conectare cheia primită de client de la server este cea autentică, orice atac ulterior din partea unui adversar activ este descoperit.

La prima conectare a programului client *ssh* la un server nou, clientul avertizează utilizatorul cu privire la faptul că nu poate verifica cheia serverului. La această primă conectare, împiedicarea unui atac al unui eventual adversar

se poate face în două moduri:

- Înainte de prima conectare, utilizatorul copiază, de pe mașina server sau dintr-o sursă autentificată, cheia publică a serverului și o introduce manual în lista de chei memorate local de programul client *ssh*. În acest fel, clientul *ssh* poate verifica cheia serverului chiar de la prima sesiune, întocmai ca în cazul unei sesiuni ulterioare.
- Utilizatorul obține, dintr-o sursă autentificată (de exemplu, vorbind la telefon cu administratorul mașinii server), dispersia criptografică a cheii publice a serverului. La prima conectare, utilizatorul compară dispersia astfel obținută cu dispersia cheii trimise de server (aceasta este afișată de clientul *ssh* împreună cu mesajul de avertisment prin care anunță imposibilitatea verificării cheii). Dacă cele două dispersii coincid, cheia trimisă de server este autentică.

Pe sistemele de tip UNIX, cheile publice ale serverului (pentru diferitele protocoale de semnătură) se găsesc în directorul `/etc/ssh`, în fișierele `ssh_host_rsa_key.pub`, respectiv `ssh_host_dsa_key.pub`. Aceste fișiere pot fi citite de orice utilizator al sistemului. Amprenta cheii dintr-un astfel de fișier se determină cu comanda `ssh-keygen -l -f fișier`. Clientul *ssh* memorează cheile serverelor în fișierul `~/.ssh/known_hosts`.

11.2.1.2. Metode de autentificare în *ssh*

În *ssh*, există autentificare reciprocă între client și server.

Așa cum am văzut, serverul se autentifică față de client cu ajutorul unui mecanism cu cheie privată și cheie publică.

După inițializarea mecanismului de protecție criptografică a conexiunii, este rândul clientului să-și declare identitatea (numele de utilizator) și să se autentifice.

Clientul poate fi autentificat de server prin mai multe metode. Cele mai comune sunt autentificarea prin parolă și autentificarea prin semnătură digitală (numită și autentificare cu cheie publică).

Autentificarea prin parolă presupune trimiterea de către client a parolei. Este esențial faptul că serverul este deja autentificat și confidențialitatea, integritatea și prospețimea comunicației sunt protejate. Ca urmare clientul nu riscă să trimită parola unui adversar și nici ca un adversar ce captează comunicația criptată să retrimite datele interceptate pentru a repeta o sesiune legitimă.

Autentificarea prin semnătură digitală presupune ca în faza de inițializare utilizatorul să configureze pe server o cheie publică, corespunzătoare

cheii sale secrete. La conectare, clientul se autentifică trimițând semnătura, cu cheia sa secretă, asupra identificatorului de sesiune creat în faza de stabilire a comunicației protejate criptografic. Serverul verifică semnătura utilizând cheia publică ce a fost configurată.

Configurarea autentificării cu cheie publică, pe sistemele de tip UNIX având server OpenSSH, este descrisă în continuare.

Perechile de chei se generează cu ajutorul utilitarului `ssh-keygen`.

Cheia publică admisibilă pentru conectarea în contul unui utilizator se scrie în fișierul `~/.ssh/authorized_keys` (sub directorul personal al utilizatorului). Deoarece acest fișier poate fi modificat doar de către posesorul contului, doar posesorul contului poate stabili cheia admisibilă pentru autentificare. Fișierul `~/.ssh/authorized_keys` poate conține mai multe chei. În acest caz, oricare dintre cheile secrete corespunzătoare este validă pentru autentificare. Este posibil ca, pentru anumite chei, să se configureze lansarea unei anumite aplicații; în acest caz, dacă clientul utilizează cheia pereche pentru autentificare, i se va lansa automat aplicația respectivă și nu o sesiune nerestricționată.

Pentru schimbarea cheii, de exemplu în cazul compromiterii cheii secrete, utilizatorul trebuie să genereze o nouă pereche de chei, să scrie noua cheie publică în fișierul `~/.ssh/authorized_keys`, să șteargă vechea cheie publică din acest fișier și să furnizeze noua cheie secretă clientului `ssh` la conectările ulterioare. Deoarece cheia publică nu este o informație secretă, compromiterea sistemului server nu duce la compromiterea, și deci la necesitatea schimbării, cheii. Acesta este un avantaj față de cazul autentificării prin parole, unde compromiterea serverului duce la compromiterea parolei și la necesitatea schimbării parolei nu numai pe acel sistem ci și pe alte sisteme pe care utilizatorul avea aceeași parolă.

Pentru furnizarea cheii secrete către clientul `ssh`, există două posibilități. Prima posibilitate este ca fișierul cu cheia secretă să fie făcut disponibil clientului `ssh`. Dacă fișierul conține cheia secretă ca atare, conectarea se face fără ca utilizatorul să mai dea vreo parolă. Dacă utilizatorul dorește să se conecteze de pe mașini (client) diferite, trebuie fie să poarte cheia cu el pe un suport amovibil, fie să pună copii ale cheii secrete pe fiecare sistem, fie să utilizeze chei diferite pentru conectarea de pe fiecare sistem. Ultima soluție oferă avantajul că, în cazul compromiterii unuia dintre sistemele client, este necesară schimbarea cheii secrete doar de pe acel sistem.

Deoarece compromiterea unui sistem client duce, în cazul stocării ca atare a cheii secrete, la compromiterea imediată a contului utilizatorului, cheile secrete se stochează, în mod obișnuit, în formă criptată în fișiere. Criptarea

se realizează printr-un algoritm simetric cu ajutorul unei chei derivate dintr-o parolă aleasă de utilizator. Stocarea cheii numai în formă criptată oferă un plus de siguranță (un intrus trebuie să obțină atât fișierul cu cheia secretă, cât și parola de decriptare a acestuia), însă duce la necesitatea de a da clientului *ssh* parola de decriptare la fiecare utilizare.

A doua posibilitate de a furniza aplicației client accesul la cheia secretă este prin intermediul unui program numit *agent de autentificare*. Agentul de autentificare este un proces server care are în memorie cheia secretă a utilizatorului, în forma decriptată. Clientul *ssh* se conectează la agentul de autentificare pentru a obține accesul la cheie.

Agentul de autentificare, având ca nume de executabil **ssh-agent**, se lansează de regulă la deschiderea sesiunii pe mașina client. Cheile secrete se încarcă în agentul de autentificare cu ajutorul unui program numit **ssh-add**. Acest program permite și listarea și ștergerea cheilor secrete. Dacă cheia secretă este stocată criptat, **ssh-add** solicită utilizatorului parola de decriptare. Cheia secretă decriptată se găsește doar în memoria agentului de autentificare, nu se stochează pe disc.

La lansare, clientul *ssh* caută să vadă dacă pe mașina locală rulează un agent de autentificare. Dacă da, contactează agentul de autentificare și-i solicită accesul la cheile secrete stocate de acesta. Clientul *ssh* pasează agentului identificatorul de sesiune și primește înapoi semnătura cu cheia secretă asupra acestuia. În acest fel, clientul *ssh* nu ajunge să cunoască efectiv cheia secretă. Deschiderea sesiunii către mașina server se face fără a solicita utilizatorului vreo parolă.

Comunicația dintre clientul *ssh* sau utilitarul *ssh-add* și agentul de autentificare se face printr-un *socket* de tip UNIX, al cărui nume este pus în variabila de mediu **SSH_AUTH_SOCKET**. Comunicația fiind locală, ea nu poate fi interceptată sau modificată. Autentificarea clientului (*ssh* sau *ssh-add*) se face prin aceea că drepturile de acces la socket-ul corespunzător sunt acordate doar proprietarului agentului de autentificare.

Protocolul *ssh* permite construcția unei conexiuni securizate dinspre mașina server *ssh* spre agentul de autentificare de pe mașina client *ssh*. În acest caz, la deschiderea sesiunii, serverul *ssh* acționează și ca un agent de autentificare. Cererile de semnătură primite de serverul *ssh* sunt trimise prin conexiunea *ssh* către client, iar clientul le trimite agentului local (de pe mașina client). Prelungirea nu se poate face în lipsa unui agent de autentificare pe mașina client.

Analizând securitatea prelungirii conexiunii la agentul de autentificare, observăm că serverul nu obține efectiv cheia secretă, însă, pe durata

conexiunii, poate deschide sesiuni în numele clientului către orice mașină care acceptă cheile stocate în agentul de autentificare. Din acest motiv, clientul *ssh* nu face prelungirea conexiunii la agentul de autentificare decât la cererea explicită a utilizatorului.

11.2.1.3. Multiplexarea conexiunii, tunelarea și aplicații

O dată deschisă conexiunea și realizată autentificarea clientului, clientul *ssh* poate solicita serverului deschiderea unei sesiuni de lucru, adică în esență lansarea unui *shell* în contul utilizatorului autentificat. Tot la solicitarea clientului, canalul de comunicație creat de server între server și *shell*-ul lansat poate fi realizat printr-un pseudoterminal (cazul obișnuit al unei sesiuni interactive) sau printr-o pereche de *pipe*-uri. A doua variantă se utilizează în cazul în care utilizatorul a lansat clientul *ssh* cu specificarea unei comenzi de executat pe server. În acest caz, comanda specificată de utilizator este transmisă serverului și acesta o execută cu intrarea și ieșirea redirectionate către server și prelungite prin conexiunea securizată către client. Redirectionând pe client intrarea sau ieșirea standard a comenzii *ssh*, se realizează, per ansamblu, redirectionarea intrării sau ieșirii standard către fișiere sau *pipe*-uri de pe mașina locală pentru comanda executată la distanță.

