Arhitectura Retelelor

Liviu P. Dinu
ldinu@fmi.unibuc.ro
Center for Computational Linguistics
nlp.unibuc.ro

Cuprins

1. Arhitectura retelelor
2. Nivelul fizic
3. Nivelul legatura de date
   a) Coduri corectoare si detectoare de erori
   b) Protocoale pentru legatura de date
4. Nivelul retea
   a) Algoritmi de dirijare si control
5. Nivelul Transport
   a) Protocoale transport
6. Nivelul aplicatie: DNS, email, ...
7. Securitatea retelelor: algoritmi cu cheie secreta si publica, protocoale de autentificare, ...
8. Retele sociale si aplicatii

Motivatie:

- În secolul XX(I), tehnologia cheie este legată de colectarea, prelucrarea şi distribuirea informaţiei
- Diferenţele între colectarea, transportul, stocarea şi prelucrarea informaţiei dispar pe zi ce trece
- Pe măsură ce posibilităţile de colectare, prelucrare şi distribuire a informaţiei cresc tot mai mult, cererea pentru o prelucrare sofisticată a informaţiei creşte şi mai rapid
- Amestecul domeniilor calculatoarelor şi comunicaţiilor au influentat modul de organizare a centrelor de calcul.

Definitii

- Vechiul model al unui singur calculator care serveşte rezolvării problemelor de calcul ale unei organizații a fost înlocuit de un model în care munca este făcută de un număr mare de calculatoare separate, dar interconectate.

- Aceste sisteme se numesc **rețele de calculatoare.**

Definitii

- O rețea de calculatoare este o colecție de calculatoare autonome interconectate folosind o singură tehnologie.

- Două calculatoare sunt interconectate dacă sunt capabile să schimbe informaţie între ele.

- **Atentie:** rețea de calculatoare <> **sistem distribuit**

Utilizari

- **Aplicaţii comerciale:**
  - Legate in special de împărţirea resurselor, cu scopul de a face toate programele, echipamentele şi în special datele disponibile pentru oricine din reţea, indiferent de localizarea fizică a resursei şi a utilizatorului
  - O reţea de calculatoare poate constitui un puternic mediu de comunicare între angajaţi (v. email)
  - Comert electronic

Utilizari (cont)

- **Aplicații domestice**
  - Accesul la informaţie de la distanţă, comunicațiile interpersonale, divertismentul interactiv, comerțul electronic
  - **Comunicatia de la egal-la-egal (peer-to-peer)**
- Utilizatorii mobili
- Aspecte sociale

Client-server

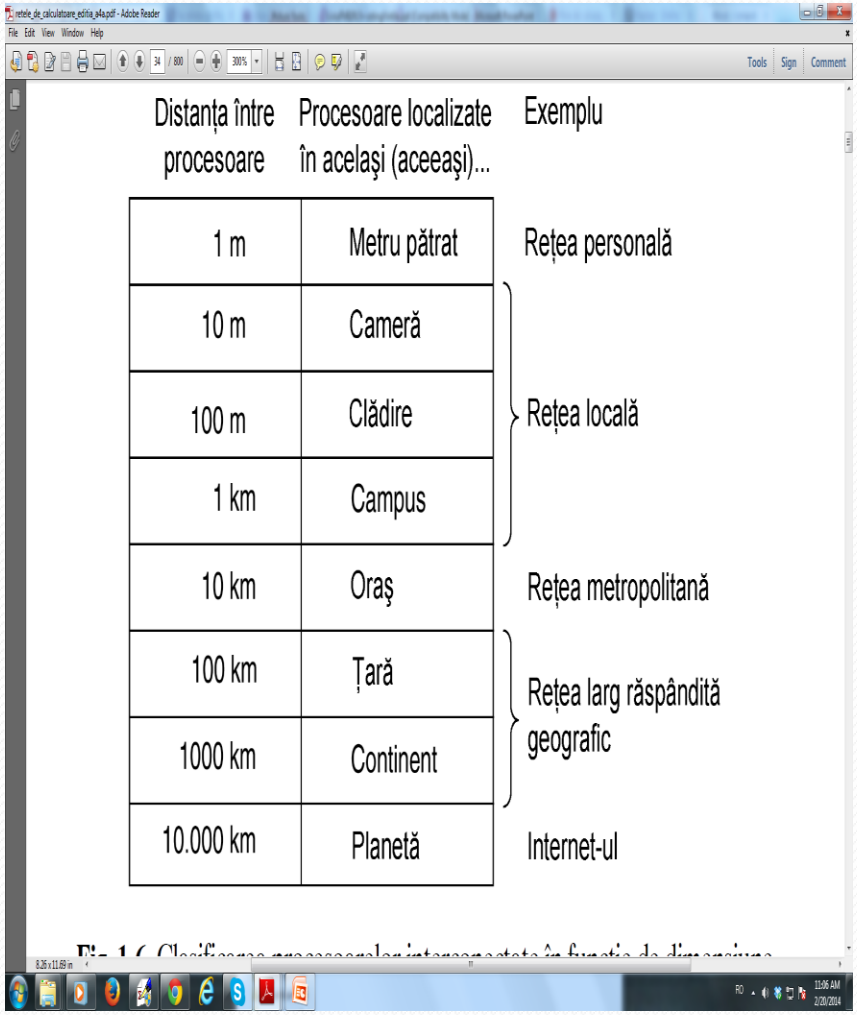
- **Modelul client-server:** datele sunt memorate în calculatoare performante, numite **servere (servers),** angajaţii (utilizatorii) au maşini mai simple, numite clienţi (clients), plasate pe birourile lor, prin intermediul cărora accesează datele aflate la distanţă

Client-server (cont)

- Sunt implicate două procese, unul aflat pe mașina client și unul aflat pe mașina server.
  - Comunicația ia forma transmiterii prin rețea a unui mesaj de la procesul client către procesul server.
  - În continuare, procesul client va aștepta un mesaj de răspuns. Atunci când procesul server primeşte cererea, execută acţiunea solicitată sau caută datele cerute şi transmite un răspuns.

Aspecte tehnice

- Criterii:
  - tehnologia de transmisie
  - scara la care operează rețeaua.
  - Un criteriu alternativ pentru clasificarea rețelelor este mărimea lor.
- Două tipuri de tehnologii de transmisie:
  - Legături cu difuzare.
  - Legături punct-la-punct



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

Reţelele cu difuzare (1)

- **Reţelele cu difuzare** au un singur canal de comunicaţii care este partajat de toate maşinile din reţea.
- Orice maşină poate trimite mesaje scurte, numite în anumite contexte **pachete, care sunt primite** de toate celelalte maşini.
- Un câmp de adresă din pachet specifică maşina căreia îi este adresat pachetul.

Reţelele cu difuzare(2)

- La recepţionarea unui pachet, o maşină controlează câmpul de adresă. Dacă pachetul îi este adresat, maşina îl prelucrează; dacă este trimis pentru o altă maşină, pachetul este ignorat.

- Sistemele cu difuzare permit adresarea unui pachet către *toate destinaţiile, prin folosirea* unui cod special în câmpul de adresă. Un pachet transmis cu acest cod este primit şi prelucrat de toate maşinile din rețea. Acest mod de operare se numește **difuzare.**

Reţelele cu difuzare(3)

- **Unele sisteme cu difuzare** suportă transmisia multipla (la un subset de maşini) **Una din schemele posibile este să se rezerve un bit pentru a indica trimiterea multiplă.**Restul de *n* - *1 biţi de adresă pot forma un număr de grup. O maşină se poate ,,abona" la orice grup* sau la toate grupurile. Un pachet trimis unui anumit grup va ajunge la toate maşinile abonate la grupul respectiv.

Reţelele cu difuzare (4)

- Statice şi dinamice, în funcţie de modul de alocare a canalului.
- O metodă tipică de alocare statică ar fi să divizăm timpul în intervale discrete şi să rulăm un algoritm round-robin, lăsând fiecare maşină să emită numai atunci când îi vine rândul.
- Alocarea statică iroseşte capacitatea canalului atunci când o maşină nu are nimic de transmis în cuanta de timp care i-a fost alocată, astfel că majoritatea sistemelor încearcă să aloce canalul dinamic (la cerere).

Reţelele cu difuzare (5)

- Metodele de alocare dinamică pentru un canal comun sunt fie centralizate, fie descentralizate.

- În cazul metodei centralizate de alocare a canalului există o singură entitate, (ex o unitate de arbitrare a magistralei) care determină cine urmează la rând. Poate face acest lucru acceptând cereri şi luând o decizie conform unui algoritm intern.

Reţelele cu difuzare (6)

- În cazul metodei descentralizate de alocare a canalului nu există o entitate centrală; fiecare maşină trebuie să hotărască pentru ea însăşi dacă să transmită sau nu.

- Exista numeroşi algoritmi proiectaţi să refacă ordinea dintr-un potenţial haos

Reţelele punct-la-punct

- **Dispun de numeroase conexiuni între perechi de maşini individuale.**

- Pentru a ajunge de la sursă la destinaţie pe o reţea de acest tip, un pachet ar putea să treacă prin una sau mai multe maşini intermediare. Sunt posibile trasee multiple, de diferite lungimi, şi de aceea descoperirea drumurilor celor mai potrivite este foarte importantă.

Reţelele punct-la-punct (cont.)

- Ca o regulă generală reţelele mai mici, localizate geografic, tind să utilizeze difuzarea, în timp ce reţelele mai mari sunt de obicei punct-la-punct.

- Transmisiile punct la punct cu un sigur transmițător și un singur receptor sunt numite uneori și **unicasting.**

Reţele locale (Local Area Networks - LAN)

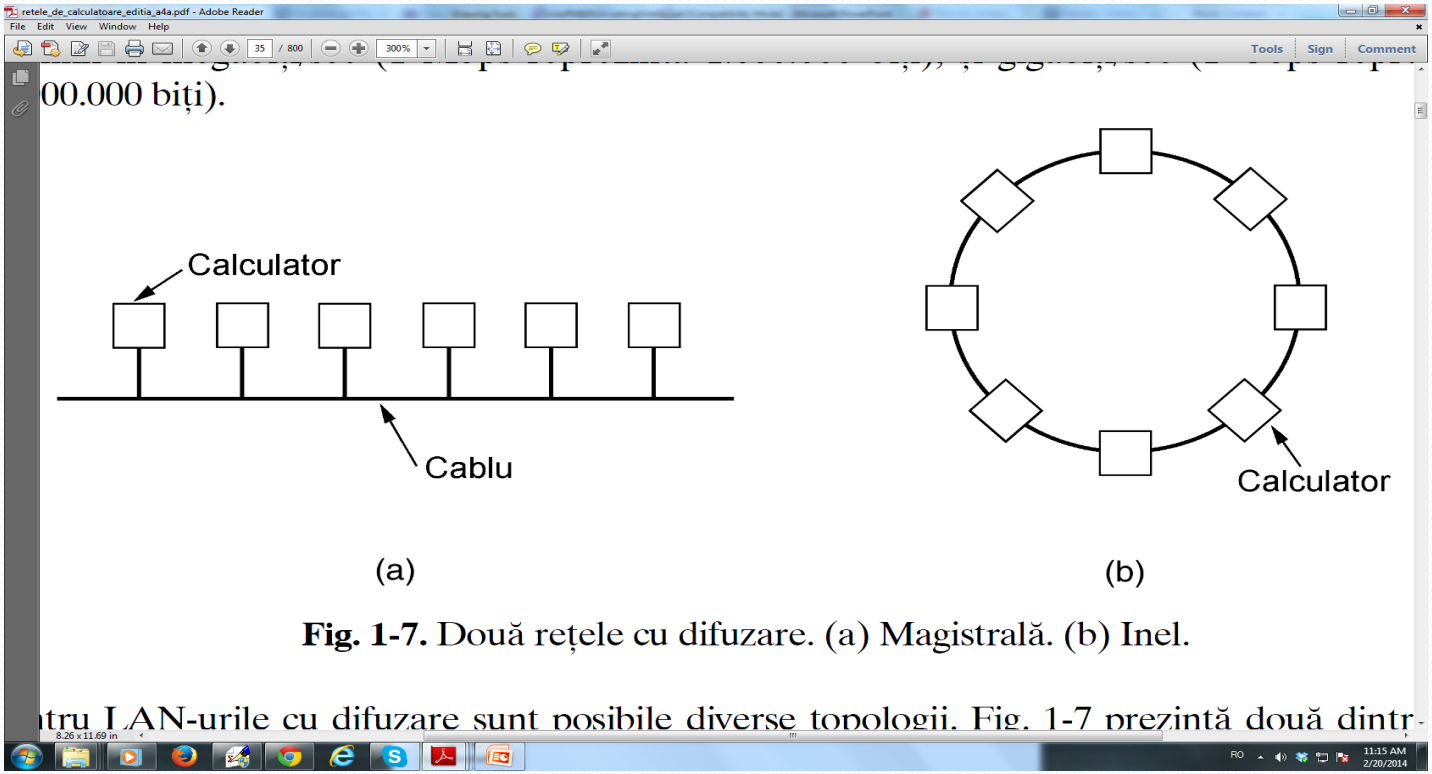
- Sunt rețele private localizate într-o singură clădire sau într-un campus de cel mult câţiva kilometri. Sunt frecvent utilizate pentru a conecta calculatoarele personale și stațiile de lucru din birourile companiilor și fabricilor, în scopul de a partaja resurse (imprimante, de exemplu) și de a schimba informaţii.
- LAN-urile se disting de alte tipuri de rețele prin trei caracteristici: (1) mărime, (2) tehnologie de transmisie și (3) topologie.

LAN (2)

- LAN-urile au dimensiuni restrânse, ceea ce înseamnă că timpul de transmisie în cazul cel mai defavorabil este limitat şi cunoscut dinainte
- LAN-urile utilizează frecvent o tehnologie de transmisie care constă dintr-un singur cablu la care sunt ataşate toate maşinile, aşa cum erau odată cablurile telefonice comune în zonele rurale.
- LAN-urile tradiţionale funcţionează la viteze cuprinse între 10 şi 10 Gbps, au întârzieri mici (microsecunde sau nanosecunde) şi produc erori foarte puţine

Topologii pt. LAN cu difuzare

- Doua topologii:
  - magistrala (cu cablu liniar)
  - inel



Reţea cu magistrală

- In fiecare moment cel mult una dintre maşini este master şi are dreptul să transmită. Restul maşinilor nu pot transmite. Când două sau mai multe maşini vor să transmită simultan, este necesar un mecanism de arbitrare. Mecanismul de arbitrare poate fi centralizat sau distribuit.

Reţea cu magistrală (cont)

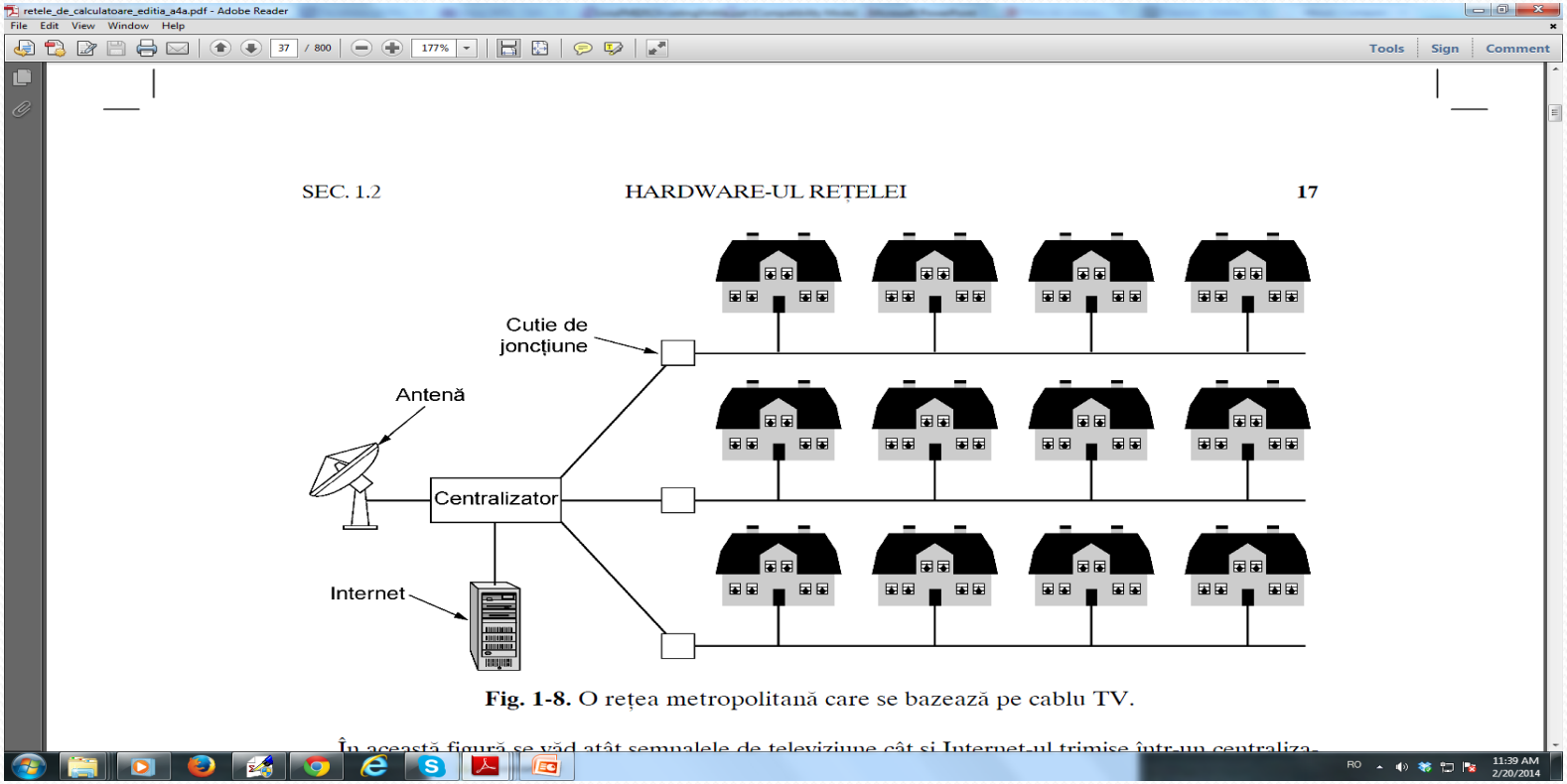
- Exemplu, IEEE 802.3, (**EthernetTM**), **este o rețea cu difuzare** bazată pe magistrală cu control descentralizat, lucrând la viteze între 10 Mbps şi 10 Gbps. Calculatoarele dintr-un Ethernet pot transmite oricând doresc; dacă două sau mai multe pachete se ciocnesc,fiecare calculator aşteaptă o perioadă de timp aleatorie şi apoi încearcă din nou.

Inel

- Într-un inel fiecare bit se propagă independent de ceilalţi, fără să aştepte restul pachetului căruia îi aparţine. În mod tipic, fiecare bit navighează pe circumferința întregului inel într-un interval de timp în care se transmit doar câțiva biți, de multe ori înainte chiar ca întregul pachet să fi fost transmis.
-
- Ca în orice alt sistem cu difuzare, este nevoie de o regulă pentru a arbitra accesele simultane la inel. (inelul cu jeton de la IBM sau **FDDI (Fiber Distributed Data Interface, rom: Interfaţă de date distribuite pe fibră optică).**

Reţea metropolitană (MAN-Metropolitan Area Network)

- Deserveşte un oraş (ex. Retea de televiziune).



Reţea larg răspândită geografic (Wide Area Network- WAN)

- Acoperă o arie geografică întinsă (o ţară sau un continent).
- Conţine o colecţie de maşini utilizate pentru a executa programele utilizatorilor (aplicaţii) - maşini **gazde.**
- **Gazdele sunt conectate printr-o subreţea de comunicaţie (pe scurt, subreţea).**
- Gazdele aparţin clienţilor (de exemplu calculatoarele personale ale oamenilor), deşi subreţeaua de comunicaţie aparţine şi este exploatată, de cele mai multe ori, de o companie de telefonie sau de un furnizor de servicii Internet (ISP).

WAN (2)

- Sarcina subreţelei este să transporte mesajele de la gazdă la gazdă, exact aşa cum sistemul telefonic transmite cuvintele de la vorbitor la ascultător.
- Prin separarea aspectelor de pură comunicaţie ale reţelei (subreţelei) de aspectele referitoare la aplicaţii (gazde), proiectarea întregii reţele se simplifică mult.
- În majoritatea WAN, subreţeaua este formată din două componente distincte:
  - liniile de transmisie
  - elementele de comutare.

WAN: transmisie si comutare

- **Liniile de transmisie transportă biţii între**
- maşini. Ele pot fi alcătuite din fire de cupru, fibră optică sau chiar legături radio.
- **Elementele de comutare** sunt calculatoare specializate, folosite pentru a conecta două sau mai multe linii de transmisie.
- Când sosesc date pe o anumită linie, elementul de comutare trebuie să aleagă o nouă linie pentru a retransmite datele mai departe. Terminologie: **ruter.**

Retele fara fir si retele casnice

- Reţelele fără fir pot fi împărţite în 3 mari categorii:
  - Interconectarea componentelor unui sistem
  - LAN-uri fără fir
  - WAN-uri fără fir

Retele casnice

- Dispozitivele din casă vor fi capabile să comunice intre ele și toate vor fi accesibile prin Internet
  - Dispozitive pentru telecomunicații (telefon, fax
  - Aparatura casnică (cuptorul cu microunde, frigiderul, ceasul, cuptorul, aparatul de aer condiționat, luminile)
  - Contoarele și alarmele (contoare pentru utilități, alarme de fum sau hoți, termostate
  - sisteme de supraveghere a copilului (parintilor?)

Retele casnice. Proprietati

- Au proprietăţi fundamental diferite de alte tipuri de reţele.

1) reţeaua şi dispozitivele trebuie să fie uşor de instalat
2) trebuie să fie protejate împotriva utilizării neglijente
3) preţul scăzut
4) Au nevoie de performanţe mai bune (multimedia)
5) să fie posibil să se pornească cu unul sau două dispozitive; interfata sa fie stabila, cablajul pe veci
6) securitatea și siguranța sunt foarte importante (alarme, etc.)
7) Cu fir sau fara fir?

Programe de retea

- În proiectarea primelor rețele de calculatoare, s-a acordat atenție în primul rând echipamentelor, iar programele au fost gândite ulterior.

Ierarhiile de protocoale

- Majoritatea rețelelor sunt organizate sub forma unei serii de **straturi sau niveluri, fiecare din ele construit peste cel de dedesubt**
- Scopul fiecărui nivel este să ofere anumite servicii nivelurilor superioare, protejându-le totodată de detaliile privitoare la implementarea efectivă a serviciilor oferite

Niveluri, protocoale

- Nivelul *n de pe o maşină conversează cu nivelul n de pe altă maşină. Regulile şi convenţiile utilizate* în conversaţie sunt cunoscute sub numele de **protocolul nivelului *n*.**
  - ***Un protocol*** reprezintă o înţelegere între părţile care comunică, asupra modului de realizare a comunicării.
- Entităţile din niveluri corespondente de pe maşini diferite se numesc **egale. Entităţile egale pot fi procese, dispozitive hardware, sau chiar fiinţe** umane. Cu alte cuvinte, entităţile egale sunt cele care comunică folosind protocolul

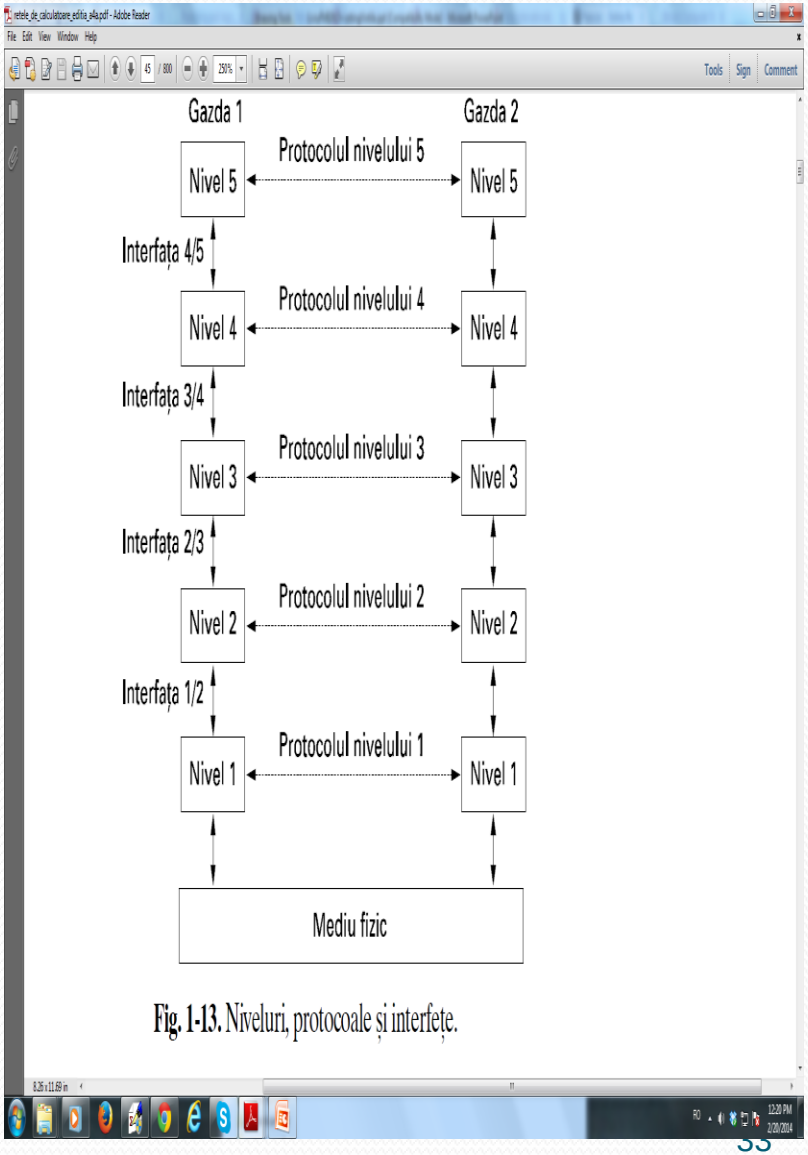


Fig. 1-13. Niveluri, protocoale și interfețe.

