Reţele de calculatoare

Principii

Radu-Lucian Lupşa

Aceasta este ediţia electronică a cărţii *Reţele de calculatoare*, publicată la Casa Cărţii de Ştiinţă, în 2008, ISBN: 978-973-133-377-9.

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

Cuprins

Principii

Protocoale

Prefaţă

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

Lucrarea de faţă se adresează în principal programatorilor de aplicaţii în reţea şi administratorilor de reţele complexe. Sunt presupuse, din partea cititorului, cunoştinţe de bază de programare, precum şi privind funcţionarea sistemelor de operare.

Ca un avertisment pentru programatori, menţionăm că, deşi lucrarea tratează chestiuni de nivel mult mai coborât decât cel al platformelor şi bibliotecilor utilizate în mod normal în aplicaţiile în reţea, este totuşi utilă în vederea unei bune înţelegeri a acestor platforme şi biblioteci.

Tot ceea ce are legătură într-un fel sau altul cu calculatoare are două caracteristici: se dezvoltă foarte repede şi est foarte complex. Reţelele de calculatoare nu fac excepţie. Ca urmare, este extrem de uşor 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 şi evoluează mult mai lent decât construcţiile tehnice elaborate pe baza lor. În consecinţă, prima parte a lucrării de faţă, *principii*, este dedicată studierii problemelor ce trebuie rezolvate de o reţea de calculatoare, precum şi a principiilor construcţiei posibilelor soluţii ale acestor probleme.

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

Lucrarea de faţă este rodul experienţei autorului în activităţi legate de administrarea reţelei de calculatoare a Departamentului de Informatică al Facultăţii de Matematică şi Informatică din cadrul Universităţii Babeş-Bolyai Cluj-Napoca, în predarea unui curs de *Reţele de calculatoare* la această fac-

ultate, precum şi din activitatea de cercetare desfăşurată 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 si managementul registrelor nationale de boli cronice netransmisibile* - CRONIS. Seturile mari de date ce se vehiculează în sistemul medical, precum şi nevoia de confidenţialitate şi securitate a lor, cer o foarte bună cunoaştere şi punere în practică a noţiunilor legate de codificarea informaţiei, de metode şi protocoale criptografice, de aplicaţii în reţele etc.

Capitolul 1

Introducere

Prin *reţea de calculatoare* înţelegem un sistem (constând din componente hard şi soft) care interconectează nişte calculatoare, permiţând unor programe ce se execută pe aceste calculatoare să comunice între ele.

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

1.1. Serviciile oferite de reţea

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

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

Principalele funcţii oferite de reţea, apelabile de către un proces utilizator, sunt o funcţie care trimite date de la procesul curent spre partenerul sau partenerii de comunicaţie şi o funcţie care recepţionează datele trimise spre procesul curent. În aceste funcţii este necesară desemnarea destinatarului spre care procesul emiţător doreşte transmiterea datelor, respectiv a emiţătorului dinspre care procesul receptor solicită să primească date. În acest scop, fiecare

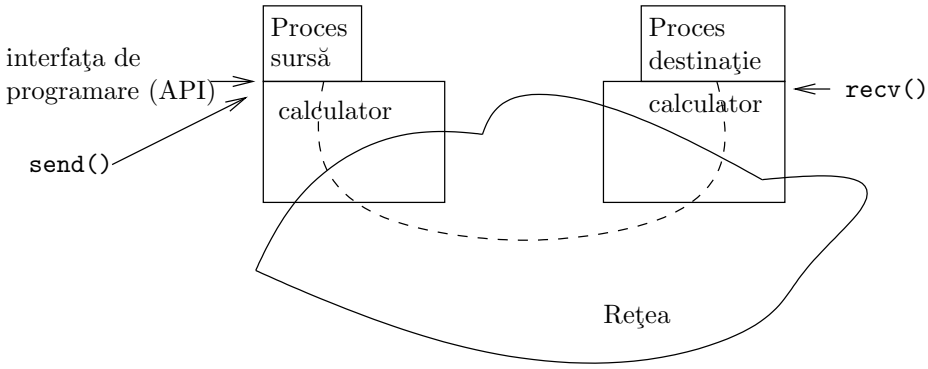


Figura 1.1: Reţeaua de calculatoare, din punctul de vedere al proceselor aplicaţie. Funcţionalitatea reţelei este oferită prin funcţii apelabile din procesele utilizator. Reţeaua oferă o aplicaţiilor o cale de transmisie a datelor (linia punctată). Construcţia efectivă a reţelei nu este vizibilă aplicaţiilor.

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

Pe lângă aceste funcţii de bază, reţeaua mai oferă funcţii pentru configurarea diferiţilor parametrii. O parte dintre aceşti parametri fixează rolul şi locul diverselor componente în cadrul reţelei (de exemplu, fiecare calculator trebuie să-şi cunoască propria adresă). Alţi parametrii sunt legaţi de calitatea serviciilor oferite de reţea (debit de transfer de date, timp de propagare, etc).

Datele transmise de procesele utilizator sunt de obicei şiruri arbitrare de octeţi. Rolul reţelei este de-a transmite întocmai şirul de octeţi trimis de procesul sursă către procesul destinaţie. Semnificaţia, pentru procesul destinaţie, a unui şir de octeţi transmis face obiectul unei înţelegeri (*protocol*) între procesele utilizator. La proiectarea reţelei nu ne interesează ce fac procesele utilizator cu datele transferate; la proiectarea programelor utilizator nu ne interesează cum lucrează reţeaua pentru a transmite datele.

În continuare vom trece în revistă principalele caracteristici ale serviciului oferit de reţea proceselor de aplicaţie.

O comunicaţie poate fi, după numărul destinatarilor:

- *punct la punct*, dacă există un singur destinatar. În mod obişnuit, destinatarul este selecţionat explicit de către procesul emiţător; o astfel de comunicaţie 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 reţeaua să aleagă destinatarul comunicaţiei, în funcţie de distanţa faţă de emiţător, dintr-o mulţime

specificată de destinatari posibili. Un astfel de comunicaţie se numeşte *anycast*.

- *difuziune*, dacă există mai mulţi destinatari. Distingem *difuziune completă* (engl. *broadcast*), în care destinatari sunt toate calculatoarele dintr-o reţea, şi *difuziune selectivă* (engl. *multicast*), în care destinatarii sunt o submulţime aleasă a calculatoarelor din reţea.

Serviciul de comunicaţie oferit de reţea poate fi de tip *conexiune* sau de tip transport de *datagrame*:

- În cazul *conexiunilor*, în cadrul comunicaţiei între două procese se disting trei faze:
  - deschiderea conexiunii, în cadrul căreia sunt făcute nişte pregătiri, inclusiv alocarea unor resurse pentru comunicaţie;
  - comunicaţia propriu-zisă, în care unul sau ambele procese transmite un şir de pachete sau de biţi celuilalt proces;
  - închiderea conexiunii, în cadrul căreia se eliberează resursele alocate la deschidere.

- În cazul transportului de datagrame, procesul emiţător pregăteşte un ansamblu, numit *datagramă* (prin analogie cu *telegramă*), cuprinzând un şir de biţi destinat procesului receptor şi anumite informaţii necesare livrării (adresa destinatarului). Apoi transmite datagrama reţelei de calculatoare, care o transmite procesului receptor. Mai multe datagrame trimise de acelaşi proces sursă către acelaşi proces destinaţie sunt transmise independent una de alta, ceea ce duce, în general, la posibilitatea inversării ordinii de recepţie faţă de ordinea de emisie a datagramelor.

Principalii parametri de calitate ai serviciului oferit de reţea sunt:

Capacitatea de transport oferită de reţea, sau *debitul maxim acceptat*, este raportul dintre numărul de biţi transportaţi în cadrul unei comunicaţii şi timpul în care aceştia sunt transmişi. Echivalent, capacitatea este inversul duratei medii între trecerea, printr-un punct dat al reţelei, a doi biţi consecutivi ai unei comunicaţii.

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 acelaşi punct, a ultimului bit. Timpul de transfer este egal cu raportul dintre dimensiunea blocului şi debitul cu care se face transferul.

Capacitatea oferită de reţea unei legături poate să varieze datorită variaţiei debitului altor comunicaţii care partajează aceleaşi echipamente.

Există aplicaţii, de exemplul legate de transfer de fişiere, pentru care este important ca reţeaua să ofere o capacitate medie cât mai mare. Penttru alte aplicaţii, cum ar fi telefonia, transmisia video (de exemplu pentru teleconferinţe) sau alte aplicaţii 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 entităţi este timpul scurs între momentul în care entitatea sursă emite un bit şi momentul în care acel bit ajunge la destinaţie. Timpul de propagare rezultă din însumarea timpului de propagare a semnalului de-a lungul mediului de comunicaţie cu diverşii timpi de aşteptare a datelor în diverse zone tampon. De remarcat că timpul de propagare a semnalului este egal cu distanţa de la emiţător la receptor împărţită la viteza de propagare a semnalului, iar viteza de propagare nu poate depăşi 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 emiţător până la finalul recepţiei blocului de către receptor este egal cu suma dintre timpul de transfer şi 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 comunicaţie până la primirea răspunsului din partea acestuia. Timpul dus-întors este suma dintre timpii de propagare pentru cele două sensuri şi timpul de procesare pentru crearea răspunsului.

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

- La unele aplicaţii timpul de propagare nu este prea important. De exemplu, la transferul unui fişier mare, la care oricum timpul de transfer este mare, timpul de propagare influenţează foarte puţin timpul total necesar transmiterii fişierului.
- 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 faţă de evenimentele transmise, însă este important să nu fie momente în care imaginea „îngheaţă“ datorită creşterii timpului de propagare şi 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 aplicaţii în care entităţile ce comunică transmit mesaje scurte şi trebuie să aştepte răspunsul la mesajul precedent pentru a putea genera mesajul următor. Exemple de astfel de aplicaţii sunt: telefonie, videoconferinţe, sesiuni interactive la distanţă.

Posibilitatea existenţei erorilor de transmisie: Erorile de transmisie apar ca urmare a diverselor perturbaţii ce afectează transmiterea semnalelor. Există metode de-a micşora 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 reţeaua). Metodele de reducere a probabilităţii erorilor de transmisie sunt studiate în § 2.4 şi § 4.1.

Transmisia sigură înseamnă ca fiecare mesaj al entităţii sursă să ajungă exact într-un singur exemplar la destinaţie (să nu se piardă şi să nu fie duplicat) şi mai multe mesaje transmise de către o aceeaşi sursă spre o aceeaşi destinaţie să ajungă la destinaţie 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 reţea sau chiar din cauză că emiţătorul transmite cu debit mai mare decât este capabil receptorul să preia informaţia transmisă. Duplicarea sau inversarea mesajelor pot fi cauzate de modificări ale configuratiei sau încărcării reţelei în timpul trecerii pachetelor prin reţea. Realizarea transmisiei sigure este studiată în § 4.3 şi § 4.4.

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

Securitatea comunicatiei înseamnă că un adversar care controlează o parte din reţea să nu poată obţine informaţia transmisă, să nu poată modifica datele transmise fără ca acest lucru să fie detectat de către receptor şi să nu poată impersona vreuna dintre entităţile ce comunică. Securitatea comunicaţiei se obţine prin metode criptografice, studiate în capitolul 6.

1.2. Principalele elemente ale unei reţele de calculatoare

Pentru ca două dispozitive aflate la distanţă unul de celălalt să poată comunica, este nevoie ca cele două dispozitive să fie legate printr-un *mediu de comunicaţie* care permite propagarea variaţiei unei mărimi fizice. Mediul fizic, împreună cu dispozitivele de adaptare între reprezentarea locală a informaţiei şi reprezentarea pe mediul de transmisie constituie *nivelul fizic* al reţelei. Nivelul fizic este deci un modul care permite transmisia unui şir de biţi între două dispozitive legate direct unul de celălalt. Constructiv, nivelul fizic este constituit din: cablul electric, fibra optică sau, după caz, antenele de emisie-recepţie, eventuale amplificatoare sau repetoare, plăcile de reţea din calculatoare şi *driver*-ele plăcilor de reţea. Construcţia 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 şi transmisia nesigură. Pentru contracararea acestora, de-o parte şi 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 parţial prin hard (parte a plăcii de reţea) şi parţial prin soft (parte a *driver*-ului plăcii de reţea). Construcţia 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 lanţ de calculatoare (sau alte dispozitive) legate fizic fiecare cu următorul din lanţ. Realizarea unei astfel de legături cade în sarcina *nivelului reţea*, constituit din câte un modul în fiecare calculator al reţelei. Modulul de reţea este construit prin soft, în nucleul sistemului de operare al fiecărui calculator din reţea. Construcţia şi funcţionarea nivelului reţea este studiată în capitolul 5.

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

Relaţiile dintre aceste componente sunt reprezentate în figura 1.2.

Fiecare dintre nivele oferă nivelului superior o interfaţă care cuprinde în principal funcţii de trimitere şi de recepţie a datelor. Aceste funcţii sunt

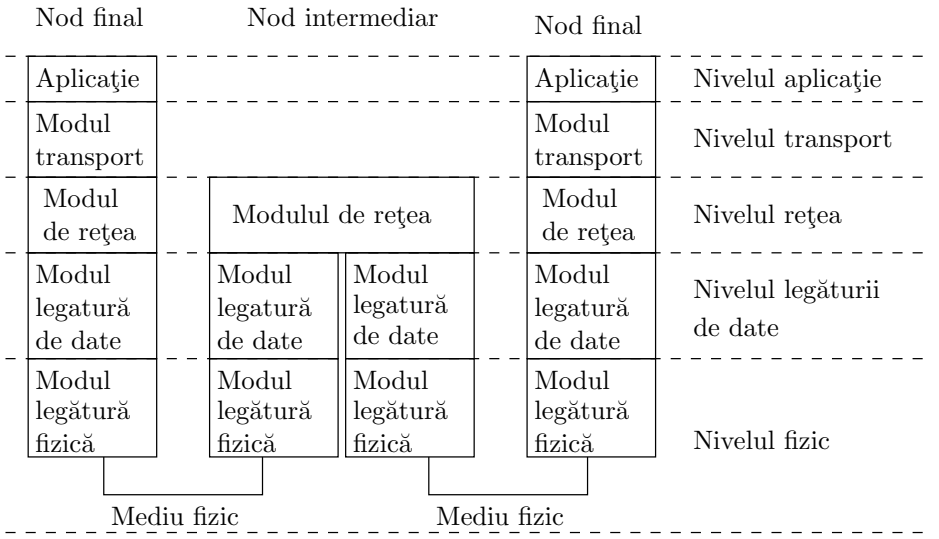


Figura 1.2: Componentele unei părţi dintr-o reţea de calculatoare. Sunt figurate doar componentele implicate în comunicaţia dintre două aplicaţii. Cele două aplicaţii se execută pe două calculatoare între care nu există o legătură directă, dar există o legătură printr-un nod intermediar.

similare celor oferite de reţea aplicaţiilor (aşa 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 şi cu riscul ca datele să fie alterate în timpul transferului sau să se piardă complet. Nivelul legăturii de date oferă nivelului reţea servicii de transfer de date mai sigure, dar în continuare cu restricţia că transferul este posibil doar între calculatoare conectate direct. Nivelul reţea oferă nivelului transport servicii de transfer de date între orice două calculatoare din reţea, dar încă neadecvate utilizării directe de către aplicaţii (lipsa transmisiei sigure, comunicaţie posibilă doar pentru un singur proces aplicaţie la un moment dat, etc.).

Construcţia fiecăruia dintre nivele este independentă de construcţia celorlalte (contează doar interfaţa dintre ele şi parametrii de calitate a serviciului oferit de un nivel celui imediat superior). De exemplu, în proiectarea nivelului reţea, nu ne interesează nici ce aplicaţii vor utiliza reţeaua (acelaşi nivel reţea din Internet este utilizat de aplicaţii de poşta electronică, web, telefonie prin Internet şi videoconferinţe), nici cum este construit nivelul fizic (perechi de conductoare, fibre optice sau legături radio prin satelit).

Modulele, de pe acelaşi nivel, din noduri diferite îşi 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 şi *date de control* necesare coordonării activităţilor modulelor din cadrul nivelului. Regulile de reprezentare a acestor date, de organizare a acestora în mesaje, precum şi regulile după care se trimit mesajele între modulele aceluiaşi nivel alcătuiesc *protocolul de comunicaţie* al nivelului respectiv.

Funcţionarea corectă a unei reţele necesită respectarea, de către toate modulele implicate, a protocoalelor de comunicaţie stabilite.

1.3. Premise generale în elaborarea şi implementarea protocoalelor în reţele

Pe lângă raţiunile pur funcţionale, studiate pe larg în capitolele următoare, în elaborarea şi implementarea protocoalelor intervin raţiuni practice, pe care le vom înşira pe scurt în continuare:

- Deoarece o reţea este formată din multe componente, frecvenţa cu care se întâmplă ca cel puţin o componentă a unei reţele să nu funcţioneze corect este mare. Este necesar ca o defecţiune să afecteze cât mai puţin din reţea, iar componentele a căror defectare duce la căderea întregii reţele trebuie să fie cât mai puţine, eventual nici una.
- Găsirea unei pene într-un sistem complex este, în general, dificilă. Reţeaua trebuie să ofere mecanisme prin care orice defecţiune să fie uşor de localizat.
- Implementări diferite ale unui protocol se pot abate în moduri diferite de la specificaţia protocolului. Este bine ca mici abateri ale partenerului de comunicaţie să fie tolerate. Rezultă de aici principiul că o implementare trebuie *să fie strictă cu ceea ce transmite şi tolerantă cu ceea ce recepţionează.*
- Reţeaua trebuie să funcţioneze 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 înţeleasă şi soluţiile 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 foloseşte nimeni deoarece înlocuirea sistemelor existente ar fi mai scumpă decât avantajul adus de protocolul mai bun.
- Protocoalele totuşi evoluează, iar oprirea întregii reţele în vederea schimbării echipamentelor afectate de schimbarea protocolului nu este rezonabilă.

Ca urmare, la o schimbare de protocol trebuie avut în vedere existenţa unei perioade de tranziţie în timpul căreia echipamentele noi trebuie să poată comunica cu cele vechi. Tranziţia este mult uşurată dacă protocolul vechi prevede anumite facilităţi. O posibilitate este ca în protocol să se prevadă o fază de negociere în care fiecare entitate anunţă ce versiuni de protocol şi ce extensii de protocol cunoaşte, iar apoi comunicaţia decurge conform versiunii celei mai recente şi cu cele mai multe extensii suportate de ambii parteneri. Altă posibilitate este stabilirea, de la prima versiune a protocolului, a acţiunilor unui dispozitiv, ce implementează o versiune veche a protocolului, la primirea unui mesaj neprevăzut în acea versiune.

- Cerinţe diferite ale diferitelor aplicaţii duc la tendinţa de-a elabora protocoale complexe, care să satisfacă pe toată lumea. Protocoale complexe duc la implementări scumpe şi cu riscuri mari de-a avea erori. Este preferabil un protocol care să ofere câteva operaţii simple care să poată fi combinate după dorinţa aplicaţiei 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 parţial; metodele utilizabile în acest scop sunt similare cu cele descrise mai sus pentru facilitarea evoluţiei protocoalelor.

Capitolul 2

Noţiuni de teoria informaţiei

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

Prin *informaţie* înţelegem cunoştinţele unei entităţi.

În cele ce urmează, ne va interesa problema transmiterii unei informaţii de la o *sursă* la o *destinaţie*. Informaţia de transmis nu este cunoscută iniţial nici de destinaţie, nici de sistemul de transmitere. Ca urmare, *a priori* informaţia de transmis poate fi văzută ca o variabilă aleatoare.

Comunicaţia dintre sursă şi destinaţie se desfăşoară prin intermediul unui *canal de comunicaţie*. Canalul de comunicaţie este capabil să transmită fie o mărime variabilă în timp, numită *semnal* (în esenţă, o funcţie reală continuă), caz în care canalul este numit *continuu*, fie un şir de simboluri dintr-o mulţime finită, caz în care canalul este numit *discret*.

Deoarece canalul nu poate transmite direct informaţia sursei, între sursă şi canal avem nevoie de un dispozitiv, numit *emiţător*, care transformă informaţia utilă, produsă de sursă, într-un semnal sau, după caz, într-un şir de simboluri. Similar, între canal şi destinaţie se plasează un dispozitiv, numit *receptor*, al cărui rol este de-a efectua operaţia inversă, şi anume de-a extrage din semnal sau din şirul de simboluri informaţia utilă pentru destinaţie (fig. 2.1).

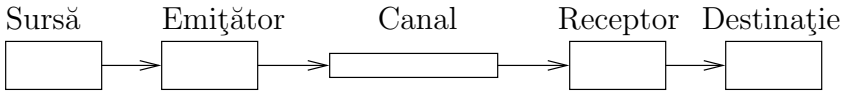


Figura 2.1: Transmisia informaţiei de la sursă la destinaţie

Semnalul sau, după caz, şirul de simboluri ce tranzitează canalul se numeşte *reprezentarea informaţiei*. Regulile de corespondenţă dintre informaţia utilă şi reprezentarea sa poartă denumirea de *schemă de reprezentare a informaţiei*, *schemă de codificare a informaţiei* sau *cod*.

Ca exemplu, o limbă scrisă este o schemă de reprezentare a informaţiei, pentru un canal discret a cărui mulţime de simboluri conţine literele alfabetului limbii respective, precum şi spaţiul şi semnele de punctuaţie. Un text scris într-o limbă este o *reprezentare a informaţiei*, iar conceptele din textul respectiv sunt efectiv *informaţia* conţinută în text.

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

Schema de codificare a informaţiei se presupune că este stabilită în prealabil şi este cunoscută atât emiţătorului cât şi receptorului. De asemenea, în construcţia schemei de reprezentare a informaţiei se ţine cont de caracteristicile canalului şi de caracteristici generale ale informaţiilor ce trebuie să se poată transmite, însă la elaborarea ei nu se cunosc informaţiile ce trebuiesc efectiv transmise. De exemplu, la elaborarea unei scheme de codificare a literelor dintr-un text utilizând un canal ce poate transmite doar simbolurile 0 şi 1 se poate ţine cont de frecvenţa obişnuită a literelor într-un text, dar nu şi de textul efectiv de transmis.

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

- proprietăţi generale ale codurilor,
- problema minimizării numărului de simboluri necesare a fi transmise prin canal, precum şi măsurarea cantităţii de informaţie,
- problema codificării în cazul în care canalul alterează şirul de simboluri pe care îl transmite (canal cu perturbaţii).

2.1. Problema codificării informaţiei pentru un canal discret

În cazul unui canal discret, canalul poate transmite un şir de simboluri dintr-o mulţime S, numită *mulţimea simbolurilor de cod* sau *alfabetul canalului*. Elementele lui S se numesc *simboluri de cod* sau, scurt, *simboluri*. Mulţimea S este finită şi are cel puţin două elemente. De regulă $S = \{0, 1\}$.

Pentru şirurile de simboluri de cod vom utiliza următoarele notaţii:

- S^* reprezintă mulţimea şirurilor finite de elemente din S.
- $u \cdot v$ reprezintă concatenarea şirurilor u şi v.

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

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

Mulţimea de mesaje M este nevidă şi cel mult numărabilă. De cele mai multe ori M este finită.

Definiţia 2.1 *Numim* funcţie de codificare *sau* cod *orice funcţie injectivă* $c : M \to S^*$, *unde* M *este mulţimea de mesaje, cel mult numărabilă, iar* S *este mulţimea simbolurilor de cod, finită şi având cel puţin două elemente.*

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

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

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

Un şir de mesaje $(m_1, \ldots, m_k) \in M^*$ (unde M^* desemnează mulţimea şirurilor finite de mesaje din M) va fi codificat prin şirul format prin concatenarea codificărilor mesajelor:

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

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

Definiţia 2.3 *Un cod* $c : M \to S^*$ *se numeşte:*

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

$$\hat{c}(m_1, m_2, \ldots, m_k) = c(m_1) \cdot c(m_2) \cdot c(m_k) \tag{2.1}$$

  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* $c(m_2)$ *şi 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 numeşte* lungimea codului*;*

Propoziţia 2.4 *Au loc următoarele proprietăţi:*

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

Demonstraţia este imediată.

Exemplul 2.1: Considerăm mulţimea mesajelor $M = \{a, b, c, d\}$ şi mulţimea 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, obţinut 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 obţinut prin oglindirea cuvintelor unui cod prefix se numeşte *cod sufix* şi 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 şirul de simboluri de cod 01 poate fi codifcarea mesajului c sau a şirului de mesaje ab.

2.2. Coduri cu proprietatea de prefix

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

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

În aceste situaţii, trebuie să ne extindem la clase mai largi decât cea a codurilor de lungime fixă. Aşa cum vom vedea în continuarea paragrafului de faţă, clasa codurilor prefix este suficientă în situaţiile enumerate mai sus şi, în acelaşi timp, permite decodificarea destul de simplă a transmisiei.

2.2.1. Reprezentarea arborescentă a codurilor prefix

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

- pentru fiecare nod intern, muchiile descendente sunt cel mult în număr de $|S|$ şi 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 lanţul ce uneşte rădăcina cu frunza ataşată mesajului.

Construcţia 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.

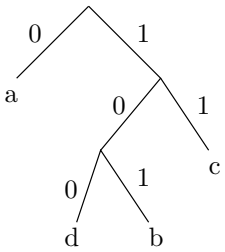


Figura 2.2: Arborele ataşat unui cod prefix

Exemplul 2.5: Fie codul prefix pentru mulţimea mesajelor

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

Algoritmul *Generează_arbore*
intrarea: M mulţime finită nevidă
    $c : M \to S^*$ cod prefix
ieşirea: T arborele asociat codului c
algoritmul:
    creează T format doar din rădăcină
    r:=rădăcina lui T
    pentru $m \in M$ execută
        $(s_1, \ldots, s_l)$:=$c(m)$
        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

şi mulţimea simbolurilor de cod $S = \{0, 1, 2\}$:

$$\begin{aligned} \text{a} &\mapsto 0 \\ \text{b} &\mapsto 10 \\ \text{c} &\mapsto 11 \\ \text{d} &\mapsto 12 \\ \text{e} &\mapsto 200 \\ \text{f} &\mapsto 201 \\ \text{g} &\mapsto 21 \\ \text{h} &\mapsto 22 \end{aligned}$$

Arborele ataşat este reprezentat în figura 2.3.

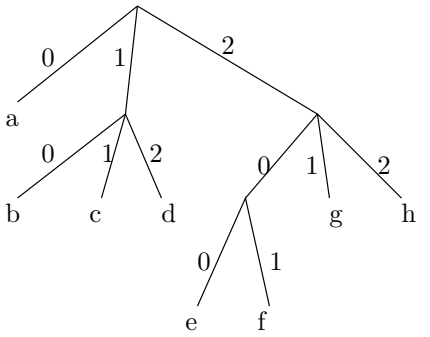


Figura 2.3: Arborele ataşat codului prefix din exemplul 2.5.

2.2.2. Decodificarea în cazul codurilor prefix

Dacă avem un şir de mesaje codificat printr-un cod prefix, decodificarea se poate face prin algoritmul 2.2. Acesta rulează în timp proporţional 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 şi prelucrat. Acest lucru este posibil numai pentru codurile prefix; din acest motiv, codurile prefix se mai numesc şi *coduri instantanee*.

Exemplul 2.6: Fie codul prefix din exemplul 2.5 (vezi fig. 2.3) şi fie şirul de decodificat:

$$s = 0112000$$

Algoritmul *Decodează*
intrarea: T arborele unui cod prefix $c : M \to S^*$
$s = (s_1, s_2, \ldots, s_l) \in S^*$ un şir finit de simboluri de cod
ieşirea: $m = (m_1, m_2, \ldots, m_k) \in M*$ şirul mesajelor a căror codificare este $s_1, \ldots, s_l$
algoritmul:
m:=ε
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 şirul s primul element; acesta are valoarea 0. Coborâm în arbore de-a lungul muchiei etichetate cu 0 şi ajungem la frunza etichetată „a“. Deoarece am ajuns la o frunză, punem mesajul din eticheta frunzai — adică „a“ — în şirul de mesaje decodificat şi revenim la rădăcină. Urmează simbolul de cod 1; coborâm de-a lungul muchiei 1 şi ajungem în nodul părinte ale nodurilor „b“, „c“ şi „d“. Urmează simbolul 1; coborâm de-a lungul muchiei 1 şi ajungem la frunza „c“; adăugăm „c“ la şirul de mesaje şi revenim la rădăcină. Continuând în acelaşi fel, vom obţine în continuare mesajele „e“ şi „a“. Şirul de mesaje transmis este deci „acea“.

2.2.3. Lungimile cuvintelor unui cod prefix

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

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

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

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

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

Vom demonstra întâi prima implicaţie, pentru cazul în care mulţimea mesajelor M este finită. Demonstraţia va fi construită prin inducţie 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 şi în consecinţă 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 $k_0 \in \mathbb{N}^*$ arbitrar, să demonstrăm că are loc şi pentru coduri de lungime maximă $k = k_0 + 1$. Pentru aceasta, să construim mulţimile de mesaje

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

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

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

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

- Dacă M_x are cel puţin 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 obţinem un nou cod prefix pentru M_x. Acest cod prefix are toate cuvintele de cod lungime cel mult k_0 şi ca urmare, conform ipotezei de inducţie, 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 puţin 1 şi 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 submulţimile M_x, obţinem:

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

În cazul unei mulţimi M numărabile, construim

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

şi notăm

$$K_l = \sum_{m\in M_k} d^{-(l_m)}.$$

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 subşir al şirului sumelor parţiale ale unei

Algoritmul *Construieşte_cod*
intrarea: $(l_m)_{m\in M} \subseteq \mathbb{N}$ satisfăcând (2.2)
ieşirea: $c : M \to S^*$ 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
                $E'{:=}E' \cup \{w \cdot x\}$
            sfârşit pentru
        sfârşit pentru
        $E{:=}E'$
        pentru $m \in M : l_m = l$ execută
            $c(m)$:= o valoare arbitrară din E
            $E{:=}E \setminus \{c(m)\}$
        sfârşit pentru
    sfârşit pentru
sfârşit algoritm

Algoritmul 2.3: Construcţia 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 şi suma ei K este la rândul ei mai mică sau egală cu 1.

Să demonstrăm acum reciproca, şi anume că inegalitatea lui Kraft implică existenţa unui cod prefix. Construcţia 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 iteraţiei $l = k$ imediat după execuţia instrucţiunii $E{:=}E'$.

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 iteraţie cuvintele din E_1 se obţin prin concatenarea câte unui simbol din S la şirul vid. Apoi, cuvintele din E_{k+1} se obţin din cuvintele rămase din E_k după atribuirea unora ca şi 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 obţine 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

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

La prima iteraţie, $|E_k| = d$ şi

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

Presupunând că (2.3) are loc la iteraţia cu $l = k$, la iteratia 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 construcţie 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 obţinute 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 obţinut şi, ca urmare, există în E_k suficiente cuvinte.♢

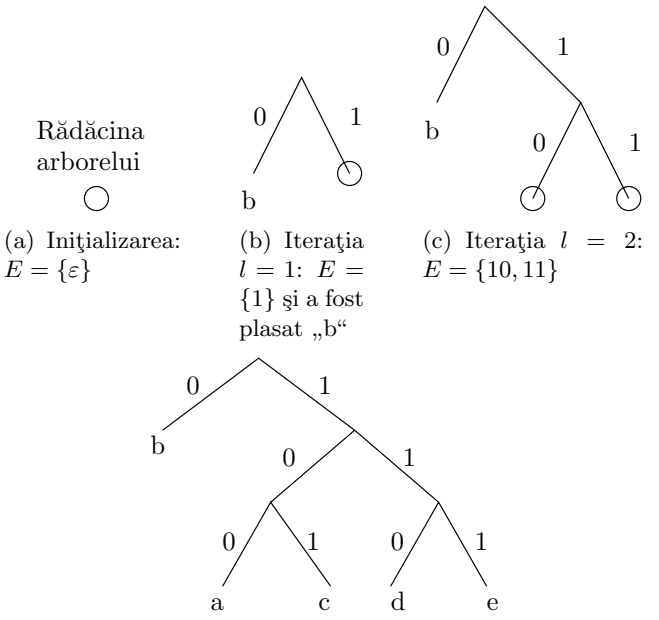
Exemplul 2.7: Dorim construirea unui cod prefix pentru mulţimea $M = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}\}$ şi mulţimea de simboluri de cod $S = \{0, 1\}$ cu următoarele lungimi ale cuvintelor de cod: $l_\mathrm{a} = 3$, $l_\mathrm{b} = 1$, $l_\mathrm{c} = 3$, $l_\mathrm{d} = 3$, $l_\mathrm{e} = 3$.

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 şi prin urmare există un cod prefix.

Construcţia propriu-zisă este arătată în figura 2.4. Cerculeţele desemnează nodurile corespunzătoare elementelor din mulţimea E.



(a) Iniţializarea: $E = \{\varepsilon\}$

(b) Iteraţia $l = 1$: $E = \{1\}$ şi a fost plasat „b“

(c) Iteraţia $l = 2$: $E = \{10, 11\}$

(d) Ultima iteraţie, $l = 3$: $E = \emptyset$ şi codul este complet generat

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

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

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

$$\sum_{m \in M}^{n} d^{-l_m} \leq 1 \tag{2.4}$$

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

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

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

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

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

Mai observăm că

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

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

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

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

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

Înlocuind în (2.6), obţinem:

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

adică

$$E^k \leq l_{\max} \cdot k. \tag{2.8}$$

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 mulţimile

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

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

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

2.3. Coduri optime

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

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

În al doilea rând, acest număr mediu de biţi 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 mulţimea M de mesaje ale sursei. Lungimea reprezentării mesajului este de asemenea o variabilă aleatoare, a cărei valoare medie este ceea ce dorim să minimizăm.

Probabilităţile diferitelor mesaje ale sursei se pot estima pe diverse căi fie analizând teoretic fenomenele pe baza cărora funcţionează sursa, fie analizând statistic şiruri de mesaje trimise de sursă. Ca exemplu, dacă mesajele sursei sunt litere ce alcătuiesc un text într-o anumită limbă, se poate determina statistic frecvenţa fiecărei litere, precum şi frecvenţele unor succesiuni de litere.

2.3.1. Cantitatea de informaţie

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

Cantitatea de informaţie purtată de un mesaj trebuie deci să fie mică dacă pentru destinatar evenimentul anunţat de mesaj era aproape sigur şi mare dacă este un eveniment total neaşteptat. Este de dorit, de asemenea, ca măsura informaţiei să fie aditivă, în sensul că privind ca un singur mesaj o succesiune de două mesaje, cantitatea de informaţie purtată de mesajul compus să fie suma cantităţilor de informaţie purtate de cele două mesaje separat.

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

De notat că cantitatea de informaţie nu are nici o legătură cu utilitatea informaţiei.

Definiţia 2.8 *Fie o sursă care emite un şir de mesaje* $m_1, m_2, \ldots, m_t \in M$. Cantitatea de informaţie *adusă de mesajul* m_t *este*

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

Altfel spus, cantitatea de informaţie adusă de un mesaj m_t în contextul (adică urmând după) m_1, m_2,...,m_{t-1} este minus logaritmul probabilităţii ca al t-lea mesaj să fie m_t, condiţionată de faptul că mesajele precedente au fost m_1, m_2,...,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 şi de poziţia (numărul de ordine) mesajului în şirul de mesaje, putem, pentru fiecare $m \in M$, să notăm cu p_m probabilitatea ca un anumit mesaj din şirul de mesaje să aibă valoarea m. Atunci cantitatea de informaţie adusă de un mesaj m este $\text{info}(m) = -\log_2 p_m$.

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

A nu se confunda bitul cu sensul de unitate de măsură pentru cantitatea de informaţie cu bitul cu sensul de cifră binară. Există o legătură între aceste noţiuni, şi anume, aşa cum vom vedea, pentru a transmite un bit de informaţie avem nevoie cel puţin de un bit (cifră binară).

Exemplul 2.8: Dacă emiţătorul anunţă receptorului rezultatul aruncării unei monede, mesajul *a căzut cu faţa în sus* poartă o cantitate de informaţie egală cu $-\log_2 \frac{1}{2} = -(-1) = 1\text{bit}$.

Exemplul 2.9: În textul acestei lucrări, 10,7% dintre litere sunt „a“, şi doar 1,1% sunt „b“. Cu aceste cunoştinţe, receptorul se va aştepta de la fiecare literă să fie „a“ cu probabilitate de 10,7% şi „b“ cu probabilitate de 1,1%. În aceste condiţii, fiecare literă „a“ poartă $-\log_2 0{,}107 \approx 3{,}224$ biţi de informaţie, şi fiecare literă „b“ poartă $-\log_2 0{,}011 \approx 6{,}5$ biţi.

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

Definiţia 2.9 *Fie o sursă de informaţie ce emite mesaje dintr-o mulţime M, fiecare mesaj $m \in M$ având o probabilitate p_m de-a fi emis. Se numeşte* entropia *sursei de informaţie cantitatea*

$$H = -\sum_{m \in M} p_m \cdot \log p_m \tag{2.9}$$

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

2.3.2. Lungimea medie a cuvintelor de cod

Definiţia 2.10 *Fie o sursă ce emite mesaje dintr-o mulţime M. Pentru fiecare $m \in M$, fie p_m probabilitatea mesajului m şi fie $c : M \to S^*$ un cod unic decodabil. Se numeşte* lungimea medie a *cuvintelor codului c valoarea*

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

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

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

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

$$\bar{l} \geq \frac{H}{\log_2 |S|}. \tag{2.10}$$

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

Definiţia 2.13 *Se numeşte* eficienţa *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 mulţimea simbolurilor de cod.*

Se numeşte redundanţa relativă *valoarea $1 - \eta$.*

Eficienţa şi redundanţa relativă sunt numere cuprinse între 0 şi 1.

Valoarea minimă, dată teorema 2.12, pentru lungimea medie a cuvintelor de cod poate fi atinsă efectiv, adică se poate obţine eficienţa $\eta = 1$, 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 şi limita din teorema 2.12 ar fi atinsă:

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

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 mulţime M, fie H entropia sursei şi fie S o mulţime de simboluri de cod. Atunci există un cod $c : M \to S^*$ unic decodabil a cărui lungime medie $\bar{l}$ a cuvintelor de cod satisface*

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

Rezultatul teoremei precedente poate fi îmbunătăţit 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 şi determinăm raportul dintre lungimea medie a cuvântului de cod şi numărul de mesaje din M codificate prin acesta. În detaliu, construcţia este următoarea:

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

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

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

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

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 obţinem 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 probabilităţi cunoscute ale mesajelor. Algoritmul cel mai utilizat pentru aceasta este algoritmul lui Huffman (algoritmul 2.4).

Ca idee de bază, algoritmul lui Huffman construieşte arborele unui cod prefix în modul următor: pleacă de la n arbori (n fiind numărul de mesaje) fiecare constând doar din rădăcină, după care uneşte 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 probabilităţilor mesajelor asociate cele mai mici; în caz de egalitate între probabilităţi, se iau oricare dintre arborii de probabilităţi egale. Algoritmul se termină în momentul în care rămâne un singur arbore.

Dacă $|S| > 2$ şi 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 şi în final să rămână exact un arbore.

Exemplul 2.11: Fie o sursă având mulţimea mesajelor posibile

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

cu probabilităţile corepsunzătoare $p_\mathrm{a} = 0{,}35$, $p_\mathrm{b} = 0{,}15$, $p_\mathrm{c} = 0{,}15$, $p_\mathrm{d} = 0{,}15$, $p_\mathrm{e} = 0{,}20$ şi 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 probabilităţi şi 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 mulţime finită de mesaje
$p_m \in (0,1)$, $m \in M$, probabilităţile mesajelor; $\sum_{m \in M} p_m = 1$ $S = \{s_1, s_2, \ldots, s_d\}$ mulţime finită de simboluri de cod, $d \geq 2$
ieşirea: $c : M \to S^*$ cod prefix
algoritmul:
E:=M
d':=$(|M| - 2) \bmod (|S| - 1) + 2$
cât timp $|E| > 1$ execută
alege $e_1, \ldots, e_{d'} \in E$ cu $p_{e_i} \leq p_{e^*}$, $\forall i \in \{1, \ldots, d'\}$, $\forall e^* \in E \setminus \{e_1, \ldots, e_{d'}\}$
crează t unic
pentru $i \in \{1, \ldots, d'\}$ execută
pune e_i ca fiu al lui t
$s_{(t,e_i)}$:=s_i
sfârşit pentru
p_t:=$\sum_{i=1}^{d'} p_{e_i}$
E:=$(E \setminus \{e_1, \ldots, e_{d'}\}) \cup \{t\}$
d':=d
sfârşit cât timp
c:=codul prefix asociat unicului arbore din E
sfârşit algoritm

Algoritmul 2.4: Algoritmul lui Huffman

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

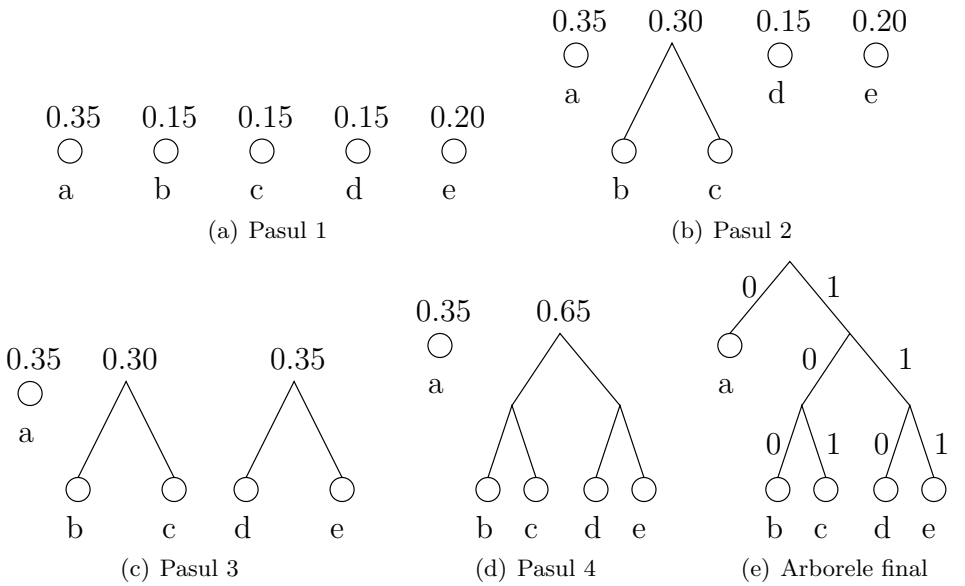
- În continuare unim din nou arborii de probabilităţi minime; acum aceştia sunt „d“ şi „e“ (fig. 2.5(c)).
- Avem acum două posibilităţi: arborele ce conţine pe „b“ şi pe „c“ poate fi unit fie cu arborele format din „a“, fie cu arborele format din „d“ şi „e“. Alegem a doua variantă.
- În final unim cei doi arbori rămaşi.

Avem acum codurile mesajelor: $c(\mathrm{a}) = 0$, $c(\mathrm{b}) = 100$, $c(\mathrm{c}) = 101$, $c(\mathrm{d}) = 110$, $c(\mathrm{e}) = 111$. 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 comparaţie, entropia este

$$\begin{aligned} H = &- 0{,}35 \log_2 0{,}35 + 0{,}15 \log_2 0{,}15 + 0{,}15 \log_2 0{,}15 + \\ &+ 0{,}15 \log_2 0{,}15 + 0{,}2 \log_2 0{,}2 \\ \approx &2{,}226121 \end{aligned}$$

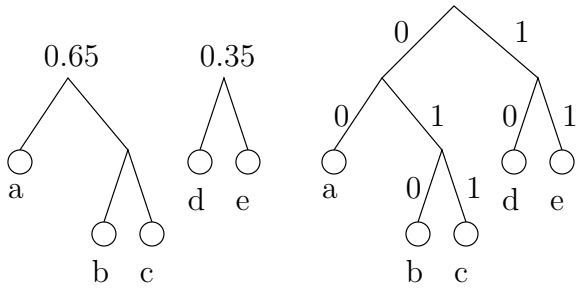


(a) Pasul 1 (b) Pasul 2 (c) Pasul 3 (d) Pasul 4 (e) Arborele final

Figura 2.5: Funcţionarea algoritmului Huffman, exemplul 2.11

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

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



(a) Pasul 4 (b) Arborele final

Figura 2.6: Variantă alternativă pentru paşii 4 şi 5 (exemplul 2.11)

EXEMPLUL 2.12: Fie o sursă având mulţimea mesajelor posibile

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

cu probabilităţile corepsunzătoare $p_{\mathrm{a}} = 0{,}4$, $p_{\mathrm{b}} = 0{,}15$, $p_{\mathrm{c}} = 0{,}15$, $p_{\mathrm{d}} = 0{,}1$, $p_{\mathrm{e}} = 0{,}1$, $p_{\mathrm{f}} = 0{,}1$ şi fie alfabetul canalului $S = \{0, 1, 2\}$.

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

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

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

Pentru demonstraţie avem nevoie de câteva leme ce descriu proprietăţi 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 mulţimea mesajelor sursei, fie p_m, $m \in M$, probabilităţile mesajelor sursei, fie S alfabetul canalului şi fie $c : M \to S^*$ un cod optim. Pentru orice mesaje $m_1, m_2 \in M$, dacă $p_{m_1} < p_{m_2}$ atunci $|c(m_1)| \geq |c(m_2)|$.*

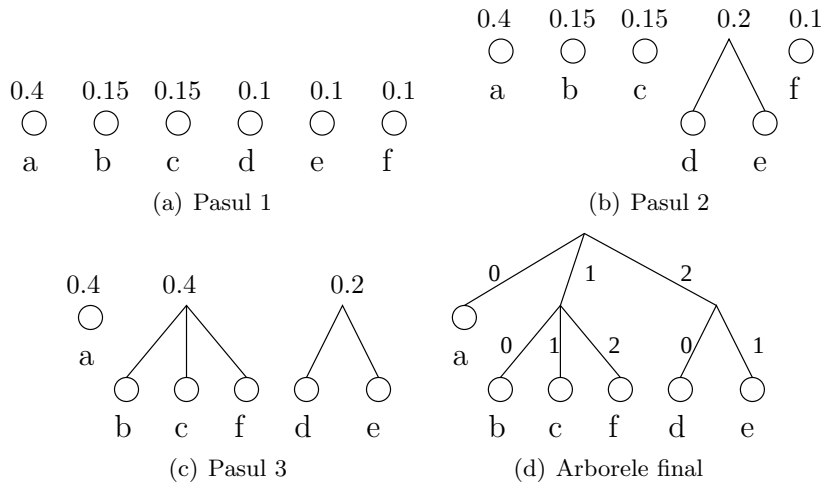


Figura 2.7: Funcţionarea algoritmului lui Huffman, exemplul 2.12

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

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

Demonstraţie. Conform corolarului 2.7, există un cod prefix cu aceleaşi lungimi ale cuvintelor de cod ca şi 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 poziţii libere* ale unui nod intern (un nod ce are cel puţin un fiu) valoarea $|S|$ minus numărul de fii. Observăm următoarele:

- Cu excepţia penultimului nivel, fiecare nod intern are zero poziţii libere Într-adevăr, în caz contrar s-ar putea muta o frunză de pe ultimul nivel ca descendent al nodului cu cel puţin o poziţe liberă; prin această operaţie ar scădea lungimea cuvântului de cod corespunzător şi ca urmare ar scădea lungimea medie a cuvintelor de cod, contrazicând ipoteza că c este optim.
- Suma numerelor poziţiilor libere ale nodurilor penultimului nivel este cel mult $|S| - 2$. Dacă arborele are înălţime 1, atunci unicul nod intern este rădăcina, aceasta are cel puţin 2 fii, deoarece $|M| \geq 2$, şi, în consecinţă, numărul poziţiilor libere este cel mult $|S| - 2$. Considerăm acum un arbore de înălţime cel puţin 2 şi să presupunând prin absurd că am avea $|S| - 1$ poziţii libere. Fie t un nod intern de pe penultimul nivel şi fie k numărul de descendenţi ai săi. Nodul t are $|S| - k$ poziţii libere, deci mai rămân cel puţin $k - 1$ poziţii libere la celelalte noduri. Mutăm $k - 1$ dintre descendenţii lui t pe poziţii 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 şi 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 şi cu numărul total de poziţii 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 inducţie după k. Dacă arborele are în total j poziţii libere, prin completarea acestora cu frunze ar rezulta un arbore cu 0 poziţii libere şi $n + j$ 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 contine cel puţin un nod intern, de unde rezultă că pe ultimul nivel există cel puţin $|S| - j$ frunze. Cum $|S| - j = q + 2$ rezultă că pe ultimul nivel avem cel puţin

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

frunze.♢

Demonstraţia teoremei 2.15. Fie n numărul de mesaje. Vom demonstra prin inducţie 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 şi să-i demonstrăm optimalitatea pentru $k+1$. Să luăm deci o mulţime de mesaje M cu $k(|S|-1)+1 \leq |M| \leq (k+1)(|S|-1)$, să notăm cu p_m, $m \in M$, probabilităţile mesajelor, să notăm cu c_h codul generat de algoritmul lui Huffman şi cu c_o un cod prefix optim pentru aceeaşi mulţime de mesaje şi aceleaşi probabilităţi şi 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 puţin $(n-2) \bmod (|S|-1)+2$ cuvinte de lungime maximă. Din lema 2.16, deducem că acestea sunt cuvintele corespunzătoare mesajelor cu probabilităţile cele mai mici, adică fie mesajele $e_1, \ldots, e_{d'}$ alese de algoritmul lui Huffman pentru prima unificare, fie mesaje de aceleaşi probabilităţi; în al doilea caz putem, prin interschimbări de cuvinte de cod, să facem ca cele $(n-2) \bmod (|S|-1)+2$ cuvinte de lungime maxmimă 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 $c'_o : (M \setminus \{e_1, \ldots, e_{d'}\}) \cup \{t\} \to S^*$, unde t este un obiect nou introdus, dând ca valoare pentru $c(t)$ prefixul comun al lui $c(e_1)$,...,$c(e_{d'})$. În acelaşi 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 $L(c'_o) = L(c_o) - p_t$ şi analog, $L(c'_h) = L(c_h) - p_t$. Să mai remarcăm că c'_h este codul produs de algoritmul lui Huffman pentru mulţimea de mesaje $(M \setminus \{e_1, \ldots, e_{d'}\}) \cup \{t\}$ şi, conform ipotezei de inducţie, el este optim; prin urmare $L(c'_h) \leq L(c'_o)$. De aici rezultă $L(c_h) \leq L(c_o)$, deci codul obtinut prin algoritmul lui Huffman este optim.♢

2.3.4. Compresia fişierelor

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

Algoritmii de compresie utilizaţi în practică se folosesc şi de dependenţele între octeţii succesivi.

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

De notat că nici un cod nu poate folosi mai puţini biţi pentru codare decât cantitatea de informaţie transmisă. În lipsa redundanţei, nu e posibilă compresia. Ca o consecinţă, nici un program de compresie nu poate comprima un şir aleator de octeţi.

2.4. Coduri detectoare şi corectoare de erori

Vom studia în cele ce urmează problema transmisiei informaţiei în situaţia unui canal discret, dar care alterează şirul de simboluri de cod transmise. În practică, o astfel de alterare este efectul zgomotelor ce se suprapun peste semnalul transmis de nivelul fizic (vezi capitolul 3); din acest motiv un astfel de canal se numeşte *canal cu zgomot* sau *canal cu perturbaţii*.

Pentru transmiterea corectă a datelor printr-un canal cu perturbaţii 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 destinaţia de acest lucru;
- *corectarea erorilor* presupune că receptorul determină cuvântul de cod cel mai probabil să fi fost transmis de către emiţător şi dă sursei mesajul corespunzător acelui cuvânt.

Ca principiu, atât detectarea cât şi corectarea erorilor se bazează pe un cod în care nu orice secvenţă (de lungime adecvată) de simboluri de cod este cuvânt de cod şi, ca urmare, alterările cele mai probabile ale şirului de simboluri transmis conduc la secvenţe 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 şi, 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 mulţimea cuvintelor de cod să fie o submulţime „rară“ a mulţimii secvenţelor de simboluri de cod.

Prin urmare, posibilităţile de detectare a erorilor ţin de construcţia codului. De aici denumirea de *cod detector de erori*, respectiv *cod corector de erori*. Deoarece la orice cod detector sau corector de erori mulţimea şirurilor de cuvinte de cod este o submulţime strictă a mulţimii şirurilor arbitrare de simboluri de cod, rezultă că orice cod detector sau corector de erori are redundanţă.

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

2.4.1. Modelul erorilor

Construcţia codului detector sau corector de erori trebuie făcută în aşa fel încât să facă suficient de mică probabilitatea unei erori nedetectate. Este deci esenţială construcţia unui model probabilistic al erorilor, adică determinarea, pentru fiecare modificare a şirului de simboluri transmis de canal, a probabilităţii 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 şir de bitî (nu neapărat toţi). Lungimea rafalei este numărul de biţi dintre primul şi ultimul bit modificat;
- *erori de sincronizare*, care determină pierderea unui bit sau introducerea unui bit, împreună cu decalarea corespunzătoare a biţilor următori.

Transmisia unui şir de biţi 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 şi că probabilitatea ca o eroare să afecteze un bit este aceeaşi pentru toţi biţii şi independentă de valorile biţilor şi de poziţiile celorlalte erori. Cu alte cuvinte, fiecare bit are o probabilitate p să fie inversat (dacă emiţătorul a transmis un 1 receptorul să primească 0 şi dacă emiţătorul a transmis 1 receptorul să primească 0) şi $1 - p$ să fie transmis corect.

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

să se producă exact k erori este

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

conform distribuţiei binomiale.

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

2.4.2. Principiile codurilor detectoare şi corectoare de erori

Vom analiza doar cazul codurilor de lungime fixă pentru mulţimea de simboluri $S = \{0, 1\}$. Notăm cu l lungimea cuvintelor de cod. Prin urmare, mulţimea cuvintelor de cod, W, este o submulţime a mulţimii şirurilor 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 şi deoarece toate cuvintele de cod au aceeaşi 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 creşterea lui k, se alege o valoare k astfel încât probabilitatea de-a avea k sau mai multe erori este neglijabil de mică şi se construieşte codul presupunând că nu se produc mai mult de $k - 1$ erori.

Definiţia 2.18 *Spunem despre codul $c : M \to \{0, 1\}^l$ 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 puţine 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 funcţie distanţă:

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

unde $u = (u_1, u_2, \ldots, u_l)$ şi $v = (v_1, v_2, \ldots, v_l)$. Astfel, distanţa 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 $c : M \to \{0,1\}^l$ 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 $c : M \to \{0,1\}^l$ 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:*

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

Propoziţia 2.21 *Fie codul $c : M \to \{0,1\}^l$ 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

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

Să mai notăm că $|M| = |W| = 2^n$.

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

- construcţia unei multimi $W \subseteq \{0,1\}^l$ 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 notaţiile din paragraful precedent.

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

Pentru un astfel de cod, emiţătorul primeşte de la sursă n biţi ce constituie informaţia utilă, calculează cei $l-n$ biti de control aplicând un algoritm asupra informaţiei utile şi transmite prin canal informaţia utilă urmată de biţii de control. Receptorul citeşte informaţia utilă şi biţii de control; pentru detectarea erorilor aplică acelaşi algoritm ca şi emiţătorul asupra informaţiei utile citite şi verifică dacă rezultatul coincide cu biţii de control citiţi.

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 mulţimea şirurilor de l biţi conţinând număr par de valori 1, şi paritate impară (engl. *odd parity*), în care W este mulţimea şirurilor de l biţi conţinând un număr impar de valori 1. Unicul bit de control se mai numeşte *bit de paritate*.

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

Bitul de paritate se calculează numărând biţii cu valoare 1 din informaţia utilă şi verificând dacă este par sau impar.

Exemplul 2.13: Pentru codul cu paritate pară şi $n = 7$, şirul de biţi 1010110 (informaţie utilă) se codifică 10101100 (bitul de control este 0). Şirul 1110110 se codifică 11101101 (bit de control 1). Şirul 11001100 este cuvântul de cod corespunzător informaţiei utile 1100110. Şirul 11001101 nu este cuvânt de cod valid.

Exemplul 2.14: Pentru codul cu paritate impară şi $n = 7$, şirul de biţi 1010110 se codifică 10101101 (bitul de control este 1). Şirul 1110110 se codifică 11101100 (bit de control 1). Şirul 11001100 nu este cuvânt de cod valid. Şirul 11001101 este cuvântul de cod corespunzător informaţiei utile 1100110.

2.4.3.2. Paritate pe linii şi coloane

La un astfel de cod informaţia utilă se consideră a fi o matrice $n_1 \times n_2$ de biţi, cu n_1 şi n_2 fixaţi. 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 şi ultima coloană cuprind biţii de control. Mulţimea cuvintelor de cod este mulţimea matricilor $(n_1 + 1) \times (n_2 + 1)$ în care pe fiecare linie şi pe fiecare coloană numărul de valori 1 este par.

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

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

Exemplul 2.15: Pentru $n_1 = n_2 = 4$, şirul 1011010111001111 se codifică astfel:

1	0	1	1	1
0	1	0	1	0
1	1	0	0	0
1	1	1	1	0
1	1	0	1	1

Astfel, cuvântul de cod rezultat este şirul: 101110101011000111101 1011.

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

Exemplul 2.16: Şirul 1010011010110100110001111111101 nu este cuvânt de cod:

1	0	1	0	0
1	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	1	1	0	1

Se observă că paritatea nu este respectată de linia a 2-a şi 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 şir de biţi $v = (v_1, \ldots, v_k) \in \{0,1\}^k$ i se asociază un polinom de grad cel mult $k - 1$:

$$v(X) = v_1 X^{k-1} + v_2 X^{k-2} + \ldots + v_{k-1} X + v_k.$$

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

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

De remarcat că polinoamele peste orice corp păstrează multe din proprietăţile polinoamelor „obişnuite“, în particular se poate defini la fel adunarea, scăderea şi înmulţirea şi are loc teorema împărţirii cu rest.

Pentru construcţia unui cod polinomial, se alege un aşa-numit *polinom generator* $g(X)$ de grad $l-n$ (reamintim că l este lungimea cuvintelor de cod, iar n este numărul de biţi ai informatiei utile; $n < l$). Mulţimea cuvintelor de cod W se defineşte ca mulţimea şirurilor de l biţi cu proprietatea că polinomul asociat şirului este divizibil cu $g(X)$.

Şirul biţilor de control se calculează astfel:

- se construieşte polinomul $i(X)$ asociat informaţiei utile,
- se calculează $r(X)$ ca fiind restul împărţirii lui $i(X) \cdot X^{l-n}$ la $g(X)$
- şirul biţilor de control este şirul de $l-n$ biţi al cărui polinom asociat este $r(X)$.

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

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

Deoarece $r(X)$ este restul împărţirii 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 simplităţii construcţiei unor circuite (hardware) care calculează biţii de control.

Dacă se doreşte corectarea erorilor, se observă că poziţiile erorilor nu depind decât de restul împărţirii polinomului asociat şirului de biţi recepţionat, $v'(X)$, la $g(X)$.

2.4.4. Coduri detectoare şi corectoare de erori în alte domenii

Ne întâlnim cu coduri detectoare sau corectoare de erori şi în situaţii mai puţin legate de calculatoare.

Limbajul natural conţine multă redundanţă; ca urmare permite detetcarea 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 radial 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 *şepte*. 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 informaţiei la nivelul fizic

Sarcina nivelului fizic este aceea de-a transmite un şir de biţi (sau, în general, un şir de simboluri) produs de o *sursă* către o *destinaţie*. Sursa şi destinaţia se află la distanţă una faţă de cealaltă.

Sursa şi destinaţia sunt „clienţii“ sistemului de comunicaţie; nivelul fizic trebuie să fie capabil să transmită datele în folosul acestora.

Şirul de biţi 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 destinaţiei şi este capabil să transmită la distanţă o anumită acţiune fizică.

Nivelul fizic cuprinde trei elemente: mediul de transmisie, *emiţătorul* şi *receptorul* (vezi fig. 3.1). *Emiţătorul* primeşte biţii de la sursă şi, în conformitate cu valorile lor, acţionează asupra mediului. *Receptorul* sesizează acţiunile emiţătorului asupra mediului şi reconstituie şirul de biţi produs de sursă. Şirul de biţi reconstituit este livrat destinaţiei.

Mărimea fizică ce măsoară acţiunea produsă de emiţător şi transmisă de către mediu până la receptor şi care este utilizată efectiv ca purtătoare a informaţiei se numeşte *semnal*. Semnalul este întotdeauna analizat ca o funcţie continuă de timp.

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

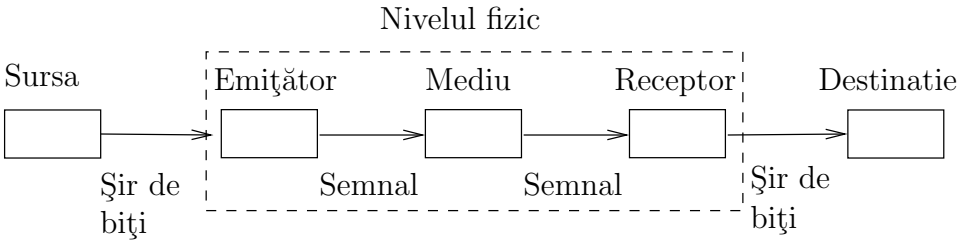


Figura 3.1: Modelarea transmisiei la nivel fizic

Emiţătorul transformă şirul de biţi recepţionat într-un semnal adecvat transmiterii prin mediul de comunicaţie. Receptorul efectuează operaţia inversă. Corespondenţa dintre şirurile de biţi posibile şi semnalele corespunzătoare poartă denumirea de *schemă de codificare a informaţiei prin semnal continuu*.

Schema de codificare utilizată trebuie să fie aceeaşi pentru emiţător şi 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 emiţător asupra mediului. Vom arăta în § 3.2 care sunt transformările suferite de semnal în timpul propagării. Schema de codificare a informaţiei trebuie să ţină 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 şi de codificarea informaţiei prin semnale. O analiză riguroasă a acestor probleme depăşeşte cu mult cadrul acestei lucrări. Prezentarea de faţă are ca scop familiarizarea cu noţiunile şi problemele respective, în vederea înţelegerii soluţiilor existente, limitărilor lor, parametrilor specificaţi în documentaţiile privind echipamentele folosite şi, mai ales, posibilităţii comunicării cu specialiştii în domeniul electronicii şi comunicaţiilor.

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 joncţiunea dintre emiţător şi mediu se numeşte *semnal emis* şi îl vom nota cu $U_e(t)$, unde t este timpul. Semnalul măsurat

la joncţiunea dintre mediu şi receptor se numeşte *semnal recepţionat* şi îl vom nota cu $U_r(t)$.

Transformările suferite de semnal sunt următoarele:

Ωîntârziereaconstă în faptul că semnalul recepţionat urmează cu o anumită întârziere semnalul emis. Cu notaţiile de mai sus şi neglijând fenomenele ce vor fi descrise la punctele următoare, avem $U_r(t) = U_e(t - \Delta t)$. Durata Δt se numeşte *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 şi, 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 recepţionat are amplitudine mai mică decât cel emis. Neglijând întârzierea, are loc $U_r(t) = g \cdot U_e(t)$, cu $0 < g < 1$. Ţinând cont şi de întârziere, avem $U_r(t) = g \cdot U_e(t - \Delta t)$. Numărul $1/g$ se numeşte *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 şi a celui recepţionat. În cazul perechii de conductoare, deoarece puterea este proporţională 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 foloseşte adesea logaritmul său: logaritmul factorului de atenuare rezultat este suma logaritmilor, în aceeaşi bază, ai factorilor de atenuare ai componentelor.

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

Valoarea logaritmului depinde de baza utilizată, baze diferite ducând la valori proporţionale. Deoarece schimbarea bazei de logaritmare are un efect similar cu schimbarea unităţii 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 *bel*ul, având simbolul B. Pseudo-unitatea de măsură utilizată curent este *decibel*ul, având simbolul dB. Avem 1 B=10 dB. O valoare exprimată în decibeli (dB) o putem vedea, echivalent, fie ca valoarea logaritmului în baza

10 înmulţită 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 B = 10 dB. Dacă factorul de atenuare este 2, logaritmul său (atenuarea) este $\log_{10} 2$ B $\approx 0{,}3$ B = 3 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, şi logaritmul său se exprimă în decibeli. Pseudo-unitatea de măsură corespunzătoare reprezentării de mai sus se numeşte decibel-miliwatt, având simbolul (neconform regulilor Sistemului Internaţional de Masuri şi Unităţi) dBm. Ca exemple: o putere de emisie de 1 mW poate fi scrisă şi 0 dBm, o putere de 1 W se scrie 30 dBm, iar 0,1 mW se scrie ca −10 dBm.

Puterea minimă a semnalului recepţionat, pentru care receptorul este capabil să decodifice corect semnalul, se numeşte *pragul de sensibilitate* al receptorului. Ca şi puterea emiţătorului, pragul de sensibilitate se poate exprima în miliwatti sau în decibel-miliwatti.

distorsiunea este o modificare deterministă a semnalului recepţionat faţă de cel emis, diferită de întârziere şi atenuare. (O modificare este deterministă dacă, oridecâteori transmitem un acelaşi semnal, modificarea se manifestă identic.) Mai multe detalii despre distorsiuni vor fi date în § 3.2.2.

zgomotele sunt modificări nedeterministe ale semnalului recepţionat, 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 (mişcarea de agitaţie termică a atomilor din dispozitivele electornice).

Zgomotul se exprimă ca diferenţa dintre semnalul recepţionat efectiv şi semnalul ce ar fi recepţionat în lipsa zgomotului. *Raportul semnal/zgomot* este raportul dintre puterea semnalului şi puterea corespunzătoare zgomotului. Uneori termenul de raport semnal/zgomot este utilizat şi 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 şi o ieşire

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

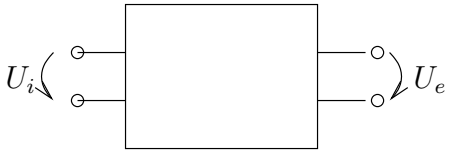


Figura 3.2: Un dispozitiv cu o intrare şi o ieşire

Tensiunea de la ieşire 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 funcţie definită pe un spaţiu de funcţii cu valori tot într-un spaţiu de funcţii). Acest operator primeşte ca argument funcţia timp-tensiune U_i care caracterizează semnalul de intrare. Valoarea operatorului este funcţia timp-tensiune $U_e = L(U_i)$ care caracterizează semnalul de ieşire.

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 funcţii f şi g şi pentru orice scalari α şi 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

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

atunci pentru semnalul de ieşire avem

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

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

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

$$U_i(t) = U_0 \cdot \cos\left(2\pi f t + \phi\right),$$

atunci semnalul de ieşire este tot sinusoidal şi, mai mult,

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

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

Orice semnal se poate scrie unic ca o sumă de semnale sinusoidale. *(Nota: condiţiile matematice asupra semnalului, şi 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 aşa-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:

$$U(t) = \int_0^{\infty} a(f) \cdot \cos\left(2\pi f t + \phi(f)\right) \mathrm{d}f. \tag{3.1}$$

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

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

unde $\hat{U}$ este o funcţie complexă care se numeşte *transformata Fourier* a funcţiei U.

Relaţia (3.1) spune că semnalul U se poate scrie ca o sumă de sinusoide cu diferite frecvenţe f având amplitudinile $a(f)$ şi defazajul (decalajul sinusoidei de-a lungul axei Ox) egal cu $\phi(f)$.

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

Pentru un dispozitiv liniar, semnalul de ieşire se poate calcula descompunând în sinusoide semnalul de intrare, calculând efectul dispozitivului asupra fiecărei sinusoide în parte şi însumând în final ieşirile:

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

unde $a_i(f)$ şi $\phi_i(f)$ sunt funcţiile $a(f)$ şi $\phi(f)$ din descompunerea, conform relaţiei (3.1), a semnalului de intrare U_i.

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

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

$$\left\{\begin{array}{lcl} g(f) &=& g_0 \\ \theta(f) &=& \theta_0 f \end{array}\right. \tag{3.4}$$

pentru toate frecvenţele f din spectrul semnalului.

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

Diferenţa $f_{\max} - f_{\min}$ se numeşte *lăţimea de bandă* a dispozitivului.

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

3.3. Codificarea informaţiei prin semnale continue

3.3.1. Scheme de codificare

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

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

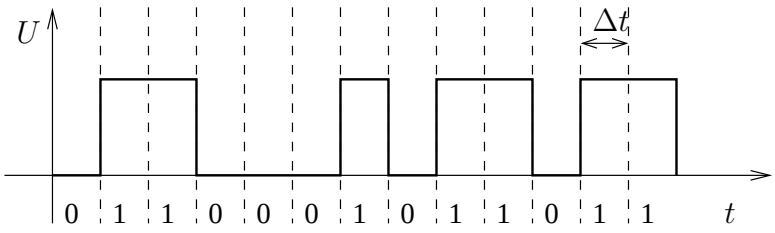


Figura 3.3: Codificarea directă

Receptorul determină intervalele corespunzătoare biţilor şi 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 funcţie de amplitudinea semnalului recepţionat pentru a ţine cont de atenuare.

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

Sincronizarea se poate face şi 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) şi va determina de câte ori se cuprinde în acest interval durata unui bit. Numărul de biţi consecutivi identici trebuie să fie limitat, căci receptorul nu va putea distinge între n biţi şi $n + 1$ biţi consecutivi având aceeaşi valoare, dacă n este prea mare.

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

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

Reţelele Ethernet de 10 Mbit/s utilizează codificarea Manchester.

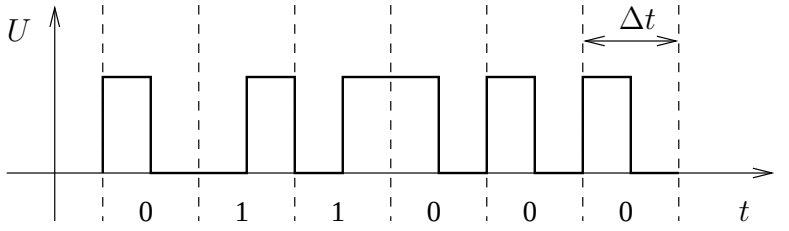


Figura 3.4: Codificarea Manchester

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

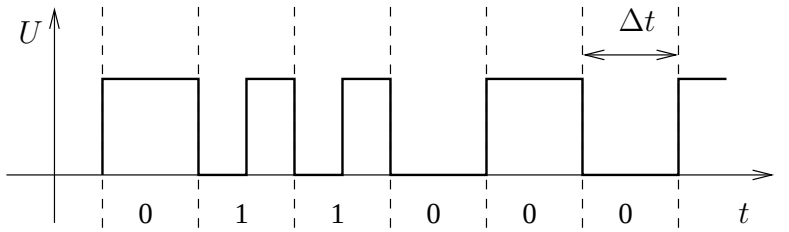


Figura 3.5: Codificarea Manchester diferenţială

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

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

Se fixează o tabelă de corespondenţă care asociază fiecăruia dintre cele 2^n şiruri de n biţi posibile un şir de m biţi, unde $m > n$ este fixat, cu restricţia ca între cei m biţi să nu fie prea multe valori egale consecutive. Codul este determinat de numrele n şi m şi de această tabelă.

Şirul de biţi de codificat se codifică astfel: mai întâi, fiecare grup de n biţi consecutivi se înlocuieşte cu şirul de m biţi asociat. Apoi şirul de biţi astfel obţinut se codifică direct, un bit 0 fiind reprezentat printr-o valoare a tensiunii şi un bit 1 prin altă valoare.

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

Să examinăm acum cerinţele 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 comunicaţie care are o lăţime de bandă finită, semnalele vor fi „rotunjite“ mai mult sau mai puţin.

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 şi Manchester diferenţială, $\tau = \frac{1}{2}\Delta t$.

Dacă banda de trecere a mediului include intervalul $\left[0, \frac{1}{2\tau}\right]$, atunci mediul păstrează suficient din forma semnalului pentru ca receptorul să poată decodifica informaţia transmisă. Dacă frecvenţa 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 şi următorul timp τ cealaltă valoare va fi distorsionat atât de mult încât „urcuşurile“ şi „coborâşurile“ semnalului nu vor mai putea fi identificate de către receptor şi ca urmare informaţia purtată nu mai poate fi obţinută.

Pentru un mediu dat, cu o bandă de trecere dată, există, prin urmare, o valoare minimă a lui τ pentru care receptorul poate extrage informaţia utilă din semnalul recepţionat. 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 şi valoarea lui τ. La codificarea directă, durata unui bit este egală cu τ şi deci $k = 1$. La codificările Manchester şi Manchester diferenţială, durata unui bit este 2τ şi avem $k = 2$. La codurile de grup, durata unui bit util este $\frac{m}{n}\tau$ şi avem $k = \frac{m}{n}$. Debitul maxim cu care se pot transmite datele este $\frac{f_{\max}}{k}$.

3.3.2. Modulaţia

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

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

Semnalul transmis efectiv este de forma:

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

unde unul dintre parametri a, f sau ϕ variază în timp, în funcţie 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 şi ϕ, se numeşte *semnal purtător*, iar frecvenţa f de repaus se numeşte *frecvenţă purtătoare* şi o vom nota în continuare cu f_p.

Semnalul rezultat în urma modulaţiei se numeşte *semnal modulat*.

Operaţia de construcţie a semnalului modulat pornind de la semnalul primar se numeşte *modulaţie*. Operaţia inversă, de obţinere a semnalului primar dându-se semnalul modulat, se numeşte *demodulaţie*.

După parametrul modificat, avem:

modulaţia 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 restricţia $s(t) > 0$.

Se observă că modulaţia în amplitudine este liniară (modulaţia sumei a două semnale $a + b$ este suma rezultatelor modulaţiei independente pentru a şi b).