EXEMPLUL 11.5: Comanda

```
ssh rlupsa@nessie.cs.ubbcluj.ro ls -l > lista
```

are ca efect final crearea, pe mașina locală, a unui fișier *lista*, conținând lista numelor de fișiere de pe mașina *nessie.cs.ubbcluj.ro* din directorul personal al utilizatorului *rlupsa*.

11.2.2. Sistemul X-Window

Sistemul X-Window, dezvoltat de Massachuttes Institute of Technology pe la mijlocul anilor 1980, este un sistem ce permite stabilirea de sesiuni la distanță în mod grafic.

Descriem în continuare arhitectura X Window. Menționăm că este diferită față de sistemele studiate până aici. Diferența provine în primul rând din faptul că sistemul X Window are și scopul de-a asigura proceselor acces la ecranul grafic local.

O primă componentă a sistemului este *serverul X*. Acesta este un proces, având de regulă acces privilegiat la sistem, care gestionează tastatura și ecranul local. O aplicație ce are nevoie de acces la un ecran grafic și la

tastatura atașată se numește *client X*. Un client X se conectează la serverul X și, după autentificare, poate:

- să ceară serverului să deseneze diverse lucruri pe ecran;
- să solicite să primească informații cu privire la tastele apășate de utilizator și la mișcările mouse-ului.

La un server se pot conecta simultan mai mulți clienți, inclusiv de pe calculatoare diferite.

Serverul ține evidența unor *ferestre*, fiecare operație de desenare având specificată o fereastră în care să deseneze. Ecranul cu totul este considerat o fereastră, iar în fiecare fereastră se pot deschide subferestre. Serverul ține evidența modului în care se suprapun ferestrele și determină ce parte din desenul efectuat într-o fereastră este vizibil și trebuie desenat pe ecran.

Un client autentificat are acces deplin la tastatura și ecranul gestionate de server. Asta înseamnă, de exemplu, că un client poate să deseneze într-o fereastră deschisă de alt client și poate să capteze tot ceea ce tastează utilizatorul în acea fereastră. De principiu, sunt admise la un moment dat să se conecteze doar aplicații rulând în contul aceluiași utilizator.

Pentru ca aplicații distincte să nu se încurce reciproc, există niște convenții pe care aplicațiile se recomandă să le respecte. În linii mari, acestea prevăd ca o aplicație să nu deseneze în ferestrele deschise de altă aplicație și să nu capteze tastele când nu este activă.

Comutarea între aplicații, precum și mutarea și redimensionarea ferestrelor principale ale aplicațiilor, cad în sarcina unui client mai special numit *window manager*. *Window manager*-ul se conectează și se autentifică ca un client obișnuit, după care solicită serverului să fie informat de cererile de deschidere de ferestre trimise de ceilalți clienți. La fiecare fereastră principală deschisă, *window manager*-ul adaugă bara de titlu și marginile. Deoarece oricum nu există protecție între clienții conectați la un server X, un client nu are nevoie de privilegii speciale ca să acționeze ca *window manager*. Totuși, câteva dintre operațiile de care are nevoie un *window manager* ca să funcționeze sunt acordate de serverul X doar unui client la un moment dat. Ca urmare, nu pot exista două *window manager*-e simultan.

11.3. Transferul fișierelor în rețea

Cerința de-a transfera fișiere în rețea poate avea diferite particularități. Există mai multe protocoale și mai multe aplicații pentru transferul fișierelor în rețea, adaptate pentru diferite necesități.

O primă categorie de protocoale și aplicații privește, în principal, transferul fișierelor unui utilizator de pe o mașină pe alta, în condițiile în care utilizatorul are cont pe ambele mașini. Protocoalele construite pentru aceasta sunt *ftp* și *ssh*. De notat că și poșta electronică poate servi ca mecanism de transfer de fișiere.

O a doua categorie privește transferul fișierelor publice de la un calculator ce stochează astfel de fișiere la calculatorul unui utilizator ce dorește să citească fișierele respective. Inițial se utiliza protocolul *ftp* în acest scop. Protocolul utilizat în mod curent este însă *http*.

O a treia categorie privește accesul proceselor de pe un calculator la fișiere stocate pe alt calculator ca și când fișierele ar fi locale. De principiu fișierele respective sunt private, ca și pentru prima categorie de protocoale. Protocoalele din această categorie trebuie să satisfacă două cerințe specifice (față de prima categorie): să permită transferul doar a unei părți mici dintr-un fișier și să permită controlul partajării fișierului între procese. Protocoalele utilizate aici sunt SMB (utilizat în rețelele Windows) și NFS.

11.3.1. Protocolul *ftp*

Descriem pe scurt conceptele de bază ale protocolului *ftp*. Pentru detalii, a se vedea [RFC 765, 1985].

Clientul deschide o conexiune TCP către portul 21 al serverului; această conexiune se numește *conexiune de control*. Prin conexiunea de control, clientul transmite comenzi în format text, câte o comandă pe o linie. Fiecare comandă începe cu numele comenzii urmat de eventuali parametrii. Parametrii sunt separați prin spații, atât față de numele comenzii cât și între ei. Serverul răspunde tot în format text, fiecare răspuns începând cu un cod care arată dacă comanda s-a executat cu succes sau ce erori s-au produs, după care urmează un text ce descrie, în limbaj natural, rezultatul execuției comenzii. Cu o singură excepție (în cazul comenzii PASV, descrisă mai jos), textul din răspuns nu este interpretat de către aplicația client. El este însă afișat, de obicei, pe ecran utilizatorului aplicației client.

Autentificarea se face la solicitarea clientului. Clientul trimite succesiv două comenzi, **USER** și **PASS**, având ca parametrii respectiv numele utilizatorului și parola acestuia. Serverul refuză execuția majorității comenzilor clientului înainte de autentificarea cu succes a acestuia. După autentificare, serverul acceptă să efectueze operațiile cerute de client doar dacă utilizatorul în contul căruia s-a făcut autentificarea are dreptul la operațiile respective. Pe sistemele de tip UNIX, reglementarea drepturilor de acces se face de obicei astfel: la lansare, serverul rulează din contul **root**; la conectarea unui client,

serverul crează (prin apelul sistem `fork()`) un proces fiu care se ocupă de acel client; după autentificare, procesul fiu trece în contul utilizatorului autentificat (prin apelul `setuid()`); în continuare, serverul acceptă orice comenzi de la client și încearcă să le execute, iar verificarea drepturilor de acces este făcută de nucleul sistemului de operare în momentul în care procesul server fiu încearcă să acceseze sistemul de fișiere.

Pentru transferul de fișiere publice, serverul este configurat să accepte autentificare cu numele de utilizator `ftp` sau `anonymous` fără să solicite parolă sau acceptând orice șir de caractere pe post de parolă. În vremurile de început ale Internet-ului, se obișnuia ca un utilizator ce dorea acces la fișiere publice să-și dea, pe post de parolă, adresa sa de poștă electronică. O dată cu răspândirea *spam*-urilor, s-a renunțat la acest obicei.

Transferul fișierelor se cere prin comenzile `SEND` (dinspre client spre server) și `RETR` (dinspre server spre client). Comenzile au ca argument numele de pe server al fișierului de transferat. Transferul propriu-zis are loc printr-o conexiune separată, numită *conexiune de date*. Pentru fiecare fișier se deschide o nouă conexiune de date, care se închide la finalul transferului fișierului. Dimensiunea fișierului nu este specificată explicit nicăieri, receptorul fișierului obținând lungimea din faptul că emițătorul închide conexiunea de date la finalul fișierului.

Există două moduri de deschidere a conexiunii de date:

- Modul *activ* prevede că serverul deschide conexiunea de date ca o conexiune TCP dinspre portul 20 al serverului către un port specificat de client. Clientul specifică portul pe care așteaptă conexiunea prin comanda `PORT`. Conexiunea se deschide ca urmare a comenzii de transfer (`SEND` sau `RETR`), nu imediat după primirea comenzii `PORT`.
- Modul *pasiv* prevede deschiderea conexiunii de date de către client, dinspre un port oarecare al său, către un port specificat de server. Portul specificat de server se obține ca răspuns al comenzii `PASV` date de client. Acesta este singurul caz în care clientul interpretează din răspunsul serverului și altceva decât codul returnat.

Listarea fișierelor de pe server este solicitată de client prin comanda `LIST`. Transferul listei de fișiere se face tot printr-o conexiune de date, ca și în cazul comenzii `RETR`.

11.3.2. Protocolul HTTP

HyperText Transmission Protocol(HTTP) este un protocol elaborat pentru transferul dinspre server spre client a fișierelor cu informații disponibile

public. El înlocuiește protocolul *ftp* utilizat cu conectare ca utilizator *anonymous*. Deși numele protocolului face referire la hipertext, el poate fi utilizat pentru a transfera orice fel de conținut.

Protocolul de bază constă în trimiterea de către client a unei cereri, în care informația principală este numele fișierului cerut. Răspunsul serverului conține niște informații despre fișier și conținutul efectiv al fișierului. Implicit, conexiunea se încheie după transferul unui fișier. Dacă clientul dorește mai multe fișiere de pe același server, va trebui să deschidă câte o conexiune pentru fiecare fișier.

Protocolul a fost însă extins, ajungând să fie folosit ca protocol de transfer de date pentru aplicații de orice tip.

11.3.2.1. Structura cererilor și a răspunsurilor

Formatul comunicației este mixt, atât la cereri cât și la răspunsuri. Partea de început este text, iar conținutul fișierului este binar.

Cererea cuprinde, pe prima linie, un cuvânt reprezentând numele operației cerută. Pentru solicitarea unui fișier public de pe server, numele este **GET**. După numele operației urmează numele fișierului și apoi identificarea versiunii de protocol în conformitate cu care este formată cererea. Cele trei elemente sunt separate prin câte un spațiu.