Proiectare si arhitectura

- Nici un fel de date nu sunt transferate direct de pe nivelul *n al unei maşini pe nivelul n al altei maşini. Fiecare nivel transferă datele şi informaţiile de control nivelului imediat inferior,* până când se ajunge la nivelul cel mai de jos. Sub nivelul 1 se află **mediul fizic prin care se produce** comunicarea efectivă.
- Între două niveluri adiacente există o **interfaţă.**
- **Interfaţa defineşte** ce operaţii şi servicii primitive oferă nivelul de jos către nivelul de sus.

Proiectare

- Proiectarea unei reţele implica cate niveluri să includă şi ce are de făcut fiecare din ele, precum si definirea de interfețe clare între niveluri.

- Aceasta presupune ca, la rândul său, fiecare nivel să execute o colecţie specifică de funcţii clar definite.

Proiectare (cont.)

- Minimizarea volumului de informaţii care trebuie transferate între niveluri; interfeţele clare permit o mai simplă înlocuire a implementării unui nivel cu o implementare complet diferită
- Aşa ceva este posibil, pentru că tot ceea ce i se cere noii implementări este să furnizeze nivelului superior exact setul de servicii pe care îl oferea vechea implementare. De altfel, este un fapt obișnuit ca doua gazde să folosească implementări diferite

Arhitectura de retea

- O mulţime de niveluri şi protocoale este numită **arhitectură de reţea.**
- **Specificaţia unei arhitecturi** trebuie să conţină destule informaţii pentru a permite unui proiectant să scrie programele sau să construiască echipamentele necesare fiecărui nivel, astfel încât nivelurile să îndeplinească corect protocoalele corespunzătoare.
- Detaliile de implementare şi specificaţiile interfeţelor nu fac parte din arhitectură, deoarece acestea sunt ascunse în interiorul maşinilor şi nu sunt vizibile din afară. Nu este necesar nici măcar ca interfeţele de pe maşinile dintr-o reţea să fie aceleaşi.

Exemplu

- O aplicaţie care se execută în nivelul 5 produce un mesaj M şi îl furnizează nivelului 4 pentru a-l transmite.
- Nivelul 4 inserează un **antet în faţa mesajului, pentru a** identifica respectivul mesaj şi pasează rezultatul nivelului 3. Antetul include informaţii de control, de exemplu numere de ordine care ajută nivelul 4 de pe maşina de destinaţie să livreze mesajele în ordinea corectă în cazul în care nivelurile inferioare nu păstrează această ordine.

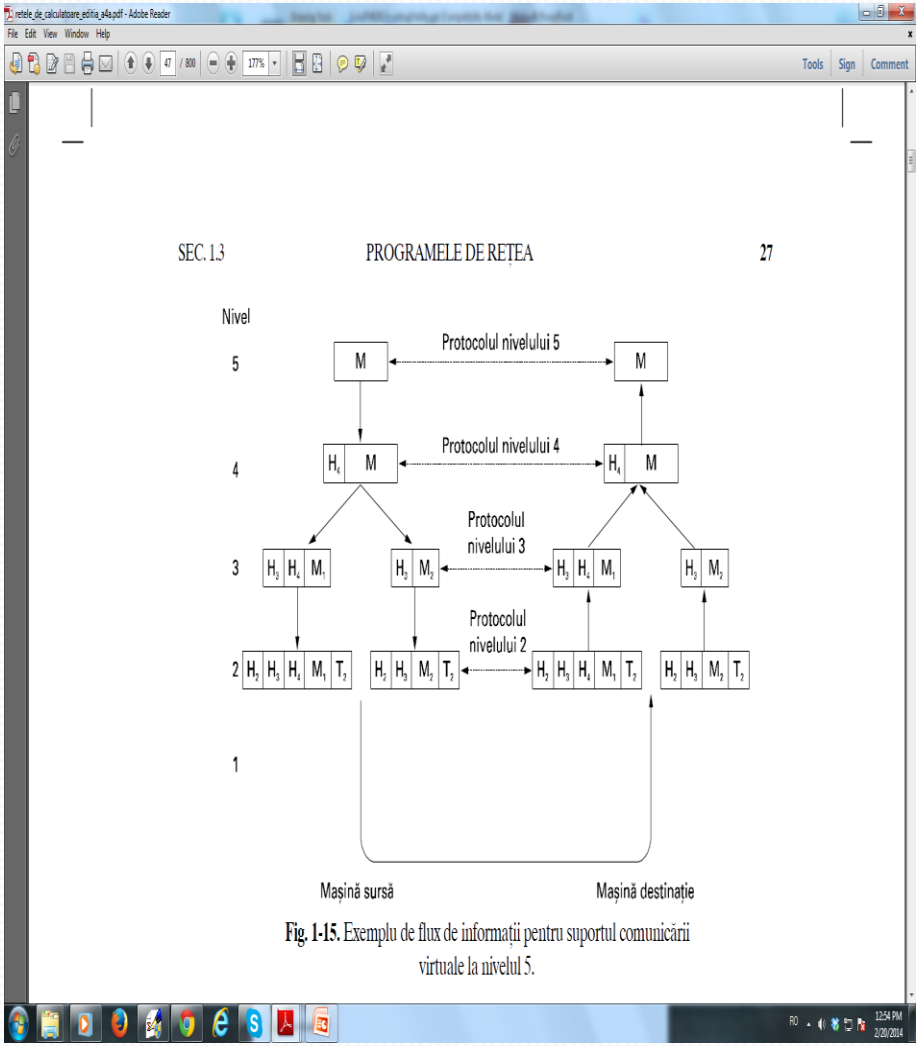


Fig. 1-15. Exemplu de flux de informaţii pentru suportul comunicării virtuale la nivelul 5.

Exemplu (2)

- Pe unele niveluri, antetele conţin de asemenea câmpuri de control pentru mărime, timp şi alte informaţii.
- În numeroase reţele nu există nici o limită cu privire la mărimea mesajelor transmise în protocolul nivelului 4, dar există aproape întotdeauna o limită impusă de protocolul nivelului 3.
- În consecinţă, nivelul 3 trebuie să spargă mesajele primite în unităţi mai mici, pachete, ataşând fiecărui pachet un antet specific nivelului 3. În exemplu, M este descompus în două părţi, *M1 şi M2*.

Exemplu (3)

- Nivelul 3 decide ce linie de transmisie să utilizeze şi trimite pachetele nivelului 2. Nivelul 2 adaugă nu numai câte un antet pentru fiecare bucată, ci şi o încheiere, după care furnizează unitatea rezultantă nivelului 1 pentru a o transmite fizic. În maşina receptoare mesajul este trimis în sus, din nivel în nivel, pe parcurs fiind eliminate succesiv toate antetele.
- Nici un antet corespunzător nivelurilorde sub *n nu este transmis în sus nivelului n.*

Probleme de proiectare a nivelurilor

- Fiecare nivel are nevoie de un mecanism pentru a identifica emițătorii și receptorii.

- **Controlul erorilor este o problemă importantă deoarece circuitele fizice de comunicații nu sunt** perfecte

Probleme

- O problemă ce intervine la fiecare nivel se referă la evitarea situaţiei în care un emiţător rapid trimite unui receptor lent date la viteză prea mare (controlul fluxului)
- incapacitatea tuturor proceselor de a accepta mesaje de lungime arbitrară.
- când există mai multe căi între sursă şi destinaţie, trebuie ales un anumit drum (**dirijare sau rutare**)

Modele de referinţă

- **ISO OSI (Open Systems Interconnection, rom: interconectarea sistemelor deschise);** se ocupă de conectarea sistemelor deschise - adică de sisteme deschise comunicării cu alte sisteme.
- Vom folosi mai ales termenul prescurtat de model OSI.
- Modelul OSI cuprinde șapte niveluri.

Principiile OSI

- 1. Un nivel trebuie creat atunci când este nevoie de un nivel de abstractizare diferit.
- 2. Fiecare nivel trebuie să îndeplinească un rol bine definit.
- 3. Funcţia fiecărui nivel trebuie aleasă acordându-se atenţie definirii de protocoale standardizate pe plan internaţional.
- 4. Delimitarea nivelurilor trebuie făcută astfel încât să se minimizeze fluxul de informaţii prin interfețe
- Numărul de niveluri trebuie să fie suficient de mare pentru a nu fi nevoie să se introducă în acelaşi nivel funcţii diferite şi suficient de mic pentru ca arhitectura să rămână funcţională

Modelul OSI

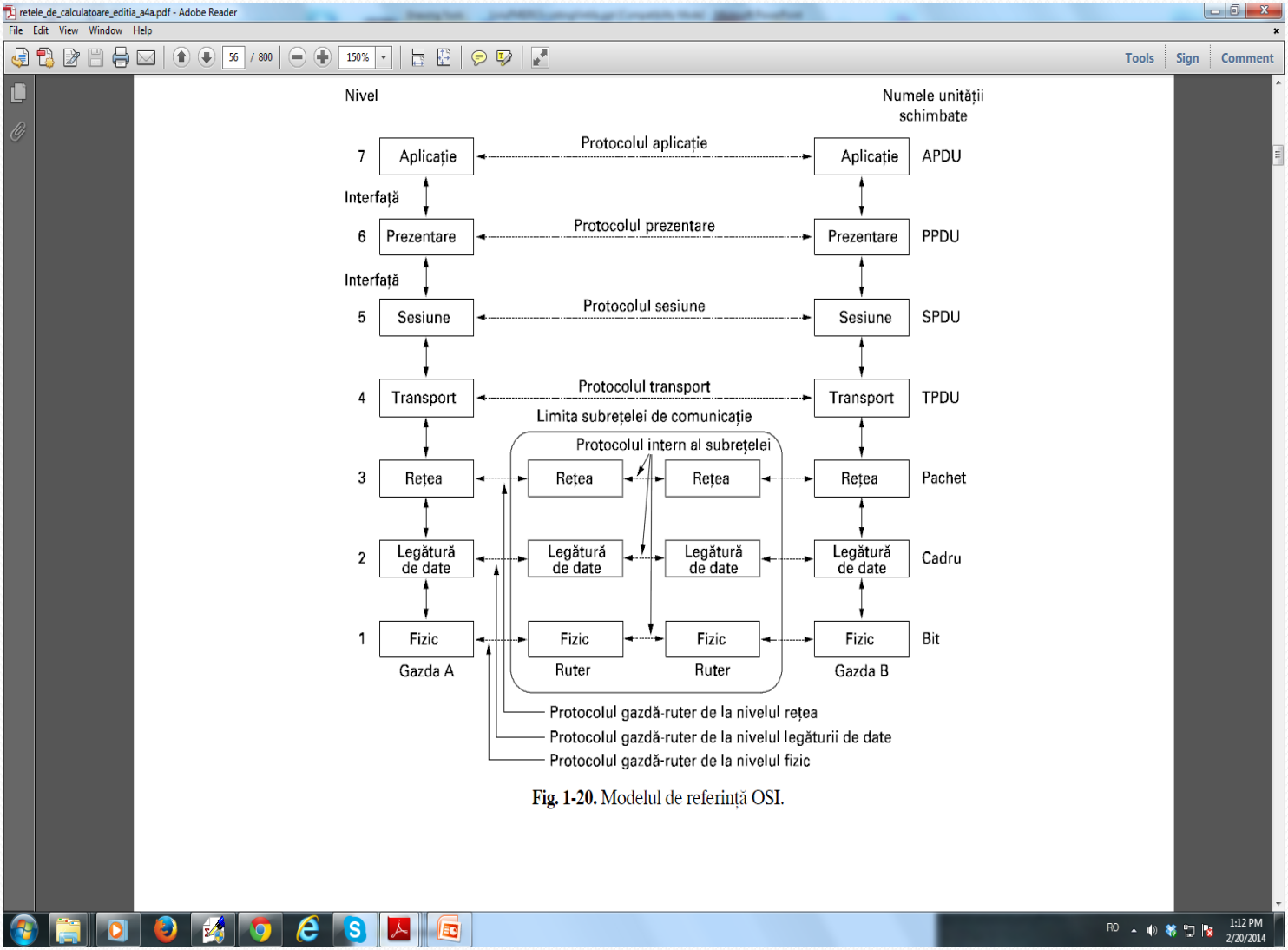


Fig. 1-20. Modelul de referință OSI.

MULȚUMESC!

Nivelul Legatura de Date (1)

Liviu P. Dinu
ldinu@fmi.unibuc.ro
Center for Computational Linguistics
nlp.unibuc.ro

Introducere

- Nivelul legătură de date se ocupă de algoritmii de obţinere a unei comunicaţii eficiente şi sigure, între două maşini adiacente la nivelul legăturii de date.

- Prin adiacenţă înţelegem că cele două maşini sunt conectate fizic printr-un canal de comunicaţie care se manifestă conceptual ca un fir (de exemplu, un cablu coaxial, o linie telefonică sau un canal de comunicaţie fără fir, de tip punct la punct).

Introducere (2)

- Calitatea esenţială a unui canal care îl face asemănător unui fir este aceea că biţii sunt livraţi în exact aceeaşi ordine în care sunt transmişi.

Probleme

- Problemă simplă?
  - maşina A pune biţii pe fir şi maşina B îi preia.
- Nimic de facut?
- Din păcate, circuitele de comunicaţie produc uneori erori.
- În plus, ele au numai o rată finită a datelor şi există o întârziere a propagării, nenulă, între momentul în care un bit este emis şi momentul în care acesta este recepţionat.

Subiect

- Aceste limitări au implicaţii importante pentru eficienţa transferului de date.
- Protocoalele utilizate pentru comunicaţie trebuie să ia în considerare toţi aceşti factori.
- Aceste protocoale reprezintă subiectul nivelului legatura de date de faţă.

ASPECTE ALE PROIECTĂRII

- Nivelul legătură de date are un număr de funcţii specifice pe care trebuie să le îndeplinească:
  - Furnizarea unei interfeţe bine-definite către nivelul reţea
  - Tratarea erorilor de transmisie
  - Reglarea fluxului cadrelor în aşa fel, încât receptorii lenţi să nu fie inundaţi de către emiţători rapizi.

ASPECTE ALE PROIECTĂRII

- Pentru a îndeplini aceste scopuri, nivelul legătură de date primeşte pachete de la nivelul reţea, pe care le încapsulează în **cadre în vederea transmiterii.**

- **Fiecare cadru conţine un antet, un câmp de** informaţie utilă pentru pachet şi încheiere.

- Gestionarea cadrelor reprezintă esența a ceea ce face nivelul legătură de date

Exemplu

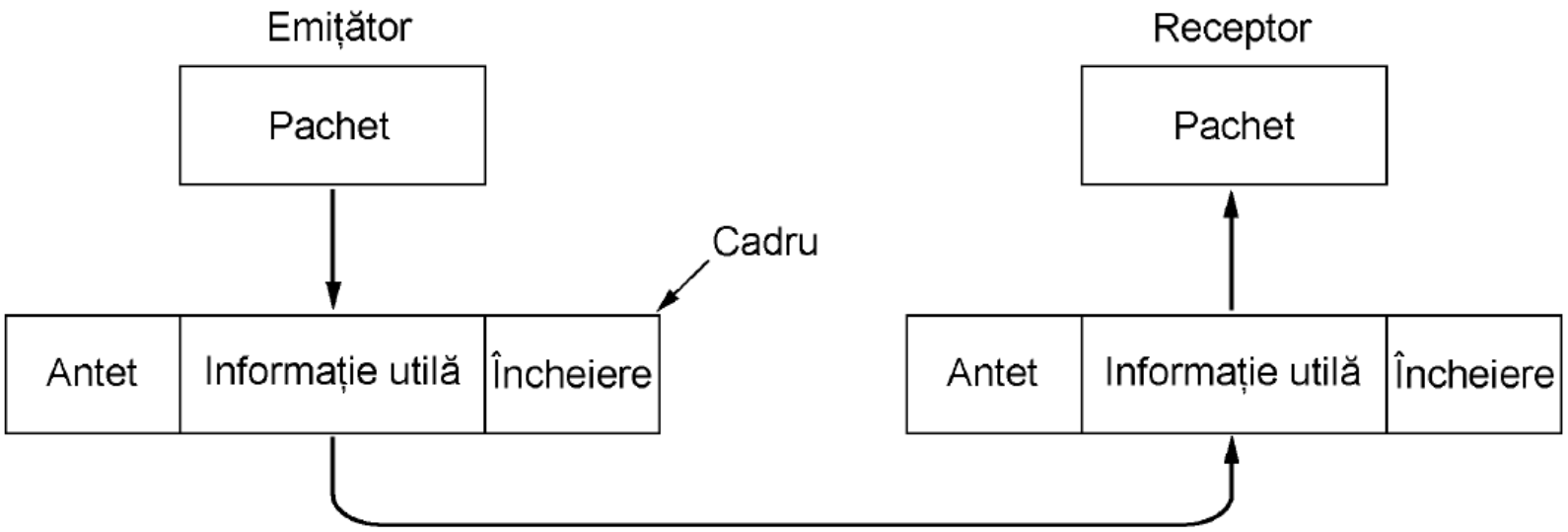


Fig. 3-1. Relația dintre pachete și cadre.

Principii

- Multe dintre principiile studiate aici (controlul erorilor, controlul fluxului) se regăsesc şi în protocoalele de transport, şi în alte protocoale.

- Indiferent de nivelul la care se găsesc, principiile sunt aproximativ aceleaşi.

- La nivelul legăturii de date ele apar de obicei în forma cea mai simplă şi cea mai pură, făcând din acest nivel un loc foarte potrivit studierii detaliate a acestor principii.

Servicii oferite nivelului reţea

- Principalul serviciu este transferul datelor de la nivelul rețea al mașinii sursă la nivelul rețea al mașinii destinație.

- La nivelul reţea al maşinii sursă există o entitate, (numita proces), care trimite biţi către nivelul legătură de date, pentru a fi transmişi la destinaţie.

- Funcţia nivelului legătură de date este să transmită biţii spre maşina destinaţie, pentru ca acolo să fie livraţi nivelului reţea

Servicii

- Nivelul legătură de date poate fi proiectat să ofere diferite servicii.
- Trei posibilităţi de bază, oferite în mod curent, sunt:
  - Serviciu neconfirmat fără conexiune.
  - Serviciu confirmat fără conexiune.
  - Serviciu confirmat orientat-conexiune.

Serviciul neconfirmat fără conexiune

- Maşina sursă trimite cadre independente către maşina destinaţie, fără ca maşina destinaţie să trebuiască să confirme primirea lor.
- Nu sunt necesare stabilirea şi desfiinţarea unei conexiuni logice.
- Dacă un cadru este pierdut datorită zgomotului de pe linie, la nivelul legătură de date nu se face nici o încercare pentru recuperarea lui.

Serviciul neconfirmat fără conexiune

- Această clasă de servicii este adecvată atunci când rata de erori este foarte scăzută, încât recuperarea este lăsată în sarcina nivelurilor superioare.
- De asemenea, este adecvată pentru traficul de timp real, cum ar fi cel de voce, unde a primi date cu întârziere este mai rău decât a primi date eronate.
- Majoritatea LAN-urilor utilizează la nivelul legăturii de date servicii neconfirmate fără conexiune.

Serviciul confirmat fără conexiune

- Nu se utilizează conexiuni, dar fiecare cadru trimis este confirmat individual.
- În acest mod, emiţătorul ştie dacă un cadru a ajuns sau nu cu bine.
- Dacă nu a ajuns într-un interval de timp specificat, poate fi trimis din nou.
- Acest serviciu este folositor pentru canale nesigure, cum ar fi sistemele fără fir.

Problema

- De obicei, cadrele au o lungime maximă impusă de hardware, iar pachetele nivelului reţea nu au aceasta limitare.
- Dacă pachetul mediu este spart în 10 cadre şi 20% din totalul cadrelor sunt pierdute, transmiterea acestuia poate lua foarte mult timp.
- În cazul în care cadrele individuale sunt confirmate şi retransmise, pachetele întregi vor fi transmise mult mai rapid.

Serviciul orientat-conexiune

- Maşinile sursă şi destinaţie stabilesc o conexiune înainte de a transfera date.
- Fiecare cadru trimis pe conexiune este numerotat şi nivelul legătură de date garantează că fiecare cadru trimis este într-adevăr recepţionat.
- Este garantat că fiecare cadru este recepţionat exact o dată şi toate cadrele sunt recepţionate în ordinea corectă

Serviciul orientat-conexiune

- In cazul serviciului fără conexiune, este posibil ca, datorită unei confirmări pierdute, un cadru să fie transmis de mai multe ori şi, prin urmare, recepţionat de mai multe ori.

- Serviciul orientat conexiune furnizează proceselor de la nivelul reţea echivalentul unui flux de biți sigur.

Faze

- Transferurile au trei faze distincte.
- În prima fază este stabilită conexiunea, ambele părţi iniţializând variabile şi contoare, utilizate pentru a ține evidența cadrelor care au fost recepționate și a celor care nu au fost.
- În a doua fază, sunt transmise unul sau mai multe cadre.
- În a treia fază, conexiunea este desfiinţată, eliberând variabilele, tampoanele și alte resurse utilizate la menținerea conexiunii.

Încadrarea

- Nivelul legătură de date trebuie să utilizeze serviciul furnizat de către nivelul fizic.
- Sarcina nivelului fizic este să primească un flux de biţi şi să încerce să-l trimită la destinaţie.

- Nu se garantează că acest flux de biţi nu conţine erori:
  - Numărul de biţi recepţionaţi poate fi mai mic, egal cu, sau mai mare decât numărul de biţi transmişi şi pot avea valori diferite.
- Este la latitudinea nivelului legătură de date să detecteze şi, dacă este necesar, să corecteze erorile.

Abordarea uzuala

- Abordarea uzuală pentru nivelul legătură de date este să spargă şirul de biţi în cadre discrete şi să calculeze suma de control pentru fiecare cadru.
- Atunci când un cadru ajunge la destinaţie, suma de control este recalculată.
- Dacă noua sumă de control este diferită de cea conţinută în cadru, nivelul legătură de date ştie că a apărut o eroare şi face operaţiile necesare pentru a o rezolva (de exemplu, elimină cadrul eronat şi probabil trimite înapoi un raport de eroare).

Spargerea

- Spargerea şirului de biţi în cadre este dificilă
- O cale pentru a realiza această încadrare este inserarea de intervale de timp între cadre, aşa cum inserăm spaţii între cuvinte într-un text normal.
- Reţelele dau rareori garanţii referitoare la timp, aşa că este posibil ca aceste intervale să fie comprimate sau ca în timpul transmisiei să fie inserate alte intervale.

Metode

- Deoarece este prea periculos să ne bizuim pe timp pentru a marca începutul şi sfârşitul fiecărui cadru, au fost elaborate alte metode:
  - Numărarea caracterelor.
  - Indicatori cu inserare de octeţi.
  - Indicatori de început şi de sfârşit, cu inserare de biţi.
  - Violarea codificărilor la nivel fizic.