Dacă semnalul modulator este sinusoidal

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

atunci

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

adică în urma modulaţiei în amplitudine cu un semnal sinusoidal de frecvenţă f_s se obţine o sumă de trei semnale sinusoidale având frecvenţele $f_p - f_s$, f_p şi $f_p + f_s$.

Din liniaritatea modulaţiei în amplitudine şi din relaţia (3.5) deducem că, pentru un semnal modulator având un anumit spectru, spectrul semnalului modulat conţine frecvenţa purtătoare şi două *benzi laterale*, stângă şi dreaptă, acestea cuprinzând diferenţele, respectiv sumele, dintre frecvenţa purtătoare şi frecvenţele din spectrul semnalului primar.

Întrucât spectrul semnalului modulat este simetric în jurul frecvenţei purtătoare, de fapt doar una dintre benzile laterale poartă informaţie utilă. Din acest motiv, adesea se suprimă total sau parţial de la transimisie una dintre benzile laterale.

modulaţia de frecvenţă (prescurtat *MF*, engl. *frequency modulation*, *FM*), care constă în modificarea frecvenţei f în jurul frecvenţei purtătoare f_p.

Semnalul transmis are forma

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

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

Analiza spectrului unui semnal modulat în frecvenţă este mult mai difcilă decât în cazul modulaţiei în amplitudine.

modulaţia de fază, care constă în modificarea fazei ϕ.

Semnalul transmis are forma

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

Este evident că, întrucât receptorul nu are de obicei un reper absolut de timp, el nu poate detecta decât variaţiile de fază ale semnalului recepţionat. Ca urmare, o valoare constantă a lui $s(t)$ nu poate fi deosebită de zero şi, mai mult, nici variaţii lente ale lui $s(t)$ nu pot fi detectate. În consecinţă, spectrul lui $s(t)$ nu poate conţine frecvenţe prea apropiate de 0.

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

Semnalul modulat are forma

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

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.

Emiţătoarele 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 coresunzător.

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

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

Deoarece pentru o schemă de codificare fixată există o relaţie de proporţionalitate între lăţimea de bandă a mediului şi debitul maxim al transmisiei, debitul maxim al transmisiei unui echipament de comunicaţie se numeşte uneori în mod impropriu tot *lăţime 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. Construcţia cablului

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

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

Ca formă şi dispunere relativă, există trei construcţii utilizate:

- *Perechea simplă*, în care conductoarele sunt paralele unul faţă de celălalt. Conductoarele pot fi alcătuite dintr-o singură sârmă de cupru, sau — pentru a fi mai flexibile — dintr-un mănunchi de sârme subţiri. 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 şi sensibilităţii la zgomote, perechea simplă se utilizează doar pe distanţe mici.

- *Perechea torsadată* (engl. *twisted pair*), în care conductoarele sunt răsucite unul în jurul celuilalt. Rolul răsucirii este de-a micşora interacţiunea cu câmpul electromagnetic înconjurător, adică micşorerea zgomotului indus de un câmp electromagnetic înconjurător şi, totodată, micşorarea câmpului electromagnetic produs de semnalul ce trece prin perechea de conductoare. Acest lucru este important în special pentru micşorarea diafoniei cu celelalte perechi de conductoare din acelaşi cablu. În afară de răsucire, restul construcţiei 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 conţine mai multe perechi torsadate, fiecare circuit de comunicaţie să utilizeze conductoarele din aceeaşi pereche şi nu un conductor dintr-o pereche şi un conductor din altă pereche. În caz contrar, apare diafonie foarte puternică între circuite (mai mare decât la perechea simplă). De remarcat că această greşeală este uşor de comis în urma unei identificări greşite a conductoarelor dintr-un cablu şi nu este pusă în evidenţă 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 şi izolat electric faţă de acesta. Conductorul exterior este format de obicei dintr-o plasă formată din sârme subţiri de cupru, înfăşurate elicoidal, o parte din fire fiind înfăşurate într-un sens şi altă parte în celălalt sens.

  Cablul coaxial este şi mai protejat de interferenţe 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 interferenţele 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 înfăşurarea unei foiţe metalice, marginile foiţei trebuie să facă contact ferm între ele) şi 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 conţine elemente cu rol de protecţie. Orice cablu este înfăşurat cel puţin într-o manta de protecţie, care ţine la un loc şi protejează mecanic conductoarele. Mantaua de protecţie este fabricată de obicei din PVC.

Un cablu destinat montării aerian trebuie să fie prevăzut cu un cablu de oţel pentru susţinere mecanică. Un cablu destinat montării subteran trebuie prevăzut cu un scut metalic contra rozătoarelor.

3.4.2. Proprietăţi ale mediului

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

Proprietăţile 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 frecvenţă maximă de ordinul a câteva sute de megahertzi sau câţiva gigahertzi. Limitările sunt date de două fenomene independente, pierderile în dielectric (la frecvenţe mari dielectricul absoare o parte din energia câmpului electric dintre conductoare) şi efectul pelicular (la frecvenţe mari curentul electric din conductoare nu circulă uniform în toată masa acestora ci doar în vecinătatea suprafeţei). Îmbătrânirea izolaţiei cablului duce la micşorarea frecvenţei maxime a benzii de trecere.

Atenuarea semnalului. Factorul de atenuare creşte exponenţial cu lungimea mediului. În consecinţă, logaritmul factorului de atenuare creşte liniar cu lungimea mediului. Ca urmare, pentru un tip de cablu se specifică raportul dintre logaritmul factorului de atenuare şi 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.

Impedanţa caracteristică a mediului. Să presupunem că ataşăm la un capăt al unei bucăţi infinite de cablu o sursă de tensiune alternativă. Se

observă că intensitatea curentului ce trece prin sursă şi prin capătul dinspre sursă al cablului este proporţională cu tensiunea. Raportul dintre tensiune şi intensitate se numeşte *impedanţa caracteristică* a cablului.

Receptorul se caracterizează şi el printr-o *impedanţă de intrare*, definită ca raportul dintre tensiunea aplicată la bornele receptorului şi intensitatea curentului absorbit de receptor. Emiţătorul se caracterizează printr-o *impedanţă de ieşire*, definită ca raportul dintre scăderea tensiunii la borne cauzată de absorbţia unui curent de către un dispozitiv montat la bornele emiţătorului şi intensitatea curentului absorbit.

Dacă la un capăt de cablu de o anumită impedanţă legăm un cablu de altă impedanţă sau dacă emiţătorul sau receptorul ataşat are altă impedanţă decât impedanţa caracteristică a cablului, spunem că avem o *neadaptare de impedanţă*. În acest caz, joncţiunea 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 refracţie 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 şi, fiind întârziat faţă de acesta, îl distorsionează.

Impedanţa se măsoară în *ohmi* (simbol Ω). Cablul pentru televiziune are impedanţa de 75 Ω. Cablul coaxial pentru reţea Ethernet are impedanţa de 50 Ω. Cablul torsadat Ethernet are 100 Ω.

3.4.3. Legătură magistrală

La o pereche de conductoare pot fi conectate mai multe emiţătoare sau receptoare. O astfel de interconectare poate avea două scopuri: pentru a realiza simplu o comunicaţie de tip difuziune (un emiţător 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 numeşte *magistrală*.

Realizarea mediului fizic, în acest caz, este complicată de necesitatea de a avea adaptare de impedanţă 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 ramificaţie, o neadaptare de impedanţă.

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

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

O altă soluţie de conectare a mai multor dispozitive (emiţătoare sau receptoare) la un cablu constă în realizarea unei ramificaţii foarte scurte, astfel încât să nu aibă comportament de linie lungă (la frecvenţele cu care se lucrează uzual, aceasta înseamnă cel mult câţiva centimetri), la capătul căreia se conectează emiţătorul sau receptorul. Emiţătorul sau receptorul astfel conectat trebuie să aibă impedanţa de ieşire, respectiv de intrare, mult mai mare decât impedanţa perechii de conductoare la care se conectează. O astfel de conectare se utilizează, de exemplu, în reţelele Ethernet vechi (vezi § 9.1.3 şi 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 joncţiune de la perechea ce are o anumită impedanţă la un dispozitiv având impedanţa infinită. Pentru evitarea reflexiilor, la capătul unei perechi de conductoare trebuie montat un dispozitiv numit *terminator*. Terminatorul este un simplu rezistor, având rezistenţa egală cu impedanţa 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, emiţătorul şi receptorul au, în mod obişnuit, impedanţa necesară, astel încât joacă şi rol de terminator.

3.4.4. Considerente practice

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

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

Astfel, între carcasele, legate la pământarea reţelei electrice, a două calculatoare sau alte echipamente, poate apare o tensiune electrică de ordinul câtorva volţi; dacă echipamentele sunt conectate la reţelele electrice a două clădiri diferite, tensiunea dintre carcase de propagarele lor poate fi chiar mai mare. Pentru ca aceasta să nu perturbe semnalul util, în construcţia plăcilor de reţea trebuie luate măsuri speciale de izolare. Dacă unul dintre conduc-

toare este expus atingerii cu mâna (este cazul la reţelele Ethernet cu cablu coaxial, unde conductorul exterior este legat la partea metalică exterioară a conectoarelor), standardele de protecţie 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 reţelei de pământare.

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

3.5. Transmisia prin unde radio

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

Frecvenţa unei unde electromagnetice este frecvenţa de oscilaţie a câmpului electromagnetic într-un punct fixat din spaţiu.

Lungimea de undă a unei unde este distanţa parcursă de undă în timpul unei oscilaţii complete. Lungimea de undă se notează cu λ şi are valoarea $\lambda = \frac{v}{f}$, unde f este frecvenţa şi 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 frecvenţei 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 frecvenţe la care pot să lucreze dispozitivele electronice; în funcţie de autori, limita de jos a frecvenţelor undelor radio este cuprinsă între 30 Hz ($\lambda = 10000$ km) şi 3 kHz ($\lambda = 100$ km), iar limita de sus a frecvenţelor este cuprinsă între 1 GHz ($\lambda = 30$ cm) şi 300 GHz ($\lambda = 1$ mm), cu observaţia că undele electromagnetice din intervalul 1 GHz – 300 GHz se numesc *microunde* şi unii autori consideră că microundele nu fac parte dintre undele radio ci sunt o categorie separată de acestea.

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

La transmisia prin unde radio, mărimile fizice utilizate ca semnal sunt intensitatea câmpului electric şi inducţia magnetică. Cele două mărimi sunt proporţionale în modul şi au direcţii perpendiculare una pe cealaltă şi pe direcţia de propagare a undei.

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

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

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

Mai multe proprietăţi ale sistemului de transmisie fac ca lăţimea benzii de trecere a întregului sistem să fie îngustă în raport cu frecvenţele între care se încadrează banda de trecere; raportul între lăţimea benzii şi limita inferioară a benzii este în mod tipic de cel mult câteva procente. Din această cauză, transmisia prin unde radio este întotdeauna cu modulaţie, iar frecvenţa purtătoare este cel puţin de câteva zeci de ori mai mare decât lăţimea de bandă.

De exemplu, pentru o viteză de transmisie de 10 Mbit/s avem în mod tipic nevoie de o lăţime de bandă apropiată de 10 MHz, pentru care frecvenţa purtătoare va fi de cel puţin 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 spaţiu: *intensitatea câmpului electric*, notată cu $\vec{E}$, şi *inducţia magnetică*, notată cu $\vec{B}$.

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

Dacă $\vec{E}$ are direcţie constantă şi îi variază doar sensul şi modulul,

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

Dacă $\vec{E}$ are modul constant şi direcţia lui se roteşte uniform, în plan perpendicular pe direcţia de deplasare a undei, fasciculul se numeşte *polarizat circular*. Se distinge *polarizare circulară stângă* dacă, privind în direcţia de propagare a undelor, dinspre emiţător spre receptor, direcţia lui $\vec{E}$ se roteşte în sens invers acelor de ceas; şi *polarizare circulară dreaptă* dacă $\vec{E}$ se roteşte 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 aceeaşi direcţie şi cu un decalaj de un sfert de ciclu între ele. Dacă cele două fascicole au amplitudini diferite, rezultă ceea ce se numeşte *polarizare eliptică*; polarizarea liniară şi polarizarea circulară sunt de fapt cazuri particulare de polarizare eliptică.

3.5.1.2. Absorbţia şi reflexia

Absorbţia undelor radio în aer este neglijabilă.

Picăturile de apă (din ploaie, nori, ceaţă) absorb destul de puternic undele radio, în special microundele. Apa absoarbe puternic toate undele radio; de aceea este greu de obţinut legătură radio sub apă. Absorbţie moderată se produce în pământ şi în diferite materiale de construcţie.

Scăderea puterii undelor radio datorită absorbţiei este exponenţială cu distanţa, ca şi în cazul propagării semnalelor prin cabluri.

Metalele reflectă undele radio. Plasele metalice care au contact bun între firele componente şi au ochiurile mult mai mici decât lungimea de undă se comportă ca o suprafaţă 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 si ionosferă, aceste unde pot parcurge uşor multe mii de kilometri.

3.5.1.3. Difracţia

Orice undă ocoleşte obstacolele mai mici decât o fracţiune din lungimea de undă, în vreme ce în spatele obstacolelor mai mari de câtea 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 distanţă de câteva sute sau chiar mii de kilometri. Prin contrast, undele cu lungime de undă sub câţiva metri se propagă aproape numai în linie dreaptă, dealurile sau clădirile mai mari putând provoca umbre.

3.5.1.4. Interferenţa undelor

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

Dacă diferenţa 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 şi se adună, semnalul recepţionat fiind mai puternic. Dacă diferenţa de drum este apropiată de un număr impar de lungimi de undă, undele se suprapun în antifază şi se anulează reciproc, semnalul recepţionat fiind slab sau nul. În aceste situaţii, deplasarea receptorului (sau emiţătorului) pe o distanţă de la un sfert din lungimea de undă şi până la de câteva ori lungimea de undă poate modifica mult calitatea semnalului (reaminitim că în transmisiile de date se utilizează lungimi de undă cuprinse între 1 cm şi 1 m). Schimbarea lungimii de undă pe care se face transmisia poate de asemenea modifica mult efectul.

Dacă diferenţa de drum între semnalele recepţionate pe căi diferite este comparabilă sau mai mare decât lungimea unui bit şi puterile semnalului pe cele două căi sunt apropiate, semnalele propagate pe cele două căi se bruiază reciproc. Situaţia apare mult mai rar decât cea prezentată mai sus, însă nu poate fi corectată decât prin mutarea staţiilor faţă de obstacolele ce produc reflexiile.

3.5.1.5. Divergenţa undelor

Pe măsură ce ne depărtăm de emiţător, puterea semnalului scade datorită extinderii frontului de undă. Densitatea puterii este invers proporţională cu suprafaţa frontului de undă, care la rândul ei este proporţională cu pătratul distanţei faţă de emiţător.

Ca urmare, puterea recepţionată P_r este invers proporţională cu pătratul distanţei d dintre emiţător şi receptor:

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

unde α este o constantă ce depinde de construcţia antenelor de emisie şi de recepţie, iar P_e este puterea emiţătorului.

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

3.5.2. Antene

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

oscilaţiile electromagnetice în mediul înconjurător antenei. Orice antenă este reversibilă: dacă i se aplică un semnal electric la borne, va radia unde electromagnetice şi, 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 undei electromagnetice radiate şi puterea absorbită din semnalul electric primit.

3.5.2.1. Directivitatea

O antenă nu radiază uniform de jur împrejur. Prin *câştigul* (engl. *gain*) unei antene *pe o direcţie* se înţelege raportul dintre puterea radiată pe acea direcţie şi puterea radiată de o antenă etalon, în aceleaşi condiţii. Ca etalon se utilizează de obicei o antenă ipotetică care ar radia egal în toate direcţiile şi ar avea randamentul 100%. Deoarece energia se conservă, câştigul este pe unele direcţii supraunitar şi pe altele subunitar, integrala lui pe întreaga sferă fiind $4\pi\eta$ (unde η reprezintă randamentul antenei). Câştigul este dat uneori direct, alteori este dat logaritmul câştigului, exprimat în decibeli.

Câştigul antenei pe diverse direcţii este reprezentat grafic prin *diagramele de câştig*. O astfel de diagramă este o reprezentare a câştigului ca funcţie de unghi pe toate direcţiile dintr-un plan.

O direcţie de maxim local al câştigului, împreună cu direcţiile apropiate, se numeşte *lob*. Lobul care cuprinde maximul global al câştigului se numeşte *lobul principal* al antenei. Ceilalţi lobi se numesc *lobi secundari* sau *lobi laterali*. Valoarea maximă, pentru toate direcţiile posibile, a câştigului este numită *câştigul antenei*.

O antenă optimizată să aibă câştig cât mai mare pe o direcţie, în detrimentul celorlalte direcţii, se numeşte *antenă directivă*. O antenă optimizată pentru a avea câştig cât mai uniform, cel puţin în planul orizontal, se numeşte *antenă nedirectivă*. O antenă cu câştig perfect uniform de jur împrejur (radiator izotrop) este imposibil de realizat.

Există o legătură între dimensiunea antenei, directivitatea şi lungimea de undă la care funcţionează. Anume, raza unghiulară a lobului principal (măsurată în radiani) nu poate fi mai mică decât raportul dintre diametrul antenei şi lungimea de undă. Ca exemplu, pentru a obţine 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 puţin 1,2 m diametru. Limitarea aceasta este legată de fenomenele de difracţie a undelor şi nu poate fi ocolită.

O antenă de recepţie plasată în calea undelor recepţionează o putere proporţională cu densitatea de putere a undei incidente. Raportul dinte puterea disponibilă la bornele antenei şi densitatea de putere a undei incidente se numeşte *aria efectivă* sau *apertura* antenei. Apertura poate fi privită ca suprafaţa, transversală pe direcţia de propagare a undelor, de pe care antena preia întreaga energie. Apertura depinde de direcţia considerată a undei incidente.

Apertura faţă de o anumită direcţie a undei incidente este proporţională cu câştigul antenei pe acea direcţie. Relaţia este:

$$S = G\frac{\lambda^2}{4\pi} \tag{3.6}$$

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

Utilizând relaţia (3.6), se poate calcula puterea recepţionată, dacă distanţa dintre emiţător şi 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 emiţătoare, d este distanţa dintre emiţător şi receptor, G_e este câştigul emiţătorului pe direcţia spre receptor, iar G_r şi S_r sunt respectiv câştigul şi apertura antenei receptoare pe direcţia spre emiţător.

Exemplul 3.1: Considerăm un emiţător (de exemplu, un calculator dintr-o reţea IEEE 802.11 — *wireless*) care emite un semnal cu puterea $P_e = 100$ mW (sau, echivalent, +20 dBm) şi frecvenţa $f = 2{,}4$ GHz (lungimea de undă este atunci $\lambda = 0{,}125$ m). Mai presupunem că receptorul se găseşte la o distanţă $d = 100$ m faţă de emiţător, că absorbţia semnalului este neglijabilă (emiţătorul şi receptorul se găsesc în câmp deschis şi nu plouă) şi că ambele antene au un câştig $G_e = G_r = 2$ pe direcţia spre partenerul de comunicaţie. Rezultă puterea semnalului recepţionat:

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

adică aproximativ −84 dBm.

3.5.2.2. Polarizarea

Antenele cele mai simple au polarizare liniară: unda emisă este polarizată liniar, pe o direcţie stabilită prin construcţia antenei. Rotirea antenei emiţătorului faţă de cea a receptorului duce la variaţia semnalului recepţionat între un maxim (când direcţiile polarizărilor celor două antene sunt paralele) şi un minim (teoretic zero) când direcţiile sunt perpendiculare.

O antenă polarizantă liniar va recepţiona întotdeauna, indiferent de direcţia de polarizare, o transmisie polarizată circular; reciproc, o antenă polarizantă circular va recepţiona o emisie polarizată liniar. O antenă polarizantă circular va recepţiona o transmisie polarizată circular numai dacă are acelaşi sens al polarizării. Rotirea antenelor în jurul dreptei ce le uneşte 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 câştig maxim în planul orizontal (perpendicular pe baston) şi zero pe direcţie verticală (în lungul bastonului). Undele produse sunt polarizate vertical.

Antenele directive cele mai răspândite pentru comunicaţii de date sunt aşa-numitele *antene parabolice* (denumire improprie, pentru că forma parabolică este a reflectorului antenei). O asrfel de antenă este alcătuită dintr-o oglindă în formă de paraboloid de rotaţie, în focarul căreia este plasată antena propriu-zisă. (În alte construcţii, 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 acţiune 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 influenţează raza de acţiune a unei legături radio. Uneori vom dori să îi contracarăm, pentru a extinde domeniul de acţiune, alteori dimpotrivă, îi vom dori să ne menţină o legătură radio într-un domeniu spaţial 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 acţiune a legăturii radio. Mai mult, din cauza interferenţei dintre undele reflectate pe diferite căi, este dificil de analizat

exact punctele în care este posibilă recepţia unei emisii radio şi punctele în care emisia este obstrucţionată.

3.5.3.2. Linia orizontului

Unul dintre obstacolele ce limitează raza de acţiune a undelor radio este însuşi Pământul, prin curbura suprafeţei sale. O staţie aflată la o anumită înălţime poate comunica cu o staţie aflată la nivelul solului dacă şi numai dacă staţia de pe sol se află mai aproape decât linia orizontului celeilalte staţii. Două staţii pot comunica dacă există cel puţin un punct comun orizontului celor două staţii.

În câmpie, distanţa până la linia orizontului este (r desemnează raza Pământului, iar h este înălţimea antenei deasupra suprafeţei 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) şi h în metri, distanţa d rezultă în kilometri.

Exemple:

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

Un turn cu înălţimea de 20 m (obişnuit pentru un releu GSM) are linia orizontului la 16 km. O staţie aflată într-un astfel de turn poate comunica cu un radiotelefon ţinut în mână la o distanţă de 16 km+4,5 km = 20,5 km (de regulă raza de acţiune a unui releu GSM este limitată de alte considerente).

De pe un turn cu înălţimea de 50 m, distanţa la linia orizontului este $d = \sqrt{12{,}7 \cdot 50}$ km ≈ 25 km. Două relee de telecomunicaţii având 50 m înălţime fiecare pot comunica direct dacă sunt la mai puţin de 50 km unul de altul.

Distanţa la linia orizontului creşte încet cu înălţimea; dacă se dublează înălţimea, distanţa la linia orizontului creşte cu un factor de $\sqrt{2} \approx 1{,}4$.

3.5.3.3. Utilizarea sateliţilor artificiali ai Pământului

Sateliţii artificiali ai Pământului sunt utilizaţi ca echivalentul unor turnuri înalte pentru montarea unor staţii radio. După altitudinea la care

sunt plasaţi, distingem trei categorii de sateliţi:

sateliţi de joasă altitudine aflaţi între 200 . . . 1000 km, cu perioada de rotaţie de 1,5 . . . 1,8 h;

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

sateliţi geostaţionari aflaţi la 35800 km deasupra ecuatorului, au perioada de rotaţie de exact o zi şi ca urmare apar ficşi faţă de Pământ.

Un satelit are o arie de acoperire incomparabil mai mare faţă de o staţie 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 distanţelor mari, comunicaţia cu sateliţii necesită fie puteri mari, fie antene cu directivitate foarte bună. Este de remarcat faptul că distanţa de la un satelit la o staţie terestră este de la câteva zeci la câteva sute de ori mai mare decât distanţa de la un releu amplasat într-un turn la o staţie terestră. Ca urmare, pentru aceleaşi antene, puterile necesare sunt de la cateva sute la câteva sute de mii de ori mai mari.

La comunicaţia între sateliţi geostaţionari şi staţii fixe de pe sol se pot utiliza relativ uşor antene cu directivitate bună, deoarece antenele de pe sol sunt fixe. Orbita geostaţionară este însă destul de „aglomerată“: presupunând că avem antene ce dau un fascicul cu diametrun 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 sateliţi distincţi.

Pentru sateliţii care nu sunt geostaţionari, 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 şi, în consecinţă, fixează puterea minimă pentru o anumită distanţă emiţător-receptor.

Nivelul de zgomot scade în general o dată cu creşterea frecvenţei.

3.5.3.5. Scăderea puterii cu distanţa

Densitatea de putere a undelor electromagnetice scade cu pătratul distanţei de la emiţător. Ca urmare, la o sensibilitate fixată a receptorului, pentru a dubla raza de acţiune a emiţătorului trebuie să-i creştem puterea de 4 ori.

Pe de altă parte, dacă două emiţătoare radio funcţionează în aceeaşi regiune geografică şi emit pe frecvenţe identice sau foarte apropiate, atunci transmisia mai puternică „acoperă“ transmisia mai slabă. Aceasta se întâmplă deoarece semnalele celor două emiţătoare se suprapun. Dacă, în punctul în care este plasat receptorul, puterea unuia dintre emiţătoare este mult mai mare decât puterea celuilalt, atunci receptorul va recepţiona doar transmisia mai puternică, chiar dacă, singură, transmisia mai slabă ar putea fi recepţionată corect. Dacă puterile sunt apropiate, receptorul nu va putea „înţelege“ nici una dintre transmisii.

3.5.3.6. Emisia direcţionată şi polarizată

Domeniul de acţiune a unui emiţător sau receptor poate fi restrâns în mod voit dotând emiţătorul sau receptorul (de obicei ambele) cu antene directive. Trebuie însă calculate cu atenţie divergenţa lobului principal, puterea emisă pe lobii secundari ai antenei şi reflexiile de teren.

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

3.5.4. Spectrul radio şi alocarea lui

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

În comunicaţii radio, termenul de frecvenţă se utilizează adesea şi cu sensul de interval de frecvenţe în care se încadrează o transmisie (efectiv, bandă în sensul de la semnale). Frecvenţe diferite, în acest sens, înseamnă de fapt benzi disjuncte. Valoarea numerică a frecvenţei, specificată în acest context, este frecvenţa purtătoare utilizată. Limitele efective ale benzii se determină din standardul de transmisie folosit.

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

Două transmisii radio ce se fac pe frecvenţe diferite, sau mai precis, a căror benzi de trecere sunt disjuncte, pot fi separate în general uşor. Separarea în frecvenţă este mult mai uşor controlabilă decât separarea spaţială studiată în § 3.5.3. Două transmisii pe aceeaşi frecvenţă şi în aceeaşi zonă geografică sunt practic imposibil de separat, dacă au puteri apropiate, sau transmisia mai slabă este imposibil de recepţionat fiind „acoperită“ de cea mai puternică.

Pentru evitarea suprapunerilor între utilizatori, utilizarea diverselor benzi de frecvenţe face obiectul unor reglementări legale în fiecare ţară, precum şi a unor acorduri internaţionale. Emiterea unui semnal radio, pe o frecvenţă pentru care operatorul emiţătorului nu este autorizat sau de o putere mai mare decât cea autorizată, poate duce la sancţionarea contravenţională sau chiar penală a operatorului.

În majoritatea cazurilor, un utilizator de comunicaţii radio care doreşte să opereze un emiţător trebuie să obţină o autorizaţie în care se specifică frecvenţa de lucru, puterea maximă, zona geografică în care operează, etc. Există frecvenţe alocate posturilor de radio, sistemelor de comunicaţii radio ale diferitelor instituţii (poliţie, controlorii de trafic aerian, dispecerate de taxiuri, operatori de telefonie mobilă, etc.). Tot în această categorie, însă cu un statut aparte sunt radioamatorii: frecvenţele sunt alocate activităţii de radioamator şi nu unei persoane sau instituţii, însă radioamatorii trebuie să se înregistreze pentru a putea emite.

Există însă benzi pentru care nu este necesară o autorizare expresă a emiţătorului, cu condiţia ca emiţătorul să nu depăşească o anumită putere. În această categorie intră frecvenţele folosite de: reţelele IEEE 802.11 (Wireless Ethernet) şi Bluetooth, tastaturi şi mauşi 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 totuşi să fie atent la eventualele diferenţe între reglementările din diferite ţări: un echipament poate funcţiona legal fără autorizaţie în ţara de origine, dar să necesite autorizaţie în altă ţară.

Echipamentele care lucrează pe frecvenţe pentru care nu trebuie autorizaţie ajung să interfereze dacă sunt plasate în apropiere. Unele dintre acestea permit selectarea frecvenţei de lucru dintre 2–4 frecvenţe predefinite. Utilizatorul va selecta o frecvenţă diferită dacă constată o funcţionare proastă

şi suspectează interferenţe cu echipamente vecine. Altă soluţie este schimbarea repetată a frecvenţei de lucru, după o schemă convenită între emiţător şi receptor, şi tolerarea unui număr de ciocniri ale transmisiilor pe perioadele în care echipamentele vecine se nimeresc aceeaşi frecvenţă. Tehnica se numeşte *frequency hopping* (salturi ale frecvenţei).

Mai menţionăm că, printre producătorii de semnale radio parazite intră şi alte dispozitive, având alte scopuri decât comunicaţiile. Ca fapt divers, enumerăm câteva:

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

3.5.5. Particularităţi ale sistemelor de comunicaţie prin radio

3.5.5.1. Topologia legăturii

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

Legăturile între sateliţii geostaţionari şi staţiile terestre sunt astfel că emisia satelitului este recepţionată de mai multe staţii de pe Pământ, şi reciproc, satelitul recepţionează emisia de la mai multe staţii de pe Pământ; staţiile de pe Pământ nu comunică însă direct între ele. O astfel de comunicaţie poate prezenta riscul ca emisiile staţiilor de pe Pământ să se ciocnească fără ca staţiile să observe direct acest lucru.

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

Pentru distanţe mici, se poate adopta o organizare mai „democratică“ (exemplu IEEE 802.11): staţiile 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 staţie de pe cablu este recepţionat de toate celelalte şi, ca urmare, ciocnirea la recepţie a două pachete este sesizată şi de către emiţătoare, la legăturile radio este posibil ca două transmisii să se ciocnească la receptor dar nici una din staţiile care le-au emis să nu recepţioneze 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 $10^{-3} \ldots 10^{-5}$. Pentru comparaţie, la transmisia prin perechi de conductoare, probabilitatea unei erori de un bit este de $10^{-7} \ldots 10^{-10}$, iar la fibrele optice, erorile sunt şi mai rare, $10^{-10} \ldots 10^{-12}$.
- La frecvenţe peste 10 GHz, datorită absorbţiei în picăturile de apă, starea legăturii poate depinde de starea vremii.
- Umbrele provocate de clădiri şi relief, precum şi interferenţele între undele reflectate, sunt imposibil de calculat în mod practic. O staţie ce ajunge în umbră va pierde legătura în mod imprevizibil.

3.5.5.3. Securitatea

La comunicaţiile prin cablu pe distanţă scurtă, securitatea comunicaţiei poate fi asigurată păzind cablul. Din acest motiv reţelele locale pe cablu pot să nu prevadă măsuri contra intruşilor.

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

3.6. Transmisia optică

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

lumină pentru aceste unde, deşi nu se încadrează în domeniul luminii vizibile ($\lambda = 780$ nm $\ldots$ 380 nm).

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

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

Pentru unele aplicaţii, presupunând comunicaţie pe distanţă de cel mult câţiva metri (de exemplu, pentru telecomenzi de televizoare sau pentru dispozitive IrDA), raza de lumină se propagă direct prin aer de la emiţător la receptor. Metoda este dificil de extins la distanţe mai mari.

Raza de lumină poate fi însă foarte uşor ghidată printr-o fibră optică. O fibră optică este în esenţă un fir dintr-un material transparent, prin interiorul căruia trece lumina. Dacă raza de lumină loveşte 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 numeşte şi *ghid de undă optic* (engl. *optical waveguide*), deoarece este identic ca şi scop şi foarte asemănător funcţional cu ghidul de undă utilizat pentru microunde.

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

3.6.1. Construcţia mediului

Constructiv, o fibră optică este alcătuită dintr-un *miez* (engl. *core*) din silica (bioxid de siliciu, SiO_2, amorf), înconjurat de un *înveliş* (engl. *cladding*), tot din silica, dar cu un indice de refracţie puţin mai mic. Diametrul miezului este principalul parametru dat la o fibră optică; este cuprins între 8 μm şi 62,5 μm. Diametrul învelişului este în mod curent de 125 μm. Pentru comparaţie, diametrul firului de păr uman este de 20 $\ldots$ 30 μm.

Între miez şi înveliş poate fi o discontinuitate netă, sau se poate ca indicele de refracţie 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 înveliş se numesc *fibre optice graduale* (engl. *grade index fiber*).

Fibra propriu-zisă fiind extrem de subţire şi fragilă, ea este învelită

în mai multe straturi cu rol de protecţie mecanică.

Ideea de bază a conducerii semnalului prin fibră este că o rază de lumină ce se propagă oblic prin miez şi atinge suprafaţa de contact dintre miez şi înveliş 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 faţă 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 şi având grijă ca să se lipească capetele dar să nu se amestece miezul cu învelişul. Conectarea prin lipire necesită echipamente mai scumpe, este nedemontabilă, dar perturbă cel mai puţin transmiterea semnalului prin fibră. O lipitură produce o atenuare a semnalului în jur de 0,1 dB, din cauza reflexiei unei părţi a luminii incidente.
- *prin conectoare optice*. Fiecare capăt de fibră se şlefuieşte foarte bine şi se prinde într-o piesă metalică cu rol de ghidaj. Piesele metalice ataşate capetelor de fibră se strâng una faţă de cealaltă, realizând alinierea faţă în faţă a capetelor de fibră. Eventual, spatiul dintre capetele de fibră se poate umple cu un gel transparent cu indice de refracţie 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 ecuaţiei undelor rezultă doar un număr finit de soluţii, numite *moduri de propagare*. Intuitiv, un mod este un posibil traseu al razei de lumină, traversând în mod repetat, în zig-zag, axul fibrei şi păstrând un unghi fixat faţă de acesta; în fibre suficient de subţiri, doar anumite unghiuri sunt permise.

Dacă o fibră permite existenţa mai multor moduri de propagare a luminii, fibra se numeşte *multimod*. Modurile diferite se propagă în general cu viteze puţin 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 direcţia

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

Opusul fibrei multimod este fibra *monomod*, în care ecuaţia undelor admite o singură soluţie. Existenţa unui singur mod elimină dispersia intermodală, îmbunătăţind calitatea propagării semnalului. Pentru a admite un singur mod, fibra trebuie să fie mult mai subţire, diametrele standard fiind 10 μm sau 8 μm. Diametrul mai mic al fibrei atrage două dificultăţi: pe de o parte, cerinţele de aliniere mecanică a fibrei faţă 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) şi, în consecinţă, la creşterea preţurilor 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, aşa 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 lăţirea impulsurilor. Sunt cauzate de mai multe fenomene, şi au ca şi consecinţă limitarea practică a produsului dintre frecvenţa maximă ce se poate transmite şi distanţa dintre emiţător şi receptor. Acest produs se numeşte (impropriu) banda de trecere şi 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 şi, la atenuare dată a fibrei, puterea emiţătorului. Captarea de paraziţi de-a lungul fibrei, şi 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. Frecvenţa purtătoare este frecvenţa undelor infraroşii. Semnalul modulator este rădăcina pătrată a puterii luminoase emise.

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

Este posibilă şi transmisia duplex pe o singură fibră optică. Pentru aceasta se realizează o construcţie 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 sensuri 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 sensuri.

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 puţin sensibile la zgomot, este dificil de interceptat comunicaţia prin ele (fără a le tăia este aproape imposibil de interceptat semnalul, iar tăierea fibrei poate fi uşor detectată), au atenuare mică şi, în sfârşit, sunt mult mai uşoare (conţin mult mai puţin material) decât perechile de conductoare.

Toate aceste avantaje fac fibrele optice să fie extrem de atractive pentru comunicaţia pe distanţe mari, precum şi pentru echipamente ce lucrează în condiţii mai speciale, de exemplu la tensiuni electrice mari sau în medii cu radiaţii electromagnetice puternice.

Principalele dificultăţi la utilizarea fibrelor optice sunt legate de cablare.

Deşi puterea luminii transportate prin fibra optică este foarte mică, secţiunea 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 emiţătorul 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 protecţie, se pot utiliza ochelari speciali prevăzuţi cu filtre care

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

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

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

Capitolul 4

Nivelul legăturii de date

Nivelul legăturii de date are ca rol realizarea unei comunicaţii stabile între calculatoare sau echipamente între care există o legătură directă la nivel fizic (există deci un mediu de comunicaţie 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 acelaşi mediu de transmisie este utilizat de mai multe emiţătoare (ceea ce se întâmplă adesea la transmisia prin unde radio, dar uneori şi la transmisia prin perechi de conductoare) şi mai multe dintre aceste emiţătoare transmit simultan, pachetele transmise se alterează reciproc.
- Dacă destinaţia nu poate prelucra datele în ritmul în care sunt transmise de către emiţător, o parte din date se vor pierde.
- Construcţia 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 aceeaşi 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 emiţătoare ce partajează acelaşi mediu de transmisie;
- *retransmiterea pachetelor pierdute* din cauza erorilor de transmisie, a ciocnirilor între pachete transmise de mai multe emiţătoare simultan sau a incapacităţii destinaţiei de-a le prelua la timp;

- *controlul fluxului* de date, adică frânarea emiţătorului în cazul în care destinaţia nu este capabilă să proceseze suficient de repede informaţia primită;
- *multiplexarea* mai multor legături logice prin aceeaşi legătură fizică.

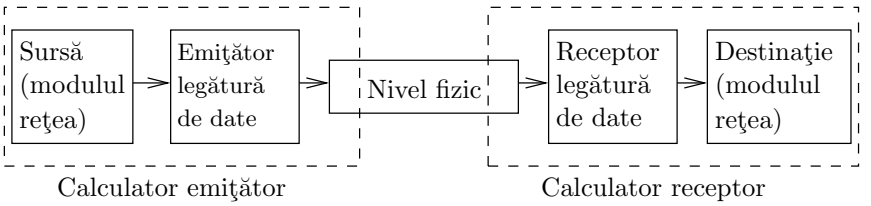


Figura 4.1: Alcătuirea nivelului legăturii de date şi locul său între nivelele reţelei.

Constructiv, nivelul legăturii de date este un modul interpus între nivelul superior (în mod normal, nivelul reţea) şi nivelul fizic (fig. 4.1). Pentru realizarea funcţiilor lor, modulele nivelului legăturii de date ale dispozitivelor ce comunică îşi transmit unul altuia, utilizând serviciile nivelului fizic, două tipuri de informaţii:

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

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

4.1. Detectarea şi corectarea erorilor

În vederea detectării sau, după caz, corectării erorilor, emiţătorul de la nivelul legăturii de date adaugă, la fiecare pachet generat de nivelul superior, un număr de *biţi de control*. Biţii de control sunt calculaţi conform unui mecanism de codificare pentru canale cu perturbaţii (vezi § 2.4). Biţii de control sunt adăugaţi, de regulă, la finalul pachetului.

Receptorul recalculează biţii de control conform conţinutului pachetului recepţionat şi-i compară cu cei de la finalul pachetului recepţionat. În caz de nepotrivire, receptorul deduce că s-a produs o eroare de transmisie. În cazul utilizării unui cod corector de erori, receptorul reconstituie conţinutul

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 situaţia în care pe un acelaşi mediu fizic acţionează mai multe emiţătoare, a căror emisie simultană interferează în aşa fel încât un receptor nu poate recepţiona corect oricare dintre transmisii. În aceste condiţii, 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 comunicaţiei bidirecţionale, între două entităţi, utilizând acelaşi mediu fizic pentru ambele sensuri.
- la comunicaţia prin unde radio, dacă există mai multe staţii care emit pe aceeaşi lungime de undă. În general, emisia unei staţii este recepţionată de toate staţiile pe o anumită rază. Este cazul aproape tuturor reţelelor fără fir: IEEE 802.11 (*wireless Ethernet*), Bluetooth, GSM, etc.
- dacă staţiile sunt conectate „tip magistrală“, adică mediul de comunicaţie — în general o pereche de conductoare — trece pe la toate staţiile. Este cazul reţelelor *Ethernet* mai vechi.

Există două strategii de control al accesului la mediu:

- *asigurarea unui interval exclusiv de emisie*, pe rând, pentru fiecare staţie;
- *acceptarea posibilităţii coliziunilor şi retransmisia pachetelor distruse* în coliziuni.

Asigurarea unui interval exclusiv de emisie permite garantarea, pentru fiecare staţie, a unui debit minim cu care poate emite şi a unui interval maxim de aşteptare din momentul în care are ceva de transmis şi până la intrarea în emisie; metoda cu coliziuni şi retransmiteri este nedeterministă şi ca atare asigură un anumit debit şi un anumit timp de aşteptare doar cu o anumită probabilitate strict mai mică decât unu. În schimb, în sistemele ce asigură un interval exclusiv de emisie, intrarea şi ieşirea unei staţii din reţea, precum şi revenirea după o pană a unei staţii, 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 staţii; 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 reţelele utilizate pentru automatizări industriale transmisie audio-video. Detectarea coliziunilor şi retransmiterea pachetelor distruse în coliziuni este favorabilă în sistemele interactive, cum ar fi reţelele „obişnuite“ de calculatoare.

Aproape în orice sistem în care mai multe dispozitive sunt conectate la acelaşi mediu fizic este necesar ca fiecare dispozitiv să aibă un identificator unic. Acest identificator se numeşte *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 reţelei, adică adresele sunt alocate fie manual, de către administratorul reţelei, fie în cadrul procesului de fabricaţie al dispozitivului conectat la reţea.

4.2.1. Protocoale bazate pe asigurarea unui interval exclusiv de emisie

Cea mai simplă metodă din această categorie este să existe o staţie desemnată ca *arbitru*, care să anunţe de fiecare dată ce staţie primeşte dreptul de emisie. Anunţul se face printr-un pachet emis de arbitru şi conţinând adresa fizică a staţiei ce poate emite. Staţia anunţată de arbitru are la dispoziţie un interval de timp în care poate să emită ceea ce are de transmis. Dacă staţia nu are nimic de transmis, protocolul poate prevede fie că staţia nu emite nimic, fie că emite un pachet special. Încheierea perioadei alocate unei staţii se poate face fie la expirarea unei durate de timp prestabilite, fie prin anunţul explicit al staţiei că a încheiat transmisia. După încheierea perioadei alocate unei staţii, arbitrul anunţă staţia următoare.

Arbitrul trebuie să aibă lista tuturor staţiilor din reţea. Ieşirea unei staţii se face simplu prin anunţarea arbitrului; ieşirea arbitrului nu este posibilă (decât eventual prin desemnarea unui alt arbitru). Intrarea unei staţii noi necesită un mecanism special de anunţare a arbitrului. Un astfel de mecanism este în general bazat pe coliziuni şi prevede ca arbitrul să întrebe, periodic, dacă există staţii ce vor să intre în reţea. Dacă o staţie, alta decât arbitrul, se defectează, staţia fie este văzută ca o staţie ce nu are nimic de transmis, fie este detectată de către arbitru că nu răspunde şi este scoasă de pe lista staţiilor din reţea. Defectarea arbitrului duce la căderea întregii reţele.

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 deţine lista completă a staţiilor, lista este distribuită, fiecare staţie cunoscând adresa staţiei următoare. În acest fel, în intervalul de emisie alocat, fiecare staţie emite datele utile, după care anunţă staţia următoare. Metoda cu jeton a fost utilizată în reţelele IEEE 802.4.

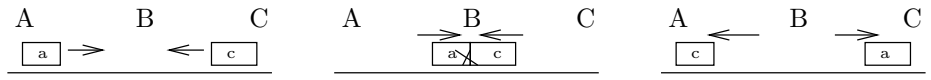
4.2.2. Protocoale bazate pe coliziuni şi retransmitere

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

Detectarea unei coliziuni, de către fiecare dintre staţiile emiţătoare, 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 staţie receptoare şi să nu fie în coliziune pentru altă staţie (fig 4.2). Din acest motiv, un emiţător trebuie să considere coliziune orice situaţie în care detectează o transmisie prea apropiată în timp de o transmisie proprie. Din acelaşi motiv, întârzierea maximă datorată propagării în reţea trebuie limitată prin standard, ceea ce impune o limită asupra întinderii geografice a reţelei.



(a) A şi C emit simultan câte un pachet scurt. Fiecare dintre ei termină emisia propriului pachet cu mult înaintea sosirii pachetului celuilalt.

(b) Ceva mai târziu, ambele pachete ajung la B, unde se produce coliziune.

(c) Şi mai târziu, A primeşte pachetul lui C şi C primeşte pachetul lui A.

Figura 4.2: Două pachete emise simultan, în condiţiile în care timpul de propagare este mai mare decât timpul de transfer. Coliziunea nu este detectată de nici unul dintre emiţătoare, însă este detectată de o staţie aflată la jumătatea distanţei dintre acestea.

La legăturile radio poate să mai apară un fenomen, şi 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 recepţia celui de-al doilea pachet (fig. 4.3). Din acest motiv, la transmisia radio este imposibil ca emiţătorul să detecteze întotdeauna coliziunile propriei transmisii cu alte transmisii simultane.

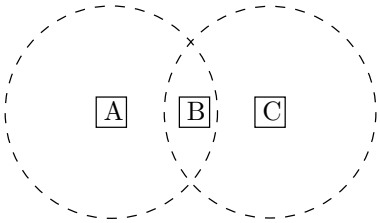


Figura 4.3: Este posibil ca două emiţătoare radio, A şi C, să fie situate prea departe pentru a-şi recepţiona una transmisia celeilalte, dar să existe o a treia staţie, B, care să recepţioneze transmisiile ambelor emiţătoare (liniile punctate delimitează zona în care transmisia unei staţii poate fi recepţionată). În acest caz, A şi C pot emite simultan fără a detecta coliziune, însă pentru B se produce coliziune între transmisiile lui A şi 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 detecteaze 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 staţii ce au emis simultan vor emite simultan şi retransmiterile, ciocnindu-şi 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 numeşte *Aloha pur*.

O variantă îmbunătăţită a protocolului *Aloha* este protocolul numit *Aloha cuantificat* (engl. *slotted Aloha*). În acest protocol, toate pachetele au aceeaşi 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 îmbunătăţiri ce pot fi aduse protocolului *Aloha pur* (nu însă şi la *Aloha cuantificat*) sunt:

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

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

O primă posibilitate este ca staţia să înceapă emisia proprie imediat după terminarea emisiei 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 emisiei 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 purtătoarei 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 reţele fără fir deoarece rezolvă şi aşa-numita *problema staţiei ascunse*: este posibil să existe trei staţii, A, B şi C, cu B situată geografic aproximativ la mijlocul distanţei între A şi C, cu distanţa dintre A şi C puţin peste raza de acţiune a transmisiei, astfel încât A nu recepţionează transmisia lui C şi nici reciproc, dar cu B suficient de aproape de A şi de C astfel încât să poată comunica cu fiecare dintre ele. În această situaţie, dacă A şi 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 doreşte să transmită la rândul lui; ca urmare, CSMA se comportă exact ca Aloha pur. În protocolul CSMA/CA, în schimb, C recepţionează CTS-ul adresat de B lui A şi amână transmisia proprie.

Altă posibilitate de combinare a celor două strategii o constituie *protocolalele cu conflict limitat*. În cazul acestor protocoale, staţiile sunt împărţite în grupuri. Fiecărui grup i se alocă intervale exclusive de emisie (ca în cazul protocoalelor bazate pe intervale exclusive de emisie, dar cu diferenţa că fiecare interval se alocă unui întreg grup, nu unei staţii). În cadrul fiecărui grup se aplică un protocol cu coliziuni şi retransmitere. Împărţirea 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 şi corectată de receptor), este necesară retransmiterea acelui pachet.

Evident, emiţătorul nu are cum să „ghicească“ dacă un anumit pachet ajunge intact la destinaţie sau este pierdut; ca urmare, trebuie stabilită o comunicaţie înapoi dinspre receptor spre emiţător. 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 şi ele pachete şi deci sunt la rândul lor supuse eventualelor erori de transmisie.

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

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

4.3.1. Principiul confirmărilor pozitive şi retransmiterilor

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

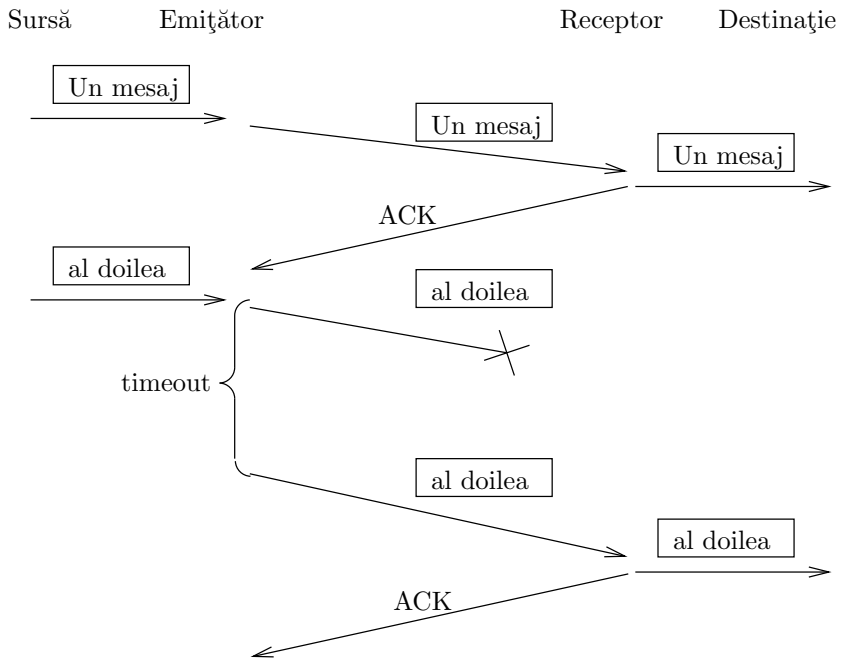


Figura 4.4: Retransmiterea pachetelor pierdute

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

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

Algoritmul receptorului este următorul: pentru fiecare pachet primit de la emiţător, trimite un pachet de confirmare spre emiţător şi livrează destinaţiei pachetul primit.

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

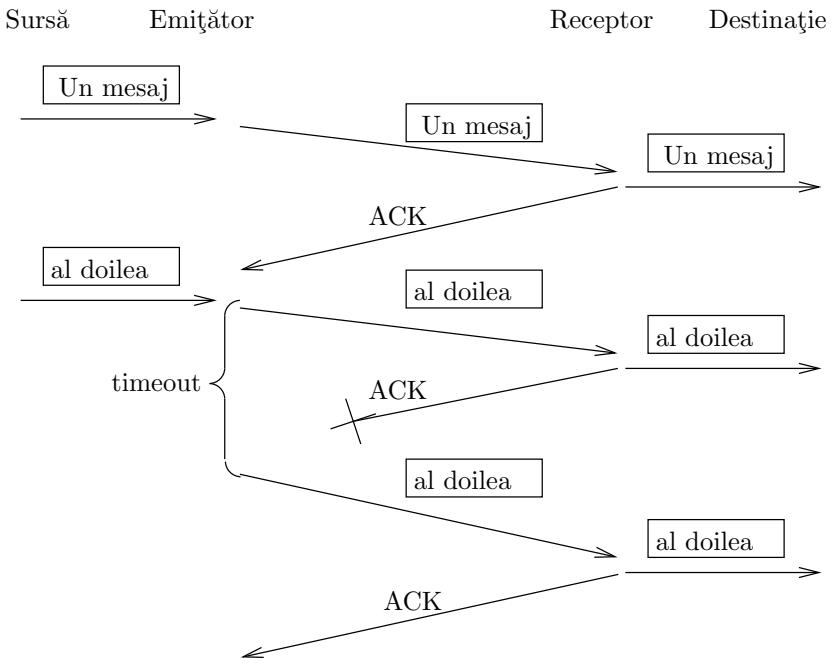


Figura 4.5: În lipsa unor măsuri adecvate, pierderea unei confirmări conduce la dublarea unui pachet.

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

Pentru a obţine un algoritm corect, trebuie să furnizăm receptorului suficientă informaţie pentru ca acesta să distingă între repetarea pachetului precedent şi pachetul următor. O soluţie este ca emiţătorul să pună în fiecare pachet trimis numărul său de ordine în cadrul fluxului de date; acest număr de ordine se numeşte *numărul de secvenţă* al pachetului. În acest fel, receptorul va reţine numărul ultimului pachet recepţionat şi va fi capabil să verifice dacă următorul pachet primit este repetarea acestuia sau este următorul pachet al

sursei.

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

Să vedem raţionamentul ce ne duce la răspuns: Emiţătorul nu are de unde să ştie dacă un pachet de date a ajuns sau nu la receptor. Dacă primeşte confirmarea unui pachet, poate fi sigur că pachetul confirmat a ajuns la receptor; dacă însă a trimis un pachet şi nu a primit (încă) confirmarea acestuia, este posibil (conform informaţiilor emiţătorului) ca pachetul să fi ajuns (şi 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).

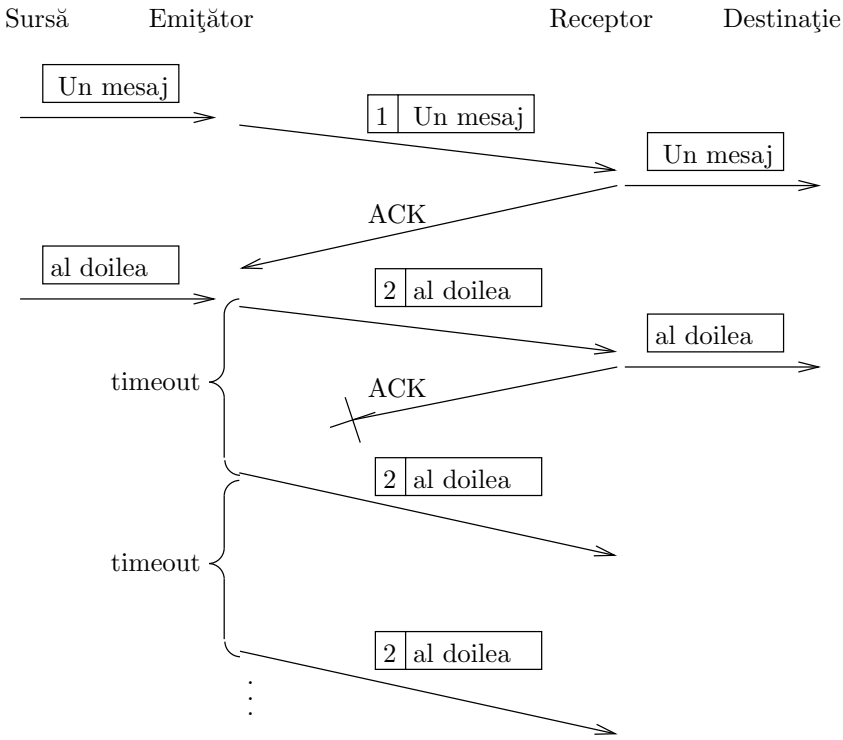


Figura 4.6: Neconfirmarea duplicatelor determină emiţătorul să repete la infinit un pachet a cărui confirmare s-a pierdut.

Suntem acum în măsură să dăm algoritmii corecţi pentru emiţător şi pentru receptor: funcţionarea emiţătorului este descrisă în algoritmul 4.1, iar cea a receptorului în algoritmul 4.2. Un exemplu de funcţionare a acestora

este dat în figura 4.7

Algoritmul *Emiţător_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.

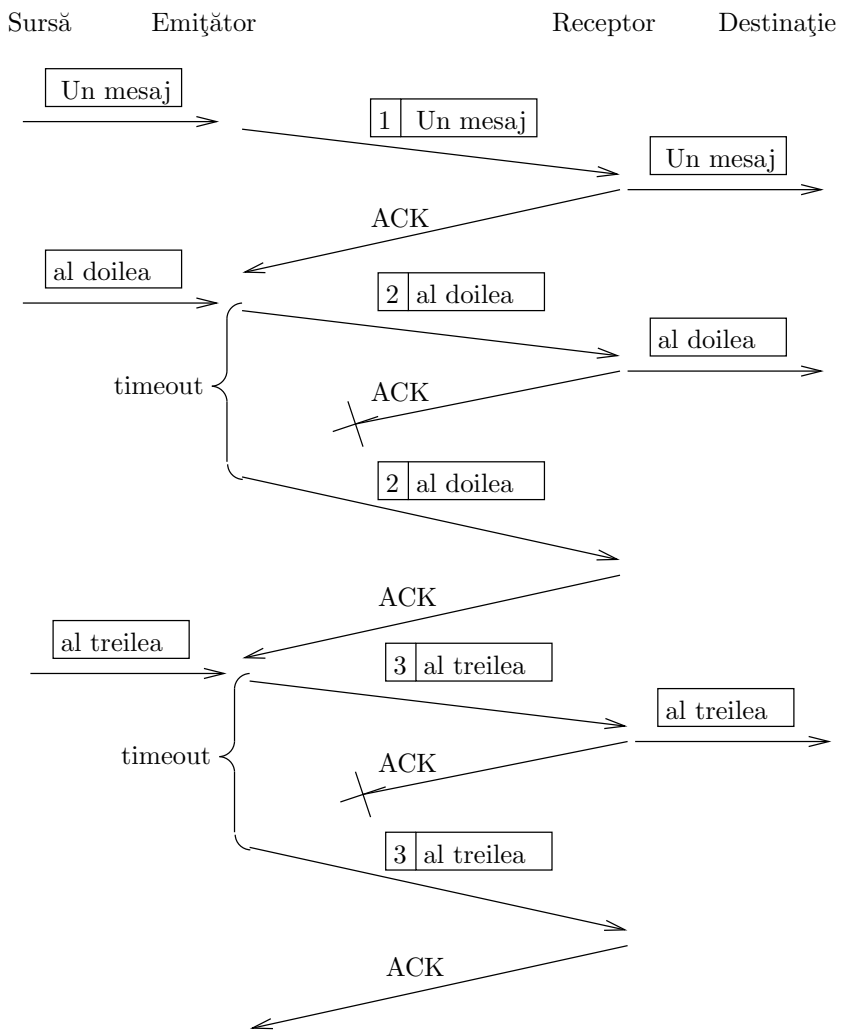


Figura 4.7: Funcţionarea corectă a unui algoritm de retrasnmisie.

4.3.2. Trimiterea în avans a mai multor pachete

Deşi corect, algorimul pas cu pas din paragraful precedent este ineficient în situaţia în care timpul dus-întors între emiţător şi 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 şi la distanţă mare. În această situaţie, emiţătorul emite repede un pachet, după care aşteaptă (cea mai mare parte a timpului) propagarea pachetului spre destinatar şi întoarcerea confirmării.

Pentru creşterea eficienţei, ar fi util ca emiţătorul să poată trimite unul după altul mai multe pachete, fără a aştepta confirmarea primului pentru a-l trimite pe următorul. Timpul maxim de aşteaptare pentru confirmarea unui pachet rămâne neschimbat, însă permitem trimiterea mai multor pachete în acest timp.

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

- Deoarece la un anumit moment pot exista mai multe pachete trimise de emiţător şi neconfirmate, pentru a putea corela confirmările cu pachetele de date, este necesar ca şi confirmările să fie numerotate. Există două strategii posibile:
  - Fiecare pachet de date este confirmat printr-un pachet de confirmare distinct, conţinând numărul de secvenţă al pachetului confirmat.
  - Un pachet de confirmare conţine numărul de secvenţă până la care receptorul a primit toate pachetele. Cu alte cuvinte, un pachet de confirmare confirmă toate pachetele de date cu număr de secvenţă 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 menţionat că alegerea între aceste variante trebuie consemnată ca parte a protocolului stabilit între emiţător şi receptor.
- În urma pierderii unui pachet, receptorul poate ajunge în situaţia de-a primi un pachet fără să fi primit mai întâi pachetul anterior. În această situaţie, receptorul nu poate livra imediat acel pachet destinaţiei. Există două acţiuni posibile pentru receptor:
  - Receptorul ignoră complet pachetul (în consecinţă, nici nu trimite confirmare).
  - Receptorul memorează pachetul, urmând să-l livreze destinaţiei după primirea pachetului (sau pachetelor) anterioare.

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

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

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

Emiţătorul trebuie să ţină şi el evidenţa unei *ferestre de emisie*, conţinând pachete, primite de la sursă în vederea trimiterii spre receptor, dar încă neconfirmate de către receptor. Notând cu n primul număr de secvenţă neconfirmat şi cu k dimensiunea ferestrei emiţătorului, fereastra emiţătorului poate conţine pachetele cu numere de la n la $n+k-1$. Emiţătorul trimite, în mod repetat, pachetele din fereastră, până ce primeşte confirmări pentru ele. În momentul în care pachetul n este confirmat, fereastra emiţătorului avansează până la următorul pachet neconfirmat.

Din cauza ferestrelor emiţătorului şi receptorului, protocolul acesta se numeşte *protocolul ferestrei glisante*. Dacă fereastra emiţătorului este de dimensiune 1, protocolul ferestrei glisante funcţionează exact ca şi protocolul pas cu pas din paragraful precedent.

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

4.3.3. Spaţiul numerelor de confirmare

Evident, în practică, numerele de secvenţă nu pot creşte la infinit. În general, numerele de secvenţă sunt reprezentate pe un număr fixat de biţi şi, ca urmare, au o valoare maximă după care se reiau de la 0.

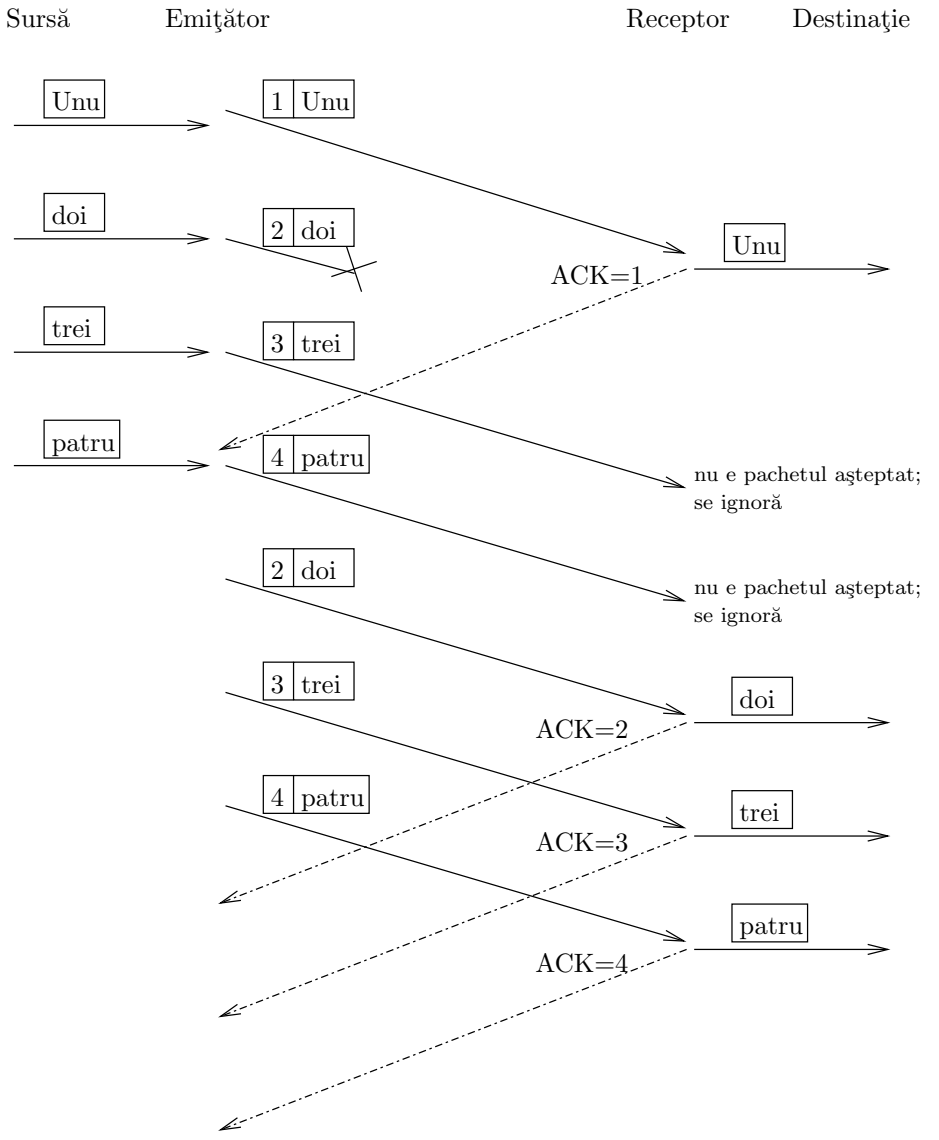


Figura 4.8: Funcţionarea ferestrei glisante în cazul în care dimensiunea ferestrei de recepţie este 1 şi dimensiunea ferestrei de emisie este cel puţin 3.

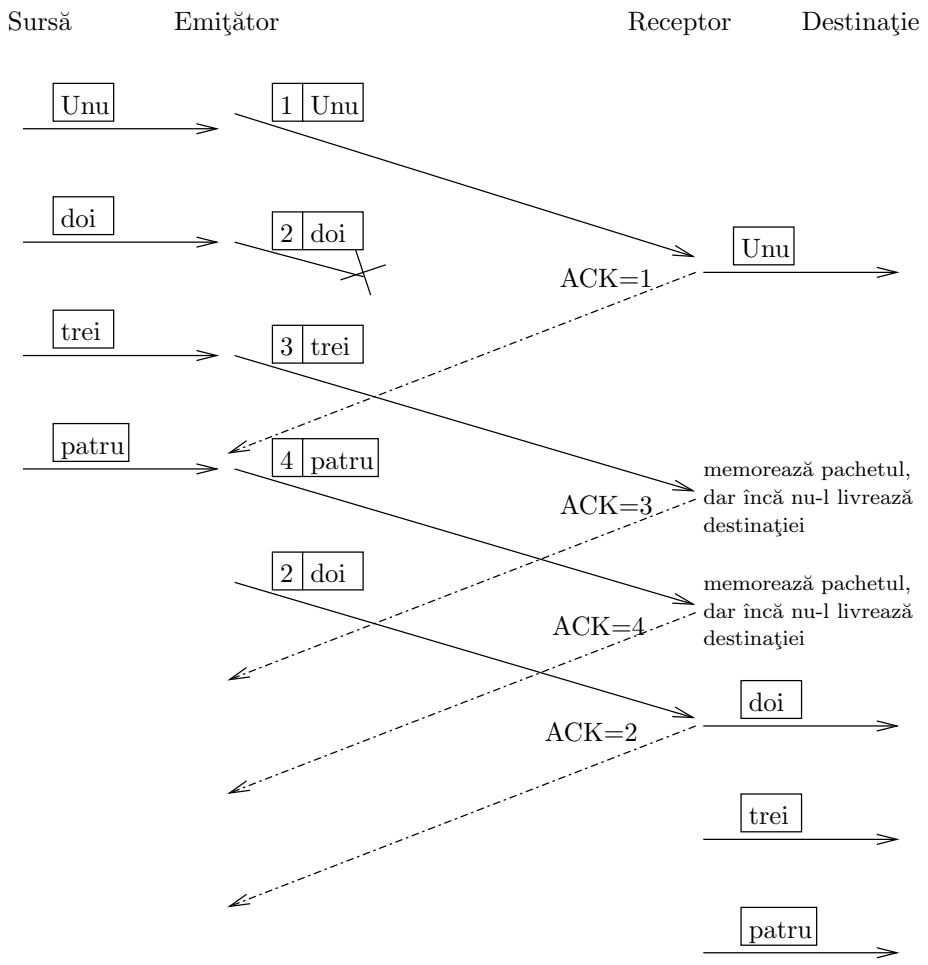


Figura 4.9: Funcţionarea ferestrei glisante în cazul în care dimensiunea ferestrei de recepţie este cel puţin 3 şi protocolul prevede confirmarea individuală a pachetelor.

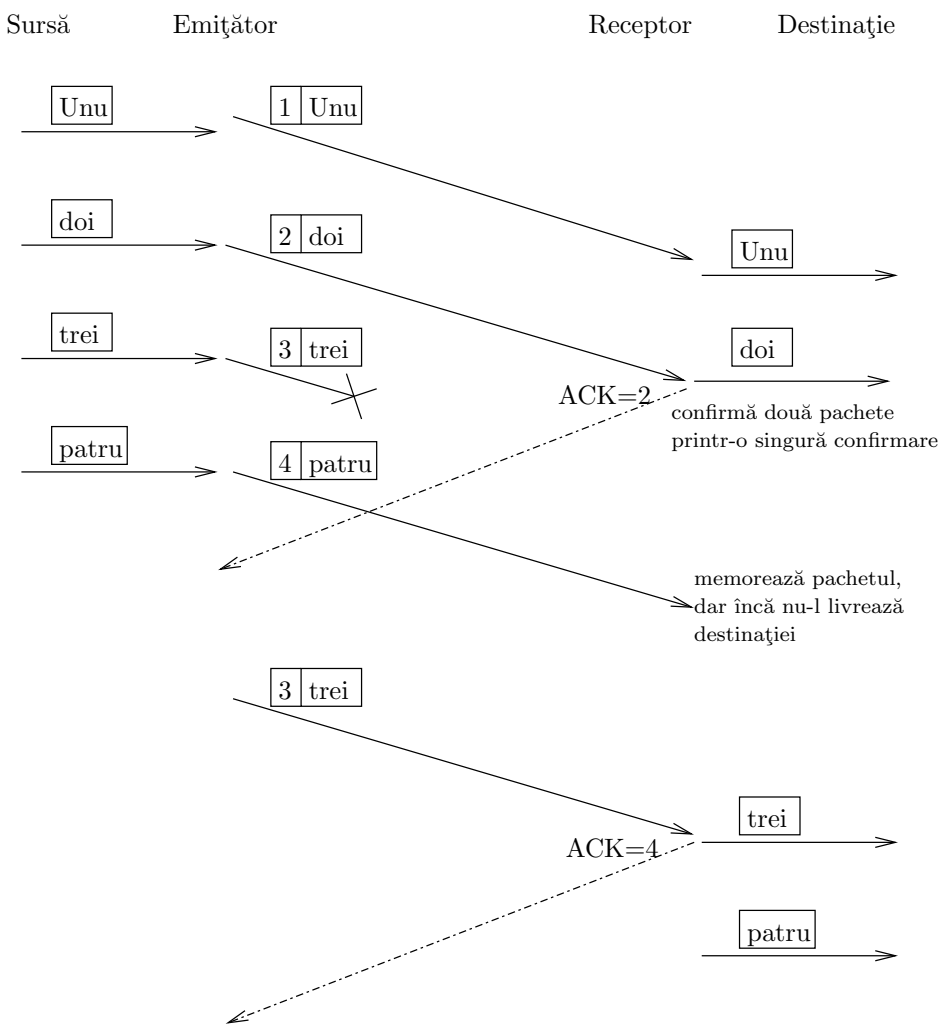


Figura 4.10: Funcţionarea ferestrei glisante în cazul în care dimensiunea ferestrei de recepţie este cel puţin 2 şi protocolul prevede că un pachet de confirmare confirmă toate pachetele de date până la numărul de secvenţă conţinut în pachet. De remarcat posibilitatea de optimizare a numărului de pachete de confirmare prin combinarea mai multor confirmări într-un singur pachet (confirmarea cu numărul 2).

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

Pentru ca mecanismele de confirmare şi retransmitere, descrise în § 4.3.1 şi § 4.3.2, să funcţioneze corect, ele trebuie modificate în aşa fel încât să compare efectiv numerele de secvenţă teoretice. Pentru aceasta, este necesar ca, în orice moment, atât receptorul cât şi emiţătorul să poată, pe baza informaţiilor pe care le au, să deducă univoc numărul de secvenţă teoretic din numărul de secvenţă transmis.

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

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

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

Presupunem că cel mai mare număr de secvenţă primit de către receptor este n. Deoarece emiţatorul a trimis deja pachetul n, rezultă că pachetele până la $n-k$ inclusiv au fost deja confirmate şi 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 confimat şi deci receptorul nu poate trimite pachete cu numere de secvenţă strict mai mari decât $n+k$. Ca urmare, dacă la un moment dat cel mai mare număr de secvenţă primit de receptor este n, următorul număr de secvenţă primit va fi în intervalul $[n-k-1, n+k]$.

Să privim acum din perspectiva emiţătorului. Fie n cel mai mare număr de secvenţă trimis. Deoarece n a fost deja trimis, rezultă că toate pachetele până la $n-k$ inclusiv au fost deja confirmate. În momentul primei transmiteri a pachetului $n-k$, pachetele până la $n-2k$ inclusiv erau deja confirmate. Ca urmare, nici unul dintre pachetele cu numere de secvenţă 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 emiţător 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 secvenţă distincte sunt într-adevăr necesare. Considerăm două scenarii:

1. Emiţătorul transmite pachetele de la 1 la k, toate acestea ajung la receptor, dar toate confirmările se pierd. Emiţătorul retransmite pachetul 1, care ajunge la receptor.
2. Emiţătorul transmite pachetele de la 1 la k, acestea ajung la receptor, sunt confirmate şi confirmările ajung înapoi la emiţător. În continuare, emiţătorul transmite pachetele de la $k+1$ la $2k$, dar toate se pierd cu excepţia pachetului $2k$.

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

4.4. Controlul fluxului

Prin *controlul fluxului* (engl. *flow control*) se înţelege procesul (şi 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 destinaţia să le prelucreze.

În lipsa controlului fluxului, dacă sursa emite date mai rapid decât este capabilă destinaţia să le prelucreze, o parte din date se pierd. De remarcat că stocarea datelor într-o memorie tampon a destinaţiei nu rezolvă problema, ci doar permite destinaţiei 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 emiţător şi receptor este sigură (fără erori şi fără pierderi, duplicări sau inversiuni de pachete).

Forma cea mai simplă de control al fluxului este standardizarea unui debit fix de transmitere a datelor şi proiectarea tuturor componentelor sistemului de comunicaţie în aşa 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 informaţiei,

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

Dacă soluţia unui debit fix de transmisie nu este satisfăcătoare, este necesar un mecanism prin care receptorul să informeze emiţătorul asupra posibilităţii sale de preluare a datelor. Pentru aceasta este necesar un al doilea canal de comunicaţie, înapoi, dinspre receptor spre emiţător.