Următoarele linii sunt de forma *nume:valoare*, similar cu antetul unui mesaj de poștă electronică. După ultima linie de antet urmează o linie vidă. Pentru unele tipuri de cereri, după linia goală se găsește un conținut. În acest caz, una dintre liniile din antet are numele **Content-length** și are ca valoare lungimea conținutului, dată ca șir de cifre zecimale.

Răspunsul este structurat similar cu cererea. Pe prima linie se află identificatorul versiunii **HTTP**, număr de trei cifre și un text. Numărul arată dacă cererea a fost satisfăcută cu succes sau nu, iar textul, neinterpretat de client, este o descriere în cuvinte a semnificației codului de trei cifre. Următoarele linii sunt de forma *nume:valoare* și dau informații despre fișierul solicitat. După ultima linie de antet urmează o linie vidă și apoi conținutul (binar) al fișierului. În antet se găsește o linie cu numele **Content-length** având ca valoare lungimea fișierului. Determinarea sfârșitului conținutului propriu-zis de către client trebuie făcută prin numărea octeților din partea de conținut.

Adesea, mai multe servere **HTTP** sunt găzduite fizic pe același calculator. În acest caz, fie numele serverelor corespund, prin **DNS**, unor adrese **IP** diferite, dar aparținând aceluiași calculator, caz în care serverul este configurat să răspundă în funcție de **IP**-ul către care a fost deschisă conexiunea,

fie numele serverelor corespund aceleiași adrese IP, caz în care este necesar ca în cererea *HTTP* să fie specificat serverul dorit. Acest lucru se realizează prin aceea că, în cererea clientului, se plasează un antet cu numele **Host** și având ca valoare numele de server dorit.

11.3.2.2. URL-urile

O pagină web este în general un fișier scris în *HyperText Markup Language* (HTTP) și oferit în acces public prin protocolul HTTP.

O pagină web constă, de obicei, din mai multe fișiere. Există un fișier de bază, scris în limbajul *HTML*, și alte fișiere, conținând anumite elemente ale paginii: imagini (în fișiere separate în formate specifice — JPEG, PNG, GIF), applet-uri (Java), specificări de formatare a paginii (fișiere *Cascading Style Sheet* — CSS). De asemenea, o pagină conține în general legături (*link*-uri) spre alte pagini. Toate acestea necesită referiri dintr-un fișier HTML către alte fișiere disponibile în acces public. Referirea acestor fișiere se face prin nume care să permită regăsirea lor ușoară.

Un *Universal Resource Locator* (URL) este un nume prin care se poate identifica și cu ajutorul cărora se potate regăsi o resursă disponibile în Internet. URL-urile au apărut ca un format standard de scriere a numelor fișierelor referite din paginile web; ele permit însă utilizări mult mai vaste.

Un URL este alcătuit în general din trei componente:

- *Tipul* identifică protocolul utilizat. Exemple mai cunoscute sunt: **http**, **ftp**, **https**, **mailto**.
- *Numele mașinii* este numele de domeniu sau adresa IP a mașinii pe care se găsește resursa (fișierul).

Pe lângă numele mașinii, în cadrul acestei componente se poate adăuga numele de utilizator în contul căruia trebuie să se autentifice un client pentru a obține accesul dorit la resursă. Numele de utilizator se dă în fața numelui sau adresei mașinii, separat de acesta prin caracterul @. Standardul original prevedea și posibilitatea de-a scrie în URL parola necesară conectării. Această utilizare este nerecomandată.

- *Calea* identifică resursa (fișierul) în cadrul serverului care o găzduiește. În principiu, ea este calea completă a fișierului cerut, relativă la un director de bază, fixat, al documentelor publice.

URL-urile se pot utiliza și se utilizează efectiv în multe alte scopuri decât identificarea paginilor web. De exemplu, sistemul *SubVersion* (SVN) utilizează URL-uri de forma `svn://mașină/cale` pentru a referi fișierele dintr-un *repository*.

11.3.2.3. Alte facilități HTTP

Antetul răspunsului HTTP oferă mai multe informații despre fișierul returnat:

- Tipul conținutului fișierului este specificat de către server prin intermediul unui antet cu numele **Content-type** și cu valoarea construită ca și în cazul antetului **Mime-type** de la poșta electronică. Tot ca și în cazul lui **Mime-type**, tipul conținutului poate fi urmat de specificarea codificării utilizate pentru text; de exemplu,

```
Content-type: text/html; charset=utf-8
```

înseamnă că fișierul este de tip HTML, iar codificarea utilizată pentru caractere este UTF-8.

- Data ultimei modificări a fișierului este specificată prin valoarea antetului cu numele **Date**.
- Tipul de compresie utilizat (dacă fișierul returnat este comprimat) este dat ca valoare a antetului **Content-transfer-encoding**.
- Limba în care este scris textul din fișier (dacă este cazul) este returnată ca valoare a antetului **Language**.

Este posibil ca unui URL să-i corespundă mai multe fișiere pe server, având conținut echivalent, dar în diverse formate, limbi sau codificări. Pentru a selecționa varianta dorită, clientul poate anunța posibilitățile și preferințele sale cu privire la tipul de fișier, limbă și codificare. Antetele corespunzătoare, din cererea clientului, sunt: **Accept**, **Accept-language** și **Accept-encoding**. Fiecare dintre acestea are ca valoare o listă de variante, în ordinea preferinței. De exemplu,

```
Accept-language: ro,en,fr
```

solicită serverului, de preferință, varianta în limba română a textului. Dacă o variantă în limba română nu este disponibilă, se solicită una în limba engleză, iar în lipsa acesteia una în limba franceză.

Protocolul HTTP permite formularea de cereri condiționate sau parțiale. O cerere parțială este utilă dacă fișierul cerut este mare și clientul dorește să-l aducă din bucăți sau dacă la o cerere precedentă a căzut legătura după transferul unei părți din fișier. O cerere condiționată determină serverul să transmită clientului fișierul numai dacă este îndeplinită o anumită condiție, cel mai adesea dacă a fost modificat mai recent decât o anumită dată specificată

de client. Dacă nu este îndeplinită condiția, serverul dă un răspuns format doar din antet, fără conținutul propriu-zis. Această facilitate este utilă dacă clientul deține o copie a unui fișier și dorește împrăștierea acesteia. Cererea parțială se specifică de către client prin intermediul antetului `Range`; cererea condiționată se specifică prin antetul `If-modified-since`.

Pentru optimizarea traficului, în cazul în care un client dorește mai multe fișiere de pe același server (aceasta se întâmplă adesea în cazul în care clientul aduce un fișier *html*, iar apoi are de adus imaginile și eventual alte obiecte din document), este prevăzută posibilitatea de-a păstra conexiunea deschisă pe durata mai multor cereri. În acest scop, clientul cere păstrarea conexiunii deschise, plasând în cerere antetul

```
Connection: keep-alive
```

Pentru a nu permite unor clienți să deschidă o conexiune și apoi să o lase deschisă la nesfârșit, ținând ocupate resurse pe server, serverul trebuie configurat să închidă automat conexiunea dacă nu se transferă date o perioadă de timp.

Este uneori util ca un utilizator care accesează un URL să fie redirecționat automat către alt URL. Aceasta se întâmplă dacă administratorul site-ului decide o reorganizare a paginilor și dorește ca utilizatorii care au reținut URL-uri desființate în urma reorganizării să fie redirecționați către paginile corespunzătoare din noua organizare. Această redirectionare se face prin trimiterea de către server a unui antet cu numele `Location` și având drept conținut URL-ul spre care se dorește redirectionarea clientului. În acest caz, programul client nu afișează conținutul returnat de server (conținut care în mod normal lipsește complet), ci cere și afișează conținutul de la URL-ul indicat în antetul `Location`.

11.3.2.4. Proxy HTTP

Un *proxy HTTP* este un proces care, față de un client HTTP, acționează aproape ca un server HTTP, iar pentru satisfacerea cererii contactează serverul solicitat de client și acționează, față de acest server, ca un client.

Un *proxy HTTP* este utilizat în mod normal pentru următoarele funcții:

- dacă dintr-o rețea locală există șanse mari ca mai mulți utilizatori să acceseze aceleași pagini web: În acest caz, clienții *HTTP* ai calculatoarelor din rețea se configurează să utilizeze un același *proxy HTTP* din rețeaua locală. Dacă există mai multe cereri pentru aceeași pagină, la prima cerere *proxy*-ul memorează pagina adusă, iar la următoarele cereri o servește clienților din memoria locală.

- dacă într-o rețea se utilizează adrese IP locale (vezi § 10.2.4.1 și § 10.7.2) și nu se dorește configurarea unui mecanism de translație de adrese (NAT, § 10.7.3): În acest caz, se instalează un *proxy HTTP* pe un calculator din rețeaua locală dar având și adresă IP publică. Clientul accesează *proxy*-ul prin rețeaua locală, iar *proxy*-ul, având adresă publică, poate accesa fără restricții serverul dorit.
- dacă este de dorit un control fin asupra paginilor ce pot fi accesate dintr-o rețea locală (de exemplu, pentru a restricționa accesul angajaților la saitari nelegate de activitatea lor normală). În acest caz, se configurează un *proxy* în care se configurează toate restricțiile de acces dorite. Apoi, pentru a împiedica accesul direct, prin evitarea *proxy*-ului, pe ruterul ce leagă rețeaua internă la Internet se configurează un filtru de pachete (§ 10.7.1) care să blocheze pachetele adresate portului TCP 80 al serverelor exterioare.

O diferență între protocolul de comunicație dintre un client și un *proxy* față de protocolul client-server este că, după o cerere (de exemplu, **GET**), la protocolul client-server urmează calea locală din URL, iar la protocolul client-*proxy* urmează URL-ul solicitat (inclusiv numele protocolului și numele serverului).