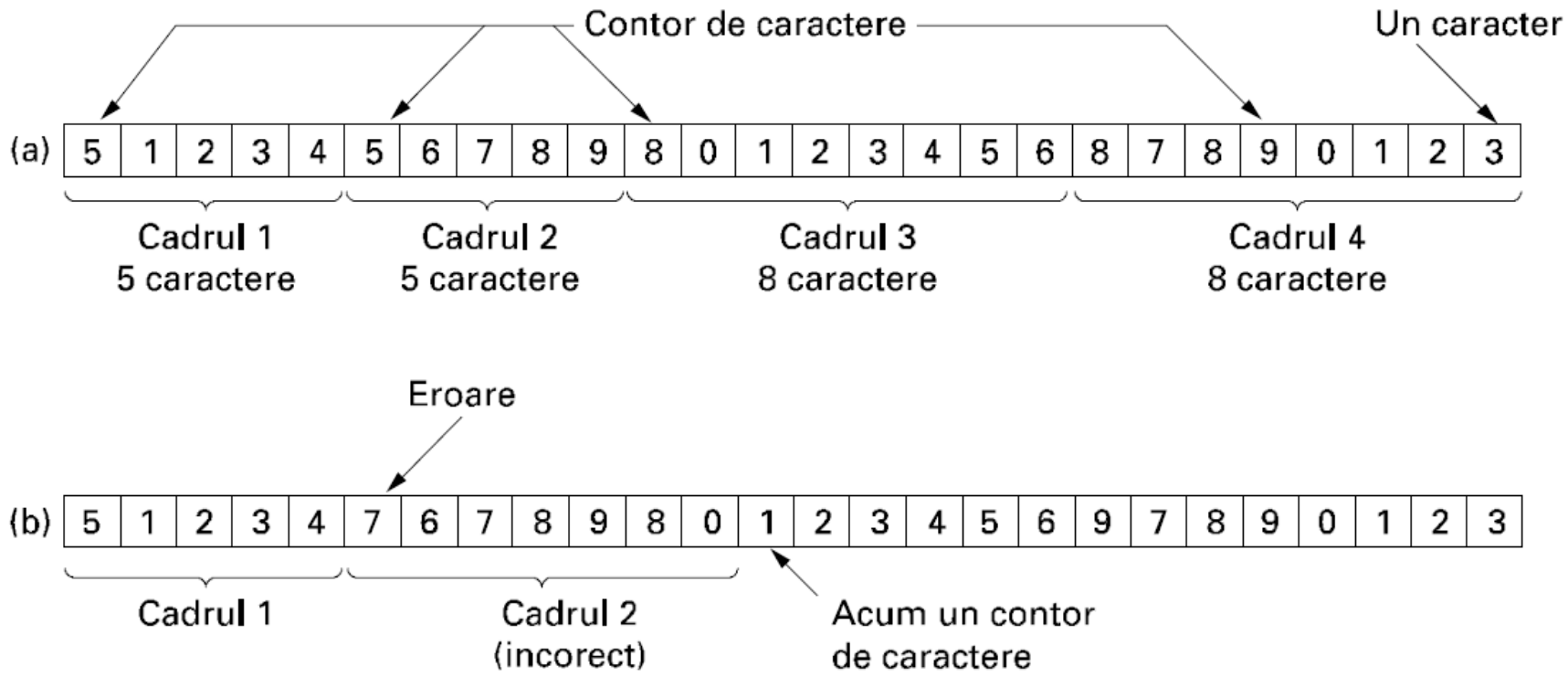
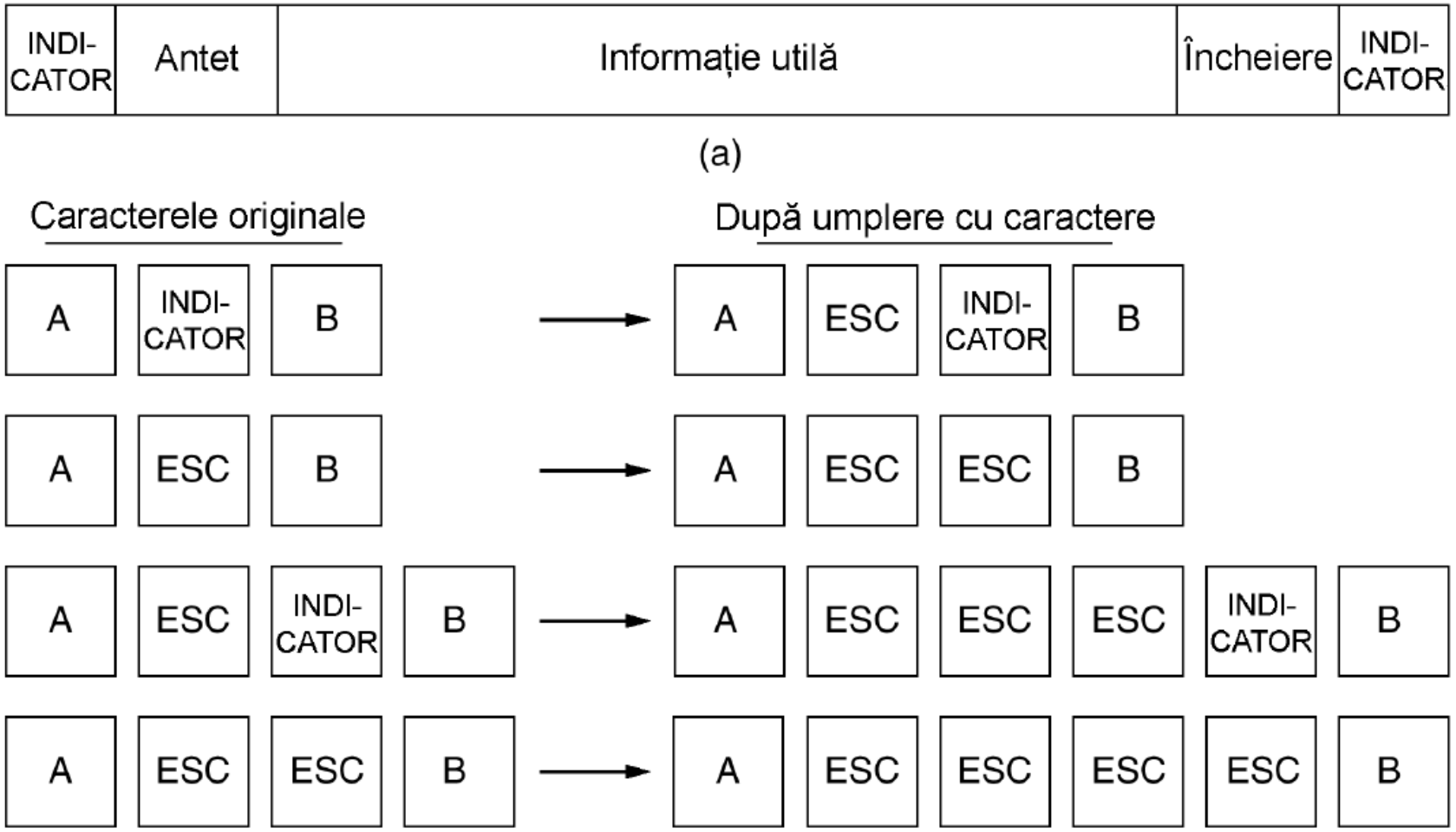


Fig. 3-4. Un șir de caractere. (a) Fără erori. (b) Cu o eroare.


INDI-CATOR
Antet
Informație utilă
Încheiere
INDI-CATOR
(a)
Caracterele originale
După umplere cu caractere
A
INDI-CATOR
B
A
ESC
INDI-CATOR
B
A
ESC
B
A
ESC
ESC
B
A
ESC
INDI-CATOR
B
A
ESC
ESC
ESC
INDI-CATOR
B
A
ESC
ESC
B
A
ESC
ESC
ESC
ESC
B

Prima metoda

- Prima metodă de încadrare utilizează un câmp din antet pentru a specifica numărul de caractere din cadru.
- Atunci când nivelul legătură de date de la destinaţie primeşte contorul de caractere, ştie câte caractere urmează şi unde este sfârşitul cadrului.
- Problema cu acest algoritm este că valoarea contorului poate fi alterată de erori de transmisie.

Byte stuffing

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

Stuffing

- Doi octeţi indicatori consecutivi indică sfârşitul unui cadru şi începutul celui care urmează.
- O problemă serioasă cu această metodă apare atunci când se transmit date binare, cum ar fi un obiect sau numere în virgulă mobilă.
- Se poate întâmpla ca în date să apară octetul folosit ca indicator.

Stuffing (2)

- Această situaţie interferează cu procesul de încadrare.
- O cale de rezolvare a acestei probleme este ca nivelul legătură de date al emiţătorului să insereze un octet special (ESC) înaintea fiecărei apariţii „accidentale” a indicatorului în date.
- Nivelul legătură de date al receptorului va elimina acest octet special înainte de a pasa datele nivelului rețea.
- Dezavantaj: este limitată la utilizarea caracterelor de 8 biți

Metoda 3: 01111110

- Permitem cadrelor de date să conţină un număr arbitrar de biţi şi permitem coduri de caractere cu un număr arbitrar de biţi per caracter.
- Funcţionează astfel: fiecare cadru începe şi se termină cu un şablon special pe biţi, 01111110, numit octet **indicator (flag).**
- **De fiecare dată când** nivelul legătură de date al emiţătorului identifică cinci de unu consecutivi în date, inserează automat un bit 0 în şirul de biţi de rezultaţi.

01111110

- Această **inserare de biţi (bit stuffing) este similară inserării de** caractere, în care un octet escape este inserat în şirul de caractere de ieşire, înainte de fiecare octet indicator din date.
- Atunci când receptorul primește o succesiune de cinci biți 1, urmați de un bit 0, extrage automat (adică şterge) bitul 0.

01111110

- La fel ca şi inserarea de caractere, care este complet transparentă pentru nivelul reţea din ambele calculatoare, aşa este şi inserarea de biţi.
- Dacă datele utilizator conţin şablonul indicator,

01111110

, acest indicator este transmis ca

011111010

dar în memoria receptorului este păstrat ca 01111110.

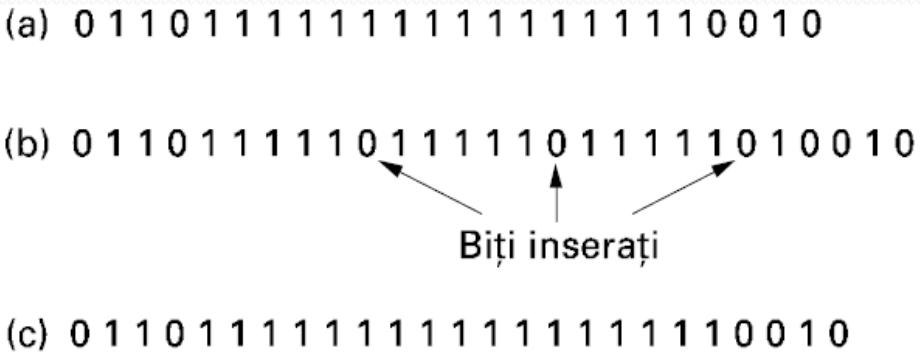


Fig. 3-6. Inserare de biți. (a) Datele originale. (b) Datele așa cum apar pe linie. (c) Datele așa cum sunt stocate în memoria receptorului după extragerea biților inserați.

MULȚUMESC!

Nivelul Legatura de Date (2)

Liviu P. Dinu

ldinu@fmi.unibuc.ro

Center for Computational Linguistics

nlp.unibuc.ro

Controlul erorilor

- Modul uzual de a asigura o transmitere sigură este de a furniza emițătorului o reacție inversă(feedback) despre ceea ce se întâmplă la celălalt capăt al liniei.
- De obicei protocolul îi cere receptorului să trimită înapoi cadre de control speciale, purtând confirmări pozitive sau negative despre cadrele sosite.

- Dacă emiţătorul recepţionează o confirmare pozitivă despre un cadru, el ştie că acel cadru a ajuns cu bine.
- o confirmare negativă înseamnă că ceva a mers prost și cadrul trebuie retransmis.
- Defectele de echipament pot determina dispariţia completă a unui cadru (de exemplu într-o rafală de zgomot

- În acest caz, receptorul nu va reacţiona în nici un fel, din moment ce nu are nici un motiv să reacţioneze.
- Trebuie să fie clar că un protocol în care emiţătorul trimite un cadru şi apoi aşteaptă o confirmare, pozitivă sau negativă, va rămâne agăţat pentru totdeauna dacă un cadru este complet pierdut datorită, de exemplu, nefuncţionării echipamentului.

Controlul fluxului

- Un alt aspect important de proiectare care apare la nivelul legătură de date este cum trebuie procedat cu un emiţător care doreşte în mod sistematic să transmită cadre mai repede decât poate să accepte receptorul.
- Această situaţie poate să apară uşor atunci când emiţătorul rulează pe un calculator rapid (sau mai puţin încărcat) şi receptorul rulează pe o maşină lentă (sau foarte încărcată).

Flux

- Emiţătorul continuă să transmită cadre la o rată înaltă până când receptorul este complet inundat.
- Chiar dacă transmisia este fără erori, la un anumit punct receptorul nu va mai fi capabil să trateze cadrele care sosesc şi va începe să piardă unele dintre ele.

- Două abordări des utilizate.
  - In cazul celei dintâi, **controlul fluxului bazat pe reacţie (feedback-based flow control), receptorul acordă emiţătorului permisiunea de a mai transmite date, sau** cel puţin comunică emiţătorului informaţii despre starea sa.
  - În cea de-a doua, **controlul fluxului bazat pe rată (rate-based flow control), protocolul dispune de un mecanism integrat care limitează rata** la care emiţătorul poate transmite, fără a folosi informaţii de la receptor. (nefolosita la acest nivel)

DETECTAREA ŞI CORECTAREA ERORILOR

- Erorile din unele medii (de exemplu radio) tind să vină mai curând în rafale decât izolate.
- Avantajul este acela că datele de la calculator sunt trimise întotdeauna în blocuri de biți.
- Dezavantajul erorilor în rafală este acela că sunt mult mai greu de detectat şi corectat decât erorile izolate.

Coduri corectoare de erori

- Două strategii de bază pentru tratarea erorilor.
- O modalitate este ca pe lângă fiecare bloc de date trimis să se includă suficientă informaţie redundantă pentru ca receptorul să poată deduce care a fost caracterul transmis.
- O altă soluţie este să se includă suficientă redundanţă pentru a permite receptorului să constate că a apărut o eroare, dar nu care este eroarea, şi să ceară o retransmisie.
- Prima strategie utilizează **coduri corectoare de erori, iar cea de-a doua** utilizează **coduri detectoare de erori.**

Coduri corectoare de erori

- **Distanta Hamming între doi vectori de dimensiuni egale este data de numarul de pozitii în care acestia difera.**
- **Ea masoara astfel numarul de schimbari care trebuie facute într-un vector pentru a îl** obtine pe celalalt, sau reformulat numarul de *erori care transforma un vector în celalalt*.

Exemple:

vector 1	codare	126359	01101011
vector 2	notate	226389	01001110
distanta Hamming	3	2	3

- Sa consideram urmatoarea solutie simpla de codare – fiecare bit este codat prin repetarea sa de doua ori.
- Introducem astfel în mod evident o redundanta care însa ne va permite sa detectam erori singulare.
- Daca la receptie obtinem coduri 00 respectiv 11 am receptionat corect 0 respectiv 1 iar daca obtinem 01 sau 10 am detectat o eroare singulara (fara a putea decide însa nimic în sensul corectiei ei).

mesaj initial:	0.1.0.0.1.0.1.1.0.1
mesaj codat:	00.11.00.00.11.00.11.11.00.11
mesaj receptionat cu eroare:	00.11.00.00.**10**.00.11.11.00.11 => **detectie** de eroare

Corectie de erori singulare

- Solutie simpla de codare – fiecare bit este codat prin repetarea sa de trei ori.
- Introducem astfel în mod evident o redundanta si mai mare care însa ne va permite sa detectam si sa corectam erori singulare.
- Daca la receptie obtinem coduri 000 respectiv 111 am receptionat corect 0 respectiv 1.

Corectie de erori singulare (2)

- Daca obtinem 001 sau 010 sau 100 am detectat o eroare singulara si putem **corecta spre 000 (deci am** receptionat 0)
- Daca obtinem 011 sau 101 sau 110 am detectat o eroare singulara si putem **corecta** spre 111 (deci am receptionat 1).

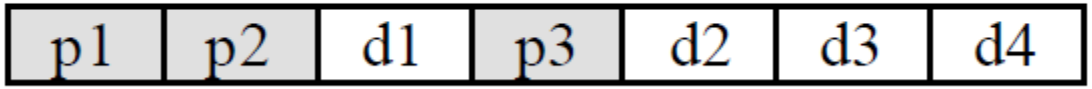
mesaj initial:	0.1.0.0.1.0.1.1.0.1
mesaj codat:	000.111.000.000.111.000.111.111.000.111
mesaj receptionat cu eroare:	000.**101**.000.000.111.000.111.111.**001**.111
mesaj corectat:	000.**111**.000.000.111.000.111.111.**000**.111

=> **detectie** si corectie de erori singulare

Codul Hamming (7,4)

- Codul Hamming (7,4) indica faptul ca avem un cod de 7 biti din care 4 sunt biti de date independenti (restul fiind biti redundanti, reprezentând paritatea a diferite combinatii a bitilor de date).
- Codul contine 4 biti de date d1, d2, d3, d4 si 3 biti de paritate p1, p2, p3.
- Bitii de paritate sunt calculati astfel:
  - p1 sumeaza d1, d2, d4
  - p2 sumeaza d1, d3, d4
  - p3 sumeaza d2, d3, d4

organizarea bitilor în cuvântul de cod este urmatoarea:



Exemplu

- **La codare: fie cuvântul de cod 1001. Calculam:**
- p1 = 1+0+1 = 0
- p2 = 1+0+1 = 0
- p3 = 0+0+1 = 1
- rezultând codul Hamming **0 0 1 1 0 0 1 (am reprezentat cu rosu bitii de paritate).**

Exemplu (2)

- **La decodare (cazul 1): presupunem ca am receptionat 0011001 (deci fara erori)**
- Reconstituim bitii de date: 1 0 0 1
- Verificam paritatile:
  - p1 + d1 + d2 + d4 = 0 + 1 + 0 + 1 = 0
  - p2 + d1 + d3 + d4 = 0 + 1 + 0 + 1 = 0
  - p3 + d2 + d3 + d4 = 1 + 0 + 0 + 1 = 0
- Cum toate aceste valori sunt 0 rezulta ca nu am avut eroare.

Exemplu

- **La decodare (cazul 2): presupunem ca am receptionat 0011011 (deci cu o eroare în zona de date)**
- Reconstituim bitii de date: 1 0 1 1
- Verificam paritatile:
  - p1 + d1 + d2 + d4 = 0 + 1 + 0 + 1 = 0
  - p2 + d1 + d3 + d4 = 0 + 1 + 1 + 1 = 1
  - p3 + d2 + d3 + d4 = 1 + 0 + 1 + 1 = 1
- Cum aceste valori nu sunt toate 0 rezulta ca am avut eroare (deci am realizat **detectie de eroare**).

Exemplu

- **În** plus codul format din pozitiile eronate (aferent p3p2p1) având valoarea 110 indica bitul 6 ca fiind eronat deci al treilea bit de date trebuie corectat (deci am realizat **corectie de eroare**).
- **Deci, bitii de** date corectati sunt 1 0 0 1.

Exemplu

- **La decodare (cazul 3): presupunem ca am receptionat 0111001 (deci cu 0 eroare în zona de** paritate)
- Reconstituim bitii de date: 1 0 0 1
- Verificam paritatile:
  - p1 + d1 + d2 + d4 = 0 + 1 + 0 + 1 = 0
  - p2 + d1 + d3 + d4 = 1 + 1 + 0 + 1 = 1
  - p3 + d2 + d3 + d4 = 1 + 0 + 0 + 1 = 0
- Cum aceste valori nu sunt toate 0 rezulta ca am avut eroare (deci am realizat **detectie de eroare:** codul format din pozitiile eronate (aferent p3p2p1) având valoarea 010 indica bitul 2 ca fiind eronat

Coduri Hamming generale

- Codul Hamming descris anterior a fost generalizat pentru orice numar de biti
- Algoritmul general pentru proiectarea unui cod Hamming **corector de erori singulare:**
  - Numerotam bitii începând cu 1: 1, 2, 3, 4, etc.
  - Fiecare pozitie care este putere a lui 2 reprezinta **bit de paritate** (**pozitiile care au un singur** bit în reprezentarea binara)

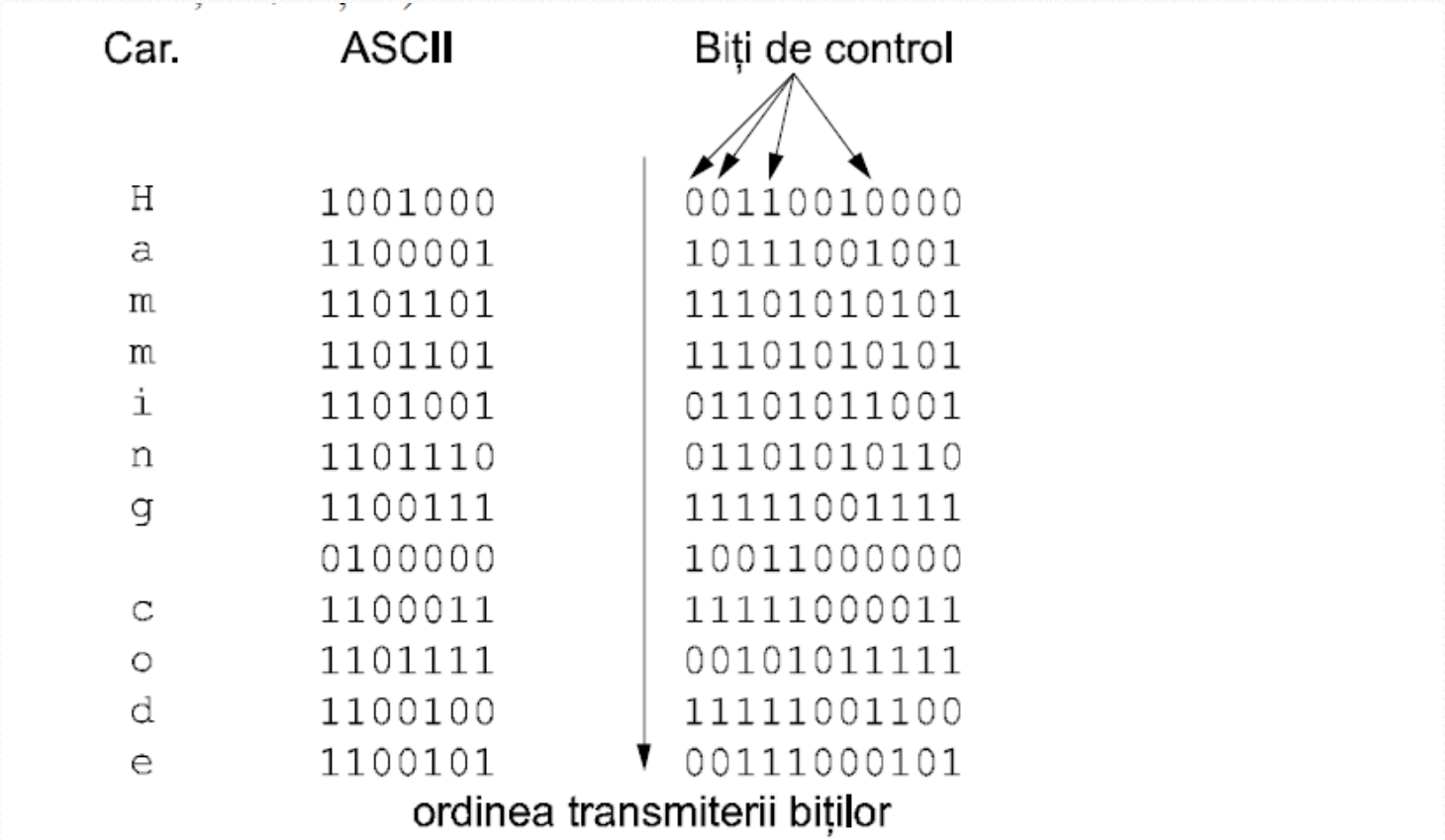
- Celelalte pozitii reprezinta **biti de date (pozitiile care au mai mult de un bit pe 1 în** reprezentarea binara).
- Fiecare bit de paritate acopera biti astfel: bitul k de paritate acopera acele pozitii care au bitul k cel mai semnificativ pe 1.

- Desi modul de calcul al paritatii (pare sau impare) nu este esential, de regula însa se foloseste paritatea impara (paritatea e 1 daca numarul de biti de 1 este impar).

Algoritmul general poate fi înteles mai bine din figura urmatoare:

pozitie		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
cod		p1	p2	d1	p3	d2	d3	d4	p4	d5	d6	d7	d8	d9	d10	d11	p5	d12	d13	...
componenta biti paritate	p1	X		X		X		X		X		X		X		X		X		...
	p2		X	X			X	X			X	X			X	X			X	...
	p3				X	X	X	X					X	X	X	X				...
	p4								X	X	X	X	X	X	X	X				...
	p5																X	X	X	...

S-au reprezentat doar 5 biti de paritate si 13 biti de date având de a face cu codul care se noteaza Hamming(18,13).



Utilizarea unui cod Hamming pentru corectarea erorilor în rafală.

Coduri detectoare de erori

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

Coduri detectoare de erori

- Metoda codului polinomial (CRC) presupune ca emițătorul și receptorul se pun de acord în avans asupra unui **polinom generator *G(x)*.**
- ***Atât bitul cel mai semnificativ cât și cel mai puțin semnificativ*** trebuie să fie 1.
- Pentru a calcula **suma de control pentru un cadru cu *m biți, corespunzător*** polinomului *M(x), cadrul trebuie să fie mai lung decât polinomul generator.*

CRC (2)

- Ideea este de a adăuga o sumă de control la sfârşitul cadrului, astfel încât polinomul reprezentat de cadrul cu sumă de control să fie divizibil prin G(x).
- Când receptorul preia cadrul cu suma de control, încearcă să-l împartă la G(x).
- Dacă se obţine un rest, înseamnă că a avut loc o eroare de transmisie.

Algoritm

- Algoritmul pentru calculul sumei de control:

- Fie *r gradul lui G(x). Se adaugă r biţi 0 la capătul mai puţin semnificativ al cadrului, aşa încât* acesta va conţine acum *n+r biți și va corespunde polinomului x^rM(x).*
- Se împarte șirul de biți ce corespund lui *G(x) într-un șir de biți corespunzând lui x^rM(x), utilizând* împărţirea modulo 2.

Algoritm (2)

- Se scade restul (care are întotdeauna r sau mai puțini biți) din șirul de biți corespunzând lui x^rM(x), utilizând scăderea modulo 2.
- Rezultatul este cadrul cu sumă de control ce va fi transmis.
- Numim polinomul său *T(x)*.