4.4.1. Cereri de suspendare şi de continuare

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

Astfel, receptorul este prevăzut cu o memorie tampon. Dacă memoria tampon a receptorului este aproape plină, receptorul trimite emiţătorului un mesaj prin care cere acestuia să suspende transmisia de date. Ulterior, când destinaţia consumă datele din memoria tampon a receptorului, receptorul cere emiţătorului 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 combinaţia *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 afişare şi, după ce citeşte datele afişate, va tasta *ctrl-Q* pentru continuarea transmisiei. Evident, cu acest mecanism de control al fluxului, caracterele cu codurile 17 şi 19 nu pot fi utilizate pentru a transmite informaţie utilă.

Acelaşi principiu, implementat puţin diferit, este mecanismul numit *hardware flow control*. În acest caz, semnalizarea de suspendare şi 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 şi până în momentul în care receptorul nu mai primeşte 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 emiţătorului când este pregătit să accepte următorul pachet. E-

mitător trimite un singr 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 înţelege un procedeu prin care printr-un acelaşi canal fizic de comunicaţie se stabilesc mai multe comunicaţii care decurg relativ independent una de alta. Serviciul oferit fiecărei comunicaţii este numit *canal logic*; fiecare comunicaţie ocupă deci câte un canal logic şi toate canalele logice sunt construite pe acelaşi canal fizic.

În § 3.3.3 şi § 3.6.2.3 am văzut mecanisme de multiplexare (în frecvenţă, 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 şiruri de biţi aparţinând fiecărui canal logic. Evident, intercalarea trebuie făcută în aşa fel încât receptorul să poată separa datele corespunzătoare fiecărui canal logic. De asemenea, emiţătorul trebuie să asigure o împărţire echitabilă a capacităţii 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 aparţin datele utile (fig. 4.11(b)).
- Se stabileşte o ordine de succesiune între canalele logice. Prin canalul fizic se transmite, pe rând, câte un pachet aparţinând 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 emiţătorul de la nivelul legăturii de date să trimită pachete vide în contul acelui canal (pentru a permite celorlalte canale logice să transmită pachete fără a încurca evidenţele receptorului).

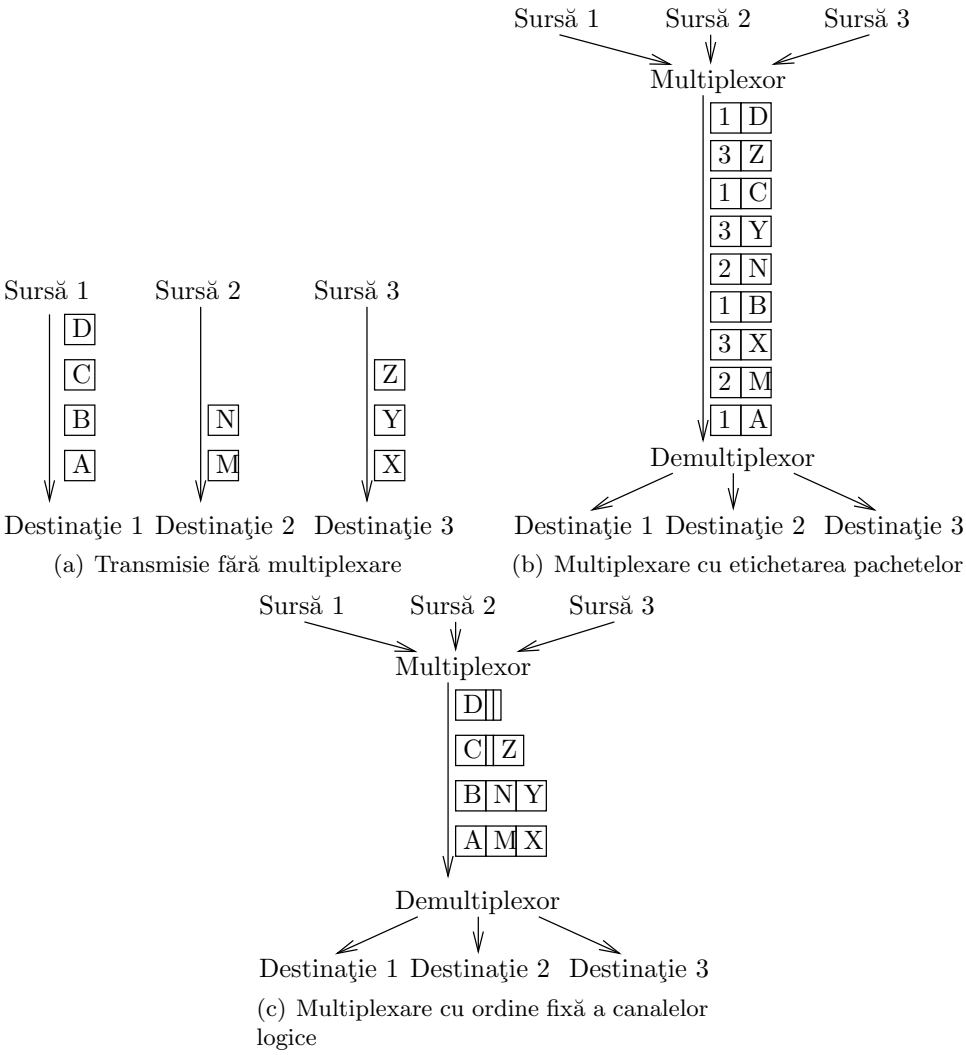


Figura 4.11: Funcţionarea mecanismelor de multiplexare în timp

Capitolul 5

Nivelul reţea şi nivelul transport

Dacă nişte dispozitive, relativ numeroase sau întinse pe distanţe 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 comunicaţii între dispozitive între care nu există o legătură fizică directă dar există legături *indirecte* prin intermediul unui şir de dispozitive legate fizic fiecare cu următorul.

O *reţea de comunicaţie* este un ansamblu de dispozitive care permit stabilirea de comunicaţii indirecte.

Într-o reţea de comunicaţie numim:

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

Notăm că în unele reţele există o distincţie netă între nodurile finale şi nodurile intermediare: de exemplu în reţeaua telefonică, aparatele telefonice

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

Unei reţele i se asociază un graf, construit astfel: fiecărui nod al reţelei i se asociază un vârf al grafului, iar fiecărei legături directe i se asociază o muchie (sau un arc, dacă legăturile sunt asimetrice). Reţeaua 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.

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

5.1. Retransmiterea datelor de către nodurile intermediare

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

Constructiv, într-un nod al unei reţele 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 şi recepţia datelor prin acea legătură. Aceasta este formată din modulul nivelului legăturii de date şi din modulul nivelului fizic.
- *Adaptarea spre aplicaţie*, pentru nodurile terminale, este o componentă ce realizează intermedierea între serviciile oferite direct de nivelul reţea şi nevoile aplicaţiilor ce se execută pe acel nod. Aceasta este, de principiu, modulul nivelului transport.
- *Modulul de reţea* este componenta care dirijează fluxul de date prin nod, fiind responsabil de alegerea vecinului căruia trebuie să-i fie transmise datele, precum şi de transmiterea efectivă a acestora către modulul de adaptare spre mediul fizic (în nodurile intermediare) sau, respectiv, către modulul de adaptare spre aplicaţie (în nodul destinaţie).

Nivelul reţea este ansamblul modulelor de reţea ale nodurilor reţelei.

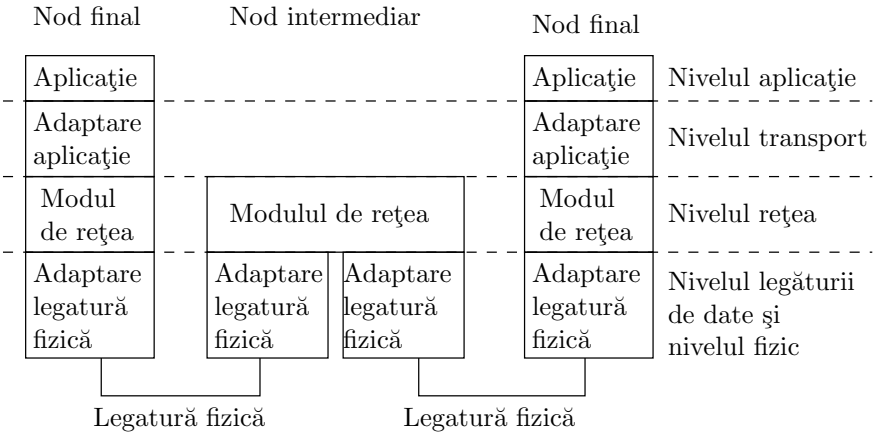


Figura 5.1: Modulele nodurilor unei reţele. Sunt figurate doar modulele din trei noduri, de-a lungul traseului datelor între două aplicaţii.

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

În lipsa unui protocol comun între modulele de reţea nu se poate stabili comunicaţia între oricare două noduri finale într-un mod uniform, fără ca aplicaţia client să trebuiască să aibe cunoştinţe despre nodurile intermediare. Din acest punct de vedere spunem că nivelul reţea, şi în special protocolul utilizat de nivelul reţea, este liantul întregii reţele.

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

- *datagrame:* Într-o reţea ce oferă serviciu tip datagrame, aplicaţia sursă crează o datagramă conţinând datele de transmis şi o paseze modulului reţea, specificând totodată adresa nodului destinaţie. Datagrama este transmisă din aproape în aproape până la nodul destinaţie, unde este pasată aplicaţiei (vezi § 5.1.1). De remarcat că două datagrame distincte generate de acelaşi nod sursă şi adresate aceluiaşi nod destinaţie sunt prelucrate, de către reţea, complet independent una de alta. Funcţionarea reţelelor ce oferă servicii de tip datagrame este similară sistemului de poştă (poşta obişnuită).
- *conexiune:* Într-o reţea ce oferă serviciu de tip conexiune, o aplicaţie ce doreşte să comunice cu o aplicaţie dintr-un alt nod începe prin a so-

licita modulului reţea deschiderea unei conexiuni către acel nod. Nivelul reţea informează nodul destinaţie despre cererea de deschidere a conexiunii şi, dacă aplicaţia destinaţie acceptă, conexiunea este deschisă şi nodul iniţiator este informat de acest lucru. După deschiderea conexiunii, unul sau ambele noduri (în funcţie de tipul conexiunii deschise, unidirecţională sau bidirecţională) poate transmite celuilalt pachete de date prin conexiunea deschisă. La terminarea comunicaţiei, una dintre aplicaţii solicită nivelului reţea închiderea conexiunii. Ca efect, nivelul reţea informează nodul partener cu privire la închiderea conexiunii şi eliberează resursele alocate conexiunii. Funcţionarea reţelelor ce oferă serviciu de tip conexiune este descrisă în § 5.1.2. Un model tipic de reţea ce oferă conexiuni este sistemul telefonic.

5.1.1. Retransmiterea în reţele bazate pe datagrame

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

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

Modulul de reţea al nodului primeşte o datagramă fie de la nivelul superior (dinspre aplicaţie), fie de la nivelul inferior (de pe o legătură directă). Modulul de reţea 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 destinaţie;
- să iniţieze 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 aşteptare. Se poate întâmpla ca memoria utilizabilă pentru coada de aşteptare 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 reţelei în acest caz sunt date în § 5.3.

5.1.2. Retransmiterea în reţele bazate pe conexiuni

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

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

Fiecare nod intermediar, pe rând, determină nodul următor prin care trebuie să treacă conexiunea şi-i trimite mai departe cererea de deschidere a conexiunii. Determinarea nodului următor se face la fel ca şi în cazul reţelelor 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 şi este acceptată de nodul destinaţie. Odată conexiunea deschisă, drumul corespunzător între cele două noduri finale este fixat pe toată durata conexiunii.

În faza de comunicare prorpiu-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 reţelele telefonice vechi, analogice).

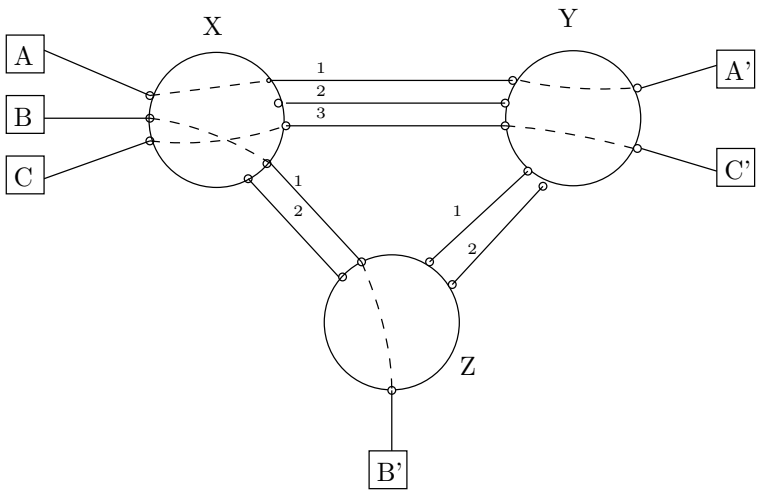


Figura 5.2: O reţea cu comutare de circuite fizice. Cercurile mari reprezintă nodurile intermediare, iar liniile punctate reprezintă interconectările mediilor fizice. De remarcat necesitatea mai multor legături fizice între câte două noduri.

- *Comutare de circuite virtuale*: Fiecare pachet ce soseşte printr-o legătură de date este memorat temporar şi 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 aceeaşi 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 distanţe mari, mediul fizic este scump, se utilizează mecanisme de multiplexare. Acestea pot lucra fie la nivel fizic (multiplexare în frecvenţă — § 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 ieşire, adică până când, în legătura fizică de ieşire, vine rândul la transmisie canalului logic prin care trebuie trimis pachetul.

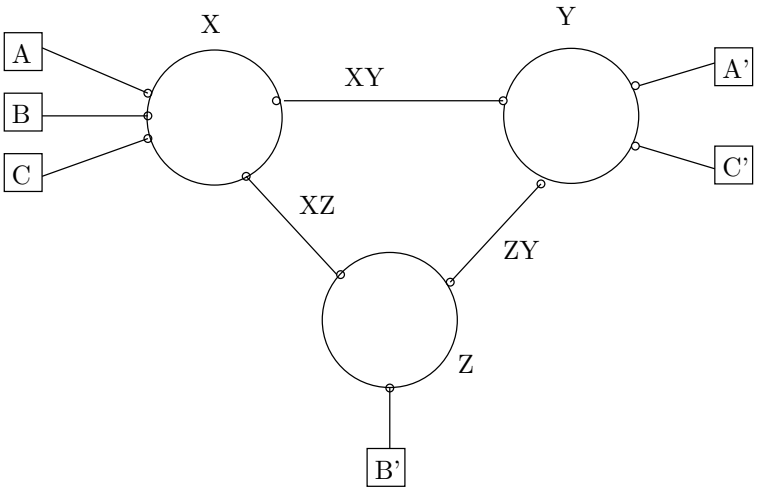


Figura 5.3: O reţea cu comutare de circuite virtuale. Desfăşurarea în timp a recepţiei şi transmitere mai departe a pachetelor, pentru nodul X, este prezentată în figura 5.4. Legăturile directe între nodurile intermediare utilizează multiplexare în timp.

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

Comutarea de circuite virtuale seamănă la prima vedere cu transmisia

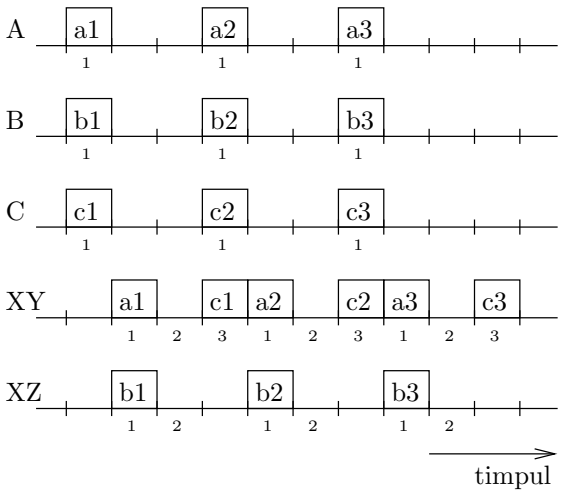


Figura 5.4: Desfăşurarea în timp a recepţiei şi a transmiterii mai departe a pachetelor, pentru nodul X din reţeaua din figura 5.3. XY şi XZ desemnează legăturile fizice între nodurile X şi Y, respectiv X şi Z. Numerele de sub axe marchează perioadele de timp alocate canalelor logice corespunzătoare.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

de datagrame. Diferenţa vine din felul în care un nod, care primeşte un pachet şi 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 funcţie de circuitul virtual căruia îi aparţine pachetul, informaţie dedusă din legătura de date prin care a intrat pachetul. Decizia se ia pe baza tabelei de circuite şi este identică pentru toate pachetele aparţinând aceluiaşi circuit. O urmare a acestui fapt este că defectarea oricărui nod sau oricărei legături de-a lungul unei conexiuni duce la închiderea forţată a conexiunii. În cazul reţelei bazate pe datagrame, decizia de dirijare se ia în funcţie de adresa destinaţie, conţinută în datagramă. Două datagrame între aceleaşi două staţii 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 reţelelor bazate pe datagrame), respectiv spre care dintre vecini să transmită cererea de iniţiere a unei conexiuni (în cazul reţelelor bazate pe conexiuni). Problema determinării acestui nod vecin se numeşte *problema dirijării.*

Rezolvarea problemei dirijării se bazează pe determinarea unui drum de cost minim, de la nodul sursă la nodul destinaţie al datagramei sau al conexiunii, în graful asociat reţelei de calculatoare.

Graful asociat reţelei de calculatoare este un graf ce are câte un vârf asociat fiecărui nod al reţelei şi câte o muchie asociată fiecărei legături directe între două noduri. Fiecărei muchii i se asociază un cost, existând următoarele posibilităţi pentru definirea costului:

- toate costurile egale;
- în funcţie de lungimea fizică a legăturii (cu cât o legătură este mai lungă, cu atât costul asociat este mai mare);
- în funcţie de capacitatea legăturii;
- în funcţie de încărcarea legăturii.

Remintim, din teoria grafelor, o proprietate importantă a drumurilor de cost minim: Dacă $v_0, v_1, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_k$ este un drum de cost minim de la v_0 la v_k, atunci $v_0, v_1, \ldots, v_{j-1}, v_j$ este un drum de cost minim de la v_0 la v_j şi $v_j, v_{j+1}, \ldots, v_k$ este un drum de cost minim de la v_j la v_k. De asemenea, dacă există cel puţin un drum de cost minim de la v_0 la v_k ce trece prin v_j, dacă $v_0, v_1, \ldots, v_{j-1}, v_j$ este un drum de cost minim de la v_0 la v_j şi $v_j, v_{j+1}, \ldots, v_k$ este un drum de cost minim de la v_j la v_k, atunci $v_0, v_1, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_k$ 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 consecinţă, 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 şi de v_0. Ca urmare, pentru a efectua retransmiterea datelor, fiecare nod v_j trebuie să cunoască doar, pentru fiecare destinaţie posibilă v_k, următorul vârf v_{j+1} de pe drumul optim spre acea destinaţie. Corespondenţa, pentru fiecare v_j, între destinaţia v_k şi 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 şi legăturile din reţea, în vederea obţinerii efective a grafului reţelei. După calculul drumurilor, este necesară distribuirea tabelelor de dirijare către toate nodurile reţelei.

Într-o reţea mică, centralizarea informaţiilor despre legături şi apoi distribuirea informaţiilor de dirijare către noduri se poate face manual, de către administratorul reţelei.

În reţelele mai mari, acest proces trebuie automatizat (total sau parţial). Deoarece nu este de dorit oprirea completă a reţelei oridecâteori 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 şi funcţionarea reţelei în acest timp să fie acceptabilă. Metodele principale de calcul pentru tabelele de dirijare sunt descrise în § 5.2.1 şi § 5.2.2.

În reţelele foarte mari, cum ar fi Internet-ul, centralizarea completă a datelor nu este rezonabilă; trebuie utilizaţi algoritmi de dirijare care să permită fiecărui nod efectuarea dirijării fără a necesita decât puţine informaţii şi doar despre o mică parte a reţelei. De asemenea, tabela de dirijare trebuie să aibă o reprezentare mai compactă decât câte un rând pentru fiecare nod destinaţie posibil. În astfel de cazuri se utilizează dirijarea ierarhică (§ 5.2.3).

Există şi metode ad-hoc de dirijare, utilizate în diverse situaţii mai deosebite, de exemplu dacă graful asociat reţelei de calculatoare are anumite particularităţi. Acestea vor fi studiate în § 5.2.4.

5.2.1. Calculul drumurilor cu informaţii complete despre graful reţelei

În cadrul acestei metode, fiecare nod al reţelei adună toate informaţiile despre graful asociat reţelei, după care calculează drumurile de la el la toate celelalte noduri.

Pentru ca fiecare nod să dispună în permanenţă de graful asociat reţelei de calculatoare, fiecare modificare a reţelei trebuie anunţată tuturor nodurilor. Pentru aceasta, fiecare nod testează periodic legăturile cu vecinii săi şi, oridecâteori constată o modificare, transmite o înştiinţare în toată reţeaua. Transmisia informaţiei respective se face astfel:

- Fiecare nod crează, periodic, un pachet ce conţine numele nodului, starea legăturilor cu vecinii (costurile actuale ale legăturilor), precum şi un *număr de secvenţă* (număr care tot creşte de la un astfel de pachet la următorul). Apoi transmite acest pachet tuturor vecinilor, printr-un protocol sigur (cu confirmare şi retransmitere).
- Fiecare nod ce primeşte un pachet descriind starea legăturilor verifică dacă este sau nu mai recent (adică cu număr de secvenţă mai mare) decât ultimul astfel de pachet primit de la acel nod. Dacă este mai recent, îl trimite tuturor vecinilor (mai puţin celui dinspre care a venit pachetul) şi actualizează reprezentarea proprie a grafului reţelei. Dacă pachetul este mai vechi, înseamnă că este o copie ce a sosit pe altă cale şi 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 — condiţie îndeplinită de graful asociat unei reţele de calculatoare — algoritmul cel mai eficient este algoritmul lui Dijkstra (algoritmul 5.1). Notând cu n numărul de vârfuri (noduri ale reţelei) şi cu m numărul de muchii (legături directe), complexitatea algoritmului lui Dijkstra este timp $O(m + n \log n)$ şi spaţiu $O(m + n)$. Calculul trebuie refăcut complet la fiecare modificare a grafului asociat reţelei de calculatoare.

5.2.2. Calculul drumurilor optime prin schimb de informaţii de distanţă

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 reţelei de calculatoare în aşa fel încât nici un nod să nu aibă nevoie de informaţii complete despre graf. Metoda se numeşte *cu vectori distanţă* deoarece prevede transmiterea, de la fiecare nod la vecinii săi direcţi, a unor vectori reprezentând distanţele de la nodul curent la toate celelalte noduri.

Algoritmul prevede că fiecare nod deţine o tabelă conţinând, pentru fiecare destinaţie posibilă, distanţa până la ea şi primul nod de pe drumul optim spre acea destinaţie. Iniţial, tabelul este iniţializat astfel: pentru vecinii direcţi, costul drumului este pus ca fiind costul legăturii directe spre acel nod, iar primul nod spre acea destinaţie este fixat chiar acel nod; pentru nodurile ce nu sunt vecini direcţi, costul este iniţializat cu infinit.

După initializare, nodurile recalculează periodic tabelele de distanţe. Pentru fiecare nod, calculul se face astfel: mai întâi, nodul cere vecinilor direcţi tabelele acestora. Apoi, pentru fiecare destinaţie posibilă, drumul optim este calculat ca fiind cel mai puţin costisitor dintre legătura directă şi drumurile prin fiecare dintre vecinii direcţi. Costul drumului printr-un vecin direct este calculat ca fiind costul legăturii dintre nodul curent şi vecinul considerat adunat cu costul, conform tabelei vecinului, al drumului de la vecinul respectiv la nodul destinaţie. De remarcat că, în calculul tabelei de distanţe a unui nod, nu se utilizează deloc tabela de distanţe a acelui nod de la iteraţia precedentă.

După câteva iteraţii ale buclei principale, algoritmul se stabilizează (converge), în sensul că tabelele calculate la fiecare iteraţie sunt identice cu cele calculate la iteraţia precedentă. Numărul de iteraţii 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 ($E \subseteq V \times V$)
$c : E \to [0, \infty)$ costurile asociate arcelor
$x_0 \in V$ vârful curent
ieşirea: $t : V \to \{(x_0, y) \in E\}$ 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 tabelei 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
$d_{x,j}$:=$\min(c_{x,j}, \min_{i\in N^{\text{out}}(x)})c_{x,i} + d_{i,j}$
$p_{x,i}$:= 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 iteraţii cel mult egal cu numărul maxim de muchii de-a lungul unui drum optim. Dacă se elimină o legătură directă sau creşte costul unei legături directe, tabelele de dirijare se stabilizează mult mai încet, aşa cum se vede în exemplul 5.2.

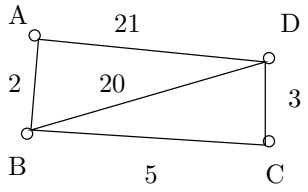


Figura 5.5: Reţeaua pentru exemplele 5.1 şi 5.2. Numerele reprezintă costurile asociate legăturilor directe.

Exemplul 5.1: Fie reţeaua din figura 5.5. Calculul tabelelor de dirijare, conform algoritmului 5.2, de la iniţializare până la stabilizare, duce la următoarele tabele:

- *Iniţializarea:* 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:

dest.	via	cost
B	B	2
C	–	∞
D	D	21

Nodul B:

dest.	via	cost
A	A	2
C	C	5
D	D	20

Nodul C:

dest.	via	cost
A	–	∞
B	B	5
D	D	3

Nodul D:

dest.	via	cost
A	A	21
B	B	20
C	C	3

- *Iteraţia 1:* Pentru fiecare destinaţie posibilă, se ia în considerare legătura directă (dacă există) şi rutele prin fiecare din vecinii direcţi. 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 şi 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 şi legătura prin B de cost 2+20=22; de notat că pentru legătura prin B se ia costul B→D raportat de B, calculat de către B la iniţializare.

Nodul A:

dest.	via	cost
B	B	2
C	B	7
D	D	21

Nodul B:

dest.	via	cost
A	A	2
C	C	5
D	C	8

Nodul C:

dest.	via	cost
A	B	7
B	B	5
D	D	3

Nodul D:

dest.	via	cost
A	A	21
B	C	8
C	C	3

- *Iteraţia 2:* Să urmărim ruta calculată de A către D. Sunt luate în considerare legătura directă de cost 21 şi legătura prin B a cărui cost este acum 2+8=10 întrucât se bazează pe costul legăturii B→D calculat de către B la iteraţia 1.

Nodul A:

dest.	via	cost
B	B	2
C	B	7
D	B	10

Nodul B:

dest.	via	cost
A	A	2
C	C	5
D	C	8

Nodul C:

dest.	via	cost
A	B	7
B	B	5
D	D	3

Nodul D:

dest.	via	cost
A	C	10
B	C	8
C	C	3

- Începând cu iteraţia 3, tabelele calculate sunt identice cu cele de la itareţia 2.

EXEMPLUL 5.2: Fie reţeaua din figura 5.5 şi fie tabelele de dirijare rezultate după stabilizarea algoritmului cu vectori distanţă (vezi exemplul 5.1). Să presupunem că legătura B–C cade, rezultând reţeaua din figura 5.6. Să urmărim evoluţia, în continuare, a tabelelor de dirijare.

La prima iteraţie, la recalcularea rutelor nodului B spre C şi 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

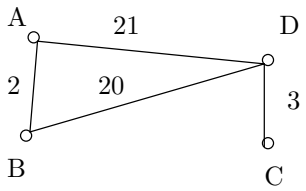


Figura 5.6: Reţeaua rezultată prin căderea legăturii B–C din reţeaua din figura 5.6.

că aceste rute nu mai sunt valide deoarece se bazau pe legătura B–C. La fel procedează şi nodul C, găsind că rutele optime spre A şi B trec prin D.

Nodul A:

dest.	via	cost
B	B	2
C	B	7
D	B	10

Nodul B:

dest.	via	cost
A	A	2
C	A	9
D	A	12

Nodul C:

dest.	via	cost
A	D	13
B	D	11
D	D	3

Nodul D:

dest.	via	cost
A	C	10
B	C	8
C	C	3

La următoarea iteraţie se vor modifica costurile rutelor din A spre C şi D şi din D spre A şi B:

Nodul A:

dest.	via	cost
B	B	2
C	B	11
D	B	14

Nodul B:

dest.	via	cost
A	A	2
C	A	9
D	A	10

Nodul C:

dest.	via	cost
A	D	13
B	D	11
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 iteraţie la alta, până când ajung la valorile rutelor reale optime. La a 3-a iteraţie de la căderea legăturii B–C, tabelele ajung în forma următoare:

Nodul A:

dest.	via	cost
B	B	2
C	B	11
D	B	14

Nodul B:

dest.	via	cost
A	A	2
C	A	13
D	A	14

Nodul C:

dest.	via	cost
A	D	19
B	D	17
D	D	3

Nodul D:

dest.	via	cost
A	C	16
B	C	14
C	C	3

Urmează, la a 4-a iteraţie, descoperirea de către D a rutelor reale spre A şi spre B:

Nodul A:

dest.	via	cost
B	B	2
C	B	15
D	B	18

Nodul B:

dest.	via	cost
A	A	2
C	A	13
D	A	14

Nodul C:

dest.	via	cost
A	D	19
B	D	17
D	D	3

Nodul D:

dest.	via	cost
A	A	21
B	B	20
C	C	3

Restul rutelor reale sunt descoperite şi mai târziu, stabilizarea tabelelor survenind abia la a 8-a iteraţie.

În general, numărul de iteratii după care se stabilizează tabelele după căderea sau creşterea costului unei legături poate fi cel mult egal cu raportul dintre cea mai mare creştere de cost între două noduri şi cel mai mic cost al unei legături directe. În cazul exemplului 5.2, costul drumului optim de la B la C creşte, prin căderea legăturii directe B–C, de la 5 la 23, o creştere de 18 unităţi. Costul cel mai mic al unei legături directe este 2 (legătura A–B). Ca urmare, stabilizarea tabelelor poate lua cel mult $\frac{18}{2} = 9$ iteraţii. În cazul în care căderea unei legături duce la deconectarea reţelei, acest lucru nu va fi detectat niciodată, numărul de iteraţii necesar fiind infinit.

Pentru a îmbunătăţi comportamentul în cazul căderii sau creşterii costului legăturilor, se poate modifica algoritmul astfel: tabelele vor ţine ruta completă spre destinaţie, iar la recalcularea rutelor, rutele ce trec de două ori prin acelaşi nod nu sunt luate în considerare.

Exemplul 5.3: Să reluăm reţeaua 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:

dest.	ruta	cost
B	B	2
C	B,C	7
D	B,C,D	10

Nodul B:

dest.	ruta	cost
A	A	2
C	C	5
D	C,D	8

Nodul C:

dest.	ruta	cost
A	B,A	7
B	B	5
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:

dest.	ruta	cost
B	B	2
C	B,C	7
D	B,C,D	10

  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	C,B,A	10
B	C,B	8
C	C	3

- *Iteraţia 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

- *Iteraţia 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 reţelele foarte mari, unde este imposibil ca fiecare nod să aibă informaţii despre toate celelalte noduri. Exemple clasice de astfel de reţele sunt Internet-ul şi reţeaua telefonică.

Ideea dirijării ierarhice este ca reţeaua să fie împărţită în *subreţele*. Subreţelele alcătuiesc o ierarhie arborescentă: o subreţea rădăcină (considerată pe nivelul 0), câteva subreţele subordonate ei (nivelul 1), subreţele subordonate câte unei subreţele de pe nivelul 1 (alcătuind nivelul 2), ş. a. m. d. Fiecare nod are informaţii de dirijare:

- către nodurile din subreţeaua 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).

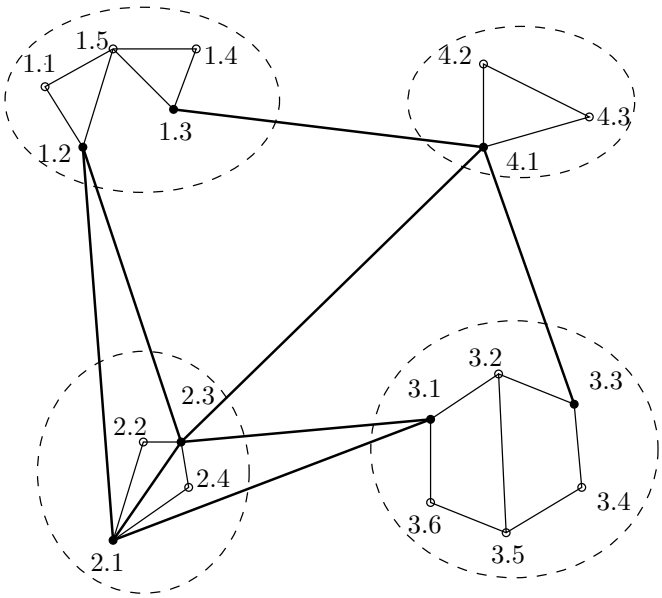
Exemplul 5.4: În figura 5.7 este reprezentată o reţea cu dirijare ierarhică pe două nivele. Reţeaua este formată dintr-o subreţea rădăcină şi patru subreţele subordonate ei.

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.

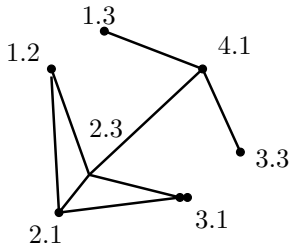
Să presupunem că nodul 1.1 are de trimis un pachet către nodul 3.4. Dirijarea decurge astfel:

- Nodul 1.1 determină că destinatia 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 reţele bazate pe datagrame. Inundarea constă în a trimite cópii ale unei datagrame prin toate legăturile directe, cu excepţia celei prin care a intrat datagrama.

Inundarea garantează că, dacă destinaţia este accesibilă şi nodurile nu sunt prea încărcate (astfel încât să se sacrifice datagrame din lipsă de spaţiu de memorare), datagrama ajunge la destinaţie. Ca avantaj faţă de alte metode, inundarea nu necesită ca nodurile să adune nici un fel de informaţie despre reţea.

Pe de altă parte, inundarea face ca fiecare datagramă să ajungă la fiecare nod al reţelei, nu doar la destinatarul dorit. Ca urmare, la fiecare nod ajung toate pachetele care circulă prin reţea. La un număr de noduri mai mare de câteva zeci, metoda inundării generează prea mult trafic pentru a fi în general acceptabilă.

Dacă graful reţelei 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 opreşte la frunze. De notat însă că o reţea al cărei graf ataşat este un arbore este extrem de vulnerabilă la pene: defectarea oricărui nod intern duce la deconectarea reţelei.

Dacă însă graful reţelei conţine cicluri, atunci o datagramă, o dată ajunsă într-un ciclu, ciclează la infinit. Pentru ca inundarea să fie utilizabilă în reţele cu cicluri, trebuie făcută o modificare pentru prevenirea ciclării infinite.

O posibilă soluţie — utilizată şi 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 şi de lungime dată.

O altă soluţie, cu avantajul suplimentar că asigură ca fiecare pachet să ajungă într-un singur exemplar la destinaţie, este ca fiecare nod al reţelei să identifice (de exemplu, prin menţinerea unor numere de secvenţă) duplicatele unui pachet şi să trimită mai departe un pachet doar la prima lui sosire.

Inundarea se utilizează în reţelele Ethernet. Graful unei reţelele Ethernet trebuie să fie întotdeauna un arbore.

5.2.4.2. Învăţarea rutelor din adresele sursă ale pachetelor

O metodă simplă de construcţie 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 destinaţia S prevăzând ca următor nod pe V. Regulile astfel introduse trebuie să aibă valabilitate limitată în timp — altfel apar probleme la modificarea legăturilor din reţea. 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 reţelele Ethernet.

5.2.5. Metode de difuziune

Ne vom ocupa în continuare de metodele de dirijare aplicabile în vederea trimiterii cópiilor unei datagrame spre mai multe destinaţii. Distingem două posibile cerinţe, *difuziune completă* (engl. *broadcast*) — trimiterea spre toate nodurile unei reţele — şi *difuziune selectivă* (engl. *multicast*) — trimiterea datagramei spre o submulţime dată a multimii 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 parţial (preferabil de cost minim) de acoperire a vârfurilor destinatie, iar apoi să se aplice metoda inundării în acest arbore. Această metodă este utilizabilă atât pentru difuzare completă cât şi pentru difuzare parţială. Datorită necesităţii calculului arborelui parţial, este favorabilă în cazul în care trebuie trimise multe datagrame aceleiaşi mulţimi de destinatari.

Descriem şi o a treia posibilitate, utilă în special în situaţia în care destinatarii sunt puţini şi nu se trimit multe datagrame aceleiaşi mulţimi de destinatari. Metoda constă în a trimite în datagramă multimea adreselor destinaţie. Fiecare nod determină legătura de ieşire pentru fiecare destinaţie din lista din datagramă. Apoi trimite câte o datagramă pe fiecare legătură directă ce apare pe ruta spre cel puţin una dintre destinaţii. Datagrama trimisă prin fiecare legătură directă va avea în lista de destinaţii 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 destinaţie, însă, cât timp drumul a două sau mai multe datagrame este comun, datagramele călătoresc reunite într-o singură datagramă cu mai multe adrese destinaţie.

5.3. Funcţionarea la trafic ridicat

Până aici am studiat comportamentul unei reţele doar pentru cazul în care debitul fluxului de date care intră într-un nod nu depăşeşte niciodată nici capacitatea legăturilor prin care trebuie trimis mai departe, nici capacitatea modulului de reţea de-a efectua prelucrările necesare. Dacă debitul cu care intră pachete într-un nod depăşeşte 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 ieşire. Un efect imediat este creşterea timpului de propagare, din cauza staţionării pachetelor în coada de aşteptare. Dacă excesul de debit de intrare se păstrează mai mult timp, coada creşte până când memoria alocabilă cozii de aşteptare 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, micşorarea debitului de intrare. De notat că reducerea şi, 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 depăşită sunt următoarele:

- Într-o reţea aglomerată, pachetele sau datagramele întârzie mult sau chiar se pierd, lucru care poate declanşa retrimiteri intempestive de pachete, ducând la aglomerare şi mai mare a reţelei şi la performanţe şi mai scăzute. O astfel de situatie, de degradare suplimentară a performanţelor în urma creşterii încărcării, se numeşte *congestie* şi trebuie evitată sau, cel puţin, ţinută sub control.
- Capacitatea disponibilă a reţelei trebuie împărţită în mod echitabil între utilzatori.
- Diferite aplicaţii au diferite priorităţi cu privire la caracteristicile necesare ale serviciului oferit de reţea. Reacţia unei reţele aglomerate trebuie să ţină cont de aceste priorităţi. De exemplu, un nod supraaglomerat poate să fie nevoit să arunce o parte dintre datagramele aflate în tranzit. Dacă datagramele aparţin unei aplicaţii de transfer de fişiere, 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 şi să trebuiască retransmise toate). Dimpotrivă, dacă datagramele aparţin unei aplicaţii de tip videoconferinţă,

este preferabil să fie aruncate datagramele mai vechi.

De notat că, adesea, rezolvarea problemelor de mai sus necesită o colaborare între nivelul reţea şi nivelele superioare.

5.3.1. Alegerea pachetelor de transmis

Considerăm un ruter ale cărui linii de ieşire sunt utilizate la maximul capacităţii. 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 menţine o singură coadă şi de a accepta un pachet proaspăt sosit dacă are loc în coadă şi de a-l distruge dacă nu are loc. Atunci când debitul de intrare este mare, soluţia duce la a accepta primul pachet ce soseşte după eliberarea unei poziţii în coadă. Ca urmare, un emiţător care produce multe pachete este avantajat faţă de un emiţător care produce puţine pachete.

O distribuire mai echitabilă a capacităţii 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 fiecare linie de intrare sau, după caz, circuit virtual obţine trimiterea aceluiaşi număr de pachete în unitatea de timp. Metoda se numeşte *aşteptare 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 biţi preluaţi. Pentru aceasta, se poate asocia fiecărei cozi numărul total de biţi ai pachetelor preluate din acea coadă şi retransmise mai departe. De fiecare dată nodul intermediar extrage următorul pachet din coada cu cel mai mic număr de biţi transmişi.

Pe lângă posibilitatea de a oferi intrărilor transmiterea aceluiaşi număr de biţi sau de pachete, se poate oferi numere de biţi 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 biţi preluaţi şi retransmişi faţă de o legătură secundară. Metoda se numeşte *aşteptare echitabilă ponderată* (engl. *weighted fair queueing*).

În afară de metodele de aşteptare echitabilă — eventual, împreună cu ele — se poate pune în aplicare şi un sistem de priorităţi. Astfel, fiecărui pachet i se poate asocia un nivel de prioritate: pachetele pentru aplicaţii în timp real şi pachetelor asociate sesiunilor interactive li se asociază nivele de prioritate mai ridicate, iar aplicaţiilor care transferă fişiere mari li se asociază nivele de prioritate coborâtă. Într-un ruter, fiecărui nivel de prioritate i se asociază o coadă separată, cu spaţiu de memorare rezervat. Atunci când linia de ieşire este liberă şi ruterul trebuie să decidă care este următorul pachet, examinează

cozile în ordine descrescătoare a nivelelor de prioritate până găseşte o coadă nevidă. Pachetul următor ce va fi transmis este extras din prima coadă nevidă.

Dacă metoda priorităţilor este combinată cu aşteptarea echitabilă, fiecărui nivel i se asociază un set de cozi, în interiorul setului funcţionând regulile de la aşteptarea echitabilă. Următorul pachet trimis este extras din setul de cozi cel mai prioritar în care există cel putin o coadă nevidă.

5.3.2. Controlul congestiei

Prin *congestie* se înţelege scăderea debitului traficului util în reţea în situaţia în care cererea de trafic printr-o legătură sau printr-un nod depăşeşte capacitatea acesteia. Scăderea debitului util, în opoziţie cu limitarea traficului la capacitatea legăturii sau nodului respectiv, este datorată unei funcţionări defectuoase a reţelei, în special datorită pierderii şi retransmiterii unui număr mare de pachete.

Ca principiu general, este bine ca, dacă reţeaua este foarte încărcată, nodurile terminale să încerce să reducă frecvenţa şi mărimea pachetelor transmise. Evident, pentru acest lucru, este necesar un mecanism care să semnaleze nodurilor finale asupra prezenţei sau iminenţei congestiei.

Descriem în continuare, pe scurt, mecanisme utilizate pentru semnalarea congestiei, precum şi mecanismele prin care nodurile finale pot reacţiona la astfel de semnale.

Prima posibilitate de semnalizare a congestiei este ca, atunci când un nod intermediar este încărcat la limita capacităţii 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 reţea 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 destinaţiei fiecărui pachet de date faptul că reţeaua este încărcată. Această semnalizare este mai uşor de făcut întrucât poate fi transmisă odată cu pachetul de date, sub forma unui bit din antetul fiecărui pachet. Dezavantajul, faţă de metoda precedentă, este că nu semnalizează sursei traficului, ci destinaţiei; rămâne deci necesar de elaborat un protocol de informare a sursei. Informarea sursei poate fi făcută simplu dacă între sursă şi destinaţie se utilizează un protocol de control al fluxului: în cazul în care destinaţiei îi este semnalizat că reţeaua este congestionată, destinaţia cere sursei, prin intermediul protocolului de control al fluxului, să reducă debitul transmisiei. Metoda semnalizării destinaţiei 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 reţeaua este încărcată constă în însăşi pierderea pachetelor. Pierderea poate fi observată de nodul sursă prin aceea că nu primeşte confirmări (în cazul utilizării unui protocol cu confirmare şi retransmitere, § 4.3) sau răspuns la mesajele trimise (în cazul unei aplicaţii care trimite o datagramă de cerere şi aşteaptă 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 puţin 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 şi retransmitere .

Indiferent de metoda de semnalizare utilizată, o implementare simplă riscă să ducă la oscilaţii: dacă un nod intermediar ajunge congestionat, semnalizează tuturor nodurilor terminale ale legăturilor stabilite prin el despre congestie. Reacţia este diminuarea traficului prin toate legăturile şi ca urmare scăderea traficului mult sub maximul admis. După un timp, nodurile terminale vor creşte din nou traficul, până la congestionarea, din nou, a nodului intermediar considerat. Soluţionarea problemei oscilaţiilor se face punând nodul intermediar să trimită semnale că este supraîncărcat cu puţin înainte de-a ajunge la limita capacităţii sale şi de-a alege aleator legăturile cărora li se semnalizează încărcarea.

5.3.3. Formarea (limitarea) traficului

Prin *formarea traficului* se înţeleg metode de uniformizare a debitului unui flux de date. Mecanismele de limitare pot fi plasate în nodul sursă sau într-un ruter şi pot acţiona asupra fluxului de pachete provenit de la un anumit nod sursă, asupra fluxulul între două staţii 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 numeşte *găleată găurită*, prin analogie cu următorul mecanism fizic: într-o găleată (reprezentând coada de aşteptare a ruterului) se toarnă apă (reprezentând pachetele unui flux). Găleata are o gaură prin care curge apă (pachete ce sunt preluate din coadă şi retransmise de către ruter). Debitul apei care curge (debitul fluxului de ieşire) este constant atât 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 ţine evidenţa capacităţii nefolosite (diferenţa dintre debitul maxim acceptat şi debitul fluxului) şi 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 depăşi o valoare maximă. Dacă există un pachet în coadă şi numărul de jetoane este mai mare sau egal cu numărul de biţi ai pachetului, pachetul este preluat din coadă şi retransmis, iar numărul de jetoane asociat cozii este scăzut cu o valoare egală cu numărul de biţi ai pachetului. Mecanismul se numeşte *găleata cu jeton*.

5.3.4. Rezervarea resurselor

Pentru a avea, în mod garantat, o anumită capacitate şi un anumit timp de propagare oferite unui flux de date, este necesar să fie rezervate fluxului resursele necesare — capacitatea de prelucrare în noduri şi capacitatea de transfer prin legăturile directe.

Rezervarea resurselor se poate face doar în reţele ce oferă servicii de tip conexiune — utilizarea rezervării în reţele cu datagrame duce la necesitatea implementării unui mecanism de ţinerea evidenţei fluxurilor de datagrame similar celui din reţelele cu circuite virtuale.

La deschiderea conexiunii, în timpul stabilirii rutei conexiunii se stabileşte şi capacitatea pe care o va garanta conexiunea şi fiecare nod se asigură că dispune de capacităţile necesare (că suma capacităţilor alocate conexiunilor ce partajează o legătură directă nu depăşeşte 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 aplicaţiilor în timp real. Reţeaua telefonică utilizează astfel de mecanisme.

Avantajul unui debit garantat se plăteşte 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, reţeaua nu este folosită la maximul de capacitate.

În acest caz, valorificarea capacităţii rămase, cu păstrarea capacităţii garantate prin rezervarea resurselor, se poate face astfel: Datelor ce aparţin conexiunilor cu trafic garantat li se asociază un nivel de prioritate ridicat.

Sunt permise şi 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 priorităţi. În acest fel, fluxurile ce au rezervat resurse au capacitate garantată, iar restul datelor sunt transmise, fără vreo garanţie, dacă mai rămân resurse şi pentru ele.

5.4. Nivelul transport

Rolul nivelului transport este de-a face o adaptare între serviciile oferite de nivelul reţea şi nevoile aplicaţiilor. Funcţiile îndeplinite de nivelul transport sunt similare cu unele dintre funcţiile nivelului legăturii de date:

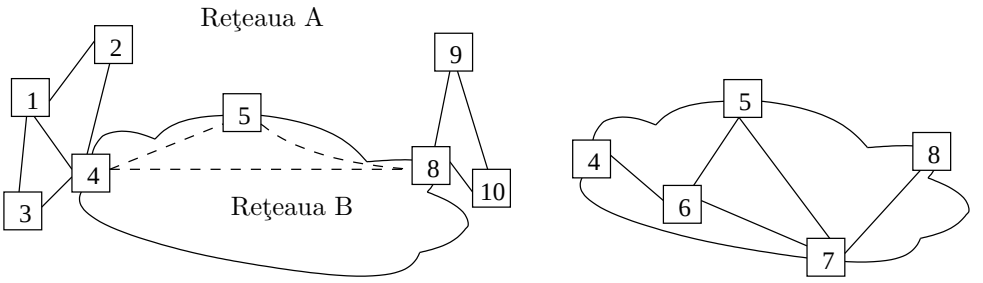
- *Transport sigur*: O reţea congestionată se poate să distrugă pachete din lipsă de spaţiu de memorare. În plus, într-o reţea ce oferă transport de datagrame este posibil ca două datagrame să ajungă la destinaţie în ordine inversă faţă de cea în care au fost emise, iar în anumite cazuri este posibil ca o datagramă să ajungă în mai multe exemplare la destinaţie (de exemplu dacă se utilizează dirijare prin inundare şi reţeaua nu este un arbore). Metodele bazate pe confirmări şi retransmiteri (vezi § 4.3) pot fi utilizate, cu mici modificări, pentru a asigura transport sigur la nivelul transport. O diferenţă importantă faţă de transportul sigur la nivelul legăturii de date este că nivelul reţea poate inversa ordinea unor datagrame, în vreme ce nivelul fizic nu inversează pachete. O altă diferenţă este că la nivelul reţea 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 viaţă a unei datagrame în reţea, iar pe de altă parte algoritmul de confirmare şi retransmitere trebuie să se aştepte să primească astfel de datagrame întârziate şi să nu le confunde cu datagrame noi — aceasta din urmă înseamnă că spaţiul numerelor de secvenţă trebuie să fie suficient de mare.
- *Controlul fluxului*: Controlul fluxului are acelaşi rol şi se implementează în acelaşi mod la nivelul transport ca şi la nivelul legăturii de date.
- *Multiplexarea*: Multiplexarea poate fi utilă în mai multe scopuri: pentru a permite mai multor aplicaţii care se execută pe acelaşi calculator să comunice independent una de alta şi pentru a permite unei perechi de aplicaţii care comunică să-şi transmită independent mai multe fluxuri de date.

5.5. Interconectarea reţelelor

Probleme privind interconectarea reţelelor se pun în situaţia în care se doreşte ca două sau mai multe reţele ce nu pot funcţiona unitar ca o singură reţea să ofere totuşi servicii de comunicare similare unei reţele unitare.

Motivele de existenţă a reţelelor distincte pot fi: reţele ce utilizează protocoale diferite la nivel reţea, reţele ce utilizează acelaşi protocol, dar există suprapuneri între adresele alocate în aceste reţele, dorinţa unor administratori de-a nu divulga informatii despre legăturile din reţea sau de-a filtra pachetele care intră sau ies şi altele.

Problema interconectării este complexă şi necesită solutii ad-hoc, adaptate nevoilor de interoperabilitate şi particularităţilor reţelelor de interconectat. Există însă câteva metode generale, dintre care vom analiza aici metoda tunelării.



(a) Reţeaua A, având legături directe proprii (reprezentate prin linii continue) şi legături realizate ca tunele prin reţeaua B

(b) Reţeaua B

Figura 5.8: Legături prin tunel. O parte dintre legăturile directe din reţeaua A, figurate cu linie punctată, sunt obţinute apelând la serviciile reţelei B. Legătura 4–5 apare ca legătură directă pentru reţeaua A, dar este construită de reţeaua B prin intermediul nodului 6.

Un *tunel* este o legătură, realizată prin intermediul unei reţele, care este utilizată de o altă reţea ca şi când ar fi o legătură directă (vezi fig. 5.8). Pachetele celei de-a doua reţele, incluzând antetele specifice acesteia, sunt transportate ca date utile printr-o conexiune sau, după caz, prin datagrame ale primei reţele.

Capitolul 6

Metode şi protocoale criptografice

Vom studia în acest capitol cum se poate proteja comunicaţia dintre două entităţi contra acţiunilor unui terţ, numit *adversar* sau *intrus*, care interceptează sau alterează comunicaţia între ele. Protecţia comunicaţiei împotriva acţiunilor unui adversar se numeşte *securizarea* comunicaţiei.

Adversarul poate fi:

- *adversar pasiv*, care doar interceptează mesajele transmise;
- *adversar activ*, care şi interceptează şi modifică mesajele.

Protecţia comunicaţiei faţă de acţiunile adversarului cuprinde:

Asigurarea confidenţialităţii are ca obiectiv să impiedice un adversar pasiv să înţeleagă un mesaj interceptat sau să extragă vreo informaţie din el.

Verificarea autenticităţii mesajelor, numită şi *autentificarea mesajelor*, are ca obiectiv detectarea, de către receptor, a *falsurilor*, adică a mesajelor create sau modificate de un adversar activ. Verificarea autenticităţii mesajelor se aseamănă cu detectarea erorilor. Spre deosebire însă de detectarea erorilor, unde modificările produse de mediul de transmisie sunt aleatoare, la verificarea autenticităţii mesajelor avem un adversar care încearcă în mod deliberat să producă modificări nedetectabile.

Asigurarea non-repudiabilităţii mesajelor are ca obiectiv să permită receptorului să dovedească autenticitatea unui mesaj în faţa unui terţ, altfel spus, emiţătorul să nu poată nega faptul că a transmis un anumit mesaj. Asigurarea non-repudiabilităţii este similară cu autentificarea mesajelor, dar în plus trebuie să nu permită nici măcar receptorului să creeze un mesaj care să pară autentic.

Verificarea prospeţimii 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 şi apoi să transmită băncii multiple copii ale ordinului de transfer de bani. În lipsa verificării prospeţimii, banca va efectua de mai multe ori transferul de bani. Verificarea autenticităţii mesajelor, singură, nu rezolvă problema, deoarece fiecare copie este identică cu originalul şi, ca atare, este autentică.

Autentificarea entităţilor are ca obiectiv verificarea, de către o entitate, a identităţii entităţii cu care comunică. Mai exact, există un server şi unul sau mai mulţi clienţi legitimi care deschid conexiuni către server. Modelul adversarului, în acest caz, este puţin diferit: adversarul poate să deschidă o conexiune spre server şi să încerce să se dea drept un client legitim. Eventual, adversarul poate să intercepteze comunicaţiile clienţilor legitimi, pentru a obţine informaţii în vederea păcălirii serverului, dar nu poate altera comunicaţia printr-o conexiune deschisă de altcineva. În prezenţa unui adversar activ, autentificarea entităţilor nu este prea utilă, deoarece adversarul poate să lase protocolul de autentificare să se desfăşoare normal şi apoi să trimită orice în numele clientului. În prezenţa unui adversar activ, este mai degrabă necesar un mecanism de *stabilirea cheii* (vezi mai jos).

Stabilirea cheii are ca obiectiv obţinerea, de către partenerii de comunicaţie legitimi, a unui şir de biţi, numit *cheie*, ce urmează a fi utilizată la asigurarea confidenţialităţii şi la verificarea autenticităţii mesajelor. Cheia obţinută 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 integrităţii mesajelor* — verificarea de către receptor că mesajul este identic cu cel emis de emiţător (că nu a fost modificat pe traseu) — şi a *autentificării sursei* mesajului — verificarea de către receptor a identităţii autorului unui mesajului. Cele două operaţii — verificarea integrităţii şi 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 integrităţii), mesajul poate fi văzut ca un mesaj produs de adversar şi 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 confidenţialităţii

6.1.1. Introducere

Problema asigurării confidenţialităţii unui mesaj constă în a transmite informaţii în aşa fel încât doar destinatarul dorit să le poată obţine; un adversar care ar intercepta comunicaţia nu trebuie să fie capabil să obţină informaţia transmisă.

Formal, presupunem că emiţătorul are un mesaj de transmis, numit *text clar* (engl. *plaintext*). Emiţătorul va genera, printr-un algoritm, plecând de la textul clar, un aşa-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 cunoaşte anumite detalii ale algoritmului aplicat de emiţător, trebuie să nu fie capabil să reconstituie textul clar. Operaţia prin care emiţătorul transformă textul clar în text cifrat se numeşte *criptare* sau, uneori, *cifrare* (engl. *encryption*). Operaţia prin care receptorul obţine textul clar din textul cifrat se numeşte *decriptare* sau *descifrare* (engl. *decryption*). Împreună, algoritmii de criptare şi decriptare constituie un *cifru*.

Pentru a formaliza notaţiile, vom nota cu T mulţimea mesajelor posibile de transmis; fiecare text clar posibil este un element $t \in T$. Criptarea este o funcţie $c : T \to M$, unde M este mulţimea textelor cifrate posibile. $m = c(t)$ este textul cifrat corespunzător textului clar t. Textul cifrat este trimis pe canalul nesigur şi este presupus accesibil adversarului. Decriptarea o vom nota cu d, unde $d : M \to T$.

Spunem că (c, d) formează o pereche criptare-decriptare dacă îndeplinesc simultan condiţiile:

- orice text cifrat poate fi decriptat corect prin d, adică $d \circ c = 1_T$;
- un adversar care cunoaşte textul cifrat $m = c(t)$ dar nu cunoaşte c sau d nu poate deduce t sau afla informaţii despre t.

În practică, este necesar ca producerea unei perechi de funcţii (c, d) să fie uşor de făcut, inclusiv de către persoane fără pregătire deosebită. Acest lucru este necesar deoarece dacă perechea (c, d) utilizată de două entităţi care comunică este aflată, sau se bănuieşte 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 şi informatică să poată utiliza singure metode criptografice.

Pentru acest scop, algoritmii de criptare şi decriptare sunt făcuţi să primească, pe lângă textul clar şi respectiv textul cifrat, încă un argument

numit *cheie*. Fiecare valoare a cheii produce o pereche criptare-decriptare distinctă. Cheia se presupune a fi uşor de generat la nevoie.

Mulţimea cheilor posibile se numeşte *spaţiul cheilor* şi o vom nota în continuare cu K. Funcţiile de criptare şi decriptare sunt de forma $c : T \times K \to M$ şi respectiv $d : M \times K \to T$. Cheia este scrisă ca parametru. Pentru o valoare fixată a cheii $k \in K$, criptarea devine $c_k : T \to M$, iar decriptarea $d_k : M \to T$, cu $d_k \circ c_k = 1_T$. Pentru fiecare $k \in K$, (c_k, d_k) formează o pereche criptare-decriptare.

Algoritmii propriu-zişi, adică funcţiile $c : T \times K \to M$ şi $d : M \times K \to T$, se presupune că sunt cunoscuţi adversarului; dacă merită să puteţi schimba cheia, înseamnă că deja aveţi îndoieli privitoare la cât de secret puteţi ţine algoritmul faţă de adversar... Ca urmare, pentru o aplicaţie, un algoritm public nu este mai nesigur decât un algoritm „secret“, necunoscut publicului dar posibil cunoscut adversarului. Un algoritm foarte cunoscut, dar fără vulnerabilităţi cunoscute, este preferabil faţă de un algoritm „secret“ deoarece există şanse mult mai mari ca autorul şi utilizatorul aplicaţiei să afle despre vulnerabilităţi înainte ca vulnerabilităţile să fie exploatate împotriva lor.

Adesea avem $T = M$; în acest caz c_k este o funcţie bijectivă (o permutare pe T). În aceste condiţii, uneori rolurile funcţiilor c şi d pot fi interschimbate, adică d_k să se folosească ca funcţie de criptare şi c_k pentru decriptare.

Exemplul 6.1 (*Substituţia monoalfabetică*): Considerăm un alfabet (finit) S şi notăm cu n numărul de litere ($n = |S|$). De exemplu,

$$S = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots, \mathrm{z}\};$$

în acest caz $n = 26$. Textele clare sunt şiruri de litere din alfabet: $T = S^*$. Mulţimea textelor cifrate este identică cu mulţimea textelor clare: $M = T$. Cheile posibile sunt permutările lui S; $|K| = n!$. Pentru un text clar $p = (s_1, s_2, \ldots, s_l)$, textul cifrat este

$$c_k(p) = (k(s_1), k(s_2), \ldots, k(s_l)).$$

Decriptarea se calculează

$$d_k((m_1, m_2, \ldots, m_l)) = (k^{-1}(m_1), k^{-1}(m_2), \ldots, k^{-1}(m_l)).$$

Criptarea şi decriptarea sunt simplu de executat, chiar şi manual. Cheile sunt uşor de reprezentat. Dacă alfabetul are o ordine cunoscută,

reprezentarea cheii poate consta în înşiruirea literelor în ordinea dată de permutare. De exemplu

qwertyuiopasdfghjklzxcvbnm

înseamnă $k(\mathrm{a}) = \mathrm{q}$, $k(\mathrm{b}) = \mathrm{w}$, etc. Cu această cheie, cuvântul „criptic“ devine, prin criptare, „ekohzoe“.

Să examinăm puţin siguranţa. Să presupunem că un adversar încearcă decriptarea textului cifrat cu fiecare cheie posibilă. O astfel de încercare se numeşte *atac prin forţă brută*. Să mai presupunem că adversarul reuşeşte 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 româna, anumite litere (de exemplu *e*, *a*, *t*, *s*) apar mai frecvent decât altele. Ca urmare, permutările primelor prin funcţia k vor apare în textul cifrat cu frecvenţă 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 frecvenţă normală de apariţie este apropiată de frecvenţa de apariţie a literei considerate în textul cifrat. În acest fel, numărul de încercări se reduce considerabil, astfel încât un astfel de cifru poate fi spart uşor în câteva minute.

Exemplul 6.2 (*Cifrul Vernam, numit şi cheia acoperitoare, engl. One time pad*): La acest cifru, $T = \{t \in \{0,1\}^* : |t| \leq n\}$ (mulţimea şirurilor de biţi de lungime mai mică sau egală cu un $n \in \mathbb{N}$ fixat), $M = T$ şi $K = \{0,1\}^n$. Funcţia de criptare este

$$c_k(t_1, t_2, \ldots, t_l) = (t_1 \oplus k_1, t_2 \oplus k_2, \ldots, t_l \oplus k_l),$$

unde $\oplus$ este operaţia *sau exclusiv*.

Decriptarea coincide cu criptarea, $d_k = c_k$.

Din punctul de vedere al siguranţei, 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 recepţionat.

Criptarea cu cheie acoperitoare este dificil de utilizat practic deoarece necesită o cheie la fel de lungă ca şi mesajul de transmis şi, în plus, cheia nu poate fi refolosită (dacă se transmit două mesaje folosind aceeaşi cheie, se pierde siguranţa metodei).

6.1.2. Refolosirea cheilor

Până aici am considerat problema criptării unui singur mesaj. Utilizarea aceleiaşi chei pentru mai multe mesaje aduce adversarului noi posibilităţi de acţiune:

1. Două mesaje identice vor fi criptate identic; adversarul poate detecta astfel repetarea unui mesaj.
2. Anumite informaţii transmise criptat la un moment dat pot deveni publice ulterior. Adversarul poate obţine 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 informaţiilor aduse de astfel de perechi text clar, text cifrat, se numeşte *atac cu text clar cunoscut*.
3. În anumite cazuri, adversarul poate determina emiţătorul să trimită mesaje conţinând părţi generate de adversar. Acest lucru poate ajuta mult tentativelor de spargere de la punctul precedent. Atacul se numeşte *cu text clar ales*. De asemenea, ţinând cont şi de posibilităţile de la punctul 1, dacă adversarul bănuieşte textul clar al unui mesaj, poate să încerce să-şi confirme sau infirme bănuiala.
4. Anumite cifruri, de exemplu cifrul cu cheie acoperitoare, sunt uşor de atacat de un adversar dispunând de două texte cifrate cu aceeaşi cheie.

Punctele 2 şi 3 pot fi contracarate prin anumite proprietăţi ale cifrului (vezi § 6.1.3).

Pentru punctele 1 şi 4, orice cifru, în forma în care este folosit în practică, mai primeşte în funcţia de criptare un argument aleator. O parte din acest argument, numită *vector de iniţializare*, are rolul de-a face ca acelaşi text clar să fie cifrat în mod diferit în mesaje diferite.

În acest caz, criptarea are forma $c : T \times K \times R \to M$ şi decriptarea $d : M \times K \to T$, 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, informaţia corespunzătoare argumentului aleator trebuie să se regăsească în textul cifrat. Ca urmare, lungimea textului cifrat trebuie să fie cel puţin 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 emiţător. De asemenea, nu este permis ca adversarul să mai aibă vreun control asupra conţinutului textului clar după ce obţine informaţii 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 cunoaşte dinainte cheia poate decripta orice text cifrat. Dacă adversarul obţine cheia, înseamnă că cifrul este complet spart.

Un cifru este *parţial spart* dacă un adversar care nu cunoaşte iniţial cheia poate dobândi informaţii despre textul clar prin observarea textului cifrat. Dacă adversarul poate decripta o parte din textul clar sau poate să verifice dacă un anumit şir apare în textul clar, înseamnă că cifrul este parţial spart.

Se poate presupune că un adversar poate estima textele clare ce ar putea fi transmise şi eventual probabilităţile lor; există cazuri în care textul clar transmis este dintr-o mulţime mică, de exemplu poate fi doar *da* sau *nu*. Dacă un adversar ce a interceptat textul cifrat poate elimina anumite texte clare, sau, estimând probabilităţile diverselor chei de cifrare, poate estima probabilităţi, pentru textele clare, diferite faţă de estimările sale iniţiale, înseamnă de asemenea că adversarul a extras informaţie din textul cifrat şi în consecinţă cifrul este parţial spart.

Exemplul 6.3: Considerăm că, din informaţiile adversarului, textul clar este este cu probabilitate de 30% ION, cu probabilitate de 40% ANA şi cu probabilitate de 30% DAN. De asemenea, presupunem că adversarul ştie că se utilizează substituţie monoalfabetică.

În momentul în care adversarul intercepteză textul cifrat `AZF`, el calculează probabilităţile diverselor texte clare cunoscând textul cifrat şi găseşte 50% ION, 0% ANA şi 50% DAN (exclude ANA deoarece ar da aceeaşi literă pe prima şi pe ultima poziţie în textul cifrat). Adversarul a dobândit o informaţie asupra textului clar, ceea ce înseamnă că cifrul a fost spart parţial.

Cu privire la informaţiile de care dispune adversarul ce încearcă spargerea cifurlui, 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 *informaţională*,
- *dificultate computaţională*.

Dificultatea informaţională constă în faptul că pot exista mai multe perechi text clar, cheie, care ar fi putut produce textul cifrat interceptat m.

Presupunând $|T| = |M|$ şi că orice bijecţie $c : T \to M$ putea fi aleasă, cu egală probabilitate, ca funcţie de criptare, adversarul care recepţionează un text cifrat m nu poate deduce nimic cu privire la textul clar t — orice text clar avea aceeaşi probabilitate de a genera m. Un astfel de cifru este perfect — textul cifrat nu aduce nici o informaţie adversarului.

Deoarece există $(|T|)!$ bijecţii posibile, lungimea necesară a cheii este $\log_2((|T|)!)$. Presupunând că T este mulţimea şirurilor de n biţi, avem $|T| = 2^n$ şi lungimea cheii este $\log_2((2^n)!)$ biţi, lungime a cărei comportament asimptotic este de forma $\Theta(n2^n)$. Pentru $n = 20$, cheia are câţiva megabiţi.

De notat că un algoritm de criptare secret nu este un cifru perfect, deoarece nu toate cele $(2^n)!$ funcţii de criptare posibile au aceeaşi 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 informaţională constă în imposibilitatea adversarului de a distinge între *DAN* şi *ION*

Incertitudinea adversarului asupra textului clar este cel mult egală cu incertitudinea asupra cheii. De aici rezultă că, pentru a obţinerea unui cifru perfect, numărul de biţi ai cheii trebuie să fie mai mare sau egal cu numărul de biţi de informaţie din mesaj. Cifrul Vernam este în acelaşi timp perfect (sub aspectul dificultăţii informaţionale a spargerii) şi optim din punctul de vedere al lungimii cheii.

Dificultatea computaţională constă în imposibilitatea adversarului de a deduce informaţii asupra textului clar cu un efort computaţional 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 forţă brută* (engl. *brute force attack*) constă în a decripta textul cifrat folosind toate cheile posibile şi a verifica dacă se obţine textul clar (sau un text clar inteligibil, dacă textul clar adevărat nu este cunoscut dinainte). Fezabilitatea unui atac prin forţă brută depinde direct de lungimea cheii (de fapt, de numărul de chei posibile). Pentru o cheie de 56 de biţi (exemplu cifrul DES), un atac prin forţă brută este perfect posibil, la viteza

actuală necesitând un efort în jur de un an-calculator. Un atac prin forţă brută este nefezabil deocamdată de la 80 de biţi în sus; se consideră că va fi fezabil în jurul anului 2015. De la 128 de biţi în sus atacul prin forţă brută necesită, din cauza unor limitări fizice teoretice, o cantitate de energie comparabilă cu producţia mondială pe câteva luni; o astfel de cheie este puţin probabil că va putea fi spartă vreodată prin forţă 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 forţă brută. Inexistenţa unei metode eficiente de spargere nu este niciodată demonstrată; în cel mai bun caz se demonstrează că spargerea unui cifru este cel puţin 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 demonstraţie 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 $x \in \{0, \ldots, p-1\}$ a ecuaţiei $a^x = b \pmod p$, 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 biţi, există 2^n posibilităţi pentru conţinutul unui bloc. Considerând o distribuţie uniformă a conţinutului fiecărui bloc, un şir de $2^{n/2}$ blocuri are probabilitate cam $1/2$ să aibă cel puţin două blocuri cu conţinut identic. (Acest fapt este cunoscut ca *paradoxul zilei de naştere:* într-un grup de 23 de persoane, probabilitatea să existe două dintre ele născute în aceeaşi zi din an este peste 50%; în general, într-un grup de $\sqrt{k}$ numere aleatoare având k valori posibile, probabilitatea ca cel puţin două să fie egale este în jur de $1/2$).

Ca o consecinţă, dimensiunea n a blocului trebuie să fie suficient de mare şi 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 biţi. La această lungime, repetarea unui bloc de text cifrat este probabil să apară începând de la 2^{32} blocuri, adică 32 GiB.

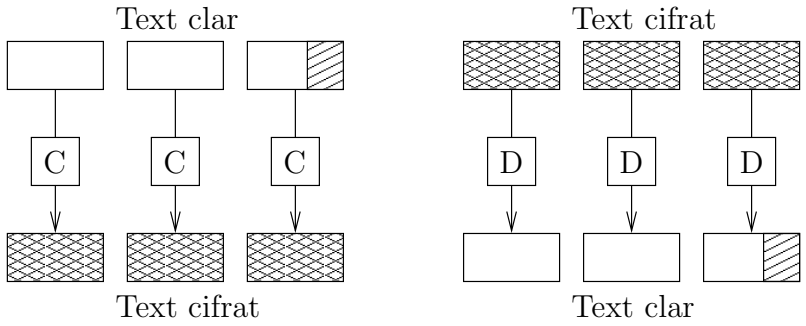
6.1.4. Algoritmi de criptare utilizaţi în practică

Cifrurile mai cunoscute şi 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 biţi), şi *cifru flux* (engl. *stream cipher*), care criptează mesaje

de lungime arbitrară şi produc biţii textului cifrat pe măsură ce primesc biţii 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 biţi.

ECB — Electronic Code Book: Textul clar se împarte în blocuri de lungime n. Ultimul bloc se completează la lungimea n; biţii adăugaţi pot fi zerouri, biţi aleatori sau se pot utiliza alte scheme de completare. Fiecare bloc se criptează apoi independent de celelalte (vezi fig. 6.1).



(a) Criptarea (b) Decriptarea

Figura 6.1: Criptarea în mod ECB

Metoda ECB nu se recomandă deoarece pentru o cheie fixă acelaşi text clar se transformă în acelaşi text cifrat.

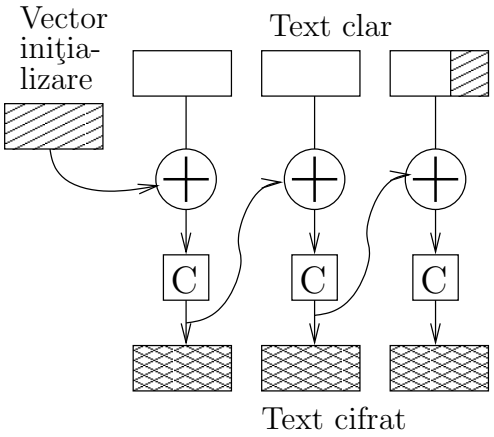
O altă critică citată frecvent este că un adversar care permută blocurile de text cifrat va obţine permutarea blocurilor corespunzătoare de text clar, chiar dacă nu înţelege nimic din textul cifrat. Deşi afirmaţia este adevărată, critica este nefondată întrucât un cifru nu are ca scop protejarea integrităţii mesajelor.

CBC — Cipher Block Chaining: (Vezi fig. 6.2.) Ca şi la ECB, textul clar se împarte în blocuri şi ultimul bloc se completează cu biţi aleatori. În plus, se alege un şir de n biţi aleatori; acesta se numeşte *vector de iniţializare*. Vectorul de iniţializare se transmite de obicei separat. Se efectuează *xor* pe biţi între vectorul de iniţializare şi primul bloc de text clar; rezultatul se cifrează şi se trimite. Apoi, se face *xor* între fiecare bloc de text clar şi blocul precedent de text cifrat, şi rezultatul se cifrează şi se transmite destinatarului.