O a doua diferență este dată de existența unei cereri, **CONNECT**, ce poate fi adresată doar unui *proxy*. La primirea unei cereri **CONNECT**, *proxy*-ul deschide o conexiune către serverul specificat în comanda **CONNECT** și apoi retrimite datele dinspre client direct spre server și, reciproc, dinspre server înapoi spre client. În cazul unei cereri **CONNECT**, *proxy*-ul nu se implică mai departe în comunicația dintre client și server. Ca urmare, în acest caz *proxy*-ul nu mai memorează paginile aduse și nu permite filtrarea cererilor decât după serverul și portul la care se conectează, în schimb permite tunelarea oricărui protocol (nu numai a protocolului HTTP) între un client dintr-o rețea cu adrese interne și un server din Internet.

11.3.2.5. Conexiuni securizate: SSL/TLS

SSL — *Secure Sockets Layer*, rom. *nivelul conexiunilor securizate* — este un protocol pentru securizarea conexiunilor. A fost creat de firma Netscape în vederea securizării comunicației între clientul și serverul HTTP. Protocolul este însă suficient de flexibil pentru a permite oricărei aplicații ce comunică prin conexiuni să-l folosească. TLS [RFC 4346, 2006] — *Transport Layer Security*, rom. *securitate la nivel transport* — este derivat din SSL versiunea 3, dar dezvoltat de IETF (Internet Engineering Task Force). În cele ce

urmează, vom discuta doar despre TLS, însă toate chestiunile prezentate sunt valabile și pentru SSL.

Protocolul TLS presupune existența unei legături nesecurizate între un client (entitatea care inițiază activ comunicația) și un server (entitatea care așteaptă să fie contactată de către client). Legătura nesecurizată este, în mod obișnuit, o conexiune TCP. Protocolul TLS oferă un serviciu de tip conexiune. TLS asigură confidențialitatea și autenticitatea datelor utile transportate. Datele utile transportate de TLS pot aparține oricărui protocol. Protocolul ale cărui date sunt transportate ca date utile de către TLS este numit *tunelat* prin TLS.

În cadrul inițierii unei conexiuni TLS, se realizează stabilirea unei chei de sesiune care este utilizată în continuare pentru securizarea transportului datelor utile. Autentificarea stabilirii cheii poate fi unilaterală, doar clientul autentificând serverul cu care a realizat negocierea cheii de sesiune, sau bilaterală. În cazul autentificării unilaterale, se poate utiliza o autentificare a clientului față de server în cadrul protocolului tunelat. În acest caz, deoarece serverul este deja autentificat, autentificarea clientului poate fi făcută prin parolă, fără riscul ca parola să fie transmisă unui adversar.

Autentificarea stabilirii cheii de sesiune se face printr-un mecanism de semnătură digitală. Pentru distribuirea sigură a cheilor publice, utilizate în cadrul autentificării, se utilizează certificate (§ 6.3.4).

Serverul trebuie să aibă o pereche de chei pentru semnătură digitală și un certificat, semnat de o autoritate în care clientul are încredere, pentru cheia publică. La inițierea conexiunii TLS, serverul transmite clientului certificatul său. Clientul verifică faptul că numele din certificat coincide cu numele serverului la care dorea conectarea, că semnătura autorității de certificare asupra certificatului este validă, că autoritatea de certificare este de încredere și, în final, utilizează cheia publică din certificatul clientului pentru a autentifica stabilirea cheii de sesiune. Dacă se dorește și autentificarea clientului față de server tot prin TLS, atunci clientul trebuie să aibă, la rândul său, o pereche de chei și un certificat.

În vederea verificării semnăturii din certificat și a faptului că semnatarul (autoritatea de certificare) este de încredere, fiecare dintre parteneri trebuie să aibă o listă cu cheile autorităților de certificare de încredere. Cheia unei autorități de certificare este, în mod obișnuit, plasată tot într-un certificat.

Certificatul unei autorități de certificare poate fi semnat de către o altă autoritate de certificare sau chiar de către autoritatea posesoare a certificatului. În acest din urmă caz, certificatul se numește *certificat autosemnat*

(engl. *self-signed certificate*) sau *certificat rădăcină* (engl. *root certificate*). În majoritatea cazurilor, clientul are o mulțime de certificate autosemnate ale autorităților în care are încredere.

Există mai multe produse soft pentru crearea perechilor de chei și pentru crearea și semnarea certificatelor. Un astfel de produs este *OpenSSL*, disponibil pe sistemele de tip UNIX.

11.3.2.6. Utilizarea TLS pentru web

Transferul securizat al paginilor web se realizează prin tunelarea protocolului HTTP peste SSL sau TLS. Construcția realizată se numește HTTPS.

URL-urile resurselor accesibile prin conexiuni securizate au, ca nume al protocolului, șirul de caractere `https` (în loc de `http`).

Un navigator web care are de adus o pagină a cărei URL are, în partea de protocol, `https`, execută următoarele:

- Afară de cazul în care URL-ul specifică explicit un număr de port, clientul deschide conexiunea către portul 443 al serverului (în loc de portul 80, implicit pentru HTTP).
- Dacă, în locul contactării directe a serverului web, se utilizează un *proxy*, clientul trimite serverului *proxy* o cerere `CONNECT` pentru stabilirea conexiunii spre server. Prin această conexiune, stabilită prin intermediul *proxy*-ului, se transmit mesajele SSL sau TLS, în cadrul cărora se transmit datele HTTP.
- La deschiderea conexiunii (fie conexiune TCP directă, fie un lanț de conexiuni TCP prin intermediul *proxy*-ului), are loc mai întâi schimbul de mesaje legat de stabilirea cheii SSL sau TLS. După inițializarea conexiunii securizate, prin canalul securizat are loc un dialog conform protocolului HTTP. Cu alte cuvinte, cererile și răspunsurile HTTP constituie date utile pentru nivelul SSL sau TLS.

Autentificarea serverului, în cadrul protocolului TLS, necesită, așa cum am văzut, ca navigatorul web să dispună de certificatele autorităților de certificare de încredere. În general, producătorii de navigatoare încorporează în acestea niște certificate, ale unor autorități de certificare larg recunoscute. Utilizatorul poate însă să dezactiveze oricare dintre aceste certificate, precum și să adauge alte certificate.

Atragem atenția asupra unor particularități legate de utilizarea HTTPS:

- Deoarece autentificarea serverului, prin mecanismul TLS, se face înaintea trimiterii cererii HTTP, certificatul trimis de server nu poate depinde de antetul `Host` transmis de către client. Ca urmare, dacă mai multe

saituri web securizate sunt găzduite de un același calculator, este necesar ca aceste saitari să fie distinse prin adresa IP sau prin numărul de port. În cazul în care s-ar utiliza doar antetul `Host` pentru ca serverul să determine saitul cerut de client, serverul ar trimite același certificat indiferent de saitul dorit de client. Ca urmare, ar exista o nepotrivire între numele din certificat și numele saitului solicitat de client. În consecință, clientul ar declara saitul ca fiind unul fals.

- O pagină web este formată, în mod obișnuit, din mai multe obiecte, cu URL-uri diferite (pagina HTML propriu-zisă și imaginile din pagină). În aceste condiții, este posibil ca, într-o aceeași pagină, unele dintre elemente să fie securizate și celelalte elemente să fie nesecurizate. De asemenea, este posibil ca diferite elemente să provină de pe saitari diferite, autentificate prin certificate diferite. Într-un astfel de caz, navigatorul web trebuie să avertizeze utilizatorul.

11.4. PGP/GPG

Pretty Good Privacy (PGP) este un program pentru criptarea și semnarea digitală a mesajelor de poștă electronică și a fișierelor în general.

Gnu Privacy Guard, abreviat *GPG* sau *GnuPG*, este o reimplementare a PGP, cu statut de soft liber.

Prezentăm în continuare principalele concepte legate de construcția și funcționarea GnuPG.

11.4.1. Structura cheilor GnuPG

PGP criptează mesajele, utilizând metode simetrice și chei efemere, transmite cheile efemere prin criptare asimetrică și crează semnături electronice asupra mesajelor.

În acest scop, fiecare utilizator GnuPG trebuie să aibă niște perechi de chei pentru criptare asimetrică și pentru semnătură.

GnuPG memorează, în niște fișiere gestionate de el, cheile publice și private ale utilizatorului ce execută comanda `gpg`, precum și cheile publice ale partenerilor utilizatorului ce execută `gpg`.

Descriem în continuare structura cheilor GnuPG, precum și operațiile ce pot fi executate asupra cheilor memorate local. Transmiterea cheilor publice între utilizatori va fi descrisă în § 11.4.2.

Afișarea cheilor publice din fișierele gestionate de GnuPG se face prin comanda

```
gpg --list-keys
```

Afișarea cheilor secrete se face prin comanda

```
gpg --list-secret-keys
```

11.4.1.1. Chei primare și subchei

Cheile GnuPG sunt de două tipuri: *chei primare* și *subchei*. O cheie primară (de fapt, o pereche primară de chei) este întotdeauna o pereche de chei pentru semnătură digitală. O subcheie (de fapt, sub-pereche de chei) este subordonată unei anumite perechi primare. Fiecare subcheie poate fi cheie de criptare sau cheie de semnătură.

Fiecare utilizator are o cheie primară și, subordonate acesteia, zero sau mai multe subchei. În modul cel mai simplu de lucru, fiecare utilizator GnuPG are asociate, două perechi de chei: o pereche primară de chei pentru semnătură digitală și o sub-pereche de chei pentru criptare asimetrică. Perechea de chei de criptare este folosită pentru a trimite mesaje secrete posesorului perechii de chei. Perechea de chei de semnătură este folosită atunci când posesorul trimite mesaje semnate.

Fiecare cheie publică are asociată așa-numita *amprentă a cheii* (engl. *key fingerprint*). Aceasta este un șir de biți, calculați, printr-o funcție de dispersie criptografică, din cheia publică respectivă.