Exemplu

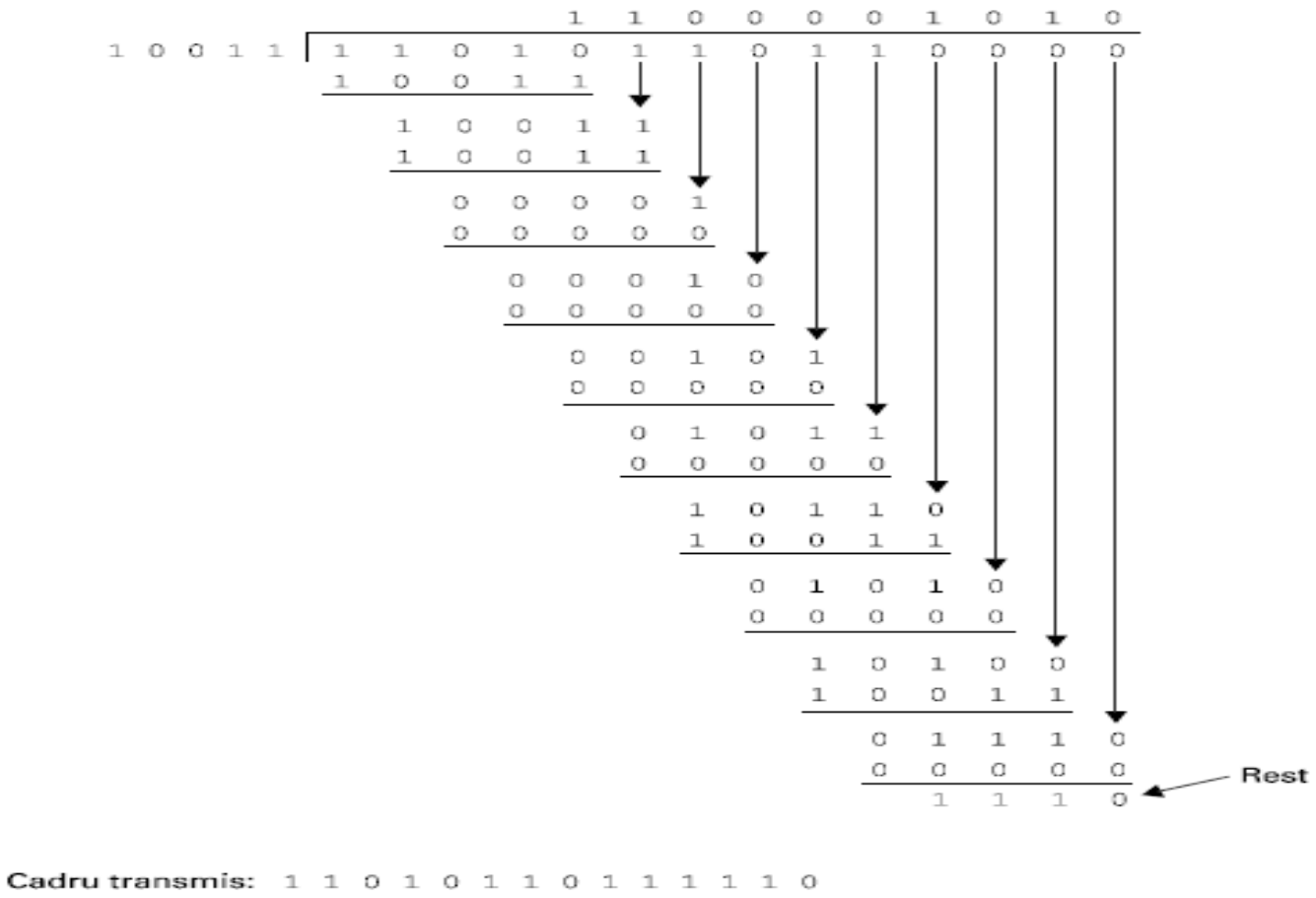


Fig. 3-8. Calculul sumei de control în cod polinomial.

MULȚUMESC!

Nivelul Legatura de date (3)

Liviu P. Dinu
ldinu@fmi.unibuc.ro
Center for Computational Linguistics
nlp.unibuc.ro

PROTOCOALE ELEMENTARE PENTRU LEGĂTURA DE DATE

- Ipotezele care stau la baza modelelor de comunicaţie:
- Pentru început, considerăm că la nivelul fizic, nivelul legătură de date şi nivelul reţea există procese independente care comunică transferându-şi mesaje în ambele sensuri.
- În multe cazuri, procesele de la nivelul fizic şi de legătură de date se vor executa pe un procesor dintr-un cip special de intrare/ieşire al reţelei, iar codul nivelului reţea în CPU.

Ipoteze

- O altă presupunere cheie este că maşina *A vrea să trimită un lung şir de date maşinii B, folosind* un serviciu sigur, orientat pe conexiune.
- *Presupunem că A are tot timpul date gata de transmis şi nu aşteaptă* niciodată ca aceste date să fie produse.
- Atunci când nivelul legătură de date al lui *A* cere date, nivelul reţea este totdeauna capabil să i le furnizeze imediat. (restricţie ce va fi abandonată mai târziu.)

Ipoteze (2)

- Presupunem de asemenea că maşinile nu se defectează. Adică, aceste protocoale tratează erorile de transmisie, dar nu şi erorile generate de defectarea sau reiniţializarea calculatoarelor.
- În ceea ce priveşte nivelul legătură de date, pachetul care trece de la nivelul reţea, prin interfaţă, către el este constituit din date pure, fiecare bit al acestora trebuind să fie trimis la nivelul reţea destinaţie.
- Faptul că nivelul reţea destinaţie poate interpreta o parte din pachetul de date ca antet nu prezintă interes pentru nivelul legătură de date.

Ipoteze

- Atunci când acceptă un pachet, nivelul legătură de date îl încapsulează într-un cadru, adăugându-i un antet şi o încheiere de legătură de date.
- Deci un cadru se compune dintr-un pachet de date, câteva informaţii de control (antetul), și suma de control (în încheiere)
- Apoi cadrul este transmis către alt nivel legătură de date.

Ipoteze

- Vom presupune că există proceduri de bibliotecă adecvate pentru transmitea şi recepţionarea unui cadru:
  - *to_physical_layer şi, respectiv,*
  - *from_physical_layer.*
- *Echipamentul de transmisie calculează şi adaugă suma de control (creând astfel* o încheiere), astfel încât programele nivelului legătură de date nu trebuie să se preocupe de aceasta.

Descriere

- Iniţial, receptorul nu are nimic de făcut. Doar stă aşteptând să se întâmple ceva
- Nivelul legătură de date aşteaptă să se producă un eveniment prin apelul de procedură *wait_for_event (&event).*
- *Această procedură redă controlul numai atunci* când s-a întâmplat ceva (adică atunci când soseşte un cadru).
- La revenire, variabila *event spune ce s-a* întâmplat.

Descriere

- Când un cadru ajunge la receptor, echipamentul calculează suma de control. Dacă aceasta este incorectă (în cazul unei erori de transmisie), atunci nivelul legătură de date este informat corespunzător:

(*event = cksum_err).*

- *Dacă un cadru ajunge nealterat, nivelul legătură de date este de asemenea* informat

event = frame_arrival,

aşa că poate primi cadrul pentru inspecţie folosind

from_physical_layer.

Descriere

- *De îndată ce nivelul de legătură de date a primit un cadru nealterat, verifică* informaţiile de control din antet şi dacă totul este în regulă, pachetul este transmis nivelului reţea.
- În nici un caz antetul nu este transmis nivelului reţea.

Definiţii necesare în protocoale

```
#define MAX_PKT 1024                    /* determină dimensiunea în octeţi a pachetului */

typedef enum {false, true} boolean;                              /* tip boolean */
typedef unsigned int seq_nr;                          /* numere de secvenţă sau de ack */
typedef struct {unsigned char data[MAX_PKT];} packet;          /* definiţia pachetului */
typedef enum {data, ack, nak} frame_kind;              /* definiţia tipurilor de cadre */

typedef struct {                                /* la acest nivel sunt transportate cadre */
 frame_kind kind;                                    /* ce fel de cadru este acesta? */
 seq_nr seq;                                                 /* număr de secvenţă */
 seq_nr ack;                                                /* număr de confirmare */
 packet info;                                          /* pachetul de nivel reţea */
} frame;

/* Aşteaptă producerea unui eveniment; întoarce tipul acestuia în variabila event */
void wait_for_event(event_type *event);

/* Preia un pachet de la nivelul reţea, spre a-l transmite prin canal */
void from_network_layer(packet *p);

/* Livrează nivelului reţea informaţia dintr-un cadru ajuns la destinaţie */
void to_network_layer(packet *p);

/* Preia cadrul sosit de la nivelul fizic şi îl copiază în r. */
void from_physical_layer(frame *r);

/* Livrează cadrul nivelului fizic, pentru transmisie */
void to_physical_layer(frame *s);

/* Porneşte ceasul şi activează evenimentul timeout */
void start_timer(seq_nr k);

/* Opreşte ceasul şi dezactivează evenimentul timeout */
void stop_timer(seq_nr k);

/* Porneşte un ceas auxiliar şi activează evenimentul ack_timeout */
void start_ack_timer(void);

/* Opreşte ceasul auxiliar şi dezactivează evenimentul ack_timeout */
void stop_ack_timer(void);

/* Permite nivelului reţea să provoace un eveniment network_layer_ready */
void enable_network_layer(void);

/* Interzice nivelului reţea generarea unui eveniment network_layer_ready */
void disable_network_layer(void);

/* Macroinstrucţiunea inc este expandată în-line: îl incrementează circular pe k */
#define inc(k) if (k ( MAX_SEQ) k = k + 1; else k = 0
```

Definitii

- 5 structuri de date: *boolean, seq_nr, packet, frame_kind, frame.*
- *Un boolean este de* tip enumerativ şi poate lua numai valorile *true sau false.*
- *Seq_nr este un număr întreg mic folosit pentru* a numerota cadrele, astfel încât să le putem identifica.
- Aceste numere de secvenţă sunt cuprinse între 0 şi *MAX_SEQ inclusiv, aceasta din urmă fiind o constantă definită în fiecare protocol în care* este necesară

Definitii (2)

- Un *pachet este unitatea de informaţie schimbată între nivelul reţea şi nivelul legătură* de date de pe aceeaşi maşină sau între niveluri reţea similare.
- În modelul nostru el conţine totdeauna *MAX_PKT octeţi, dar mai realist ar fi să aibă lungime variabilă.*
- Un *frame (cadru) este compus din patru câmpuri:*

kind, seq, ack, și info,

Primele trei conțin informații de control, iar ultimul poate conține datele efective care trebuie transferate. Ansamblul acestor câmpuri de control este numit **antetul cadrului (frame header).**

Definitii

- Câmpul *kind (tip) spune dacă există sau nu date în cadru, deoarece unele protocoale fac distincţie* între cadrele care conţin exclusiv informaţii de control şi cele care conţin şi date.
- Câmpurile *seq şi ack sunt utilizate pentru numere de secvenţă şi, respectiv, confirmări (acknowledgements);*
- Câmpul *info al unui cadru de date conţine un singur pachet de* date; câmpul *info al unui cadru de control nu este utilizat.*

Rutine de biblioteca

- Procedura *wait_for_event ciclează, aşteptând să se întâmple ceva*.
- Procedurile *to_network_layer şi from_network_layer sunt utilizate de nivelul de legătură de date* pentru a trimite, respectiv a accepta, pachete de la nivelul de reţea. *from_physical_layer* şi *to_physical_layer sunt utilizate pentru trimiterea cadrelor între nivelurile fizic şi legătură de date.*
- *to_network_layer şi from_network_layer sunt folosite pentru a trimite pachetele între* nivelul legătură de date şi nivelul reţea.

Timp

- În cele mai multe dintre protocoale se consideră că se utilizează un canal nesigur care, ocazional,poate pierde cadre întregi.
- Pentru a contracara efectul unor asemenea calamităţi, nivelul legătură de date care transmite trebuie să pornească un contor de timp sau un ceas intern de fiecare dată când trimite un cadru.
- Dacă nu s-a primit un răspuns într-un interval de timp predefinit, la expirarea acestuia, nivelul legătură de date primeşte un semnal de întrerupere.

Timp (2)

- Acest lucru este asigurat de procedura *wait_for_event care întoarce event = timeout.*
- *Procedurile start_timer și stop_ timer sunt utilizate pentru a porni, respectiv a opri contorul* de timp.
- Expirarea timpului este posibilă numai atunci când contorul de timp este pornit.
- Este permis explicit să se apeleze *start_ timer în timp ce contorul de timp lucrează;*

Timp (3)

- *Un astfel de apel va* reseta pur şi simplu contorul de timp, determinându-l să genereze următorul semnal de expirare de timp după ce întregul interval de timp se va epuiza (exceptând cazul în care este resetat sau dezactivat în acest interval timp

Timp (4)

- Procedurile *start_ack_timer şi stop_ack_timer sunt folosite pentru a controla un contor de timp* auxiliar, utilizat pentru a genera confirmări în anumite condiţii.
- Procedurile *enable_network_layer şi disable_network_layer sunt utilizate în protocoalele mai sofisticate, în care nu mai presupunem că* nivelul de reţea are tot timpul pachete de trimis

- Când nivelul legătură de date activează nivelul reţea, acestuia i se permite să întrerupă atunci când are un pachet de trimis.
- Indicăm aceasta cu *event = network_layer_ready. Când un nivel reţea este dezactivat, acesta nu poate provoca asemenea* evenimente.
- Gestionând cu prudenţă activarea şi dezactivarea nivelului reţea, nivelul legătură de date îl poate împiedica pe acesta să-l inunde cu pachete atunci când nu mai are spaţiu în tampon.

Protocol simplex fără restricţii

- Cel mai simplu protocol posibil.
- Datele sunt transmise într-o singură direcţie.
- Cele două niveluri reţea, de transmisie şi de recepţie, sunt considerate tot timpul pregătite.
- Timpul de prelucrare poate fi ignorat.
- Memoria de stocare disponibilă este infinită.
- Canalul de comunicaţie între niveluri legătură de date nu pierde şi nu alterează niciodată cadrele.

Utopia

- Protocolul constă din două proceduri distincte, una de emisie şi cealaltă de recepţie.
- Emiţătorul lucrează la nivelul legătură de date al maşinii sursă, iar receptorul la nivelul legătură de date al maşinii de destinaţie.
- Nu se folosesc nici numere de secvenţă, nici confirmări, aşa că nu este nevoie de *MAX_SEQ*.
- *Singurul eveniment posibil este frame_arrival (sosirea unui cadru nealterat).*

Utopia. Descriere

- Emiţătorul este într-un ciclu infinit care doar inserează datele pe linie cât poate de repede.
- Ciclul constă din trei acţiuni:
  - preluarea unui pachet de date de la nivelul reţea (care este întotdeauna serviabil),
  - construirea unui cadru de ieşire folosind variabila *s*
  - *trimiterea cadrului pe drumul său.*

Utopia. Descriere

- *Acest* protocol utilizează numai câmpul *info al cadrului*, *deoarece celelalte câmpuri se referă la erori şi* secvenţe de control, iar în cazul nostru nu există erori sau restricţii de control.
- Receptorul: inițial așteaptă să se întâmple ceva, singura posibilitate fiind sosirea unui cadru nealterat.
- În cele din urmă, cadrul ajunge, iar procedura *wait_for_event*

se întoarce cu *event setat la frame_arrival (care este oricum ignorat).*

Apelul rutinei from_physical_layer mută cadrul nou sosit din zona tampon a echipamentului în variabila *r.*

Utopia

```
/* Protocolul 1 (utopia) asigură transmitere de date doar într-o direcţie, de la transmiţă-
tor la receptor. Canalul de comunicaţie se presupune a fi fără erori, iar despre receptor
se presupune că este capabil să prelucreze infinit de repede tot ce primeşte de la intrare.
Deci, transmiţătorul nu face decât să stea într-o buclă, pompând date pe linie cât de repe-
de poate */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
 frame s;                                        /* tampon pentru cadrul transmis */
 packet buffer;                                  /* tampon pentru pachetul transmis */

 while (true) {
    from_network_layer(&buffer);                 /* preia ceva de transmis */
    s.info = buffer;                             /* îl copiază în s pentru transmitere */
    to_physical_layer(&s);                       /* îl trimite pe traseu */
 }
/* Tomorrow, and tomorrow, and tomorrow

Creeps in this petty pace from day to day

To the last syllable of recorded time

- Macbeth, V, v */
 }

void receiver1(void)
{
 frame r;
 event_type event;                  /* completat de rutina wait, dar neutilizat aici */

 while (true) {
    wait_for_event(&event);         /* singura posibilitate este sosirea unui cadru */
    from_physical_layer(&r);                     /* preia cadrul sosit */
    to_network_layer(&r.info);                   /* predă datele nivelului reţea */
 }
}
```

Fig. 3-10. Un protocol simplex fără restricţii.

Protocol Stop-and-Wait (pas-cu-pas)

- Renunţam la cea mai nerealistă restricţie utilizată în protocolul 1: posibilitatea ca nivelul reţea receptor să prelucreze datele de intrare cu viteză infinită (sau echivalent, prezenţa în nivelul legăturii de date receptor a unui tampon infinit în care să fie memorate, cât timp îşi aşteaptă rândul, toate cadrele sosite).
- Presupunem în continuare că nu se produc erori pe canalul de comunicaţie şi că traficul de date este încă simplex.

Stop and wait

- Principala problemă pe care trebuie să o rezolvăm aici este cum să evităm ca emiţătorul să inunde receptorul cu date care sosesc mai rapid decât poate acesta să prelucreze.
- În esenţă, dacă receptorul are nevoie de un timp Δt ca să execute

  from_physical_layer şi to_network_layer,

  atunci emiţătorul trebuie să transmită la o viteză medie mai mică de un cadru la fiecare interval de timp de Δt

Stop and wait

- O soluţie mult mai generală este ca receptorul să furnizeze o reacţie către emiţător.
- După trimiterea unui pachet către nivelul său reţea, receptorul trimite un mic cadru fictiv către emiţător care, de fapt, îi dă emiţătorului permisiunea să transmită următorul cadru.
- După ce a transmis un cadru, emiţătorul este obligat de protocol să intre în aşteptare un timp, până când soseşte micul cadru fictiv (deci confirmarea).

stop-and-wait

- Utilizarea reacţiei de la receptor pentru a anunţa emiţătorul că poate trimite date este un exemplu de control al fluxului.
- Protocoalele în care emiţătorul trimite un cadru şi apoi, înainte de a continua, aşteaptă o confirmare, se numesc **stop-and-wait (pas-cu-pas).**

```
/* Protocolul 2 (stop-and-wait) asigură la rândul său un flux de date unidirecțional, de la
emițător la receptor. Despre canalul de comunicație se presupune din nou că este fără
erori, ca și în protocolul 1. Totuși, de data aceasta, receptorul are doar un tampon de ca-
pacitate limitată și o viteză de prelucrare finită, așa că protocolul trebuie să împiedice
în mod explicit emițătorul să inunde receptorul cu date mai repede decât le poate trata
acesta. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
 frame s;                                        /* tampon pentru cadrul trimis */
 packet buffer;                                  /* tampon pentru pachetul trimis */
 event_type event;                       /* singura posibilitate este sosirea unui cadru */

 while (true) {
   from_network_layer(&buffer);                          /* preia ceva de transmis */
   s.info = buffer;                          /* îl copiază în s pentru transmitere */
   to_physical_layer(&s);                                /* la revedere, micuț cadru */
   wait_for_event(&event);                  /* nu continuă până nu primește semnalul */
 }
}

void receiver2(void)
{
 frame r, s;                                             /* zone tampon pentru cadre */
 event_type event;                       /* singura posibilitate este sosirea unui cadru */

 while (true) {
   wait_for_event(&event);               /* singura posibilitate este sosirea unui cadru */
   from_physical_layer(&r);                                /* preia cadrul sosit */
   to_network_layer(&r.info);                        /* livrează datele nivelului rețea */
   to_physical_layer(&s);            /* trimite un cadru fictiv pentru a trezi emițătorul */
 }
}
```

Fig. 3-11. Un protocol simplex stop-and-wait.

MULȚUMESC!

Nivelul Legatura de date (4)

Liviu P. Dinu

ldinu@fmi.unibuc.ro

Center for Computational Linguistics

nlp.unibuc.ro

Protocol simplex pentru un canal cu zgomote

- Considerăm situaţia normală a unui canal de comunicaţie care pot apărea erori.
- Cadrele pot fi fie modificate, fie complet pierdute.
- Dacă un cadru a fost modificat în tranzit, echipamentul receptor va detecta acest lucru atunci când calculează suma de control.

Eroarea?

- O variantă a protocolului 2 va funcţiona?
- adăugarea unui contor de timp (ceas).
- emiţătorul poate trimite un cadru, dar receptorul va trimite un cadru de confirmare numai dacă informaţia a fost recepţionată corect.
- dacă la receptor ajunge un cadru modificat, el va fi eliminat.
- după un timp, emiţătorul va ieşi din aşteptare şi va retrimite cadrul.
- Acest proces va fi repetat până când cadrul va ajunge în final intact

Eroarea!

- Nivelul reţea de pe *B nu are nici o posibilitate să ştie că un pachet a fost pierdut sau duplicat, aşa* că nivelul legătură de date trebuie să garanteze că nici o combinaţie de erori de transmisie, indiferent cât de puţin probabile, nu poate produce un pachet duplicat care să fie transmis nivelului reţea.

Scenariu

- Nivelul reţea de pe *A trimite pachetul 1 către nivelul său legătură de date.*
- *Pachetul este corect* recepționat de *B și este trimis nivelului rețea de pe B.*
- *B trimite un cadru de confirmare* înapoi lui *A*.
- Cadrul de confirmare s-a pierdut complet.
- El nu va mai ajunge deloc. (canalul altereazaa sau pierde şi cadre de control)

Scenariu

- Nivelul de legătură de date de pe *A aşteaptă expirarea timpului limită.*
- *Nerecepţionând o confirmare,* el presupune (incorect) că acel cadru de date a fost modificat sau pierdut şi trimite încă o dată cadrul conţinând pachetul 1.
- Cadrul duplicat ajunge şi el cu bine la nivelul legătură de date *B şi este trimis nivelului reţea de* acolo.
- Dacă *A trimite un fişier lui B, o porţiune de fişier va fi duplicată (adică, copia fişierului făcută* de *B va fi incorectă şi eroarea nu va fi detectată).*
- *Cu alte cuvinte, protocolul va eşua.*

Solutie

- Este necesară o soluţie ca receptorul să poată distinge un cadru pe care îl vede pentru prima dată de o retransmisie.
- Soluţia evidentă este aceea ca emiţătorul să pună un număr de secvenţă în antetul fiecărui cadru pe care îl trimite.
- Apoi receptorul poate verifica numărul de secvenţă al fiecărui cadru sosit pentru a vedea dacă este un cadru nou sau un duplicat ce trebuie eliminat.

Minim de biti

- Care este numărul minim de biţi necesari pentru numărul de secvenţă?
- Singura ambiguitate în acest protocol este între un cadru *m şi succesorul său m+1*.
- *În cazul în care cadrul m este pierdut sau* modificat, receptorul nu îl va confirma, aşa încât emiţătorul va încerca să-l retransmită.
- Odată ce a fost corect recepţionat, receptorul va trimite o confirmare înapoi la emiţător.
- După cum cadrul de confirmare ajunge sau nu corect înapoi la emiţător, emiţătorul va încerca să transmită *m sau m+1*.

Deosebiri

- Protocolul 3 se deosebeşte de predecesorii săi prin aceea că şi emiţătorul şi receptorul au o variabilă a cărei valoare este păstrată cât timp nivelul legătură de date este în starea de aşteptare.
- Emiţătorul păstrează numărul de secvenţă al următorului cadru de transmis în *next_frame_to_send;*
- *Receptorul* păstrează numărul de secvenţă al următorului cadru aşteptat în *frame_expected.*
- *Fiecare protocol* are o scurtă fază de iniţializare înainte de a intra în bucla infinită.

Functionare

- După transmiterea unui cadru, emiţătorul declanşează contorul de timp.
- Dacă acesta era deja pornit, atunci va fi resetat pentru un nou interval complet.
- Intervalul de timp trebuie să fie ales astfel, încât să permită sosirea cadrului la receptor, prelucrarea sa de către receptor, chiar şi în cazul cel mai defavorabil, şi propagarea cadrului de confirmare înapoi la emiţător.

Functionare

- Numai atunci când a expirat acest interval de timp se poate spune cu siguranţă că s-a pierdut fie cadrul transmis, fie confirmarea
- sa şi se poate trimite un duplicat.
- Dacă intervalul de timp este prea mic, emiţătorul va transmite cadre care nu sunt necesare.
- Aceste cadre nu influenţează corectitudinea protocolului, dar îi afectează performanţele

Functionare (3)

- După transmiterea unui cadru şi pornirea contorului de timp, emiţătorul aşteaptă să se întâmple ceva interesant.
- Există doar trei posibilităţi:
  - un cadru de confirmare ajunge intact,
  - soseşte un cadru de confirmare eronat
  - expiră timpul.

Variante

- Dacă soseşte o confirmare validă, atunci emiţătorul preia următorul pachet de la nivelul reţea şi îl pune în tampon, scriind peste pachetul anterior.

- De asemenea avansează numărul de secvenţă.
- Dacă soseşte un cadru modificat sau nu soseşte nici un cadru, nu se modifică nici tamponul şi nici numărul de secvenţă, aşa că poate fi transmis un duplicat.

Final

- Atunci când la receptor soseşte un cadru corect, este verificat numărul de secvenţă, pentru a vedea dacă nu cumva este un duplicat.
- Dacă nu, este acceptat, transmis nivelului reţea şi este generată o confirmare.
- Cadrele duplicate şi cele modificate nu sunt trimise către nivelul reţea.

Protocolul 3

```
/* Protocolul 3 (par) permite un flux de date unidirecţional, printr-un canal nesigur. */

#define MAX_SEQ 1                                   /* trebuie să fie 1 pentru protocolul 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
 seq_nr next_frame_to_send;           /* numărul de secvenţă al următorului cadru trimis */
 frame s;                                                    /* variabilă temporară */
 packet buffer;                                  /* tampon pentru pachetul transmis */
 event_type event;

 next_frame_to_send = 0;                 /* iniţializează numerele de secvenţă de ieşire */
 from_network_layer(&buffer);                                /* preia primul pachet */
 while (true) {
   s.info = buffer;                              /* construieşte un cadru pentru transmitere */
   s.seq = next_frame_to_send; /* inserează în cadru un număr de secvenţă */
   to_physical_layer(&s);                                    /* îl trimite pe traseu */
   start_timer(s.seq);           /* dacă răspunsul întârzie prea mult, timpul va expira */
   wait_for_event(&event);                       /* frame_arrival, cksum_err, timeout */
   if (event == frame_arrival) {                             /* preia confirmarea */
     from_physical_layer(&s);
    if(s.ack==next_frame_to_send) {
      stop_timer(s.ack);                                     /* opreşte ceasul */
      from_network_layer(&buffer);              /* preia următorul pachet de transmis */
      inc(next_frame_to_send);                  /* inversează next_frame_to_send */
   }
 }
 }
}

void receiver3(void)
{
 seq_nr frame_expected;
 frame r, s;
 event_type event;

 frame_expected = 0;
 while (true) {
   wait_for_event(&event);                       /* variante: frame_arrival, cksum_err */
   if (event == frame_arrival) {                          /* a sosit un cadru corect */
     from_physical_layer((r);                              /* preia cadrul nou sosit */
     if (r.seq == frame_expected) {                /* este cel pe care îl aşteptam */
       to_network_layer((r.info);                /* transferă datele nivelului reţea */
       inc(frame_expected);                       /* incrementează număr de secvenţă */
     }
     s.ack=1-frame_expected;                      /* spune care frame este confirmat */
     to_physical_layer(&s);              /* nici unul dintre câmpuri nu este utilizat */
   }
 }
}
```

PROTOCOALE CU FEREASTRĂ GLISANTĂ

- În protocoalele anterioare, cadrele cu date erau transmise într-o singură direcţie.
- În cele mai multe situaţii practice, este necesar să se transmită date în ambele direcţii.
- O modalitate de a realiza transmisia de date full-duplex este de a avea două canale de comunicaţie separate, fiecare dintre ele fiind utilizat pentru traficul de date simplex (în direcţii diferite).