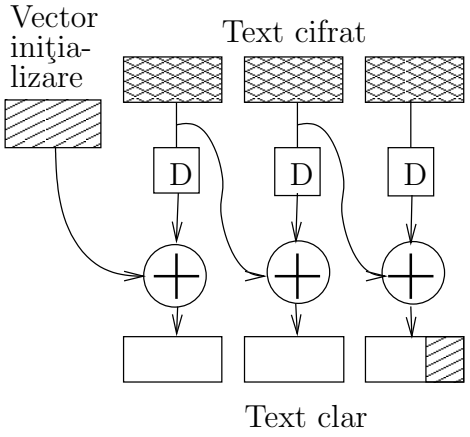
Faţă de ECB, metoda CBC face ca un acelaşi bloc de text clar să se cripteze diferit, în funcţie de vectorul de iniţializare utilizat. Dacă

Nume	lungime bloc	lungime cheie	observaţii
DES	64	56	A fost utilizat pe scară destul de largă, fiind susţinut ca standard de către guvernul Statelor Unite ale Americii. În prezent este nesigur datorită lungimii mici a cheii (a fost deja spart prin forţă brută). Au existat şi speculaţii cum că ar fi fost proiectat astfel încât să fie uşor de spart de către cei care ar cunoaşte nişte 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 forţă 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 şi Vincent Rijmen şi publicat sub numele *rijndael*.
CAST-128	64	între 40 şi 128 biţi	Creat de Carlisle Adams şi Stafford Tavares în 1996.
Blowfish	64	până la 448 biţi	Creat de Bruce Schneier în 1993.
Twofish	128	până la 256 biţi	Creat de Bruce Schneier şi alţii şi a participat la concursul pentru AES.
Serpent	128	128, 192 sau 256	Creat de Ross Anderson, Eli Biham şi 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 biţi	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 iniţializare este ales aleator, repetările unui bloc de text cifrat vor fi extrem de rare (imposibil de exploatat de adversar).

Vectorul de iniţializare trebuie ales satisfăcând :

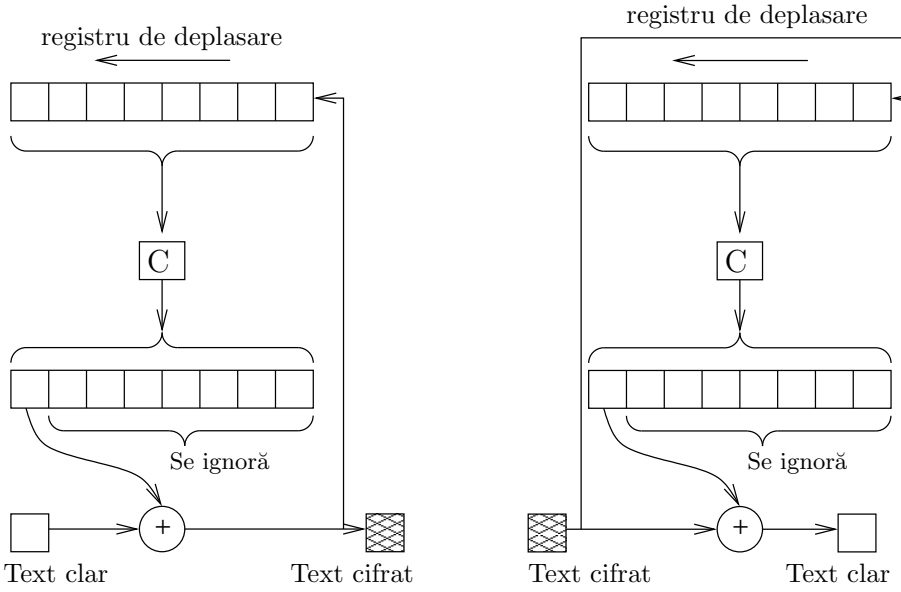
- să fie distribuit uniform şi necorelat cu textul clar sau alţi vectori de iniţializare;
- să nu poată fi controlat de adversar;
- să nu poată fi aflat de adversar cât timp adversarul ar putea influenţa textul clar, pentru a împiedica un atac cu text clar ales.

Vectorul de iniţializare se construieşte utilizând una din următoarele variante:

- se generează cu un generator de numere aleatoare criptografic (vezi § 6.4) şi se transmite în clar înaintea mesajului;
- se stabileşte prin metode asemănătoare cu stabilirea cheii (vezi § 6.3);
- dacă se transmit mai multe mesaje unul după altul, vectorul de iniţializare pentru un mesaj se ia ca fiind ultimul bloc al mesajului precedent (ca la înlănţiurea blocurilor în cadrul aceluiaşi 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 şi cu conţinut aleator.

CFB — Cipher Feedback: CFB şi următorul mod, OFB, sunt utilizate în special acolo unde mesajele sunt mult mai scurte decât dimensiunea blocului, însă emiţătorul transmite aceluiaşi receptor o secvenţă 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 şi trimis).

CFB criptează fragmente de text clar de dimensiune fixă m. Dimensiunea m a fragmentului trebuie să îndeplinească o singură restricţie, şi anume să fie un divizor al dimensiunii n a blocului cifrului. Se poate lua $m = 8$ şi atunci cifrul criptează câte un caracter. CFB funcţionează astfel (vezi fig. 6.3): Emiţătorul generează aleator un vector de iniţializare de n biţi, pe care îl transmite receptorului şi îl încarcă totodată într-un registru de deplasare. Apoi, pentru fiecare caracter de



(a) Criptarea (b) Decriptarea

Figura 6.3: Criptarea în mod CFB

criptat, emiţătorul:

- criptează conţinutul registrului de deplasare utilizând cheia secretă,
- execută *xor* pe biţi între următorii m biţi din textul clar şi primii m biţi din rezultatul criptării,
- transmite ca text cifrat rezultatul pasului precedent,
- deplasează conţinutul registrului de deplasare cu m biţi spre stânga,
- introduce, pe poziţiile cele mai din dreapta ale registrului de deplasare, m biţi 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 emiţător şi receptor, sincronismul se reface automat după n biţi.

De remarcat, de asemenea, că decriptarea CFB utilizează tot funcţia 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 şir pseudoaleator generat cu un algoritm de criptare. Primii n biţi din şirul pseudoaleator se obţin criptând vectorul de iniţializare, următorii n biţi

se obţin criptând precedenţii n biţi pseudoaleatori, ş. a. m. d.

La OFB utilizarea unui vector de iniţializare aleator este chiar mai importantă decât la celelalte moduri de criptare, întrucât refolosirea unui vector de iniţializare conduce la repetarea şirului pseudoaleator (cheia Vernam), rezultând un cifru relativ uşor de spart.

CTR — Counter: Se construieşte similar cu OFB, însă şirul pseudoaleator se obţine criptând numerele v, $v+1$, $v+2$, etc., reprezentate pe n biţi, v fiind vectorul de iniţializare. Modul CTR este foarte asemnănător 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 fişierelor pe disc. Totuşi, deoarece cifrul Vernam aflat la bază este vulnerabil unui adversar având la dispoziţie două texte clare criptate cu aceeaşi cheie, metoda este vulnerabilă dacă adversarul are posibilitatea de-a obţine două variante ale unui fişier.

6.1.5. Criptografie asimetrică (cu cheie publică)

Intuitiv, s-ar putea crede că, dacă funcţia de criptare este complet cunoscută (inclusiv cheia), funcţia inversă — decriptarea — este de asemenea calculabilă în mod rezonabil. În realitate, există funcţii de criptare (injective) a căror cunoaştere nu permite decriptarea în timp rezonabil. În esenţă, ideea este că, deşi $m = c(t)$ este rezonabil de uşor de calculat, determinarea lui t din ecuaţia $c(t) = m$ 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 totuşi decripta în timp rezonabil. Pentru aceasta, se cere ca funcţia de criptare c să poată fi inversată uşor de către cineva care cunoaşte o anumită informaţie, dificil de dedus din c. Această informaţie va fi făcută cunoscută doar destinatarului mesajului criptat. De fapt, în cazul criptografiei asimetrice, destinatarul mesajului este chiar autorul acestei informaţii secrete şi a funcţiei de criptare.

Ca şi în cazul criptografiei simetrice, funcţia de criptare c primeşte 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 şi cu k_d informaţia 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 computaţional, să se calculeze cheia secretă k_d corespunzătoare. Pentru ca sistemul de criptare să fie util mai este necesar să existe un procedeu eficient (computaţional) 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 algoritmii de criptare c şi decriptare d şi un algoritm de generare aleatoare a perechilor de chei (k_c, k_d).

Pentru ca sistemul criptografic să fie sigur trebuie ca rezolvarea ecuaţiei $c_{k_c}(t) = m$ cu necunoscuta t să fie dificilă computaţional. Implicit, determinarea cheii secrete k_d corespunzătoare unei chei publice k_c trebuie de asemenea să fie dificilă computaţional.

Exemplul 6.4 (*Cifrul RSA*): Generarea cheilor se face astfel:

- se generează numerele prime p şi q (de ordinul a 500 cifre zecimale fiecare);
- se calculează $n = pq$ şi $\phi = (p-1)(q-1)$;
- se generează aleator un număr $e \in \{2, 3, \ldots, \phi - 1\}$, 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 $k_d = (n, d)$.

Spaţiul textelor clare şi spaţiul textelor cifrate sunt

$$P = M = \{0, 1, \ldots, n-1\}.$$

Criptarea şi decriptarea sunt:

$$c_{k_c}(p) = p^e \pmod{n} \tag{6.1}$$
$$d_{k_d}(m) = m^d \pmod{n} \tag{6.2}$$

Cifrul a fost inventat de Rivest, Shamir şi Adelman în 1977. Numele RSA vine de la iniţialele autorilor.

6.1.5.1. Utilizarea criptografiei asimetrice

Pentru pregătirea comunicaţiei, receptorul generează o pereche de chei (k_c, k_d) şi face publică k_c. Emiţătorul poate cripta un mesaj, folosind k_c, şi 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 (deşi autorul — şi dealtfel oricine — poate verifica dacă un text cifrat dat corespunde sau nu unui text clar dat).

Dacă se doreşte comunicaţie bidirecţională, se utilizează câte o pereche de chei (k_c, k_d) distinctă pentru fiecare sens.

O aceeaşi pereche de chei poate fi utilizată de o entitate pentru toate mesajele pe care le primeşte, indiferent cu câţi parteneri comunică. Astfel, fiecare entitate îşi stabileşte o pereche de chei din care cheia publică o transmite tuturor partenerilor de comunicaţie şi cheia secretă o foloseşte pentru a decripta mesajele trimise de toţi către ea. Pentru comparaţie, în criptografia simetrică, fiecare pereche de parteneri ce comunică trebuie să aibă propria cheie secretă.

Un sistem criptografic asimetric este în esenţă un cifru bloc. Pentru a cripta o cantitate arbitrară de text clar, se pot utiliza modurile ECB sau CBC. Modurile CFB, OFB şi CTR nu sunt utilizabile deoarece utilizează doar funcţia de criptare, care în cazul criptografiei asimetrice este publică.

Algoritmii criptografici asimetrici sunt mult mai lenţi 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 şi § 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 şi că mesajul nu a fost modificat de altcineva.

Verificarea autenticităţii unui mesaj constă în aplicarea de către receptor a unui *test de autenticitate* asupra mesajului primit. Un test de autenticitate trebuie să îndeplinească două proprietăţi:

- orice mesaj autentic să treacă testul;
- pentru un mesaj neautentic, produs cu un efort computaţional 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 conţinut inteligibil; trebuie doar să fie acceptat de algoritmul de autentificare.
- *fals ales* (engl. *choosen forgery*): în plus faţă de falsul existent, conţinutul mesajului neautentic este (total sau în mare parte) la alegerea adversarului.

Evident, un mesaj, acceptat ca autentic o dată, va fi acceptat ca autentic şi în cazul unei repetări ulterioare. Prevenirea unor atacuri bazate pe

repetarea unor mesaje anterioare este o problemă separată şi 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 privinţa autenticităţii. De exemplu, textul unei legi este o informaţie publică, dar un cetăţean 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) şi este inteligibil nu e o garanţie 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 cunoaşte efectiv textul clar. Din acest motiv, metodele de asigurare a confidenţialităţii se separă de metodele de control a autenticităţii.

6.2.1. Funcţii de dispersie criptografice

În general (nu neapărat în criptografie), prin *funcţie de dispersie* (engl. *hash function*) se înţelege o funcţie h care asociază unui şir de biţi t, de lungime oricât de mare, o valoare întreagă într-un interval de forma $[0, 2^n)$ cu n fixat (sau echivalent, un şir de biţi de lungime n), satisfăcând condiţia că, pentru şirurile care apar in problema unde se foloseşte funcţia de dispersie, două şiruri distincte să nu aibă aceeaşi valoare a funcţiei de dispersie cu probabilitate semnificativ mai mare de 2^{-n}. Valoarea funcţiei de dispersie aplicată unui şir se numeşte *dispersia* acelui şir.

O *funcţie de dispersie criptografică* este o funcţie de dispersie care are anumite proprietăţi suplimentare, dintre cele enumerate în continuare:

1. *rezistenţa la preimagine* (engl. *preimage resistence*): dându-se $h(t)$, să fie dificil de regăsit t. Eventual, dificultatea să se păstreze chiar în cazul cunoaşterii unei părţi din t.
2. *rezistenţa la a doua preimagine* (engl. *second preimage resistence*): dându-se un şir t, să fie dificil de găsit un al doilea şir t', cu $t' \neq t$, astfel încât $h(t) = h(t')$.
3. *rezistenţa la coliziuni* (engl. *collision resistence*): să fie dificil de găsit două şiruri distincte t_1 şi t_2 ($t_1 \neq t_2$), astfel încât $h(t_1) = h(t_2)$.

De remarcat că cele trei condiţii sunt diferite. Totuşi, condiţia de rezistenţă la coliziuni implică rezistenţa la a doua preimagine. De asemenea,

majoritatea funcţiilor rezistente la coliziuni satisfac şi condiţia de rezistenţă la preimagine.

Numărul de biţi n ai dispersiei trebuie să fie suficient de mare pentru a împiedica căutarea unei coliziuni prin forţă brută. Conform paradoxului zilei de naştere, există şanse mari de găsire a unei coliziuni într-o mulţime de $2^{n/2}$ intrări. Pentru a face impractic un atac prin forţă brută, trebuie ca $n/2 \geq 64$ (şi mai bine $n/2 \geq 80$), de unde $n \geq 128$ sau mai bine $n \geq 160$.

Funcţiile de dispersie mai cunoscute sunt descrise în tabelul 6.2.

Nume	lungime	observaţii
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.
RIPEMD-160	160	Dezvoltată la Katholieke Universiteit Leuven în 1996.

Tabelul 6.2: Funcţii de dispersie criptografice.

6.2.1.1. Utilizarea funcţiilor de dispersie

Presupunem existenţa între emiţător şi receptor a două canale de transmitere a informaţiei: un canal principal nesigur şi 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 funcţie de dispersie rezistentă la a doua preimagine şi preferabil rezistentă la coliziuni.

Emiţătorul unui mesaj t calculează $s = h(t)$. Apoi, transmite t prin canalul principal şi 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 proprietăţii de rezistenţă la a doua preimagine a funcţiei de dispersie h.

Există situaţii practice în care t este (parţial) la dispoziţia adversarului. De exemplu, presupunem că secretara redactează un mesaj t la cererea şefului, secretara putând alege formularea exactă a mesajului t. Şeful îşi exprimă acordul asupra mesajului calculând şi trimiţând destinatarului $s = h(t)$. Dacă adversarul este secretara, ea nu se găseşte în situaţia de-a crea un t' satisfăcând $h(t') = h(t)$ pentru t fixat (adică de-a crea a doua preimagine) ci

este în situaţia de-a crea t şi t' distincte cu $h(t) = h(t')$ (adică de-a găsi o coliziune). Din acest motiv, o funcţie de dispersie utilizată pentru controlul autenticităţii mesajelor se cere să fie rezistentă la coliziuni.

Există pe sistemele Linux comenzile `md5sum` şi `sha1sum` care calculează şi afişează dispersia *md5* respectiv *sha1* a conţinutului unui fişier. Dispersia este afişată în hexa. Dacă notăm într-un loc sigur dispersia unui fişier, putem controla ulterior dacă fişierul a fost sau nu modificat între timp.

6.2.2. Funcţii de dispersie cu cheie

O *funcţie de dispersie cu cheie* (engl. *keyed hash function*), numită şi *MAC* (*message authentication code*), este o funcţie de dispersie $h_k(t)$, parametrizată cu o cheie k, având proprietatea că, pentru cineva care nu cunoaşte dinainte cheia k, este nefezabil computaţional să obţină o (nouă) pereche (s, t) în care $s = h_k(t)$, chiar dacă cunoaşte un număr de perechi (s_i, t_i) cu $s_i = h_k(t_i)$.

O funcţie de dispersie cu cheie se utilizează astfel: Mai întâi, emiţătorul şi receptorul se înţeleg asupra unei chei secrete k (de exemplu conform metodelor din § 6.3). La trimiterea unui mesaj t, emiţătorul calculează $s = h_k(t)$ şi 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 şi B cele două părţi care comunică şi cu k cheia de dispersie utilizată pentru autentificarea mesajelor. Un adversar activ poate intercepta un mesaj trimis de A către B şi să-l trimită înapoi lui A. Dacă aceeaşi cheie k este utilizată pentru autentificarea ambelor sensuri de comunicaţie (şi de la A la B, şi de la B la A) şi dacă mesajele au acelaşi format, atunci A acceptă mesajul ca venind de la B. Pentru a preveni un atac prin reflexie, există două soluţii:

- Se utilizează chei distincte pentru cele două sensuri.
- Fiecare mesaj conţine numele entităţii emiţătoare. Eventual, numele entităţii nu apare efectiv în mesajul transmis, dar participă la calculul dispersiei: $s = h_k(t \cdot \mathtt{A})$ şi A trimite spre B perechea (t, s).

Argumente similare cu cele privind dimensiunea blocurilor la cifrurile bloc şi dimensiunea cheii de cifrare conduc la cerinţe pentru împiedicarea atacurilor prin forţă brută: dimesiunea cheii şi dimensiunea dispersiei de minim 64 de biţi, preferabil 80 de biţi.

O construcţie uzuală pentru funcţii de dispersie cu cheie pornind de la funcţii de dispersie rezistente la coliziuni şi 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 operaţia *sau exclusiv*,
- hash este funcţia de dispersie criptografică (de exemplu *md5* sau *sha1*),
- K este cheia k completată la o lungime B aleasă în funcţie de anumite particularităţi ale funcţiei de dispersie de la bază; pentru *md5* şi *sha1*, B se ia de 64 de octeţi.
- ipad şi opad sunt şiruri obţinute prin repetarea de B ori a octetului cu valoarea (hexa) 36, respectiv 5C.

Rezultatul funcţiei hash se poate trunchia la lungime mai mică (notăm că funcţia de dispersie hash dă 128–160 biţi, iar pentru o dispersie cu cheie sunt suficienţi 64–80 de biţi). Trunchierea are ca avantaj micşorarea cantităţii de informaţie pusă la dispoziţia adversarului.

O construcţie uzuală pentru funcţii 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 iniţializare zero (sau iniţializat cu dispersia mesajului precedent);
- rezultatul criptării ultimului bloc se criptează utilizând o a doua cheie (cheia funcţiei de dispersie este considerată ca fiind concatenarea celor două chei de criptare), rezultând valoarea dispersiei.

6.2.3. Semnătura digitală

Semnătura digitală este o construcţie similară dispersiei cu cheie, studiată în paragraful precedent. Construcţia este însă asimetrică, utilizând chei diferite pentru crearea dispersiei (numită, în acest caz, semnătură) şi, respectiv, pentru verificare dispersiei. Astfel, relaţia dintre semnătura digitală şi dispersia cu cheie este similară cu cea dintre criptografia asimetrică şi 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 funcţie de semnare h;
- o funcţie de verificare v.

În faza pregătitoare, autorul de mesaje semnate generează o pereche de chei (k_s, k_v) şi 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* $s = h_{k_s}(t)$ şi transmite perechea (s, t). Receptorul verifică dacă $v_{k_v}(t, s) = \text{true}$.

Aşa cum se vede, semnătura s depinde şi de mesajul de semnat t şi de semnatarul acestuia (mai exact de cheia k_s). Ca urmare, o semnătură nu poate fi tăiată de pe un mesaj şi plasată pe alt mesaj.

Unii algoritmi de semnătură digitală necesită un al treilea argument pentru funcţia de semnătură; acest argument trebuie să fie un număr aleator. În acest caz există mai multe semnături valide pentru un acelaşi mesaj.

O posibilitate de construcţie pentru semnătură este pe baza unui mecanism de criptare asimetric în care criptarea este bijectivă; de exemplu RSA are această proprietate. Construcţia simplificată este:

$$\begin{aligned} h_{k_s}(t) &= d_{k_s}(t) \\ v_{k_v}(t, s) &= (c_{k_v}(s) = t) \end{aligned}$$

Construcţia de mai sus se bazează pe nefezabilitatea calculului lui $s = d_{k_s}(t)$ fără cunoaşterea lui k_s. Totuşi, construcţia aceasta permite adversarului să producă un fals existent: un adversar poate alege aleator un s şi calcula $t = c_{k_v}(s)$.

O îmbunătăţire 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 şi la coliziuni a lui t. Metoda are două avantaje, pe de o parte că algoritmul de criptare asimetric, lent, se aplică asupra dispersiei şi 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 şi nonrepudiabilitatea mesajelor. Acest lucru se întâmplă deoarece cheia de semnătură este cunoscută doar de către emiţător. Ca urmare, doar emiţătorul poate genera semnătura. Prin urmare, prezenţa unei semnături verificabile atestă faptul că documentul a fost produs de emiţător. Funcţiile de dispersie cu cheie nu realizează (direct) mesaje nerepudiabile deoarece receptorul poate produce mesaje semnate la fel de bine ca şi emiţătorul.

6.2.4. Verificarea prospeţimii mesajelor

Este adesea necesar ca receptorul să poată distinge între un mesaj (autentic) „nou“ şi o copie a unui mesaj mai vechi. De exemplu, dacă mesajul cere destinatarului să execute o operaţie 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 prospeţime 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 aceeaşi sumă de bani către acelaşi destinatar), iar pe de altă parte, memorarea tuturor mesajelor deja primite nu este fezabilă.

Soluţiile problemei verificării prospeţimii sunt similare cu metodele de transmisie sigură (§ 4.3), cu diferenţa că trebuie să reziste la atacuri voite, nu numai la disfuncţionalităţi întâmplătoare.

Ideea este să introducem în mesajul autentificat un „identificator de mesaj“ care să fie diferit de la un mesaj la altul şi asupra căruia să se execute de fapt testul de prospeţime. Un astfel de element se numeşte *număr unic* (engl. *nonce*, de la *number (used) once*).

Numărul unic poate fi:

un număr de ordine: Emiţătorul ţine evidenţa unui număr curent de ordine. Pentru fiecare mesaj, emiţătorul scrie numărul curent în mesaj şi incrementează apoi numărul curent. Receptorul ţine de asemenea evidenţa 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 menţinerea pe termen lung a numerelor de ordine curente şi necesită un contor separat de număr de ordine pentru fiecare partener de comunicaţie.

ora curentă: Emiţătorul 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 puţin câteva zecimi de secundă, deoarece ceasurile nu sunt

perfect sincronizate şi 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: Emiţătorul se asigură că două mesaje distincte vor avea numărul unic distinct. Pentru aceasta, fie rezoluţia marcajului de timp se face mai fină decât timpul necesar emiterii unui mesaj, fie emiţătorul mai ţine un contor care se incrementează la fiecare mesaj trimis şi 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 prospeţime (de exemplu, dacă mesajul este trimis la ora 08:12:45 şi testul de prospeţime 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) şi dacă numărul unic nu este identic cu cel al unuia dintre mesajele memorate.

Utilizarea orei la verificarea prospeţimii necesită un mecanism sigur care să menţină sincronismul ceasurilor dispozitivelor care comunică.

un număr (aleator) ales de receptor: Receptorul care aşteaptă un mesaj trimite emiţătorului un număr aleator proaspăt generat. Numărul aleator trebuie să aibă cel puţin 64–80 biţi, pentru ca să nu se repete (decât cu probabilitate neglijabil de mică) şi să nu poată fi prezis de către adversar. Emiţătorul adaugă numărul la mesajul trimis. Receptorul acceptă un mesaj ca proaspăt doar dacă numărul aleator din mesaj coincide cu numărul aleator tocmai trimis. Ca şi 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 emiţător. În plus, este necesar ca receptorul să ştie că urmează să primească un mesaj, ceea ce necesită adesea încă un mesaj (emiţătorul spune *vezi că vreau să-ţi spun ceva*, receptorul răspunde trimiţând numărul aleator şi, în final, emiţătorul trimite mesajul propriu-zis). Acest lucru face aplicarea metodei scumpă şi în anumite cazuri imposibilă — de exemplu metoda nu este aplicabilă pentru securizarea poştei 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. Emiţătorul include în fiecare pachet al conexiunii numărul aleator primit la deschiderea conexiunii şi numărul de ordine al pachetului.

6.2.5. Combinarea criptării, autentificării şi verificării prospeţimii

Verificarea prospeţimii 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 şi 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. Deşi nu există o slăbiciune cunoscută, unii criptografi susţin că prezenţa 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 şi 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 vulnerabilităţi (în caz contrar, se poate arăta că funcţia 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 bănuieşte textul clar să verifice dacă textul clar este cel bănuit de el.

6.3. Stabilirea cheilor

În paragrafele precedente, am presupus că partenerii de comunicaţie dispun deja de cheile necesare criptării şi 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 cerinţe distincte.

În cazul unei chei simetrice, acţiunea prin care cheia este generată şi făcută disponibilă partenerilor de comunicaţie se numeşte *stabilirea cheii*. Un protocol de stabilire a cheii trebuie să îndeplinească următoarele cerinţe:

- Cheia stabilită să nu poată fi cunoscută de altcineva decât de entităţile ce doresc să comunice. Acţiunea de satisfacere a acestei cerinţe poartă denumirea de *autentificarea cheilor* şi este obligatorie în orice proces de stabilire a cheilor.
- Cheia stabilită să fie proaspătă şi, eventual, să nu poată fi impusă unilateral de vreuna dintre părţi ci să fie calculată din elemente generate de fiecare dintre parteneri. Această cerinţă este utilă pentru a preîntâmpina situaţia în care un adversar, care reuşeşte să obţină o cheie mai veche, ar putea determina entităţile care comunică să refolosească acea cheie.
- Fiecare entitate ce comunică să aibă confirmarea că partenerul de comunicaţie a primit efectiv cheia. Acţiunea care verifică satisfacerea acestei cerinţe 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 comunicaţiei propriu-zise cu ajutorul cheii respective constituie confirmarea cheii.

Notăm că, în anumite cazuri, autentificarea cheii stabilite se face unilateral (adică doar una dintre entităţi ştie cu certitudine ce entitate mai cunoaşte 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, acţiunea prin care cheia este făcută disponibilă partenerilor se numeşte *certificarea cheii*. Spre deosebire de cazul cheilor simetrice, unde transmisia unei chei între partenerii de comunicaţie se face doar pentru o cheie proaspăt generată, în cazul cheilor publice se întâmplă frecvent ca o aceeaşi cheie publică să fie transmisă de mai multe ori către o aceeaşi entitate. Certificarea unei chei are următoarele cerinţe:

- Receptorul unei chei publice să poată verifica dacă cheia primită este într-adevăr cheia publică a partenerului de comunicaţie.
- 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 prezenţa unui adversar, stabilirea cheilor simetrice şi certificarea cheilor publice necesită mecanisme de securizare a comunicaţiei (criptare şi autentificare).

Stabilirea cheilor sau certificarea cheilor între două entităţi care nu au deja chei care să le permită o comunicaţie securizată între ele se poate face prin două metode:

- *cu ajutorul unui terţ de încredere*: Această metodă necesită existenţa unei a treia entităţi care să poată deja comunica securizat cu fiecare din primele două şi care să prezinte încredere acestora.
- *prin intermediul unui utilizator uman* (§ 6.3.5).

După rolul lor faţă de un mecanism de stabilire a cheilor, cheile se numesc:

- *chei efemere* (engl. *ephemeral key*) sau *chei de sesiune* (engl. *session key*), utilizate pentru comunicaţia propriu-zisă între două entităţi. 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 şi este distrusă imediat după aceea. Deoarece, din motive de viteză, comunicaţia propriu-zisă este protejată prin criptografie simetrică şi, 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 posede o entitate pentru a putea comunica trebuie să fie cât mai mic. Ideal, fiecare entitate dispune de o singură cheie de lungă durată, permiţând o comunicaţie securizată cu un terţ de încredere. Atunci când entitatea are nevoie să comunice cu o altă entitate, obţine, prin intermediul terţului de încredere, cheia necesară comunicării cu partenerul dorit. După utilizare, cheia respectivă poate fi ştearsă.
- Intervenţia manuală în stabilirea cheilor trebuie să fie minimă. Totuşi, 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 comunicaţia legată de stabilirea unei chei de sesiune şi, ulterior, obţine o cheie de lungă durată utilizată în stabilirea acelei chei de sesiune, este bine să nu poată să obţină cheia de sesiune, pentru a nu putea mai departe decripta sesiunea.

6.3.1. Stabilirea cheilor în prezenţa unui adversar pasiv

Ne vom ocupa în continuare de stabilirea unei chei de sesiune între două entităţi, A şi B, în prezenţa unui adversar pasiv şi în lipsa vreunei chei deja disponibile. Cu alte cuvinte, problema este ca A şi B să ajungă la un secret partajat printr-o comunicaţie integral la vedere.

În acest paragraf nu ne vom ocupa de situaţia în care un adversar ar putea participa activ în schimbul de mesaje. Aplicarea metodelor din acest paragraf în prezenţa 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 îmbunătăţită 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 şi cheia publică proaspătă), iar B criptează cheia de sesiune cu cheia proaspătă iar rezultatul îl criptează cu cheia de lungă

durată. Avantajul obţinut este că dacă un adversar obţine cheia secretă de lungă durată a lui A, dar între timp cheia proaspătă a expirat şi a fost distrusă, adversarul nu mai poate decripta comunicaţiile vechi.

Soluţia în varianta simplă este utilizată de PGP/GPG. Aici emiţătorul mesajului are rolul lui B şi receptorul are rolul lui A. Emiţătorul obţine cheia publică a receptorului, iar la trimiterea unui mesaj generează aleator o cheie de sesiune, criptează mesajul folosind cheia de sesiune şi criptează cheia de sesiune folosind cheia publică a destinatarului. Mesajul transmis conţine cele două elemente criptate.

Soluţia în varianta îmbunătăţită este aplicată de protocolul *ssh* versiunea 1. Serverul are rolul lui A, generând perechile de chei şi transmiţând clientului cele două chei publice. Clientul generează cheia de sesiune şi 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 genenerează (pe o cale oarecare) un număr prim p mare şi un număr g;
2. A alege un număr aleator x şi calculează şi-i transmite lui B numărul
$$n_A = g^x \bmod p;$$
3. B alege un număr aleator y şi calculează şi-i transmite lui A numărul
$$n_B = g^y \bmod p;$$
4. A şi 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 aceeaşi:

$$(g^x \bmod p)^y \bmod p = (g^y \bmod p)^x \bmod p = g^{xy} \bmod p,$$

iar calcularea lui $g^{xy} \bmod p$ cunoscând doar g, n, $g^x \bmod p$ şi $g^y \bmod p$ este foarte dificilă.

Ca avantaj faţă de soluţia 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 comunicaţie interactivă; protocolul Diffie-Hellman nu este aplicabil transmiterii mesajelor prin poştă electronică.

6.3.1.3. Atacul man-in-the-middle

Protocoalele descrise în paragrafele 6.3.1.1 şi 6.3.1.2 sunt aplicabile doar în absenţa unui adversar activ. Un adversar activ I se poate interpune între A şi 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 citeşte, eventual le modifică, după care le criptează (şi eventual le autentifică) utilizând cheia k_2.

Rezultatul atacului este că A şi 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; totuşi, 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 cunoaşterea unei chei secrete.

Ca urmare, în orice comunicaţie sigură trebuie să existe cel puţin un pas din iniţializare în care să se transfere o cheie între B şi A, sau între B şi un terţ de încredere şi între terţ şi A.

6.3.2. Stabilirea cheilor în prezenţa unui adversar activ

Vom studia în continuare problema stabilirii unei chei de sesiune între două entităţi, A şi B, în prezenţa unui adversar activ. În urma protocolului, cheia de sesiune trebuie să fie cunoscută doar de A şi de B şi să fie proaspătă (un adversar să nu poată impune alegerea unei chei stabilite la o execuţie anterioară a protocolului). În acest scop, presupunem că A şi B şi-au stabilit chei de lungă durată, simetrice sau asimetrice, pentru criptare şi autentificare.

Există multe protocoale de stabilire a cheilor în această situaţie. În general, aceste protocoale se construiesc astfel:

1. Fie una dintre părţi generează aleator o cheie şi o transmite criptat partenerului, fie se aplică un protocol de tipul Diffie-Hellman.
2. Fiecare parte trimite celeilalte un *nonce* (de exemplu un număr aleator).

3. Fiecare parte trimite celeilalte o semnătură sau o dispersie calculată din informaţiile de la punctele 1 şi 2.

Primul punct are ca scop obţinerea unei chei pe care să o cunoască doar părţile între care a avut loc schimbul de mesaje. Al doilea punct are rolul de-a asigura că negocierea şi, în consecinţă, cheia rezultată este proaspătă. Al treilea punct are rolul de-a asigura fiecare parte că mesajele de la punctele 1 şi 2 au fost schimbate cu partenerul dorit. Dacă verificarea semnăturii sau dispersiei de la punctul 3 eşuează, înseamnă că protocolul a fost influenţat de un adversar activ şi, în consecinţă, cheia negociată este compromisă. Este esenţial deci ca, până în momentul în care verificarea auenticităţii 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 şi B cele două entităţi care comunică, cu K_c o cheie pentru criptare simetrică, cunoscută doar de A şi de B şi cu K_h o cheie pentru dispersie, de asemenea cunoscută doar de A şi de B.

1. A alege un număr aleator r_A şi-l transmite lui B.
2. B alege cheia de sesiune k. Apoi calculează şi transmite $x = c_{K_c}(k)$ şi $s = h_{K_h}(r_A \cdot k)$.
3. A decriptează x obţinând k şi verifică dispersia s.

De remarcat că, dacă transmisia conţinând cheia de sesiune k criptată cu cheia de lungă durată K_c este interceptată de un adversar, iar adversarul obţine ulterior cheia K_c, atunci adversarul poate decripta cheia de sesiune şi întreaga sesiune protejată cu această cheie.

Exemplul 6.6: Stabilirea cheii de sesiune prin schimb Diffie-Hellman şi 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 şi B cele două entităţi, cu g şi p baza şi modulul din schimbul Diffie-Hellman, cu K_{sA} şi K_{sB} cheile (secrete) de semnătură ale lui A şi B şi cu K_{vA} şi K_{vB} cheile (publice) de verificare corespunzătoare.

1. A alege două numere aleatoare, r_A, utilizat pentru asigurarea prospeţimii schimbului de chei, şi x_A utilizat pentru derivarea cheii prin metoda Diffie-Hellman. Apoi A transmite lui B numerele r_A şi $n_A = g^{x_A} \bmod p$ (unde g şi p sunt parametrii pentru Diffie-Hellman).
2. B procedează similar, alegând r_B şi x_B şi transmiţând r_B şi $n_B = g^{x_B} \bmod p$.
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 $s_B = h_{K_{sB}}(r_B \cdot n_B \cdot r_A \cdot n_A)$.
5. A verifică semnătura s_B şi, dacă se potriveşte, calculează cheia de sesiune $k = n_B^{x_A} \bmod p$.
6. B verifică semnătura s_A şi, dacă se potriveşte, calculează cheia de sesiune $k = n_A^{x_B} \bmod p$.

6.3.3. Stabilirea cheilor cu ajutorul unui terţ de încredere

Vom studia în continuare problema stabilirii unei chei între două entităţi A şi B care nu partajează în prealabil nici un fel de cheie, în prezenţa unui adversar activ. Presupunem, în schimb, existenţa unei a treia entităţi (un *terţ de încredere*, engl. *trusted third party*), T, satisfăcând condiţiile:

- A şi T partajează chei pentru criptare şi autentificare (K_{cA}, respectiv K_{hA});
- B şi T partajează chei pentru criptare şi autentificare; (K_{cB}, respectiv K_{hB});
- A şi B au încredere în serviciile oferite de T.

T se mai numeşte *server de distribuire a cheilor* (engl. *Key Distribuţion Center* — KDC).

Mai presupunem că protocolul de stabilire a cheii este iniţiat de către A.

Ideea soluţiei este următoarea: serverul T generează aleator o cheie de sesiune, pe care o transmite lui A şi lui B. Transmisia către A este criptată şi autentificată cu cheia partajată între A şi T, iar transmisia către B este criptată şi autentificată cu cheia partajată între B şi T.

Deoarece este de presupus că T face acest serviciu pentru mai multe entităţi, pachetele transmise către A şi B trebuie să conţină şi numele lor, astfel încât, la primirea pachetului, A şi B să ştie 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

$$(\mathtt{A}, \mathtt{B}, c_{K_{cA}}(k), h_{K_{cA}}(k \cdot \mathtt{A} \cdot \mathtt{B}))$$

4. T transmite spre B un pachet

$$(\mathtt{A}, \mathtt{B}, c_{K_{cB}}(k), h_{K_{cB}}(k \cdot \mathtt{A} \cdot \mathtt{B}))$$

5. A decriptează cheia de sesiune k şi verifică dispersia, precum şi numele A şi B;
6. B procedează la fel ca şi A.

Protocolul de mai sus oferă doar autentificarea cheii; nu verifică şi prospeţimea acesteia. Vulnerabilitatea introdusă este dată de faptul că un adversar poate trimite mesajele de la paşii 3 şi 4 în locul serverului T pentru a forţa stabilirea unei chei vechi. Dacă adversarul reuşeşte să obţină o cheie de sesiune, el poate forţa astfel stabilirea aceleiaşi chei, compromise, în sesiunile următoare.

Pentru verificarea prospeţimii, trebuie introduse în mesaje nişte elemente de control al prospeţimii.

O primă posibilitate, exploatată de protocolul Kerberos, este adăugarea unei perioade de valabilitate. Perioada de valabilitate este adăugată în mesajele transmise în paşii 1, 3 şi 4. A şi B resping, în cadrul paşilor 5 şi 6, cheia de sesiune dacă timpul curent este în afara perioadei de valabilitate înscrise în pachete lângă cheie.

Protocolul Kerberos mai aduce câteva modificări faţă 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 şi B transmit către T câte un număr aleator, iar T include numărul aleator transmis de A în mesajul de la pasul 3 şi numărul aleator transmis de B în mesajul de la pasul 4. A şi B verifică, în cadrul paşilor 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 terţ de încredere se face construind un server T care crează chei de sesiune, la cerere, pentru toate entităţile din reţea. Astfel, T partajează o cheie simetrică (de fapt, două: una pentru criptare, cealaltă pentru autetificare) cu fiecare dintre entităţile din reţea.

Într-o reţea 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 entităţi în aceste

condiţii, se construieşte 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 şi de legăturile securizate trebuie să fie conex.

Atunci când o entitate A doreşte să comunice cu o entitate B, se caută un lanţ de servere, $T_1,\ldots,T_n$, astfel încât A să aibă cheie partajată cu T_1, T_1 să aibă cheie partajată cu T_2, ş. a. m. d. Apoi, se stabileşte, cu ajutorul lui T_1, o cheie între A şi T_2; cu ajutorul lui T_2 se stabileşte o cheie între A şi T_3 ş. a. m. d. până la obţinerea unei chei între A şi B.

În acest sistem, în loc să aibă toată lumea încredere într-un singur server, fiecare pereche de entităţi care doresc să comunice trebuie să aibă încredere în lanţul 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 doreşte să comunice cu o entitate B îşi pune problema să dobândească cheia publică a lui B. Cheia publică nu este un secret, însă trebuie ca autenticitatea ei să fie verificabilă.

Soluţia 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 soluţie aplicabilă în reţele mari constă în utilizarea certificatelor.

Un *certificat* este un ansamblu cuprinzând o cheie publică, un nume de entitate şi o semnătură a unui terţ pe ansamblul celor două. Terţul semnatar se numeşte *autoritate de certificare*. Prin semnătură, autoritatea de certificare atestă că cheia publică din certificat aparţine entităţii 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 şi, dacă este corectă, poate prelua cheia lui B de pe certificat.

Schema de mai sus poate cuprinde mai multe autorităţi de certificare, din care o primă autoritate a cărei cheie se obţine prin transport de către om şi un lanţ de autorităţi din care fiecare semnează certificatul următoareia.

Pentru schimbarea cheilor, în cazul compromiterii unei chei secrete, se prevăd două mecanisme:

expirarea cheilor. Un certificat are durată de valabilitate limitată; data creerii şi data expirării sunt de asemenea înscrise în certificat şi semnate de autoritatea de certificare. O entitate a cărui certificat se apropie de expirare va crea o nouă pereche de chei şi va solicita autorităţii de certificare eliberarea unui nou certificat pentru noua cheie publică. După expirare, un certificat nu mai este recunoscut de nimeni ca valid şi nu mai este folosit.

revocarea certificatelor. Se ţin, pe servere accesibile public, aşa-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 şi semnătura autorităţii de certificare a certificatului revocat asupra acelor date de identificare.

Publicarea unui certificat de revocare pentru un certificat anunţă 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 puţin 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 intervenţia omului, şi 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 destinaţie. Eventual această cheie poate fi cheia publică a unei autorităţi recunoscute, cheie înglobată într-un produs soft (de regulă browserele web au înglobate astfel de chei);
2. Omul citeşte o informaţie de pe sistemul sursă şi o introduce sau o verifică pe sistemul destinaţie. Cantitatea de informaţie astfel trans-

portabilă este mică, de ordinul câtorva sute de biţi cel mult. Metoda este greu de aplicat pentru informaţii secrete, deoarece informaţia este afişată pe ecranul sistemului sursă şi ca urmare este vizibilă persoanelor din apropiere. Informaţiile astfel transportate sunt de obicei dispersiile criptografice ale unor chei publice.

3. Omul inventează o parolă şi 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 şi ca generator de „numere“ aleatoare.

Ne vom ocupa în continuare de aspecte privind implementarea metodelor 2 şi 3.

Metoda 2 necesită o scriere a unui şir de biţi într-o formă uşor de citit de către un om. Trebuie ţinut 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-şi poată lua puncte de reper pentru a şti unde a rămas. Ca urmare, un grup de cifre sau simboluri fără legătură intre ele (adică care nu constituie un cuvînt 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 aceeaşi dispersie *md5* este scrisă ca 8 grupuri de câte 4 cifre, cu spaţiu sau alt separator între grupuri, devine mult mai uşor de citit. Pentru şiruri mai lungi, devine necesar un al doilea nivel de grupare.

O altă metodă, mai dificil de pus în practică, este transformarea şirului de biţi într-un şir de cuvinte dintr-un dicţionar standard sau într-un şir de silabe ce alcătuiesc cuvinte, probabil fără sens, dar pronunţabile. Pentru un dicţionar de 4096 cuvinte, un cuvânt codifică 12 biţi. O dispersie *md5* poate fi scrisă ca un şir de 11 cuvinte. Un astfel de şir de cuvinte poate fi memorat mai uşor decât secvanţa de 32 cifre hexa echivalentă.

Pentru a aplica metoda 3, remarcăm pentru început că o parolă inventată şi 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 biţi pe caracter. Se estimează că un text în limbaj natural nu are mai mult de 1–2 biţi 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 şi cheie. Acest „concentrator de entropie“ poate fi o funcţie de dispersie (nu neapărat criptografică).

- Parola trebuie să fie suficient de lungă (şi de aleator aleasă) ca să furnizeze efectiv biţii de entropie necesari. O frază în limbaj natural, utilizată pentru derivarea unei chei de 128 biţi, 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, micşorarea 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 aşa fel încât să nu permită unui adversar să facă încercarea exhaustivă a cheilor pe maşina lui (unde poate dispune de putere mare de calcul şi nu lasă urme) ci să trebuiască să contacteze una din maşinile care cunosc cheia (maşini care înregistrează tentativa şi 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, şi *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 biţi este suficientă. O parolă bună, în acest context, trebuie să aibă doar 5–6 cuvinte, sau, dacă conţine cifre şi punctuaţie, 10–12 caractere.

6.4. Numere aleatoare

Un sistem criptografic utilizează numere aleatoare, în diverse scopuri:

- generarea cheilor,
- generarea vectorilor de iniţializare sau a *salt*-urilor,
- generarea numerelor unice (*nonce*).

Numerele aleatoare generate trebuie să fie uniform distribuite şi independente unul de altul. În plus faţă de alte aplicaţii, numerele aleatoare pentru scopuri criptografice trebuie să nu poată fi prevăzute sau influenţate de un eventual adversar. (Condiţii 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 funcţionează 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 şirul 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, şi 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 substanţe radioactive sau curenţii de aer din interiorul carcasei unui harddisc. Dispozitivele dedicate sunt cele mai sigure şi relativ rapide. Pe de altă parte, utilizatorul este nevoit să aibă încredere că producătorul dispozitivului l-a construit corect şi nu a instalat o „uşă dosnică“, permiţându-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 apăsate două taste consecutive şi să exprimăm printr-un număr durata respectivă. Vom constata că cifrele cele mai puţin semnificative ale numărului respectiv sunt destul de aleatoare. Prin anumite prelucrări, se poate extrage de aici un şir 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 reţea (însă aici trebuie multă grijă, deoarece pachetele primite de placa de reţea pot fi supravegheate de adversar), un ceas (de fapt, două ceasuri independente, exploatând fluctuaţia derivei relative a celor două ceasuri).

Dispozitivele nededicate au ca avantaj preţul mai redus şi riscul mai mic de-a fi „măsluite“ de către fabricant. Debitul de biţi aleatori produşi variază însă în funcţie de utilizarea sistemului; în particular, pentru un server care nu admite utilizatori locali este dificil de obţinut un debit satisfăcător de biţi aleatori.

6.4.2. Generatoare de numere pseudoaleatoare

Un generator de numere pseudoaleatoare funcţionează astfel:

- În fiecare moment t, generatorul are asociată o *stare internă* s_t. Starea internă este o valoare dintr-o mulţime arbitrară suficient de mare.
- Atunci când este nevoie de un număr aleator, acest număr este obţinut aplicând o funcţie asupra stării interne. Numărul produs este deci $x_t =$

$f(s_t)$, unde f este o funcţie fixată.

- După producerea unui număr aleator, starea internă este modificată, pentru ca următorul număr să nu mai aibă aceeaşi valoare. Actualizarea stării interne se face pe baza unei a doua funcţii, g. Avem deci $s_{t+1} = g(s_t)$.

Dacă funcţiile f şi g „amestecă“ suficient de bine biţii, şirul pseudoaleator produs, $(x_t)_{t\in\mathbb{N}}$ are proprietăţi statistice suficient de apropiate de numerele cu adevărat aleatoare.

De remarcat că întregul şir depinde de valoarea stării iniţiale s_0 a generatorului; pentru aplicaţii non-criptografice şi pentru teste, acest lucru este bun deoarece face comportamentul programelor reproductibil.

De asemenea, trebuie remarcat că, deoarece, în practică, mulţimea stărilor interne posibile este finită, mai devreme sau mai târziu starea internă se repetă şi, ca urmare, întregul şir pseudoaleator este periodic. Perioada şirului pseudoaleator este cel mult egală cu numărul de stări interne posibile. Pentru o funcţie g cu comportament apropiat de o funcţie aleatoare sau de o permutare aleatoare, perioada şirului este de ordinul rădăcinii pătrate din numărul de stări interne posibile.

Pentru scopuri criptografice, sunt necesare câteva proprietăţi suplimentare:

- Un adversar care cunoaşte funcţiile f şi g utilizate şi cunoaşte o parte dintre elementele şirului pseudoaleator (x_t) să nu poată calcula alte elemente ale şirului pseudoaleator. O condiţie necesară pentru aceasta este ca funcţia f să fie rezistentă la preimagine. O altă condiţie necesară este ca mulţimea stărilor interne posibile să fie suficient de mare pentru ca explorarea ei prin forţă brută să nu fie posibilă practic.
- Starea iniţială s_0 trebuie să fie creată pornind de la un generator fizic. În caz contrar, starea iniţială este previzibilă pentru un adversar şi, ca urmare, întregul şir este previzibil.
- Este bine ca, dacă un adversar reuşeşte să obţină starea internă s_t, să nu poată calcula numerele pseudoaleatoare deja generate (adică $x_0 \dots x_{t-1}$). Aceast lucru este util pentru ca, dacă un adversar reuşeşte să spagră un calculator, să nu poată decripta comunicaţiile anterioare. Pentru a îndeplini această cerinţă, este necesar ca şi g să fie rezistentă la preimagine.

Generatoarele de numere pseudoaleatoare utilizate în practică nu se bazează pe generatoare fizice doar pentru starea iniţială s_0, ci modifică starea

internă şi în timpul funcţionării generatorului, pe măsură ce obţin biţi aleatori de la generatorul fizic. Acest lucru are scopul ca, dacă un adversar reuşeşte să obţină starea internă la un moment dat sau să afle starea internă iniţială, să nu poată prevedea decât o mică parte dintre numerele aleatoare produse ulterior. De asemenea, starea iniţială nu este generată la pornirea calculatorului, întrucât, de obicei, în acel moment nu sunt disponibili prea mulţi biţi aleatori de la generatoarele fizice. Starea internă este salvată la oprirea calculatorului, într-un fişier ce nu poate fi citit de utilizatorii obişnuiţi, şi 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, fişierul special `/dev/random` oferă numere aleatoare de la un generator fizic bazat pe acţiunile utilizatorului. De asemenea, fişierul special `/dev/urandom` oferă numere pseudoaleatoare utilizabile pentru aplicaţii criptografice.
- Sistemul Windows oferă funcţia `CryptGenRandom()`, declarată în fişierul antet `wincrypt.h`.

De remarcat că utilizarea unor funcţii 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. Soluţia trivială este ţinerea unei baze de date cu corespondenţa nume, parolă. Această soluţie (stocarea parolelor în clar) are un dezavantaj: un adversar care reuşeşte să spargă sistemul sau un administrator indiscret poate obţine direct parolele tuturor utilizatorilor. „Recolta“ este valoroasă dacă utilizatorii folosesc aceleaşi parole şi pe alte sisteme.

O îmbunătăţire a securităţii 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 şi verifică, pentru parola introdusă p', dacă $h(p') = s$. Deoarece h este rezistentă la preimagine, probabilitatea ca un adversar, care nu cunoaşte parola p, să găsească o parolă p' satisfăcând $h(p') = s$ este neglijabil de mică.

Soluţia are în continuare o slăbiciune, legată de faptul că un adversar care obţine s poate obţine p printr-un dicţionar creat off-line: adversarul ia o mulţime de parole şi, pentru fiecare parolă, calculează şi memorează dispersia, obţinând astfel un dicţionar care asociază dispersiei parola corespunzătoare. Înarmat cu un astfel de dicţionar, un adversar care obţine s poate regăsi foarte rapid p dacă p era în dicţionarul încercat.

Metoda de mai sus poate fi îmbunătăţită, împiedicând crearea unui dicţionar de dispersii şi obligând adversarul să facă toată munca de spargere a parolelor după obţinerea lui s. Conform noii metode, la iniţializarea parolei, sistemul generează un număr aleator r şi scrie în fişierul 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 dicţionar ar avea nevoie de un număr de dispersii egal cu produsul dintre numărul de parole de încercat şi 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 obţină parola cu care se conectează un utilizator la acel sistem. De exemplu, adversarul poate înlocui programul obişnuit 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 reţea, 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 reţea şi orice adversar care cunoaşte $h(p)$ poate impersona utilizatorul, fără a avea nevoie de p

6.5.2. Parole de unică folosinţă

O *parolă de unică folosinţă* (engl. *One Time Password*) este o parolă care este acceptată de sistem ca fiind parolă validă cel mult o dată.

Una din aplicaţiile parolelor de unică folosinţă este conectarea la un sistem, în prezenţa unui adversar pasiv, fără a recurge la criptare. Notăm că aplicativitatea este foarte limitată:

- comunicaţia nefiind criptată, metoda este inaplicabilă dacă datele trans-

mise trebuie să rămână secrete;

- un adversar activ poate „deturna“ conexiunea după deschidere şi poate da orice comenzi în numele utilizatorului conectat.

Unul dintre sistemele de parole de unică folosinţă este descris în continuare. În cele ce urmează, h este o dispersie rezistentă la preimagine, iar $h^n(x) = h(h(\ldots h(x)\ldots))$.

1. Utilizatorul alege o *parolă primară*, de lungă durată, p.
2. La iniţializarea parolei pe sistem, sistemul generează şi afişează un număr aleator, nesecret, r. De asemenea, sistemul afişează un număr de iteraţii, 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 doreşte să-şi configureze autentificarea.
4. Sistemul memorează în baza de date ansamblul (p_n, n, r).
5. La prima conectare, sistemul afişează r şi $n - 1$ şi cere utilizatorului să calculeze şi să introducă parola de unică folosinţă $p_{n-1} = h^{n-1}(p \cdot r)$. Sistemul verifică parola de unică folosinţă testând dacă $h(p_{n-1}) = p_n$. Apoi sistemul înlocuieşte 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 folosinţă. În particular, calculatorul care autentifică utilizatorul nu obţine niciodată şi nici nu poate deduce parola primară. Administratorul unui astfel de calculator nu poate impersona utilizatorul pe un alt calculator pe care utilizatorul utilizează aceeaşi parolă primară.

Dezavantajul sistemului, faţă de parola clasică, este că utilizatorul are nevoie de un dispozitiv de calcul în vederea calculării parolelor de unică folosinţă. Proceduri de lucru pentru utilizator pot fi:

- Utilizatorul rulează local un program pentru calculul parolelor de unică folosinţă. 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 folosinţă şi 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 şi mai ales nu permite aflarea, de către adversar, a parolei primare.
- Utilizatorul foloseşte un dispozitiv de calcul dedicat. Aceasta este metoda cea mai sigură, dar şi cea mai scumpă.

Reţele de calculatoare

Protocoale

Radu-Lucian Lupşa

Aceasta este ediţia electronică a cărţii *Reţele de calculatoare*, publicată la Casa Cărţii de Ştiinţă, în 2008, ISBN: 978-973-133-377-9.

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

Cuprins

Principii

Protocoale

Capitolul 7

Codificări de interes practic

7.1. Probleme privind reprezentarea numerelor întregi

Până aici am privit un mesaj transmis printr-o reţea ca un şir de simboluri dintr-un alfabet finit. Între simbolurile ce alcătuiesc şirul se stabileşte o *ordine*, existând un prim element al şirului, un al doilea, etc. Transmisia elementelor se face în ordinea în care apar ele în şir, primul simbol transmis fiind cel care ocupă prima poziţie din şir. Până aici am considerat că transmisia unui şir de la un dispozitiv la altul conservă ordinea între elemente.

Aşa cum vom vedea însă în paragraful de faţă, din raţiuni legate de standardizarea reprezentării numerelor întregi, transmiterea unui şir nu conservă întotdeauna ordinea elementelor. În cele ce urmează, vom examina interferenţele între reprezentarea numerelor în calculator şi ordinea simbolurilor ce alcătuiesc un mesaj. Vom oferi cititorului o perspectivă, inspirată din [Cohen 1980] şi mai puţin întâlnită în alte lucrări, asupra relaţiilor dintre biţi, octeţi şi reprezentarea numerelor.

7.1.1. Reprezentări pe biţi

În paragraful de faţă vom face abstracţie de anumite complicaţii constructive ale sistemelor de calcul reale. Vom considera reprezentări pe biţi, ignorând deocamdată aspectele legate de gruparea biţilor în octeţi şi de faptul că, în memoria calculatorului, adresele identifică octeţi şi nu biţi.

Ca urmare, rugăm cititorul să uite, pentru moment, noţiunea 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 şi naturaleţe construcţiei matematice. Pe de altă parte, din punct de vedere practic, al construcţiei echipamentelor fizice, dispozitive cu două stări stabile sunt mult mai uşor de construit decât dispozitive cu mai multe stări.

În scris, cele două simboluri sunt notate în mod obişnuit cu 0 şi 1. Atragem atenţia că:

- Alegerea celor două simboluri utilizate, precum şi a corespondenţei dintre starea dispozitivului fizic şi simbolul asociat, poate fi făcută oricum. Odată însă făcută o alegere, aceasta trebuie respectată de toate entităţile implicate în comunicaţie.
- 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 numeşte *bit*, de la *binary digit* (rom. *cifră binară*).

7.1.1.2. Şiruri de biţi

În cadrul unui şir de biţi, avem nevoie să identificăm fiecare bit al şirului. Pentru aceasta, se stabileşte o *ordine* a biţilor: avem un prim bit, un al doilea bit, etc. Trebuie remarcat însă că ordinea este o convenţie: nu există o legătură directă între ordinea convenţională a unui şir de biţi şi amplasamentul dispozitivelor fizice care memorează acei biţi. Numărul de ordine al unui bit, în cadrul acestei ordini convenţionale, se numeşte în mod obişnuit *poziţia* (sau, eventual, *adresa* sau *deplasamentul*) bitului. Numerotarea poziţiilor 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 şir de biţi, este natural ca primul bit transmis, considerând ordinea cronologică, să fie primul bit al şirului (poziţia 0). La memorarea unui şir într-o memorie cu acces direct (memorie RAM sau fişier pe disc), este natural ca în celula cu adresa cea mai mică, dintre celulele alocate şirului, să fie plasat primul bit al şirului (bitul de pe poziţia 0). În acest fel, *primul bit* al unui şir înseamnă, simultan, bitul de pe poziţia (convenţională) 0, bitul transmis primul (cronologic) şi bitul memorat la adresa cea mai mică.

7.1.1.3. Reprezentarea pe biţi a numerelor întregi

Reprezentarea numerelor naturale prin şiruri de biţi se bazează pe ceea ce matematicienii numesc *reprezentare poziţională în baza 2*, pe care o presupunem cunoscută. În reprezentarea într-o bază de numeraţie, distingem cifra unităţilor, 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 şi poziţiile lor în şir:

1. *Primul bit al şirului este cifra de pondere maximă.* Aceasta alegere este identică celei utilizate în scrierea numerelor în limbile „obişnuite“ (indo-europene), cu scriere de la stânga spre dreapta. Se mai numeşte *big endian*.

   În această schemă de reprezentare, un şir de biţi $b_0 b_1 \ldots b_{n-1}$ reprezintă numărul

   $$b_0 \cdot 2^{n-1} + b_1 \cdot 2^{n-2} + \ldots + b_{n-1} \cdot 2^0.$$

2. *Primul bit al şirului este cifra de pondere 1.* Această reprezentare este asemănătoare scrierii numerelor în limbile semite (araba şi ebraica, cu scriere de la dreapta spre stânga şi unde numerele sunt scrise tot cu cifra unităţilor în dreapta). Se mai numeşte *little endian*.

   Această alegere are avantajul unei scrieri mai simple a relaţiei dintre valoarea numărului şi şirul de biţi: valoarea unui număr reprezentat pe n biţi este

   $$b_0 \cdot 2^0 + b_1 \cdot 2^1 + \ldots + b_{n-1} \cdot 2^{n-1}.$$

Fiind două scheme de reprezentare distincte, dacă un sistem transmite un număr în reprezentare *little endian*, iar celălalt sistem interpretează şirul de biţi primit ca fiind într-o reprezentare *big endian*, receptorul „înţelege“ alt număr decât cel transmis de emiţător. 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 şirul de biţi 11001, în care am scris primul bit (în sensul din § 7.1.1.2) pe poziţia cea mai din stânga.

Dacă acest şir 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 distincţia dintre schema de reprezentare *big endian* si schema *little endian* există numai acolo unde pe de o parte avem o ordine a a biţilor 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 octeţi

În paragraful precedent, am ignorat în mod deliberat noţiunea de *octet* (*byte*) şi 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 biţilor în octeţi şi de faptul că, în general, ordinea biţilor în octeţi nu este vizibilă programatorului.

7.1.2.1. Octeţi

În memoria calculatoarelor, biţii sunt grupaţi în grupuri de dimensiune fixă, în mod obişnuit câte 8 biţi. Un astfel de grup se numeşte *octet* (denumire intrată pe filieră franceză) sau *bait* (adaptare a englezescului *byte*).

Un octet poate fi privit în două moduri distincte:

- ca un şir de 8 biţi,
- ca un număr întreg cuprins între 0 şi 255.

Echivalenţa între aceste două moduri de-a privi un octet este o problemă ce necesită multă atenţie. Dacă identificăm biţii după ponderile lor, există o corespondenţă biunivocă între un astfel de grup de 8 biţi şi un număr întreg între 0 şi 255. Dacă însă identificăm biţii după poziţia lor în şirul de 8 biţi (bitul 0, bitul 1,..., bitul 7), atunci corespondenţa biunivocă între număr şi şir de biţi există doar după ce am stabilit o corespondenţă între poziţia unui bit în şir şi ponderea sa (*big endian*, *little endian* sau eventual o corespondenţă mai complicată).

După modul în care se identifică biţii unui octet în definiţia operaţiilor efectuate de diverse componente ale unui sistem de calcul, operaţiile se pot împărţi în trei categorii:

1. *Operaţii pentru care biţii se identifică după pondere* sau, echivalent, octetul este privit ca număr. Aici se încadrează operaţiile aritmetice şi operaţiile de deplasare pe biţi. De remarcat că operaţiile *deplasare la stânga* (engl. *shift left*), respectiv *deplasare la dreapta* (engl. *shift right*) pot fi descrise în termeni de operaţii aritmetice: deplasarea la stânga este o înmulţire cu 2, iar deplasarea la dreapta este o împărţire la 2. În acest context, „spre stânga“ şi „spre dreapta“ înseamnă spre poziţiile cu pondere mai mare, respectiv mai mică, neavând nici o legătură cu

primele sau cu ultimele poziţii. Aceste operaţii sunt efectuate de unitatea aritmetică din microprocesorul calculatorului.

2. *Operaţii pentru care biţii sunt identificaţi după numărul lor de ordine* sau, echivalent, octetul este privit ca un şir arbitrar de biţi, fără a avea asociată o valoare numerică. Aici intră transmiterea bit cu bit (transmitere serială) a unui octet. Această operaţie este efectuată de placa de reţea şi de alte adaptoare seriale (de exemplu, adaptoarele USB). Tot aici s-ar încadra, dacă ar exista, o operaţie de obţinere sau de modificare a unui bit (al octetului) identificat prin numărul său de ordine. O asemenea operaţie nu este oferită, în mod normal, într-un sistem de calcul — nu există o instrucţiune care să extragă, de exemplu, bitul numărul 5 dintr-un octet.
3. *Operaţii care pot fi definite fie identificând biţii după ponderea lor, fie identificând biţii după numărul lor de ordine.* În această categorie se încadrează transmiterea unui octet ca un tot unitar, verificarea egalităţii a doi octeţi şi operaţiile logice pe bit (*şi*, *sau*, *sau exclusiv* şi negaţia).

   Pentru oricare dintre aceste operaţii, dacă definim operaţia identificând biţii după numărul lor de ordine, efectul ei asupra valorii numerice a octetului nu depinde de corespondenţa aleasă între poziţiile biţilor şi ponderile lor.

În aceste condiţii, în interiorul unui calculator, biţii din cadrul unui octet sunt identificaţi după ponderea lor. În construcţia 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 poziţie cu aceeaşi 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 reţelei, biţii sunt transmişi secvenţial, biţii unui octet sunt aici identificaţi prin numărul lor de ordine in cadrul transmisiei. Pentru a păstra valoarea octetului în timpul transmisiei prin mediul reţelei, corespondenţa dintre numărul de ordine al unui bit şi ponderea sa (*little endian* sau *big endian*) trebuie să facă parte din specificaţiile protocolului de nivel fizic al reţelei.

Numerotarea biţilor unui octet intervine, de asemenea, în descrierea unor scheme de reprezentare a datelor unde un număr este reprezentat pe un grup de biţi ce nu formează un număr întreg de octeţi. Este cazul schemelor de reprezentare pentru structuri de date ce conţin câmpuri de 1 bit, 2 biţi, 12 biţi, etc. Şi aici este necesar să se specifice dacă numerotarea biţilor 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. Şiruri de octeţi

Ca şi în cazul biţilor (vezi § 7.1.1.2), şi cu octeţii putem construi şiruri. În cadrul unui şir de octeţi, octeţii sunt aşezaţi într-o ordine, existând un prim octet (numerotat ca octetul 0), al doilea octet (poziţia 1), etc. La transmisia printr-o legătură în reţea, primul octet al şirului 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 şir de n octeţi poate fi, la rândul lui, privit ca:

- un şir de $8n$ biţi,
- un şir de n numere, fiecare cuprins între 0 şi 255.

Pentru a putea privi un şir de n octeţi ca un şir de $8n$ biţi, este necesar să avem o numerotare, bine definită, a biţilor în cadrul unui octet. Rezultă un şir de biţi în care între poziţia p_B a unui bit în şirul de $8n$ biţi, poziţia p_{BO} a bitului în cadrul octetului în care se găseşte şi poziţia p_O a acelui octet în şirul de octeţi are loc relaţia:

$$p_B = 8 \cdot p_O + p_{BO}. \tag{7.1}$$

Relaţia de mai sus are această formă simplă dacă se utilizează numerotare de la 0; pentru numerotarea de la 1, forma relaţiei e mai complicată.

Transmiterea unui şir de octeţi printr-o conexiune, precum şi memorarea şirului într-un fişier pe disc urmată de citirea lui înapoi în memorie, păstrează ordinea şi valorile octeţilor din şir. Valorile octeţilor sunt păstrate dacă privim octeţii ca numere între 0 şi 255; dacă privim octeţii ca şiruri de 8 biţi, valorile octeţilor se păstrează numai dacă pe ambele sisteme utilizăm aceeaşi corespondenţă între numerele de ordine şi ponderile biţilor.

7.1.2.3. Reprezentarea numerelor pe un număr întreg de octeţi

Cel mai mare număr ce poate fi reprezentat pe un octet este 255, ceea ce este mult prea puţin pentru majoritatea aplicaţiilor. Pentru a putea reprezenta numere din intervale mai largi, sunt necesare scheme de reprezentare pe mai mult de 8 biţi. Schemele cele mai simple sunt cele care utilizează un număr întreg de octeţi; acestea vor fi prezentate în continuare. Schemele de reprezentare ce utilizează şiruri de biţi ce nu formează neapărat un număr întreg de octeţi vor fi studiate în § 7.1.2.4.

Deoarece un octet are valoarea între 0 şi 255, putem considera fiecare octet ca fiind o cifră în baza 256. Reprezentarea unui număr printr-un şir de octeţi se face ca reprezentare poziţională 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 octeţi ai reprezentării), al doilea octet are ponderea 256^{n-2} ş. a. m. d., penultimul octet are ponderea 256, iar ultimul octet are ponderea 1.

Reamintim că prin *primul octet* înţelegem octetul care este transmis primul, în ordine cronologică, de la un dispozitiv la altul şi, totodată, octetul memorat la adresa cea mai mică.

Exemplul 7.2: Descriem mai jos reprezentarea numărului 300 în schemele de reprezentare *little endian* şi *big endian*, pe 2 şi pe 4 octeţi. Pentru fiecare dintre aceste patru scheme de codificare, este dat şirul de octeţi ce reprezintă numărul 300.

poziţie octet (nr. ordine)	Valorile octeţilor pentru diverse reprezentări			
	2 octeţi big endian	2 octeţi little endian	4 octeţi big endian	4 octeţi 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 octeţi în format *big endian*, valoarea numărului reprezentat se regăseşte ca valoarea octetului 0 înmulţită cu 256 plus valoarea octetului 1, anume: $1 \cdot 256 + 44 = 300$. În cadrul reprezentării pe 4 octeţi 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 operaţii aritmetice asupra numerelor reprezentate pe 2 octeţi sau, pentru unele calculatoare, pe 4 sau 8 octeţi. Pe unele calculatoare, unitatea aritmetică lucrează cu numere reprezentate după sistemul *big endian*; pe alte calculatoate, unitatea

aritmetică cere reprezentare *little endian*. După acest criteriu, calculatoarele ale căror unităţi aritmetice lucrează cu numere reprezentate pe mai mult de un octet se împart în calculatoare *little endian* şi calculatoare *big endian*.

Variabilele de tip întreg, în majoritatea limbajelor de programare, sunt reprezentate pe 2, 4 sau 8 octeţi, în ordinea fixată de unitatea aritmetică.

Este posibilă utilizarea, pentru anumite variabile întregi, a unei reprezentări diferite de cea a unităţii aritmetice. De asemenea, se pot utiliza reprezentări pe mai mulţi octeţi decât permite unitatea aritmetică. La manipularea acestor variabile, programatorul trebuie să aibă în vedere că operaţiile 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 şi numărul de octeţi 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 fişiere prevăd formate *little endian*. Există şi formate (de exemplu, formatul TIFF pentru imagini, formatele UTF-16 şi UTF-32 pentru texte) care permit emiţătorului să aleagă formatul dorit şi prevăd un mecanism prin care emiţătorul informează receptorului despre alegerea făcută.

Dacă formatul de pe disc sau de pe conexiune coincide cu formatul unităţii aritmetice locale, un program poate transfera direct şiruri de octeţi între o variabilă întreagă locală şi fişierul de pe disc sau, respectiv, conexiunea spre celălalt calculator. Dacă formatul de pe disc sau de pe conexiune este invers faţă de cel local, un program care transferă date trebuie să inverseze ordinea octeţilor imediat înainte de scrierea pe disc sau de trimiterea pe conexiune, precum şi imediat după citirea de pe disc sau recepţionarea de pe conexiune.

7.1.2.4. Reprezentarea numerelor pe un şir arbitar de biţi

Ne vom ocupa în continuare de metode de reprezentare, pentru numere întregi, în care biţii ce intră în reprezentarea unui număr nu formează neapărat un număr întreg de octeţi. 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 şi § 7.1.1.3). Pentru ca o astfel de schemă să fie complet definită, este necesar să fie stabilită (standardizată) corespondenţa dintre poziţia fiecărui bit din reprezentare şi ponderea asociată. În descrierea

unei astfel de scheme de reprezentare, trebuie precizate trei lucruri:

- dacă reprezentarea numărului prin şirul de biţi se face după metoda *big endian* sau *little endian*;
- dacă numerotarea biţilor 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 biţilor în cadrul şirului de biţi se face după relaţia (7.1) sau după o altă metodă.

Într-o schemă de reprezentare „raţională“, 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 relaţia (7.1).

Rezultă astfel două metode coerente:

- *big endian.* În această metodă, numerotarea biţilor începe cu cel mai semnificativ bit al primului octet, iar după cel mai puţin semnificativ bit al primului octet urmează cel mai semnificativ bit al celui de-al doilea octet. Orice număr, indiferent pe câţi biţi s-ar reprezenta, se reprezintă în format *big endian.*
- *little endian.* În această metodă, numerotarea biţilor începe cu cel mai puţin semnificativ bit al primului octet, iar după cel mai semnificativ bit al primului octet urmează cel mai puţin semnificativ bit al celui de-al doilea octet. Orice număr, indiferent pe câţi biţi s-ar reprezenta, se reprezintă în format *little endian.*

În cadrul acestor două metode, dacă reprezentăm un număr folosind un şir de biţi ce formează un număr întreg de octeţi, formatele de reprezentare rezultate coincid cu formatele de reprezentare pe octeţi studiate în paragraful precedent.

Exemplul 7.3: Considerăm o schemă de reprezentare pentru două numere întregi, a şi b, în care a se reprezintă pe 4 biţi şi b se reprezintă pe 12 biţi. În total avem $4 + 12 = 16$ biţi, adică 2 octeţi.

Dacă alegem metoda *big endian*, schema de reprezentare va utiliza cei mai semnificativi 4 biţi ai primului octet pentru a-l reprezenta pe a, ceilalţi 4 biţi ai primului octet vor fi cei mai semnificativi 4 biţi ai lui b, iar cel de-al doilea octet va conţine cei mai puţin semnificativi 8 biţi din b. Această schemă de reprezentare este ilustrată în figura 7.1, cu valori concrete $a = 11$ şi $b = 300$.

Dacă alegem metoda *little endian*, schema de reprezentare va utiliza cei mai puţin semnificativi 4 biţi ai primului octet pentru a-l reprezenta pe a, ceilalţi 4 biţi ai primului octet vor fi cei mai puţin semnificativi 4 biţi ai lui b,

iar cel de-al doilea octet va conţine cei mai semnificativi 8 biţi din b. Această schemă de reprezentare este ilustrată în figura 7.2, cu valori concrete $a = 11$ şi $b = 300$.

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 şir de biţi

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 octeţilor. La scrierea valorilor biţilor în octet (coloana din mijloc) s-a utilizat convenţia obişnuită, de-a scrie cifrele mai semnificative în stânga.

Figura 7.1: Reprezentare *big endian* pentru numărul 11 pe 4 biţi urmat de numărul 300 reprezentat pe 12 biţi (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 şir de biţi

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 octeţilor. La scrierea valorilor biţilor în octet (coloana din mijloc) s-a utilizat convenţia obişnuită, de-a scrie cifrele mai semnificative în stânga.

Figura 7.2: Reprezentare *little endian* pentru numărul 11 pe 4 biţi urmat de numărul 300 reprezentat pe 12 biţi (exemplul 7.3).

Un alt exemplu în care avem de-a face cu numere reprezentate pe şiruri arbitrare de biţi este legat de aşa-zisa *codificare în baza 64*, descrisă în § 7.4.2.

7.1.3. Probleme privind reprezentarea lungimii şirurilor

Oridecâteori se transmite un şir de obiecte, este necesar ca receptorul să poată determina numărul de obiecte transmise. Acest lucru este valabil

indiferent de natura obiectelor: biţi, octeţi, cifre zecimale, caractere ale unui text, numere în cadrul unui şir 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 acelaşi. Metoda este utilizată frecvent la memorarea unui şir în memoria RAM sau pe disc, deoarece permite alocarea de la început a memoriei pentru reprezentarea lui şi permite accesul direct la datele memorate după şirul în discuţie. Dezavantajul principal al metodei este acela că dimensiunea fixată trebuie astfel aleasă încât să fie suficientă în orice caz ce poate să apară la execuţie. De asemenea, trebuie să existe o valoare potrivită pentru poziţiile „neutilizate“ din şir; de exemplu, în reprezentarea numerelor, poziţiile cele mai semnificative se completează cu zerouri.