Pentru a ne putea referi la o pereche de chei, fiecare pereche de chei (fie ea primară sau subcheie) are asociate doi *identificatori de cheie* (engl. *key ID*):

- *Identificatorul lung* (engl. *long key ID*) este format din 16 cifre hexa. Șansele ca două chei distincte să aibă același identificator lung sunt extrem de mici, astfel încât identificatorul lung este suficient pentru a identifica unic orice cheie. Totuși, identificatorul lung nu este utilizabil, în locul amprentei cheii, pentru verificarea autenticității acesteia.
- *Identificatorul scurt* (engl. *short key ID*) este format din ultimele 8 cifre hexa ale identificatorului lung. Dacă, într-un anumit context, nu există două chei cu același identificator scurt, el poate fi folosit pentru a ne referi la o cheie.

Identificatorul unei perechi de chei este calculat, printr-o funcție de dispersie, din cheia publică din pereche.

Identificatorii scurți ai cheilor pot fi listați prin comanda

```
gpg --list-keys
```

Pentru a lista identificatorii lungi, se poate da comanda

```
gpg --with-colons --list-keys
```

Amprenta unei chei primare se poate afla prin comanda

```
gpg --fingerprint id-cheie
```

unde *id-cheie* este fie identificatorul scurt sau lung al cheii primare sau al unei subchei subordonate acesteia, fie numele real, adresa de poştă electronică sau numele complet al proprietarului cheii.

11.4.1.2. Utilizatori și identități

Fiecărei chei primare îi este asociată una sau mai multe *identități*. Fiecare identitate este un nume complet de utilizator, format din trei componente: *numele real* (numele și prenumele persoanei), *adresa de poştă electronică* și, opțional, un *comentariu*. În scrierea numelui complet, adresa de poştă electronică se scrie între semne *mai mic* și *mai mare*, iar comentariul se scrie între paranteze. Exemple:

```
Ion Popescu <ion@example.com>
```

```
Gheorghe Ionescu (Presedinte ONG) <gion@ong.example.com>
```

Este posibil ca o cheie primară să aibă asociate mai multe identități. Acest lucru este util dacă un utilizator are mai multe adrese de poştă electronică și dorește asocierea tuturor acestora cu aceeași cheie.

Reciproc, un același nume complet poate fi asociat mai multor chei primare. Acest lucru se întâmplă deoarece nu poate nimeni să împiedice doi utilizatori să genereze două chei și să le asocieze același nume complet.

Mai mult, această posibilitate este utilizată frecvent în situația în care cheia primară a unui utilizator expiră sau este revocată. În această situație, utilizatorul poate crea o nouă cheie primară căreia să-i asocieze același nume complet.

11.4.1.3. Generarea și modificarea cheilor

Generarea unei chei primare se face cu comanda

```
gpg --gen-key
```

Comanda este interactivă, solicitând utilizatorului următoarele informații: tipul cheilor generate și dimensiunea acestora, durata de valabilitate a cheilor, numele complet al utilizatorului și parola utilizată pentru criptarea cheii secrete.

Comanda generează o pereche primară de chei (de semnătură) și îi asociază o identitate. Opțional, comanda poate genera și o sub-pereche de

chei de criptare, subordonată perechii primare. Ulterior, se pot adăuga noi subchei și identități sau se pot șterge subcheile și identitățile asociate.

La generarea cheilor, GnuPG afișează amprenta a cheii primare generate. Este bine ca utilizatorul să noteze amprenta cheii generate. Acest lucru este util la transmiterea cheii publice către partenerii de comunicație.

Pentru o cheie primară dată, proprietarul ei poate crea (și, eventual, șterge) subchei. Pentru acestea, se lansează comanda

```
gpg --edit-key cheie
```

La lansarea acestei comenzi, **gpg** așteaptă, de la utilizatori, subcomenzi pentru modificarea unor date privitoare la cheia primară identificată prin parametrul *cheie*. Terminarea seriei de subcomenzi se face dând, mai întâi, subcomanda **save** pentru a memora efectiv modificările efectuate, urmată de subcomanda **quit**.

Crearea unei noi subchei se face cu subcomanda **addkey**. Subcheia creată poate fi o cheie de criptare sau o cheie de semnătură.

La ștergerea unei subchei se utilizează subcomenzile **key** și **delkey**. Ștergerea unei subchei este utilă doar dacă subcheia nu a fost încă transmisă nimănui. Nu există o metodă simplă de a propaga ștergerea asupra copiilor subcheii respective. Ca urmare, dacă proprietarul dorește ca o subcheie, deja transmisă partenerilor săi, să nu mai fie utilizată, soluția este revocarea subcheii și nu ștergerea ei.

Pentru a adăuga, șterge sau revoca o identitate asociată unei chei, se lansează comanda

```
gpg --edit-key cheie
```

și apoi se utilizează subcomenzile: **adduid**, **uid**, **deluid**, **revuid**, **primary**. După modificarea identităților asociate unei chei primare, este necesară retransmiterea cheii spre partenerii de comunicație (vezi § 11.4.2.1).

11.4.1.4. Controlul perioadei de valabilitate a cheilor

Valabilitatea unei chei sau subchei este controlată pe două căi: prin fixarea unei perioade de valabilitate, după expirarea căreia cheia nu mai este validă, și prin revocarea cheii. Controlul valabilității unei chei este necesar pentru a preîntâmpina utilizarea unei chei publice în cazul în care cheia secretă corespunzătoare a fost aflată de o persoană neautorizată.

Perioada de valabilitate a unei chei sau subchei se fixează la generarea acesteia. Ulterior, perioada de valabilitate poate fi modificată cu comanda

```
gpg --edit-key cheie
```

cu subcomenzile `key` și `expire`.

Pentru revocarea unei chei primare, se crează un *certificat de revocare*, semnat de proprietarul cheii primare. Certificatul de revocare se transmite apoi partenerilor de comunicație.

Generarea certificatului de revocare se face prin comanda

```
gpg -a -o fișier --edit-key cheie
```

Certificatul de revocare este scris în fișierul cu nume *fișier*. Pentru ca revocarea să aibă efect, certificatul de revocare trebuie importat prin comanda

```
gpg --import fișier
```

Odată importat un certificat de revocare pentru o cheie primară, semnăturile create cu acea cheie primară sau cu o subcheie a acesteia sunt considerate invalide și, în general, la orice utilizare a acelei chei sau a unei subchei `gpg` dă un avertisment. De asemenea, atunci când acea cheie primară este transmisă spre alți utilizatori (vezi § 11.4.2.1), certificatul de revocare este transmis împreună cu cheia revocată.

Ca utilizare recomandabilă, este bine ca, la crearea unei chei primare, proprietarul ei să genereze imediat un certificat de revocare pe care să-l țină într-un loc sigur. În cazul în care pierde cheia sau bănuiește că acea cheie secretă a fost aflată de un adversar, proprietarul transmite partenerilor săi certificatul de revocare. Înainte de revocare, certificatul de revocare trebuie să nu poată fi citit de nimeni; în caz contrar, un adversar care obține certificatul de revocare poate provoca neplăceri proprietarului revocându-i cheia.

Revocarea unei subchei constă în adăugarea la subcheie a unui marcaj, semnat de proprietarul subcheii, prin care se anunță că acea subcheie trebuie să nu mai fie utilizată. O subcheie revocată este tratată similar cu o subcheie sau cheie expirată: dacă se încearcă utilizarea ei, `gpg` dă un mesaj de avertisment.

Revocarea unei subchei se face cu ajutorul comenzii

```
gpg --edit-key cheie
```

cu subcomenzile `key` și `revkey`.

De notat că, după revocarea sau schimbarea perioadei de valabilitate a unei subchei, subcheia modificată trebuie să ajungă la partenerii proprietarului cheii (vezi § 11.4.2.1).

11.4.1.5. Gestiunea cheilor secrete

GnuPG plasează cheile secrete într-un fișier gestionat de GnuPG. Acest fișier este creat cu drepturi de citire (în sistemul de operare) doar pentru utilizatorul curent.

Cheile sunt, în mod normal, criptate cu o parolă dată de utilizator. Parola de criptare poate fi schimbată cu comanda

```
gpg --edit-key cheie
```

subcomanda `passwd`.

Cheile secrete pot fi exportate, prin comanda

```
gpg -a -o fișier --export-secret-keys cheie
```

Această comandă exportă cheia secretă primară identificată prin parametrul *cheie*, precum și subcheile sale secrete. Cheile secrete pot fi importate prin comanda

```
gpg --import fișier
```

Acest lucru este util dacă utilizatorul lucrează pe mai multe calculatoare și dorește să utilizeze aceeași cheie pe mai multe calculatoare. Cheia secretă este exportată în forma criptată.

Există posibilitatea de-a exporta doar subcheile secrete ale unei chei primare. Acest lucru se face prin comanda

```
gpg -a -o fișier --export-secret-subkeys cheie
```

Cu ajutorul acestei comenzi, un utilizator poate ține cheia primară secretă pe un calculator sigur și poate transmite subcheile secrete către un calculator mai puțin sigur pe care îl utilizează frecvent. În acest mod, el poate utiliza calculatorul mai nesigur pentru transmite mesaje semnate și primi mesaje criptate utilizând subcheile, fără însă a risca compromiterea cheii primare în cazul în care cineva ar sparge acel calculator. Pentru o astfel de utilizare, subcheile se creează cu durată de valabilitate scurtă și se revocă la nevoie. Cheia primară este bine să aibă durată lungă de utilizare pentru a beneficia de semnăturile obținute de la alți utilizatori asupra ei (vezi § 11.4.2.2).

11.4.2. Transmiterea și certificarea cheilor publice

11.4.2.1. Transmiterea cheilor publice

Cheile publice primare, identitățile asociate, semnăturile asupra identităților (vezi § 11.4.2.2), subcheile publice subordonate cheilor primare și certificatele de revocare ale cheilor primare sau subcheilor sunt memorate într-un fișier gestionat de GnuPG.

Transmiterea acestor obiecte de la un utilizator la altul se poate face prin două metode:

- prin fişiere (transmise, de exemplu, prin poştă electronică sau prin web);
- prin *servere de chei*.