Duplex

- Dacă se face aceasta, vom avea două circuite fizice separate, fiecare cu un canal "direct" (eng.: forward) pentru date și un canal "invers” (eng.: reverse) pentru confirmări.
- În ambele cazuri lărgimea de bandă a canalului invers este irosită aproape în totalitate. Ca efect, utilizatorul plăteşte pentru două circuite, dar utilizează doar capacitatea unuia.

Duplex

O idee mai bună este să se utilizeze acelaşi circuit pentru date în ambele direcţii

- Uitându-se la câmpul *kind din antetul cadrului ce a sosit, receptorul* poate spune dacă este vorba de un cadru de date sau de confirmare
- Mai este posibilă încă o îmbunătăţire.

Atasarea

- Atunci când sosește un cadru cu date, în locul emiterii imediate a unui cadru de control separat, receptorul stă şi aşteaptă până când nivelul reţea îi dă următorul pachet.
- Confirmarea este ataşată cadrului cu date de ieşire (utilizând câmpul *ack din antetul cadrului).*
- *De fapt, confirmarea este transportată* pe gratis de către următorul cadru cu date de ieşire.
- Tehnica întârzierii confirmării, astfel încât să poată fi agăţată de următorul cadru de date, este cunoscută ca **ataşare (eng.: piggybacking).**

Atasarea

- Principalul avantaj al utilizării tehnicii de ataşare în comparaţie cu utilizarea cadrelor de confirmare distincte este o mai bună utilizare a lărgimii de bandă disponibile.
- Câmpul *ack din antetul cadrului* ocupă doar câţiva biţi, în timp ce un cadru separat va necesita un antet, confirmarea şi o sumă de control.
- În plus, mai puţine cadre transmise înseamnă mai puţine întreruperi datorate "sosirii cadrelor" şi probabil mai puţine zone tampon în receptor, în funcţie de modul în care sunt organizate programele receptorului.

Complicatii

- Tehnica de ataşare introduce o complicaţie care nu era prezentă în cazul confirmărilor separate.
- Cât timp trebuie să aştepte nivelul legătură de date pachetul la care să ataşeze confirmarea?
- Dacă nivelul legătură de date aşteaptă mai mult timp decât perioada de timeout a emiţătorului, cadrul va fi retransmis, anulând complet rolul confirmărilor.

Asteptarea

- Dacă nivelul legătură de date ar fi un oracol şi ar putea prezice viitorul, ar putea şti când va sosi următorul pachet de la nivelul reţea şi ar putea decide dacă să îl aştepte sau să trimită imediat o confirmare separată, în funcţie de cât de lungă urmează să fie aşteptarea.
- Nivelul legătură de date nu poate prezice viitorul, aşa că trebuie să recurgem la câteva scheme ad-hoc, cum ar fi aşteptarea pentru un număr fixat de milisecunde.
- Dacă un nou pachet soseşte repede, confirmarea este adăugată în el; altfel, dacă până la sfârşitul acestei perioade de timp nu a sosit un nou pachet, nivelul legătură de date trimite un cadru de confirmare separat.

Fereastră glisantă

- Trei protocoale bidirecţionale care aparţin unei clase de protocoale numite protocoale cu **fereastră glisantă.**
- In toate protocoalele cu fereastră glisantă, fiecare cadru expediat conţine un număr de secvenţă cuprins între 0 şi o valoare maximă.
- Maximul este de obicei 2^n -1, ca numărul de secvenţă să se încadreze exact într-un câmp de *n biţi. Protocoalele cu fereastră glisantă pas-cu-pas utilizează n=1, restricţionând numerele de secvenţă la 0 şi 1, dar versiuni mai sofisticate pot utiliza o valoare* arbitrară a lui *n.*

Esenta

- Esenţa protocoalelor cu fereastră glisantă este aceea că, la orice moment de timp, emiţătorul menţine o mulţime de numere de secvenţă care corespund cadrelor pe care are permisiunea să le trimită.
- Se spune că aceste cadre aparţin **ferestrei de transmisie (eng.: sending window).**
- **Similar, receptorul** menţine de asemenea o **fereastră de recepţie (eng.: receiving window), ce corespunde mulţimii** de cadre care pot fi acceptate.

Limite

- Fereastra emiţătorului şi fereastra receptorului nu trebuie să aibă aceleaşi limite minime şi maxime şi nici măcar aceeaşi dimensiune.
- În unele protocoale ele au dimensiune fixă, dar în altele ele pot creşte sau scădea pe măsură ce cadrele sunt emise sau recepţionate.

Numere de secventa

- Numerele de secvenţă din cadrul ferestrei emiţătorului reprezintă cadre transmise sau cadre ce pot fi transmise, dar încă neconfirmate.
- De fiecare dată când de la nivelul reţea soseşte un nou pachet, acestuia îi este atribuit următorul număr de secvenţă, iar marginea superioară a ferestrei este avansată cu unu. Atunci când soseşte o confirmare, creşte cu unu limita inferioară a ferestrei.
- În acest mod, fereastra menţine continuu o listă de cadre neconfirmate.

Schema

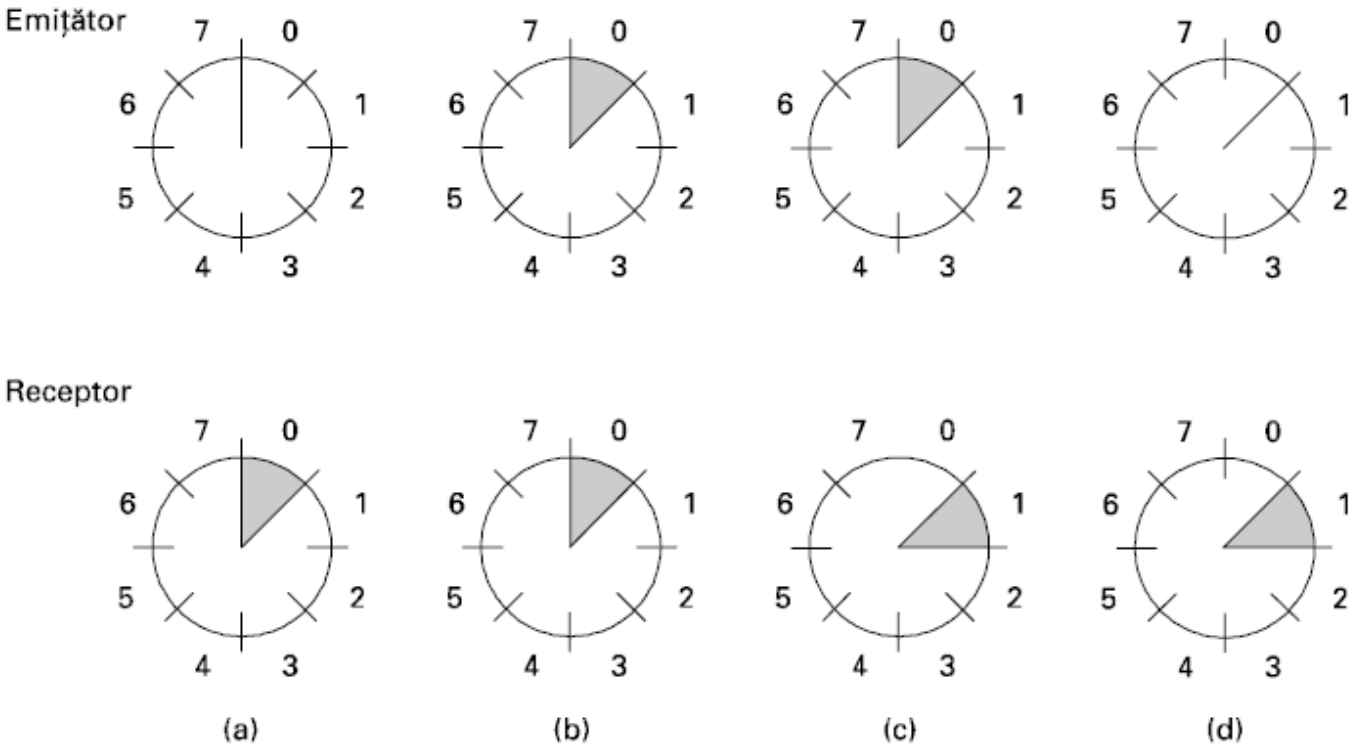


Fig. 3-13. O fereastră glisantă de dimensiune 1, cu număr de secvență de 3 biți. (a) Inițial. (b) După ce a fost transmis primul cadru. (c) După ce a fost recepționat primul cadru. (d) După ce a fost recepționată prima confirmare.

Dimensiune

- Deoarece cadrele din fereastra curentă a emiţătorului pot fi pierdute sau modificate pe traseu, emiţătorul trebuie să păstreze toate aceste cadre în memoria sa pentru o posibilă retransmisie.
- Astfel, dacă dimensiunea maximă a ferestrei este *n*, *emiţătorul are nevoie de n tampoane pentru a păstra* cadrele neconfirmate.
- Dacă fereastra creşte la dimensiunea maximă, nivelul legătură de date al emiţătorului trebuie să forţeze închiderea nivelului reţea până când se eliberează un tampon.

Fereastra

- Fereastra nivelului legătură de date receptor corespunde cadrelor pe care acesta le poate accepta.
- Orice cadru din afara ferestrei este eliminat fără comentarii.
- Atunci când este recepţionat un cadru al cărui număr de secvenţă este egal cu marginea inferioară a ferestrei, acesta este trimis nivelului reţea, este generată o confirmare şi fereastra se deplasează cu o unitate.

Fereastra

- Spre deosebire de fereastra emiţătorului, fereastra receptorului rămâne întotdeauna la dimensiunea iniţială.
- O fereastră de dimensiune 1 înseamnă că nivelul legătură de date acceptă numai cadre ordonate, dar pentru ferestre mari afirmaţia nu mai este valabilă.
- Nivelul reţea este, dimpotrivă, alimentat întotdeauna cu date în ordine corectă, indiferent de dimensiunea ferestrei nivelului legătură de date.

MULȚUMESC!

Nivelul Legatura de date (5)

Liviu P. Dinu
ldinu@fmi.unibuc.ro
Center for Computational Linguistics
nlp.unibuc.ro

Protocol cu fereastră glisantă de un bit

- Dimensiunea maximă a ferestrei= 1.
- Utilizează metoda stop-and-wait, deoarece emiţătorul transmite un cadru şi aşteaptă confirmarea sa înaintea transmiterii următorului cadru.
- Definitii:
- *Next_frame_to_send arată ce cadru încearcă să transmită emiţătorul. Similar,*
- *frame_ expected* arată ce cadru este aşteptat de receptor. În ambele cazuri singurele posibilităţi sunt 0 şi 1.

Fereastră glisantă de un bit

- Unul dintre cele două niveluri legătură de date porneşte primul trimiţând primul cadru.
- Numai unul din programele nivelului legătură de date va conţine apelurile procedurilor *to_physical_layer şi start_timer în afara buclei principale*.
- *Dacă ambele* niveluri legătură de date pornesc simultan, apare o situaţie specială.
- Maşina care porneşte prima preia primul pachet de la nivelul reţea propriu, construieşte din el un cadru şi îl trimite.

Fereastră glisantă de un bit

- Când acest cadru (sau oricare altul) soseşte, nivelul legătură de date receptor verifică dacă nu cumva este un duplicat, exact ca în protocolul 3.
- Dacă respectivul cadru este cel aşteptat, atunci este trimis nivelului reţea şi fereastra receptorului este deplasată.

Fereastră glisantă de un bit

- Câmpul de confirmare conţine numărul ultimului cadru recepţionat fără eroare.
- Dacă acest număr corespunde cu numărul de secvenţă al cadrului pe care emiţătorul încearcă să-l transmită, emiţătorul ştie că a terminat cu cadrul memorat în tampon şi poate prelua următorul pachet de la nivelul său reţea.
- Dacă numărul de secvenţă nu corespunde, el trebuie să continue să trimită acelaşi cadru.
- De fiecare dată când este recepţionat un cadru, un alt cadru este trimis de asemenea înapoi

Exemplu

- Presupunem că *A încearcă* să trimită cadrul 0 lui *B şi B încearcă să trimită cadrul său 0 lui A.*
- *Presupunem că A trimite un* cadru lui *B, dar timpul de expirare al lui A este puţin prea scurt.*
- *În consecinţă, datorită expirării repetate* a timpului, *A va trimite o serie de cadre identice, toate cu seq=0 şi ack=1.*
- Atunci când la *B sosește primul cadru corect, el va fi acceptat și frame_expected va fi setat la 1.*

Exemplu (2)

- Toate cadrele următoare vor fi respinse, deoarece B aşteaptă cadre cu numărul de secvenţă 1, nu 0.
- Mai mult, deoarece toate duplicatele au *ack=1 şi B este încă în aşteptarea confirmării lui 0, B nu va* prelua un nou pachet de la nivelul său reţea.
- După sosirea fiecărui duplicat respins, B trimite lui A un cadru conţinând *seq=0 şi ack=0.*
- *În cele* din urmă, unul dintre acestea soseşte corect la *A, făcându-l pe A să înceapă să trimită următorul pachet.*

Exemplu (3)

- Nici o combinaţie de cadre pierdute sau de intervale de ceas reduse nu poate face ca protocolul să furnizeze pachete duplicate către vreunul dintre nivelurile reţea, să sară un pachet sau să se blocheze.
- Dacă ambele părţi trimit simultan un pachet iniţial, atunci apare o situaţie specială.
- Această dificultate de sincronizare este ilustrată de slide-ul urmator

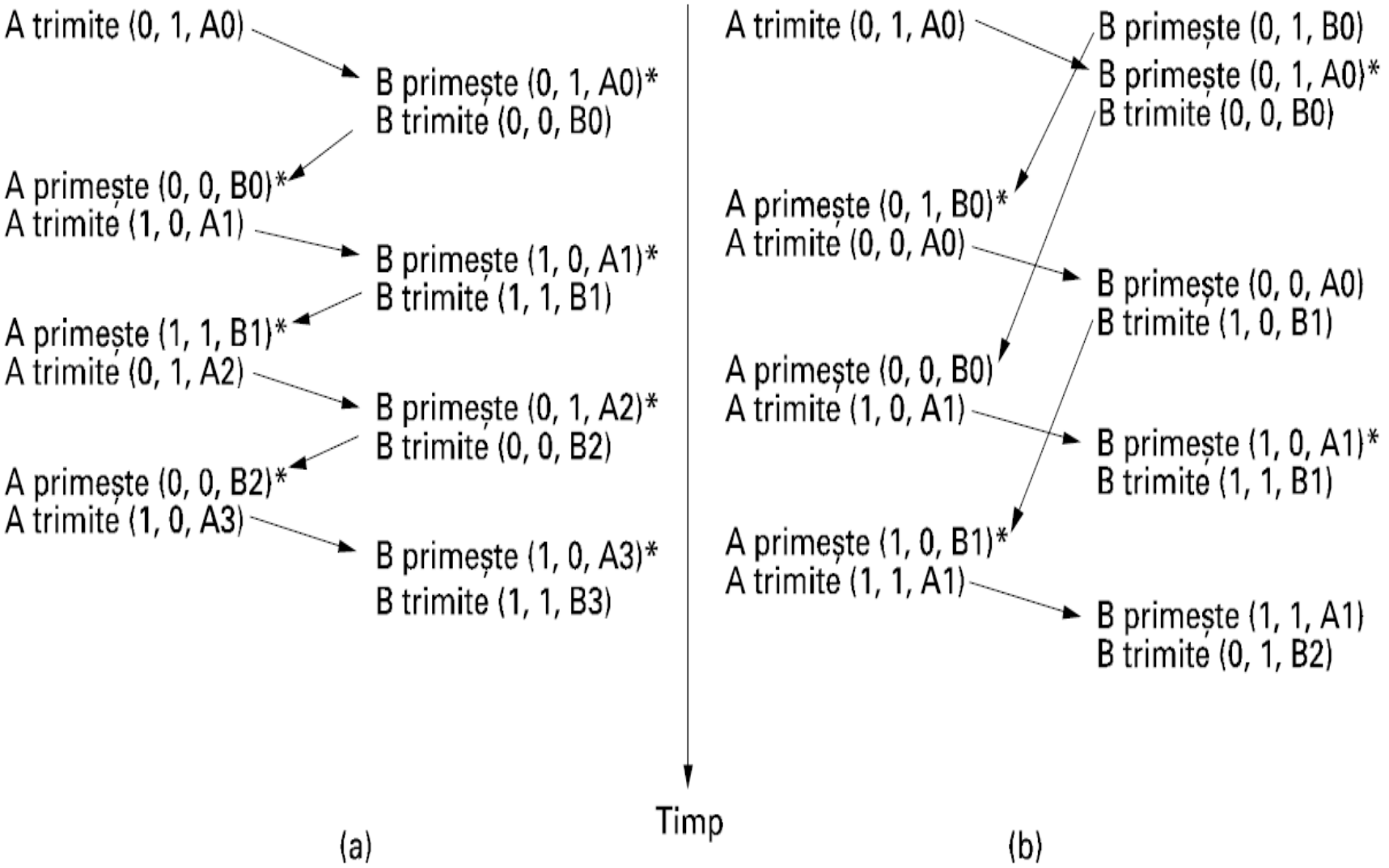


Fig. 3-15. Două scenarii pentru protocolul 4. (a) Cazul normal. (b) Cazul special. Notația este (seq, ack, packet number). Un asterisc arată că nivelul rețea acceptă un pachet.

Special

- Situaţia specială:
- Dacă *B aşteaptă primul cadru de la A înainte de a-l trimite pe al său, secvenţa de acţiuni este cea arătată în (a) şi fiecare cadru este acceptat.*
- Totuşi, dacă *A şi B iniţiază comunicaţia simultan, primele lor cadre se încrucişează şi nivelurile* legătură de date ajung în situaţia (b).
- În (a) fiecare sosire de cadru aduce un nou pachet pentru nivelul reţea; nu există duplicate.

Special (2)

- În (b) jumătate din cadre conţin duplicate, chiar dacă nu există erori de transmisie.
- Situaţii similare pot să apară ca rezultat al expirării premature a timpului, chiar dacă una dintre părţi începe prima.
- Dacă intervin mai multe expirări premature, atunci cadrele pot fi trimise de trei sau mai multe ori.

Protocol de revenire cu *n paşi (Go Back n)*

- Presupunere tacită: timpul de transmisie necesar pentru ca un cadru să ajungă la receptor plus timpul de transmisie a confirmării este neglijabil.
- Uneori această presupunere este în mod cert falsă.
- În aceste situaţii timpul mare de transfer poate avea implicaţii importante pentru eficienţa utilizării lărgimii de bandă.

Exemplu

- Considerăm un canal de satelit de 50 Kbps cu timpul de întârziere datorită propagării dus-întors de 500 milisecunde.
- Incercăm să utilizăm protocolul 4 pentru a trimite cadre de 1000 de biţi prin satelit.
- La *t = 0 emiţătorul începe să* trimită primul cadru.
- In condiţii optime (fără aşteptare la receptor şi un cadru de confirmare scurt), cadrul nu poate ajunge în totalitate la receptor înainte de *t = 270 milisecunde,* iar confirmarea nu poate ajunge înapoi la emiţător înainte de *t = 520 milisecunde*.

Exemplu (2)

- *Aceasta înseamnă* că emiţătorul a fost blocat pentru 500/520 sau 96% din timp.
- Cu alte cuvinte, a fost utilizată doar 4% din lărgimea de bandă.
- Combinaţia dintre un timp de tranziţie lung, lărgime de bandă mare şi un cadru de lungime mică este dezastruoasă din punct de vedere al eficienţei.

Relaxare

- Consecinţă a regulii care cere ca un emiţător să aştepte o confirmare înaintea trimiterii unui alt cadru.
- Relaxăm această restricţie:
  - permitem emiţătorului să transmită până la *w cadre*, *în loc de unul singur.*
- *Cu o alegere potrivită a lui w emiţătorul va putea să transmită continuu* cadre pentru un timp egal cu timpul de tranzit, fără a umple fereastra

Exemplu (3, cont)

- În exemplul anterior *w va* fi minim 26.
- Emiţătorul începe emiterea cadrului 0 ca mai înainte.
- În momentul în care se termină trimiterea a 26 de cadre, la *t = 520, va sosi şi confirmarea pentru cadrul 0.*
- *Apoi, confirmările vor* sosi la fiecare 20 milisecunde, aşa încât emiţătorul primeşte întotdeauna permisiunea să continue exact atunci când doreşte.
- În permanenţă există 25 sau 26 cadre neconfirmate.
- Cu alte cuvinte dimensiunea maximă a ferestrei emiţătorului este de 26.

Bandă de asamblare

- Nevoia pentru o fereastră mare la emiţător apare atunci când lărgime de bandă x timpul de propagare dus-întors este mare.
- Dacă lărgimea de bandă este mare, chiar şi pentru întârzieri moderate, emiţătorul îşi va termina repede fereastra.
- Daca întârzierea este mare (de exemplu, canal de satelit), emiţătorul îşi va termina fereastra chiar şi pentru lărgimi de bandă moderate.

Bandă de asamblare (2)

- Produsul acestor doi factori spune de fapt care este capacitatea canalului, iar pentru a opera la eficienţă maximă, emiţătorul trebuie sa fie capabil sa o umple fără să se oprească.
- Această tehnică este cunoscută ca **bandă de asamblare (pipelining).**
- **Considerând capacitatea** canalului de *b biţi pe secundă, dimensiunea cadrului de l biţi şi timpul de propagare dus-întors R* secunde, timpul necesar pentru a transmite un singur cadru este *l/b secunde*.

Utilizarea liniei

- *După ce a fost transmis* ultimul bit al unui cadru de date, apare o întârziere de *R/2 înainte ca biţii să ajungă la receptor şi o* altă întârziere de cel puţin *R/2 pentru sosirea confirmării, rezultând o întârziere totală de R.*
- *În cazul* protocoalelor pas-cu-pas, linia este ocupată pentru un timp egal cu *l/b şi în aşteptare pentru un timp* egal cu *R, rezultând:*

$$\text{utilizarea liniei} = l/(l+bR).$$

Eficienta

- Dacă *l < bR, eficienţa va fi mai mică de 50%.*
- *Deoarece până la întoarcerea confirmării există întotdeauna* o întârziere nenulă, în principiu poate fi folosită banda de asamblare, pentru a ţine linia ocupată tot acest interval
- Dacă intervalul este mic, complexitatea suplimentară face efortul inutil.

Probleme

- Ce se întâmplă dacă un cadru din mijlocul unui şir lung este modificat sau pierdut?
- Multe cadre succesive vor ajunge la receptor înainte ca emiţătorul să observe că cevaeste greşit.
- Atunci când un cadru modificat ajunge la receptor este evident că el trebuie eliminat.
- Ce trebuie să facă receptorul cu toate cadrele corecte care urmează?