  Deoarece la transmisia printr-o conexiune nu se poate pune problema accesului direct (adică altfel decât secvenţial) la date, metoda este puţin utilizată în transmisia datelor prin reţea. Şiruri de lungime fixă se utilizează la reprezentarea binară a numerelor.
- *Numărul de obiecte este transmis separat, în faţa şirului.* Această metodă uşurează munca receptorului, care ştie exact ce să astepte şi poate aloca memorie pentru recepţionarea datelor. În schimb, munca emiţătorului este complicată prin faptul că acesta trebuie să cunoască de la început numărul de obiecte din şir. Acest fapt face metoda inaplicabilă în anumite situaţii.

  Transmiterea de la început a numărului de obiecte este utilizată, de exemplu, de protocolul *HTTP* (§ 11.3.2) la transmiterea, de către server, a conţinutului paginii cerute de client. Serverul transmite întâi numărul de octeţi ai paginii şi apoi şirul de octeţi ce formează pagina.
- *După ultimul obiect din şirul propriu-zis, se transmite o valoare specială, cu rol de* terminator. Această metodă uşurează munca emiţătorului, care poate să înceapă transmiterea şirului înainte de-a şti câte elemente are, în schimb îngreunează munca receptorului, care trebuie să citească elementele şirului unu câte unu şi 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 şir de caractere. Rolul de terminator poate fi acordat caracterului *null* (caracterul cu codul ASCII zero), caracterului *newline* (sfârşit de rând), caracterului

spaţiu, etc. Orice alegere s-ar face, caracterul sau caracterele astfel alese pentru a marca sfârşitul unui şir nu pot să apară în şirul propriu-zis. Ca urmare, transmiterea unui fişier cu conţinut arbitrar (şir de octeţi cu valori arbitrare) nu se poate face prin metoda cu terminator (decât dacă un octet al fişierului se codifică pe mai mult de 8 biţi).

7.1.4. Alte metode de reprezentare a numerelor întregi

Schemele de reprezentare (formatele) pentru numere întregi, studiate în § 7.1.2.3 şi § 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 (obişnuite) a numărului.
- Formatul *binar-zecimal*, numit şi *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 biţi conform metodelor din § 7.1.2.4.

Descriem puţin mai pe larg reprezentarea numerelor în format text, deoarece o astfel de reprezentare se utilizează frecvent în protocoalele în reţea. Motivul principal al utilizării formatului text este uşurinţa depanării aplicaţiilor ce utilizeaza astfel de protocoale: comunicaţia poate fi înregistrată într-un fişier şi examinată cu un program obişnuit pentru vizualizarea fişierelor text.

În format text, se utilizează convenţiile de reprezentare a numerelor în textele scrise: se începe cu cifra cea mai semnificativă, numărul de cifre este variabil (depinde de valoarea numărului) şi prima cifră (cea mai semnificativă) scrisă nu este zero, cu excepţia 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, 30_{16}).

Deoarece lungimea reprezentării este variabilă, este necesar să fie transmisă sub o formă sau alta informaţia privind lungimea reprezentării numărului (numărul de cifre). În acest scop, în reprezentările text, numerele sunt separate de obicei prin spaţii, caractere *tab*, caractere *newline* etc.

Exemplul 7.4: Redăm mai jos reprezentările text ASCII terminat cu spaţiu, BCD *big endian* pe 4 octeţi şi BCD *little endian* pe 4 octeţi, pentru numărul 300. Valorile octeţilor sunt scrise în baza 2, bitul cel mai semnificativ fiind scris în stânga.

poziţie octet (nr. ordine)	Valorile octeţilor		
	text ASCII	BCD *big endian*	BCD *little endian*
0	00110011	00000000	00000000
1	00110000	00000000	00000011
2	00110000	00000011	00000000
3	00100000	00000000	00000000

7.2. Codificarea textelor

Prin text înţelegem 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 punctuaţie, cifre şi diferite alte semne grafice. Nu se face distincţie între diferitele variante de-a scrie o aceeaşi 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 faţă de litera simplă din care provine sau să fie format din caracterul corespunzător literei respective fără semne diacritice şi un caracter de control care să marcheze semnul diacritic adăugat. În aceeaşi idee, s-ar putea face şi distincţia dintre literele mari (majuscule) şi literele mici (minuscule) corespunzătoare tot pe baza unor caractere de control cu rol de modificator.

Operaţiile efectuate asupra textelor, care trebuie să fie permise de codificarea aleasă, sunt în principal următoarele:

- afişarea textului;
- concatenarea unor texte sau alte operaţii de sinteză;

- căutarea unui cuvânt, extragerea unor cuvinte sau unor fragmente de text şi diverse alte operaţii de analiză a textului;
- sortarea alfabetică.

De notat că regulile de sortare alfabetică sunt complexe şi 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*, *şiret*; de notat că orice cuvânt ce începe cu *ş* 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* şi *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 şir de biţi sau ca şir de octeţi.

Pentru o schemă de codificare trebuie aşadar specificate trei elemente:

- setul de caractere;
- numerotarea (codificarea) caracterelor;
- reprezentarea pe biţi sau pe octeţi 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 *spaţiu* (considerat de unii ca fiind caracter imprimabil şi de alţii ca fiind caracter de control);
- 94 de caractere imprimabile, cuprinzând: 52 de litere (cele 26 litere ale alfabetului latin, cu cele două forme, majuscule şi minuscule) cele 10 cifre zecimale şi un număr de 32 de semne de punctuaţie şi alte simboluri.

Codurile asociate caracterelor ASCII sunt cuprinse între 0 şi 127, caracterele de control primind codurile 0–31 şi 127, spaţiul are codul 32, iar

(celelalte) caractere imprimabile au codurile cuprinse între 33 şi 126. Pentru a uşura 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 şi că diferenţa dintre codul oricărei litere mici şi codul literei mari corespunzătoare este 32 (20 hexa).

Pentru reprezentarea unui caracter ASCII sunt necesari doar 7 biţi, 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 aplicaţiilor, rămân multe coduri reprezentabile (cca. 140) care nu sunt utilizate. Se poate extinde setul de caractere, asociind noilor caractere coduri între 128 şi 255 sau coduri între 0 şi 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 şi numere de la 128 la 255 pentru caracterele suplimentare.

Reprezentarea pe octeţi pentru un text ISO-8859-n se face cu câte un octet pentru fiecare caracter, octetul conţinând codul caracterului.

Fiecare cod din familie este extensie a codului ASCII în sensul că mulţimea caracterelor din fiecare astfel de cod include mulţimea caracterelor

Caracter	Cod (hexa)	Caracter	Cod (hexa)
Ă	C3	ă	E3
Â	C2	â	E2
Î	CE	î	EE
Ş	AA	ş	BA
Ţ	DE	ţ	FE

Tabelul 7.1: Caracterele cu diacritice din alfabetul limbii române şi codificările ISO-8859-2 corespunzătoare

ASCII şi 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 şi interpretat ca ISO-8859-m va fi evident interpretat greşit.

Ordinea numerică a codurilor din oricare dintre codificările ISO-8859 este diferită de ordinea alfabetică. În general, în ordinea alfabetică, literele cu diacritice îşi 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 şi 123.

7.2.3. Codificările Unicode

Unicode este un set de caractere ce se doreşte 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 aceloraşi caractere ca şi în codificarea ASCII.

Reprezentarea codurilor unicode ca şiruri de octeţi poate fi făcută în mai multe moduri. Cele mai răspândite codificări sunt:

- *UTF-8* este o codificare de lungime variabilă, între 1 şi 4 octeţi pentru un caracter;
- *UTF-16*, *UTF-16LE*, *UTF-16BE* sunt codificări de lungime variabilă, 2 sau 4 octeţi pentru un caracter;
- *UTF-32*, *UTF-32LE*, *UTF-32BE* sunt codificări de lungime fixă, de 4 octeţi pentru fiecare caracter.

Caracter	Cod *unicode* (hexa)	Cod *unicode* (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
Ţ	162	354	C5 A2
ţ	163	355	C5 A3

Tabelul 7.2: Caracterele cu diacritice din alfabetul limbii române şi codificările *unicode* corespunzătoare. Notă: caracterele *Ş*, *ş*, *Ţ* şi *ţ* 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

Corespondenţa de la codul caracterului la şirul de octeţi 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 şi c_0 cea mai puţin 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} şi $7FF_{16}$) poate fi principial aplicată şi la coduri mai mici (de exemplu, pentru $c = 41_{16}$, rezultând doi octeţi, $C1_{16}$ 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 câţiva octeţi poate regăsi unde începe un caracter nou), deoarece fiecare caracter nou începe cu un octet cuprins între 0 şi 127 sau între 192 şi 255, iar ceilalţi octeţ din codificarea unui caracter sunt cuprinşi între 128 şi 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 biţi, iar în a doua etapă fiecare astfel de număr este scris ca 2 octeţi consecutivi.

Caracterele cu codul *unicode* între 0 şi $D7FF_{16}$ sau între $E000_{16}$ şi $FFFF_{16}$ se scriu ca un singur întreg pe 16 biţi.

Caracterele cu codul *unicode* între 10000_{16} şi $10FFFF_{16}$ se scriu ca doi întregi de câte 16 biţi astfel: Mai întâi, din codul *unicode* se scade 10000_{16}, rezultând o valoare între 0 şi $FFFFF_{16}$ (20 biţi). Primul întreg de 16 biţi se formează punând cifrele 110110 urmate de primii 10 din cei 20 de biţi. Al doilea întreg se formează punând cifrele 110111 urmate de ultimii 10 din cei 20 de biţi. 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 biţi ca un şir de doi octeţi. Există două modalităţi de a reprezenta fiecare astfel de întreg, începând de la octetul mai semnificativ (de rang mai mare) sau începând de la octetul mai puţin semnificativ. Pentru a reflecta aceste variante diferite de alegere, există trei codificări distincte numite generic *UTF-16*:

- *UTF-16LE*: Primul octet este cel mai puţin semnificativ (*little endian*);
- *UTF-16BE*: Primul octet este cel mai semnificativ (*big endian*);
- *UTF-16*: Ordinea octeţilor poate fi fie *big endian*, fie *little endian*, la alegerea emiţătorului. Primul caracter codificat trebuie să fie caracterul cu codul FEFF_{16} (definit iniţial 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 octeţilor). Ordinea octeţilor este dedusă de receptor prin examinarea primilor doi octeţi: dacă aceştia sunt FE_{16} urmat de FF_{16}, înseamnă că ordinea octeţilor este *big endian*; dacă este FF_{16} urmat de FE_{16}, înseamnă că ordinea octeţilor este *little endian*.

7.2.3.3. Codificările UTF-32

Codificarea UTF-32 constă în codificarea fiecărui caracter ca un întreg pe 32 de biţi, reprezentat la rândul lui ca un şir de 4 octeţi. Ca şi în cazul codificărilor *UTF-16*, există trei codificări *UTF-32*:

- *UTF-32LE*: Primul octet este cel mai puţin semnificativ (*little endian*);
- *UTF-32BE*: Primul octet este cel mai semnificativ (*big endian*);
- *UTF-32*: Ordinea octeţilor poate fi fie *big endian*, fie *little endian*, la alegerea emiţătorului. Primul caracter codificat trebuie să fie caracterul cu codul $FEFF_{16}$. Ordinea octeţilor este dedusă de receptor prin examinarea primilor patru octeţi: dacă aceştia sunt 0, 0, FE_{16}, FF_{16}, înseamnă că ordinea octeţilor este *big endian*; dacă este FF_{16}, FE_{16}, 0, 0, înseamnă că ordinea octeţilor este *little endian*.

7.3. Reprezentarea datei şi orei

Determinarea datei şi orei producerii unui eveniment, precum şi memorarea sau transmiterea acestora, sunt operaţii frecvente într-o reţea de calculatoare.

Problema reprezentării datei şi orei este mult mai dificilă decât pare la prima vedere. Din acest motiv, vom începe prin a studia ce se poate înţelege

prin „ora curentă“, iar apoi vom vedea ce scheme de reprezentare ale datei şi orei există şi ce avantaje şi dezavantaje aduce fiecare dintre ele.

7.3.1. Măsurarea timpului

Există două metode utilizate pentru indicarea timpului curent (datei şi orei curente):

- pe baza unor fenomene astronomice, anume alternanţa zi-noapte (în termeni astronomici, *ziua solară mijlocie*), alternanţa anotimpurilor (*anul tropic*) şi, eventual, fazele lunii (*luna sinodică*);
- pe baza unui fenomen fizic repetabil, de exemplu oscilaţia unui pendul sau vibraţia unui cristal de cuarţ.

Prima variantă este de interes practic imediat pentru sincronizarea activităţilor umane. Are însă complicaţii inerente legate de următoarele fapte:

- alternanţa zi-noapte nu este simultană pe tot Pământul ci este decalată pe longitudine;
- anul, luna şi ziua sunt incomensurabile (rapoartele duratelor lor sunt numere iraţionale);
- anul, luna şi ziua nu au durate constante (fenomenele corespunzătoare nu sunt perfect periodice) şi nici măcar previzibile exact (în special rotaţia Pământului are neuniformităţi imprevizibile datorate redistribuirii masei în interiorul Pământului).

A doua variantă măsoară direct timpul ca mărime fizică şi 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 mişcarea Pământului doar după dezvoltarea, începând cu anii 1950, a ceasurilor atomice, mai precise decât mişcările Pământului. Măsurarea timpului se face pe baza *secundei* definite în Sistemul Internaţional de unităţi (SI) ca *9192631770 de perioade ale oscilaţiei corespunzătoare tranziţiei între cele două nivele hiperfine ale stării fundamentale a atomului de cesiu 133*.

Ca urmare a acestor complicaţii, există mai multe standarde de măsurare şi reprezentare a timpului:

Timpul atomic internaţional (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 rotaţia Pământului, exprimată în unităţi 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 neuniformităţii mişcării de rotaţie a Pământului; după media ultimilor câţiva 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 menţine diferenţa dintre UT1 şi 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 corecţie (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 şi 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 corecţie şi nu a fost eliminată nici una. A 24-a secundă de corecţie se va adăuga la sfârşitul anului 2008, astfel încât ziua de 31 decembrie 2008 va avea 86401 secunde. Datorită unei diferenţe iniţiale de 10 s între TAI şi UTC, diferenţa TAI−UTC este în prezent de 33 s.

Timpul legal în fiecare ţară este definit fie pe baza UT1, fie pe baza UTC (diferenţa este neglijabilă pentru uzul practic), ca fiind UTC (sau UT1) plus sau minus un anumit număr de ore şi uneori şi fracţiuni de oră (exemplu, India are ora legală UTC+5h30min).

În ţările în care există oră de vară, la trecerea de la ora de iarnă la cea de vară şi invers, diferenţa dintre ora legală şi UTC creşte, 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) şi UTC+3 h în timpul verii.

Ora suplimentară introdusă la trecerea de la ora de vară la cea de iarnă nu are notaţie distinctă, de tipul secundelor de corecţie 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 şi 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 notaţia 3:30:00 poate corespunde la două momente de timp, ora de vară 3:30:00 (la 30 min înaintea schimbării orei) şi ora de iarnă 3:30:00 (la 30 min după schimbarea orei).

Pentru gruparea zilelor în unităţi mai mari, în special în ani, sunt utilizate mai multe sisteme (calendare):

Calendarul gregorian , introdus în anul 1582 şi în vigoare în România din anul 1924, este calendarul actual. Anii bisecţi (de 366 de zile) sunt anii cu numărul anului divizibil cu 4, cu excepţia celor divizibili cu 100 fără a fi divizibili cu 400. Ani bisecţi sunt 1600, 2000, 2400 etc; ani nebisecţi 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. şi având regula mai simplă cum că sunt bisecţi toţi anii cu număr divizibil cu 4. Este utilizat adesea de istorici pentru a data şi 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 referinţă. Acest sistem este utilizat frecvent în astronomie deoarece permite uşor calculul duratelor dintre două date; din acelaşi motiv reprezintă o schemă potrivită pentru reprezentarea datei în calculator. Există în două variante. Prima, JD (*julian day*), are ca referinţa data de 1 ianuarie 4713 î.e.n. conform calendarului iulian proleptic, la amiaza UT1. Momentul respectiv este JD 0,0, miezul nopţii următoare este JD 0,5, etc. Cealaltă, MJD (*modified julian day*), are ca referinţă 17 noiembrie 1858 ora 0, adică este JD$-$2400000,5 .

7.3.2. Obiectivele în alegerea reprezentării timpului în calculator

De obicei, operaţiile ce trebuiesc efectuate asupra reprezentării timpului sunt:

1. Citirea sau scrierea timpului ca oră legală conform calendarului gregorian în formatul obişnuit, precum şi efectuarea de operaţii aritmetice de genul adunării sau scăderii unei durate formale (exemplu: mâine la aceeaşi oră; aceasta înseamnă în mod obişnuit 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 aplicaţii speciale, citirea sau afişarea 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 aplicaţii şi rezolvarea lui corectă interzice mutarea ceasului înapoi. Punctul 3 este important în aplicaţiile în timp real; de asemenea, funcţionarea 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, fracţiuni 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 operaţii 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 ştie, despre fiecare oră, pe ce fus orar este considerată şi care sunt regulile privind ora de vară. Punctul 3 ridică, pe lângă problemele comune cu cele legate de punctul 2, complicaţii legate de saltul cu o oră înainte la trecerea de la ora de iarnă la ora de vară şi calculele legate de calendar; de asemenea, pentru calcule exacte ale duratelor, sunt necesare informaţii cu privire la secundele de corecţie.

Reprezentarea orei UTC permite determinarea ordinii cronologice şi 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 şi timpul legal.

Reprezentarea TAI ca număr de unităţi de timp scurse de la un anumit moment fixat rezolvă extrem de simplu punctele 2 şi 3 în schimb mută dificultăţile la rezolvarea punctului 1.

7.3.3. Formate utilizate în practică

Deoarece într-o reţea pot fi prezente calculatoare situate pe fusuri orare distincte, aproape orice format util în reţea fie transmite direct ora UTC sau TAI, fie transmite suficientă informaţie pentru ca receptorul să poată calcula uşor ora UTC.

7.3.3.1. Formatul utilizat de poşta electronică

Pentru poşta electronică (§ 11.1), reprezentarea datei se face ca text şi conţine, în ordine:

- opţional ziua din săptămână, ca prescurtare de 3 litere din limba engleză),
- ziua, ca număr între 1 şi 31,
- luna, ca şir 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 funcţie şi de prezenţa unei secunde de corecţie), eventual fracţiunile de secundă şi eventual specificarea fusului orar. Ora, minutul şi secunda sunt separate prin *două puncte*, fracţiunile 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 diferenţă urmat de caracterul *două puncte* şi încă două cifre pentru numărul de minute diferenţă. Exemplu:

```
21:12:58.342+02:00
```

reprezintă acelaşi moment de timp, dar pe alt fus orar, cu

```
14:12:58.342-05:00
```

Data şi 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 obişnuit se obţine grupând secundele în minute, ore, zile, luni şi ani conform regulilor obişnuite. 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 corecţie adăugate pentru menţinerea în sincronism a UTC cu rotaţia Pământului. De aceea, „timpul unix“ are salturi înapoi de câte o secundă la fiecare introducere a unei astfel de secunde de corecţie. Timpul POSIX este reprezentat în mod obişnuit ca întreg cu semn pe 32 de biţi, şi 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 secundele de corecţie. Numărul de secunde este reprezentat pe 63 de biţi (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 aplicaţii ce au nevoie de rezoluţie mai bună de o secundă, standardul prevede încă două câmpuri de câte 32 de biţi (total 128 de biţi), reprezentând respectiv numărul de nanosecunde şi de attosecunde (valori între 0 şi $10^9 - 1$).

7.4. Recodificări

Este necesar uneori să codificăm un şir mai mult sau mai puţin arbitrar de octeţi sub forma unui şir de caractere supus unor restricţii. Astfel de situaţii apar:

- La trimiterea fişierelor ataşate la mesajele de poştă electronică, întregul mesaj, inclusiv partea ce cuprinde fişierele ataşate, trebuie să îndeplinească anumite restricţii, între altele, să nu conţină caractere ASCII de control cu excepţia perechilor *carriage return–line feed* de la finalul fiecărui rând, să nu aibă rânduri prea lungi, etc. Pe de altă parte, fişierele ataşate pot fi fişiere binare cu conţinut arbitrar.
- La stocarea în fişiere text a unor informaţii reprezentate natural ca şir arbitrar de octeţi, de exemplu la stocarea în fişiere text a unor chei de criptare, semnături electronice, etc.
- În limbaje de programare, la scrierea în şirurile de caractere a unor caractere cu rol special, ca de exemplu a ghilimelelor (care în mod normal sunt interpretate ca terminatorul şirului de caractere).

În astfel de situaţii, este necesar să se codifice un şir arbitrar de octeţi (fiecare octet poate lua orice valori între 0 şi 255) pe un alfabet constând în litere, cifre şi câteva simboluri speciale. Deoarece numărul de simboluri disponibile este mai mic decât numărul de valori posibile ale octeţilor, 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 mulţimea cuprinzând cifrele zecimale (0–9) şi literele *A–F*.

Exemplu: şirul de octeţi 120, 0, 23, 45, 20 se scrie: `7800172D14`.

Uneori, în şirul de cifre hexa se inserează spaţii 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 şirului.

7.4.2. Codificarea în baza 64

Codificarea *în baza 64* codifică un şir de 3 octeţi cu valori arbitrare ca un şir de 4 caractere din mulţimea cuprinzând literele mari, literele mici, cifrele şi caracterele `+`, `/` şi `=`.

Codificarea se face în modul următor:

1. Şirul iniţial de octeţi se completează la un multiplu de 3 octeţi prin adăugarea a 0, 1 sau 2 octeţi cu valoarea 0.
2. Se formează un şir de 24 de biţi punând unul după altul cei câte 8 biţi din fiecare octet; din fiecare octet se începe cu bitul cel mai semnificativ.
3. Şirul de 24 de biţi se împarte în 4 grupuri de câte 6 biţi.
4. Fiecare şir de 6 biţi se consideră ca fiind un număr cuprins între 0 şi 63, considerând primul bit din şir ca fiind cel mai semnificativ.
5. Fiecare număr obţinut 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`) şi caracterele `+` şi `/` (62→`+`, 63→`/`). Dacă o valoare 0 provine din biţi 0 proveniţi integral dintr-un octet completat la pasul 1, în loc de `A` se scrie `=`.

De exemplu, şirul de octeţi 120, 0, 23, 45, 20 devine şirul de biţi

01111000 00000000 00010111 00101101 00010100 00000000

ultimul octet fiind adăugat la pasul 1. Şirul de biţi se regrupează

011110 000000 000000 010111 001011 010001 010000 000000

rezultând şirul de numere „în baza 64“: 30, 0, 0, 23, 11, 17, 16, 0, care se codifică `eAAXLRQ=`

7.4.3. Codificări bazate pe secvenţe de evitare

Recodificările prin secvenţe de evitare sunt utilizate în situaţia în care majoritatea octeţilor sau caracterelor din textul de recodificat sunt codurile unor caractere ce se regăsesc în alfabetul în care se face recodificarea. În această situaţie, este favorabil ca, pe cât posibil, octeţii sau caracterele din textul de recodificat să fie codificate prin ele însele, în special pentru ca textul să poată fi înţeles (parţial, cel puţin) de către un utilizator uman direct în forma recodificată.

Recodificarea se face astfel. Se distinge un caracter în alfabetul destinaţie, caracter ce este denumit *caracter de evitare* (enlg. *escape character*).

- Orice caracter din alfabetul sursă care se regăseşte în alfabetul destinaţie şi este diferit de caracterul de evitare se recodifică ca el însuşi.

- Caracterul de evitare (dacă face parte din alfabetul sursă), precum şi caracterele din alfabetul sursă ce nu se găsesc în alfabetul destinaţie, se codifică ca secvenţe de caractere ce încep cu un caracter de evitare.

  Exemple:

- Pentru poşta electronică, un caracter ce nu pot fi transmise direct este codificat ca o secvenţă de trei caractere, un caracter *egal* (`=`) şi două cifre hexa reprezentând codul caracterului de transmis. Această recodificare se numeşte *quouted printables*. Caracterele ASCII imprimabile, cu excepţia caracterului *egal*, se transmit direct (fără recodificare). Ca urmare, un text în care caracterele ce trebuie recodificate sunt rare poate fi înţeles de către un utilizator uman fără prea mari dificultăţi. Ca exemplu, fraza precedentă se scrie (presupunând o codificare ISO-8859-2 pentru caractere şi apoi o recodificare *quouted 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 dificult=E3=FEi.
  ```

- În *URL*-uri, caracterele ce au rol sintactic (*spaţiu*, `/`, 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 *şir* ar avea un URL de forma

  ```
  http://example.com/%C8%98ir
  ```

- În limbajul C şi în alte limbaje derivate din el, ghilimelele (`"`) servesc la delimitarea şirurilor de caractere. Dacă se doreşte introducerea unor astfel de caractere într-un şir, sau a caracterelor ASCII de control, se introduc secvenţe *escape* cum ar fi: `\"` pentru ghilimele (`"`), `\\` pentru *backslash* (`\`), `\n` pentru *newline* (caracterul ASCII cu codul 10).
- În HTML, caracterele *unicode* sunt scrise prin secvenţe `&#`*cod*`;`. De exemplu, litera *ţ* se scrie `&#539;`.

Capitolul 8

Programarea în reţea — introducere

8.1. Interfaţa de programare *socket BSD*

Interfaţa *socket* este un ansamblu de funcţii sistem utilizate de programe (de fapt, de procese) pentru a comunica cu alte procese, aflate în execuţie pe alte calculatoare. Interfaţa *socket* a fost dezvoltată în cadrul sistemului de operare BSD (sistem de tip UNIX, dezvoltat la Universitatea Berkley) — de aici denumirea de *socket BSD*. Interfaţa socket este disponibilă în aproape toate sistemele de operare actuale.

Termenul *socket* se utilizează atât pentru a numi ansamblul funcţiilor sistem legate de comunicaţia prin reţea, cât şi pentru a desemna fiecare capăt al unei conexiuni deschise în cadrul reţelei.

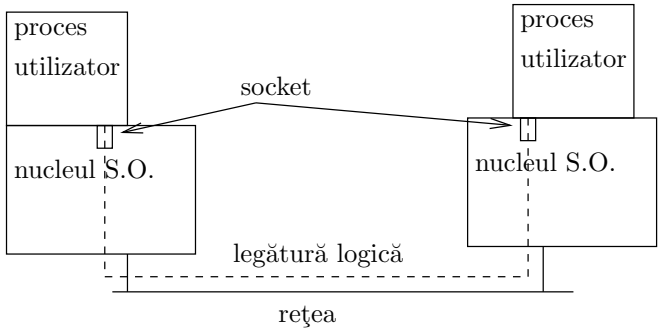


Figura 8.1: Comunicaţia între două procese prin reţea

Prezentăm în continuare principiile de bază ale interfeţei *socket* (vezi

şi figura 8.1):

- Pe fiecare calculator rulează mai multe procese şi fiecare proces poate avea mai multe căi de comunicaţie 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 recepţionarea de date pe o conexiune şi configurarea parametrilor unei conexiuni se fac de către sistemul de operare, la cererea procesului. Cererile procesului se fac prin apelarea funcţiilor sistem din familia *socket*.
- În cadrul comunicaţiei dintre procesul utilizator şi sistemul de operare local (prin intermediul apelurilor din familia *socket*), capetele locale ale conexiunilor deschise sunt numite *socket*-uri şi sunt identificate prin numere întregi, unice în cadrul unui proces la fiecare moment de timp.
- Fiecare entitate care comunică în cadrul reţelei este identificat printr-o *adresă* unică. O adresa este asociată de fapt unui socket. Adresa este formată conform regulilor protocolului de reţea utilizat.
- Interfaţa socket conţine funcţii pentru comunicaţiei atât conform modelului conexiune cât şi conform modelului cu datagrame.
- Funcţiile sistem oferite permit stabilirea comunicaţiei prin diferite protocoale (de exemplu, IPv4, IPv6, IPX), dar au aceeaşi sintaxă de apel independent de protocolul dorit.

8.1.1. Comunicaţia prin conexiuni

În cele ce urmează, prin *client* desemnăm procesul care solicită în mod activ deschiderea conexiunii către un partener de comunicaţie specificat printr-o *adresă*, iar prin *server* înţelegem procesul care aşteaptă în mod pasiv conectarea unui *client*.

Vom da în cele ce urmează o scurtă descriere a operaţiilor pe care trebuie să le efectueze un proces pentru a deschide o conexiune şi a comunica prin ea. Descrierea este împărţită în patru părţi: deschiderea conexiunii de către client, deschiderea conexiunii de către server, comunicaţia propriu-zisă şi închiderea conexiunii.

O descriere mai amănunţită a funcţiilor sistem apelate şi a parametrilor mai des utilizaţi este făcută separat (§ 8.1.3), iar pentru detalii suplimentare se recomandă citirea paginilor corespunzătoare din documentaţia 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 reţea utilizat (TCP/IPv4, TCP/IPv6, etc), dar încă nu se specifică partenerul de comunicaţie. 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) şi numărul portului (vezi § 10.3.1).

Funcţiile sistem apelate sunt: `socket()` pentru crearea socket-ului şi `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 aşteptarea conectării clienţilor; ca urmare este numit uneori *socket de aşteptare*. După crearea acestui socket, serverul trebuie să ceară sistemului de operare stabilirea adresei la care serverul aşteaptă cereri de conectare (desigur, acea parte din adresă care identifică maşina serverului nu este la alegerea procesului server) şi apoi cere efectiv începerea aşteptării clienţilor. Funcţiile apelate în această fază sunt, în ordinea în care trebuie apelate: `socket()` pentru crearea socket-ului, `bind()` pentru stabilirea adresei şi `listen()` pentru începerea aşteptării clienţilor.

Preluarea efectivă a unui client conectat se face prin apelarea unei funcţii sistem numită `accept()`. La apelul funcţiei `accept()`, sistemul de operare execută următoarele:

- aşteaptă cererea de conectare a unui client şi 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 aşteptare poate fi utilizat pentru a aştepta noi clienţi, iar socket-ul de conexiune nou creat se utilizează pentru a comunica efectiv cu acel client.

8.1.1.3. Comunicaţia propriu-zisă

O dată deschisă conexiunea, clientul poate trimite şiruri de octeţi către server şi invers, serverul poate trimite şiruri de octeţi către client. Cele două sensuri de comunicaţie funcţionează identic (nu se mai distinge cine a fost client şi cine a fost server) şi complet independent (trimiterea datelor pe un sens nu este condiţionată de recepţionarea datelor pe celălalt sens).

Pe fiecare sens al conexiunii, se poate transmite un şir arbitrar de octeţi. Octeţii trimişi de către unul dintre procese spre celălalt sunt plasaţi într-o coadă, transferaţi prin reţea la celălalt capăt şi citiţi de către procesul de acolo. Comportamentul acesta este similar cu cel al unui *pipe* UNIX.

Trimiterea datelor se face prin apelul funcţiei `send()` (sau, cu funcţionalitate mai redusă, `write()`). Apelul acestor funcţii plasează datele în coadă spre a fi transmise, dar nu aşteaptă 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 maşini), apelul `send()` se blochează, returnând controlul procesului apelant abia după ce partenerul de comunicaţie citeşte date din coadă, ducând la scăderea dimensiunii datelor din coadă sub valoarea prag.

Recepţionarea datelor trimise de către partenerul de comunicaţie se face prin intermediul apelului sistem `recv()` (cu functionalitate mai redusă se poate utiliza `read()`). Aceste funcţii returnează procesului apelant datele deja sosite pe calculatorul receptor şi le elimină din coadă. În cazul în care nu sunt încă date disponibile, ele aşteaptă sosirea a cel puţin un octet.

Sistemul garantează sosirea la destinaţie a tuturor octeţilor trimişi (sau înştiinţarea receptorului, printr-un cod de eroare, asupra căderii conexiunii), în ordinea în care au fost trimişi. Nu se păstrează însă demarcarea între secvenţele de octeţi trimise în apeluri `send()` distincte. De exemplu, este posibil ca emiţătorul să trimită, în două apeluri succesive, şirurile `abc` şi `def`, iar receptorul să primească, în apeluri `recv()` succesive, şirurile `ab`, `cde` şi `f`.

8.1.1.4. Închiderea conexiunii

Închiderea conexiunii se face separat pentru fiecare sens şi pentru fiecare capăt. Există două funcţii:

- `shutdown()` închide, la capătul local al conexiunii, sensul de comunicaţie cerut de procesul apelant;
- `close()` închide la capătul local ambele sensuri de comunicaţie şi în plus distruge socket-ul, eliberând resursele alocate (identificatorul de socket

şi memoria alocată în spaţiul nucleului).

Terminarea unui proces are efect identic cu un apel `close()` pentru toate socket-urile existente în acel moment în posesia acelui proces.

Dacă capătul de emisie al unui sens de comunicaţie 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 recepţie 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 şi la primirea, de către procesul emiţător, a unui semnal SIGPIPE).

8.1.2. Comunicaţia prin datagrame

În comunicaţia prin datagrame, datagramele sunt transmise independent una de cealaltă şi fiecare datagramă are o adresă sursă, o adresă destinaţie şi nişte date utile. Un proces ce doreşte să trimită sau să primească datagrame trebuie mai întâi să creeze un *socket* de tip *dgram*; un astfel de socket conţine în principal adresa de reţea 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 funcţiei `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 funcţiei `sendto()`. Datagramele trimise vor avea ca adresă sursă adresa socket-ului şi ca adresă destinaţia şi conţinut util valorile date ca parametri funcţiei `sendto()`. De pe un socket se pot trimite, succesiv, oricâte datagrame şi 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 funcţia `recvfrom()`. Această funcţie ia următoarea datagramă adresată socket-ului dat ca parametru la `recvfrom()` şi o transferă din buffer-ele sistemului local în memoria procesului apelant. Funcţia oferă apelantului conţinutul datagramei (datele utile) şi, 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) şi nici nu oferă vreun mecanism de informare a expeditorului în cazul unei pierderi. Mai mult, există posibilitatea (e drept, rară) ca o datagamă să fie duplicată (să ajungă două copii la destinatar) şi

este posibil ca două sau mai multe datagrame adresate aceluiaşi destinatar să ajungă la destinaţie în altă ordine decât cea în care au fost emise. Dacă astfel de situaţii sunt inadmisibile pentru aplicaţie, atunci protocolul de comunicaţie trebuie să prevadă confirmări de primire şi repetarea datagramelor pierdute, precum şi numere de secvenţă sau alte informaţii pentru identificarea ordinii corecte a datagramelor şi a duplicatelor. Implementarea acestor mecanisme cade în sarcina proceselor.

La terminarea utilizării unui socket, procesul posesor poate cere distrugerea socket-ului şi 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 funcţiei `close()`.

În mod curent, într-o comunicaţie prin datagrame, unul dintre procese are rol de *client*, în sensul că trimite cereri, iar celălalt acţionează ca *server*, în sensul că prelucrează cererile clientului şi trimite înapoi clientului răspunsurile la cereri. Într-un astfel de scenariu, serverul crează un socket căruia îi asociază o adresă prestabilită, după care aşteaptă 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 acestă adresă. La primirea unei datagrame, serverul recuperează datele utile şi adresa sursă, procesează cererea şi trimite răspunsul către adresa sursă a cererii. În acest fel, răspunsul este adresat exact socket-ului de pe care clientul a trimis cererea. Clientul obţine 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 conţine interogarea şi daatgrama răspuns conţine atât cererea la care se răspunde cât şi răspunsul la interogare. Serverul ia (în mod repetat) câte o cerere, calculează răspunsul şi trimite o înapoi o datagramă cu cererea primită şi răspunsul la cerere. Clientul trimite cererile sale şi aşteaptă răspunsurile. Deoarece fiecare răspuns conţine în el şi cererea, clientul poate identifica fiecare răspuns la ce cerere îi corespunde, chiar şi în cazul inversării ordinii datagramelor. Dacă la o cerere nu primeşte răspuns într-un anumit interval de timp, clientul repetă cererea; deoarece procesarea unei cereri nu modifică starea serverului, duplicarea cererii de către reţea sau repetarea cererii de către client ca urmare a pierderii

răspunsului nu au efecte nocive. Clientul trebuie să ignore răspunsurile duplicate la o aceeaşi interogare.

8.1.3. Principalele apeluri sistem

8.1.3.1. Funcţia `socket()`

Funcţia are sintaxa:

```
int socket(int proto_family, int type, int protocol)
```

Funcţia crează un socket şi returnează identificatorul său. Parametrii sunt:

- `type`: desemnează tipul de servicii dorite:
  - `SOCK_STREAM`:conexiune punct la punct, flux de date bidirecţional la nivel de octet, asigurând livrare sigura, cu pastrarea ordinii octeţilor şi transmisie fara erori.
  - `SOCK_DGRAM`:datagrame, atât punct la punct cât şi difuziune; transmisia este garantată a fi fără erori, dar livrarea nu este sigura şi nici ordinea datagramelor garantata.
  - `SOCK_RAW`:acces la protocoale de fivel coborât; este de exemplu utilizat de către comanda `ping` pentru comunicaţie prin protocolul ICMP.
- `proto_family` identifică tipul de reţea 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`:comunicaţie locală pe o maşină UNIX.
- `protocol` selectează protocolul particular de utilizat. Acest parametru este util dacă pentru un tip de reţea dat şi pentru un tip de serviciu fixat există mai multe protocoale utilizabile. Valoarea 0 desemnează protocolul implicit pentru tipul de reţea şi tipul de serviciu alese.

8.1.3.2. Funcţia `connect()`

Funcţia are sintaxa:

```
int connect(int sock_id, struct sockaddr* addr, int addr_len)
```

Funcţia 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` şi `addr_len`. La adresa respectivă trebuie să existe deja un server care să aştepte 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; conţinutul 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 complicaţii este legat de faptul că funcţia `connect()` trebuie să poată lucra cu formate diferite de adresă, pentru diferite protocoale, iar unele protocoale au adrese de lungime variabilă.

Funcţia `connect()` returnează 0 în caz de succes şi -1 în caz de eroare. Eroarea survenită poate fi constatată fie verificând valoarea variabilei globale `errno`, fie apelând funcţia `perror()` imediat după funcţia sistem ce a întâmpinat probleme. Eroarea cea mai frecventă este lipsa unui server care să asculte la adresa specificată.

8.1.3.3. Funcţia `bind()`

```
int bind(int sd, struct sockaddr* addr, socklen_t len)
```

Funcţia are ca efect atribuirea adresei specificate în parametrul `addr` socket-ului identificat prin identificatorul `sd`. Această funcţie se apelează în mod normal dintr-un proces server, pentru a pregăti un socket *stream* de aşteptare sau un socket *dgram* pe care se aşteaptă cereri de la clienţi.

Partea, din adresa de atribuit socket-ului, ce conţine adresa maşinii poate fi fie una dintre adresele maşinii 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, şi nu şi cele adresate altora dintre adresele maşinii server.

Exemplul 8.1: Să presupunem că maşina server are adresele 193.226.40.130 şi 127.0.0.1. Dacă la apelul funcţiei `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 şi 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 maşinii locale, adică atât adresei 193.226.40.130 cât şi adresei 127.0.0.1.

Adresa atribuită prin funcţia `bind()` trebuie să fie liberă în acel moment. Dacă în momentul apelului `bind()` există un alt socket de acelaşi tip având aceeaşi adresă, apelul `bind()` eşueaza.

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.

Funcţia `bind()` poate fi apelată doar pentru un socket proaspăt creat, căruia nu i s-a atribuit încă o adresă. Aceasta înseamnă că funcţia `bind()` nu poate fi apelată de două ori pentru acelaşi socket. De asemenea, funcţia `bind()` nu poate fi apelată pentru un socket de conexiune creat prin funcţia `accept()` şi nici pentru un socket asupra căruia s-a apelat în prealabil vreuna dintre funcţiile `connect()`, `listen()` sau `sendto()` — aceste funcţii având ca efect atribuirea unei adrese libere aleatoare.

Funcţia returnează 0 în caz de succes şi -1 în caz de eroare. Eroarea cea mai frecventă este că adresa dorită este deja ocupată.

8.1.3.4. Funcţia `listen()`

```
int listen(int sd, int backlog)
```

Funcţia 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), funcţia `listen()` îi atribuie o adresă aleasă aleator.

Parametrul `backlog` fixează dimensiunea cozii de aşteptare în acceptarea conexiunilor. Anume, vor putea exista `backlog` clienţi 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 clienţi conectaţi, preluaţi deja prin apelul `accept()`.

8.1.3.5. Funcţia `accept()`

```
int accept(int sd, struct sockaddr *addr, socklen_t *addrlen)
```

Apelul funcţiei `accept()` are ca efect crearea unui socket de conexiune, asociat unui client conectat (prin apelul `connect()`) la socket-ul de aşteptare `sd`. Dacă nu există încă nici un client conectat şi pentru care să nu se fi creat socket de conexiune, funcţia `accept()` aşteaptă până la conectarea următorului client.

Funcţia returnează identificatorul socket-ului de conexiune creat.

Dacă procesul server nu doreşte să afle adresa clientului, va da valori `NULL` parametrilor `addr` şi `addrlen`. Dacă procesul server doreşte să afle adresa clientului, atunci va trebui să aloce spaţiu 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 şi să pună adresa acestui întreg în parametrul `adrlen`. În acest caz, la revenirea din apelul `accept()`, procesul server va găsi în structura de adresă adresa socket-ului client şi în variabila întreagă a cărui adresă a fost dată în parametrul `adrlen` va găsi dimensiunea efectiv utilizată de sistemul de operare pentru a scrie adresa clientului.

8.1.3.6. Formatul adreselor

Pentru funcţiile socket ce primesc de la apelant (ca parametru) o adresă din reţea (`bind()`, `connect()` şi `sendto()`), precum şi pentru cele ce returnează apelantului adrese de reţea (`accept()`, `recvfrom()`, `getsockname()` şi `getpeername()`), sunt definite structuri de date (`struct`) în care se plasează adresele socket-urilor.

Pentru ca funcţiile de mai sus să poată avea aceeaşi sintaxă de apel independent de tipul de reţea (şi, în consecinţă, de structura adresei), funcţiile primesc adresa printr-un pointer la zona de memorie ce conţine adresa de reţea. Structura zonei de memorie respective depinde de tipul reţelei utilizate. În toate cazurile, aceasta începe cu un întreg pe 16 biţi reprezentând tipul de reţea.

Dimensiunea structurii de date ce conţine adresa de reţea depinde de tipul de reţea şi, în plus, pentru anumite tipuri de reţea, dimensiunea este variabilă. Din acest motiv:

- funcţiile care primesc de la apelant o adresă (`connect()`, `bind()` şi `sendto()`) au doi parametri: un pointer către structura de adresă şi un întreg reprezentând dimensiunea acestei structuri;
- funcţiile care furnizează apelantului o adresă (`accept()`, `recvfrom()`, `getsockname()` şi `getpeername()`) primesc doi parametri: un pointer către structura de adresă şi un pointer către o variabilă de tip întreg pe care apelantul trebuie s-o iniţializeze, înaintea apelului, cu dimensiunea pe care a alocat-o pentru structura de adresă şi în care funcţia pune, în timpul apelului, dimensiunea utilizată efectiv de astuctura de adresă.

În ambele cazuri, parametrul pointer către structura de adresă este declarat ca fiind de tip `struct sockaddr*`. La apelul acestor funcţii este necesară conversia a pointer-ului către structura de adresă de la pointer-ul specific tipului de reţea 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 maşinii şi numărul de port (vezi § 10.2.3.1, § 10.3.1.6 şi § 10.3.2). Pentru nevoile funcţiilor de mai sus, adresele socket-urilor TCP şi UDP se pun, în funcţie de protocolul de nivel

reţea (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 conţină constanta `AF_INET`;

`sin_port`:de tip întreg de 16 biţi (2 octeţi), fără semn, în ordine reţea (cel mai semnificativ octet este primul), reprezentând numărul portului;

`sin_addr`:conţine adresa IP. Are tipul `struct in_addr`, având un singur câmp, `s_addr`, de tip întreg pe 4 octeţi în ordine reţea.

Adresa IPv4 poate fi convertită de la notaţia obişnuită (notaţia zecimală cu puncte) la `struct in_addr` cu ajutorul funcţiei

```
int inet_aton(const char *cp, struct in_addr *inp);
```

Conversia inversă, de la structura `in_addr` la string în notaţie zecimală cu punct se face cu ajutorul funcţiei

```
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 functiei.

Pentru adrese IPv6 este definită structura `sockaddr_in6` având următorii membrii:

`sin6_family`:trebuie să conţină constanta `AF_INET6`;

`sin6_port`:de tip întreg de 16 biţi (2 octeţi), fără semn, în ordine reţea (cel mai semnificativ octet este primul), reprezentând numărul portului;

`sin6_flow`:eticheta de flux.

`sin6_addr`:conţine adresa IP. Are tipul `struct in6_addr`, având un singur câmp, `s6_addr`, de tip tablou de 16 octeţi.

Obţinerea unei adrese IPv4 sau IPv6 cunoscând numele de domeniu (vezi § 10.4) se face cu ajutorul funcţiei

```
struct hostent *gethostbyname(const char *name);
```

care returnează un pointer la o structură ce conţine 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 şir de pointeri către adresele IPv4 sau IPv6 ale maşinii cu numele `name`, în formatul `in_addr` sau respectiv `in6_addr`;

`int h_length`:lungimea şirului `h_addr_list`.

8.1.3.7. Interacţiunea dintre `connect()`, `listen()` şi `accept()`

La apelul `connect()`, sistemul de operare de pe maşina client trimite maşinii server o cerere de conectare. La primirea cererii de conectare, sistemul de operare de pe maşina server acţionează 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 clienţi conectaţi şi nepreluaţi încă prin `accept()`. Dacă plasarea reuşeşte (coada fiind mai mică decât valoarea parametrului `backlog` din apelul `listen()`), sistemul de operare trimite sistemului de operare de pe maşina 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 maşina server. Revenirea din apelul `connect()` nu este deci condiţionată de apelul `accept()` al procesului server.

Apelul `accept()` preia un client din coada descrisă mai sus. Dacă coada este vidă în momentul apelului, funcţia aşteaptă sosirea unui client. Dacă coada nu este vidă, apelul `accept()` returnează imediat.

Parametrul `backlog` al apelului `listen()` se referă la dimensiunea cozii de clienţi conectaţi (prin `connect()`) şi încă nepreluaţi prin `accept()`, şi nu la clienţii deja preluaţi prin `accept()`.

8.1.3.8. Funcţiile `getsockname()` şi `getpeername()`

```
int getsockname(int sd, struct sockaddr *name, socklen_t *namelen);
int getpeername(int sd, struct sockaddr *name, socklen_t *namelen);
```

Funcţia `getsockname()` furnizează apelantului adresa socket-ului `sd`. Funcţia `getpeername()`, apelată pentru un socket de tip conexiune deja conectat, furnizează adresa partenerului de comunicaţie.

Funcţia `getsockname()` este utilă dacă un proces acţionează ca server, creînd în acest scop un socket de aşteptare, dar numărul portul pe care aşteaptă 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 şi transmite această adresă viitorilor clienţi.

8.1.3.9. Funcţiile `send()` şi `recv()`

Apelurile sistem `send()` şi `recv()` sunt utilizate în faza de comunicaţie pentru socket-uri de tip conexiune. Descriem în continuare utilizarea acestor funcţii considerând un singur sens de comunicaţie şi ca urmare ne vom referi la un *proces emiţător* şi un *proces receptor* în raport cu sensul considerat. De notat însă că o conexiune *socket stream* este bidirecţională şi comunicarea în cele două sensuri se desfăşoară independent şi prin aceleaşi mecanisme.

Sintaxa funcţiilor 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);
```

Funcţia `send()` trimite pe conexiunea identificată prin socket-ul `sd` un număr de `len` octeţi din variabila a cărui adresă este indicată de pointer-ul `buf`. Funcţia returnează controlul după plasarea datelor de transmis în buffer-ele sistemului de operare al maşinii locale. Valoarea returnată de funcţia `send()` este numărul de octeţi scrişi efectiv, sau -1 în caz de eroare. Datele plasate în buffer-e prin apelul `send()` urmează a fi trimise spre receptor fără alte acţiuni din partea emiţătorului.

În modul normal de lucru, dacă nu există spaţiu suficient în buffer-ele sistemului de operare, funcţia `send()` aşteaptă ca aceste buffer-e să se elibereze (prin transmiterea efectivă a datelor către sistemul de operare al receptorului şi citirea lor de către procesul receptor). Această aşteptare are ca rol frânarea procesului emiţător dacă acesta produce date la un debit mai mare decât cel cu care este capabilă reţeaua 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 spaţiu în buffer-ele sistemului de operare, funcţia `send()` scrie doar o parte din datele furnizate şi returnează imediat controlul procesului apelant. În cazul în care funcţia `send()` nu scrie nimic, ea returnează valoarea -1 şi setează variabila globală `errno` la valoarea `EAGAIN`. În cazul în care funcţia `send()` scrie cel puţin un octet, ea returnează numărul de octeţi scrişi efectiv. În ambele cazuri, este sarcina procesului emiţător să apeleze din nou, la un moment ulterior, funcţia `send()` în vederea scrierii octeţilor rămaşi.

Deoarece funcţia `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 reţelei şi închiderea conexiunii de către receptor. Aceste erori vor fi raportate de către sistemul de operare al emiţătorului procesului emiţător prin aceea că apeluri `send()` ulterioare pentru acelaşi socket vor returna -1. În plus, pe sistemele de tip UNIX, apelul `send()` pentru o conexiune al cărui capăt destinaţie este închis duce la primirea de către procesul emiţător a unui semnal SIGPIPE, care are ca efect implicit terminarea imediată a procesului emiţător.

Funcţia `recv()` extrage date sosite pe conexiune şi aflate în buffer-ul sistemului de operare local. Funcţia primeşte ca argumente identificatorul socket-ului corespunzător conexiunii, adresa unei zone de memorie unde să plaseze datele citite şi numărul de octeţi de citit.

Numărul de octeţi de citit reprezintă numărul maxim de octeţi pe care funcţia îi va transfera din buffer-ul sistemului de operare în zona procesului apelant. Dacă numărul de octeţi disponibili în buffer-ele sistemului de operare este mai mic, doar octeţii disponibili în acel moment vor fi transferaţi. Dacă în momentul apelului nu există nici un octet disponibil în buffer-ele sistemului de operare local, funcţia `recv()` aşteaptă sosirea a cel puţin un octet. Funcţia returnează numărul de octeţi transferaţi (citiţi 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, funcţia `recv()` returnează valoarea -1 şi setează variabila globală `errno` la valoarea `EAGAIN`;

`MSG_WAITALL`:funcţia `recv()` aşteaptă să fie disponibili cel puţin `len` octeţi şi citeşte exact `len` octeţi.

Este important de notat că datele sunt transmise de la sistemul de operare emiţător spre cel receptor în fragmente (pachete), că împărţirea datelor în fragmente este independentă de modul în care au fost furnizate prin apeluri `send()` succesive şi că, în final, fragmentele ce vor fi disponibile succesiv pentru receptor sunt independente de fragmentele furnizate în apelurile `send()`. Ca urmare, este posibil ca emiţătorul să trimită, prin două apeluri `send()` consecutive, şirurile de octeţi `abc` şi `def`, iar receptorul, apelând repetat `recv()` cu `len`=3 şi `flags=0`, să primească `ab`, `cd` şi `ef`. Singurul lucru garantat este că prin concatenarea tuturor fragmentelor trimise de emiţător se obţine acelaşi şir de octeţi ca şi prin concatenarea tuturor fragmentelor primite de receptor.

În cazul închiderii conexiunii de către emiţător, 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 indetificatorul 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 conţine adresa de reţea a destinatarului; `tolen` reprezintă lungimea structurii pointate de `to`;
- `buf`: pointer spre o zonă de memorie ce conţine datele utile; `len` reprezintă lungimea datelor utile. Datele utile sunt un şir arbitrar de octeţi.

Funcţia returnează numărul de octeţi ai datagramei trimise (adică valoarea lui `len`) în caz de succes şi -1 în caz de eroare. Funcţia returnează controlul apelantului înainte ca pachetul să fie livrat destinatarului şi ca urmare eventuala pierdere a pachetului nu poate fi raportată apelantului.

Funcţia `recvfrom()` citeşte din bufferele sistemului de operare local următoarea datagramă adresată socket-ului dat ca parametru. Dacă nu există nici o datagramă, funcţia aşteaptă sosirea următoarei datagrame, cu excepţia cazului în care `flags` conţine valoarea `MSG_DONTWAIT`, caz în care `recvfrom()` returnează imediat valoarea -1 şi setează `errno` la valoarea `EAGAIN`.

Datagrama este citită în zona de memorie pointată de parametrul `buf` şi a cărei dimensiune este dată în variabila `len`. Funcţia `recvfrom()` returnează dimensiunea datagramei. Dacă datagrama este mai mare decât valoara parametrului `len`, finalul datagramei este pierdut; funcţia `recvfrom()` nu scrie niciodată dincolo de `len` octeţi în memoria procesului.

Adresa emiţătorului datagramei este plasată de funcţia `recvfrom()` în variabila pointată de `from`. Parametrul `fromlen` trebuie să pointeze la o variabilă de tip întreg a cărui valoare, înainte de apelul `recvfrom()`, trebuie să fie egală cu dimensiunea, în octeţi, a zonei de memorie alocate pentru adresa emiţătorului. Funcţia `recvfrom()` modifică această variabilă, punând în ea dimensiunea utilizată efectiv pentru scrierea adresei emiţătorului.

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 maşinii şi numărul portului TCP (date ca argumente în linia de comandă), îi trimite un şir de caractere fixat (`abcd`), după care citeşte şi afişează 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

şi afişează pe ecran tot ce transmite clientul (până la închiderea sensului de conexiune de la client la server) şi 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 maşinii sau adresa IP şi numărul de port. Clientul trimite serverului o datagramă de 4 octeţi conţinând textul `abcd` şi aşteaptă o datagramă ca răspuns, a cărei conţinut îl afişează.

```
#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 aşteaptă o datagramă de la un client, afişează adresa de la care a fost trimisă datagrama precum şi conţinutul datagramei primite. Apoi trimite înapoi, la adresa de la care a sosit datagrama de la client, o datagramă conţinând şirul de 3 octeţi 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 şi recepţia 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 informaţii prin instrucţiuni de forma

```
Tip msg;
...
send(sd, &msg, sizeof(msg), 0);
```

şi recepţia prin

```
Tip msg;
...
recv(sd, &msg, sizeof(msg), MSG_WAITALL);
```

unde `Tip` este un tip de date oarecare declarat identic în ambele programe (emiţător şi 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 şi 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 biţi);
- ordinea octeţilor în memorie (*big endian* sau *little endian*) depinde de arhitectura calculatorului.

Dacă scriem un program pentru un anumit tip de calculatoare şi pentru un anumit compilator, pentru care ştim exact dimensiunea unui `int` şi ordinea octeţilor, putem transmite şi recepţiona date prin secvenţe de tipul:

```
int a;
...
send(sd, &a, sizeof(a), 0);
```

pentru emiţător şi

```
int a;
...
recv(sd, &a, sizeof(a), MSG_WAITALL);
```

pentru receptor. Dacă însă emiţătorul este compilat pe o platformă pe care `int` are 16 biţi şi este reprezentat *big endian*, iar receptorul este compilat pe o platformă pe care `int` are 32 de biţi şi este *little endian*, cele două programe nu vor comunica corect.

Pentru a putea scrie programe portabile, biblioteca C standard pe sisteme de tip UNIX conţine, în header-ul `arpa/inet.h`:

- `typedef`-uri pentru tipuri întregi de lungime standardizată (independentă de compilator): `uint16_t` de 16 biţi şi `uint32_t` de 32 de biţi;
- funcţii de conversie între formatul locat (*little endian* sau *big endian*, după caz) şi formatul *big endian*, utilizat cel mai adesea pentru datele transmise în Internet. Aceste funcţii sunt: `htons()` şi `htonl()` (de la *host to network, short*, respectiv *host to network, long*), pentru conversia de la format local la format *big endian* (numit şi *format reţea*), şi `ntohs()` şi `ntohl()` pentru conversia în sens invers. Variantele cu `s` (`htons()` şi `ntohs()`) convertesc întregi de 16 biţi (de tip `uint16_t`, iar cele cu `l` convertesc întregi de 32 de biţi (`uint32_t`).

Implementarea acestor `typedef`-uri şi functii depinde de platformă (de arhitectură şi 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 recepţionează, un întreg reprezentat pe 32 de biţi în format *big endian*.

8.2.1.2. Şiruri de caractere şi 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 şi transmiterea unui element cu valoare speciale (terminator) după ultimul element.

Pe lângă numărul de elemente efective ale tabloului este necesară cunoaşterea numărului de elemente alocate. La reprezentarea în memorie sau în fişiere 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 şi pot fi accesate direct; dezavantajul este un consum sporit de memorie şi o limită mai mică a numărului de obiecte ce pot fi puse în tablou.

La transmiterea tablourilor prin conexiuni în reţea, 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 citeşte întâi numărul de elemente, după care alocă spaţiu (sau verifică dacă spaţiul alocat este suficient) şi citeşte elementele. În cazul reprezentării cu terminator, receptorul citeşte pe rând fiecare elemen şi-i verifică valoarea; la întâlnirea terminatorului se opreşte. Înainte de-a citi fiecare element, receptorul trebuie să verifice dacă mai are spaţiu alocat pentru acesta, iar în caz contrar fie să realoce spaţiu pentru tablou şi să copieze în spaţiul nou alocat elementele citite, fie să renunţe şi să semnaleze eroare.

Exemplul 8.2: Se cere transmiterea unui şir de caractere. Reprezentarea şirului pe conexiune este: un întreg pe 16 biţi *big endian* reprezentând lungimea şirului, urmat de şirul propriu-zis (reprezentare diferită deci de reprezentarea uzuală în memorie, unde şirul se termină cu un caracter nul). Descriem în continuare emiţătorul:

```
char* s;
uint16_t l;
...
l=htons(strlen(s));
send(sd, &l, 2, 0);
send(sd, s, strlen(s), 0);
```

şi 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 reţea.

Exemplul 8.3: Se cere transmiterea unui şir de caractere. Reprezentarea pe conexiune va fi ca un şir de caractere urmat de un caracter nul (adică reprezentare identică celei din memorie, dar pe lungime variabilă, egală cu minimul necesar). Emiţătorul 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 şi în memorie şi pe conexiune — se reprezintă câmpurile unul după altul — şi ca urmare transmiterea lor nu ridică probleme deosebite.

Din păcate însă, reprezentarea unei structuri în memorie depinde de platformă (arhitectura calculatorului şi compilator), datorită problemelor

privind alinierea întregilor. Din considerente legate de arhitectura magistralei de date a calculatorului (detalii ce ies din cadrul cursului de faţă), accesarea de către procesor a unei variabile de tip întreg sau real a cărui adresă în memorie nu este multiplu de un anumit număr de octeţi este pentru unele procesoare imposibilă iar pentru celelalte ineficientă. Numărul ce trebuie să dividă adresa se numeşte *aliniere* şi este de obicei minimul dintre dimensiunea variabilei şi lăţimea magistralei. Astfel, dacă magistrala de date este de 4 octeţi, întregii de 2 octaţi trebuie să fie plasaţi la adrese pare, iar întregii de 4 sau 8 octeţi trebuie să fie la adrese multiplu de 4; nu există restricţii cu privire la caractere (întregi pe 1 octet). Compilatorul, împreună cu funcţiile de alocare dinamică a memoriei, asigură alinierea recurgând la următoarele metode:

- adaugă octeţi nefolosiţi între variabile,
- adaugă octeţi nefolosiţi între câmpurile unei structuri,
- adaugă octeţi nefolosiţi la finalul unei structuri ce face parte dintr-un tablou,
- alocă variabilele de tip structură la adrese multiplu de o lăţimea magistralei.

Ca urmare, reprezentarea în memorie a unei strcturi depinde de platformă şi, în consecinţă, 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 funcţie de lăţimea magistralei, de faptul că alinierea incorectă duce la imposibilitatea accesării variabilei sau doar la ineficienţă şi, în acest din urmă caz, de opţiunile de compilare, dimensiunea structurii `Msg` de mai sus poate fi 5, 6 sau 8 octeţi, între cele două câmpuri fiind respectiv 0, 1 sau 3 octeţi neutilizaţi.

Rezolvarea problemei de portabilitate se face transmiţând 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 esenţă un şir de caractere terminat cu spaţiu, tab, *newline* sau un alt caracter specificat prin standard. Metodele descrise pentru transmiterea şi recepţionarea unui şir de caractere se aplică şi la obiectele transmise în formate de tip text.

8.2.3. Probleme de robusteţe şi securitate

Orice apel de funcţie sistem poate eşua 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 comunicaţie, inclusiv în cazul încălcării de către acesta a standardului de reprezentare a datelor şi inclusiv în cazul în care emiţătorul închide conexiunea în mijlocul transmiterii unei variabile.

Validitatea datelor trebuie verificată întotdeauna după citire. Dacă receptorul aşteaptă un întreg pozitiv, este necesar să se verifice prin program că numărul primit este într-adevăr pozitiv. Este de asemenea necesar să se stabilească şi să se impună explicit nişte limite maxime. Astfel, să presupunem că programul receptor primeşte un şir de întregi reprezentat prin lungimea, ca număr de elemente, pe 32 de biţi, urmată de elementele propriu-zise. Chiar dacă de principiu numărul de elemente ne aşteptăm să fie între 1 şi câteva sute, trebuie să ne asigurăm că programul se comportă rezonabil dacă numărul de elemente anunţat de emiţător 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 şi să încheie curat operaţiile începute.

Orice program care nu este robust este un risc de securitate.

8.2.4. Probleme privind costul apelurilor sistem

Apelul funcţiilor `send()` şi `recv()` este scump, în termeni de timp de procesor, deoarece, fiind funcţii sistem, necesită o comutare de drepturi în procesor, salvarea şi restaurarea contextului apelului şi o serie de verificări din partea nucleului sistemului de operare; în total, echivalentul câtorva sute de instrucţiuni. Acest cost este independent de numărul de octeţi transferaţi.

Este, prin urmare, eficient ca fiecare apel `send()` sau `recv()` să transfere cât de mulţi octeţi se poate. Un program care trimite date este bine să pregătească datele într-o zonă tampon locală şi să trimită totul printr-un singur apel `send()`. Un program care primeşte date este bine să ceară (prin `recv()`) câte un bloc mai mare de date şi apoi să analizeze datele primite. Acest mod de lucru se realizează cel mai bine prin intermediul unor funcţii de bibliotecă adecvate.

Descriem în continuare funcţiile din biblioteca standard C utilizabile în acest scop. Biblioteca poate fi utilizată atât pentru emisie şi recepţie printr-o conexiune *socket stream* cât şi pentru citire sau scriere într-un fişier sau pentru comunicare prin *pipe* sau *fifo*.

Elementul principal al bibliotecii este structura `FILE`, ce conţine:

- un identificator de fişier deschis, conexiune *socket stream*, *pipe* sau *fifo*;
- o zonă de memorie tampon, împreună cu variabilele necesare gestionării ei.

Funcţiile de citire ale bibliotecii sunt `fread()`, `fscanf()`, `fgets()` şi `fgetc()`. Fiecare dintre aceste funcţii extrage datele din zona tampon a structurii `FILE` dată ca parametru. Dacă în zona tampon nu sunt suficienţi octeţi pentru a satisface cererea, aceste funcţii apelează funcţia sistem `read()` asupra identificatorului de fişier din structura `FILE` pentru a obţine octeţii următori. Fiecare astfel de apel `read()` încearcă să citească câţiva kiloocteţi. Pentru fiecare din funcţiile de mai sus, dacă datele de returnat aplicaţiei se găsesc deja în zona tampon, costul execuţiei este de câteva instrucţiuni pentru fiecare octet transferat. Ca urmare, la citirea a câte un caracter o dată, utilizarea funcţiilor de mai sus poate reduce timpul de execuţie 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 ) {}
```

şi

```
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 şir de octeţi terminat cu un caracter nul. Primul fragment de cod apelează `recv()` o dată pentru fiecare caracter al şirului. Al doilea fragment apelează `fgetc()` o dată pentru fiecare caracter al şirului. La o mică parte dintre aceste apeluri, functia `fgetc()` va apela în spate funcţia 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 şi terminării datelor se poate face apelând funcţia `foef()`. De notat că această funcţie poate să returneze `false` chiar dacă s-a ajuns la finalul datelor; rezultatul `true` este garantat doar după o tentativă nereuşită de-a citi dincolo de finalul datelor transmise.

Pentru scriere, funcţiile de bibliotecă corespunzătoare sunt `fwrite()`, `fprintf()`, `fputs()` şi `fputc()`. Aceste funcţii scriu datele în zona tampon din structura `FILE`. Transmiterea efectivă pe conexiune (sau scrierea în fişier) se face automat la umplerea zonei tampon. Dacă este necesar să ne asigurăm că datele au fost transmise efectiv (sau scrise în fişier), funcţia `fflush()` efectuează acest lucru. Funcţia `fclose()` de asemenea trimite sau scrie ultimele date rămase în zona tampon.

Asocierea unei zone tampon unei conexiuni deja deschise se face apelând funcţia `fdopen()`. Funcţia `fdopen()` primeşte doi parametri. Primul parametru este identificatorul de socket căruia trebuie să-i asocieze zona tampon (identificatorul returnat de funcţia `socket()` sau `accept()`). Al doilea parametru specifică funcţiei `fdopen()` dacă trebuie să asocieze zona tampon pentru citire sau pentru scriere; este de tip şir de caractere şi poate avea valoarea `"r"` pentru citire sau `"w"` pentru scriere. Pentru un *socket stream*, cele două sensuri functionând complet independent, aceluiaşi socket i se pot asocia două zone tampon (două structuri `FILE`), câte una pentru fiecare sens.

Funcţia `fclose()` scrie informaţiile rămase în zona tampon (dacă zona tampon a fost creată pentru sensul de scriere), eliberează memoria alocată zonei tampon şi închide conexiunea.

8.3. Probleme de concurenţă în comunicaţie

O particularitate a majorităţii programelor ce comunică în reţea este aceea că trebuie să răspundă prompt la mesaje provenind din surse diferite şi într-o ordine necunoscută dinainte.

Să luăm de exemplu un server *ssh* (§ 11.2.1). Serverul poate avea mai mulţi clienţi conectaţi simultan. La fiecare moment, este imposibil de prezis care dintre clienţi va trimite primul o comandă.

Dacă serverul execută un apel `recv()` blocant de pe socket-ul unui client, serverul va fi pus în aşteptare 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 aşteptarea datelor de la primul client. Ca urmare, datele de la al doilea client vor aştepta cel puţin 10 minute pentru a fi procesate.

Există mai multe soluţii la problema de mai sus:

- Serverul citeşte de la clienţi, 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 aşteptare dacă un client nu i-a trimis nimic. Dezavantajul soluţiei 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 soluţiei anterioare, sistemele de operare de tip UNIX oferă o funcţie sistem, numită `select()`, care primeşte o listă de identificatori de *socket* şi, opţional, o durată, şi pune procesul în aşteptare până când fie există date disponibile pe vreunul din *socket*-ii daţi, 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 execuţie (thread-uri) în cardul procesului server — fiecare proces sau fir de execuţie urmărind un singur client. În acest caz, procesul sau firul de execuţie poate executa `recv()` blocant asupra socket-ului corespunzător clientului său. În lipsa

activităţii clienţilor, fiecare proces sau fir de execuţie al serverului este blocat în apelul `recv()` asupra socket-ului corespunzător. În momentul în care un client trimite date, nucleul sistemului de operare trezeşte procesul sau firul ce executa `recv()` pe socket-ul corespunzător; procesul sau firul execută prelucrările necesare după care probabil execută un nou `recv()` blocant.

Cazul unui server cu mai mulţi clienţi nu este singura situaţie în care este nevoie de a urmări simultan evenimente pe mai multe canale. Alte situaţii sunt:

- un client care trebuie să urmărească simultan acţiunile utilizatorului şi mesajele sosite de la server;
- un server care poate trimite date cu debit mai mare decât capacitatea reţelei sau capacitatea de prelucrare a clientului; în acest caz serverul are de urmărit simultan, pe de o parte noi cereri ale clienţilor, iar pe de altă parte posibilitatea de-a trimite noi date spre clienţi în urma faptului că vechile date au fost prelucrate de aceştia.
- un server care trebuie să preia mesaje de la clienţii conectaţi şi, în acelaşi timp, să poată accepta clienţi noi.

Un aspect important ce trebuie urmărit în proiectarea unui server concurent este servirea echitabilă a clienţilor, 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 celorlalţi clienţi, apoi să revină la primul şi să mai proceseze o parte din cereri şi aşa mai departe. Nu este permis ca un client care inundă serverul cu cereri să acapareze întreaga putere de calcul a serverului şi ceilalţi clienţi să aştepte la infinit.

Capitolul 9

Reţele IEEE 802

Vom studia în continuare standardul utilizat de cele mai multe reţele locale. IEEE 802 defineşte de fapt o familie de tipuri de reţele locale, dintre care cele mai des întâlnite sunt:

- reţele *Ethernet* (de 10, 100 sau 1000 Mbit/s), construite conform standardului IEEE 802.3;
- reţele numite *Wireless Ethernet*, conform standardului IEEE 802.11.

9.1. Reţele IEEE 802.3 (Ethernet)

Cele mai multe reţele locale (reţele cu întinderi geografice reduse, de până la câţiva kilometri) sunt construite pe baza standardului IEEE 802.3 [IEEE 802.3, 2005]. Astfel de reţele mai sunt numite, în mod curent, reţele *Ethernet* (denumirea standardului original, din care a fost dezvoltat standardul IEEE 802.3) sau reţele *UTP 10/100/1000* (UTP vine de la *unshielded twisted pairs — perechi torsadate neecranate* — şi desemnează tipul de cablu utilizat cel mai frecvent în instalaţiile actuale, iar 10/100/1000 sunt capacităţile posibile ale legăturilor, măsurate în megabiţi pe secundă).

Standardul este complex (are peste 1500 de pagini) şi a rezultat în urma unei evoluţii 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 reţea Ethernet sunt:

Interfaţa de reţea sau *placa de reţea* (engl. *Network Interface Card — NIC*) este dispozitivul prin care se conectează un calculator la reţea.

Cablul magistrală. O reţea constrită cu cablu magistrală consta într-un cablu, format din două conductoare izolate între ele, la care sunt

conectate, în paralel, interfeţele de reţea ale calculatoarelor (fig. 9.1). În acest sistem, semnalul emis de orice interfaţă de reţea este recepţionat de toate celelalte interfeţe de reţea conectate la acel cablu.

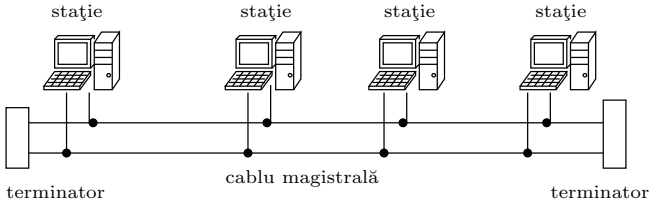


Figura 9.1: O reţea Ethernet construită cu un cablu magistrală

Comunicaţia se face prin pachete de dimensiune variabilă.

Două interfeţe care emit simultan îşi bruiază reciproc semnalele emise; este necesar deci un mecanism de control al accesului la mediu (vezi § 4.2). IEEE 802.3 alege soluţia cu detectarea coliziunilor şi retransmiterea pachetelor distruse de coliziuni.

Deoarece fiecare interfaţă de reţea „aude“ toate pachetele emise în reţea, este prevăzut un mecanism prin care interfaţa să identifice şi să livreze sistemului de operare numai pachetele ce îi sunt destinate. Anume, fiecare interfaţă de reţea are o adresă unică, numită *adresă fizică* sau *adresă MAC* şi fiecare pachet poartă adresa sursei şi adresa destinaţiei.

Repetorul (engl. *repeater*) este un dispozitiv care este conectat la mai multe cabluri de reţea şi copiază pachetele de date de pe fiecare cablu pe celelalte.

Repetorul este conectat la fiecare cablu de reţea întocmai ca o interfaţă de reţea a unui calculator. Interfaţa repetorului către cablul de reţea se numeşte *port*.

Oridecâteori repetorul recepţionează un pachet printr-unul dintre porturile sale (printr-unul din cablurile de reţea conectate la repetor), îl retransmite (repetă) pe toate celelalte cabluri de reţea conectate (toate cu excepţia celui prin care a intrat pachetul). Retransmiterea se face cu întârziere de ordinul duratei câtorva biţi, în orice caz mai puţin decât durata unui pachet. Dacă repetorul recepţionează simultan pachete prin două sau mai multe porturi, consideră că are loc o *coliziune* şi semnalizează acest lucru emiţând, prin toate porturile, un semnal special de anunţare a coliziunii. Acest semnal de coliziune se propagă în toată reţeaua.