La transmiterea prin fişiere, un utilizator exportă una sau mai multe chei primare, împreună cu identităţile, semnăturile, subcheile şi certificatele de revocare asociate acelor chei primare, într-un fişier. Celălalt utilizator primeşte fişierul (transmis prin poştă electronică, web, pe o dischetă sau prin alte mijloace) şi îi importă conţinutul în GnuPG-ul local.

Exportul unei chei publice primare, împreună cu toate identităţile, subcheile publice, semnăturile şi certificatele de revocare asociate, se face prin comanda

```
gpg -a -o fişier --export cheie
```

unde parametrul *cheie* este identificatorul cheii sau a uneia dintre subchei sau numele utilizatorului căreia îi aparţine, iar parametrul *fişier* reprezintă fişierul în care se vor scrie datele. Parametrul *cheie* poate lipsi sau poate să nu identifice o unică cheie primară; în acest caz, toate cheile primare respective sunt exportate.

Importarea unei chei dintr-un fişier se face prin comanda

```
gpg --import fişier
```

Prin operaţia de import, cheile şi celelalte obiecte din fişierul importat sunt adăugate celor locale sau, eventual, le modifică pe acestea. Niciodată însă nu sunt şterse obiecte locale pe motiv că nu se regăsesc în fişierul importat. Din acest motiv, ştergerea unei chei primare, subchei, identităţi sau semnături nu poate fi transmisă asupra partenerilor de comunicaţie. Invalidarea unei chei, identităţi sau semnături se poate face doar prin revocarea acesteia şi apoi transmiterea certificatului de revocare.

La transmiterea prin servere de chei, primul utilizator încarcă, pe un *server de chei*, cheile şi celelalte obiecte de transmis, iar celălalt utilizator le descarcă de pe serverul de chei.

Transmiterea unei chei primare şi a obiectelor asociate către un server de chei se face prin comanda

```
gpg --keyserver server --send-key cheie
```

Descărcarea unei chei şi a obiectelor asociate de pe un server de chei se face prin comanda

```
gpg --keyserver server --recv-key cheie
```

unde *cheie* este identificatorul unei chei (nu poate fi numele posesorului cheii). Aflarea identificatorului cheii unui utilizator se poate face prin comanda

```
gpg --keyserver server --search-key nume-utilizator
```

Este important de notat că, implicit, GnuPG nu consideră o cheie proaspăt importată ca fiind autentică. La utilizarea unei chei publice a cărei autenticitate nu a putut fi verificată, GnuPG dă un mesaj de avertizare. Verificarea autenticității este descrisă în paragraful următor.

11.4.2.2. Verificarea autenticității cheilor

GnuPG verifică automat, înainte de utilizarea unei chei publice, autenticitatea acesteia. Autenticitatea cheilor se verifică cu ajutorul certificatelor (vezi § 6.3.4). În terminologia GnuPG, un certificat este numit *semnătură asupra unei chei*.

O sub-cheie este în mod normal semnată cu cheia primară căreia îi este subordonată. O sub-cheie a cărei semnătură este validă este considerată autentică dacă și numai dacă cheia primară corespunzătoare este considerată autentică. În consecință, dacă se importă noi sub-chei pentru o cheie primară declarată autentică, sub-cheile respective sunt imediat considerate autentice.

Restul paragrafului de față tratează doar cheile primare. Reamintim că o cheie primară este întotdeauna o cheie de semnătură.

O cheie publică pentru care GnuPG dispune de cheia secretă corespunzătoare este automat considerată autentică. De asemenea, este considerată autentică orice cheie specificată printr-o opțiune

```
--trusted-key cheie
```

fie la execuția comenzii `gpg`, fie în fișierul cu opțiunile implicite. În afara acestor două cazuri, GnuPG consideră o cheie autentică numai dacă dispune de un certificat valid pentru ea și mai sunt îndeplinite următoarele condiții:

- cheia cu care este semnat certificatul este deja declarată ca autentică,
- semnatul certificatului este considerat de încredere.

GnuPG reține, asociate fiecărei chei primare, două informații (independente una de alta):

- *dacă autenticitatea ei este verificată sau nu;*
- *nivelul de încredere*, acordat de utilizatorul care execută `gpg`, proprietarului acelei chei.

Un utilizator poate acorda proprietarului unei chei:

- *încredere deplină (full trusting)* — semnătura aceluși utilizator asupra unei identități este suficientă pentru ca acea identitate să fie considerată verificată;
- *încredere minimală (marginally trusting)* — o identitate semnată doar de utilizatori de încredere minimală să fie considerată verificată este necesar un anumit număr de astfel de semnături (implicit 3).
- *zero încredere (no trusting)* — semnătura unui astfel de utilizator nu este luată în considerare.

Nivelul de încredere acordat proprietarului unei chei este implicit zero. El poate fi modificat cu comanda

```
gpg --edit-key cheie
```

și subcomanda `trust` a acesteia.

Implicit, GnuPG are încredere deplină în proprietarul unei chei pentru care dispune de cheia secretă corespunzătoare (altfel spus, în utilizatorul care lansează comanda `gpg`), precum și în proprietarii cheilor specificate prin opțiunea `--trusted-key`.

Crearea, de către utilizatorul ce execută `gpg`, a unei semnături asupra unei identități asociate unei chei se face prin comanda

```
gpg --sign-key cheie
```

sau

```
gpg --lsign-key cheie
```

În cazul primeia dintre comenzi, semnătura poate fi transmisă și altor utilizatori GnuPG (vezi § 11.4.2.1). A doua comandă crează o semnătură pentru uz local.

Este esențial, pentru securitatea sistemului, ca un utilizator să nu semneze un set de chei fără să-i verifice mai întâi autenticitatea. Autenticitatea setului de chei se poate asigura în două moduri:

- Fișierul din care se importă setul de chei este adus pe o cale sigură, de exemplu printr-o dischetă dată personal de către proprietarul cheii sau este descărcat de pe un sait web securizat (*https*) și de încredere.
- Întâi, amprenta cheii primare este transmisă pe o cale sigură, de exemplu pe un bilet scris de către proprietarul cheii sau printr-o convorbire telefonică cu proprietarul cheii. Apoi, la importarea și semnarea setului de chei, utilizatorul verifică amprenta cheii primare din set.

11.4.3. Transmiterea mesajelor criptate sau semnate

Crearea unui mesaj criptat și semnat se face astfel:

```
gpg -o fiș-ieșire -r cheie-dest -se fiș-intrare
```

sau

```
gpg -a -o fiș-ieșire -r cheie-dest -se fiș-intrare
```

unde *fiș-intrare* este fișierul ce trebuie semnat și criptat, *fiș-ieșire* este fișierul în care comanda `gpg` va pune datele criptate și semnate, iar *cheie-dest* reprezintă numele utilizatorului destinație sau identificatorul cheii de criptare de utilizate. Cea de-a doua variantă produce un fișier text ASCII, prin recodificare în baza 64.

Un mesaj poate fi adresat mai multor destinatari; pentru aceasta, se pot da, în comanda `gpg`, mai multe opțiuni `-r`, fiecare urmată de numele unui utilizator destinație. Oricare dintre destinatarii astfel specificați poate să decripteze mesajul criptat.

La criptarea unui mesaj, GnuPG generează aleator o cheie efemeră, criptează textul clar utilizând cheia efemeră, iar apoi criptează cheia efemeră utilizând cheia publică a destinatarului. Dacă sunt mai mulți destinatari, GnuPG criptează, pentru fiecare destinatar, câte o copie a cheii efemere, utilizând cheia publică a acelui destinatar.

Decriptarea unui mesaj se face prin comanda

```
gpg -o fiș-ieșire --decrypt fiș-intrare
```

unde *fiș-intrare* este fișierul semnat și criptat, iar *fiș-ieșire* este fișierul în care comanda `gpg` va pune rezultatul decriptării. Comanda verifică și semnătura și afișează pe ecran rezultatul verificării.

Se pot genera mesaje numai criptate sau numai semnate.

Transmiterea unui mesaj criptat dar nesemnat nu este recomandabilă, deoarece destinatarul nu poate avea nici un fel de certitudine asupra autenticității mesajului. Comanda de criptare este similară cu cea pentru criptare și semnare, dar cu `-e` în loc de `-se`. Comanda de decriptare este identică cu cea pentru un mesaj criptat și semnat.

Pentru generarea unui mesaj semnat dar necriptat există trei posibilități: semnătură inclusă în mesaj, semnătură detașată și semnătură în clar.

Semnătura inclusă se generează similar cu generarea unui mesaj criptat și semnat, dar lipsesc destinatarii (opțiunile `-r`) și în loc de `-se` se dă doar `-s`. Fișierul generat conține datele originale și semnătura. Extragerea datelor

și verificarea semnăturii se face exact ca în cazul unui mesaj criptat și semnat, adică prin comanda:

```
gpg -o fiș-ieșire --decrypt fiș-intrare
```

Semnătura detașată se generează prin comanda

```
gpg -a -o fiș-sign --detach-sign fiș-date
```

Rezultatul comenzii este scrierea în fișierul *fiș-sign* a semnăturii conținutului fișierului *fiș-date*. Fișierul produs, *fiș-sign*, este mic și conține doar semnătura; datele utile nu pot fi recuperate din el. Verificarea semnăturii se face prin comanda:

```
gpg --verify fiș-sign fiș-date
```

Semnătura detașată este utilă deoarece păstrează intact fișierul de date, nefiind nevoie de `gpg` pentru recuperarea datelor. De asemenea, permite mai multor utilizatori să semneze un același fișier de date.

În fine, semnătura în clar se poate utiliza doar dacă datele sunt text ASCII. Semnătura se generează prin comanda:

```
gpg -o fiș-ieșire --clearsign fiș-intrare
```

Fișierul astfel produs este un fișier text, care poate fi citit ușor de către utilizatorul uman. Textul original este pus între niște marcaje, iar semnătura este adăugată la sfârșit. Verificarea semnăturii se face prin comanda:

```
gpg --verify fiș
```

Semnătura în clar este utilă pentru semnarea documentelor text. Acestea rămân ușor de citit de către om și, spre deosebire de semnătura detașată, datele utile și semnătura sunt puse într-un singur fișier.