Exemplu

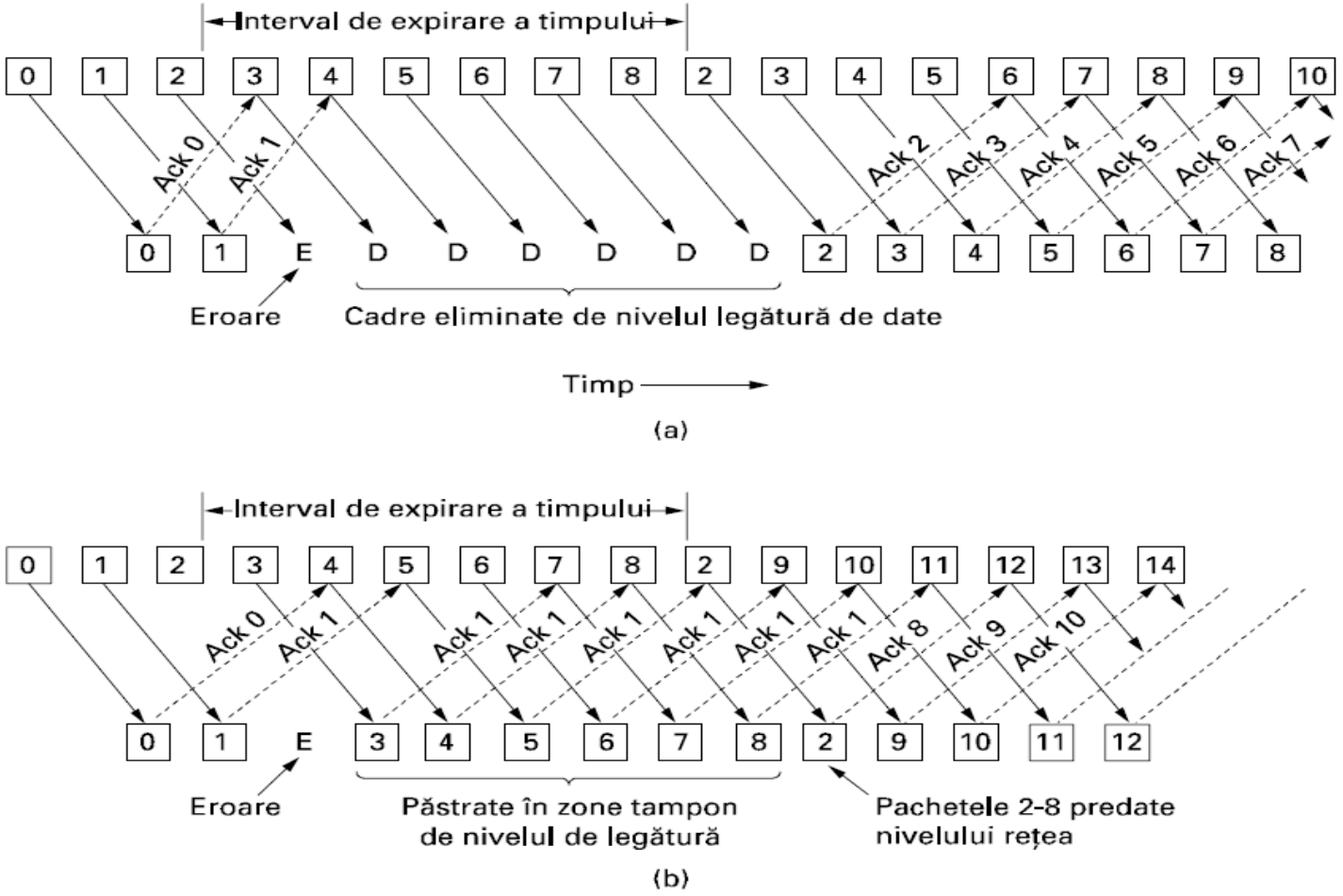


Fig. 3-16. Folosirea benzii de asamblare și revenirea din eroare. Efectul unei erori când (a) dimensiunea ferestrei receptoare este 1 și (b) dimensiunea ferestrei receptorului este mare.

Tratarea erorilor

- Două moduri de bază de tratare a erorilor în prezenţa benzii de asamblare.
- **revenire cu n paşi (eng.: go back n),**
- **repetare selectivă (eng.: selective repeat).**

Go back n

- **Inseamna ca receptorul să elimine pur şi simplu cadrele care urmează,** netrimiţând confirmări pentru cadrele eliminate.
- Această strategie corespunde unei ferestre de recepţie de dimensiune 1.
- Cu alte cuvinte, nivelul legătură de date refuză să accepte orice cadru exceptându-l pe următorul care trebuie livrat către nivelul reţea.

Go back n (cont)

- Dacă fereastra emiţătorului se umple înaintea expirării contorului de timp, banda de asamblare va începe să se golească.
- În cele din urmă, timpul emiţătorului va expira şi se vor retransmite toate cadrele neconfirmate, în ordine, începând cu cadrul pierdut sau modificat.
- Dacă rata erorilor este mare, această abordare poate risipi o mare parte din lărgimea de bandă.

```
/* Protocolul 5 (revenire cu n paşi) permite mai multe cadre în aşteptare. Emiţătorul poate
trimite până la MAX_SEQ cadre fără a aştepta confirmare. În plus, spre deosebire de proto-
coalele precedente, acesta nu presupune că nivelul reţea ar avea tot timpul un nou pachet.
În schimb, nivelul reţea provoacă un eveniment network_layer_ready atunci când are de tri-
mis un pachet */

#define MAX_SEQ 7                                        /* trebuie să fie 2^n - 1 */
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
 /* întoarce adevărat dacă a <= b < c în mod circular şi fals în caz contrar */
 if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
   return(true);
 else
   return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
 /* construieşte şi trimite un cadru de date */
 frame s;                                                /* variabilă temporară */

 s.info = buffer[frame_nr];                        /* inserează pachetul în cadru */
 s.seq = frame_nr;                              /* inserează numărul de secvenţă în cadru */
 s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);          /* ataşează confirmarea */
 to_physical_layer(&s);                                  /* transmite cadrul */
 start_timer(frame_nr);                                  /* porneşte ceasul */
}

void protocol5(void)
{
 seq_nr next_frame_to_send;              /* MAX_SEQ > 1; utilizat pentru fluxul de ieşire */
 seq_nr ack_expected;                          /* cel mai vechi cadru încă neconfirmat */
 seq_nr frame_expected;               /* următorul cadru aşteptat, din fluxul de intrare */
 frame r;                                                /* variabilă auxiliară */
 packet buffer[MAX_SEQ+1];                        /* zone tampon pentru fluxul de ieşire */
 seq_nr nbuffered;                /* număr de zone tampon de ieşire utilizate în prezent */
 seq_nr i;                              /* utilizat ca index în vectorul de zone tampon */
 event_type event;
 enable_network_layer();                        /* permite evenimente network_layer_ready */
 ack_expected = 0;                                  /* următoarea confirmare aşteptată */
 next_frame_to_send = 0;                                /* următorul cadru transmis */
 frame_expected = 0;                             /* numărul cadrului aşteptat să sosească */
 nbuffered = 0;                     /* iniţial în zonele tampon nu există nici un pachet */
 while(true) {
   wait_for_event(&event);                 /* patru posibilităţi: vezi event_type, mai sus */
   switch(event) {
     case network_layer_ready:                      /* nivelul reţea are un pachet de trimis */
       /* acceptă, salvează şi transmite un nou cadru */
       from_network_layer(&buffer[next_frame_to_send]);              /* preia noul pachet */
       nbuffered = nbuffered + 1;                        /* extinde fereastra emiţătorului */
       send_data(next_frame_to_send, frame_expected, buffer);         /* transmite cadrul */
       inc(next_frame_to_send);      /* creşte limita superioară a ferestrei emiţătorului */
       break;
```

```
      case frame_arrival:                          /* a sosit un cadru de date sau de control */
        from_physical_layer(&r);                    /* preia de la nivelul fizic cadrul sosit */

        if (r.seq == frame_expected) {
                                                    /* cadrele sunt acceptate doar în ordine */
        to_network_layer(&r.info);                        /* predă pachetul nivelului rețea */
                  inc(frame_expected);/* crește limita inferioară a ferestrei emițătorului*/
        }
        /* Confirmarea lui n implică n-1, n-2 etc. Verifică acest lucru. */
        while (between(ack_expected, r.ack, next_frame_to_send)) {
        /* tratează confirmarea atașată */
        nbuffered = nbuffered - 1;                    /* un cadru mai puțin în zonele tampon */
        stop_timer(ack_expected);                   /* cadrul a sosit intact; oprește ceasul */
        inc(ack_expected);                               /* contractă fereastra emițătorului */
        }
        break;
      case cksum_err: break;                    /* cadrele eronate sunt pur și simplu ignorate */
      case timeout:                            /* necaz; retransmite toate cadrele neconfirmate */
        next_frame_to_send = ack_expected;                  /* începe retransmiterea de aici */
        for (i=1; i <= nbuffered; i++) {
              send_data(next_frame_to_send, frame_expected, buffer);/* retransmite 1 cadru */
          inc(next_frame_to_send);                    /* pregătește transmiterea următorului */
        }

    }

    if (nbuffered < MAX_SEQ)
      enable_network_layer();
    else
      disable_network_layer();
  }
}
```

Fig. 3-17. Un protocol cu fereastră glisantă utilizând revenirea cu n pași.

Repetare selectivă.

- Când aceasta este utilizată, un cadru incorect este respins, dar toate cadrele corecte care îl urmează sunt memorate.
- Când contorul de timp al emiţătorului expiră, cel mai vechi cadru neconfirmat este retransmis.
- Dacă acest cadru ajunge corect, receptorul poate transmite către nivelul reţea, în ordine, cadrele pe care le-a memorat.
- Repetarea selectivă este deseori combinată cu utilizarea confirmărilor negative (NAK), care sunt trimise atunci când se detectează o eroare

Repetare selectivă.

- Confirmările negative simulează retransmisia înainte de expirarea contorului de timp corespunzător, îmbunătăţind astfel performanţa.
- Strategia de repetare selectivă corespunde unei ferestre a receptorului mai mare ca 1.
- Orice cadru din interiorul ferestrei poate fi acceptat şi memorat până când toate cele precedente vor fi trimise nivelului rețea.
- Dacă fereastra este mare, această abordare poate necesita un spaţiu mare de memorie pentru nivelul legătură de date.

Exemplu (cont slide 83, fig. b)

- În fig. 3-16 (b), cadrele 0 şi 1 sunt recepţionate corect, dar confirmarea pentru cadrul 2 este pierdută.
- Când cadrul 3 soseşte la receptor, nivelul legătură de date observă că a pierdut un cadru, şi trimite o confirmare negativă pentru 2, memorând însă cadrul primit.
- Când cadrele 4, 5 ajung la receptor, sunt la rândul lor memorate de nivelul legătură de date, în loc de a fi transmise nivelului reţea.

NAK

- În cele din urmă, confirmarea negativă pentru 2 ajunge înapoi la emiţător, care va retransmite cadrul 2.
- Când acesta ajunge la receptor, nivelul legătură de date are cadrele 2, 3, 4, 5, şi le poate pasa nivelului reţea în ordinea corectă.
- De asemenea, poate confirma toate cadrele până la 5 inclusiv, aşa cum se prezintă în figură.

Confirmari negative

- Dacă NAK-ul se pierde, în cele din urmă contorul de timp al emiţătorului va expira și acesta va iniția retransmisia, dar în acest fel se pierde mai mult timp.
- În concluzie, utilizarea confirmărilor negative accelerează retransmiterea unui anumit cadru.

Concluzii

- Strategia de repetare selectivă corespunde unei ferestre a receptorului mai mare ca 1.
- Orice cadru din interiorul ferestrei poate fi acceptat şi memorat până când toate cele precedente vor fi trimise
- nivelului reţea.
- Dacă fereastra este mare, această abordare poate necesita un spaţiu mare de memorie pentru nivelul legătură de date.

Concluzii

- Aceste două alternative reprezintă compromisuri între lărgimea de bandă şi spaţiul ocupat de tampoane la nivelul legătură de date.
- În funcţie de care resursă este mai deficitară, poate fi utilizată una sau cealaltă.

MULȚUMESC!

Verificarea protocoalelor

Liviu P. Dinu
ldinu@fmi.unibuc.ro
Center for Computational Linguistics
nlp.unibuc.ro

Introducere

- **Fiecare automat al protocolului (adică transmiţător sau receptor) se afla în fiecare moment de** timp într-o stare specifică.
- Stările sale constau din toate valorile variabilelor sale, incluzând contorul de instrucţiuni al programului.

Introducere (cont)

- În cele mai multe cazuri, un număr mare de stări pot fi grupate împreună, în vederea analizei.

- Exemplu: pt. receptorul din protocolul 3, abstractizam toate stările posibile în două stări importante: aşteptarea cadrului 0 sau aşteptarea cadrului 1.

Automate

- Toate celelalte stări pot fi considerate ca fiind tranzitorii, simpli paşi pe calea spre una din stările principale.

- De obicei, stările sunt alese ca fiind acele momente în care automatul protocolului aşteaptă să se petreacă următorul eveniment [să execute apelul de procedură *wait(event) din exemplele noastre]*.

Automate

- *În acest punct, starea* automatului este complet determinată de stările variabilelor sale. Numărul de stări este deci 2*n*, unde *n este numărul de biți necesari pentru reprezentarea tuturor combinațiilor de variabile.*

- Starea întregului sistem este combinaţia tuturor stărilor celor două automate ale protocolului şi a stării canalului. Starea canalului este determinată de conţinutul său.

Automate

- Exemplu: pentru protocolul 3, canalul are patru stări posibile:
  - un cadru zero sau un cadru unu circulând de la transmiţător la receptor,
  - un cadru de confirmare circulând în sens invers sau nici un cadru.
- Dacă modelăm transmiţătorul sau receptorul prin două stări, întregul sistem are 16 stări distincte

Automate

- Din fiecare stare, există zero sau mai multe **tranziţii posibile spre alte stări. Tranziţiile au loc** atunci când se petrece un eveniment.
- Pentru un automat, o tranziţie trebuie să se facă atunci când:
  1. este trimis un cadru,
  2. când soseşte un cadru,
  3. când expiră un interval de timp,
  4. când apare o întrerupere

Evenimente canal

- Pentru canal, evenimentele tipice sunt:
  1. introducerea unui nou cadru pe canal de către automatul protocolului,
  2. livrarea cadrului unui automat
  3. pierderea unui cadru datorată unei rafale de zgomote.
- Pentru o descriere completă a automatelor protocolului şi a caracteristicilor canalului, putem trasa graful orientat care prezintă toate stările automatului ca noduri şi toate tranziţiile ca arce orientate.

Stari

- **Starea iniţială: această unica stare corespunde descrierii sistemului** (atunci când el începe să funcţioneze), sau unui punct de pornire convenabil imediat următor.
- Unele stări (posibil chiar toate stările) pot fi atinse din starea iniţială printr-o secvenţă de tranziţii.
- Se determina usor care stări sunt accesibile şi care nu. (**analiza accesibilităţii**) (Lin ş.a., 1987). Această analiză poate fi utilă în determinarea corectitudinii protocolului.

Model automat

- Formal, un model de tip automat finit al unui protocol poate fi privit ca un cvadruplu *(S, M, I, T) unde:*
  1. *S este mulţimea stărilor în care se pot găsi procesele şi canalul*
  2. *M este mulţimea cadrelor care pot fi schimbate prin canal*
  3. *I este mulţimea stărilor iniţiale ale proceselor*
  4. *T este mulţimea tranziţiilor între stări*

Descriere

- La începutul intervalului de timp, toate procesele se găsesc în stările lor inițiale.
- Apoi încep să se producă evenimente, (disponibilizarea unor cadre pentru transmisie, expirarea unor intervale de timp, etc).

Descriere

- Fiecare eveniment poate face ca unul dintre procese sau canalul să execute o acţiune şi să comute într-o nouă stare.
- Prin enumerarea atentă a fiecărui succesor posibil pentru fiecare stare, se poate construi graful de accesibilitate și se poate analiza protocolul.

Analiza accesibilitatii

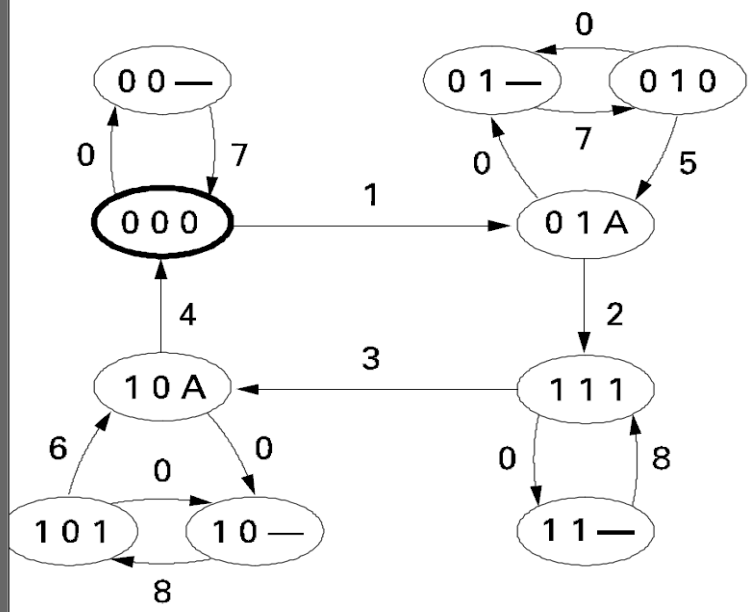
- Analiza accesibilităţii poate fi folosită pentru a detecta diferite erori în specificaţia protocolului.
- Exemplu:
  1. dacă este posibil ca un anumit cadru să apară într-o anumită stare şi automatul finit să nu ştie ce acţiune trebuie întreprinsă, atunci specificaţia este eronată (incompletitudine).

Analiza accesibilitatii (cont)

1. Dacă există o mulţime de stări fără ieşire şi din care nu se poate progresa, avem o altă eroare (interblocare).
2. Specificaţia protocolului spune cum să se trateze un eveniment într-o stare în care evenimentul nu se poate produce (tranziţie neesenţială).

```
/* Protocolul 3 (par) permite un flux de date unidirecţional, printr-un canal nesigur. */

#define MAX_SEQ 1                              /* trebuie să fie 1 pentru protocolul 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
 seq_nr next_frame_to_send;          /* numărul de secvenţă al următorului cadru trimis */
 frame s;                                                    /* variabilă temporară */
 packet buffer;                                    /* tampon pentru pachetul transmis */
 event_type event;

 next_frame_to_send = 0;               /* iniţializează numerele de secvenţă de ieşire */
 from_network_layer(&buffer);                                /* preia primul pachet */
 while (true) {
   s.info = buffer;                          /* construieşte un cadru pentru transmitere */
   s.seq = next_frame_to_send; /* inserează în cadru un număr de secvenţă */
   to_physical_layer(&s);                                   /* îl trimite pe traseu */
   start_timer(s.seq);           /* dacă răspunsul întârzie prea mult, timpul va expira */
   wait_for_event(&event);                        /* frame_arrival, cksum_err, timeout */
   if (event == frame_arrival) {                             /* preia confirmarea */
     from_physical_layer(&s);
    if(s.ack==next_frame_to_send) {
      stop_timer(s.ack);                                          /* opreşte ceasul */
      from_network_layer(&buffer);               /* preia următorul pachet de transmis */
      inc(next_frame_to_send);                      /* inversează next_frame_to_send */
   }
 }
 }
}

void receiver3(void)
{
 seq_nr frame_expected;
 frame r, s;
 event_type event;

 frame_expected = 0;
 while (true) {
   wait_for_event(&event);                      /* variante: frame_arrival, cksum_err */
   if (event == frame_arrival) {                          /* a sosit un cadru corect */
     from_physical_layer((r);                              /* preia cadrul nou sosit */
     if (r.seq == frame_expected) {                /* este cel pe care îl aşteptam */
       to_network_layer((r.info);                /* transferă datele nivelului reţea */
       inc(frame_expected);                       /* incrementează număr de secvenţă */
     }
     s.ack=1-frame_expected;                                /* spune care frame este confirmat */
     to_physical_layer(&s);              /* nici unul dintre câmpuri nu este utilizat */
   }
 }
}
```

Fig. 3-12. Un protocol cu confirmare pozitivă şi retransmitere.



(a)

Tranziție	Cine rulează?	Cadru acceptat	Cadru emis	Spre nivelul rețea
		(cadru pierdut)		
0	—			—
1	R	0	A	Da
2	S	A	1	—
3	R	1	A	Da
4	S	A	0	—
5	R	0	A	Nu
6	R	1	A	Nu
7	S	(timp expirat)	0	—
8	S	(timp expirat)	1	—

(b)

Fig. 3-21. (a) Diagrama de stare pentru protocolul 3. (b) Tranzițiile.

Exemplu

- fiecare automat de protocol are două stări, iar canalul are patru stări.
- Există un total de 16 stări, nu toate accesibile din starea inițială. Stările inaccesibile nu sunt reprezentate
- Fiecare stare este etichetată cu trei caractere, *SRC, unde S este 0 sau 1, corespunzător cadrului* pe care transmiţătorul (S) încearcă să îl expedieze;
- .

Exemplu (cont)

- *R este de asemenea 0 sau 1, corespunzător* cadrului pe care receptorul (R) îl aşteaptă,
- *C este 0, 1, A sau vid (-), corespunzător stării canalului*
- Starea iniţială a fost aleasă ca fiind (000). (transmiţătorul tocmai a trimis cadrul 0, receptorul așteaptă cadrul 0 și cadrul 0 este actualmente pe canal.

Tranzitii

- Sunt nouă tipuri de tranziţii:
  1. Tranziţia 0 corespunde pierderii conţinutului canalului.
  2. Tranziţia 1 corespunde livrării corecte a pachetului 0 la receptor, receptorul schimbânduşi starea pentru a aştepta cadrul 1 și emiţând o confirmare. Tranziţia 1 include și livrarea pachetului 0 de către receptor spre nivelul reţea.
  3. Celelalte tranziţii sunt listate în fig. 3-21(b). Sosirea unui cadru cu suma de control eronată nu a fost pusă în evidenţă, deoarece nu trebuie schimbată starea (în protocolul 3).

Tranzitii (cont)

- Pe parcursul operării normale, tranziţiile 1, 2, 3 şi 4 sunt repetate în ordine, la nesfârşit.
- În fiecare ciclu sunt livrate două pachete, aducând transmiţătorul înapoi în starea iniţială, în care se încearcă transmiterea unui nou cadru cu numărul de secvenţă 0.

Tranzitii

- În cazul în care canalul pierde cadrul 0, el face o tranziţie din starea (000) în starea (00-). La final, transmiţătorului îi expiră intervalul de timp (tranziţia 7) şi sistemul revine în starea (ooo).

- Pierderea unei confirmări este mai complicată, necesitând două tranziţii, 7 şi 5, sau 8 şi 6, pentru a repara eroarea.

Proprietati

- Una din proprietăţile pe care protocolul cu număr de secvenţă pe 1 bit trebuie să le aibă este aceea că, indiferent de secvenţa de evenimente ce are loc, receptorul nu trebuie să livreze niciodată două pachete impare fără un pachet par intermediar şi invers.

Proprietati (cont)

- Din graful din fig. 3-21 se vede că această cerinţă poate fi formulată mai riguros astfel:
  - „nu trebuie să existe căi din starea iniţială care să conţină două apariţii ale tranziţiei 1 fără ca între ele să apară tranziţia 3 sau invers.”
- Din figură se vede că protocolul este corect în raport cu această cerinţă.

Cerinte

- O cerinţă similară este aceea că nu trebuie să existe căi pe care transmiţătorul să-şi schimbe starea de două ori (de exemplu din 0 în 1 şi înapoi în 0) în timp ce starea receptorului rămâne constantă.
- Dacă ar exista o astfel de cale, atunci, în secvența corespunzătoare de evenimente, două cadre ar fi iremediabil pierdute, fără ca receptorul să observe.
- Secvenţa de pachete livrată ar avea în ea o pierdere nedetectată a două pachete.

Interblocare

- O altă proprietate importantă a unui protocol este absenţa interblocărilor.

- O **interblocare (deadlock) este situaţia în care protocolul nu mai înregistrează nici un progres la transmitere (adică** livrare de pachete spre nivelul rețea), indiferent de secvența de evenimente produse.

Interblocare

- O interblocare este caracterizată de existenţa unei submulţimi de stări care este accesibilă din starea iniţială şi care are două proprietăţi:
  1. Nu există nici o tranziţie într-o stare din afara submulţimii de stări.
  2. În submulţimea de stări, nu există tranziţii care să determine continuarea transmiterii.

Interblocare(3)

- Odată ajuns în situaţia de interblocare, protocolul rămâne aici pentru totdeauna.

- Din nou, este uşor de văzut din graf că protocolul 3 nu are interblocări.

MULȚUMESC!