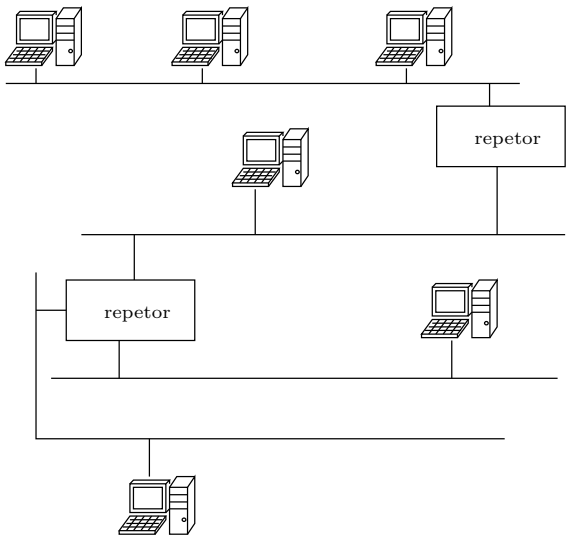


Figura 9.2: O reţea construită din mai multe cabluri magistrală interconectate prin repetoare.

Repetoarele permit construirea unei reţele întinse pe o distanţă mai mare decât lungimea maximă a unui singur cablu (lungime limitată de atenuarea semnalului pe cablu). O reţea construită cu repetoare este desenată în figura 9.2.

Odată cu ieftinirea repetoarelor, a devenit curentă utilizarea câte unui cablu pentru conectarea fiecărui calculator la un repetor. În acest fel, un cablu de reţea va avea legate la el doar două echipamente: fie o interfaţă de reţea şi un repetor, fie două repetoare, fie două interfeţe de reţea (în acest din urmă caz rezultă o reţea formată doar din două calculatoare). De regulă, cablul de legătură folosit în aceste cazuri este o legătură duplex (vezi mai jos) şi poate conecta doar două echipamente. Repetoarele utilizate în această situaţie se mai numesc *hub*-uri (engl. hub = butuc de roată).

Comutatoarele. Un *comutator* (eng. *switch*) este un dispozitiv asemănător cu un repetor, dar cu următoarele modificări:

- este capabil să memoreze câte un pachet întreg pentru fiecare port;
- dacă primeşte simultan două sau mai multe pachete, le memorează şi le retransmite pe rând;
- dacă este posibil, în loc să retransmită un pachet prin toate porturile comutatorului, îl retrimite doar pe calea către interfaţa de reţea

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 şi echipamentele conectate sunt capabile să emită şi să recepţioneze simultan, este posibilă realizarea unei comunicaţii 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 excepţia debitului de comunicaţie şi a existenţei sau absenţei posibilităţii comunicaţiei duplex, tipul cablului de legătură ales nu afectează restul reţelei.

Pentru echipamente capabile să funcţioneze după mai multe standarde privind nivelul fizic (debite diferite şi 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 şi 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.

Condiţiile de izolare electrică şi de pământare nu permit utilizarea sigură pentru legături aeriene prin exteriorul clădirilor. Pentru astfel de scopuri standardul recomandă utilizarea fibrelor optice.

Descriem în continuare particularităţile 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 comunicaţie (10 Mbit/s), codificarea (în banda de bază) şi tipul mediului (*Twisted pairs — perechi torsadate*).

Cablul de legătură constă, aşa 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 interferenţelor 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 reţeaua telefonică şi 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:

- impedanţa 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 puţin $0{,}585 \cdot c$ (adică cel puţin de 0,585 de ori viteza luminii în vid).

Conectarea cablului la interfaţa de reţea 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 şi 2 şi recepţia între pinii 3 şi 6. Pinii 4, 5, 7 şi 8 sunt neutilizaţi.

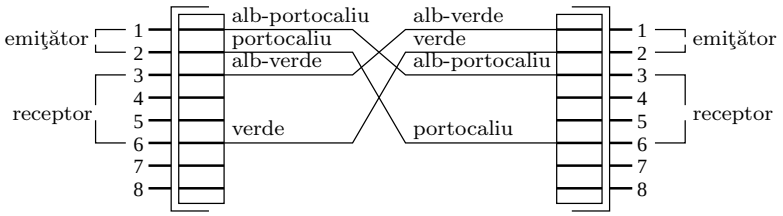
Conductoarele legate la emiţător la un capăt trebuie legate la receptor la celălalt capăt (fig. 9.3). Acest lucru se poate realiza în două moduri:

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 şi respectiv 6 cu 2, conform fig. 9.3(a)). Un astfel de cablu se numeşte *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 şi 6 cu 6), iar inversarea se face în dispozitivul de la un capăt al cablului, prin legarea inversată a conectorului la circuite: pinii 1 şi 2 la receptor şi 3 şi 6 la emiţător, ca în fig. 9.3(b).

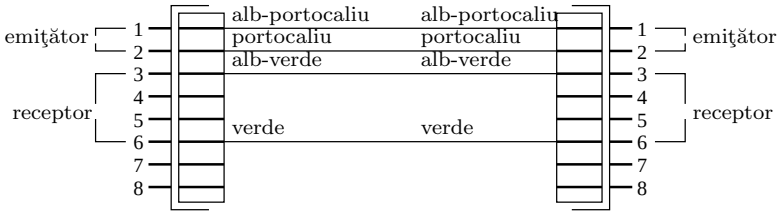
   Standardul recomandă inversarea legăturilor în conectoarele repetoarelor şi cere marcarea conectoarelor cu inversare printr-un simbol „X“.

Conectarea a două echipamente prevăzute cu inversare în conector se face cu ajutorul unui cablu inversor, ca în figura 9.3(c).

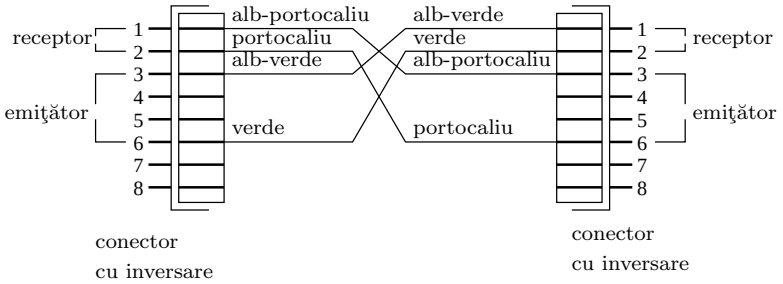
Există şi dispozitive care detectează automat pinii folosiţi la emisie şi recepţie. Aceste dispozitive sunt desemnate *auto MDI/MDIX*. Dacă la unul



(a) Inversare realizată în cablul de legătură



(b) Inversare realizată de unul dintre dispozitivele de la capete



(c) Conectarea a două echipamente prevăzute cu inversare în conector

Figura 9.3: Conectarea a două echipamente 10 Base T. Sunt date şi culorile standard pentru izolaţiile conductoarelor din cablu.

dintre capete se găseşte un astfel de dispozitiv, se poate utiliza atât cablu unu-la-unu cât şi cablu inversor, fără nici un fel de restricţii. Mecanismul de detectare a pinilor utilizaţi se bazează pe pulsurile pentru verificarea mediului, descrise mai jos.

Transmiterea biţilor se face în codificare Manchester. Cele două nivele de tensiune, la emiţător, sunt unul între 2,2 V şi 2,8 V şi celălalt între −2,2 V şi −2,8 V.

Pe lângă transmiterea informaţiei utile, standardul prevede emiterea periodică, de către fiecare echipament, a unui puls de testare a cablului. O interfaţă de reţea sau un repetor care nu primeşte 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 reţea semnalează starea legăturii printr-un bit de control ce poate fi citit de driver-ul plăcii de reţea.

O adăugire ulterioară la standard prevede ca în secvenţa de pulsuri de testare a cablului să se codifice disponibilitatea echipamentului ce le emite de a funcţiona 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 comunicaţie duplex şi care este informat că echipamentul de la celălalt capăt este capabil de asemenea de comunicaţie duplex va intra automat în mod duplex.

Un echipament vechi, datând dinaintea acestei adăugiri la standard, va funcţiona numai în regim semiduplex. Păstrarea compatibilităţii este asigurată de faptul că echipamentul vechi va înţelege pulsurile doar ca testarea liniei, iar pulsurile generate de el este puţin probabil să coincidă întâmplător cu pulsurile de negociere a modului de transmisie.

100 Base Tx. Este foarte asemănător cu 10 Base T, dar obţine o viteză de transmisie de 100 Mbit/s.

Cablul constă tot din două perechi de conductoare torsadate, însă cu proprietăţi mai bune de transmitere a semnalului (obţine aceleaşi caracteristici de atenuare până la frecvenţe de 10 ori mai mari). Cablurile utilizate sunt cele desemnate *UTP Cat 5*. Lungimea maximă a unui tronson este de 100 m.

Conectoarele şi 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 şi 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 şi calitatea cablului; din acest motiv, dacă legăm o placă de reţea 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 informaţiei (adică, întotdeauna trei perechi sunt utilizate pentru transmiterea informaţiei şi a patra este temporar nefolosită).

Codificarea informaţiei este mai specială, utilizând 3 nivele de semnalizare în loc de obişnuitele 2 şi transmiţând simultan pe trei canale, pentru a obţine un semnal ce se încadrează în banda de trecere a unui cablu *Cat 3*.

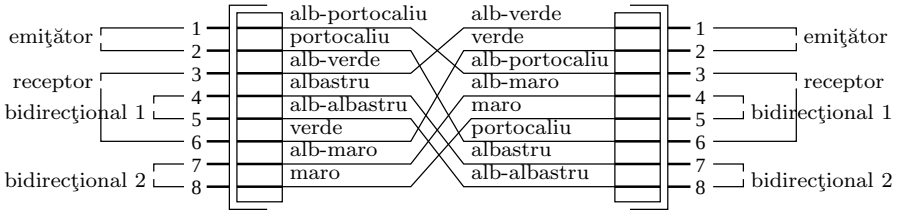


Figura 9.4: Utilizarea pinilor şi conectarea între echipamente la 100 Base T4.

Conectoarele sunt tot RJ45, cu următoarea utilizare a pinilor: emisie: pinii 1 şi 2; recepţie: 3 şi 6, bidirecţional 1: pinii 4 şi 5; bidirecţional 2: pinii 7 şi 8. Ca şi la celelalte cabluri, descrise mai sus, este necesară o încrucişare, realizată fie în cablu, fie în conectorul unuia dintre echipamente. Standardul cere inversarea atât a emisiei cu recepţia cât şi a celor două legături bidirecţionale (fig. 9.4).

Întrucât în majoritatea instalaţiilor 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 şi la 100 Base T4, se utilizează o codificare complicată, însă obţine performanţele 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 raţiuni de compatibilitate, legăturile trebuie să realizeze aceeaşi inversare a unor perechi de fire ca şi la 100 Base T4 (fig. 9.4).

Majoritatea plăcilor de reţea şi celorlalte echipamente pentru reţele IEEE 802.3, produse recent şi desemnate *Ethernet gigabit*, implementează standardele 10 Base T, 100 Base Tx şi 1000 Base T.

1000 Base Cx. Transmite 1 Gbit/s duplex utilizând 2 perechi de conductoare de construcţie specală.

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 nişte conectoare cu 8 pini asemănătoare, dar incompatibile, cu RJ45.

Datorită incompatibilităţii cu 10 Base T şi 100 Base Tx, puţine 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), invelite toate într-o teacă protectoare. Doar 2 perechi (4 fire) sunt utilizate efectiv de legăturile 10 Base T şi 100 Base Tx.

În cadrul fiecărei perechi, unul din fire are izolaţia într-o culoare plină iar celălalt este combinat, alb alternând cu culoarea firului pereche. Culorile folosite pentru perechi sunt portocaliu, verde, albastru şi maro. Menţionăm că se comercializează, din păcate, şi cabluri în care firele ce ar trebui să fie colorate cu alb plus o culoare sunt doar albe şi, 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 şi 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 funcţionează decât pentru reţele 10 Base T şi 100 Base Tx, care nu utilizează deloc perechile albastru şi maro. Pentru 1000 Base T, varianta „semi-inversat“ nu este prevăzută de standard; ca urmare unele echipamente funcţionează cu astfel de cabluri, iar alte echipamente nu funcţionează.

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-maro	albastru
5	alb-albastru	maro	alb-albastru
6	verde	portocaliu	portocaliu
7	alb-maro	albastru	alb-maro
8	maro	alb-albastru	maro

Tabelul 9.1: Ataşarea conectoarelor RJ45

Atragem atenţia că este foarte important să se respecte schema de conectare din următoarele motive:

- Răsucirea firelor afectează transmiterea semnalului şi sensibilitatea la paraziţi. 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 ataşarea corectă a conectorului la al doilea capăt, dacă ataşarea primului s-a făcut nestandard. Amatorii să se gândească la cazul când capătul cablat nestandard se găseşte într-un dulap înghesuit, iar capătul unde trebuie ataşat 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 reţea tensiuni suficient de mari pentru a distruge plăcile de reţea sau hub-urile sau switch-urile ataşate. 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 funcţionării; diferenţele sunt aproape în totalitate aspecte minore legate de realizarea nivelului fizic.

Toate legăturile pe fibră optică sunt punct-la-punct (nu magistrală). Există totuşi un echipament, numit *stea pasivă*, la care se pot conecta mai

multe plăci de reţea şi 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, obţinută cu o astfel de legătură, este în jur de 10^{-9} (1 bit eronat la 10^9 biţi transmişi).

Grupează trei variante, cu diferenţe foarte mici între ele (în general, echipamentele corespunzătoare pot fi interconectate fără probleme):

- 10 Base FP, destinat utilizării în configuraţii cu stea pasivă, ca atare funcţionând în mod semi-duplex.
- 10 Base FB, destinat utilizării în conjuncţie cu multe repetoare în cascadă şi funcţionând în mod semi-duplex.
- 10 Base FL, funcţionând în mod duplex.

Se utilizează o fibră optică cu diametrul miezului de 62,5 μm (şi diametrul exterior, al învelişului, 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 micşorată corespunzător) şi o bandă de 160 MHzkm. Lungimea unui tronson de cablu este maxim 2 km (pentru 10 Base FP, lungimea fiecărei conexiuni dintre placa de reţea şi steaua pasivă este de cel mult 1 km).

Conectarea cablului la echipamente se face cu ajutorul a două conectoare (câte unul pentru fiecare fibră; reamintim că se utilizează o fibră pentru fiecare sens). Conectoarele sunt descrise de standardul IEC 60874-10:1992 sub numele BFOC/2.5. Atenuarea introdusă de conector nu trebuie să depăşească 1 dB, iar puterea undei reflectate nu trebuie să depăşească -25 dB din puterea undei incidente.

Pentru semnalizare se folosesc unde infraroşii cu lungimea de undă cuprinsă între 800 nm şi 910 nm. Nivelul semnalului emiţătorului este între -15 dBm şi -11 dBm pentru 10 Base FP şi între -12 dBm şi -11 dBm pentru 10 Base FB şi 10 Base FL. Nivelul acceptabil pentru semnalul recepţionat este între -41 dBm şi -27 dBm pentru 10 Base FP şi între $-32{,}5$ dBm şi -12 dBm pentru 10 Base FB şi 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, aceşti parametri trebuie configuraţi manual.

100 Base FX: oferă o viteză de transfer de 100 Mbit/s. Pe lângă viteza mai mare, 100 Base FX aduce câteva modificări faţă de 10 Base F, şi anume utilizarea unor conectoare duble (conectând ambele fibre simultan) şi un mecanism de negociere a vitezei de transfer.

1000 Base SX şi 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 şi 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 şi 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 comunicaţie este un cablu format dintr-o pereche de conductoare coaxiale. Impedanţa caractersitică a cablului este de 50 Ω (este deci incompatibil cu cablul utilizat pentru televiziune, care are impedanţa de 75 Ω). Cablul nu are voie să aibă ramificaţii şi trebuie încheiat la ambele capete prin terminatoare. Ramificaţiile sau lipsa terminatoarelor duc la reflexia semnalului la ramificaţie sau la capătul fără terminator, rezultând bruierea semnalului de către reflexia lui.

Pe cablu se leagă, în paralel, interfeţele de reţea şi eventual repetoarele. Derivaţia pentru legătura la interfaţa de reţea este construită special pentru a reduce la minim reflexiile produse (impedanţa emiţătorului şi receptorului este mult mai mare decât impedanţa cablului, anume de cel puţin 100 kΩ); din acest motiv circuitele emiţătorului şi receptorului trebuie plasate la cel mult câţiva 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 faţă de codificarea Manchester clasică, peste semnal este suprapusă o componentă continuă, în scopul simplificării detectării coliziunilor. Pe cablul în repaus, tensiunea între conductoare este 0 V. Dacă o interfaţă emite date, apare o tensiune continuă între conductorul central şi tresă. Dacă două sau mai multe interfeţe emit simultan, tensiunea continuă creşte peste un anumit prag la care se declară coliziune. La detectarea unei coliziuni, interfeţele de reţea conectate la cablu opresc transmisia, conform metodei CSMA/CD (§ 4.2.2).

Există două sub-standarde privitoare la caracteristicile mecanice şi electrice ale cablului de conectare: *10 Base 5* şi *10 Base 2*.

10 Base 5, numit şi *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*) şi lungimea maximă a cablului, în sute de metri.

Cu titlu informativ, dăm câteva detalii, specificate prin standard, cu privire la caracteristicile cablului:

- impedanţa 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 şi maxim 12 dB/km (6 dB pe tot cablul) la 5 MHz;
- se acceptă maxim 100 interfeţe de reţea pe un tronson de cablu.

Cablul trebuie conectat la pământ (la instalaţia de pământare a clădirii) *într-un singur punct*. Se specifică explicit prin standard că atât cablul cât şi elementele legate de el trebuie să fie izolate faţă de pământ sau faţă de alte conductoare (cu excepţia sus-menţionatei unice legături de pământare). De asemenea, interfeţele de reţea trebuie să realizeze o izolare electrică între cablul de reţea şi circuitele calculatorului care să reziste la o tensiune de 1500 V. La efectuarea lucrărilor asupra reţelei, persoanele care lucrează trebuie să aibă grijă să nu atingă simultan cablul de reţea şi un conductor legat la pământ, iar în cazul în care conectează sau deconectează două tronsoane de reţea 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 instalaţiilor 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 reţelelor de alimentare electrică din apropiere sau din cauza fulgerelor.

Circuitele electronice ale interfeţelor de reţea sunt împărţite în două module: un modul, conţinând emiţătorul şi receptorul propriu-zise, se ataşează direct pe cablu; al doilea modul cuprinde logică de comandă şi este construit sub forma unei plăci ce se introduce în calculator.

Ataşarea modulului de semnal la cablul de reţea se poate realiza în două moduri:

- prin conectarea celor două segmente de cablu de-o parte şi de alta prin conectoare standardizate (conectoare coaxiale, numite *conectoare N*, cu prindere cu filet);
- prin realizara 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 emiţător-receptor (engl. *transceiver*) şi modulul de logică al plăcii de reţea sau al repetorului este de asemenea standardizată, sub numele de interfaţă AUI. Cablul de legătură dintre cele două module constă din 5 perechi de conductoare torsadate şi ecranate individual, utilizează conectoare trapezoidale cu 15 pini şi poate avea lungime maximă de aproximativ 50m.

10 Base 2 se mai numeşte *cablu subţire*, *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 foloseşte un cablu mai potrivit pentru cablaje în interior. Lungimea maximă a unui tronson este de 185 m.

Cablul este tot coaxial, dar este mai subţire (≈ 5 mm) pentru a putea fi îndoit mai uşor (standardul cere să poată fi îndoit la raza de 5 cm), în schimb este admis să aibă atenuare mai mare şi, 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 şi maxim 6 dB la 5 MHz;
- maxim 30 interfeţe ataşate pe un tronson.

Conectarea interfeţelor de reţea şi a terminatoarelor se face prin conectori standardizaţi sub numele *BNC*(conectorii BNC sunt standardizaţi pentru aparatură electronică în general, nu se folosesc doar la reţele Ethernet), astfel (fig. 9.5): Fiecare bucată de cablu trebuie să aibă montate pe capete conectoare BNC mamă. Există elemente numite *joncţiuni T* care conţin o ramificaţie şi sunt prevăzute cu un conector BNC mamă (pe mijlocul T-ului) şi două conectoare BNC tată. La conectoarele tată se ataşează bucăţile de cablu de-o parte şi de alta, iar la conectorul mamă al T-ului se ataşează conectorul tată de pe placa de reţea. Terminatoarele sunt prevăzute cu conector BNC mamă şi se ataşează pe T-urile de la plăcile de reţea extreme.

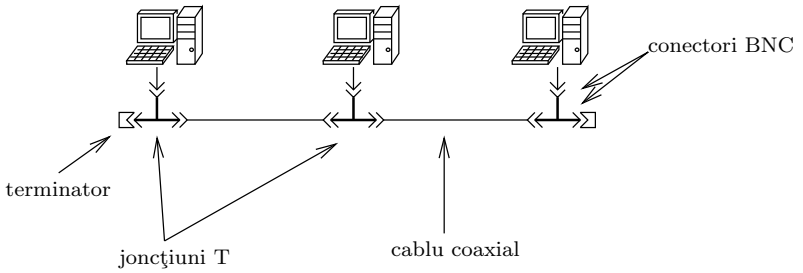


Figura 9.5: Conectarea unei reţele 10 Base 2

Pana cea mai frecventă ce apare la o reţea, afectând conexiunile directe, este întreruperea unui fir (de obicei o pană de contact între sârmă şi conector sau între contactele unui conector). O întrerupere a cablului magistrală duce de regulă la oprirea funcţionării întregii reţele, nu doar „ruperea“ în două a reţelei. Aceasta se întâmplă deoarece capătul de cablu unde s-a produs întreruperea reflectă semnalul (este ca un cablu fără terminator) şi, ca urmare, orice pachet emis pe acel cablu se ciocneşte cu reflexia lui. Soluţia cea mai eficientă de găsire a penei este o căutare binară prin izolarea — neapărat cu terminatoare — a unor porţiuni din ce în ce mai lungi din cablul reţelei.

9.1.4. Repetoarele şi comutatoarele

Ca funcţie în cadrul unei reţele, atât repetorul cât şi comutatorul este un dispozitiv la care sunt conectate mai multe cabluri de reţea şi care, la primirea unui pachet pe un cablu, retransmite pachetul pe toate *celelalte* cabluri conectate. Interfaţa repetorului sau comutatorului către fiecare dintre cabluri se numeşte *port*.

Excepţie făcând cazul comutatoarelor mai evoluate (vezi § 9.1.6.4), o reţea construită cu repetoare sau comutatoare trebuie să aibă o *topologie*

arborescentă, adică între orice două interfeţe de reţea trebuie să existe *un drum şi numai unul* format din cabluri directe şi repetoare sau comutatoare.

Într-o reţea construită corect, arborescent, un pachet emis de o placă de reţea se propagă prin cabluri şi repetoare sau comutatoare din ce în ce mai departe de interfaţa de origine, sfârşind prin a ajunge la toate plăcile din reţea.

În cazul în care reţeaua, în loc să fie arborescentă, conţine circuite, se va întâmpla ca două copii ale aceluiaşi pachet să ajungă pe două căi distincte la un anumit repetor sau comutator. Dacă este un repetor, cele două copii, ajungând aproximativ simultan, vor produce o coliziune. Cum acest lucru se întâmplă cu orice pachet trimis în reţea, fiecare pachet va suferi o coliziune cu el însuşi şi va fi repetat la infinit cu acelaşi insucces, 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 consecinţă, le va memora şi retransmite, fiecare copie fiind retransmisă inclusiv pe calea prin care a intrat cealaltă copie. În acest fel, copiile ciclează la infinit prin reţea, rezultând o „furtună de pachete“, adică o multiplicare incontrolabilă a pachetelor.

În cazul utilizării repetoarelor, pe lângă topologia în arbore mai trebuie respectate nişte condiţii, şi anume:

- toate componentele legate la repetoare trebuie să lucreze la aceeaşi viteză (fie 10 Mbit/s, fie 100 Mbit/s, fie 1 Gbit/s). Aceasta deoarece un repetor nu memorează pachetul de retransmis şi, ca urmare, nu-l poate retransmite la altă cadenţă a biţilor decât cea cu care îl recepţionează.
- să nu existe mai mult de 4 repetoare de-a lungul nici unui drum între două interfeţe de reţea. Această restricţie este impusă pentru ca diferenţele de viteză de transmisie a repetoarelor şi variaţia întârzierii introduse de repetoare să nu ducă la micşorarea sub o anumită limită a timpului dintre două pachete consecutive.
- întârzierea cea mai mare a transmisiei între două interfeţe de reţea (întârzierea pe cablu plus întârzierea introdusă de repetoare) să nu fie mai mare decât jumătate din durata necesară emiterii unui pachet. Pentru o reţea de 10 Mbit/s, aceasta înseamnă o lungime maximă totală de 2500 m între oricare două interfeţe de reţea.

În cazul switch-urilor, nu apare nici una din limitările expuse mai sus, cu excepţia faptului că pe eventualele legături semi-duplex întârzierea trebuie să fie de cel mult jumătate din durata minimă a pachatului.

La reţele ce utilizează atât switch-uri cât şi repetoare, restricţiile de la repetoare se aplică, separat, pe fiecare subreţea formată din repetoare

interconectate şi legăturile acestora spre interfeţele de reţea şi 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 ţine o tabelă cu asocierea între adresa fizică (adresa MAC) a unei interfeţe de reţea şi *portul* (conectorul) la care este conectată, direct sau indirect, acea interfaţă.

Dacă un comutator primeşte un pachet cu o anumită adresă MAC sursă, comutatorul va asocia acea adresă cu portul prin care a intrat pachetul. Ulterior, dacă comutatorul primeşte un pachet având ca destinaţie acea adresă MAC, îl va trimite doar prin portul asociat acelei adrese.

Asocierea între adresa MAC şi port este menţinută doar un timp scurt (de ordinul secundei) pentru ca să se asigure actualizarea asocierii în cazul mutării interfeţei de reţea de la un port la altul (prin mutarea fizică a cablului).

Mai multe adrese MAC pot avea asociat acelaşi port; această situaţie 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 excepţia celui prin care a intrat. Evident, pachetele de broadcast se încadrează în această din urmă categorie.

9.1.6. Facilităţi avansate ale switch-urilor

9.1.6.1. Switch-uri configurabile

În mod obişnuit, switch-urile nu au parametri configurabili şi nu au adresă; ele sunt transparente faţă 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 interfaţă 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 şi cum ar mai avea intern o interfaţă de reţea conectată la el însuşi. Această interfaţă de reţea are o adresă MAC (adesea scrisă pe eticheta aplicată pe carcasă) şi o adresă IP

configurabilă (adresa IP iniţială 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 interfaţa 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 interfaţă web:* Ca şi la conectarea prin telnet, switch-ul prezintă o adresă IP. Administratorul se poate conecta cu orice navigator web la această adresă şi va primi pagini ce conţin parametrii actuali şi formulare pentru modificarea parametrilor. Ca şi în cazul configurării prin telnet, accesul poate şi este recomandabil să fie restricţionat 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 ţine 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 găseşte adresa.

Această facilitate este introdusă pentru a împiedica eventuali intruşi să intre în reţea racordându-se pur şi simplu la prizele de reţea accesibile.

Deşi îmbunătăţeşte securitatea unei reţele, soluţia are câteva limitări:

- lista adreselor asociabile unui port este limitată (de multe ori la 8 sau 16 adrese);
- multe plăci de reţea permit schimbarea (prin soft) a adresei MAC.

9.1.6.3. Trunking

Prin *trunking* se înţelege 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 capacităţilor 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 swithc-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 şi dezactivarea ciclurilor (în sensul teoriei grafelor) formate de legăturile dintre switch-uri.

Majoritatea switch-urilor nu implementează însă acest algoritm şi, ca urmare, în majoritatea cazurilor existenţa ciclurilor duce la „furtuni de pachete“ (multiplicarea incontrolabilă a pachetelor în reţea).

Dacă toate swicth-urile de pe traseul unui ciclu implementează protocolul de descoperire a ciclurilor, ele colaborează automat pentru dezactivarea uneia dintre legături şi utilizarea doar a unui arbore parţial al grafului iniţial al legăturilor. La căderea unei legături, switch-urile vor colabora pentru reactivarea unei legături dezactivate, în vederea păstrării conexităţii reţelei.

Menţionăm că, în cazul existenţei unui ciclu, nu este posibilă împărţirea traficului între drumurile alternative. Unul din drumuri va fi obligatoriu dezactivat complet, cât timp celălalt este funcţional.

9.1.6.5. Reţele virtuale (VLAN)

Mecanismul de *reţele virtuale* (*Virtual Local Area Network*) constă în împărţirea unei reţele fizice în mai multe reţele virtuale disjuncte. Fiecare reţea virtuală se comportă exact ca o reţea IEEE 802.3 independentă. Constructiv, reţelele virtuale partajează aceleaşi echipamente (comutatoare, cabluri sau chiar plăci de reţea).

A nu se confunda VLAN cu *VPN* (*Virtual Private Network — reţea privată virtuală* — descrisă în § 10.7.4).

Partiţionarea 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 posibilităţi de construcţie a VLAN-urilor.

O primă posibilitate constă în partiţionarea 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 aparţină doar unui singur VLAN.

O a doua posibilitate este cea definită în [IEEE 802.1Q, 2003]. Fiecare VLAN constă dintr-o parte din echipamentele (interfeţe de reţea, cabluri şi switch-uri) reţelei fizice; VLAN-uri distincte pot partaja în voie echipamente fizice. Astfel, fiecare interfaţă de reţea aparţine unuia sau mai multor VLAN-uri, fiecare cablu aparţine unuia sau mai multor VLAN-uri şi fiecare port al fiecărui switch aparţine unuia sau mai multor VLAN-uri. Fiecare switch, la primirea unui pachet de broadcast sau pentru a cărui destinaţie nu are asociere,

va trimite pachetul prin toate porturile aparţinând VLAN-ului pachetului, cu excepţia portului prin care a intrat pachetul.

Pentru ca mecanismul descris mai sus să poată funcţiona, este necesar ca, pe cablurile ce aparţin mai multor VLAN-uri, pentru fiecare pachet să se poată deduce cărui VLAN aparţine. Pentru aceasta, fiecare pachet este etichetat cu un *identificator de VLAN* (*VLAN-ID*); acest VLAN-ID este un număr reprezentabil pe 12 biţi.

Pentru păstrarea compatibilităţii cu echipamentele ce nu suportă VLAN-uri 802.1Q, un segment de reţea care aparţine doar unui singur VLAN poate fi configurat să utilizeze pachete neetichetate; switch-ul ce realizează legătura dintre un astfel de segment şi restul reţelei fizice realizează adăugarea şi eliminarea etichetei de VLAN pe pachetele ce tranzitează spre, respectiv dinspre, restul reţelei. Echipamentele incompatibile 802.1Q pot fi montate doar pe cabluri prin care trac pachete neetichetate.

O placă de reţea 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 reţea se comportă ca şi când ar fi de fapt mai multe plăci de reţea, virtuale, câte una în fiecare VLAN. Fiecare placă virtuală are asociat un VLAN-ID, primeşte doar pachetele ce poartă acel VLAN-ID şi marchează cu VLAN-ID-ul său toate pachetele emise.

Pe fiecare switch trebuie configurate porturile care aparţin 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 aparţine unui singur VLAN.

9.1.7. Considerente privind proiectarea unei reţele

Proiectarea şi construcţia unei reţele Ethernet este în general extrem de uşoară; acesta este şi unul din motivele popularităţii Ethernet-ului.

De obicei este necesar să se construiască o reţea în care toate calculatoarele să se „vadă“ între ele (la nivel de reţea 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 şi cabluri astfel încât fiecare calculator să fie legat la un switch şi switch-urile să fie legate între ele într-o reţa conexă şi fără „bucle“.

Se utilizează de obicei o structurare ierarhică a legăturilor (aşa-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 reţelele mai mari, între switch-ul central şi switch-urile asociate încăperilor se mai adaugă un nivel.

În instalaţiile mici şi fără pretenţii, cablurile se duc aparent şi se fixează de pereţi sau pe mobilă numai acolo unde este strict necesar. Uşurinţa realizării şi reconfigurării este plătită prin faptul că apar dificultăţi la curăţenie, cablurile se degradează uşor dacă se calcă pe ele şi, în sfârşit, se mai întâmplă ca cineva să se împiedice de un cablu, rezultând echipamente trase pe jos sau cabluri smulse din conectoare.

Pentru evitatarea neajunsurilor expuse mai sus, se preferă să se tragă cablurile prin paturi de cablu, tuburi îngropate (ca la instalaţiile 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 posibile 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 necesităţi.
- 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 reţele distincte, o parte din calculatoarele din încăpere conectându-se la o reţea şi altele la altă reţea, se vor pune două switch-uri în încăpere, fiecare conectat prin câte un cablu la switch-ul central.

9.2. Reţele IEEE 802.11 (Wireless)

9.2.1. Arhitectura reţelei

Elementul de bază într-o reţea 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 staţii (STA) situate într-o zonă geografică destul de restrânsă (de ordinul câtorva zeci de metri) pentru ca semnalul emis de fiecare staţie să fie recepţionat de toate staţiile din celulă.

Fiecare celulă are asociat un identificator de 48 de biţi, unic, numit *Basic Service Set ID* — BSSID. Acest identificator este înscris în fiecare pachet de date vehiculat în reţea, astfel încât pentru orice pachet de date recepţionat prin antenă se poate determina celula wireless căreia îi aparţine. Mai multe celule wireless pot coexista în aceeaşi zonă, traficul din cadrul fiecărei celule putând fi distins pe baza BSSID-ului de traficul celorlalte celule.

Fiecare staţie aparţine (este asociată) la un anumit moment cel mult unei celule. Asocierile sunt dinamice — o staţie poate să intre sau să iasă

oricând dintr-o celulă. Fiecare staţie se identifică printr-o *adresă* unică de 48 de biţi, numită în mod curent *adresa MAC* a staţiei.

Accesul la mediu este controlat în principal prin metode bazate pe urmărirea traficului pe mediu, detectarea coliziunilor şi, într-o anumită măsură, metode de rezervare în prealabil a accesului la mediu. Acestea vor fi descrise în detaliu în § 9.2.2.

Prezenţa unei celule wireless organizate într-o anumită zonă este manifestată prin emiterea periodică de către una dintre staţii a unui pachet special, numit *beacon*. Pe lângă BSSID-ul celulei, pachetele *beacon* mai conţin un şir de caractere numit SSID sau uneori *numele reţelei* (engl. *network name*). Acest şir este fixat de administratorul reţelei şi serveşte la identificarea reţelei pentru utilizatorii umani.

O staţie poate obţine lista celulelor active în zona sa ascultând pachetele *beacon*. Lista afişată utilizatorului va conţine SSID-urile reţelelor.

Există două moduri de lucru în care poate funcţiona o reţea 802.11:

- Reţea formată dintr-o singură celulă independentă, neconectată prin mijloace IEEE 802 de alte echipamente. În terminologia standardului, o astfel de celulă se numeşte *Independent BSS* — IBSS; în mod curent reţeaua astfel formată se numeşte *ad-hoc*.
- Reţea formată din una sau mai multe celule, operând împreună şi posibil conectate la o infrastructură IEEE 802 (de exemplu la o reţea Ethernet — 802.3). Un astfel de mod de lucru se numeşte *mod infrastructură* sau *managed*.

În mod infrastructură, în cadrul fiecărei celule există o staţie care are rolul legării celulei la infrastructură (altfel spus, la restul reţelei IEEE 802.11). O astfel de staţie poartă denumirea de *Access Point* — AP. Un AP este o staţie, şi ca atare are o adresă MAC. Într-o celulă a unei reţele de tip infrastructură, o staţie ce intră sau iese dintr-o celulă trebuie să anunţe AP-ul responsabil de celula respectivă.

AP-ul este responsabil de generarea pachetelor *beacon* şi BSSID-ul celulei este adresa MAC a AP-ului.

AP-urile unei aceleiaşi reţele 802.11 trebuie să fie interconectate, formând aşa-numitul *Distribution System* (DS). DS-ul poate fi conectat la alte reţele din familia IEEE 802 prin intermediul unor dispozitive numite *portal*-uri. Celulele din aceeaşi reţea vor avea acelaşi SSID.

Standardul original nu prevede nimic în legătură cu modul de conectare a AP-urilor şi deci de realizare a DS-ului. Ca urmare, fiecare fabricant de AP-uri şi-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* conţine un AP şi un portal către reţele Ethernet. Un astfel de dispozitiv prezintă un modul radio prin intermediul căruia se comportă ca o staţie cu rol de AP şi un conector Ethernet. Într-o primă aproximaţie, un astfel de dispozitiv poate fi privit ca un *switch* conectat pe de o parte la fiecare dintre staţiile membre ale celulei şi pe de altă parte la un dispozitiv Ethernet.

Unele *access point*-uri ce se găsesc în comerţ oferă funcţionalităţi suplimentare faţă de un AP combinat cu un portal. Aceste funcţii sunt oferite prin extensii ale protocolului şi ca urmare pot fi utilizate de regulă doar împreună cu echipamente produse de aceeaşi firmă. Funcţionalităţile sunt:

- funcţie de *switch* (punte) între o reţea Ethernet (fixă) şi o celulă *wireless*, acţionând însă ca şi staţie oarecare (nu AP). Această funcţie se numeşte *wireless bridge* sau *AP client* (uneori există funcţii cu ambele nume, cu diferenţe minore între ele);
- funcţie de AP, dar utilizând tot reţeaua *wireless* pentru partea de infrastructură. În acest mod, dispozitivul este în acelaşi timp AP pentru o celulă şi staţie oarecare în altă celulă, iar a două legătură este utilizată pentru dirijarea spre reţeaua fixă a datelor din celula în care dispozitivul este AP.

9.2.2. Accesul la mediu

Deoarece într-o reţea 802.11 avem un mediu partajat între mai mulţi emiţători, este necesar să avem un mecanism de control al accesului la mediu. Metoda de control al accesului la mediu în IEEE 802.11 se numeşte *Carrier-Sense Multiple Access with Collision Avoidance* (CSMA/CA; rom: acces multiplu cu detectarea semnalului purtător şi evitarea coliziunilor).

În principal, strategia de control al accesului la mediu se bazează pe detectarea coliziunilor şi repetarea pachetelor ce au suferit coliziuni, adică aceeaşi strategie ca şi pentru *Ethernet*-ul pe cablu coaxial.

Datorită condiţiilor specifice reţelelor fără fir, sunt aduse câteva modificări. În principal, la transmisia radio nu există o delimitare comună între zonele de acţiune pentru diverse staţii din aceeaşi celulă: este posibil ca o staţie B să recepţioneze bine transmisia staţiei A, staţia C să recepţioneze transmisia lui B, dar staţia C să nu recepţioneze transmisia lui A. Într-un astfel de caz, dacă A şi C transmit simultan, pachetele emise se ciocnesc la B, dar deoarece nici

una din staţiile A şi C nu recepţionează transmisia celeilalte ele nu au cum să detecteze coliziunea.

În reţelele IEEE 802.11, o staţie care doreşte să trimită un pachet va trimite întâi un pachet de control, numit *Request To Send* (RTS; rom: *cerere de transmisie*), în care specifică destinatarul şi durata de timp necesară transmiterii pachetului. Dacă destinatarul a primit pachetul RTS şi este liber, va trimite înapoi un pachet de control *Clear To Send* (CTS; rom: *accept transmisia*). La primirea pachetului CTS, emiţătorul trimite pachetul de date.

O staţie care recepţionează un pachet CTS destinat altei staţii nu are voie să trimită nimic pe durata rezervată de pachetul CTS, pentru a nu interfera cu transmisia acceptată prin acel CTS. Această restricţie trebuie respectată şi în cazul recepţiei unui pachet CTS destinat altei reţele din aceeaşi zonă (adică purtând un BSS-ID diferit).

Utilizarea pachetelor RTS şi CTS nu este obligatorie. Pentru pachetele mici este preferabilă trimiterea direct a pachetului de date şi repetarea acestuia în cazul unei coliziuni. Pentru pachetele de *broadcast*, utilizarea RTS şi 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 staţiile membre ale celulei IBSS. Simplificat, o staţie care nu recepţionează un *beacon* într-un anumit interval de timp predefinit generează ea însăşi pachetul *beacon*.

9.2.4. Securitatea reţelelor 802.11

Deoarece la reţelele 802.11 comunicaţia este prin unde radio, a căror domeniu de acţiune nu poate fi net limitat, utilizarea unor metode care să asigure confidenţialitatea şi integritatea datelor transportate este esenţială.

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 doreşte să se ofere acces public la Internet. De remarcat însă că, datorită lipsei oricărui mecanism de confidenţialitate sau asigurarea integrităţii mesajelor, oricine poate asculta sau modifica comunicaţia oricui în cadrul celulei.
- *Wired Equivalent Privacy — WEP* (rom. *securitate echivalentă cu reţeaua cablată*): oferă confidenţialitate şi autentificarea şi verificarea integrităţii

mesajelor. În acest scop, toţi 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 iniţierea celulei sau, după caz, la introducerea staţiei în celulă. Criptarea se face utilizând cifrul RC4, cu o cheie construită din secretul partajat şi dintr-un *vector de iniţializare* ales aleator, pentru fiecare pachet, de către emiţător şi transmis în antetul pachetului. Controlul integrităţii pachetului este făcut tot pe baza secretului partajat. WEP are două slăbiciuni: pe de o parte, datorită existenţei unei slăbiciuni a cifrului RC4 (există câteva chei slabe, foarte uşor de spart), WEP poate fi spart destul de uşor; pe de altă parte, modelul de securitate oferit este destul de neflexibil.

- *WiFi Protected Access — WPA*: corectează problemele WEP, păstrând compatibilitatea cu plăcile de reţea existente. În privinţa 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 privinţa obţinerii unui model de securitate mai flexibil, WPA are două moduri de lucru:

  - *WPA-Personal*, numit şi *WPA-PSK* (de la *Pre-Shared Key*), în care se utilizează un secret partajat între toţi membrii celulei, fiind similar cu WEP (dar mult mai sigur).
  - *WPA-Entreprise*, în care cheile se obţin pe baza unor chei individuale ale utilizatorilor. Controlul accesului şi obţinerea cheilor se face printr-un mecanism numit *Extensible Authentication Protocol* (*EAP*), descris mai jos.

- *IEEE 802.11i* [IEEE 802.11i, 2004], numit şi *WPA2*, extinde *WPA* adăugând, între altele, posibilitatea utilizării cifrului AES. Ca şi 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 şi de alte protocoale în afară de *WPA* şi *WPA2*, şi anume poate fi utilizat în cadrul legăturilor *PPP* [RFC 1661, 1994], precum şi pentru autentificarea conectărilor la o reţea cablată IEEE 802.3, conform [IEEE 802.1X, 2001].

Arhitectura *EAP* conţine următoarele componente:

- *clientul* ce trebuie să-şi dovedească identitatea în scopul obţinerii accesului

la reţea. Rolul clientului îl are placa de reţea 802.11 (sau placa de reţea 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 reţelei. Rolul de punct de acces îl are AP-ul 802.11 (sau switch-ul 802.3 sau serverul PPP). În terminologia *EAP*, acesta se numeşte *authenticator*.
- *serverul de autentificare* este entitatea care deţine baza de date cu cheile clienţilor şi realizează efectiv autentificarea.

Protocolul *EAP* prevede un schimb de mesaje între client şi serverul de autentificare. Dacă serverul de autentificare este distinct faţă de punctul de acces, comunicaţia dintre client şi serverul de autentificare trece prin punctul de acces, iar porţiunea din calea de comunicaţie dintre punctul de acces şi serverul de autentificare este protejată criptografic pe baza unui secret partajat între punctul de acces şi 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. Funcţionarea mecanismului necesită ca serverul să aibă în baza de date, în clar, parola clientului. *EAP-MD5* permite doar autentificarea clientului, nu şi stabilirea unor chei pentru criptarea sau autentificarea mesajelor.
- *EAP-TLS* necesită ca atât clientul cât şi serverul de autentificare să aibă prestabilite chei secrete SSL/TLS, iar fiecare dintre ei să aibă certificatul TLS al celuilalt (vezi şi § 11.3.2.5). Se stabileşte o conexiune TLS între client şi serverul de autentificare, utilizând certificatele acestora, iar în cadrul acestei conexiuni stabilesc cheile pentru comunicaţia ulterioară între client şi punctul de acces.
- *PEAP* (de la *Protected EAP*) prevede utilizarea TLS pentru deschiderea unei conexiuni securizate între client şi 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 comunicaţiei între client şi punctul de acces. În terminologia *PEAP*, conexiunea *TLS* se numeşte *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 reţea (*Internet Protocol*, *IP*, *protocolul Internet*), iar pe de altă parte reţeaua Internet, care este o reţea la scară mondială bazată pe protocolul Internet.

Capitolul de faţă prezintă:

- protocolul Internet (IP), împreună cu celelalte protocoale de bază ale reţelelor de tip Internet (TCP, DNS, ARP, etc.);
- câteva aspecte administrative legate de reţeaua mondială Internet.

10.1. Arhitectura reţelei

Facem în continuare o scurtă trecere în revistă a conceptelor de bază ale unei reţele bazate pe protocolul Internet. Aceste concepte vor fi detaliate în paragrafele care urmează.

Serviciul de comunicaţie oferit de o reţea Internet este de tip datagrame; în terminologia Internet acestea se numesc *pachete*.

Ca orice reţea (vezi capitolul 5), o reţea Internet este alcătuită din *noduri*, interconectate între ele. Într-o reţea Internet, toate nodurile pot acţiona ca noduri finale (adică să fie sursă sau destinaţie pentru comunicaţie). Sunt numite *staţii* (engl. *hosts*) nodurile ce nu pot acţiona ca noduri intermediare şi *rutere* nodurile ce pot acţiona ca noduri intermediare.

Staţiile sunt în mod uzual calculatoare (PC-uri, mainframe-uri), dispozitive mobile (PDA-uri), imprimantele de reţea sau alte dispozitive. Remarcăm că switch-urile Ethernet sunt noduri IP numai dacă sunt configurabile. În acest caz, ele au doar rol de staţie şi 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, reţele locale IEEE 802, cablu TV, etc. Modul de utilizare a fiecărui tip de legătură directă de către o reţea Internet este standardizat prin standarde auxiliare (§ 10.5). Există chiar un standard [RFC 1149, 1990] de utilizare ca legături directe a porumbeilor călători; deşi standardul a fost publicat ca o glumă de 1 aprilie, el ilustrează foarte bine independenţa între nivele într-o reţea.

Din punctul de vedere al unei reţele Internet, o legătură directă este orice fel de canal de comunicaţie pe care reţeaua de tip Internet o poate folosi.

Fiecare nod este identificat prin una sau mai multe *adrese IP*. Cu excepţia 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*. Corespondenţa între un nume de domeniu şi adresa IP se realizează cu ajutorul sistemului DNS (*Domain Name Service*), descris în § 10.4.

Protocolul Internet a fost proiectat pentru a asigura o toleranţă deosebit de mare la pene. După căderea unor noduri sau a unor legături, dacă mai există totuşi un drum între două noduri el va fi găsit şi utilizat în cele din urmă. Această toleranţă la pene vine cu un preţ: nu există garanţii 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 şi în caz de încărcare mare a reţelei), să ajungă în dublu exemplar sau două pachete să ajungă la destinaţie în ordine inversă a trimiterii.

Este sarcina nivelelor superioare să se descurce în aceste condiţii. În acest scop, între aplicaţie şi nivelul reţea este plasat un nivel intermediar, *nivelul transport*, cu rolul de-a furniza aplicaţiei un serviciu mai potrivit.

10.2. Protocolul IP

Protocolul Internet (engl. *Internet Protocol — IP*) descrie formatul pachetelor şi câteva aspecte privind activitatea nodurilor reţelei.

Protocolul IP are două versiuni aflate curent în uz: versiunea 4 (cea mai utilizată în prezent, numită prescurtat *IPv4*) standardizată prin [RFC 791, 1981] şi versiunea 6 (care se răspândeşte 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 opţiuni şi, în final, datele utile.

Antetul fix conţine datele necesare pentru dirijarea pachetului. Conţinutul antetului fix este dat în tabelele 10.1 (pentru versiunea 4) şi 10.2 (pentru versiunea 6). Semnificaţia diferitelor câmpuri va fi descrisă în paragrafele care urmează.

Nume câmp	Lungime (biţi)	Rol
Versiune	4	Versiunea protocolului; valoarea este fixă: 4.
IHL	4	Lungimea antetului, inclusiv opţiunile, în grupuri de 32 biţi (valoarea minimă este 5, adică 160 biţi).
TOS	8	Tip serviciu (vezi § 10.2.6.2).
Lungime totala	16	Lungimea totală, antet plus date utile, în octeţi.
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 viaţă	8	Timpul rămas până la distrugerea pachetului (vezi § 10.2.5.3).
Protocol	8	Identificarea protocolului de nivel superior căruia îi aparţin datele utile.
Suma de control	16	Suma de control a antetului.
Adresă sursă	32	Adresa nodului ce a creat pachetul.
Adresă destinaţie	32	Adresa destinatarului final al pachetului.

Tabelul 10.1: Antetul IP versiunea 4

Opţiunile sunt informaţii pentru dirijarea pachetului pentru cazuri mai speciale; deoarece aceste informaţii nu sunt necesare decât pentru anumite tipuri de pachete, ele sunt prezente doar în pachetele în care este nevoie de

Nume câmp	Lungime (biţi)	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 octeţi.
Tip antet următor	8	Dacă există opţiuni, tipul primului antet opţional; altfel, protocolul căruia îi aparţin 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 destinaţie	128	Adresa destinatarului final al pachetului.

Tabelul 10.2: Antetul IP versiunea 6

ele.

Datele utile sunt un şir de octeţi asupra căruia protocolul IP nu impune nici o restricţie, cu excepţia lungimii. Lungimea maximă admisă de protocol este de 65515 octeţi (65535 octeţi pachetul întreg) pentru IPv4 şi 65535 octeţi, inclusiv antetele opţionale, pentru IPv6. Este permis ca unele noduri să nu poată procesa pachete în care datele utile sunt mai lungi de 556 octeţi (576 octeţi tot pachetul) pentru IPv4 şi 1240 octeţi (1280 octeţi tot pachetul) pentru IPv6 (a se vedea şi § 10.2.6.1).

10.2.2. Bazele dirijării pachetelor IP

10.2.2.1. Subreţele şi interfeţe

O *subreţea* este o mulţime de noduri legate direct fiecare cu fiecare. De exemplu, o reţea Ethernet construită cu cabluri magistrală este o subreţea IP. O reţea Ethernet cu hub-uri sau switch-uri este de asemenea o subreţea IP întrucât, din punctul de vedere al calculatorului la care este ataşată o placă de reţea, o reţea Ethernet construită cu cablu magistrală se comportă identic cu o reţea construită cu hub-uri sau switch-uri. Ca alt exemplu, o linie serială construieşte o subreţea cu două calculatoare.

Interfaţa de reţea este un concept abstract care desemnează legătura dintre un nod şi o subreţea. În cazul în care legătura directă este realizată de

o reţea IEEE 802, interfaţa de reţea este placa de reţea împreună cu driver-ul ei.

Fiecare interfaţă de reţea are propria adresă IP. Ca urmare, un nod ce are n plăci de reţea va avea n adrese IP distincte.

Are sens să vorbim despre *interfeţele membre ale unei subreţele*, ca fiind interfeţele prin care nodurile din subreţea sunt conectate la acea subreţea. Adresele IP dintr-o subreţea sunt adresele IP ale interfeţelor din acea subreţea.

10.2.2.2. Prefixul de reţea

Fiecare subreţea are asociat un *prefix de reţea*, adică un anumit şir de biţi de lungime mai mică decât lungimea unei adrese IP. Toate adresele IP ale interfeţelor din acea subreţea trebuie să înceapă cu acel prefix de reţea. Prefixul unei subreţele nu este permis să fie prefix al unei adrese IP din afara acelei subreţele. Ca urmare, un prefix identifică unic o subreţea.

Sufixul unei adrese, adică şirul de biţi din adresă care nu fac parte din prefixul subreţelei, îl vom numi *adresa în cadrul subreţelei*. Numărul de biţi ai sufixului determină numărul de noduri ce pot fi membre ale subreţelei.

Adresele în care sufixul este format numai din biţi 0 sau numai din biţi 1 (aşadar două adrese pentru fiecare subreţea) sunt rezervate şi nu pot fi asignate nodurilor reţelei (a se vedea şi [RFC 1700, 1994]).

Exemplul 10.1: Pentru simplificarea exemplului vom presupune că adresele IP sunt doar de 8 biţi. Presupunem că o subreţea ar avea prefixul de reţea 10110. Adresa 10110010, dacă există, trebuie să desemneze o interfaţă din acea reţea.

Adresa 10111010 nu poate face parte din acea subreţea, deoarece nu începe cu prefixul subreţelei. Notăm că un nod care are o interfaţă în subreţeaua 10110 şi o interfaţă în altă reţea ar putea avea adresa 10111010 pe cea de-a doua interfaţă.

Din subreţeaua considerată, cu prefixul 10110, pot face parte adresele, în număr de $2^3 = 8$, din intervalul 10110000–10110111. Adresele 10110000 şi 10110111 sunt rezervate; rămân deci 6 adrese ce pot fi asignate nodurilor subreţelei.

Un exemplu de asignare a adreselor este prezentat în figura 10.1. Pătrăţelele numerotate reprezintă calculatoarele, iar liniile reprezintă legăturile directe, figurate aici ca şi când ar fi realizate prin cabluri magistrală. De remarcat că nodul cu numărul 3 are două adrese, 10110001 şi 10111010, câte una pentru fiecare interfaţă.

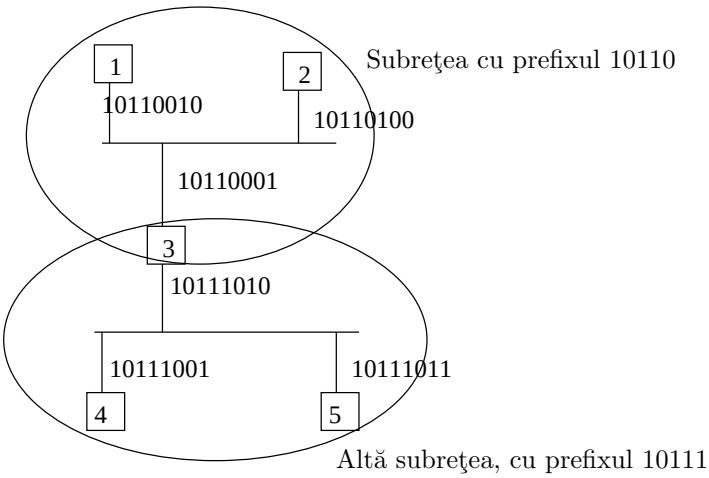


Figura 10.1: O reţea formată din două subreţele. Vezi exemplul 10.1

10.2.2.3. Tabela de dirijare

La primirea unui pachet IP, un nod execută următorul algoritm:

1. dacă adresa destinaţie este una din adresele nodului curent, pachetul este livrat local (nivelului superior);
2. altfel, dacă adresa destinaţie este adresa unui vecin direct, pachetul este livrat direct acelui vecin;
3. altfel, se determină vecinul direct cel mai apropiat de destinatarul pachetului şi i se dă pachetul, urmând ca acesta să-l trimită mai departe.

Pentru pasul 2, este necesar ca nodul să determine dacă adresa destinaţie corespunde unui vecin direct şi care este interfaţa prin care se realizează legătura. Livrarea efectivă este realizată de interfaţa de reţea; acesteia i se dă pachetul şi 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 interfeţe, trebuie utilizată interfaţa prin intermediul căruia el este vecin nodului curent. O dată determinată adresa interfeţei, trimiterea pachetului se face ca la pasul 2.

Deciziile de la paşii 2 şi 3 se iau pe baza *tabelei de dirijare* a nodului curent. O tabelă de dirijare constă dintr-o mulţime de *reguli de dirijare*. Fiecare regulă asociază o *ţintă* unui *grup de adrese* destinaţie.

Grupul de adrese este specificat printr-un prefix de reţea. Pentru un pachet dat se aplică acea regulă de dirijare în care prefixul ce specifică grupul este prefix al adresei destinaţie 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.

Ţinta poate fi fie o interfaţă, fie o adresă IP.

Dacă ţinta este o interfaţă, destinaţia trebuie să fie un vecin direct, accesibil prin acea interfaţă; în acest caz pachetul de dirijat este livrat direct destinatarului prin interfaţa dată în regulă, conform pasului 2.

Dacă ţinta este o adresă, aceasta trebuie să fie adresa unei interfeţe 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* şi trebuie să fie configurat să acţioneze ca nod intermediar.

De notat că adresa sursă şi adresa destinaţie din antetul IP nu se modifică în cursul acestei proceduri. Sursa rămâne nodul care a emis pachetul, iar destinaţia rămâne nodul căruia trebuie să-i fie livrat în cele din urmă pachetul. Atunci când modulul de reţea pasează unei interfeţe de reţea un pachet în vederea transmiterii pachetului către un nod vecin, modulul de reţea va comunica interfeţei două lucruri: pachetul, în care adresa destinaţie reprezintă destinatarul final, şi adresa vecinului direct căruia interfaţa îi va livra pachetul. Acesta din urmă poate fi diferit faţă de destinatarul final dacă este doar un intermediar pe drumul către destinatarul final.

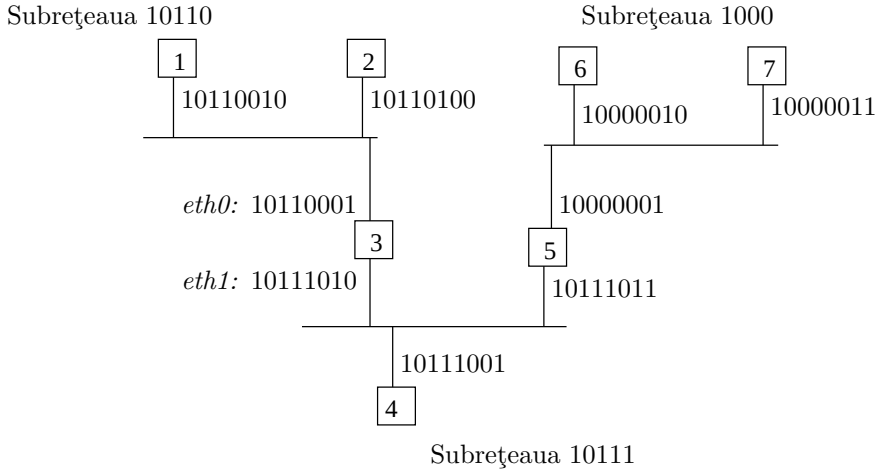


Figura 10.2: O reţea pentru exemplul 10.2

Exemplul 10.2: Fie reţeaua din figura 10.2, formată din trei subreţele. Pentru nodul nr. 3, au fost figurate şi numele interfeţelor de reţea: *eth0* către subreţeaua de sus şi *eth1* către subreţeaua de jos. Tabela de dirijare a nodului 3 este cea ilustrată în tabelul 10.3.

Nr. crt.	Grup de adrese (prefix)	Ţinta
1.	10110	interfaţa *eth0*
2.	10111	interfaţa *eth1*
3.	1000	nodul 10111011

Tabelul 10.3: Tabela de dirijare pentru nodul 3 din figura 10.2 (exemplul 10.2).

Considerăm că nodul 3 primeşte un pachet cu destinaţia 10110010 (nodul 1). Singura regulă aplicabilă este regula 1, deoarece 10110 este prefix pentru 10110010. Conform acestei reguli, pachetul poate fi livrat direct prin interfaţa *eth0*.

Fie acum un pachet cu destinaţia 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 destinaţia 10111011 şi găseşte regula 2, conform căreia pachetul se trimite prin interfaţa *eth1*. Prin urmare, pachetul destinat lui 10000010 va fi trimis lui 10111011 prin interfaţă *eth1* (urmând ca nodul 5 să-l livreze mai departe nodului 6).

De remarcat că nodurile ce apar ca ţintă în regulile tabelelor de dirijare trebuie specificate prin adresele interfeţelor accesibile direct din nodul curent. În exemplul 10.2, este esenţial ca, în ultima regulă a tabelei de dirijare a nodului 3, nodul 5 să fie specificat prin adresa 10111011 şi nu prin adresa 10000001. Nodul cu adresa 10111011 este accesibil prin interfaţa *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 şi, ca urmare, aplicată pentru pachetele pentru care nu este aplicabilă nici o altă regulă. Această regulă, dacă există, are totdeauna ca ţintă 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 şi prefixelor

10.2.3.1. Scrierea adreselor IP

Atunci când o adresă IP este scrisă pe hârtie sau într-un fişier text, se afişează pe ecran sau se citeşte de la tastatură, adresa este scrisă într-un format standard descris în continuare.

Adresele IP versiunea 4, care sunt şiruri de 32 de biţi, se scriu ca şir de 4 numere, scrise în baza 10, separate prin puncte. Fiecare număr este de fapt valoarea câte unui grup de 8 biţi, văzut ca număr. Această scriere se numeşte *notaţie zecimală cu punct*.

Pe lângă notaţia zecimală cu punct, adresele IP versiunea 4 pot fi scrise în *notaţia 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 (liniuţele au fost scrise numai pentru uşurarea citirii). Notaţia zecimală cu punct este 192.168.0.34. Notaţia zecimală simplă este 3232235554.

Adresele IP versiunea 6 sunt şiruri de 128 de biţi. Scrierea lor obişnuită se face ca un şir de 32 cifre hexa, fiecare reprezentând câte 4 biţi 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 compatibilităţii cu IPv4, este acceptată scrierea în care primii 96 biţi sunt scrişi în format IPv6, iar ultimii 32 de biţi sunt scrişi în format IPv4, separaţi de primii printr-un caracter *două puncte*.

Exemplul 10.5: Adresa `0:0:0:0:0:0:c100:e122` se poate scrie şi

`0:0:0:0:0:0:193.0.225.34`

sau, mai compact,

`::193.0.225.34`

10.2.3.2. Scrierea prefixelor de reţea

Prefixele de reţea fiind de lungime variabilă, trebuie precizată atât valoarea efectivă a prefixului cât şi lungimea acestuia. Există două notaţii: notaţia cu adresa subreţelei şi lungimea prefixului şi notaţia cu adresa subreţelei şi masca de reţea.

În notaţia cu adresă şi lungime, prefixul se completează cu zerouri la lungimea unei adrese IP (adică la 32 de biţi pentru versiunea 4 şi la 128 de biţi pentru versiunea 6); rezultatul se numeşte *adresa de reţea*. Adresa de reţea se scrie ca şi când ar fi o adresă IP normală, după care se scrie (fără spaţiu) 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 notaţia cu adresă de reţea şi lungime (notaţie cu slash) 192.168.192.0/19. Prefixul 1100-0000-1010-1000-1100-0000 se scrie 192.168.192.0/24. De remarcat importanţa specificării lungimii.

În notaţia cu adresă de reţea şi mască de reţea, se scrie mai întâi adresa de reţea, ca şi în cazul scrierii cu adresă şi lungime, după care se scrie (cu un *slash* între ele sau în rubrici separate) aşa-numita *mască de reţea*. Masca de reţea constă dintr-un şir de biţi 1 de lungimea prefixului de reţea urmat de un şir de biţi 0, având în total lungimea unei adrese IP. Mască de reţea, se scrie ca şi când ar fi o adresă IP.

Notaţia cu adresă şi mască se utilizează numai pentru IP versiunea 4.

Exemplul 10.7: Prefixul 1100-0000-1010-1000-110 se scrie, în notaţia cu adresă de reţea şi masca, 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 şi prefixelor de reţea

Alocarea adreselor IP pentru reţeaua mondială Internet se realizează de către *Internet Assigned Numbers Authority* (*IANA*). Mai multe despre alocare se găseşte la [IANA,]. Deşi nu este actualizat, este instructiv de citit şi [RFC 1700, 1994].

10.2.4.1. Alocarea pe utilizări

Adresele IPv4 sunt împărţite după cum urmează:

- Adresele cu prefixele 0.0.0.0/8 şi şi 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 îşi afle propria adresă.
- Adresele cu prefixul 224.0.0.0/4 sunt utilizate ca adrese de multicast (aşa-numita *clasă D*).
- Adresele cu prefixul 240.0.0.0/4 sunt rezervate (aşa-numita *clasă E*).
- Adresele cu prefixele 10.0.0.0/8, 172.16.0.0/12 şi 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 condiţia ca pachetele purtând astfel de adrese ca sursă sau destinaţie să nu ajungă în afara nodurilor gestionate de acea persoană sau instituţie. Aceste adrese se utilizează pentru acele noduri, din reţeaua proprie a unei instituţii, 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 şi 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 soluţie nu este realizabilă practic la scară mondială. Din această cauză, adresele se alocă instituţiilor doritoare în blocuri de adrese, fiecare bloc având un prefix unic, întocmai ca în cazul subreţelelor.

Un bloc de adrese se alocă unei subreţele sau grup de subreţele interconectate care apar, din restul Internetului, ca o singură subreţea. Din afara subreţelei 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, şi nu câte o regulă pentru fiecare nod.

Pentru stabilirea dimensiunilor blocurilor, iniţial adresele IP versiunea 4 au fost împărţite în *clase*:

A: Adresele cu prefixul 0.0.0.0/1 au fost împărţite în 128 blocuri alocabile, fiecare bloc având câte 2^{24} adrese. Lungimea prefixului unui bloc este de 8 biţi.

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 ineficientă, 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 ruterelor 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 situaţii, 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 reţea al unui nod, ca urmare a unei erori apărute în livrarea unui pachet IP. Pachete ICMP mai pot fi generate şi de programe utilizator, prin intermediul *socket*-urilor de tip `SOCK_RAW`. Astfel de aplicaţii servesc la testarea funcţionării reţelei.

O dată generat, un pachet ICMP este transmis prin reţea ca orice alt pachet IP. Ajuns la destinaţie, modulul de reţea (IP) al nodului destinaţie examinează câmpul *protocol* şi, constatând că este vorba de un pachet ICMP, îl livrează modulului ICMP al nodului destinaţie. Modulul ICMP trebuie să fie prezent şi funcţional în orice nod IP; în implementările obişnuite este parte a nucleului sistemului de operare al calculatorului ce constituie nodul.

Datele utile sunt formatate conform standardului ICMP şi încep cu doi întregi pe câte 8 biţi reprezentând tipul şi 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 octeţi din pachetului IP care a dus la generarea pachetului ICMP.

Situaţiile ce duc la generarea pachetelor ICMP, precum şi acţiunile 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 destinaţiei pachetului; sau
- interfaţa de reţea prin care trebuie trimis pachetul nu este funcţională 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 destinaţie:* 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 şi 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 şi flagul *nu fragmenta* setat; vezi § 10.2.6.1
	5 — source route failed	pachetul a avut opţiunea *dirijare de către sursă* şi ruta specificată este invalidă.
11 — time exceeded	0 — TTL exceeded	pachetul se află de prea mult timp în reţea (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 1 — host 2 — TOS & network 3 — TOS & host	redirecţionare, § 10.2.5.5
8 — echo request		cerere ecou, § 10.2.5.2
9 — echo reply		răspuns ecou, § 10.2.5.2

Tabelul 10.4: Tipuri şi subtipuri mai importante de pachete ICMPv4

10.2.5.2. Diagnosticarea funcţionării rutelor

Testarea funcţionării comunicaţiei la nivel reţea este un test simplu şi extrem de util în găsirea penelor dintr-o reţea.

În acest scop, pe majoritatea sistemelor există o comandă utilizator, numită `ping`, care testează legătura dintre nodul curent şi nodul specificat.

Comanda `ping` funcţionează prin trimiterea unui pachet ICMP cu tipul *echo request* (rom. cerere ecou) către nodul specificat. Nodul destinaţie 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* şi *echo reply* se mai numesc uneori *ping* şi *pong*. Pachetele cu tipurile *ping* şi *pong* au prevăzute, conform standardului, două câmpuri, *identificare* şi *nr. secvenţă*, pe baza cărora nucleul sistemului şi comanda `ping` identifică corespondenţele între pachetele *ping* trimise şi pachetele *pong* recepţionate. Pachetele *ping* şi *pong* au prevăzut şi un câmp, de dimensiune arbitrară, de date utile; scopul acestui câmp este testarea transmiterii pachetelor mari.

Pe lângă comanda `ping` care testează funcţionarea unei legături, există o comandă, `traceroute` (pe sisteme de tip Unix) sau `tracert` (pe Windows), care afişează adresele ruterelor prin care trece un pachet pentru o anumită destinaţie.

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 viaţă (vezi § 10.2.5.3). Un astfel de pachet parcurge începutul drumului spre nodul destinaţie, însă, după parcurgerea unui număr de noduri intermediare egal cu valoarea iniţială a timpului de viaţă, provoacă trimiterea înapoi a unui pachet ICMP de tip *TTL exceeded*. Trimiţând pachete *ping* cu diferite valori pentru timpul de viaţă, se primesc pachete *TTL exceeded* de la diferitele noduri de pe traseul spre destinaţie.

O altă posibilitate de-a afla ruta spre un anumit nod este furnizată de un antet opţional, standardizat şi în IPv4 şi în IPv6, care cere ruterelor să-şi scrie fiecare adresa în acest antet opţional.

10.2.5.3. Ciclarea pachetelor IP

Este posibil să existe (temporar) inconsistenţe 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 primeşte un pachet destinat lui

C i-l va trimite lui B, B va pasa pachetul înapoi lui A, ş. a. m. d.

Pentru a preveni ciclarea nelimitată a pachetelor în astfel de cazuri, în antetul IP este prevăzut un câmp numit *timp de viaţă*. Valoarea acestui câmp este iniţializată de către nodul sursă al pachetului (valoarea iniţială este de ordinul zecilor) şi este scăzută cel puţin 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. *depăşire timp de viaţă*) pentru a semnala situaţia.

10.2.5.4. Congestia

În general, prin *congestie* se înţelege situaţia în care într-un nod intră pachete într-un ritm mai rapid decât poate nodul să retrimită pachetele, rezultând de aici o funcţionare proastă a reţelei (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. Redirecţionarea

Un nod, care primeşte un pachet şi constată că trebuie trimis mai departe în aceeaşi subreţea 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*.

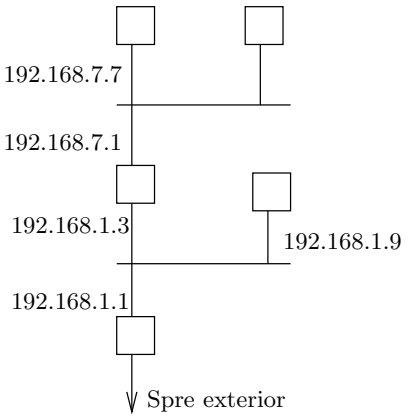


Figura 10.3: O reţea pentru ilustrarea redirecţionării pachetelor (vezi exemplul 10.8)

Exemplul 10.8: Considerăm reţeaua din figura 10.3. Pentru nodurile din

subreţeaua 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 excepţia nodului 192.168.1.1. În consecinţă, o staţie din subreţeaua 192.168.1.0/24, de exemplu 192.168.1.9, care are de trimis un pachet către un nod din subreţeaua 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 şi, totodată, trimite un pachet *ICMP redirect* către 192.168.1.9; aceasta din urmă îşi poate actualiza tabela de dirijare pentru a trimite direct la 192.168.1.3 următoarele pachete destinate nodurilor din subreţeaua 192.168.7.0/24.

10.2.6. Alte chestiuni privind dirijarea pachetelor

10.2.6.1. Dimensiunea maximă a pachetelor şi fragmentarea

Dimensiunea maximă a unui pachet IP este de 64 KiB.

Pe de altă parte, legătura directă între două noduri, dacă are noţiunea 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 octeţi.

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 acţiuni posibile:

- se face o fragmentare şi reasamblare la nivelul legăturii directe, în mod invizibil faţă de nivelul reţea;
- se face o fragmentare şi reasamblare la nivelul reţea (IP);
- se refuză livrarea pachetelor IP şi 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 priveşte 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 şi fragmentarea la nivel reţea a pachetelor.

Pentru IP versiunea 4, câmpurile necesare pentru fragmentarea şi reasamblarea pachetelor sunt prevăzute în antetul standard. De asemenea, există un flag, *nu fragmenta*, care cere ruterelor 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 opţional, deoarece este probabil să nu fie utilizate frecvent. Fragmentarea poate fi făcută doar de emiţătorul pachetului, ruterele de pe traseu fiind obligate să abandoneze transmiterea în cazul în care pachetul este prea mare.

Fragmentele sunt pachete IP obişnuite, care se transmit independent unul de altul din punctul în care s-a efectuat fragmentarea.

Nodul destinaţie efectuează reasamblarea pachetelor. În acest scop se utilizează câmpurile *identificare* şi *deplasament* şi flagul *mai urmează fragmente*. Astfel, un pachet se va asambla din fragmente având toate aceeaşi valoare în câmpurile *identificare*, *adresă sursă*, *adresă destinaţie* şi *protocol*. Pachetul asamblat va avea antetul identic cu al fragmentelor (mai puţin 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 şi ordinea lor de sosire poate fi inversată). Lungimea pachetului este determinată din faptul că, în ultimul fragment, flagul *mai urmează fragmente* are valoarea 0.

Destinaţia încearcă reasamblarea unui pachet din momentul în care a primit primul fragment al pachetului. Dacă celelalte fragmente nu sosesc într-un interval de timp suficent de scurt, nodul abandonează reasamblarea şi 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, faţă de altele care pot fi ţinute mai mult în coada de aşteptare;
- la umplerea memoriei ruterului, care pachete pot fi aruncate (distruse).

Câmpul *tip serviciu* din antetul IP conţine informaţii despre nivelul de calitate a serviciului cerut de emiţătorul pachetului; în funcţie de acesta, modulul de reţea ia deciziile privind ordinea de prioritate a pachetelor.

10.2.7. Configurarea şi testarea unei reţele IP locale

10.2.7.1. Alegerea parametrilor

În majoritatea cazurilor, într-o reţea locală, subreţelele IP, adică legăturile directe între nodurile IP, se realizează prin intermediul unor reţele din familia IEEE 802 (Ethernet sau 802.11). Primul lucru ce trebuie stabilit este alcătuirea subreţelelor.