Dacă GnuPG are mai multe chei secrete (inclusiv subchei, § 11.4.1.1) utilizabile pentru semnătură, se poate specifica ce cheie trebuie utilizată pentru crearea semnăturii. Specificarea cheii se face adăugând opțiunea

```
-u cheie
```

GnuPG se utilizează curent pentru autentificarea softului liber. În acest scop, alături de programul distribuit, se distribuie un fișier ce conține semnătura detașată a fișierului ce conține programul.

Bibliografie

- [Boian 1999] FLORIAN MIRCEA BOIAN. *Programarea distribuită în Internet*. Editura Albastră, 1999.
- [Cohen 1980] DANNY COHEN. *On holy wars and a plea for peace*, 1980.
<http://www.ietf.org/rfc/ien/ien137.txt>.
- [Crstici et al. 1981] B. CRSTICI, T BÂNZARU, O. LIPOVAN, M. NEAGU, N. NEAMŢU, N. NEUHAUS, B. RENDI, D. RENDI, I STURZ. *Matematici speciale*. Editura didactică și pedagogică, Bucureşti, 1981.
- [Howard & LeBlanc 2003] MICHAEL HOWARD, DAVID LEBLANC. *Writing secure code*. Microsoft Press, 2003.
- [IANA,] <http://www.iana.org>.
- [IEEE 802.11, 1999] IEEE Computer Society. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999.
- [IEEE 802.11F, 2003] IEEE Computer Society. *802.11FTM IEEE Trial-Use Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11TM Operation*, 2003.
- [IEEE 802.11i, 2004] IEEE Computer Society. *802.11iTM Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 6: Medium Access Control (MAC) Security Enhancements*, 2004.
- [IEEE 802.1D, 2004] IEEE Computer Society. *Media Access Control (MAC) Bridges*, 2004.

- [IEEE 802.1Q, 2003] IEEE Computer Society. *802.1QTM. IEEE Standards for Local and metropolitan area networks: Virtual Bridged Local Area Networks*, 2003.
- [IEEE 802.1X, 2001] IEEE Computer Society. *802.1XTM. Port-Based Network Access Control*, 2001.
- [IEEE 802.3, 2005] IEEE Computer Society. *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*, 2005.
- [Kurose & Ross 2003] JAMES F. KUROSE, KEITH W. ROSS. *Computer networking — A top-down approach featuring the Internet*. Addison-Wesley, 2003.
- [Menezed et al. 1997] A. MENEZED, P. VAN OORSCHOT, S. VANSTONE. *Handbook of Applied Cryptography*. CRC Press, 1997.
- [Nicolau 1987] EDMOND NICOLAU. *Radiotehnica*. Manualul inginerului electronist. Editura tehnică, Bucureşti, 1987.
- [Prasad 2003] K.V. PRASAD. *Principles of Digital Communication Systems and Computer Networks*. Charles River Media, 2003.
- [RFC 1034, 1987] *Domain names — concepts and facilities*, 1987.
- [RFC 1035, 1987] *Domain names — implementation and specification*, 1987.
- [RFC 1149, 1990] *A Standard for the Transmission of IP Datagrams on Avian Carriers*, 1990.
- [RFC 1323, 1992] *TCP Extensions for High Performance*, 1992.
- [RFC 1518, 1993] *An Architecture for IP Address Allocation with CIDR*, 1993.
- [RFC 1661, 1994] *The Point-to-Point Protocol (PPP)*, 1994.
- [RFC 1700, 1994] *Assigned numbers*, 1994.
- [RFC 1918, 1996] *Address Allocation for Private Internets*, 1996.
- [RFC 1981, 1996] *Path MTU Discovery for IP version 6*, 1996.
- [RFC 1995, 1996] *Incremental Zone Transfer in DNS*, 1996.

- [RFC 1966, 1996] *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)*, 1996.
- [RFC 2045, 1996] *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, 1996.
- [RFC 2046, 1996] *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, 1996.
- [RFC 2047, 1996] *Multipurpose Internet Mail Extensions (MIME) Part Three: Message Header Extensions for Non-ASCII Text*, 1996.
- [RFC 2104, 1997] *HMAC: Keyed-Hashing for Message Authentication*, 1997.
- [RFC 2183, 1997] *Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field*, 1997.
- [RFC 2317, 1998] *Classless IN-ADDR.ARPA delegation*, 1998.
- [RFC 2440, 2007] *OpenPGP Message Format*, 2007.
- [RFC 2460, 1998] *Internet Protocol, Version 6 (IPv6) Specification*, 1998.
- [RFC 2463, 1998] *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, 1998.
- [RFC 2822, 2001] *Internet Message Format*, 2001.
- [RFC 3156, 2001] *MIME Security with OpenPGP*, 2001.
- [RFC 3596, 2003] *DNS Extensions to Support IP Version 6*, 2003.
- [RFC 3748, 2004] *Extensible Authentication Protocol (EAP)*, 2004.
- [RFC 4253, 2006] *The Secure Shell (SSH) Transport Layer Protocol*, 2006.
- [RFC 4346, 2006] *The Transport Layer Security (TLS) Protocol Version 1.1*, 2006.
- [RFC 765, 1985] *File Transfer Protocol (FTP)*, 1985.
- [RFC 791, 1981] *Internet Protocol — DARPA Internet Program Protocol Specification*, 1981.
- [RFC 792, 1981] *Internet Control Message Protocol — DARPA Internet Program Protocol Specification*, 1981.

- [RFC 793, 1981] *Transmission Control Protocol — DARPA Internet Program Protocol Specification*, 1981.
- [RFC 822, 1982] *Standard for the format of ARPA Internet text messages*, 1982.
- [Rogaway 1995] P. ROGAWAY. *Problems with Proposed IP Cryptography*, 1995. <http://www.cs.ucdavis.edu/~rogaway/papers/draft-rogaway-ipsec-commen%ts-00.txt>.
- [S/MIME,] *S/MIME Mail Security (smime)*.
<http://www.ietf.org/html.charters/smime-charter.html>.
- [Spătaru 1965] AL. SPĂTARU. *Teoria transmisiunii informației*. Editura Tehnică, Bucureşti, 1965.
- [Stevens 1994] RICHARD STEVENS. *TCP-IP illustrated*. Addison-Wesley, 1994.
- [Tanenbaum 1995] ANDREW S. TANENBAUM. *Distributed Operating Systems*. Prentice Hall, 1995.
- [Tanenbaum 2003] ANDREW S. TANENBAUM. *Rețele de calculatoare*. Byblos, 2003.

Index

Speciale

0.0.0.0, 303, 341
0.0.0.0/8, 303
10 Base T, 268
10.0.0.0/8, 303, 349
100 Base Tx, 271
1000 Base T, 272
127.0.0.0/8, 303
127.0.0.1, 303
172.16.0.0/12, 303, 349
192.168.0.0/16, 303, 349
224.0.0.0/4, 303
240.0.0.0/4, 303

A

access point, 286
ACK, *Vezi confirmare*
acknowledge, *Vezi confirmare*
ad hoc (wireless), *Vezi IBSS*
adresă
 fizică, 98, 266
 Internet, 294, 297, 300
 IP, *Vezi adresă, Internet*
 MAC, 98, *Vezi adresă fizică*, 340
 privată, 303, 348, 349
 de rețea, 119
 în subrețea, 297
 unei subrețele, 302
 tranzlație, 349
adversar, O entitate care interceptează sau modifică mesajele schimbate între alte două entități, cu scopul obținerii sau modificării informației transmise. *Vezi pg. 149*

activ, Un adversar care interceptează și modifică după voie mesajele schimbate între două entități. *Vezi pg. 149*
pasiv, Un adversar care interceptează doar comunicația, fără a o modifica. *Vezi pg. 149*

agent

de autentificare, 380

Aloha, 100

AM, *Vezi modulație de amplitudine*

antena, 78

anycast, 17

AP, *Vezi access point*

ARP, 340

atenuare, 61

factor de, Raportul între puterea semnalului măsurat la bornele emițătorului și puterea semnalului măsurat la bornele receptorului. *Vezi pg. 61*

AUI, 278

autentificare

entitate, 150

mesaj, 149

sursă, 150

B

B, *Vezi bel*

bandă

laterală, 70

lățime de, Diferența dintre frecvența maximă și frecvența minimă a benzii de trecere. Prin abuz