Nivelul Retea (1)

Liviu P. Dinu
ldinu@fmi.unibuc.ro
Center for Computational Linguistics
nlp.unibuc.ro

Introducere

- **Legatura de date asigura comunicarea corecta a cadrelor intre oricare doua noduri adiacente ale unei retele de calculatoare.**

- **Transferul datelor intre doua noduri neadiacente utilizeaza mai multe legaturi, puse cap la cap.**

Introducere (cont)

- **Rolul de releu este indeplinit de nivelul retea.**
- **Unitatea de date caracteristica nivel este pachetul.**
- **Principala functie a nivelului retea este *dirijarea* pachetelor transmise intre oricare doua noduri, pe cai convenabil alese.**

Nivelul Retea

- **Fiecare pachet receptionat de un nod este inspectat, determinindu-se nodul destinatar.**
- **Se alege apoi legatura convenabila, pachetul fiind transmis in continuare pe aceasta legatura.**
- **In cazul in care legatura este ocupata, pachetul este pus intr-o coada de asteptare asociata legaturii, urmind a fi transmis mai tirziu.**

Nivelul retea (2)

- **Nivelul retea trebuie sa evite congestionarea retelei, sau supraincarcarea sa.**

- **O alta functie este furnizarea unui mecanism uniform de adresare pentru nivelul transport (reglementarea comunicarii intre surse si destinatii aflate in retele diferite interconectate)**

Servicii

- **Servicile oferite nivelului transport pot fi grupate in doua categorii: orientate pe conexiuni si neorientate pe conexiuni.**
- **Primitivele orientate pe conexiuni permit:**
  - **- stabilirea sau desfiintarea unei conexiuni;**
  - **- transferul normal sau expeditiaal datelor;**
  - **- confirmarea datelor;**
  - **- resetarea unei conexiuni.**
- **Utilizarea unei conexiuni permite pastrarea ordinii pachetelor comunicate intre sursa si destinatar.**

Servicii (2)

- **Stabilirea si resetarea conexiunilor sint servicii cu confirmare, cuprinzind toate cele patru categorii de primitive (cerere, indicare, raspuns, confirmare).**

- **Transferul, expeditia si confirmarea datelor sint negociabile la stabilirea conexiunii de retea, la fel si calitatea serviciului, exprimata prin intirzierea transmisiei, rata de erori, costul si securitatea, etc.**

Servicii (3)

- **Primitivele neorientate pe conexiuni permit:**
- **- transferul individual al pachetelor;**
- **- obtinerea unor informatii despre transferul pachetelor catre o anumita destinatie (de exemplu procentajul de pachete livrate);**
- **- transmiterea unor rapoarte ale retelei catre nivelul transport, la producerea unor incidente.**

Datagrama si CV

- **Doua tehnici diferite:**
  - **circuit virtual (prin analogie cu circuitele fizice ale retelelor telefonice),**
  - **datagrama (prin analogie cu telegramele)**

CV (orientat pe conexiune)

- **Presupune transmiterea unui pachet initial de stabilire a circuitului, care este dirijat corespunzator intre nodul sursa si nodul destinatar.**

- **Aceeasi ruta este folosita de toate celelalte pachete transmise pe acelasi circuit virtual.**

CV (2)

- **Fiecare pachet contine in antet numarul circuitului logic, iar fiecare comutator pastreaza un tabel cu toate circuitele virtuale care il traverseaza.**

- **La receptia unui pachet, pe baza numarului circuitului virtual se determina o intrare a tabelului, in care este specificata legatura pe care pachetul va fi transmis.**

CV (3)

- **La stabilirea unui circuit virtual, nodul sursa alege un numar de circuit disponibil in acel nod, care poate insa sa coincida cu numarul ales de nodul vecin pentru un alt circuit virtual.**
- **Pentru a evita ambiguitatile, un circuit virtual este renumerotat in fiecare nod prin care trece, corespondenta intre vechea si noua numerotare find pastrata in tabelul circuitelor virtuale**
- **Renumerotarea implica modificarea numarului circuitului virtual din antet, in fiecare nod traversat de pachet.**

Datagrama (neorientat pe conexiune)

- **Fiecare pachet este dirijat independent de predecesoarele sale. Pachetul trebuie sa contina adresa completa a destinatarului (care ocupa mai mult spatiu decit numarul circuitului virtual).**

- **Fiecare comutator are un tabel de dirijare, indicind legatura pe care trebuie transmis pachetul in functie de adresa destinatarului.**

Datagrama (2)

- **Sunt necesare si in cazul circuitelor virtuale, pentru a determina ruta pachetelor de stabilire a circuitelor.**

- **La receptia unui pachet, comutatorul inspecteaza adresa dest., determina intrarea corespunzatoare din tabela de dirijare si de aici legatura pe care trebuie transmis in continuare pachetul.**

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

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

Algoritmi de dirijare

- **Principala functie a unei retele cu comutare de pachete este preluarea pachetelor de la nodurile surse si livrarea lor nodurilor destinatare.**

- **In acest scop, se alege o cale de transmitere a fiecarui pachet prin retea.**

Algoritmi de dirijare (cont)

- **Uzual, exista mai multe cai posibile, selectarea uneia din ele urmarind satisfacerea unor cerinte (uneori contradictorii) ale utilizatorilor si ale administratorilor retelei:**
  1. **- transferul sa se faca corect si cu operativitate;**
  2. **- sa nu existe utilizatori defavorizati;**
  3. **- nodurile si legaturile sa fie folosite eficient.**

Principiul optimalităţii

- **Principiul optimalităţii (optimality principle) stabileşte că dacă ruterul** *J este pe calea optimă de la ruterul I către ruterul K, atunci calea optimă de la J la K este pe aceeaşi* rută.
- Pentru a vedea aceasta, să notăm cu *r1 partea din cale de la I la J, iar cu r2 restul rutei.*
- *Dacă ar* exista o rută mai bună decât *r2 de la J la K, ea ar putea fi concatenată cu r1 şi ar îmbunătăţi ruta de la I la K, ceea ce ar contrazice presupunerea că r1r2 este optimală.*

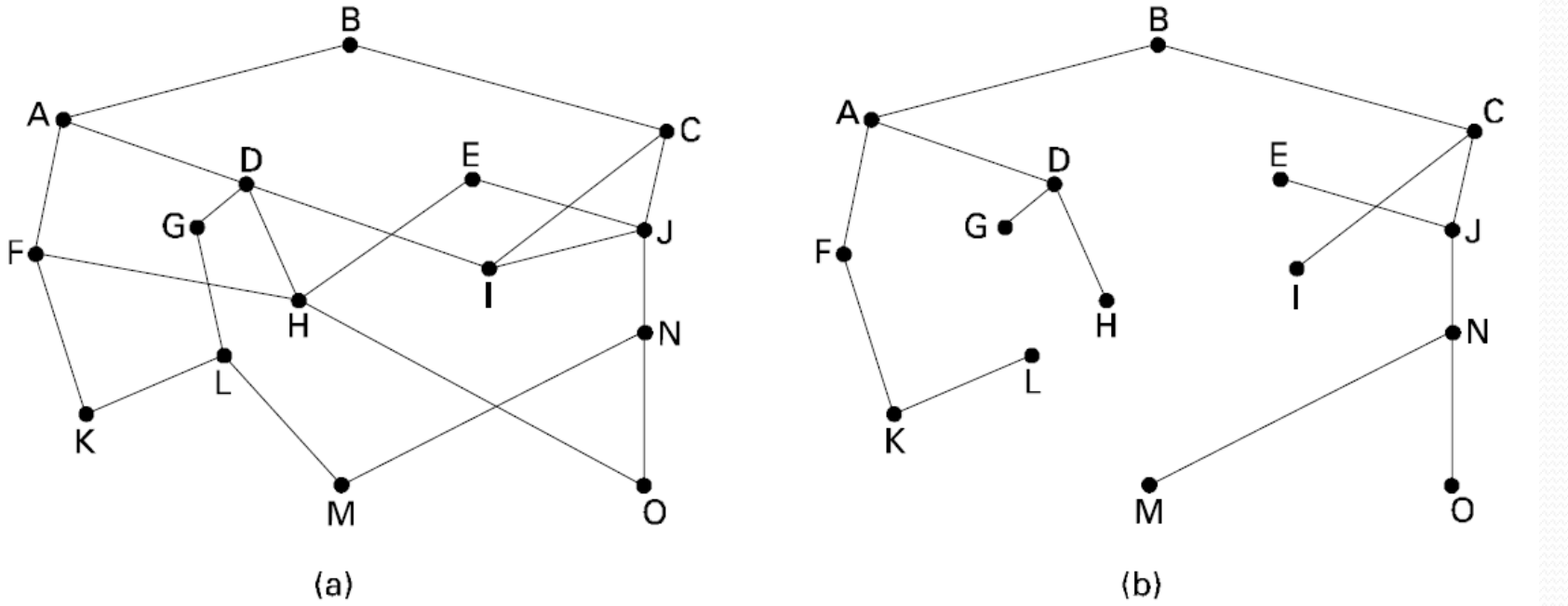


Fig. 5-6. (a) O subrețea. (b) Un arbore de scufundare pentru ruterul *B*.

Clasificari

- **In functie de adaptarea la conditiile de trafic:**

1. **statica (atunci cind continutul tabelelor de dirijare este fix),**
2. **adaptiva (atunci cind continutul tabelelor de dirijare se modifica in functie de traficul curent sau de topologia retelei).**

Clasificari (2)

- **Dupa locul unde se realizeaza calculele:**
  1. **centralizata (un algoritm global utilizeaza informatii despre intreaga retea pentru a lua decizii optime de dirijare),**
  2. **izolata(care utilizeaza in fiecare nod informatii disponibile local)**
  3. **distribuita (care utilizeaza o combinatie de informatii locale si globale).**

Clasificari (3)

- **Dupa obiectivele urmarite:**
  1. **algoritmii care asigura transmiterea pe calea cea mai scurta, pentru fiecare pereche sursa-destinatie.**
  2. **Algoritmii care minimizeaza intirzierea medie globala de transmitere a pachetelor (considerind toate perechile sursa-destinatie din retea).**

Clasificari (4)

- **Multe retele operationale includ algoritmi din prima categorie.**

- **Cei din a doua categorie conduc la o dirijare bifurcata, mai greu de gestionat practic. Sunt folositi cu predilectie in proiectarea topologica a retelelor de calculatoare si mai putin ca metode efective de dirijare.**

Calea cea mai scurta

- **Modelul topologic al unei retele este un graf in care nodurile corespund comutatoarelor de pachete, iar muchiile corespund liniilor de comunicatie.**

- **Asociind fiecarei muchii o lungime, se poate calcula calea cea mai scurta intre oricare doua noduri, deci cea mai indicata pentru dirijarea pachetelor intre nodurile respective (algoritmul lui Dijkstra).**

Calea cea mai scurta (2)

- **Lungimea poate avea diverse semnificatii. Daca toate liniile au lungimea unu, gasim caile cu numar minime de noduri intermediare.**

- **Lungimea poate fi distanta geografica intre noduri, costul comunicatiei, intirzierea medie masurata etc.**

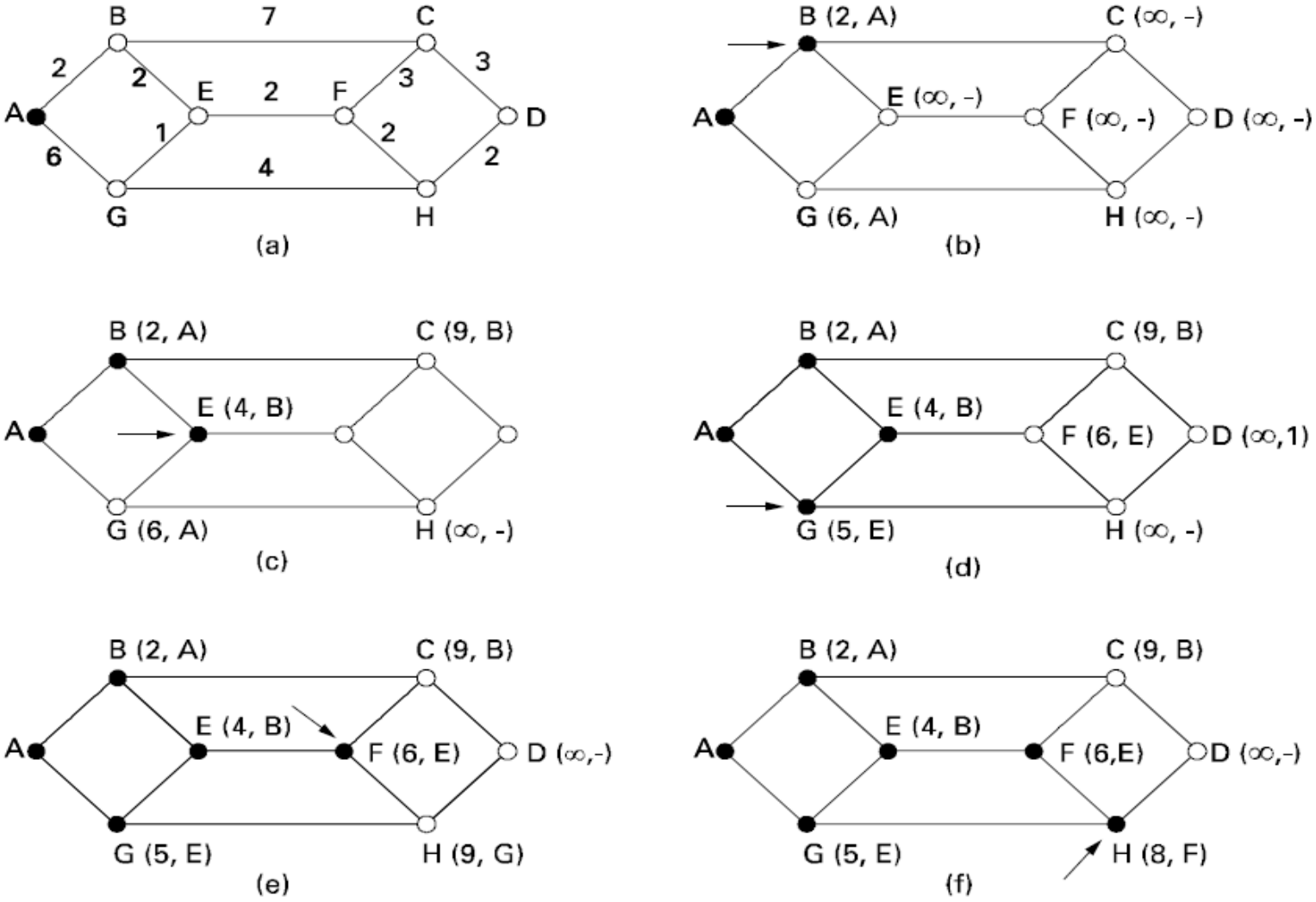


Fig. 5-7. Primii cinci pași folosiți în calcularea celei mai scurte căi de la A la D. Săgețile indică nodul curent.

Algoritmul lui Dijkstra

- **Algoritmul lui Dijkstra gaseste caile cele mai scurte de la o sursa la toate celelalte noduri.**

- **El trebuie sa dispuna de informatii topologice generale asupra retelei: listele nodurilor si legaturilor,costurile asociate legaturilor.**
- **Prin natura sa el este centralizat**

- **Algoritmul este iteratiasi calculeaza la fiecare iteratie ë cea mai scurta cale de la sursa la un nod al retelei.**

Dirijare cu vectori distanţă

- Rețelele moderne de calculatoare folosesc de obicei algoritmi dinamici de dirijare în locul celor statici:
  - Algoritmul de dirijare cu vectori distanţă
  - Algoritmul de dirijare bazat pe starea legăturilor.

Distance vector routing

- Algoritmul de **dirijare cu vectori distanţă (distance vector routing) presupune că fiecare ruter** menţine o tabelă (de exemplu un vector) care păstrează cea mai bună distanţă cunoscută spre fiecare destinaţie şi linia care trebuie urmată pentru a ajunge acolo.
- Aceste tabele sunt actualizate prin schimbul de informaţii între nodurile vecine.
- Algoritmul de dirijare cu vectori distanţă este cunoscut şi sub alte nume:algoritmul distribuit de dirijare **Bellman-Ford și algoritmul Ford-Fulkerson.**

Metrici

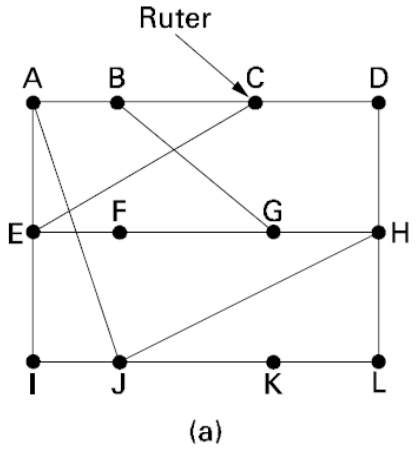
- În dirijarea pe baza vectorilor distanţă, fiecare ruter păstrează o tabelă de dirijare conţinând câte o intrare pentru fiecare ruter din subreţea.
- Această intrare are două părţi: linia de ieşire preferată care se foloseşte pentru destinaţia respectivă şi o estimare a timpului sau distanţei până la acea destinaţie.
- Metrica folosită poate fi numărul de salturi, întârzierea în milisecunde, numărul total de pachete care aşteaptă în cozi de-a lungul căii, sau ceva asemănător.

Exemplu

- Presupunem că se foloseşte metrica întârzierilor şi că ruterul cunoaşte întârzierea spre fiecare dintre vecinii săi.
- O dată la fiecare *T msec fiecare ruter trimite spre fiecare vecin o listă a* estimărilor proprii spre fiecare destinaţie.
- El recepţionează o listă similară de la fiecare vecin

Exemplu

- Presupunem că una dintre aceste tabele tocmai a sosit de la vecinul *X*, *cu Xi fiind estimarea* lui *X despre cât timp este necesar pentru a ajunge la ruterul i.*
- *Dacă ruterul ştie că întârzierea spre X* este *m msec, el ştie de asemenea că poate atinge ruterul i trecând prin X în Xi + m msec.*
- *Făcând* aceste calcule pentru fiecare vecin, un ruter poate stabili care estimare pare a fi cea mai bună, pentru a folosi această estimare, împreună cu linia corespunzătoare în noua tabelă de dirijare.



(a)

Noua întârziere estimată de la J

La	A	I	H	K	Noua întârziere estimată de la J	Linia
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	-
K	24	22	22	0	6	K
L	29	33	9	9	15	K

întîrzierile

JA	JI	JH	JK
8	10	12	6

Vectorii primiți de la cei patru vecini ai lui J

Noua tabelă de rutare pentru J

(b)

Fig. 5-9. (a) O subrețea. (b) Intrări de la *A*, *I*, *H* și *K* și noua tabelă de dirijare pentru *J*.

Exemplu. Descriere

- Partea (a) prezintă o subreţea.
- Primele patru coloane din partea (b) conţin vectorii de întârzieri primiţi de la vecinii ruterului *J*.
- *A afirmă că are* 12 msec întârziere spre *B, 25 msec întârziere spre C, 40 msec întârziere spre D etc.*
- *Presupunem că J şi-a măsurat sau estimat întârzierea faţă de vecinii săi A, I, H şi K, obţinând valorile 8, 10, 12 şi 16* ms, respectiv.

Calcul

- Să vedem cum calculează *J noua cale spre ruterul G.*
- *El ştie că poate ajunge la A în 8 msec şi A* pretinde că este în stare să ajungă la *G în 18 msec, astfel încât J poate conta pe o întârziere de 26* msec spre G dacă dirijează pachetul spre *A.*
- *Similar, el calculează întârzierea spre G prin I, H, K ca* fiind 41 (31 + 10), 18 (6 + 12) şi 37 (31 + 6) respectiv.
- Cea mai bună valoare este 18, aşa că va crea o intrare în tabela de dirijare cu întârzierea către *G de 18 msec şi ruta de urmat trecând prin H.*
- *Aceleaşi* calcule se fac pentru toate destinaţiile

MULȚUMESC!

Nivelul Retea (2)

Liviu P. Dinu

ldinu@fmi.unibuc.ro

Center for Computational Linguistics

nlp.unibuc.ro

Dirijarea centralizata

- **Varianta centralizata a algoritmului drumurilor minime (Floyd) utilizeaza un tablou A al distantelor minime**
- **A[i][j] este distanta minima de la nodul i la nodul j.**

- **Initial:**
  - **A[i][j] = l[i][j] pentru orice i si j.**

Dirijarea centralizata

- **Calculul drumurilor minime se face iterativ.**
- **La iteratia k, A[i][j] va avea ca valoare cea mai buna distanta intre i si j, pe cai care nu contin noduri numerotate peste k (exceptind i si j).**
- **Se utilizeaza relatia:**
  - **A[i][j] /la pas k/ = min (A[i][j] ,A[i][k] + A[k][j]) /pas k-1/**

Dirijarea centralizata

- **Deoarece:**

A[i][k] /la pas k = A[i][k] /la pas k-1/ si

A[k][j] /la pas k = A[k][j] /la pas k-1/

nici o intrare avind unul din indici egal cu k nu se modifica la iteratia k.

- **Drept urmare, calculul se poate realiza cu o singura copie a tabloului A.**

Dirijarea centralizata

- **Tabloul A contine,de data aceasta, tabelele de dirijare ale tuturor nodurilor:**

- **V[i][j] este nodul vecin lui i, pe calea cea mai scurta de la i la j.**
- **Linia i a tabloului este tabelul de dirijare al nodului i,**
- **Linia i a tabloului Á este tabelul costurilor minime corespunzatoare.**

Dirijare Centralizata

- **Algoritmul poate fi folosit pentru adaptarea dirijarii la modificarile de trafic sau de topologie (caderea unor legaturi sau noduri).**
-
- **Un centru de control al dirijarii primeste de la comutatoarele de pachete rapoarte periodice despre starea locala, calculeaza noile tabele de dirijare pentru fiecare comutator si le transmite acestora.**

Deficiente

- **Deficientele acestei metode**
  1. **- vulnerabilitatea retelei,dependenta de functionarea centrului de control (se recurge la dublarea lui);**
  2. **- supraincarcarea traficului prin transmiterea rapoarelor si a tabelelor de dirijare;**
  3. **- utilizarea in noduri,in anumite perioade, a unor tabele necorelate, datorita receptiei la momente de timp distincte a noilor tabele.**

Dirijarea izolata

- **Pachetul receptionat de nod este plasat in coada cea mai scurta. (O varianta ia in consideratie lungimea cozilor anumitor linii, selectate conforme cailor celor mai scurte)**

- **Dirijarea delta este o combinatie a politicilor izolata si centralizata. Comutatoarele trimit rapoarte unui centru de control care calculeaza cele mai bune ë rute.**

Dirijarea izolata

- **Caile sunt echivalente daca lungimile lor difera intre ele cu o valoare mai mica decit un delta specificat.**
- **In algoritmul inundarii, pachetul este transmis pe fiecare legatura, cu exceptia celei de origine.**
- **Copiile sint distruse dupa traversarea unui anumit numar de noduri.**
- **Desi nepractic, algoritmul este folosit in aplicatii militare (datorita robustetii sale) sau in comparatii de performanta cu alte tehnici (deoarece are un timp de intirziere minim).**

Dirijarea distribuita

- **Varianta modificata a algoritmului lui Dijkstra care calculeaza drumurile minime de la toate nodurile catre o anumita destinatie.**
- **Conduce in mod natural la o varianta descentralizata**
- **Algoritmul este convergent,asigurind gasirea drumurilor minime intr-un numar finit de pasi**

Dirijare distribuita

- **Poate fi utilizat doar pentru datagrame, deoarece:**
  1. **sint posibile modificari ale cailor, pe durata transmiterii pachetelor, astfel incit pachetele trimise pe o cale pot ajunge pe o alta cale;**
  2. **- pe durata intervalului de convergenta, algoritmul nu evita producerea buclelor (trecerea pachetelor de mai multe ori prin acelasi nod).**

Dirijare distribuita

- **Pentru a putea realiza calculele, fiecare nod al retelei pastreaza trei tablouri:**

- **C - tabloul distantelor; C[d][v] este lungimea (sau costul) drumului de la nodul curent la nodul destinatar,prin nodul vecin v;**

- **D - tabloul distantelor minime; D[d] este lungimea drumului minim de la nodul curent la nodul destinatar d;**

Dirijare distribuita

- **V - tabloul de dirijare; V[d] este nodul vecin prin care se transmit datele,pe drumul minim,spre destinatarul d.**
- **Oricare nod trebuie sa poata detecta modificarile lungimii legaturilor sale.**
- **La producerea unei modificari,fiecare nod care o sesizeaza actualizeaza tabloul distantelor minime si de dirijare.**

Dirijare distribuita

- **Toate modificarile tabelei de dirijare sint comunicate vecinilor, in forma unor mesaje de forma (s,d,Dsd), unde Dsd este distanta minima de la s la d.**

- **In cazul adaugarii unei noi legaturi la retea, noului vecin i se transmite toata tabela de dirijare.**
- **Mesajele mentionate pot fi transmise izolat sau grupate in aceeasi unitate.**

Dirijare distribuita

- **Primirea unui mesaj de forma (s,d,Dsd) declanseaza in receptor procesul de calcul si actualizare a tabelei de dirijare, modificarile fiind transmise, la rindul lor, tuturor vecinilor.**

- **Procesul este convergent, la un moment dat, schimbul de mesaje si actualizarea tabelelor incheindu-se.**

Dirijare distribuita

- **Algoritmul trateaza trei evenimente distincte:**
  1. **- adaugarea unei noi legaturi;**
  2. **- sesizarea modificarii lungimii unei linii;**
  3. **- primirea unui mesaj de control de la un nod vecin.**

Dirijare distribuita

- **algoritmul creeaza probleme prin dirijarea pachetelor in bucla, intre noduri vecine: daca A trimite pachetele catre C prin B si daca legatura B-C se defecteaza, B va trimite pachetele catre C prin A, care are o estimare mai buna.**
- **Dar A va trimite pachetele catre C lui B, recirculind pachetele in bucla.**

Dirijare distribuita

- **Pentru a ocoli acest neajuns se foloseste principiul optimalitatii:**
  - **daca B este pe ruta optima de la A la C, atunci calea cea mai buna de la B la C este inclusa in prima.**
  - **Deci, caile catre C formeaza un arbore a carui cunoastere permite evitarea buclelor infinite.**

Dirijarea ierarhica

- **Se utilizeaza pentru retele de mari dimensiuni la care tabelele de dirijare ar fi voluminoase.**

- **Comutatoarele sint grupate in regiuni,fiecare comutator cunoscind detaliat caile din regiunea proprie,dar necunoscind structura interna a altor regiuni.**

Dirijarea ierarhica

- **Doua regiuni sint legate prin conectarea unui anumit nod din prima regiune cu un anumit nod din a doua regiune.**

- **Tabela de dirijare se poate reduce,ea avind cite o intrare pentru fiecare nod din regiunea proprie si cite o intrare pentru fiecare din celelalte regiuni.**

Dirijarea cu difuzare (broadcast)

- **Dirijarea unui pachet catre toate celelalte noduri se poate face prin tehnica inundarii. Pentru a evita degradarea performantelor retelei se poate recurge la alti algoritmi.**

- **In dirijarea multidestinatie, pachetul contine o lista cu adresele destinatarilor.**

Dirijarea cu difuzare (2)

- **Cind pachetul ajunge la un comutator, acesta determina, pe baza adreselor, pe ce linii trebuie sa transmita in continuare pachetul, partitionind totodata lista adreselor intre duplicatele transmise pe aceste linii.**

- **In dirijarea cu difuzare se poate utiliza ca traseu orice arbore de acoperire minimal.**

Dirijarea cu difuzare (3)

- **Ca alternativa, algoritmul se poate baza pe urmarirea cailor inverse:**
  - **cind un pachet ajunge la un comutator, daca el a fost receptionat pe legatura folosita de obicei pentru a transmite catre sursa acestui pachet, atunci el a sosit pe calea cea mai scurta si este de obicei prima copie receptionata de comutator.**
- **Ca urmare,ea este acceptata, iar comutatorul o transmite in continuare pe fiecare linie cu exceptia celei pe care a sosit.**

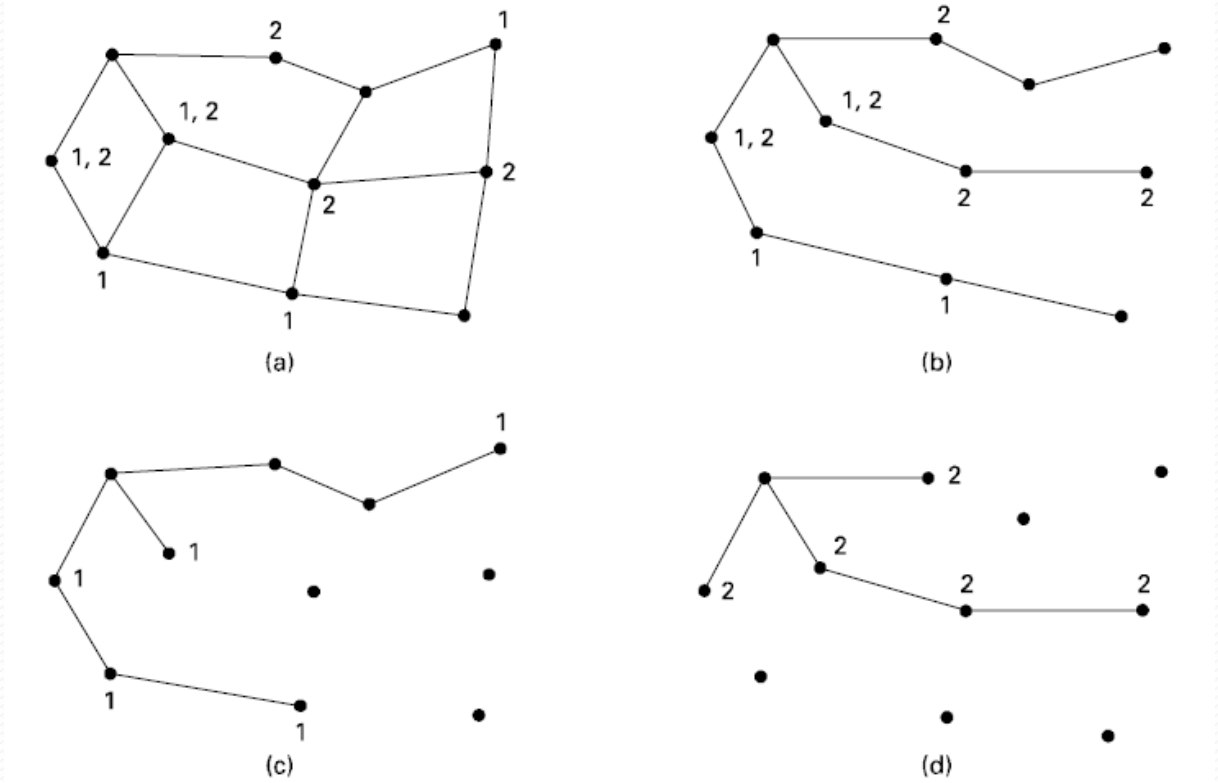


Fig. 5-17. (a) O rețea. (b) Un arbore de acoperire pentru cel mai din stânga ruter. (c) Un arbore multicast al grupului 1. (d) Un arborele multicast al grupului 2.