În continuare se stabileşte, pentru fiecare subreţea IP, prefixul de reţea corespunzător. Prefixul fiecărei subreţele trebuie, pe de o parte, să permită alocarea unui număr suficient de adrese nodurilor din subreţea şi, pe de altă parte, să ducă la alocarea de adrese dintre adresele alocate de furnizorul de acces Internet sau dintre adresele rezervate pentru reţele private (vezi şi § 10.7.2 pentru alte considerente privind utilizarea adreselor private).

Pentru fiecare subreţea IP, nodurile componente trebuie să facă parte din acelaşi VLAN 802.1Q (dacă se definesc VLAN-uri) şi ca urmare trebuie să facă parte din aceeaşi reţea 802 fizică. Această cerinţă este determinată de faptul că, în cadrul unei subreţele IP, fiecare nod trebuie să poată comunica cu orice alt nod al subreţelei fără a implica dirijare la nivel IP; comunicarea trebuie să fie realizată de nivelul inferior, adică de reţeaua 802.

De notat însă că în cadrul aceleiaşi reţele IEEE 802, şi chiar în cadrul aceluiaşi VLAN 802.1Q, se pot defini mai multe subreţele IP. Astfel de subreţele lucrează independent una de cealaltă şi necesită rutere pentru a fi legate logic între ele. Pentru ca un nod să fie membru în subreţelele IP stabilite în aceeaşi reţea fizică este suficient să definească mai multe adrese IP pe aceeaşi placă de reţea (vezi § 10.5, în special § 10.5.1 pentru detalii).

Notă: independenţa subreţelelor IP de pe acelaşi VLAN este limitată de faptul că subreţelele partajează debitul maxim de transmisie şi că un intrus care ar sparge un calculator ar putea avea acces la toate subreţelelel IP stabilite pe VLAN-ul sau reţeaua fizică din care face parte calculatorul spart.

Configurarea tabelelor de dirijare trebuie să fie făcută astfel încât, pentru orice nod sursă şi pentru orice nod destinaţie, fiecare nod de pe traseul unui pachet să găsească corect următorul nod. În reţelele cu structură arborescentă (în care între oricare două subreţele există un singur drum posibil), acest lucru se realizează de regulă astfel:

- Pentru fiecare subreţea se alege, dintre nodurile ce acţionează ca rutere către alte subreţele, un *default gateway*. Acesta se alege de regulă ca

fiind nodul din subreţea cel mai apropiat de ieşirea spre restul Internetului. Se obişnuieşte ca nodul ales ca *default gateway* să primească adresa IP cea mai mică din subreţea (adică adresa în care sufixul are valoarea 1).

- Pe toate nodurile subreţelei se configurează ca *default gateway* nodul ales ca *default gateway* al subreţelei. Pentru nodurile care fac parte din mai multe subreţele, se ia *default gateway*-ul din subreţeaua cea mai apropiată de exterior (astfel un nod nu va avea ca *default gateway* pe el însuşi).
- Pe nodul ales ca *default gateway* pentru o subreţea se vor configura rutele către subreţelele „din subordine“ — subreţelele mai depărtate de legătura spre exterior decât subreţeaua considerată.

Mai notăm că într-o tabelă de dirijare statică nu se pot configura, pentru toleranţă la pene, mai multe căi spre o aceeaşi destinaţie. Dacă se doreşte aşa ceva este necesară instalarea unui program de calcul automat al tabelei de dirijare.

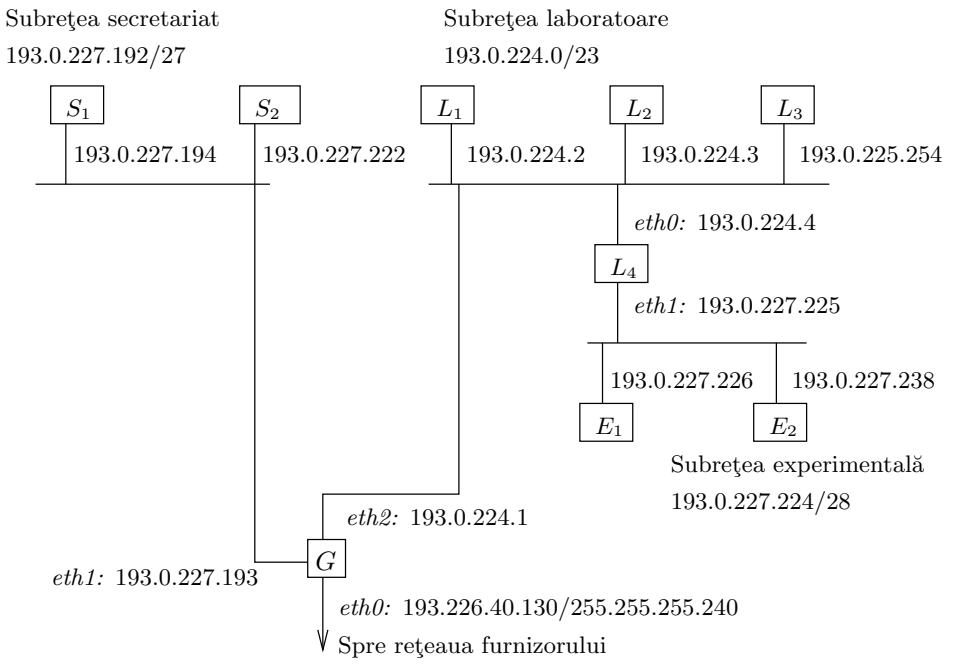


Figura 10.4: Reţea pentru exemplul 10.9

Exemplul 10.9: Să considerăm că avem de construit o reţea într-o şcoală. Presupunem că am obţinut alocarea blocului de adrese 193.0.224.0/22 pentru utilizare în reţeaua proprie şi că ruterul ce asigură legătura cu reţeaua furnizorului de acces Internet a primit adresa (în reţeaua furnizorului) 193.226.40.130 cu masca 255.255.255.240 (prefix de 28 de biţi).

Să presupunem, de asemenea, că s-a decis împărţirea reţelei interne în trei subreţele (fig. 10.4), respectiv pentru secretariat, laboratorul de informatică şi o reţea specială pentru experimente. Împărţirea luată ca exemplu este o împărţire tipică din considerente de trafic şi de securitate: reţeaua experimentală trebuie să poată fi izolată uşor de restul reţelei, iar secretariatul este separat faţă de traficul şi eventual atacurile dinspre laboratorarele de informatică.

Fiecare subreţea este construită dintr-un număr de switch-uri Ethernet, *access point*-uri 802.11 şi calculatoarele respective. Switch-urile şi *access point*-urile nu au fost figurate explicit deoarece ele nu sunt vizibile din punctul de vedere al nivelului IP.

Pentru conectarea celor trei subreţele împreună şi la Internet configurăm două rutere: G, dotat cu trei plăci de reţea, care leagă reţeaua secretariatului, reţeaua din laboratoare şi reţeaua furnizorului de acces Internet şi L_4 (probabil amplasat în laborator, pentru a fi la îndemână în timpul experimentelor), dotat cu două plăci de reţea, care leagă reţeaua experimentală de reţeaua din laborator.

Odată stabilite subreţelele, să alocăm adresele. Blocul de adrese disponibile este 193.0.224.0/22, conţinând 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 aloca 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 şi 16 adrese reţelei experimentale. Este bine să le alocăm cât mai compact, pentru ca dintre adresele nealocate să păstrăm posibilitatea de-a aloca blocuri cât mai mari. Vom aloca cele două blocuri de 32 şi 16 adrese din ultimul bloc de 64 de adrese din cele 512 libere: 193.0.227.192/27 pentru secretariat şi 193.0.227.224/28 pentru reţeaua experimentală.

Pentru fiecare din cele trei subreţele, există o alegere naturală pentru *default gateway:* G pentru reţeaua secretariatului şi pentru reţeaua din laboratoare şi, respectiv, L_4 pentru reţeaua experimentală. În fiecare caz, *default gateway*-ul este nodul cel mai apropiat de exterior. În fiecare subreţea, adresa dată ruterului cu rol de *default gateway* este cea mai mică adresă din acea subreţea.

Să vedem acum cum trebuie configurate tabelele de dirijare. Pentru staţii, tabelele sunt formate din câte două reguli: o regulă pentru livrarea directă, care asociază prefixului subreţelei unica interfaţă de reţea, şi 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 interfeţe, 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 interfeţe, o regulă implicită indicând *default gateway*-ul reţelei furnizorului de acces Internet şi o regulă pentru dirijarea spre subreţeaua „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 reţea pe diverse sisteme de operare

Pe sistemele Windows, există două posibilităţi de configurare: comanda `ipconfig` (în mod text) şi seria de casete de dialog din Start/ Control panel/ Network/ *interfaţă*. Prin ambele interfeţe se realizează atât modificarea parametrilor din modulul IP din nucleul sistemului de operare cât şi 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 puţin mai complicată, dar şi mult mai flexibilă. Comanda `ifconfig` setează parametrii legaţi de interfeţele de reţea, anume adresa IP şi masca de reţea. Tot comanda `ifconfig` introduce în tabela de dirijare regulile de livrare directă (de fapt acesta este motivul pentru care are nevoie de masca de reţea). Tabela de dirijare se afişează şi se modifică cu comenzile `route` sau `ip` (prima este mai simplă şi se găseşte pe toate sistemele, a doua este mai complexă şi serveşte şi altor scopuri). De remarcat că oprirea unei interfeţe de reţea cu `ifconfig` duce automat la eliminarea din tabela de dirijare a tuturor regulilor ce au ca ţintă

adrese accesibile prin acea interfaţă. Activarea comportamentului de ruter se face cu o comandă `sysctl`.

Comenzile `ifconfig`, `route`, `ip` şi `sysctl` au efect imediat asupra modulului IP din nucleu, dar configurările respective se pierd la repornirea sistemului. Parametrii de reţea utilizaţi la pornirea sistemului sunt luaţi de script-urile de iniţializare din nişte fişiere text; din păcate, fiecare distribuţie de Linux are propriile fişiere de configurare. De exemplu, Fedora plasează datele în fişiere în directorul `/etc/sysconfig/network-scripts`.

10.2.7.3. Testarea şi depanarea reţelelor

Cel mai util instrument de depanare pentru problemele de conectivitate este comanda `ping`. Pentru buna funcţionare a acesteia şi 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* şi *time exceeded*.

Primul lucru ce trebuie testat, în cazul unei probleme de conectivitate sau după o lucrare mai amplă la reţea, este dacă legăturile directe funcţionează. Acest lucru se face dând comanda `ping` pentru câte un vecin din fiecare subreţea din care face parte calculatorul de pe care se efectuează testul. Dacă `ping`-ul merge, înseamnă că legătura funcţionează (plăcile de reţea, cablurile, switch-urile şi *access point*-urile de pe traseu) şi că adresele IP şi măştile de reţea sunt „suficient de bune“ pentru ca nodurile să fie văzute ca făcând parte din aceeaşi subreţea. Dacă `ping`-ul nu merge şi pachetele *ping* şi *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 subreţele diferite. Dacă o legătură indirectă nu funcţionează deşi toate legăturile directe ce ar trebui să fie utilizate funcţionează, problema este de la regulile de dirijare (sau de la un filtru de pachete; de aceea este bine ca pachetele *ping* şi *pong* să nu fie filtrate). Există două lucruri uşor de scăpat din vedere aici: faptul că pentru ca `ping`-ul să meargă este necesar ca dirijarea să funcţioneze corect în ambele sensuri şi faptul că între regulile de dirijare intră inclusiv regulile de *default gateway* de pe staţii.

De exemplu, o staţie care nu are configurat *default gateway* va putea comunica cu vecinii direcţi, dar nu şi cu alte calculatoare — nici măcar cu *default gateway*-ul, dacă esre specificat prin altă adresă decât cea din aceeaşi subreţea cu staţia configurată. Alt exemplu: la reţeaua 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 subreţeaua 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 şi el regula implicită şi le trimite către 193.226.40.129 (*default gateway*-ul din reţeaua furnizorului, nefigurat pe desen). De aici pachetele se întorc înapoi spre G, deoarece furnizorul ştie că toată reţeaua 193.0.224.0/22 este în spatele lui G. Astfel, pachetele *pong* ciclează între G şi 193.226.40.129.

10.3. Nivelul transport

Aplicaţiile nu folosesc direct protocolul IP din mai multe motive:

- dacă două aplicaţii se execută pe acelaşi calculator, este necesar ca nucleul sistemului de operare să determine cărei aplicaţii îi este destinat fiecare pachet sosit;
- serviciul oferit direct de nivelul reţea (pachete ce se pot pierde, pot sosi în altă ordine şi, în anumite cazuri, pot fi duplicate) este de obicei inadecvat.

Adaptarea între nevoile aplicaţiilor şi serviciile oferite de nivelul reţea IP cade în sarcina *nivelului transport*. Nivelul transport constă dintr-o componentă a nucleului sistemului de operare, la care se adaugă uneori nişte funcţii de bibliotecă.

Componenta nivelului transport situată în nucleul din sistemului de operare furnizează aplicaţiei funcţiile sistem din familia `socket()`. Serviciile oferite aplicaţiei 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 reţelei IP şi locul modulelor TCP şi 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 octeţi, bidirecţională, 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, comunicaţie şi închidere conexiune;
- trasnportă *flux de octeţi*, adică emiţătorul trimite un şir de octeţi, negrupaţi în mesaje, de lungime arbitrară;

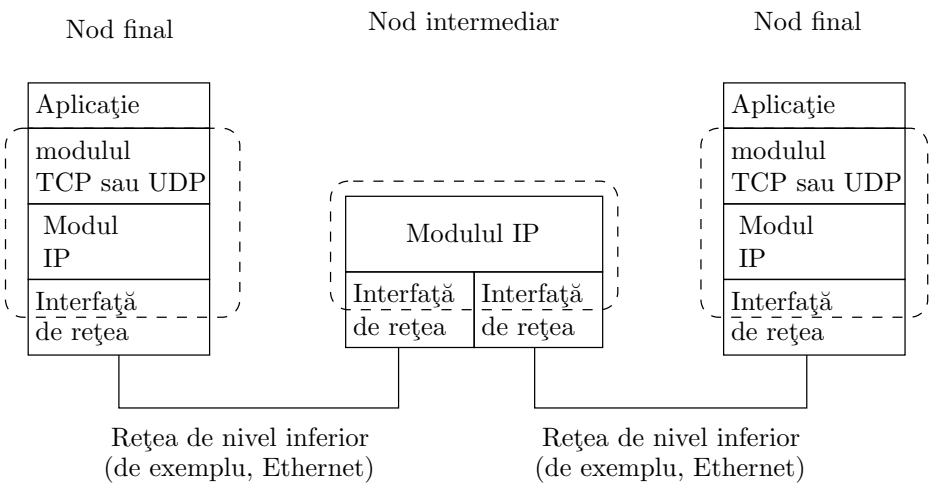


Figura 10.5: Modulele unei reţele IP. Partea inclusă în sistemul de operare este delimitată cu linie punctată.

- legătură *bidirecţională*, adică fiecare din cei doi parteneri de comunicaţie poate trimite date celuilalt;
- *livrare sigură*, adică octeţii trimişi de emiţător ajung la receptor sigur, fără duplicate şi în aceeaşi ordine în care au fost trimişi.

Modulul TCP are la dispoziţie, pentru realizarea conexiunilor TCP, facilităţile oferite de reţeaua IP.

10.3.1.1. Principiul conexiunii TCP

Livrarea sigură este obţinută pe baza unui mecanism de numerotare şi confirmare a pachetelor, aşa cum am văzut în § 4.3. Mecanismul este implementat după cum urmează. Pentru început considerăm transmisia unidirecţională şi conexiunea deja deschisă.

Zona de date utile a unui pachet IP ce transportă date pentru protocolul TCP conţine un *antet TCP* şi *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 secvenţă*; octeţi consecutivi au asociate numere de secvenţă consecutive.

Fiecare pachet TCP conţine, în zona de date utile, un şir de octeţi utili consecutivi. Câmpul *număr de secvenţă* din antetul TCP conţine numărul de secvenţă al primului octet din datele utile.

Nume câmp	Dim. (biţi)	Observaţii
Port sursă	16	
Port destinaţie	16	
Nr. secvenţă	32	
Nr. confirmare	32	
Deplasament date	4	poziţia datelor utile în pachet, dependent de dimensiunea opţiunilor
Rezervat	6	valoarea 0
Urgent	1	vezi § 10.3.1.11
Acknowledge	1	indică faptul că numărul de confirmare este valid
Push	1	arată că pachetul trebuie trimis urgent, fără a fi ţinut în diverse zone tampon
Reset	1	cere închiderea forţată a conexiunii
Synchronize	1	cere deschiderea conexiunii
Finalize	1	cere închiderea conexiunii
Dim. fereastră	16	vezi § 10.3.1.8
Suma de control	16	suma de control a antetului
Poziţie date urgente	16	vezi § 10.3.1.11
Opţiuni	variabil	

Tabelul 10.5: Antetul TCP

Modulul TCP receptor ţine evidenţa numărului de secvenţă al ultimului octet primit. La primirea unui pachet TCP, modulul receptor determină dacă datele utile sunt octeţi deja primiţi (duplicate), octeţi ce vin imediat în continuarea celor primiţi până în acel moment sau octeţi până la care există octeţi lipsă (nerecepţionaţi încă din cauza unui pachet pierdut sau întârziat). Octeţii primiţi în continuarea celor deja primiţi sunt puşi într-o coadă pentru a fi livraţi aplicaţiei 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 şi în câmpul *număr de confirmare* numărul de secvenţă al următorului octet aşteptat de la emiţător. Un pachet cu număr de confirmare n informează emiţătorul că toţi octeţii cu numere de secvenţă mai mici sau egale cu $n-1$ au fost primiţi de receptor şi nu mai trebuie retransmişi.

Emiţătorul retransmite octeţii neconfirmaţi. Datele utile, furnizate de aplicaţie emiţătorului, sunt păstrate într-o zonă tampon şi ţinute acolo până la confirmarea primirii lor de către receptor. Datele trimise şi 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 şi între prima trimitere şi momentul retrimiterii au mai sosit date de la aplicaţie, emiţătoul poate la retrimitere 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 aplicaţia sursă are de trimis şirul `abcdefghi`, şi că acesta este pasat modulului TCP emiţător în etape, întâi şirul `abcd`, apoi `efg`, `h` şi în final `i`. Mai presupunem conexiunea TCP deja deschisă şi numărul curent de secvenţă 10. Să analizăm puţin schimbul de pachete:

- Emiţătorul trimite un prim pachet, cu număr de secvenţă 10 şi date utile şirul de 4 octeţi `abcd`. Aceşti 4 octeţi au numere de secvenţă respectiv 10, 11, 12 şi 13; primul dintre acestea este scris în câmpul *număr de secvenţă* al antetului TCP.
- La primirea acestui pachet, receptorul răspunde cu un pachet de confirmare, cu numărul de confirmare 14 (acesta este următorul număr de secvenţă aşteptat).
- Emiţătorul trimite acum următorul pachet de date, cu numărul de secvenţă 14 şi date utile `efg`. Presupunem că acest pachet se pierde.
- Ca urmare a primirii de la aplicaţia sursă a următorului octet, `h`, emiţătorul TCP trimite imediat următorul pachet, cu numărul de secvenţă

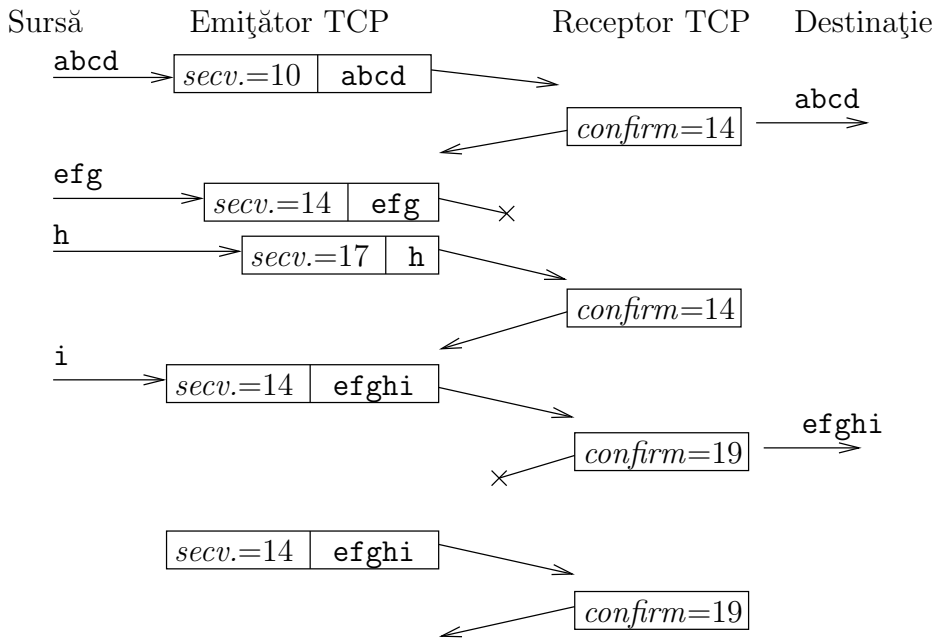


Figura 10.6: O secvenţă de pachete TCP schimbate între emiţător (stânga) şi receptor (dreapta); vezi exemplul 10.10.

17 şi date utile `h` (presupunem că emiţătorul nu utilizează algoritmul Nagle, § 10.3.1.10).

- La primirea acestui pachet (cu număr de secvenţă 17), receptorul nu îl poate confirma deoarece nu a primit numerele de secvenţă 14–16; dacă în acest moment receptorul ar trimite un pachet cu număr de confirmare 18, emiţătorul ar crede că toate numerele de secvenţă până la 17 inclusiv au fost primite şi nu ar mai retrimite niciodată numerele de secvenţă 14–16. Ca urmare, receptorul are două posibilităţi: să ignore pachetul primit (adică să nu trimită nici un pachet înapoi) sau să retrimită numărul de confirmare 14; în exemplul de faţă am considerat a doua variantă.
- Presupunem acum că pe de o parte a expirat time-out-ul emiţătorului pentru numerele de secvenţă 14–17 şi pe de altă parte că a primit de la aplicaţia sursă următorul octet (`i`). Emiţătorul împachetează acum tot ce are de (re)trimis într-un singur pachet şi trimite numărul de secvenţa 14 şi datele utile `efghi`.
- Receptorul confirmă pachetul primit, trimiţând numărul de confirmare 19. Presupunem că pachetul de confirmare respectiv se pierde.
- Emiţătorul 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. Netrimiterea în acest moment a confirmării ar duce la repetarea la infinit de către emiţător a pachetului precedent. De notat că, deşi receptorul primeşte un duplicat, emiţătorul încă nu ştie de primirea datelor de către receptor.

În continuarea schimbului de pachete ilustrat, nu se vor mai trimite pachete până ce aplicaţia sursă nu va da emiţătorului TCP noi date de transmis.

Am presupus pâna aici că numerele de secvenţă sunt numere naturale care pot creşte nedeterminat de mult. În realitate, numerele de secvenţă sunt reprezentate (vezi tabelul 10.5) pe 32 de biţi. Cum un număr de secvenţă este asociat unui octet transmis, rezultă că există numere de secvenţă distincte doar pentru 4 GiB de date; după aceea numerele de secvenţă încep să se repete. Aceasta nu era o problemă în anii ’80, deoarece la 10 Mbit/s repetarea unui număr de secvenţă apare cel mai repede după aproape o oră, timp în care este puţin probabil să mai existe în reţea un pachet vechi cu acelaşi număr de secvenţă. Într-o reţea 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 reţea timp de 34 s şi un pachet nou care poartă informaţie situată, în cadrul conexiunii, 4 GiB mai încolo, şi are acelaşi număr de secvenţă.

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 opţional cuprinzând un marcaj de timp. Metoda se aplică doar la comunicaţia 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 secvenţă mai devreme de câteva minute, oprind efectiv transmisia la nevoie.

10.3.1.2. Comunicaţia bidirecţională

Cele două sensuri de comunicaţie ale unei conexiuni TCP funcţionează (aproape) complet independent. Numerele de secvenţă utilizate pe cele două sensuri evoluează independent.

Totuşi, datele utile pentru un sens sunt plasate în acelaşi pachet TCP cu confirmarea pentru celălalt sens. Astfel, fiecare pachet TCP conţine întotdeauna date pentru un sens şi confirmare pentru celălalt sens.

Dacă este necesară transmiterea unor date, dar nu şi 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 secvenţă* se pune numărul de secvenţă al următorului octet ce va fi transmis.

10.3.1.3. Deschiderea şi închiderea conexiunii

Pentru fiecare sens de comunicaţie se consideră câte doi octeţi fictivi: un *octet de pornire* aflat înaintea primului octet util al datelor transmise şi un *octet de încheiere* aflat după ultimul octet al datelor utile. Aceşti octeţii fictivi au asociate numere de secvenţă ca şi când ar fi octeţi obişnuiţi. Ei nu sunt transmişi în zona de date utile; prezenţa lor într-un pachet este semnalizată prin setarea flagurilor *synchronize* şi respectiv *finalize* din antetul TCP.

Astfel, dacă un pachet TCP are flagul *synchronize* setat, numărul de secvenţă n şi zona de date utile conţinând k octeţi, înseamnă că pachetul transmite $k+1$ octeţi dintre care primul, cu numărul de secvenţă n, este octetul fictiv de pornire, urmat de cei k octeţi din zona de date utile, cu numere de la $n+1$ la $n+k$.

Iniţiatorul unei comunicaţii TCP trimite un pachet TCP conţinând un octet fictiv de pornire, fără a seta flagul *acknowledge* (acesta este singurul pachet ce nu are flagul *acknowledge* setat) şi punând o valoare arbitrară (care va fi ignorată la destinaţie) în câmpul *număr de confirmare*.

Numărul de secvenţă al octetului fictiv de pornire este la alegerea iniţiatorului comunicaţiei.

Receptorul pachetului iniţial răspunde, dacă doreşte să accepte conexiunea, printr-un pachet TCP care, pe de o parte, confirmă pachetul de iniţiere primit, iar pe de altă parte conţine la rândul lui un octet fictiv de pornire.

Fiecare parte consideră conexiunea deschisă în momentul în care sunt satisfăcute simultan condiţiile:

- 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 aplicaţiei până la deschiderea completă a conexiunii.

Închiderea conexiunii se face separat pe fiecare sens de comunicaţie. Marcarea închiderii unui sens se face trimiţând un octet fictiv de încheiere. După trimitera 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 secvenţă numărul imediat următor octetului fictiv de încheiere şi date utile vide. Octetul fictiv de încheiere se confirmă normal. O conexiune poate funcţiona oricât de mult timp cu un sens închis.

Nr. pachet	Sens	Nr. secv.	Nr. confirm.	Flaguri	Date utile
1	$A \to B$	123	0	syn	—
2	$A \leftarrow B$	25	124	syn,ack	—
3	$A \to B$	124	26	ack	`abc`
4	$A \leftarrow B$	26	127	ack	—
5	$A \to B$	127	26	ack,fin	`de`
6	$A \leftarrow B$	26	130	ack	—
7	$A \leftarrow B$	26	130	ack	`xyz`
8	$A \to B$	130	29	ack	—
9	$A \leftarrow B$	29	130	ack,fin	—
10	$A \to B$	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 uşor schimbul de pachete, să remarcăm că A trimite lui B un număr de 7 octeţi din care 2 fictivi, „⟨syn⟩`abcde`⟨fin⟩“, cu numerele de secvenţă de la 123 la 129 inclusiv, iar B trimite lui A un număr de 5 octeţi din care 2 fictivi, „⟨syn⟩`xyz`⟨fin⟩“, cu numerele de secvenţă de la 25 la 29 inclusiv. Fiecare pachet care transportă octeţi numerotaţi, indiferent dacă aceştia sunt date utile sau octeţi fictivi ⟨syn⟩ sau ⟨fin⟩, trebuie confirmat. Pachetele ce conţin 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 retrimită pachetul 5) şi să trimită confirmări. B consideră conexiunea complet închisă după primirea pachetului 10 (a primit un ⟨fin⟩ de la A şi a trimis şi i s-a confirmat ⟨fin⟩-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 şi 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 şi memoria asociată poate fi eliberată.

Din punctul de vedere al unui modul TCP, conexiunea este închisă în momentul în care sunt îndeplinite condiţiile:

- 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) şi retrimite octetul special de încheiere până la confirmarea acestuia. Modulul TCP local ar trebui să păstreze informaţiile despre conexiune până când modulul TCP partener primeşte 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 soluţie 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 iniţial de secvenţă

Numărul de secvenţă al octetului fictiv de pornire, numit şi *număr iniţial de secvenţă* (engl. *initial sequence number*, *ISN*), trebuie ales în aşa fel încât să nu poată exista confuzie între numere de secvenţă dintr-o conexiune veche şi cele din conexiunea curentă între aceleaşi două părţi (aceleaşi adrese IP şi numere de port).

Ideal, modulul TCP ar trebui să păstreze datele despre o conexiune atât timp cât mai pot exista în reţea pachete aparţinând conexiunii. În aceste condiţii, la redeschiderea unei conexiuni între aceleaşi două părţi, fiecare parte poate atribui octetului fictiv de pornire numărul de secvenţă imediat următor numărului de secvenţă 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 secvenţe de octeţi fictivi ⟨fin⟩⟨syn⟩.

Dacă de la precedenta conexiune a trecut destul timp pentru ca pachetele corespunzătoare să nu mai existe în reţea (fie au ajuns la destinaţie, fie au fost distruse ca urmare a depăşirii timpului de viaţă), alegerea numărului iniţial de secvenţă 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 înşela un mecanism de autentificare bazat pe adresa IP a partenerului de comunicaţie sau de-a deturna o conexiune TCP deja autentificată. În acest din urmă caz, adversarul lasă un client legitim să se conecteze la server şi, după efectuarea autentificării, adversarul injectează un mesaj destinat serverului şi având ca adresă sursă adresa clientului autentificat. Mesajul este interpretat de server ca provenind de la clientul autentificat şi, dacă conţine o comandă autorizată pentru client, comanda este executată, deşi în realitate provine de la adversar.

De multe ori, într-un atac de tip *spoofing* adversarul nu are şi posibilitatea de-a determina ruterele de pe traseu să-i permită recuperarea pachetelor de răspuns. Un atac în astfel de condiţii se numeşte *blind spoofing*.

Un atac *blind spoofing* se contracarează foarte simplu generând aleator numărul iniţial de secvenţă, adică făcând ca valoarea lui să fie imprevizibilă pentru adversar. Cum adversarul trebuie să emită pachete cu numere de secvenţă şi de confirmare valide, în cazul acestor măsuri adversarul trebuie

efectiv să ghicească numărul de secvenţă.

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 şi *acknowledge* pe 0), cu diferite valori pentru câmpurile *port sursă* şi uneori *adresă sursă* (este posibil ca adversarul să execute şi *IP spoofing*). Maşina 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, mulţumindu-se să ţină ocupată memorie pe maşina atacată; este vorba deci de un atac *denial of service*.

Contramăsura, numită *syn cookie*, se realizează astfel. O maşină care aşteaptă cereri de conectare generează aleator un şir de biţi, pe care îl ţine secret. La primirea unei cereri de conectare, maşina alege, ca număr de secvenţă iniţial (pentru sensul dinspre ea spre iniţiatorul conexiunii), valoarea unei funcţie de dispersie criptografică, aplicată asupra numărului de secvenţă al pachetului primit concatenat cu un şirul secret. Apoi, maşina ţintă a conexiunii trimite pachetul de răspuns, acceptând numărul de secvenţă propus de iniţiatorul conexiunii şi conţinând numărul de secvenţă astfel generat. Maşina ţintă nu înregistrează încă, în tabela de conexiuni, conexiunea astfel deschisă. În cazul unei conexiuni reale, la trimiterea următorului pachet de către iniţiatorul conexiunii, maşina ţintă verifică numărul de secvenţă folosind aceeaşi funcţie 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 forţată a conexiunii

Refuzul cererii de deschidere a unei conexiuni se face trimiţând iniţiatorului 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 şi 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 şi a fost repornit, pierzând evidenţa conexiunilor deschise, să informeze toţi partenerii de comunicaţie care încearcă să continue comunicaţia cu el că datele despre comunicaţie 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 forţat conexiunea. Aplicaţia ce utilizează acea conexiune este informată, printr-un cod de eroare, la următoarea operaţie privind conexiunea.

Alte situaţii care conduc la abandonarea unei conexiuni sunt:

- depăşirea unui număr de încercări de trimitere de date fără a primi confirmare;
- primirea unui pachet ICMP cu tipul *destination unreachable*.

Timpul cât emiţătorul încearcă retrimiterea 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 reţea nu duce la întreruperea conexiunilor TCP ce utilizau acel cablu. De asemenea, dacă un nod sau o legătură a reţelei IP cade, dar rămâne un drum alternativ între capetele conexiunii TCP, iar nivelul reţea reface tabelele de dirijare pentru a folosi acel drum alternativ şi acest lucru se întâmplă suficient de repede pentru ca modulul TCP să nu declare conexiunea căzută, atunci conexiunea TCP continuă să funcţioneze normal, cu pachetele IP circulând pe noua rută.

Dacă, pe o conexiune TCP deschisă, toate datele vechi au fost trimise şi confirmate, atunci nu se transmit pachete IP câtă vreme nu sunt date utile noi de transmis. Ca urmare, dacă aplicaţiile 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 situaţie, 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. Opţiunea aceasta se numeşte *keep alive* (rom. *menţine în viaţă*), deşi mai degrabă ar trebui numită *testează că mai este în viaţă*. Utilizarea opţiunii *keep alive* are şi un dezavantaj: căderea temporară a reţelei are şanse mai mari să ducă la abandonarea conexiunii.

10.3.1.6. Identificarea aplicaţiei destinaţie

Pentru a identifica aplicaţia căreia îi sunt destinate datele primite, conexiunile TCP sunt identificate printr-un cvadruplet format din adresele IP ale celor două capete şi *porturile* celor două capete. Porturile sunt numere în intervalul 1–65535 utilizate pentru a deosebi conexiunile stabilite între aceleaşi adrese IP.

Sistemul de operare ţine evidenţa conexiunilor deschise. Pentru fiecare conexiune, sistemul reţine adresa IP locală, portul local, adresa IP a celuilalt capăt şi portul de la celălalt capăt. De asemenea, sistemul reţine aplicaţia care a deschis conexiunea.

La primirea unui pachet TCP, sistemul ia adresele sursă şi destinaţie din antetul IP şi portul sursă şi destinaţie din antetul TCP. Pe baza acestora sistemul identifică conexiunea căreia îi aparţine pachetul. De aici sistemul regăseşte pe de o parte informaţiile necesare gestionării conexiunii (zone tampon, numere de secvenţă, etc) şi pe de altă parte aplicaţia căreia îi sunt destinate datele.

10.3.1.7. Corespondenţa între funcţiile `socket()` şi acţiunile modulului TCP

Funcţionalitatea modulului TCP este oferită programului utilizator prin intermediul funcţiilor sistem din familia `socket()`. Funcţiile `socket()` şi modul lor de utilizare într-o aplicaţie au fost studiate în § 8.1. Aici vom arăta legătura dintre apelarea de către o aplicaţie a funcţiilor *socket* şi acţiunile modulului TCP de expediere şi de recepţie 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()` şi `listen()` au, de asemenea, ca efect doar completarea unor informaţii în structurile de date. Dacă aplicaţia 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 şi un număr de port pentru care se aşteaptă deschiderea unei conexiuni (a fost deja apelat `listen()`), modulul TCP execută schimbul de pachete de deschidere a conexiunii. După aceea, modulul TCP aşteaptă ca aplicaţia să apeleze `accept()` pentru a-i putea semnaliza deschiderea unei noi conexiuni. În cazul în care nu se aşteaptă deschiderea unei conexiuni, modulul TCP trimite un pachet cu flagul *reset*.

Dacă, în adresa locală dată funcţiei `bind()`, s-a specificat constanta `INADDR_ANY`, este acceptat un pachet de deschidere de conexiune ce specifică ca adresă destinaţie 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 destinaţie 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*. Funcţia `connect()` aşteaptă 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, funcţia `connect()` returnează 0, în caz defavorabil semnalizează eroarea survenită.

10.3.1.8. Controlul fluxului

TCP are şi rol de control al fluxului, adică de-a încetini emiţătorul 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 octeţilor ce nu pot fi procesaţi. Metoda aceasta are însă dezavantajul că determină emiţătorul să retrimită octeţii respectivi, generând trafic inutil.

Metoda utilizată de TCP este ca receptorul să semnalizeze emiţătorului, prin valoarea câmpului *dimensiune fereastră* din antetul TCP, numărul de octeţi pe care receptorul este capabil să-i recepţioneze în acel moment. Emiţătorul nu va trimite mai mulţi octeţi decât dimensiunea ferestrei trimisă de receptor. Există o singură excepţie: dacă receptorul a semnalizat dimensiunea ferestrei 0, emiţătorul poate trimite un singur octet; aceasta se face pentru cazul în care receptorul a anunţat o fereastră de dimensiune 0 şi apoi a anunţat o fereastră mai mare dar pachetul ce anunţa mărirea ferestrei s-a pierdut.

Pe lângă mecanismul sus-menţionat, emiţătorul TCP reduce debitul de date emise şi 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 excepţia 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

Aşa cum am văzut, pachetele TCP neconfirmate într-un anumit interval de timp sunt retransmise. Valoarea aleasă a timpului după care se face retransmiterea influenţează performanţele transferului de date. O valoare prea mică duce la repetarea inutilă a unor pachete ce ajung la destinaţie însă într-un timp mai lung şi, 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 destinaţie, să fie procesat şi să se întoarcă şi să se recepţioneze confirmarea. Acest timp se numeşte *timp dus-întors* (engl. *round-trip time*, *RTT*). Timpul dus-întors nu este fix, ci depinde de perechea emiţător–receptor considerată şi de încărcarea reţelei în momentul considerat. Modulul TCP estimează statistic media şi dispersia timpului dus-întors pentru fiecare conexiune deschisă şi fixează timpul după care se retrimit pachetele neconfirmate la o valoare ceva mai mare decât media timpului dus-întors.

10.3.1.10. Algoritmul lui Nagle şi optimizarea numărului de pachete

La primirea datelor de la programul aplicaţie, prin apelul sistem `send()`, modulul TCP poate trimite imediat un pachet sau poate aştepta; aşteptarea este utilă dacă astfel se pot plasa mai multe date în acelaşi pachet IP.

Algoritmul lui Nagle prevede că, la primirea unor date de la utilizator prin `send()`:

- dacă nu sunt date trimise şi neconfirmate, sau dacă datele noi sunt suficient de mari pentru a umple un pachet, datele se trimit imediat;
- altfel, modulul TCP aşteaptă până la primirea confirmării sau expirarea timpului de retransmitere, şi abia atunci trimite datele primite între timp de la aplicaţie.

O altă optimizare constă în a întârzia câteva fracţiuni de secundă trimiterea confirmării pentru un pachet TCP primit. Ideea este că, dacă aplicaţia care recepţionează datele din acel pachet are de trimis date ca răspuns la datele primite, datele ce constituie răspunsul să fie trimise în acelaşi 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 ataşate conexiunii, unul pentru datele „obişnuite“ şi celălalt pentru date „speciale“. Datele speciale poartă denumirea, în terminologia angloamericană, *out of band data* (*OOB*).

O posibilă utilizare este următoarea: presupunem o aplicaţie care transferă fişiere. Presupunem că emiţătorul trimite mai întâi lungimea fişierului şi apoi conţinutul. Dacă utilizatorul lansează trimiterea unui fişier mare (să zicem câţiva gigaocteţi) şi apoi se răzgândeşte şi doreşte să abandoneze operaţia, partea de emiţător a aplicaţiei nu poate semnaliza receptorului în nici un fel abandonarea transmiterii. Aceasta deoarece octeţii transmişi sunt interpretaţi de receptor ca fiind conţinutul fişierului. Aici intervin datele cu marcajul *special* (*out of band*): emiţătorul trimite conţinutul fişierului 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 mari repede cu putinţă aplicaţiei destinaţie. 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 *poziţie date urgente* conţine dimensiunea datelor speciale rămase de transmis (în pachetul curent şi în următoarele).

De principiu un pachet conţinând date speciale va avea setat şi flagul *push*, care indică faptul că modulul TCP receptor ar trebui să livreze datele către aplicaţia destinaţie cât mai repede.

10.3.2. Datagrame nesigure: UDP

Există aplicaţii care se descurcă acceptabil cu datagrame nesigure. Pentru acestea, nivelul transport trebuie să rezolve doar problema găsirii aplicaţiei căreia îi este destinată datagrama. Această problemă este rezolvată de protocolul UDP.

Fiecărei aplicaţii ce utilizează UDP i se acordă, de către modulul UDP al sistemului de operare local, un *port UDP*. Un port UDP alocat unei aplicaţii nu va fi acordat şi altei aplicaţii decât după ce este eliberat de prima.

O datagramă UDP poartă ca adrese sursă şi destinaţie perechi formate din câte o adresă IP şi un număr de port. Adresa IP este adresa nodului pe care rulează aplicaţia sursă, respectiv destinaţie, iar portul este portul alocat de sistem aplicaţiei.

O datagramă UDP este construită dintr-un pachet IP cu identificatorul protocolului UDP în *protocol* şi cu zona de date utile conţinând un *antet UDP* şi datele datagramei UDP. Antetul UDP conţine portul sursă şi portul destinaţie. Adresele IP sursă şi destinaţie 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 datagrame UDP este limitată de dimensiunea maximă a pachetului IP.

Programul utilizator cere trimiterea unei datagrame UDP prin intermediul apelului `sendto()` sau `sendmsg()`. Programul trebuie să specifice adresa destinaţie — adresa IP şi 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, recepţia datagramei trimise nu este confirmată în nici un fel, funcţiile `sendto()` sau `sendmsg()` nu au cum să returneze programului apelant informaţii despre livrarea pachetului. Dealtfel, ambele funcţii se termină (returnează controlul apelantului) înainte ca pachetul să fie efectiv livrat.

Funcţiile sistem `recvfrom()` şi `recvmsg()` aşteaptă recepţia unei datagrame având ca adresă destinaţie adresa asociată socket-ului dat ca argu-

ment. Funcţia returnează datele utile din datagramă precum şi 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 ţinut minte de către oameni;
- adresele IP sunt legate de topologia reţelei şi se schimbă odată cu aceasta; spre exemplu, dacă o firmă schimbă furnizorul de Internet folosit, adresele staţiilor 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 reţea printr-un nume uşor de memorat de către om şi care să fie independent de topologia reţelei.

DNS este construit ca o bază de date ce cuprinde înregistrări ce asociază unui nume o adresă IP. Această bază de date este împărţită între mai multe *servere de nume*, care pot fi interogate de oricine. O aplicaţie care doreşte să comunice şi să folosească nume în loc de adrese IP interoghează întâi nişte servere de nume, până ce află adresa IP corespunzătoare numelui dat, după care lucrează cu adresa astfel obţinută.

Baza de date DNS este stocată distribuit (pe mai multe servere, pentru a nu avea un server supraaglomerat) şi redundant (există mai multe servere capabile să răspundă la o cerere dată).

DNS este proiectat în ideea că informaţiile se citesc frecvent şi se modifică rar. Cu ocazia modificărilor se admite să existe temporar incoerenţe — un utilizator să obţină informaţia nouă în timp ce altul deţine informaţia veche.

10.4.1. Numele de domeniu

Numele de domeniu [RFC 1034, 1987] sunt numele ce pot fi date nodurilor şi altor obiecte, în cadrul DNS. Un nume de domeniu este compus dintr-un şir de componente. Fiecare componentă este un şir de caractere; RFC 1034 nu impune restricţii, însă recomandă ca fiecare componentă să fie formată din cel mult 63 de caractere, ce pot fi litere, cifre sau caracterul *minus* (`-`), cu restricţia ca primul caracter să fie literă şi 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, şi 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 Babeş-Bolyai Cluj-Napoca,
- `cs` — Departamentul de Informatică (din engl. *Computer Science*),
- `nessie` — numele staţiei.

Această scriere este inspirată din scrierea adreselor poştale, care încep cu numele destinatarului şi se termină cu ţara.

În majoritatea cazurilor, aplicaţiile acceptă specificarea numelor de domenii fără punctul final. În lipsa punctului final, interpretarea este însă diferită. Anume, dacă numele nu este terminat cu punct, aplicaţia va încerca să adauge la nume şiruri 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 semnificaţia numelui dat de utilizator.

Exemplul 10.13: Presupunem că lista de căutare configurată de administrator pentru un sistem conţine, în ordine, `scs.ubbcluj.ro`, `cs.ubbcluj.ro` şi `ubbcluj.ro`. O căutare pentru numele `www` va duce la căutarea numelui `www.scs.ubbcluj.ro.` care va fi găsit şi vor fi returnate informaţiile 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.` şi `www.cs.ubbcluj.ro.`; acesta din urmă este găsit şi căutarea este încheiată.

Structurarea numelui în mai multe componente serveşte la administrarea ierarhică a spaţiului de nume. O organizaţie care dobândeşte un nume de domeniu poate crea şi administra după voie numele formate prin adăugare de componente ierarhic inferioare. De exemplu, Universitatea Babeş-Bolyai din Cluj-Napoca a obţinut numele `ubbcluj.ro.` . Crearea numelui `nessie.cs.ubbcluj.ro.` este decizia exclusivă a Universităţii Babeş-Bolyai.

O instituţie care doreşte un nume de domeniu trebuie să contacteze instituţia care administrează domeniul părinte şi să ceară alocarea unui nume.

Alocarea numelui se plăteşte — fie o taxă plătită o singură dată, fie o taxă anuală. Instituţia ce a obţinut numele este responsabilă de întreţinerea unui server de nume (vezi § 10.4.3 şi § 10.4.4) pentru domeniul ce i-a fost alocat.

Adesea firmele care oferă acces Internet oferă gratuit clienţilor 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 maşinilor 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-şi schimbe numele de domeniu ale serverelor sale, ceea ce poate duce la pierderea clienţilor ce nu află noul nume al site-ului firmei.

De remarcat că, deşi organizarea ierarhică a numelor de domeniu seamănă cu organizarea numelor de fişiere şi directoare, nu există o restricţie similară cu aceea că într-un director nu e permis să aibă acelaşi nume un fişier şi un subdirector. Anume, ca exemplu, numele `ubbcluj.ro.` şi `cs.ubbcluj.ro.` pot desemna simultan noduri în reţea.

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* şi *termenul de valabilitate*. Tipul şi clasa se utilizează pentru a putea pune în DNS şi alte informaţii în afară de adrese IP. Înregistrările corespunzătoare adreselor IP au tipul *A* (de la *address*=adresă) şi clasa *IN* (de la *Internet*). Câmpul *valoare* a unei înregistrări cu tipul *A* şi clasa *IN* conţine adresa IP a nodului cu numele de domeniu dat.

DNS permite căutarea unei înregistrări pentru care s-au specificat numele de domeniu, clasa şi tipul.

Este permis să existe mai multe înregistrări pentru acelaşi nume, clasă şi tip, dacă au valori diferite. Cineva care obţine prin interogarea DNS mai multe adrese IP pentru un nume dat poate folosi oricare din adrese; acestea se presupune că sunt ale aceluiaşi 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* şi valoarea *nume2* defineşte *nume1* ca pseudonim pentru *nume2*. În acest caz, *nume2* este considerat numele canonic al acelui obiect. Dacă pentru un nume există o înregistrare *CNAME*, nu este permis să mai existe vreo altă înregistrare pentru acel nume.

Tip	Valoare	Observaţii
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 informaţiilor 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 şi prioritate	serverele de poştă electronică pentru domeniul solicitat, § 11.1

Tabelul 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 aplicaţie care caută o înregistrare de un anumit tip pentru un nume trebuie să caute şi o înregistrare *CNAME* pentru acel nume. Dacă găseşte o înregistrare *CNAME*, trebuie să caute o înregistrare cu numele canonic găsit şi având tipul căutat iniţial. Valoarea astfel găsită trebuie utilizată ca şi când ar fi fost găsită pentru numele original.

10.4.3. Împărţirea în domenii de autoritate

Mulţimea numelor de domeniu este împărţită în *zone*. Pentru fiecare zonă există unul sau mai multe *servere de nume* sau *server DNS* care deţin toate înregistrările corespunzătoare numelor din acea zonă.

Privim spaţiul de nume ca un arbore în care rădăcina este domeniul rădăcină şi fiecare nume are ca părinte numele obţinut din el prin înlăturarea celei mai din stânga componente. O zonă este o submulţime 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 deţină toate înregistrările corespunzătoare numelor din zonă. Faptul că un nume care ar

aparţine zonei nu figurează în tabela ţinută de acel server înseamnă că numele respectiv nu există.

Din motive de toleranţă la pene, pentru fiecare zonă există de regulă cel puţin două servere responsabile. De principiu, tabelele despre o zonă dată ale serverelor responsabile de acea zonă trebuie să fie identice; pot exista însă temporar incoerenţe între tabele cu ocazia modificărilor unor informaţii.

Pe lângă înregistrările despre zonele pentru care este responsabil, un server de nume trebuie să mai deţină înregistrări care să permită regăsirea serverelor de nume ale zonelor adiacente — zona imediat superioară ierarhic şi zonele imediat subordonate, dacă există — şi 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 conţine nişte date administrative despre zonă (vezi § 10.4.5);
- Una sau mai multe înregistrări *NS* (*Name Server*) care conţin 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 ţinute de un server cu privire la zonele vecine.

Un server poate ţine şi alte înregistrări, în afară de înregistrările din zonele pentru care este responsabil şi 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 aşteaptă cereri prin datagrame UDP trimise pe portul 53 şi prin conexiuni TCP pe portul 53. Clientul trimite cererea întâi ca datagramă UDP. Dacă răspunsul este prea lung pentru a încape într-o datagramă UDP atunci clientul repune întrebarea printr-o conexiune TCP.

Interogarea cuprinde numele de domeniu căutat, tipul şi clasa. Răspunsul conţine interogarea şi un şir de înregistrări, împărţite în trei categorii: înregistrări din zonele pentru care este responsabil, înregistrări de legătură şi 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 însuşi) 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 reţină înregistrările astfel obţinute, 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 ţinută un timp cel mult egal cu termenul de valabilitate al înregistrării, după care se consideră că informaţia s-ar fi putut modifica şi 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 funcţii de bibliotecă, cum ar fi `gethostbyname()`. Aceste funcţii 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 fişieru `/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 întreţine uşor 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 informaţiile de pe *master*.

De notat că, într-o astfel de configuraţie, atât serverul *master* cât şi 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 oridecâteori 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 aşteptare după o încercare eşuată de-a contacta serverul *master* înainte de-a încerca din nou;
- *expire* este timpul după care, în cazul în care nu a reuşit să contacteze serverul *master*, serverul *slave* trebuie să nu se mai considere responsabil de zonă.

Există şi 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 deţine 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, şi 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* şi având ca valoare numele de domeniu al nodului respectiv. Interogarea şi administrarea acestor nume de domeniu se fac întocmai ca şi pentru numele obişnuite.

De principiu, un subdomeniu din `in-addr.arpa.` corespunde unui bloc de adrese IP alocat unei instituţii; subdomeniul corespunzător din domeniul `in-addr.arpa.` este administrat de aceeaşi instituţie.

În situaţia în care alocarea blocurilor de adrese IP se face după schema CIDR, graniţele blocurilor nu coincid cu graniţele 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 conţine şi alte adrese decâte 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 nişte nume din domenii create special pentru blocurile alocate. Pentru exemplul dat, construcţia 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 foloseşte un mecanism asemănător, definit în [RFC 3596, 2003]. Anume, fiecărei adrese IPv6 i se asociază un nume în domeniul `ip6.arpa` . Numele se construieşte astfel:

- Se iau grupuri de câte 4 biţi din adresa IPv6 şi se scrie cifra hexa corespunzătoare ca o componentă separată.
- Ordinea ierarhică a componentelor astfel obţinute este aceea în care componentele corespunzătoare biţilor mai semnificativi din adresa IP sunt superioare ierarhic componentelor corespunzătoare biţilor mai puţin 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 reţele Internet este o legătură directă, este de fapt o legătură între două noduri într-o reţea de alt tip, de exemplu o reţea IEEE 802. O astfel de reţea joacă rolul nivelului legăturii de date în contextul protocolului Internet şi ca urmare o vom numi aici *reţea de nivel inferior*.

Transmiterea unui pachet IP de la un nod A către un nod B, în cadrul aceleiaşi reţele de nivel inferior, se face astfel:

- mai întâi, nodul A determină adresa, în cadrul reţelei 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 reţelei 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 asemnea 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 vedera testării, a unei cereri ARP către o staţie se poate face cu ajutorul comenzii `arping`.

10.6. Configurarea automată a staţiilor — DHCP

Sunt situaţii în care este util ca un nod să-şi determine propria adresă IP, împreună cu alţi câţiva parametri (masca de reţea, *default gateway*, servere DNS) prin interogări în reţea, în loc ca aceşti parametri să fie stocaţi într-o memorie nevolatilă (disc sau memorie flash) a nodului. Situaţii 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 reţea în alta, unde parametrii trebuie configuraţi de fiecare dată în funcţie de reţeaua la care este conectat;
- într-o reţea mare în care este de dorit ca parametrii de reţea ai staţiilor să poată fi schimbaţi uşor de pe un calculator central.

Există trei protocoale ce au fost utilizate de-a lungul timpului pentru determinarea parametrilor de reţea: RARP, BOOTP şi 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 doreşte să-şi determine parametrii de reţea (adresa IP proprie, masca de reţea, *default gateway*-uri, numele propriu, adresele serverelor DNS) se numeşte *client DHCP*. Clientul DHCP trimite o cerere, la care răspunde un *server DHCP* stabilit pentru reţeaua respectivă. Răspunsul conţine parametrii solicitaţi. Vom studia în continuare:

- cum se transmit cererea şi răspunsul DHCP, în condiţiile în care modulul IP al clientului nu este configurat şi ca urmare nu este complet funcţional;
- cum determină serverul parametri clientului.

Transmiterea cererii şi a răspunsului DHCP. Presupunem în continuare că nodul client este conectat la o singură reţea de tip IEEE 802.

Cererea DHCP este transmisă ca un pachet UDP. Adresa IP destinaţie a pachetului este adresa de broadcast locală (255.255.255.255), iar portul destinaţie 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) şi purtând ca adresă sursă adresa plăcii de reţea locale. Serverul DHCP trebuie să fie în aceeaşi subreţea cu clientul

(sau să existe în aceeaşi subreţea un server *proxy DHCP* care să preia cererea clientului şi s-o retrimită serverului).

Răspunsul DHCP este plasat la rândul lui într-un pachet UDP purtând ca adresă sursă adresa serverului DHCP şi ca adresă destinaţie adresa alocată clientului DHCP, cu portul destinaţie 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 eşua 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 subreţea pentru care acţionează, un server DHCP trebuie să aibă două categorii de date: parametrii comuni tuturor nodurilor din subreţea (masca de reţea, gateway-uri, servere DNS) şi parametrii specifici fiecărui nod în parte (adresa IP a nodului).

Parametrii comuni sunt setaţi de administratorul serverului DHCP într-un fişier de configurare al serverului.

Pentru adresele IP ale nodurilor din reţea 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 fişier de configurare al serverului toate adresele MAC ale plăcilor de reţea şi adresele IP corespunzătoare. În al doilea caz, pe serverul DHCP se scriu numele staţiilor şi adresele corespunzătoare iar pe fiecare staţie se setează numele staţiei (a doua soluţie nu este aplicabilă pe calculatoare fără harddisc).

Alocare dinamică: Serverul dispune de o mulţime de adrese pe care le alocă nodurilor. Serverul păstrează corespondenţa dintre adresele MAC ale clienţilor şi 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 (faţă de atribuirea pe durată nedeterminată):

- la alocarea dinamică a adreselor, dacă clientul este scos din reţea 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 clienţii vor primi noile adrese cu ocazia încercării de prelungire a atribuirii adresei.

10.7. Situaţii speciale în dirijarea pachetelor

Vom studia în paragraful de faţă anumite procedee mai deosebite utilizate în dirijarerea pachetelor. Aceste procedee se aplică îndeosebi în reţelele interne ale unor instituţii.

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 funcţionare ale unui nod IP ci, în funcţie de anumite reguli, ignoră complet sau trimite pachete ICMP de eroare pentru anumite pachete.

Scopul unui filtru de pachete este de-a interzice anumite acţiuni în reţea, 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 posibilităţile oferite de mecanismul *iptables* din sistemul Linux, cu menţiunea că facilităţile de bază se regăsesc în toate sistemele.

O *regulă de filtrare* este o pereche formată dintr-o *condiţie* şi o *acţiune*. Regulile sunt grupate în şiruri numite *lanţuri*. Există trei lanţuri predefinite:

- *INPUT* aplicat pachetelor destinate nodului curent,
- *OUTPUT* aplicat pachetelor generate de nodul curent,
- *FORWARD* aplicat pachetelor generate de alt nod şi având ca destinaţie alt nod (pentru care nodul curent acţionează ca ruter).

Pentru fiecare pachet ajuns la modulul IP, acesta aplică prima regulă, din lanţul corespunzător traseului pachetului, pentru care pachetul îndeplineşte condiţia specificată în regulă. Aplicarea regulii înseamnă executarea acţiunii specificate de regulă.

Principalele *acţiuni* ce pot fi specificate sunt:

- *ACCEPT* — pachetul este livrat normal,
- *DROP* — pachetul este ignorat (ca şi când nu ar fi fost primit),

- *REJECT* — se trimite înapoi un pachet semnalând o eroare — implicit *ICMP destination unreachable*.

Condiţiile specificate într-o regulă pot privi:

- interfaţa prin care a intrat pachetul (cu excepţia lanţului *OUTPUT*),
- interfaţă prin care ar ieşi pachetul (cu excepţia lanţului *INPUT*),
- adresa IP sursă şi adresa IP destinaţie (se poate specifica şi un prefix de reţea, condiţia 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 destinaţie (pentru pachete ce intră, respectiv ies, prin interfeţe ce au conceptul de adresă MAC),
- protocolul (TCP, UDP, ICMP),
- portul sursă sau destinaţie (pentru protocoale care au noţiunea de port),
- tipul şi subtipul ICMP (pentru pachete ICMP),
- flag-uri ale diverselor protocoale,
- dimensiunea pachetului,
- starea conexiunii TCP căreia îi aparţine pachetul (vezi mai jos).

Un nod (intermediar) prin care trec toate pachetele asociate unei conexiuni TCP poate, examinând antetul TCP al fiecărui pachet, să ţină evidenţă stării conexiunii. Ca urmare, nodul poate stabili dacă un pachet deschide o conexiune nouă, aparţine 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 reţea. Ca urmare, modulele de reţea nu ar trebui să interpreteze protocolul TCP (antetele TCP ar trebui considerate pur şi simplu date utile). Ca urmare, nodurile intermediare, din care nu acţionează asupra pachetelor în tranzit decât modulul reţea şi modulele inferioare, nu ar trebui să „înţeleagă“ protocolul TCP.

Regulile de filtrare se configurează de către administratorul sistemului. Ca şi în cazul parametrilor IP:

- Pe sistemele Linux, regulile aplicate de nucleul sistemului de operare se examinează şi se modifică cu ajutorul unei comenzi — `iptables`. Regulile valabile la iniţializarea sistemului sunt configurate de *script*-urile invocate la pornire, fiind încărcate dintr-un fişier text.

- Pe sistemele Windows există o interfaţă grafică cu care se configurează simultan regulile curente aplicate de nucleu şi în acelaşi 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 interfaţă către Internet şi au ca adresă sursă o adresă din reţeaua internă,
  - pachete ce au ca sursă o adresă de broadcast (clasa D) sau de clasă E,
  - pachete ce intră dinspre o anumită subreţea au ca sursă o adresă ce nu face parte din subreţeaua respectivă şi nici din alte subreţele din direcţia respectivă.

- Să fie interzise conexiunile din afara reţelei locale către servicii care sunt oferite doar pentru reţeaua locală. De exemplu, accesul la un *share* Windows este adesea de dorit să nu fie posibil din afara reţelei locale.

  Pentru aceasta există două strategii:
  - se blochează pachetele destinate porturilor pe care aşteaptă conexiuni serviciile respective;
  - se permit conexiunile către serviciile ce se doresc a fi accesibile din afară (web, mail, eventual ssh), se permit conexiunile iniţiate din interior şi 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 iniţiate din interior. Aceasta se întâmplă deoarece o conexiune iniţiată din interior are alocat un port local cu număr imprevizibil; ca urmare nu se poate stabili o regulă simplă pentru permiterea intrării pachetelor destinate acelui port.

  Soluţia uzuală este:
  - pentru conexiun TCP, se permit pachetele asociate unei conexiuni deja deschise, se permit pachetele către exterior, se permit pachetele destinate serviciilor publice şi se interzic toate celelalte pachete.

- pentru UDP, unde nu se poate ţine evidenţa unor conexiuni, se interzic pachetele destinate unor servicii private, se permit pachetele spre exterior, se permit pachetele provenite de la serviciile ce se doreşte a fi accesate în exterior (serverele DNS sau NTP utilizate) şi se interzic toate celelalte pachete.

- Să fie interzise diferite alte pachete „dubioase“, cum ar fi:
  - pachete destinate adresei de broadcast a reţelei locale sau adresei de broadcast generale (255.255.255.255),
  - pachete având ca adresă sursă sau destinaţie adresa maşinii locale (127.0.0.1) sau o adresă privată.

Exemplul 10.15: Fie un ruter având în spate o reţea internă având adrese cu prefixul 193.226.40.128/28. Ruterul are interfaţa *eth0* cu adresa 193.0.225.20 către exterior şi interfaţa *eth1* cu adresa 193.226.40.129 către subreţeaua locală.

Din reţeaua 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* şi *https* de pe 193.226.40.130 şi către serverele *ssh* de pe toate maşinile din reţeaua locală.

Din reţeaua locală dorim să putem accesa servicii DNS şi NTP. Acestea le furnizăm astfel:

- pe ruter instalăm un server DNS şi un server NTP, accesibile din reţeaua locală; acestea furnizează serviciile respective pentru reţeaua locală
- permitem cererile emise de serverele DNS şi NTP de pe ruter, precum şi răspunsurile corespunzătoare. Cererile NTP provin de pe portul UDP 123 al ruterului şi 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 şi sunt adresate portului DNS 53 de pe un nod extern.

Pentru diagnosticarea funcţionării reţelei vom mai permite trecerea pachetelor ICMP *ping*, *pong*, *destination unreachable* şi *time exceeded*.

Traficul în interiorul reţelei locale îl permitem fără restricţii.

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. Reţele private

O *reţea privată* este o reţea, 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 instituţie A are o reţea proprie de calculatoare. Din această reţea proprie, o parte dintre calculatoare — să le numim *publice* — trebuie să comunice nerestricţionat cu alte calculatoare din Internet, în vreme ce restul calculatoarelor — le vom numi *private* — este acceptabil să poată comunica doar cu alte calculatoare din reţeaua internă.

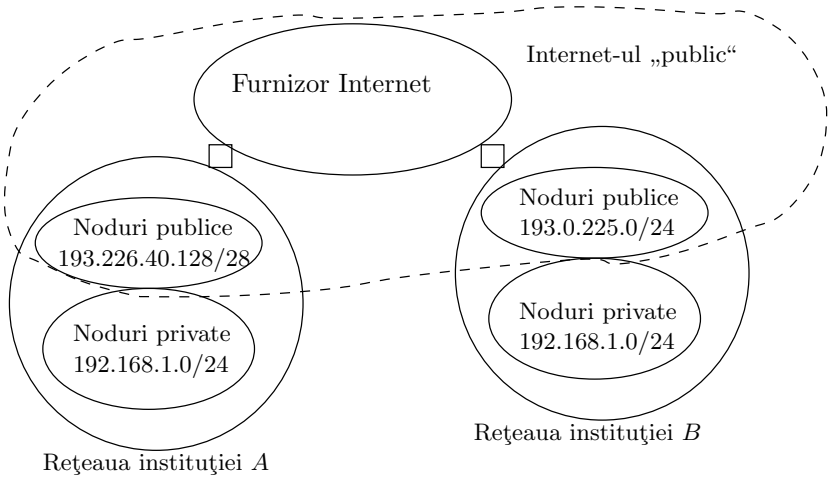


Figura 10.7: Două reţele locale având fiecare aceleaşi adrese private pentru o parte din calculatoare.

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 reţele interne, de exemplu de cele ale instituţiei B din figură. De fapt, în general, o adresă trebuie să fie unică doar în mulţimea nodurilor cu care un anumit nod ar putea dori să comunice.

În situaţia descrisă, este necesar ca adresele din Internet-ul „public“ să fie unice, adresele din reţeaua locală să fie unice şi să nu existe suprapuneri între adresele din Internet şi adresele din reţeaua locală. Cerinţa din urmă este determinată de cerinţa ca nodurile cu adrese publice din reţeaua proprie să poată comunica şi cu nodurile private şi cu nodurile din Internet.

Un pachet a cărui adresă destinaţie este o adresă privată este dirijat de către rutere astfel:

- dacă ruterul face parte dintr-o reţea locală în care există acea adresă,

pachetul este dirijat către unicul nod din reţeaua locală purtând adresa respectivă;

- altfel, ruter-ul declară pachetul nelivrabil.

Aşa cum am văzut în § 10.2.4.1, următoarele blocuri de adrese IP sunt alocate pentru astfel de utilizări în reţele private: 10.0.0.0/8, 172.16.0.0/12 şi 192.168.0.0/16.

De cele mai multe ori, calculatoarelor private li se oferă posibilităţi limitate de-a comunica cu calculatoare din Internet, prin intermediul unor mecanisme descrise în paragrafele următoare.

10.7.3. Translaţia adreselor (NAT)

Translaţia adreselor este un mecanism prin care un ruter modifică adresa sursă sau adresa destinaţie a unor pachete.

10.7.3.1. Translaţia adresei sursă

Presupunem că avem o reţea privată şi 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 doreşte să deschidă o conexiune către un calculator S din Internet trimite un pachet IP având ca adresă sursă adresa proprie (privată) C, ca adresă destinaţie adresa serverului S (adresă care este publică) şi conţinând o cerere de deschidere de conexiune TCP. Pachetul ajunge la destinaţie, iar serverul S vom presupune că acceptă conexiunea. Serverul S trimite înapoi un pachet IP având ca adresă sursă adresa proprie S şi ca adresă destinaţie adresa, privată, a clientului C. Deoarece adresa clientului nu este unică la nivelul Internet-ului (ci doar la nivelul propriei reţele interne), pachetul de răspuns nu poate fi livrat.

Translaţia adresei sursă rezolvă problema de mai sus în modul următor: În primul rând, trebuie să existe un nod (ruter) G în reţeaua internă, având adresă publică şi 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 primeşte pachetul ca provenind de la G şi 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 destinaţie a pachetului, punând adresa lui C în locul propriei adrese,

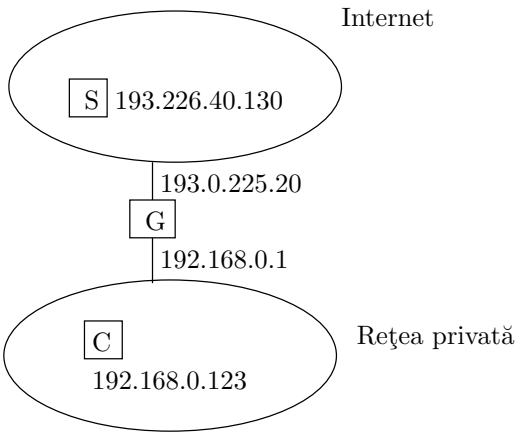


Figura 10.8: Reţea privată pentru exemplificarea mecanismului de translaţie a adresei sursă

după care trimite mai departe pachetul către C — acest lucru este acum posibil deoarece G este în reţeaua internă, şi ca urmare adresa lui C indică singurul nod din reţeaua 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 destinaţie 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 urmăreşte pachetele de deschidere a conexiunii. La primirea unui pachet de deschiderea 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 preferinţă $p_g = p_c$, însă dacă p_c nu este liber se poate aloca un p_g diferit). Pachetul de deschidere a conexiunii este modificat de G astfel încât adresa sursă să fie G şi portul sursă să fie p_g. G păstrează asocierea (C, p_c, p_g). La sosirea unui pachet cu adresa destinaţie G şi portul destinaţie p_g, ruterul G pune adresa destinaţie C şi portul destinaţie p_c; la primirea unui pachet cu adresa sursă C şi portul sursă p_c ruterul G pune adresa sursă G şi 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 reţeaua 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 şi G		Între G şi S	
	sursă	destinaţie	sursă	destinaţie
$C \to S$	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 şi destinaţie ale pachetelor schimbate între clientul C şi serverul S în exemplul 10.16

portul 80 (pentru a aduce o pagină web). Ruterul G având adresa publică 193.0.225.20 efectuează translaţia adresei sursă. Adresele pachetelor transmise între C şi S sunt date în tabelul 10.8.

Pentru alte protocoale, asocierea este mai dificil de făcut. Pentru UDP, există noţiunea de port şi ca urmare identificarea pachetelor primite de G pe baza portului destinaţie p_g este posibilă. Este însă dificil de determinat durata valabilităţii asocierii (C, p_c, p_g), întrucât 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* şi *pong* există un număr de identificare care poate fi folosit cu acelaşi rol ca şi un număr de port. Translaţia adreselor pentru astfel de pachete funcţionează întocmai ca şi în cazul UDP.

Mecanismul de translaţie 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.

Faţă de utilizarea adreselor publice, utilizarea adreselor private şi a translaţiei 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, informaţii privind adresa IP şi portul de pe client, vor fi constatate incoerenţe între IP-ul şi portul clientului văzute de către server (acestea fiind G şi respectiv p_g) şi IP-ul şi portul clientului văzute de client (acestea fiind C şi respectiv p_c).

Cea de-a doua limitare poate fi eliminată dacă G cunoaşte protocolul de nivel aplicaţie dintre C şi S şi modifică datele despre conexiune schimbate

între C şi S. Prima limitare poate fi eliminată în măsura în care este vorba de conexiuni iniţiate în urma unor negocieri pe o conexiune anterioară (de exemplu, conexiunile de date din protocolul FTP); pentru aceasta, G trebuie, din nou, să urmărească comunicaţia dintre C şi S şi să modifice datele privitoare la adresa şi portul pe care clientul aşteaptă conexiune dinspre server.

10.7.3.2. Translaţia adresei destinaţie

Presupunem că avem o reţea privată, un server S cu adresă privată, un ruter G situat în reţeaua proprie dar având adresă publică şi un client C din Internet. Clientul C doreşte 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 destinaţie, 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.

Translaţia adresei destinaţie poate fi utilă în următoarele situaţii:

- dacă avem o singură adresă publică şi dorim să avem mai multe servere accesibile din exterior, servere ce nu pot funcţiona pe aceeaşi maşină. Putem furniza astfel un server *HTTP* şi un server *SMTP* care apar din Internet ca fiind la aceeaşi adresă IP dar sunt găzduite în realitate pe calculatoare distincte. Necesitatea utilizărea calculatoarelor distincte pentru aceste servicii poate rezulta din motive de securitate sau din motive legate de performanţele calculatoarelor utilizate.
- dacă dorim totuşi acces din afară către calculatoarele din reţeaua 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 redirecţionată 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, ..., S_n în reţeaua internă. Conexiunile deschise din Internet către G vor fi redirecţionate echilibrat către serverele S_1, S_2, ... S_n. Mai exact, la deschiderea unei conexiuni către G, G alege un server S_k către care redirecţionează acea conexiune. Orice pachet ulterior de pe acea conexiune va fi redirecţionat

către S_k. Conexiuni noi pot fi redirecţionte către alte servere.

10.7.4. Tunelarea

Prin *tunelare* se înţelege, în general, transmiterea unor date aparţinând unui anumit protocol ca date utile în cadrul unui protocol de acelaşi 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 reţea sau de nivel aplicaţie.

O situaţie în care este necesară tunelarea este aceea în care există două reţele private şi se doreşte ca nodurile din cele două reţele să poată comunica nerestricţionat între ele. De exemplu, avem o instituţie care are două sedii şi are o reţea privată in fiecare sediu. Există mai multe soluţii pentru a realiza legătura:

- *Translaţia adreselor* realizează o legătură supusă unor restricţii, studiate în § 10.7.3.
- *Unificarea fizică* a celor două reţele private, ducând o legătură fizică între ele (fig. 10.9), oferă conectivitate completă, însă ducerea unui cablu specual pentru aceasta poate fi extrem de costisitor.

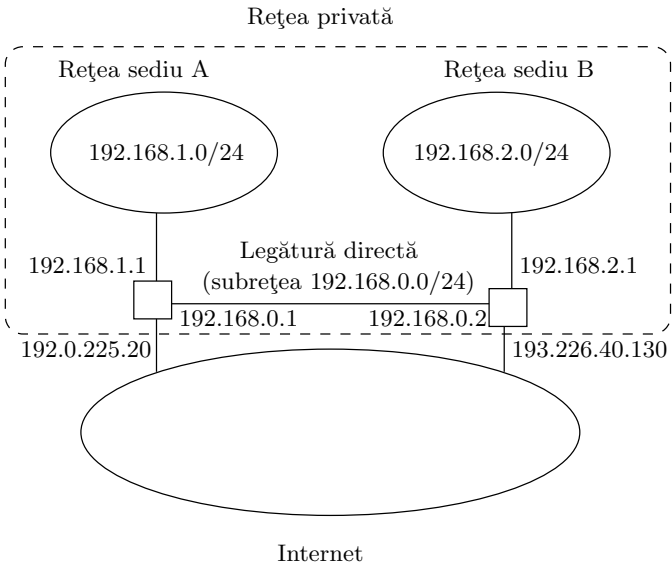


Figura 10.9: Unificarea reţelelor private printr-o legătură fizică directă.