- de limbaj, mai are sensul de debit maxim de transmitere a informației al unui dispozitiv. Vezi pg. 65, 72
- de trecere**, Interval de frecvențe în care dacă se încadrează spectrul unui semnal, semnalul se transmite cu distorsiuni acceptabil de mici prin dispozitivul considerat. Prin abuz de limbaj, mai are sensul de debit maxim al unui canal de transmitere a informației. Vezi pg. 65, 72
- de trecere (fibră optică)**, 92
- Basic Service Set**, Vezi *celulă wireless*
- baza 64**, Vezi *codificare în baza 64*
- BCD**, Vezi *codificare binar-zecimală (pentru numere)*
- beacon**, 286, 288
- bel**, Pseudo-unitate de măsură pentru logaritmul raportului între două mărimi, de regulă mărimile fiind puterile a două semnale. Indică faptul că numărul din fața unității este un logaritm zecimal. Are simbolul B. Este utilizat mai mult submultiplul numit *decibel*. Vezi pg. 62
- BNC**, 279
- broadcast**, Vezi *difuziune completă*, 140
- BSS**, Vezi *celulă wireless*
- BSS-ID**, 285
- C**
- cablu**
- inversor, 269
 - unu-la-unu, 269
- canal**
- de comunicație, 25
 - continuu, 25
 - discret, 25
 - cu perturbații, Vezi *canal cu zgomot cu zgomot*, 51
- capacitate**, 17
- caracter**
- de evitare, 231
- Cat 3**, 269
- Cat 5**, 271
- celulă**
- wireless**, 285
- certificat**, 182
- cheie**, 152
- de durată lungă, 175
 - efemeră, 175
 - de sesiune, Vezi *cheie efemeră*
 - stabilire, 150, 174
- CIDR**, 304
- cifrare**, Vezi *criptare*
- cifru**, 151
- bloc**, 157
 - flux**, 157
- ciphertext**, Vezi *text cifrat*
- clasă**
- DNS, 334
 - IP, 303, 303
- clear to send**, 288
- cod**, 25
- corector de erori**, 52
 - detector de erori**, 52
 - instantaneu**, 31
 - de lungime fixă**, 28
 - prefix**, 27
 - unic decodabil**, 27
- codificare**
- în baza 64, 231
 - big endian, 255
 - binar-zecimală (pentru numere), 216
 - binară (pentru numere), 255
 - hexazecimală, 230
 - little endian, 255
 - rețea (pentru numere), 255
 - text (pentru numere), 216
- coliziune**, 266, 277
- comutator**, 267
- confidențialitate**, 149
- confirmare**, 102
- și retransmitere, 103, 318
- congestie**, 308
- controlul fluxului**, 114
- corectarea erorilor**, 51
- criptare**, 151
- cryptographic hash function**, Vezi *dispersie criptografică, funcție de*
- CSMA**, 100
- CSMA/CA**, 101, 287
- CSMA/CD**, 101, 277

CTS, *Vezi clear to send*

D

date

de control, 22

speciale (TCP), 330

utile, 21, 59

dB, *Vezi decibel*

dBm, *Vezi decibel-miliwatt*

debit, 17

decibel, Pseudo-unitate de măsură având ca valoare o zecime de *bel*. Are simbolul dB. *Vezi pg. 62*

decibel-miliwatt, Pseudo-unitate de măsură pentru logaritmul puterii unui semnal, indicând logaritmul, în decibeli, ai raportului dintre puterea semnalului măsurat și o putere de 1 mW. *Vezi pg. 62*

decriptare, 151

decryption, *Vezi decriptare*

descifrare, *Vezi decriptare*

destinație, 25, 59

detectarea erorilor, 51

diafonie, Zgomot ce are ca proveniență un semnal transmis pe un mediu apropiat fizic de mediul ce transmite semnalul considerat. *Vezi pg. 62*

difuziune, 17

completă, 17, 140

selectivă, 17, 140

dirijare, 126, 298

dispersie, 166

criptografică

funcție de, 166

funcție de, 166

intermodală, 92

distorsiune, Modificare determinată a semnalului recepționat față de cel emis, diferită de întârziere și atenuare. Distorsiunea se manifestă la fel oricâteori se transmite un același semnal prin același dispozitiv, în opoziție cu *zgomotul* care este aleator. *Vezi pg. 62*

distribution system, 286

DS, *Vezi Distribution System*

duplex

legătură Ethernet, 268

E

echo

reply, *Vezi ecou, răspuns*

request, *Vezi ecou, cerere*

ecou

cerere, 307

răspuns, 307

ecran, 73

eficiență

unui cod, 42

emițător, 25, 59

encryption, *Vezi criptare*

envelope, *Vezi plic*

F

fals

ales, 165

existent, 165

fibră

monomod, 92

multimod, 91

filtru

anti-spam, 371

IP, 315, 343

MAC, 282

firewall, *Vezi filtru IP*

flow control, *Vezi controlul fluxului*

FM, *Vezi modulație de frecvență*

forgery

chosen, *Vezi fals ales*

existential, *Vezi fals existent*

format, *Vezi codificare*

frecvență, 77

purtătoare, 69

FTP, 383

G

gateway, 299

default, 300

H

hash function, *Vezi dispersie, funcție de*

host, *Vezi nod final*

htonl, 255

htons, 255

HTTP, 384

HTTPS, 391

hub, 267

I

IBSS, 286

ICMP, *Vezi Internet Control Message Protocol*

ICMPv4, 304

ICMPv6, 304

IMAP, 356

impedanță

caracteristică, 74

de ieșire, 75

de intrare, 75

informație, 25

cantitate de, 40

infrastructură (wireless), 286

întârziere, 61

integritate

verificare, 150

interfață

de rețea, 265, 296

Internet, 1. Protocol de comunicație de nivel rețea. 2. Rețea la scară mondială construită pe baza protocolului internet *Vezi pg. 293*

Internet Control Message Protocol, 304

intrus, *Vezi adversar*

IP, *Vezi protocolul Internet*

IPv4, 294

IPv6, 294

K

key, *Vezi cheie*

ephemeral, *Vezi cheie efemeră*

long-term, *Vezi cheie de durată lungă*

session, *Vezi cheie efemeră*

L

lățime

de bandă, *Vezi bandă, lățime*

lob

al antenei, 81

lungimea de undă, *Vezi undă, lungime de*

M

magistrală, 75, 265

mail

transfer agent, 355, 364

user agent, 355

managed (wireless), *Vezi infrastructură*

mască

de rețea, 302

master

DNS, 337

mediu

de transmisie, Dispozitiv capabil să transmită la distanță o acțiune fizică de la *emițător* la *receptor*. *Vezi pg. 59*

mesaj, 27

microunde, 77

MIME, 360

mod

de propagare (fibre optice), 91

modulație, 68

de amplitudine, 69

în cuadratură, 70

de fază, 70

de frecvență, 70

MTA, *Vezi mail transfer agent*, 367

MUA, *Vezi mail user agent*

multicast, *Vezi difuziune selectivă*

multimod (fibră optică), *Vezi fibră multimod*

multiplexare

în frecvență, Procedeu prin care mai multe comunicații simultane pot partaja același mediu fizic prin transmiterea semnalelor corespunzătoare comunicațiilor prin modulație utilizând frecvențe purtătoare diferite.

în lungimea de undă, Procedeu de multiplexare în care mai multe semnale optice utilizând lungimi de undă diferite sunt transmise prin aceeași fibră optică. *Vezi pg. 93*

în timp, 117

MX, 367, 369

N

NAT, *Vezi adresă, translație*

nerepudiabilitate, 149

network

address translation, *Vezi adresă, translație*
interface card, *Vezi interfață de rețea*
name (wireless), *Vezi SSID*
NIC, *Vezi interfață de rețea*
nod, 119, 293
 final, 119, 293
 intermediar, 119, 293
nonce, *Vezi număr unic*
notație
 zecimală
 cu punct, 301
 simplă, 301
ntohl, 255
ntohs, 255
număr
 de secvență, 104, 318
 unic, 171
nume
 de domeniu, 332

O

OOB, *Vezi date speciale*
out of band, *Vezi date speciale*

P

pachet, 293
 Internet, 295
paritate, 55
pereche
 torsadată, 73, 268
 torsadată neecranată, 269
ping, *Vezi ecou, cerere*
plaintext, *Vezi text clar*
plic, 357
polinom generator, 57
pong, *Vezi ecou, răspuns*
POP3, 356
port, 266
 TCP, 327
 UDP, 331
portal, 286
prag de sensibilitate, 62
prefix
 de rețea, 297, 302
priză
 vampir, 278

prospețime
 verificare, *Vezi verificare prospețime*
protocol, 16, 22
punct la punct, 16
purtătoare, *Vezi frecvență, purtătoare*

R

raport semnal/zgomot, 62
receptor, 25, 59
redundanță, 42
regulă
 de dirijare, 298
reliable
 transmission, *Vezi transmisie sigură*
repeater, *Vezi repeto*
repetor, 266
reprezentare
 a informației, 25
request to send, 288
rețea
 privată, 348
RJ45, 269
round-trip time, *Vezi timp dus-întors*
router, *Vezi nod intermediar*
RTS, *Vezi request to send*
RTT, *Vezi timp dus-întors*
rută, 119
ruter, *Vezi nod intermediar*

S

securizare, 149
semi-duplex, 97
semnal, Mărimea fizică ce măsoară acțiunea produsă de emițător și transmisă de către mediu până la receptor și care este utilizată efectiv ca purtătoare a informației. *Vezi pg. 25, 59*
 modulat, 69
 sinusoidal, 63
simbol de cod, 26
slave
 DNS, 337
SMTP, 355, 364
socket, 233
spectru, 64
spoofing, 346
 blind, 326

IP, 325
SSH, 375
SSID, 286
SSL, 389
stație, *Vezi nod final*
subrețea, 136
 IP, 296
sufix
 în subrețea, 297
sursă, 25, 59
switch, *Vezi comutator*

T

tabelă
 de dirijare, 126, 298
TCP, 317
terminator, 76
text cifrat, 151
text clar, 151
time
 to live, *Vezi timp, de viață*
timp
 dus-întors, 18, 329
 de propagare, 18
 de valabilitate (DNS), 334, 337
 de viață (pachet IP), 308
tip
 înregistrare DNS, 334
TLS, 389
translația adresei, *Vezi adresă, translație*
transmisie sigură, 103
trunking, 282
TTL, *Vezi timp, de viață*
twisted pair, *Vezi pereche torsadata*

U

uint16_t, 255

uint32_t, 255
undă
 electromagnetică, 77
 lungime de, 77
 radio, 77
unicast, 16
unshielded twisted pairs, *Vezi pereche torsată neecranată*
UTP, *Vezi perechi torsate neecranate*

V

vecin, 119
vector de inițializare, 154
verificare
 integritate, *Vezi integritate, verificare*
 prospețime, 150
VLAN, 283
VLAN-ID, 283

W

wavelength division multiplexing, *Vezi multiplexare în lungimea de undă*
WDM, *Vezi multiplexare în lungimea de undă*
window manager, 382

Z

zgomot, *Modificare nedeterministă a semnalului recepționat față de cel emis. Pentru comparație, vezi și distorsiune. Vezi pg. 62*
zonă
 DNS, 335