- *Tunelarea* oferă conectivitate completa ca şi legătura fizică folosind legăturile la Internet existente. Construcţia constă în realizarea unei conexiuni (de exemplu TCP) între două rutere cu adrese publice din cele două reţele interne (fig. 10.10). Conexiunea dintre rutere este un „cablu virtual“ ce preia rolul conexiunii fizice.

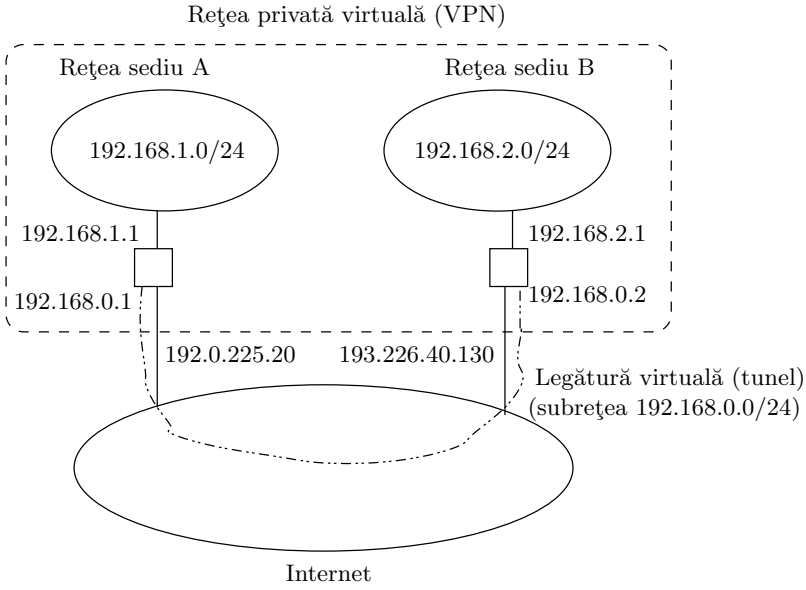


Figura 10.10: Unificarea reţelelor private printr-un tunel

Tunelul se prezintă faţă de nivelul reţea ca şi când ar fi o legătură fizică. Ca urmare, fiecare capăt al tunelului este o interfaţă de reţea, având o adresă IP şi o mască de reţea.

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 subreţea cu mai multe interfeţe conectate la ea — de exemplu o reţea Ethernet.

Capitolul 11

Aplicaţii în reţele

11.1. Poşta electronică

Poşta electronică serveşte la transferul de mesaje electronice între utilizatori aflaţi pe sisteme diferite. Protocolul a fost creat iniţial pentru mesaje text şi a fost modificat ulterior pentru a permite transferul de fişiere cu conţinut arbitrar. Sistemul este conceput în ideea că este acceptabil ca transferul mesajului să dureze câteva ore, pentru a putea funcţiona pe sisteme ce nu dispun de o legătură permanentă în reţea.

Poşta electronică este una dintre primele aplicaţii ale reţelelor de calculatoare şi a fost dezvoltată în aceeaşi 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 doreşte expedierea mesajului. Acesta interacţionează cu utilizatorul pentru a-l asista în compunerea mesajului şi stabilirea adresei destinatarului. La final, *mail user agent*-ul trimite mesajul unui proces de tip *mail transfer agent* (*MTA*). Transferul este iniţiat de *MUA*-ul utilizatorului expeditor şi utilizează protocolul SMTP (§ 11.1.2.1).
- O serie de procese de tip *mail transfer agent* care trimit fiecare următorului mesajul. Transferul este iniţiat de *MTA*-ul emiţător şi utilizează tot protocolul SMTP.
- De fiecare adresă destinaţie este răspunzător un *mail transfer agent* care memorează mesajele destinate acelei adrese. Odată mesajul ajuns la

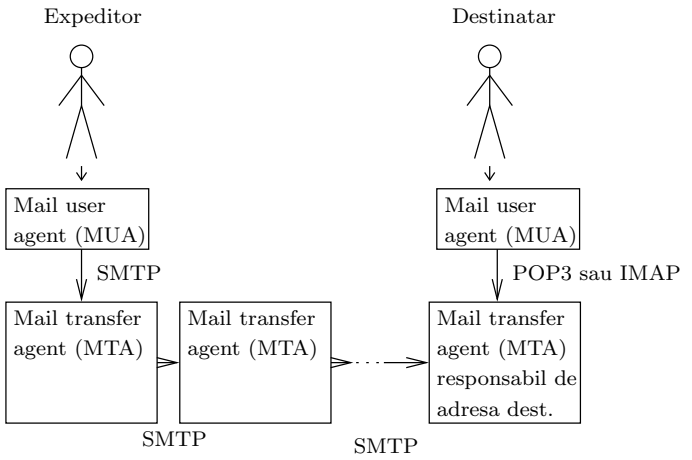


Figura 11.1: Elementele sistemului de transmitere a poştei electronice. Săgeţile arată sensul în care se iniţiază comunicaţia (de la client spre server), nu sensul în care se transferă mesajul de poştă electronică.

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. 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 biţi (ci doar caractere de 7 biţi), este recomandabil ca mesajele să nu conţină caractere cu codul între 128 şi 255.

Datorită unor protocoale folosite pentru transmiterea şi pentru stocarea mesajelor, se impun următoarele restricţii suplimentare asupra conţinutului 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 menţionat că în protocolul de comunicaţie dintre două *mail transfer agent*-uri sunt transferate informaţii privind adresa expeditorului şi adresa destinatarului, independente de cele plasate în antetul mesajului. Aceste informaţii formează aşa-numitul *plic* (engl. *envelope*) al mesajului. Expeditorul şi destinatarul date în antetul mesajului sunt informaţii pentru utilizatorul uman; informaţiile 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* şi o *valoare*. De principiu, fiecare câmp este un rând separat conţinând numele, caracterul *două puncte* (`:`) şi valoarea; secvenţa *carriage return* urmat de *line feed* acţionează ca separator între câmpuri. Antentul se termină cu două secvenţe *carriage return – line feed* consecutive.

Dacă un câmp este prea lung pentru a încape într-un rând (standardul recomandă ca rândurile să nu depăşească 78 de caractere şi interzice rândurile mai lungi de 998 caractere), câmpul poate fi continuat pe rândurile următoare, care trebuie să înceapă cu spaţiu sau *tab*; spaţiile şi *tab*-urile de la începutul rândurilor următoare se consideră ca făcând parte din câmp.

Exemplul 11.1: Un posibil document (vezi mai jos explicaţiile privind semnificaţiile 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` şi `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 corespondenţa oficială între un angajat al unei firme şi 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 şefului (care trebuie informat cu privire la comunicaţie).

  Adresele din câmpul `Bcc` reprezintă pesoane cărora le va fi livrat mesajul, fără ca ceilalţi destinatari să fie informaţi despre aceasta. Câmpul `Bcc` este completat de către *mail user agent* şi este eliminat de către primul *mail transfer agent* de pe traseu.

- `From`, `Sender` şi `Reply-to` reprezintă adresa expeditorului şi adresa la care trebuie răspuns.

  În condiţii obişnuite, un mesaj conţine doar câmpul `From`, reprezentând adresa expeditorului mesajului şi 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 organizaţii pe care o reprezintă. Există două situaţii practice care conduc la această situaţie: Dacă mesajul provine de la şef, dar este scris şi trimis efectiv de secretara şefului, atunci adresa şefului este pusă în câmpul `From`, iar adresa secretarei în câmpul `Sender`. Dacă o persoană trimite un mesaj în numele unei organizaţii, atunci adresa organizaţiei va fi trecută în câmpul `From` şi 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 funcţionalitatea de *reply* a *mail user agent*-ului său, *MUA*-ul oferă implicit, ca adresă destinaţie 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 prezenţa 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 discuţii are ca `From` adresa autorului mesajului şi ca `To` adresa listei. Mesajul se transmite centrului de

distribuţie al listei (un *mail transfer agent*), care retransmite mesajul către abonaţii listei. Mesajul retransmis către abonaţi va avea adăugat un câmp `Reply-to` indicând adresa listei. Astfel, mesajul primit de abonat are `From` autorul mesajului, `To` adresa listei şi `Reply-to` tot adresa listei. Abonatul listei va răspunde foarte uşor pe adresa listei deoarece *mail user agent*-ul său va prelua adresa listei din `Reply-to` şi o va pune ca adresă destinaţie în mesajul de răspuns. Totuşi, e bine ca utilizatorul să nu folosească orbeşte această facilitate, întrucât uneori răspunsul este bine să ajungă doar la autorul mesajului original, nu la toată lista...

- `Received` şi `Return-path` au ca rol diagnosticarea sistemului de livrare a mesajelor. Fiecare *mail transfer agent* de pe traseul mesajului adaugă în faţa mesajului un câmp `Received` în care scrie numele maşinii sale, numele şi adresa IP a *mail transfer agent*-ului care i-a trimis mesajul, data şi ora curentă şi emiţătorul şi destinatarul mesajului conform cererii *mail transfer agent*-ului care îi transmite mesajul (cei indicaţi pe „plicul“ mesajului, nu cei din câmpurile `To` sau `From`). Astfel, un mesaj începe cu un şir de antete `Received` conţinând, în ordine inversă, nodurile prin care a trecut mesajul.

  Ultimul *mail transfer agent* adaugă un câmp `Return-path`, conţinând adresa sursă a mesajului conform *plicului* (datele furnizate de MTA-ul sursă).

  Câmpurile `Received` şi `Return-path` sunt utilizate pentru a returna un mesaj la autorul său în cazul în care apar probleme la livrarea mesajului. De notat diferenţa între `Reply-to`, care reprezintă destinatarul unui eventual răspuns dat de utilizator la mesaj, şi `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 şi ora locală a expeditorului precum şi indicativul fusului orar pe care se găseşte 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, opţional încă un caracter *două puncte* urmat de secundă şi în final indicativul fusului orar. Indicativul fusului orar cuprinde diferenţa, în ore şi minute, între ora locală şi 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 şir 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 şi de la nişte numere aleatoare, astfel încât să fie extrem de improbabil ca două mesaje să aibă acelaşi `Message-ID`.

  În cazul răspunsului la un mesaj prin funcţia *reply* a *mail user agent*-ului, mesajul de răspuns conţine un câmp `In-reply-to` având ca valoare `Message-ID`-ul mesajului la care se răspunde. Valoarea câmpului `Reference` se crează din câmpurile `Reference` şi `Message-ID` ale mesajului la care se răspunde. Câmpurile `Reference` şi `Message-ID` pot fi folosite de exemplu pentru afişarea, de către *mail user agent*-ul destinaţie, a succesiunilor de mesaje legate de o anumită problemă şi date fiecare ca replică la precedentul.
- `Resent-from`, `Resent-sender`, `Resent-to`, `Resent-cc`, `Resent-bcc`, `Resent-date` şi `Resent-msg-id` sunt utilizate dacă destinatarul unui mesaj doreşte să retrimită mesajul, fără modificări, către altcineva. O astfel de retrimitere se poate efectua printr-o comandă *bounce* sau *resend* a *MUA*-ului. În acest caz, câmpurile din antetul mesajului original (inclusiv `From`, `To` sau `Date`) sunt păstrate, reflectând expeditorul, destinatarii şi data trimiterii mesajului original. Pentru informaţiile privind retransmiterea (adresa utilizatorului ce efectuează retransmiterea, destinatarii mesajului retransmis, data retransmiterii, etc.), se utilizează câmpurile `Resent-`... enumerate mai sus. Semnificaţia fiecăruia dintre aceste câmpuri `Resent-`... este identică cu semnificaţia câmpului fără `Resent-` corespunzător, dar se referă la retransmitere, nu la mesajul original.

11.1.1.2. Extensii MIME

Standardul original pentru mesaje de poştă electronică (rfc 822) a suferit o serie de extensii. Acestea sunt cunoscute sub numele *Multipurpose Internet Mail Extension* (*MIME*) şi sunt descrise în [RFC 2045, 1996], [RFC 2046, 1996] şi [RFC 2047, 1996].

Extensiile MIME servesc în principal pentru a putea transmite fişiere ataşate unui mesaj de poştă 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. Ataşarea fişierelor şi mesaje din mai multe părţi

Standardul *MIME* oferă posibilitatea ataşării unor fişiere la un mesaj de poştă electronică. Mecanismul se încadrează într-unul mai general, care permite formarea unui mesaj din mai multe părţi. Un mesaj cu ataşamente este dat în exemplul 11.2.

Fiecare mesaj are în antet un câmp, `Content-type`, care arată ce tip de date conţine şi 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 faţa 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 conţin doar text obişnuit trebuie să aibă `Content-type: text/plain`. Acesta este dealtfel tipul implicit în cazul absenţei câmpului `Content-type`.

Mesajele cu ataşamente au `Content-type: multipart/mixed`. În general, un mesaj de tip `multipart/`*subtip* este format de fapt din mai multe „fişiere“ (oarecum ca un fişier arhivă *zip*). Într-un mesaj `multipart/mixed`, de obicei una dintre părţi este mesajul propriu-zis, iar fiecare dintre celelalte părţi este câte un fişier ataşat.

Fiecare parte a unui mesaj `multipart` are un antet şi un corp, similar cu un mesaj de sine stătător. Antetul părţii poate conţine 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 şi simplu mai multe componente puse împreună, ca un fişier arhivă.
- `multipart/alternative` arată că părţile sunt variante echivalente ale aceluiaşi document, în formate diferite.

Separarea părţilor unui mesaj de tip `multipart` se face printr-un rând ce conţine un anumit text, ce nu apare în nici una dintre părţile mesajului. Textul utilizat ca separator este plasat în câmpul `Content-type` după `multipart/`*subtip*. Este scris sub forma (utilizabilă şi în alte câmpuri şi pen-

tru alte informaţii) unui şir `boundary=`*şir* situat după şirul `multipart/`*subtip* şi separat prin punct şi virgulă faţă de aceasta. Exemplu:

```
Content-type: multipart/mixed; boundary="abcdxxxx"
```

Corpul mesajului `multipart` este separat după cum urmează:

- în faţa primei părţi precum şi între fiecare două părţi consecutive se găseşte un rând format doar din textul de după `boundary=` precedat de două caractere minus (`--`);
- după ultima parte se găseşte 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 destinaţie cu partea de mesaj în care se găseşte acest antet. Valori posibile sunt:

- `inline` arată că mesajul sau partea de mesaj trebuie să fie afişată utilizatorului;
- `attachment` arată că mesajul sau partea de mesaj nu trebuie afişată decât la cerere. Un astfel de câmp poate avea în continuare o informaţie suplimentară, `filename=`*nume*, arată numele sugerat pentru salvarea părţii respective. De notat că, din motive de securitate, *mail user agent*-ul destinaţie trebuie să nu salveze orbeşte partea de mesaj sub numele extras din mesaj, ci să ceară mai întâi permisiunea utilizatorului. În caz contrar, un adversar poate să trimită un fişier având ataşat un anumit fişier, cu care să suprascrie un fişier al destinatarului.

11.1.1.4. Codificarea corpului mesajului şi a ataşamentelor

Standardul original al formatului mesajelor prevede că mesajele conţin doar text ASCII, cu utilizare restricţionată a caracterelor de control. Singurele caractere de control (cele cu codurile între 0 şi 31) admise sunt *carriage return* (cod 13) şi *line feed* (cod 10), utilizate pentru separarea rândurilor din mesaj. De asemenea, se recomandă să nu se utilizeze caracterele cu coduri între 128 şi 255, datorită imposibilităţii transmisiei lor în unele sisteme.

Ca urmare, transmiterea unui fişier arbitrar (inclusiv a unui text ISO-8859) nu este posibilă direct.

Transmiterea unui conţinut arbitrar se face printr-o recodificare a acestuia utilizând doar caracterele permise în corpul mesajului. Ca urmare,

mesajele apar, faţă 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 afişa într-un mod mai puţin inteligibil pentru utilizator.

Recodificarea este aplicată doar asupra corpului mesajului, nu şi asupra antetului. Antetul conţine un câmp, `Content-transfer-encoding`, a cărui valoare arată dacă şi ce recodificare s-a aplicat asupra conţinutului. Codificările definite de [RFC 2045, 1996] sunt:

- `7bit` — înseamnă de fapt lipsa oricărei recodificări. În plus, arată că mesajul nu conţine decât caractere ASCII (cu codurile cuprinse între 0 şi 127).
- `8bit` — arată că mesajul nu a fost recodificat, dar poate conţine orice caractere (cu coduri între 0 şi 255).
- `quoted-printables` — arată că fiecare caracter de control şi fiecare caracter egal (`=`) a fost recodificat ca o secvenţă 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 şi 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 ataşamente are corpul mesajului codificat `7bit`, partea corespunzătoare mesajului propriu-zis este codificată `7bit` sau `quoted-printables`, iar părţile corespunzătoare ataşamentelor sunt codificate `base64`.

Exemplul 11.2: Un mesaj cu fişiere ataşate 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 fisiere 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

Aşa cum am văzut, mesajele sunt transmise din aproape în aproape, fiecare mesaj parcurgând un lanţ de *MTA*-uri. Fiecare *MTA*, cu excepţia ultimului, determină următorul *MTA* şi-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 aparţine *MTA*-ului ce primeşte mesajul. Clientul deschide o conexiune TCP către portul 25 al serverului.

După deschiderea conexiunii, serverul trimite un mesaj conţinând 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 şi se încheie printr-o secvenţă *carriage return – line feed*. Răspunsurile sunt formate dintr-un număr, transmis ca secvenţă de cifre, urmat de un text explicativ. Numărul

arată, într-o formă uşor de procesat de către calculator, dacă cererea s-a executat cu succes sau nu şi cauza erorii. Textul cuprinde informaţii suplimentare pentru diagnosticarea manuală a transmiterii mesajelor.

Clientul trebuie să înceapă printr-o cerere `HELO` care are ca parametru numele maşinii clientului. Prin aceasta clientul se identifică faţă 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 poştă 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 destinaţie. Fiecare cerere `RCPT TO:` poate fi acceptată sau refuzată de către server independent de celelalte. Serverul va transmite mesajul fiecăreia dintre adresele destinaţie 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 găseşte doar caracterul punct. După primirea corpului mesajului, serverul trimite încă un răspuns.

Mesajele primite de MTA sunt plasate într-o coadă de aşteptare, stocată de obicei în fişiere 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 şi MTA-ul reîncearcă periodic să-l trimită. După un număr de încercări eşuate sau în cazul unei erori netemporare (de exemplu, dacă nu există adresa destinaţie), MTA-ul abandonează şi încearcă trimiterea unui mesaj (e-mail) de eroare înapoi către expeditor.

Dacă adresa destinaţie este locală, MTA-ul plasează mesajul în fişierul corespunzător cutiei poştale a destinatarului.

De notat că în trimiterea mai departe, livrarea locală sau trimiterea unui mesaj de eroare, MTA-ul utilizează doar informaţiile de pe *plic*, adică parametrii comenzilor `MAIL FROM:` şi `RCPT TO:`, şi 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 aşa 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 şi 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 informaţiile sale de configurare (vezi § 11.1.2.3), dacă are vreo regulă privind adresa destinaţie. Dacă MTA-ul este responsabil de adresa destinaţie a mesajului, îl memorează local. Dacă MTA-ul este poartă de intrare a mesajelor pentru MTA-urile din reţeaua locală, transmite mesajul către MTA-ul determinat conform configurării.
2. Dacă nu există informaţii de configurare pentru adresa destinaţie, MTA-ul caută în DNS o înregistrare cu tipul MX pentru numele de domeniu din adresa destinaţie. O astfel de înregistrare conţine una sau mai multe nume de servere SMTP capabile să preia mesajele destinate unei adrese din acel domeniu. Dacă găseşte o astfel de înregistrare, MTA-ul contactează una din maşinile specificate în înregistrările MX găsite şi-i transmite mesajul.
3. Dacă nu există nici înregistrări MX, MTA-ul contactează maşina cu numele de domeniu din adresa destinaţie şi-i transmite mesajul.
4. Dacă nu există nici un server cu numele de domeniu din adresa destinaţie, adică dacă toate cele trei variante de mai sus eşuează, atunci MTA-ul declară că mesajul este nelivrabil şi transmite înapoi spre emiţător 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 facilităţi de editare şi să prezinte utilizatorului o interfaţă prietenoasă, iar MTA-ul are toate complicaţiile legate de livrarea mesajelor. Pentru transmiterea oricărui mesaj, un MUA contactează un acelaşi MTA, a cărui adresă este configurată în opţiunile MUA-ului. Pe sistemele de tip UNIX, MUA-urile contactează implicit MTA-ul de pe maşina locală (`localhost`).

11.1.2.3. Configurarea unui MTA

De cele mai multe ori, un MTA este resposnabil 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 întreagă 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 regulare, 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 poştă, situate în domeniul numelui maşinii 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 (configuraţi în fişierul `/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 maşinii MTA-ului. De exemplu, se poate configura MTA-ul de pe `nessie.cs.ubbcluj.ro` să accepte mesajele destinate lui `ion@example.com` şi 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 maşina `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 poşta electronică a unor clienţi care au nume propriu de domeniu dar nu deţin servere de poştă electronică care să preia poşta adresată adreselor din domeniul respectiv.
- Dacă utilizatorii expediază mesaje de pe mai multe maşini dintr-o reţea locală, nu este de dorit ca numele maşinii interne să apară în adresa de poştă electronică. De exemplu, dacă un acelaşi utilizator are cont pe mai multe maşini, nu este de dorit ca adresa expeditorului să depindă de maşina de pe care utilizatorul scrie efectiv mesajul. De exemplu, nu este de dorit ca dacă scrie un mesaj pe maşina `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 reţeaua internă va schimba, atât în antetul mesajului (valoarea câmpului `From`) cât şi pe „plic“ (valoarea parametrului comenzii SMTP `MAIL FROM:`), adresa expeditorului, eliminând numele maşinii locale şi 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` şi 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 priveşte 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 reţeaua locală şi mesajele destinate adreselor locale, dar să nu accepte trimiterea mai departe a mesajelor provenite din exterior şi 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-şi identitatea.

11.1.3. Securitatea poştei 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 şi fără a fi legate de informaţii pe care destinatarii le-ar dori;
- viruşii — programe executabile sau documente, ataşate unui mesaj electronic, a căror execuţie sau respectiv vizualizare duce la trimiterea de mesaje electronice către alţi destinatari.

Falsificarea adresei sursă este extrem de simplă deoarece, în transmiterea obişnuită 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 şi verificând dacă există neconcordanţe între numele declarat prin `HELO` a unui client SMTP şi adresa sa IP sau neconcordanţe între numele primului MTA şi partea de domeniu din adresa expeditorului. De notat că anumite neconcordanţe pot fi legitime, în cazul în care căsuţele poştale dintr-un domeniu sunt ţinute 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] şi 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 poştă electronică trebuie să asigure livrarea mesajelor nesemnate şi în consecinţă cu risc de a fi falsificate.

Uşurinţa falsificării adresei sursă şi uşurinţa 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 speranţa de-a găsi printre aceştia unii care să se lase păcăliţi.

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 şters din greşeală.

Există detectoare automate de spam-uri, bazate pe diferite metode din domeniul inteligenţei artificiale. Astfel de detectoare se instalează pe MTA-uri şi resping sau marchează prin antete speciale mesajele detectate ca spam-uri. Un MTA care detectează şi respinge sau marchează spam-urile se numeşte *filtru anti-spam*.

De menţionat filtrele anti-spam nu pot fi făcute 100% sigure deoarece nu există un criteriu clar de diferenţiere. Ca urmare, orice filtru anti-spam va lăsa să treacă un anumit număr de spam-uri şi există şi riscul de-a respinge mesaje legitime.

Majoritatea furnizorilor de servicii Internet nu permit, prin contract, clienţilor să trimită spam-uri şi depun eforturi pentru depistarea şi penalizarea autorilor. În acest scop, ei primesc sesizări şi întreţin 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 poştă electronică. Acest lucru se realizează cel mai uşor prin căutarea, în paginile web, a tot ceea ce arată a adresă de poştă electronică. Contramăsura la această recoltare este scrierea adreselor, din paginile web, doar în forme dificil de procesat automat — de exemplu, ca imagine (într-un fişier *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 execuţie duce la inserarea unor cópii ale sale în alte programe de pe calculatorul pe care se execută virusul. Impropriu, prin *virus* se mai desemnează un fragment, inserat într-un program util, care execută acţiuni 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-şi insereze

cópii în alte programe).

În contextul poştei electronice, un *virus* este un fragment dintr-un program plasat ca fişier ataşat la un mesaj electronic. Virusul se poate reproduce fie prin mijloace independente de poşta electronică, fie prin expedierea, către alţi utilizatori, a unor cópii ale mesajului. În acest al doilea caz, virusul utilizează, de obicei, adrese extrase din lista, ţinută de MUA, a adreselor partenerilor de corespondenţă ai utilizatorului care primeşte mesajul virusat.

Indiferent de forma de propagare (infectarea fişierelor locale sau transmiterea de mesaje spre alţi utilizatori), pentru a-şi realiza scopul, un virus trebuie să ajungă să determine *execuţia, cu drepturile utilizatorului victimă, a unei secvenţe de instrucţiuni aleasă de autorul virusului.* Acest lucru se poate întâmpla în două moduri:

- Virusul se găseşte într-un program executabil, pe care utilizatorul îl execută.
- Virusul este un document astfel construit încât, exploatând o eroare din programul utilizat pentru vizualizarea documentului, să determine programul de vizualizare să execute acţiunea dorită de autorul virusului.

Pentru a păcăli destinatarul şi a-l determina să execute sau să vizualizeze fişierul ataşt, corpul mesajului este construit astfel încât să câştige încrederea utilizatorului. Astfel, mesajul este adesea construit ca şi când ar proveni de la administratorul de sistem sau de la un prieten al destinatarului.

Metodele de prevenire a viruşilor de poştă electronică sunt aceleaşi ca şi metodele de prevenire a viruşilor 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, execuţia 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 instrucţiunile potenţial nocive. Această din urmă abordare este utilizată de *applet*-urile Java.

Pe lângă aceste metode de prevenire, există câteva acţiuni care micşorează riscul sau consecinţele execuţiei unui virus. Una dintre ele este reducerea la minimul necesar a lucrului din cont de administrator. Altă măsură preventivă este aceea de a nu vizualiza sau executa fişierele ataşate unui mesaj suspect; pentru aceasta, este bine ca expeditorul unui mesaj să nu trimită niciodată un mesaj numai cu fişiere ataşate, ci să scrie un mic text explicativ, care să-i permită destinatarului să identifice autorul şi scopul fişierelor ataşate.

11.2. Sesiuni interactive la distanţă

O sesiune interactivă *locală* a unui utilizator (vezi fig. 11.2) presupune că tastatura, ecranul şi eventual alte dispozitive cu ajutorul cărora utilizatorul interacţionează cu sistemul de calcul (mouse, difuzoare, etc.) sunt conectate direct la calculatorul pe care se desfăşoară sesiunea utilizatorului. Conectarea este realizată printr-o interfaţă hardware de conectare a dispozitivelor periferice (RS232, PS/2, VGA, USB, DVI), care permite legături pe distanţe 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).

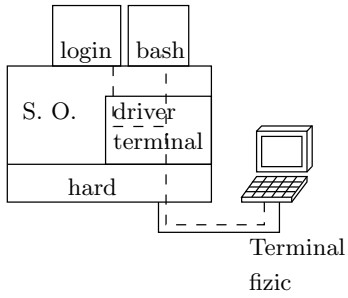


Figura 11.2: Componentele implicate într-o sesiune interactivă locală

În general ne gândim că pe un astfel de sistem lucrează un singur utilizator la un moment dat. Totuşi, există şi posibilitatea de-a conecta mai multe ansambluri tastatură–ecran la un acelaşi calculator, în felul acesta lucrând simultan mai mulţi utilizatori. Acest mecanism s-a utilizat masiv în anii 1970, sistemele fiind numite *cu time-sharig*. 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 soluţii ad-hoc (deturnarea întreruperilor în DOS, sistemul Windows până la versiunile 3.x), sisteme multitasking fără protecţie între utilizatori (Windows 9x şi ME) şi în final sisteme multitasking propriu-zise (Windows NT/2000/XP şi sistemele de tip UNIX — Linux şi porturile FreeBSD, Solaris, etc).

Linux (prin mecanismul consolelor virtuale) şi Windows XP (prin mecanismul *switch user*) permit deschiderea simultană a mai multor sesiuni locale de la acelaşi ansamblu tastatură–ecran, pentru acelaşi utilizator sau pentru utilizatori distincţi. O singură sesiune poate fi activă la un moment dat, celelalte fiind „îngheţate“. Sistemul permite comutarea între sesiuni.

În cazul unei *sesiuni la distanţă*, în locul unui terminal, conectat printr-o interfaţă specializată la calculatorul pe care se desfăşoară sesiunea, se utilizează un calculator, conectat prin reţea la calculatorul pe care se desfăşoară sesiunea. În felul acesta, un utilizator aflat în faţa unui calculator conectat la Internet poate deschide o sesiune la distanţă către orice alt calculator din Internet (bineînţeles, cu condiţia să aibă un cont acolo). Principial, numărul de sesiuni deschise simultan către un calculator este limitat doar de resursele calculatorului (memorie şi viteză de procesare).

Deschiderea unei sesiuni prin mecanismul de sesiune la distanţă se poate face şi către calculatorul local. Acest mecanism poate fi utilizat pentru deschiderea unei sesiuni ca alt utilizator, fără a închide prima sesiune.

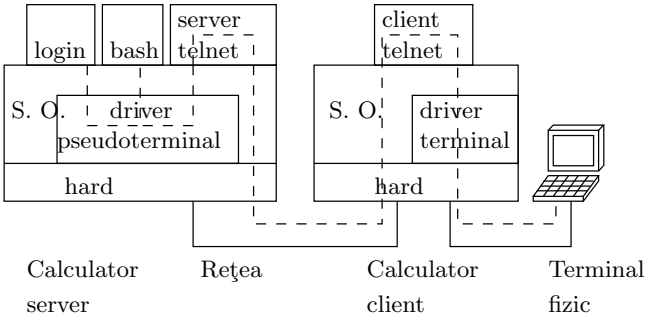


Figura 11.3: Componentele implicate într-o sesiune interactivă la distanţă.

Sistemul pentru deschiderea sesiunilor la distanţă (vezi fig. 11.3) constă din două componente majore:

- Pe sistemul la care este conectat fizic utilizatorul rulează o aplicaţie care trimite prin reţea ceea ce utilizatorul introduce de la tastatură şi afişează pe ecran ceea ce trimite sistemul de la distanţă. Afişarea se poate face pe tot ecranul sau într-o fereastră. Această aplicaţie deschide în mod activ conexiunea la deschiderea sesiunii, motiv pentru care este un *client*.
- Pe sistemul de la distanţă, pe care are loc sesiunea, rulează o aplicaţie care primeşte prin reţea datele trimise de aplicaţia client şi le livrează proceselor ce rulează în cadrul sesiunii. De asemenea, preia datele de ieşire ale acestor procese — datele care în cazul unei sesiuni locale s-ar afişa pe ecran — şi le trimite prin reţea clientului. Această aplicaţie este lansată la pornirea sistemului şi aşteaptă conexiuni — fiind în acest sens un server. La conectarea unui client, aplicaţia 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 execuţie 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 comunicaţie între procese. Mecanismul de comunicaţie trebuie să apară faţă de procesele din sesiunea utilizatorului ca şi când ar fi tastatura şi 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 şi 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 primeşte 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, permiţând tunelarea protejată criptografic a conexiunilor TCP pentru alte aplicaţii.

Protocolul *ssh* este construit pe mai multe nivele. Nivelul cel mai de jos [RFC 4253, 2006] realizează protejarea criptografică a conexiunii şi 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 aplicaţia propriu-zisă şi permite sesiuni de lucru, interactive sau nu, în mod text, către sisteme de tip UNIX, tunelarea unor conexiuni TCP arbitrare, transferul de fişiere şi transmiterea informaţiilor 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 secvenţa de iniţializare a conexiunii — care va fi descrisă mai jos — clientul şi serverul stabilesc un *identificator de sesiune* şi, pentru fiecare sens, o cheie de criptare şi o cheie de autentificare. De asemenea, se stabilesc algoritmii de criptare simetrică, compresie şi dispersie cu cheie utilizaţi pentru fiecare sens al comunicaţiei. Comunicaţia 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 construieşte şi se transmite pe conexiune un pachet generat astfel:

1. Datele utile sunt comprimate utilizând algoritmul de compresie curent pentru sensul de comunicaţie curent.
2. Se adaugă, după datele comprimate, un şir de octeţi aleatori, iar în faţa lor se adaugă un octet reprezentând lungimea şirului aleator. Apoi, în faţa şirului astfel obţinut, se adaugă lungimea totală a şirului, reprezentată pe patru octeţi. Numărul de octeţi aleatori adăugaţi trebuie astfel ales încât să rezulte în urma concatenării un şir de lungime multiplu de lungimea blocului cifrului utilizat.
3. Rezultatul pasului precedent se criptează.
4. În faţa 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 şi creşte cu 1 la fiecare pachet independent de eventuala schimbare a cheilor sau algoritmilor criptografici utilizaţi.
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 creşte entropia datelor de criptat, făcând mai dificilă spargerea cifrului. Octeţii adăugaţi la finalul blocului fac ca în cazul repetării aceluiaşi bloc de date utile să rezulte blocuri criptate diferite. Lungimea completării aleatoare este şi ea criptată, făcând dificilă determinarea lungimii datelor utile din blocul criptat. Pe de altă parte, dispersia criptografică cu cheie se calculează dintr-un bloc conţinând datele utile şi numărul de ordine al blocului, fapt ce permite receptorului să verifice că datele sunt autentice şi că sunt proaspete — numărul de ordine al blocului primit este cel aşteptat. 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 şi nivelele inferioare corectând erorile de transmisie la nivel fizic şi eventualele pierderi de pachete IP datorate unei congestii în reţea.

La deschiderea conexiunii *ssh*, compresia, criptarea şi dispersia cu cheie sunt dezactivate. Negocierea primului set de chei şi a algoritmilor de compresie, criptare şi dispersie cu cheie se face în clar. O dată alese cheile

şi algoritmii, acestea sunt activate şi se poate începe comunicaţia în folosul aplicaţiilor. Algoritmii şi cheile pot fi renegociate ulterior oridecâteori una dintre părţi (clientul sau serverul) o solicită.

Negocierea cheilor şi algoritmilor se face după cum urmează. Fiecare parte trimite liste, în ordinea descrescătoare a preferinţei, cu algoritmii de criptare, compresie, dispersie cu cheie, semnătură digitală şi schimb de chei suportate. Algoritmul utilizat, pentru fiecare categorie, este primul algoritm de pe lista clientului care se regăseşte şi în lista serverului (adică cel mai favorabil clientului, dintre cei acceptaţi de server). Urmează schimbul de mesaje conform protocolului Diffie-Hellman (*ssh* nu are definite deocamdată alte metode de schimb de chei). Din informaţia secretă construită prin schimbul Diffie-Hellman se construiesc (pe baza unor funcţii de dispersie fără cheie) cheile secrete pentru criptare şi 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 faţă de client (adică autentificarea, faţă 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 informaţie de până atunci — adică listele de algoritmi suportaţi şi pachetele corespunzătoare protocolul Diffie-Hellman, emise de ambele părţi. Prin verificarea semnăturii, clientul se asigură că negocierea a avut loc într-adevăr cu serverul autentic. Autentificarea clientului faţă 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 comunicaţiei 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 locat; 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 descoperită.

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 maşina server sau dintr-o sursă autentificată, cheia publică a serverului şi 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 obţine, dintr-o sursă autentificată (de exemplu, vorbind la telefon cu administratorul maşinii server), dispersia criptografică a cheii publice a serverului. La prima conectare, utilizatorul compară dispersia astfel obţinută cu dispersia cheii trimise de server (aceasta este afişată de clientul *ssh* împreună cu mesajul de avertisment prin care anunţă 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 fişierele `ssh_host_rsa_key.pub`, respectiv `ssh_host_dsa_key.pub`. Aceste fişiere pot fi citite de orice utilizator al sistemului. Amprenta cheii dintr-un astfel de fişier se determină cu comanda `ssh-keygen -l -f` *fişier*. Cleintul *ssh* memorează cheile serverelor în fişierul `~/.ssh/known_hosts`.

11.2.1.2. Metode de autentificare în *ssh*

În *ssh*, există autentificare reciprocă între client şi server.

Aşa cum am văzut, serverul se autentifică faţă de client cu ajutorul unui mecanism cu cheie privată şi cheie publică.

După iniţializarea mecanismului de protecţie criptografică a conexiunii, este rândul clientului să-şi declare identitatea (numele de utilizator) şi să se autentifice.

Clientul poate fi autentificat de server prin mai multe metode. Cele mai comune sunt autentificarea prin parolă şi autentificarea prin semnătură digitală (numită şi autentificare cu cheie publică).

Autentificarea prin parolă presupune trimiterea de către client a parolei. Este esenţial faptul că serverul este deja autentificat şi confidenţialitatea, integritatea şi prospeţimea comunicaţiei sunt protejate. Ca urmare clientul nu riscă să trimită parola unui adversar şi nici ca un adversar ce captează comunicaţia criptată să retrimită datele interceptate pentru a repeta o sesiune legitimă.

Autentificarea prin semnătură digitală presupune ca în faza de iniţializare utilizatorul să configureze pe server o cheie publică, corespunzătoare

cheii sale secrete. La conectare, clientul se autentifică trimiţând semnătura, cu cheia sa secretă, asupra identificatorului de sesiune creat în faza de stabilire a comunicaţiei 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 fişierul `~/.ssh/authorized_keys` (sub directorul personal al utilizatorului). Deoarece acest fişier poate fi modificat doar de către posesorul contului, doar posesorul contului poate stabili cheia admisibilă pentru autentificare. Fişierul `~/.ssh/autthorized_keys` poate conţine 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 aplicaţii; în acest caz, dacă clientul utilizează cheia pereche pentru autentificare, i se va lansa automat aplicaţia respectivă şi nu o sesiune nerestricţionată.

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 fişierul `~/.ssh/authorized_keys`, să şteargă vechea cheie publică din acest fişier şi să furnizeze noua cheie secretă clientului *ssh* la conectările ulterioare. Deoarece cheia publică nu este o informaţie secretă, compromiterea sistemului server nu duce la compromiterea, şi deci la necesitatea schimbării, cheii. Acesta este un avantaj faţă de cazul autentificării prin parole, unde compromiterea serverului duce la compromiterea parolei şi la necesitatea schimbării parolei nu numai pe acel sistem ci şi pe alte sisteme pe care utilizatorul avea aceeaşi parolă.

Pentru furnizarea cheii secrete către clientul *ssh*, există două posibilităţi. Prima posibilitate este ca fişierul cu cheia secretă să fie făcut disponibil clientului *ssh*. Dacă fişierul conţine cheia secretă ca atare, conectarea se face fără ca utilizatorul să mai dea vreo parolă. Dacă utilizatorul doreşte să se conecteze de pe maşini (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 soluţie 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 obişnuit, în formă criptată în fişiere. 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 siguranţă (un intrus trebuie să obţină atât fişierul cu cheia secretă, cât şi 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 aplicaţiei 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 obţine accesul la cheie.

Agentul de autentificare, având ca nume de executabil `ssh-agent`, se lansează de regulă la deschiderea sesiunii pe maşina client. Cheile secrete se încarcă în agentul de autentificare cu ajutorul unui program numit `ssh-add`. Acest program permite şi listarea şi ştergerea cheilor secrete. Dacă cheia secretă este stocată criptat, `ssh-add` solicită utilizatorului parola de decriptare. Cheia secretă decriptată se găseşte doar în memoria agentului de autentificare, nu se stochează pe disc.

La lansare, clientul *ssh* caută să vadă dacă pe maşina locală rulează un agent de autentificare. Dacă da, contactează agentul de autentificare şi-i solicită accesul la cheile secrete stocate de acesta. Clientul *ssh* pasează agentului identificatorul de sesiune şi primeşte înapoi semnătura cu cheia secretă asupra acestuia. În acest fel, clientul *ssh* nu ajunge să cunoască efectiv cheia secretă. Deschiderea sesiunii către maşina server se face fără a solicita utilizatorului vreo parolă.

Comunicaţia dintre clientul *ssh* sau utilitarul *ssh-add* şi agentul de autentificare se face printr-un *socket* de tip UNIX, al cărui nume este pus în variabila de mediu `SSH_AUTH_SOCK`. Comunicaţia 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 construcţia unei conexiuni securizate dinspre maşina server *ssh* spre agentul de autentificare de pe maşina client *ssh*. În acest caz, la deschiderea sesiunii, serverul *ssh* acţionează şi 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 maşina client). Prelungirea nu se poate face în lipsa unui agent de autentificare pe maşina client.

Analizând securitatea prelungirii conexiunii la agentul de autentificare, observăm că serverul nu obţine efectiv cheia secretă, însă, pe durata

conexiunii, poate deschide sesiuni în numele clientului către orice maşină 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 şi aplicaţii

O dată deschisă conexiunea şi realizată autentificarea clientului, clientul *ssh* poate solicita serverului deschiderea unei sesiuni de lucru, adică în esenţă lansarea unui *shell* în contul utilizatorului autentificat. Tot la solicitarea clientului, canalul de comunicaţie creat de server între server şi *shell*-ul lansat poate fi realizat printr-un pseudoterminal (cazul obişnuit 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 şi acesta o execută cu intrarea şi ieşirea redirecţionate către server şi prelungite prin conexiunea securizată către client. Redirecţionând pe client intrarea sau ieşirea standard a comenzii *ssh*, se realizează, per ansamblu, redirecţionarea intrării sau ieşirii standard către fişiere sau *pipe*-uri de pe maşina locală pentru comanda executată la distanţă.

Exemplul 11.5: Comanda

```
ssh rlupsa@nessie.cs.ubbcluj.ro ls -l > lista
```

are ca efect final crearea, pe maşina locală, a unui fişier `lista`, conţinând lista numelor de fişiere de pe maşina `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 distanţă în mod grafic.

Descriem în continuare arhitectura X Window. Menţionăm că este diferită faţă de sistemele studiate până aici. Diferenţa provine în primul rând din faptul că sistemul X Window are şi 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 şi ecranul local. O aplicaţie ce are nevoie de acces la un ecran grafic şi la

tastatura ataşată se numeşte *client X*. Un client X se conectează la serverul X şi, după autetificare, poate:

- să ceară serverului să deseneze diverse lucruri pe ecran;
- să solicite să primească informări cu privire la tastele apăsate de utilizator şi la mişcările mouse-ului.

La un server se pot conecta simultan mai mulţi clienţi, inclusiv de pe calculatoare diferite.

Serverul ţine evidenţa unor *ferestre*, fiecare operaţie 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 ţine evidenţa modului în care se suprapun ferestrele şi determină ce parte din desenul efectuat într-o fereastră este vizibil şi trebuie desenat pe ecran.

Un client autentificat are acces deplin la tastatura şi ecranul gestionate de server. Asta înseamnă, de exemplu, că un client poate să deseneze într-o fereastră deschisă de alt client şi poate să capteze tot ceea ce tastează utilizatorul în acea fereastră. De principiu, sunt admise la un moment dat să se conecteze doar aplicaţii rulând în contul aceluiaşi utilizator.

Pentru ca aplicaţii distincte să nu se încurce reciproc, există nişte convenţii pe care aplicaţiile se recomandă să le respecte. În linii mari, acestea prevăd ca o aplicaţie să nu deseneze în ferestrele deschise de altă aplicaţie şi să nu capteze tastele când nu este activă.

Comutarea între aplicaţii, precum şi mutarea şi redimensionarea ferestrelor principale ale aplicaţiilor, cad în sarcina unui client mai special numit *window manager*. *Window manager*-ul se conectează şi se autentifică ca un client obişnuit, după care solicită serverului să fie informat de cererile de deschidere de ferestre trimise de ceilalţi clienţi. La fiecare fereastră principală deschisă, *window manager*-ul adaugă bara de titlu şi marginile. Deoarece oricum nu există protecţie între clienţii conectaţi la un server X, un client nu are nevoie de privilegii speciale ca să acţioneze ca *window manager*. Totuşi, câteva dintre operaţiile de care are nevoie un *window manager* ca să funcţioneze 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 fişierelor în reţea

Cerinţa de-a transfera fişiere în reţea poate avea diferite particularităţi. Există mai multe protocoale şi mai multe aplicaţii pentru transferul fişierelor în reţea, adaptate pentru diferite necesităţi.

O primă categorie de protocoale şi aplicaţii priveşte, în principal, transferul fişierelor unui utilizator de pe o maşină pe alta, în condiţiile în care utilizatorul are cont pe ambele maşini. Protocoalele construite pentru aceasta sunt *ftp* şi *ssh*. De notat că şi poşta electronică poate servi ca mecanism de transfer de fişiere.

O a doua categorie priveşte transferul fişierelor publice de la un calculator ce stochează astfel de fişiere la calculatorul unui utilizator ce doreşte să citească fişierele respective. Iniţial se utiliza protocolul *ftp* în acest scop. Protocolul utilizat în mod curent este însă *http*.

O a treia categorie piveşte accesul proceselor de pe un calculator la fişiere stocate pe alt calculator ca şi când fişierele ar fi locale. De principiu fişierele respective sunt private, ca şi pentru prima categorie de protocoale. Protocoalele din această categorie trebuie să satisfacă două cerinţe specifice (faţă de prima categorie): să permită transferul doar a unei părţi mici dintr-un fişier şi să permită controlul partajării fişierului între procese. Protocoalele utilizate aici sunt SMB (utilizat în reţelele Windows) şi 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 numeşte *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 separaţi prin spaţii, atât faţă de numele comenzii cât şi 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 execuţiei comenzii. Cu o singură excepţie (în cazul comenzii PASV, descrisă mai jos), textul din răspuns nu este interpretat de către aplicaţia client. El este însă afişat, de obicei, pe ecran utilizatorului aplicaţiei client.

Autentificarea se face la solicitarea clientului. Clientul trimite succesiv două comenzi, `USER` şi `PASS`, având ca parametrii respectiv numele utilizatorului şi parola acestuia. Serverul refuză execuţia majorităţii comenzilor clientului înainte de autentificarea cu succes a acestuia. După autentificare, serverul acceptă să efectueze operaţiile cerute de client doar dacă utilizatorul în contul căruia s-a făcut autentificarea are dreptul la operaţiile 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 şi 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 fişiere.

Pentru transferul de fişiere publice, serverul este configurat să accepte autentificare cu numele de utilizator `ftp` sau `anonymous` fără să solicite parolă sau acceptând orice şir de caractere pe post de parolă. În vremurile de început ale Internet-ului, se obişnuia ca un utilizator ce dorea acces la fişiere publice să-şi dea, pe post de parolă, adresa sa de poştă electronică. O dată cu răspândirea *spam*-urilor, s-a renunţat la acest obicei.

Transferul fişierelor se cere prin comenzile `SEND` (dinspre client spre server) şi `RETR` (dinspre server spre client). Comenzile au ca argument numele de pe server al fişierului de transferat. Transferul propriu-zis are loc printr-o conexiune separată, numită *conexiune de date*. Pentru fiecare fişier se deschide o nouă conexiune de date, care se închide la finalul transferului fişierului. Dimensiunea fişierului nu este specificată explicit nicăieri, receptorul fişierului obţinând lungimea din faptul că emiţătorul închide conexiunea de date la finalul fişierului.

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 aşteaptă 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 obţine ca răspuns al comenzii `PASV` date de client. Acesta este singurul caz în care clientul interpretează din răspunsul serverului şi altceva decât codul returnat.

Listarea fişierelor de pe server este solicitată de client prin comanda `LIST`. Trasnferul listei de fişiere se face tot printr-o conexiune de date, ca şi în cazul comenzii `RETR`.

11.3.2. Protocolul HTTP

HyperText Transmission Protocol(HTTP) este un protocol elaborat pentru transferul dinspre server spre client a fişierelor cu informaţii disponibile

public. El înlocuieşte protocolul *ftp* utilizat cu conectare ca utilizator *anonymous*. Deşi numele protocolului face referire la hipertext, el poate fi utilizat pentru a transfera orice fel de conţinut.

Protocolul de bază constă în trimiterea de către client a unei cereri, în care informaţia principală este numele fişierului cerut. Răspunsul serverului conţine nişte informaţii despre fişier şi conţinutul efectiv al fişierului. Implicit, conexiunea se închide după transferul unui fişier. Dacă clientul doreşte mai multe fişiere de pe acelaşi server, va trebui să deschidă câte o conexiune pentru fiecare fişier.

Protocolul a fost însă extins, ajungând să fie folosit ca protocol de transfer de date pentru aplicaţii de orice tip.

11.3.2.1. Structura cererilor şi a răspunsurilor

Formatul comunicaţiei este mixt, atât la cereri cât şi la răspunsuri. Partea de început este text, iar conţinutul fişierului este binar.

Cererea cuprinde, pe prima linie, un cuvânt reprezentând numele operaţiei cerută. Pentru solicitarea unui fişier public de pe server, numele este `GET`. După numele operaţiei urmează numele fişierului şi apoi identificarea versiunii de protocol în conformitate cu care este formată cererea. Cele trei elemente sunt separate prin câte un spaţiu.

Următoarele linii sunt de forma *nume*: *valoare*, similar cu antetul unui mesaj de poştă electronică. După ultima linie de antet urmează o linie vidă. Pentru unele tipuri de cereri, după linia goală se găseşte un conţinut. În acest caz, una dintre liniile din antet are numele `Content-length` şi are ca valoare lungimea conţinutului, dată ca şir de cifre zecimale.

Răspunsul este structurat similar cu cererea. Pe prima linie se află identificatorul versiunii HTTP, număr de trei cifre şi un text. Numărul arată dacă cererea a fost satisfăcută cu succes sau nu, iar textul, neinterpretat de client, este o descriere în cuvinte a semnificaţiei codului de trei cifre. Următoarele linii sunt de forma *nume*: *valoare* şi dau informaţii despre fişierul solicitat. După ultima linie de antet urmează o linie vidă şi apoi conţinutul (binar) al fişierului. În antet se găseşte o linie cu numele `Content-length` având ca valoare lungimea fişierului. Determinarea sfârşitului conţinutului propriu-zis de către client trebuie făcută prin numărea octeţilor din partea de conţinut.

Adesea, mai multe servere *HTTP* sunt găzduite fizic pe acelaşi calculator. În acest caz, fie numele serverelor corespund, prin DNS, unor adrese IP diferite, dar aparţinând aceluiaşi calculator, caz în care serverul este configurat să răspundă în funcţie de IP-ul către care a fost deschisă conexiunea,

fie numele serverelor corespund aceleiaşi 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` şi având ca valoare numele de server dorit.

11.3.2.2. URL-urile

O pagină web este în general un fişier scris în *HyperText Markup Language* (HTTP) şi oferit în acces public prin protocolul HTTP.

O pagină web constă, de obicei, din mai multe fişiere. Există un fişier de bază, scris în limbajul *HTML*, şi alte fişiere, conţinând anumite elemente ale paginii: imagini (în fişiere separate în formate specifice — JPEG, PNG, GIF), applet-uri (Java), specificări de formatare a paginii (fişiere *Cascading Style Sheet* — CSS). De asemenea, o pagină conţine în general legături (*link*-uri) spre alte pagini. Toate acestea necesită referiri dintr-un fişier HTML către alte fişiere disponibile în acces public. Referirea acestor fişiere se face prin nume care să permită regăsirea lor uşoară.

Un *Universal Resource Locator* (URL) este un nume prin care se poate identifica şi cu ajutorul cărora se potate regăsi o resursă disponibile în Internet. URL-urile au apărut ca un format standard de scriere a numelor fişierelor 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 maşinii* este numele de domeniu sau adresa IP a maşinii pe care se găseşte resursa (fişierul).

  Pe lângă numele maşinii, în cadrul acestei componente se poate adăuga numele de utilizator în contul căruia trebuie să se autentifice un client pentru a obţine accesul dorit la resursă. Numele de utilizator se dă în faţa numelui sau adresei maşinii, separat de acesta prin caracterul `@`. Standardul original prevedea şi posibilitatea de-a scrie în URL parola necesară conectării. Această utilizare este nerecomandată.
- *Calea* identifică resursa (fişierul) în cadrul serverului care o găzduieşte. În principiu, ea este calea completă a fişierului cerut, relativă la un director de bază, fixat, al documentelor publice.

URL-urile se pot utiliza şi se utilizează efectiv în multe alte scopuri decât identificarea paginilor web. De exemplu, sistemul *SubVersion* (SVN) utilizează URL-uri de forma `svn://`*maşină*`/`*cale* pentru a referi fişierele dintr-un *repository*.

11.3.2.3. Alte facilităţi HTTP

Antetul răspunsului HTTP oferă mai multe informaţii despre fişierul returnat:

- Tipul conţinutului fişierului este specificat de către server prin intermediul unui antet cu numale `Content-type` şi cu valoarea construită ca şi în cazul antetului `Mime-type` de la poşta electronică. Tot ca şi în cazul lui `Mime-type`, tipul conţinutului poate fi urmat de specificarea codificării utilizate pentru text; de exemplu,

  ```
  Content-type: text/html; charset=utf-8
  ```

  înseamnă că fişierul este de tip HTML, iar codificarea utilizată pentru caractere este UTF-8.
- Data ultimei modificări a fişierului este specificată prin valoarea antetului cu numele `Date`.
- Tipul de compresie utilizat (dacă fişierul returnat este comprimat) este dat ca valoare a antetului `Content-transfer-encoding`.
- Limba în care este scris textul din fişier (dacă este cazul) este returnată ca valoare a antetului `Language:`.

Este posibil ca unui URL să-i corespundă mai multe fişiere pe server, având conţinut echivalent, dar în diverse formate, limbi sau codificări. Pentru a selecţiona varianta dorită, clientul poate anunţa posibilităţile şi preferinţele sale cu privire la tipul de fişier, limbă şi codificare. Antetele corespunzătoare, din cererea clientului, sunt: `Accept`, `Accept-language` şi `Accept-encoding`. Fiecare dintre acestea are ca valoare o listă de variante, în ordinea preferinţei. De exemplu,

```
Accept-language: ro,en,fr
```

solicită serverului, de preferinţă, 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 condiţionate sau parţiale. O cerere parţială este utilă dacă fişierul cerut este mare şi clientul doreşte să-l aducă din bucăţi sau dacă la o cerere precedentă a căzut legătura după transferul unei părţi din fişier. O cerere condiţionată determină serverul să transmită clientului fişierul numai dacă este îndeplinită o anumită condiţie, cel mai adesea dacă a fost modificat mai recent decât o anumită dată specificată

de client. Dacă nu este îndeplinită condiţia, serverul dă un răspuns format doar din antet, fără conţinutul propriu-zis. Această facilitate este utilă dacă clientul deţine o copie a unui fişier şi doreşte împrospătarea acesteia. Cererea parţială se specifică de către client prin intermediul antetului `Range`; cererea condiţionată se specifică prin antetul `If-modified-since`.

Pentru optimizarea traficului, în cazul în care un client doreşte mai multe fişiere de pe acelaşi server (aceasta se întâmplă adesea în cazul în care clientul aduce un fişier *html*, iar apoi are de adus imaginile şi 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 clienţi să deschidă o conexiune şi apoi să o lase deschisă la nesfârşit, ţinând 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 redirecţionat automat către alt URL. Aceasta se întâmplă dacă administratorul saitului decide o reorganizare a paginilor şi doreşte ca utilizatorii care au reţinut URL-uri desfiinţate în urma reorganizării să fie redirecţionaţi către paginile corespunzătoare din noua organizare. Această redirectionare se face prin trimiterea de către server a unui antet cu numele `Location` şi având drept conţinut URL-ul spre care se doreşte redirecţionarea clientului. În acest caz, programul client nu afişează conţinutul returnat de server (conţinut care în mod normal lipseşte complet), ci cere şi afişează conţinutul de la URL-ul indicat în antetul `Location`.

11.3.2.4. Proxy HTTP

Un *proxy HTTP* este un proces care, faţă de un client HTTP, acţionează aproape ca un server HTTP, iar pentru satisfacerea cererii contactează serverul solicitat de client şi acţionează, faţă de acest server, ca un client.

Un *proxy HTTP* este utilizat în mod normal pentru următoarele funcţii:

- dacă dintr-o reţea locală există şanse mari ca mai mulţi utilizatori să acceseze aceleaşi pagini web: În acest caz, clienţii *HTTP* ai calculatoarelor din reţea se configurează să utilizeze un acelaşi *proxy HTTP* din reţeaua locală. Dacă există mai multe cereri pentru aceeaşi pagină, la prima cerere *proxy*-ul memorează pagina adusă, iar la următoarele cereri o serveşte clienţilor din memoria locală.

- dacă într-o reţea se utilizează adrese IP locale (vezi § 10.2.4.1 şi § 10.7.2) şi nu se doreşte configurarea unui mecanism de translaţie de adrese (NAT, § 10.7.3): În acest caz, se instalează un *proxy HTTP* pe un calculator din reţeaua locală dar având şi adresă IP publică. Clientul accesează *proxy*-ul prin reţeaua locală, iar *proxy*-ul, având adresă publică, poate accesa fără restricţii serverul dorit.
- dacă este de dorit un control fin asupra paginilor ce pot fi accesate dintr-o reţea locală (de exemplu, pentru a restricţiona accesul angajaţilor la saituri nelegate de activitatea lor normală). În acest caz, se configurează un *proxy* în care se configurează toate restricţiile de acces dorite. Apoi, pentru a împiedica accesul direct, prin evitarea *proxy*-ului, pe ruterul ce leagă reţeaua internă la Internet se configurează un filtru de pachete (§ 10.7.1) care să blocheze pachetele adresate portului TCP 80 al serverelor exterioare.

O diferenţa între protocolul de comunicaţie dintre un client şi un *proxy* faţă 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 şi numele serverului).

O a doua diferenţă este dată de existenţa 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` şi apoi retrimite datele dinspre client direct spre server şi, reciproc, dinspre server înapoi spre client. În cazul unei cereri `CONNECT`, *proxy*-ul nu se implică mai departe în comunicaţia dintre client şi server. Ca urmare, în acest caz *proxy*-ul nu mai memorează paginile aduse şi nu permite filtrarea cererilor decât după serverul şi portul la care se conectează, în schimb permite tunelarea oricărui protocol (nu numai a protocolului HTTP) între un client dintr-o reţea cu adrese interne şi 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 comunicaţiei între clientul şi serverul HTTP. Protocolul este însă suficient de flexibil pentru a permite oricărei aplicaţii 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 şi pentru SSL.

Protocolul TLS presupune existenţa unei legături nesecurizate între un client (entitatea care iniţiază activ comunicaţia) şi un server (entitatea care aşteaptă să fie contactată de către client). Legătura nesecurizată este, în mod obişnuit, o conexiune TCP. Protocolul TLS oferă un serviciu de tip conexiune. TLS asigură confidenţialitatea şi autenticitatea datelor utile transportate. Datele utile transportate de TLS pot aparţine oricărui protocol. Protocolul ale cărui date sunt transportate ca date utile de către TLS este numit *tunelat* prin TLS.

În cadrul iniţierii 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 faţă 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 şi un certificat, semnat de o autoritate în care clientul are încredere, pentru cheia publică. La iniţierea conexiunii TLS, serverul transmite clientului certificatul său Clientul verifică faptul că numele din certificat coincide cu numele serverului la care dorea conectarea, că semnătura autorităţii de certificare asupra certificatului este validă, că autoritatea de certificare este de încredere şi, în final, utilizează cheia publică din certificatul clientului pentru a autentifica stabilirea cheii de sesiune. Dacă se doreşte şi autentificarea clientului faţă de server tot prin TLS, atunci clientul trebuie să aibă, la rândul său, o pereche de chei şi un certificat.

În vederea verificării semnăturii din certificat şi a faptului că semnatarul (autoritatea de certificare) este de încredere, fiecare dintre parteneri trebuie să aibă o listă cu cheile autorităţilor de certificare de încredere. Cheia unei autorităţi de certificare este, în mod obişnuit, plasată tot într-un certificat.

Certificatul unei autorităţi 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 numeşte *certificat autosemnat*

(engl. *self-signed certificate*) sau *certificat rădăcină* (engl. *root certificate*). În majoritatea cazurilor, clientul are o mulţime de certificate autosemnate ale autorităţilor în care are încredere.

Există mai multe produse soft pentru crearea perechilor de chei şi pentru crearea şi 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. Construcţia realizată se numeşte HTTPS.

URL-urile resurselor accesibile prin conexiuni securizate au, ca nume al protocolului, şirul 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 lanţ de conexiuni TCP prin intermediul *proxy*-ului), are loc mai întâi schimbul de mesaje legat de stabilirea cheii SSL sau TLS. După iniţializarea conexiunii securizate, prin canalul securizat are loc un dialog conform protocolului HTTP. Cu alte cuvinte, cererile şi răspunsurile HTTP constituie date utile pentru nivelul SSL sau TLS.

Autentificarea serverului, în cadrul protocolului TLS, necesită, aşa cum am văzut, ca navigatorul web să dispună de certificatele autorităţilor de certificare de încredere. În general, producătorii de navigatoare încorporează în acestea nişte certificate, ale unor autorităţi de certificare larg recunoscute. Utilizatorul poate însă să dezactiveze oricare dintre aceste certificate, precum şi să adauge alte certificate.

Atragem atenţia asupra unor particularităţi 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 acelaşi calculator, este necesar ca aceste saituri 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 acelaşi certificat indiferent de saitul dorit de client. Ca urmare, ar exista o nepotrivire între numele din certificat şi numele saitului solicitat de client. În consecinţă, clientul ar declara saitul ca fiind unul fals.

- O pagină web este formată, în mod obişnuit, din mai multe obiecte, cu URL-uri diferite (pagina HTML propriu-zisă şi imaginile din pagină). În aceste condiţii, este posibil ca, într-o aceeaşi pagină, unele dintre elemente să fie securizate şi celelalte elemente să fie nesecurizate. De asemenea, este posibil ca diferite elemente să provină de pe saituri diferite, autentificate prin certificate diferite. Într-un astfel de caz, navigatorul web trebuie să avertizeze utilizatorul.

11.4. PGP/GPG

Preety Good Privacy (PGP) este un program pentru criptarea şi semnarea digitală a mesajelor de poştă electronică şi a fişierelor 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 construcţia şi funcţionarea GnuPG.

11.4.1. Structura cheilor GnuPG

PGP criptează mesajele, utilizând metode simetrice şi chei efemere, transmite cheile efemere prin criptare asimetrică şi crează semnături electronice asupra mesajelor.

În acest scop, fiecare utilizator GnuPG trebuie să aibă nişte perechi de chei pentru criptare asimetrică şi pentru semnătură.

GnuPG memorează, în nişte fişiere gestionate de el, cheile publice şi private ale utilizatorului ce execută comanda `gpg`, precum şi cheile publice ale partenerilor utilizatorului ce execută `gpg`.

Descriem în continuare structura cheilor GnuPG, precum şi operaţiile ce pot fi executate asupra cheilor memorate local. Transmiterea cheilor publice între utilizatori va fi descrisă în § 11.4.2.

Afişarea cheilor publice din fişierele gestionate de GnuPG se face prin comanda

```
gpg --list-keys
```

Afişarea cheilor secrete se face prin comanda

```
gpg --list-secret-keys
```

11.4.1.1. Chei primare şi subchei

Cheile GnuPG sunt de două tipuri: *chei primare* şi *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 şi, 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 şi 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 aşa-numita *amprentă a cheii* (engl. *key fingerprint*). Aceasta este un şir de biţi, calculaţi, printr-o funcţie 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. Şansele ca două chei distincte să aibă acelaşi identificator lung sunt extrem de mici, astfel încât identificatorul lung este suficient pentru a identifica unic orice cheie. Totuşi, identificatorul lung nu este utilizabil, în locul amprentei cheii, pentru verificarea autenticităţii 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 acelaşi identificator scurt, el poate fi folosit pentru a ne referi la o cheie.

Identificatorul unei perechi de chei este calculat, printr-o funcţie de dispersie, din cheia publică din pereche.

Identificatorii scurţi ai cheilor pot fi listaţi 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 şi identităţi

Fiecărei chei primare îi este asociată una sau mai multe *identităţi*. Fiecare identitate este un nume complet de utilizator, format din trei componente: *numele real* (numele şi prenumele persoanei), *adresa de poştă electronică* şi, opţional, un *comentariu*. În scrierea numelui complet, adresa de poştă electronică se scrie între semne *mai mic* şi *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 identităţi. Acest lucru este util dacă un utilizator are mai multe adrese de poştă electronică şi doreşte asocierea tuturor acestora cu aceeaşi cheie.

Reciproc, un acelaşi 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 şi să le asocieze acelaşi nume complet.

Mai mult, această posibilitate este utilizată frecvent în situaţia în care cheia primară a unui utilizator expiră sau este revocată. În această situaţie, utilizatorul poate crea o nouă cheie primară căreia să-i asocieze acelaşi nume complet.

11.4.1.3. Generarea şi modificarea cheilor

Generarea unei chei primare se face cu comanda

```
gpg --gen-key
```

Comanda este interactivă, solicitând utilizatorului următoarele informaţii: tipul cheilor generate şi dimensiunea acestora, durata de valabilitate a cheilor, numele complet al utilizatorului şi parola utilizată pentru criptarea cheii secrete.

Comanda generează o pereche primară de chei (de semnătură) şi îi asociază o identitate. Opţional, comanda poate genera şi o sub-pereche de

chei de criptare, subordonată perechii primare. Ulterior, se pot adăuga noi subchei şi identităţi sau se pot şterge subcheile şi identităţile asociate.

La generarea cheilor, GnuPG afişează 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 comunicaţie.

Pentru o cheie primară dată, proprietarul ei poate crea (şi, eventual, şterge) subchei. Pentru acestea, se lansează comanda

`gpg --edit-key` *cheie*

La lansarea acestei comenzi, `gpg` aşteaptă, 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 ştergerea unei subchei se utilizează subcomenzile `key` şi `delkey`. Ştergerea unei subchei este utilă doar dacă subcheia nu a fost încă transmisă nimănui. Nu există o metodă simplă de a propaga ştergerea asupra cópiilor subcheii respective. Ca urmare, dacă proprietarul doreşte ca o subcheie, deja transmisă partenerilor săi, să nu mai fie utilizată, soluţia este revocarea subcheii şi nu ştergerea ei.

Pentru a adăuga, şterge sau revoca o identitate asociată unei chei, se lansează comanda

`gpg --edit-key` *cheie*

şi apoi se utilizează subcomenzile: `adduid`, `uid`, `deluid`, `revuid`, `primary`. După modificarea identităţilor asociate unei chei primare, este necesară retransmiterea cheii spre partenerii de comunicaţie (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, şi prin revocarea cheii. Controlul valabilităţii 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` şi `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 comunicaţie.

Generarea certificatului de revocare se face prin comanda

`gpg -a -o` *fişier* `--edit-key` *cheie*

Certificatul de revocare este scris în fişierul cu nume *fişier*. Pentru ca revocarea să aibă efect, certificatul de revocare trebuie importat prin comanda

`gpg --import` *fişier*

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 şi, 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 alţi 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 ţină într-un loc sigur. În cazul în care pierde cheia sau bănuieşte 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 obtine 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 anunţă 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` şi `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 fişier gestionat de GnuPG. Acest fişier 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` *fişier* `--export-secret-keys` *cheie*

Această comandă exportă cheia secretă primară identificată prin parametrul *cheie*, precum şi subcheile sale secrete. Cheile secrete pot fi importate prin comanda

`gpg --import` *fişier*

Acest lucru este util dacă utilizatorul lucrează pe mai multe calculatoare şi doreşte să utilizeze aceeaşi chei 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` *fişier* `--export-secret-subkeys` *cheie*

Cu ajutorul acestei comenzi, un utilizator poate ţine cheia primară secretă pe un calculator sigur şi poate transmite subcheile secrete către un calculator mai puţin sigur pe care îl utilizează frecvent. În acest mod, el poate utiliza calculatorul mai nesigur pentru transmite mesaje semnate şi 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 crează cu durată de valabilitate scurtă şi se revocă la nevoie. Cheia primară este bine să aibă durată lungă de utilizare pentru a beneficia de semnăturile obţinute de la alţi utilizatori asupra ei (vezi § 11.4.2.2).

11.4.2. Transmiterea şi certificarea cheilor publice

11.4.2.1. Transmiterea cheilor publice

Cheile publice primare, identităţile asociate, semnăturile asupra identităţilor (vezi § 11.4.2.2), subcheile publice subordonate cheilor primare şi certificatele de revocare ale cheilor primare sau subcheilor sunt memorate într-un fişier 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 autenticităţii este descrisă în paragraful următor.

11.4.2.2. Verificarea autenticităţii 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 şi numai dacă cheia primară coresunzătoare este considerată autentică. În consecinţă, dacă se importă noi sub-chei pentru o cheie primară declarată autentică, sub-cheile respective sunt imediat considerate autentice.

Restul paragrafului de faţă 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 opţiune

`--trusted-key` *cheie*

fie la execuţia comenzii `gpg`, fie în fişierul cu opţiunile implicite. În afara acestor două cazuri, GnuPG consideră o cheie autentică numai dacă dispune de un certificat valid pentru ea şi mai sunt îndeplinite următoarele condiţii:

- cheia cu care este semnat certificatul este deja declarată ca autentică,
- semnatatul certificatului este considerat de încredere.

GnuPG reţine, asociate fiecărei chei primare, două informaţii (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 acelui utilizator asupra unei identităţi 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*

şi 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 şi în proprietarii cheilor specificate prin opţiunea `--trusted-key`.

Crearea, de către utilizatorul ce execută `gpg`, a unei semnături asupra unei identităţi 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 şi altor utilizatori GnuPG (vezi § 11.4.2.1). A doua comandă crează o semnătură pentru uz local.

Este esenţial, 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:

- Fişierul 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*) şi 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 şi semnarea setului de chei, utilizatorul verifică amprenta cheii primare din set.

11.4.3. Transmiterea mesajelor criptate sau semnate

Crearea unui mesaj criptat şi semnat se face astfel:

`gpg -o` *fiş-ieşire* `-r` *cheie-dest* `-se` *fiş-intrare*

sau

`gpg -a -o` *fiş-ieşire* `-r` *cheie-dest* `-se` *fiş-intrare*

unde *fiş-intrare* este fişierul ce trebuie semnat şi criptat, *fiş-ieşire* este fişierul în care comanda `gpg` va pune datele criptate şi semnate, iar *cheie-dest* reprezintă numele utilizatorului destinaţie sau identificatorul cheii de criptare de utilizate. Cea de-a doua variantă produce un fişier 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 opţiuni `-r`, fiecare urmată de numele unui utilizator destinaţie. Oricare dintre destinatarii astfel specificaţi 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 mulţi 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` *fiş-ieşire* `--decrypt` *fiş-intrare*

unde *fiş-intrare* este fişierul semnat şi criptat, iar *fiş-ieşire* este fişierul în care comanda `gpg` va pune rezultatul decriptării. Comanda verifică şi semnătura şi afişează 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 autenticităţii mesajului. Comanda de criptare este similară cu cea pentru criptare şi semnare, dar cu `-e` în loc de `-se`. Comanda de decriptare este identică cu cea pentru un mesaj criptat şi semnat.

Pentru generarea unui mesaj semnat dar necriptat există trei posibilităţi: semnătură inclusă în mesaj, semnătură detaşată şi semnătură în clar.

Semnătura inclusă se generează similar cu generarea unui mesaj criptat şi semnat, dar lipsesc destinatarii (opţiunile `-r`) şi în loc de `-se` se dă doar `-s`. Fişierul generat conţine datele originale şi semnătura. Extragerea datelor

şi verificarea semnăturii se face exact ca în cazul unui mesaj criptat şi semnat, adică prin comanda:

`gpg -o` *fiş-ieşire* `--decrypt` *fiş-intrare*

Semnătura detaşată se generează prin comanda

`gpg -a -o` *fiş-sign* `--detach-sign` *fiş-date*

Rezultatul comenzii este scrierea în fişierul *fiş-sign* a semnăturii conţinutului fişierului *fiş-date*. Fişierul produs, *fiş-sign*, este mic şi conţine doar semnătura; datele utile nu pot fi recuperate din el. Verificarea semnăturii se face prin comanda:

`gpg --verify` *fiş-sign fiş-date*

Semnătura detaşată este utilă deoarece păstrează intact fişierul de date, nefiind nevoie de `gpg` pentru recuperarea datelor. De asemenea, permite mai multor utilizatori să semneze un acelaşi fişier de date.

În fine, semnătura în clar se poate utiliza doar dacă datele sunt text ASCII. Semnătura se generează prin comanda:

`gpg -o` *fiş-ieşire* `--clearsing` *fiş-intrare*

Fişierul astfel produs este un fişier text, care poate fi citit uşor de către utilizatorul uman. Textul original este pus între nişte marcaje, iar semnătura este adăugată la sfârşit. Verificarea semnăturii se face prin comanda:

`gpg --verify` *fiş*

Semnătura în clar este utilă pentru semnarea documentelor text. Acestea rămân uşor de citit de către om şi, spre deosebire de semnătura detaşată, datele utile şi semnătura sunt puse într-un singur fişier.

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 opţiunea

`-u` *cheie*

GnuPG se utilizează curent pentru autentificarea softului liber. În acest scop, alături de programul distribuit, se distribuie un fişier ce conţine semnătura detaşată a fişierului ce conţine 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 şi 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. *802.11F^{TM} IEEE Trial-Use Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11^{TM} Operation*, 2003.

[IEEE 802.11i, 2004] IEEE Computer Society. *802.11i^{TM} 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. *802.1Q^{TM}. IEEE Standards for Local and metropolitan area networks: Virtual Bridged Local Area Networks*, 2003.

[IEEE 802.1X, 2001] IEEE Computer Society. *802.1X^{TM}. 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] *Icremental Zone Transfer in DNS*, 1996.

[RFC 1996, 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 informaţiei*. 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. *Reţele de calculatoare*. Byblos, 2003.

Index