Algoritmi de evitare a congestionarii

1. ***Prealocarea zonelor tampon:***
   - **Este aplicabila circuitelor virtuale.**
   - **Consta in rezervarea uneia sau mai multor zone tampon in fiecare nod intermediar, la deschiderea circuitului.**
   - **In lipsa de spatiu, se alege o alta cale sau se rejecteaza cererea de stabilire a circuitului.**

Evitarea congestionarii

1. ***Distrugerea pachetelor:***
   - **Daca nu exista spatiul necesar memorarii, pachetul receptionat de un nod este ignorat.**
   - **Deoarece astfel se pot ignora pachete de confirmare, care ar duce la eliberarea spatiului ocupat de pachetele confirmate, se mentine cel putin un tampon de receptie pentru fiecare linie, permitindu-se inspectarea pachetelor primite.**
   - **Se poate limita numarul zonelor tampon de transmisie ale fiecarei linii.**

Evitarea congestionarii

3. ***Pachete de permisiune.***
   - **Se initializeaza reteaua cu pachete de permisiune (in numar fix).**
   - **Cind un nod vrea sa transmita, el captureaza un pachet de permisiune si trimite in locul lui pachete de date.**
   - **Receptorul regenereaza pachetul de permisiune.**

Evitarea congestionarii

- **Se garanteaza astfel ca numarul maxim de pachete nu depaseste numarul de pachete de permisiune, fara a se asigura distribuirea lor conform necesitatilor nodurilor.**
- **In plus, pierderea pachetelor de permisiune conduce la scaderea capacitatii retelei.**

4. ***Pachete de soc.***

- **Sunt transmise de comutatoare surselor de date pentru a micsora rata de generare a pachetelor.**

Evitarea blocarii definitive.

- Blocarea reprezinta o situatie limita a unei retele congestionate, cind lipsa de spatiu impiedica transmiterea vreunui pachet.
  - O solutie de evitare a blocarii definitive este utilizarea in fiecare nod a ***m+1*** zone tampon, **m** fiind lungimea maxima a cailor retelei.
  - Un pachet sosit de la calculatorul gazda local este acceptat in zona 0.
  - In urmatorul nod trece in 1, apoi in 2 s.a.m.d.

Evitarea blocarii definitive (2)

- Zona "m" a unui nod poate fi goala, poate contine un pachet pentru gazda locala, care este livrat, sau are un pachet pentru un nod distant, care este distrus.

- In toate cazurile zona "m" se elibereaza, putind avansa un pachet din zona "m-1",apoi "m-2" etc.

Evitarea blocarii definitive (3)

- O alta varianta pastreaza pentru fiecare pachet o informatie de vechime.
- La comunicarea dintre doua noduri A si B putem intilni situatiile urmatoare (presupunem ca A are de transmis lui B un pachet mai vechi decit B catre A):

Evitarea blocarii definitive (4)

- B are un tampon liber si poate primi cel mai vechi pachet al lui A catre B;
- B nu are un tampon liber, dar are un pachet pentru A si poate primi, prin schimb, cel mai vechi pachet al lui A catre B;
- B nu are nici un tampon liber si nici un pachet catre A; in acest caz, B este fortat sa transmita lui A un pachet la alegere si sa primeasca cel mai vechi pachet al lui A catre B

MULȚUMESC!

Nivelul Retea (3)

Liviu P. Dinu
ldinu@fmi.unibuc.ro
Center for Computational Linguistics
nlp.unibuc.ro

INTERCONECTAREA REŢELELOR

- Scopul interconectării acestor reţele este de a permite utilizatorilor din orice reţea să comunice cu utilizatorii celorlalte reţele şi de asemenea de a permite unui utilizator din orice reţea să acceseze date pe orice reţea.
- Realizarea acestui scop înseamnă trimiterea pachetelor dintr-o reţea în alta.
- Cum reţelele diferă deseori în puncte esenţiale, transmiterea pachetelor dintr-o reţea în alta nu este întotdeauna uşoară,

Element	Câteva posibilități
Serviciu oferit	Orientat pe conexiuni față de cel fără conexiune
Protocol	IP, IPX, SNA, ATM, MPLS, AppleTalk etc.
Adresare	Plată (802) opusă celei ierarhice (IP)
Trimitere multiplă	Prezentă sau absentă (de asemenea, difuzarea totală)
Dimensiune pachet	Fiecare rețea își are propriul maxim
Calitatea serviciului	Poate fi prezentă sau absentă; multe tipuri diferite
Tratarea erorilor	Livrare fiabilă, ordonată sau neordonată
Controlul fluxului	Fereastră glisantă, controlul ratei, altele sau nimic
Controlul congestiei	Algoritmul găleții găurite, algoritmul găleții cu jeton, RED, pachete de șoc etc.
Securitate	Reguli de secretizare, criptare etc.
Parametri	Diferite limitări de timp, specificări ale fluxului etc.
Contabilizare	După timpul de conectare, după pachet, după octeți sau fără.

Fig. 5-43. Câteva din multele moduri în care pot diferi retelele.

Probleme

- Când pachetele trimise de o sursă dintr-o reţea trebuie să tranziteze una sau mai multe reţele străine înainte de a ajunge în reţeaua destinaţie (care, de asemenea, poate fi diferită faţă de reţeaua sursă), pot apărea multe probleme la interfeţele dintre reţele.
- Pentru început, atunci când pachetele dintr-o reţea orientată pe conexiuni trebuie să tranziteze o reţea fără conexiuni, poate interveni o reordonare a acestora, lucru la care emiţătorul nu se aşteaptă şi căruia receptorul nu este pregătit să-i facă faţă.

Probleme

- Vor fi necesare frecvente conversii de protocol, care pot fi dificile dacă funcţionalitatea cerută nu poate fi exprimată.
- Vor fi necesare conversii de adresă, ceea ce poate cere un sistem de catalogare.
- Trecerea pachetelor cu trimitere multiplă printr-o reţea care nu oferă trimitere multiplă necesită generarea de pachete separate pentru fiecare destinaţie.
- Diferența dintre dimensiunile maxime ale pachetelor folosite de diferite rețele poate produce mari neplăceri.

Circuite virtuale concatenate

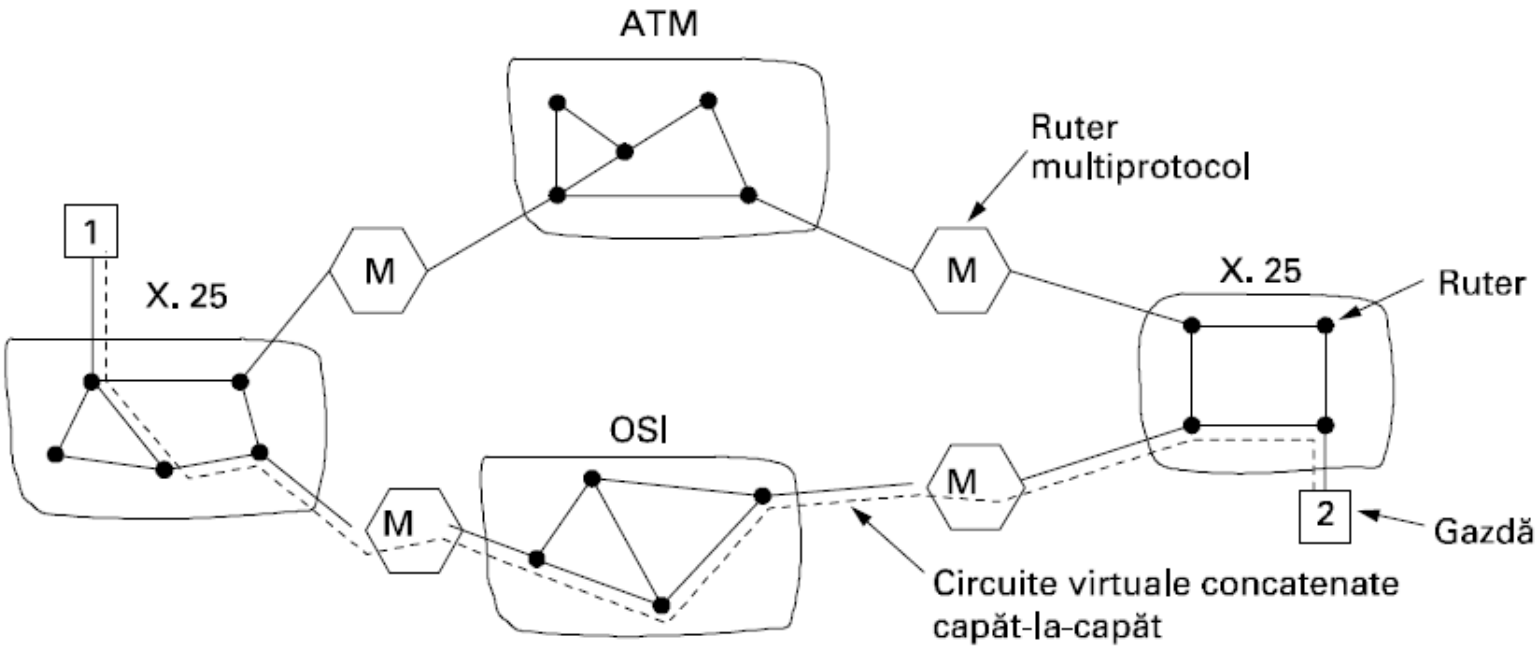


Fig. 5-45. Interconectarea rețelelor folosind circuite virtuale concatenate.

CVC

- În modelul circuitelor virtuale concatenate, o conexiune către o gazdă dintr-o reţea îndepărtată este stabilită într-un mod similar cu modul în care sunt stabilite conexiunile în cazul normal.
- Subreţeaua observă că destinaţia este îndepărtată şi construieşte un circuit virtual spre ruterul aflat cel mai aproape de rețeaua destinație.
- Apoi construiește un circuit virtual de la acel ruter la o **poartă externă (un ruter multiprotocol).**

CVC(2)

- **Această poartă înregistrează existenţa circuitului** virtual în tabelele sale şi continuă să construiască un alt circuit virtual către un ruter din următoarea subreţea.
- Acest proces continuă până când se ajunge la gazda destinaţie.
- O dată ce pachetele de date încep să circule de-a lungul căii, fiecare poartă retransmite pachetele primite, făcând, după nevoie, conversia între formatele pachetelor şi numerele de circuite virtuale.

CVC (3)

- Evident, toate pachetele de date trebuie să traverseze aceeaşi secvenţă de porţi, deci ajung în ordine.
- Caracteristica esenţială a acestei abordări este că se stabileşte o secvenţă de circuite virtuale de la sursă, prin una sau mai multe porţi, până la destinaţie.
- Fiecare poartă menţine tabele spunând ce circuite virtuale o traversează, unde vor fi dirijate şi care este numărul noului circuit virtual.

Interconectarea reţelelor fără conexiuni

- Modelul nu necesită ca toate pachetele care aparţin unei conexiuni să traverseze aceeaşi secvenţă de porţi
- O decizie de dirijare este luată separat pentru fiecare pachet, eventual în funcţie de traficul din momentul în care este trimis pachetul.
- Această strategie poate utiliza rute multiple şi atinge astfel o capacitate mai mare decât modelul circuitelor virtuale concatenate.
- Nu există nici o garanţie că pachetele ajung la destinaţie în ordine, presupunând că vor ajunge.

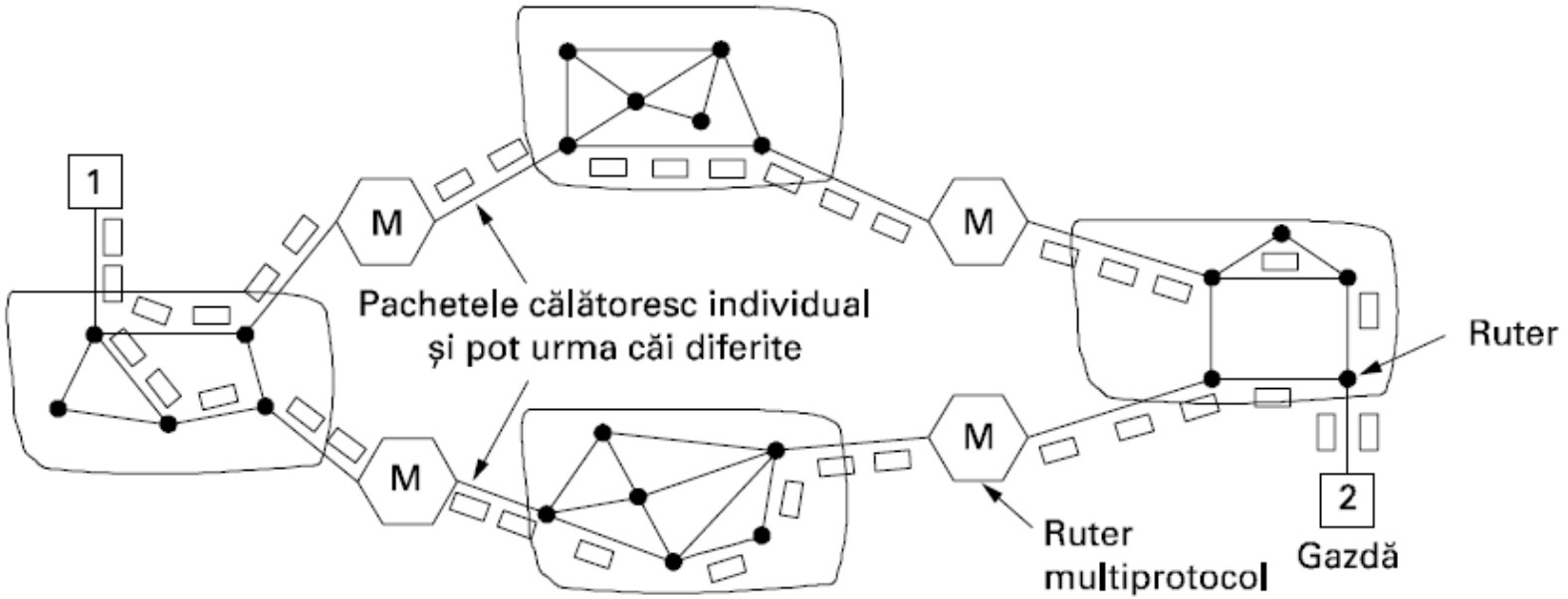


Fig. 5-46. O interconectare fără conexiuni.

Fragmentarea

- Fiecare reţea impune o anumită dimensiune maximă pentru pachetele sale.
- Aceste limite au diferite cauze, printre care:
- 1. Hardware
- 2. Sistemul de operare
- 3. Protocoale
- 4. Concordanţa cu unele standarde (inter)naţionale.
- 5. Dorinţa de a reduce la un anumit nivel retransmisiile provocate de erori.
- 6. Dorinţa de a preveni ocuparea îndelungată a canalului de către un singur pachet.

- O problemă evidentă apare când un pachet mare vrea să traverseze o rețea în care dimensiunea maximă a pachetului este prea mică.
- Soluție: permiterea porților să spargă pachetele în **fragmente,** trimițând fiecare pachet ca un pachet inter-rețea separat.

- Două strategii opuse pentru reconstituirea pachetului original din fragmente.
- Prima strategie este de a face fragmentarea cauzată de o rețea cu „pachete mici” transparentă pentru toate rețelele succesive prin care pachetul trebuie să treacă pe calea către destinaţia finală.
- Rețeaua cu pachete mici are porți (cel mai probabil, rutere specializate) către celelalte rețele.

- Când un pachet supradimensionat ajunge la poartă, poarta îl sparge în fragmente.
- Fiecare fragment este adresat aceleiași porți de ieșire, unde piesele sunt recombinate.
- În acest mod, trecerea printr-o rețea cu pachete mici a devenit transparentă

Probleme

- Poarta de ieşire trebuie să ştie când a primit toate piesele, aşa încât în fiecare pachet trebuie inclus fie un câmp contor, fie un bit „sfârşit-de-pachet„.
- Un alt motiv este că toate pachetele trebuie să iasă prin aceeaşi poartă.
- Performanţele se pot degrada nepermiţând ca unele pachete să urmărească o cale către destinaţia finală şi alte pachete o cale diferită.
- O ultimă problemă este timpul suplimentar necesar pentru

reasamblarea şi apoi refragmentarea repetată a unui pachet mare care traversează o serie de reţele cu pachete mici.

Protocolul IP

- O datagramă IP constă dintr-o parte de antet și o parte de text.
- Antetul are o parte fixă de 20 de octeţi şi o parte opţională cu lungime variabilă
- Este transmis în ordinea *big endian (cel mai semnificativ primul): de la stânga la dreapta*, începând cu bitul cel mai semnificativ al câmpului *Versiune*.

IP

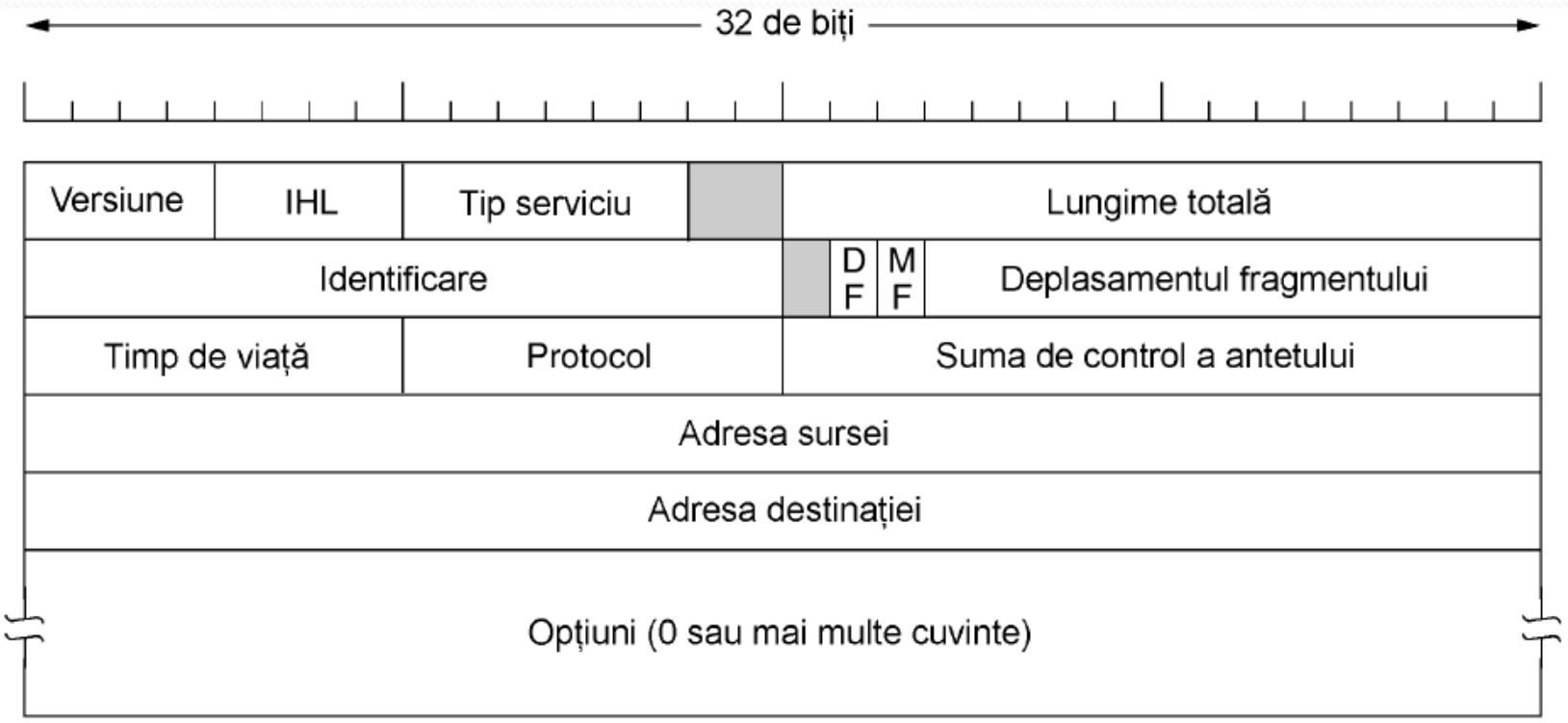


Fig. 5-53. Antetul IPv4 (protocolul Internet).

IP

- *Versiune memorează cărei versiuni de protocol îi aparţine datagrama. Prin includerea* unui câmp versiune în fiecare datagramă, devine posibil ca tranziţia între versiuni să dureze ani de zile, cu unele maşini rulând vechea versiune, iar altele rulând-o pe cea nouă
- *IHL, este pus la dispoziţie* pentru a spune cât de lung este antetul, în cuvinte de 32 de octeţi. Valoarea minimă este 5, care se aplică atunci când nu sunt prezente opţiuni. Valoarea maximă a acestui câmp de 4 biţi este 15

- *Tip serviciu este unul dintre puținele câmpuri care și-a schimbat sensul (oarecum) dea* lungul anilor. A fost şi este în continuare menit să diferenţieze diferitele clase de servicii.
- *Precedență reprezintă prioritatea, de la 0 (normal) la 7 (pachet* de control al reţelei). Cei trei biţi indicatori permiteau calculatorului gazdă să specifice ce îl afectează cel mai mult din mulţimea {Delay (Întârziere),Throughput (Productivitate), Reliability (Fiabilitate)}

IP

- *Lungimea totală include totul din datagramă - atât antet cât şi date. Lungimea maximă este de* 65535 octeţi.
- *Identificare este necesar pentru a permite* gazdei destinaţie să determine cărei datagrame îi aparţine un nou pachet primit. Toate fragmentele unei datagrame conţin aceeaşi valoare de *Identificare*.
- un bit nefolosit şi apoi două câmpuri de 1 bit.
- *DF însemnă Don't Fragment (A nu se* fragmenta).
- *MF însemnă More Fragments (mai urmează fragmente). Toate fragmentele, cu excepţia ultimului,* au acest bit activat.

IP

- *Deplasamentul fragmentului spune unde este locul fragmentului curent în cadrul datagramei.*
- *Timp de viaţă este un contor folosit pentru a limita durata de viaţă a pachetelor. Este* prevăzut să contorizeze timpul în secunde, permiţând un timp maxim de viaţă de 255 secunde
- *Protocol spune cărui proces de transport trebuie să predea datagrama*
- *Suma de control a antetului verifică numai antetul. O astfel de sumă de control este utilă pentru* detectarea erorilor generate de locaţii de memorie proaste din interiorul unui ruter

- *Adresa sursei şi Adresa destinaţiei indică numărul de reţea şi numărul de gazdă*
- *Opţiuni a fost proiectat pentru a oferi un subterfugiu care* să permită versiunilor viitoare ale protocolului să includă informaţii care nu sunt prezente în proiectul original, pentru a permite cercetătorilor să încerce noi idei şi pentru a evita alocarea unor biţi din antet pentru informaţii folosite rar.
- Opţiunile sunt de lungime variabilă

IP. Optiuni

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

Fig. 5-54. Unele dintre opțiunile IP.

IP

- *Securitate menţionează cât de secretă este informaţia*
- *Dirijare strictă de la sursă dă calea completă de la sursă la destinaţie ca o secvenţă de* adrese IP. Datagrama este obligată să urmărească această cale precisă
- *Dirijare aproximativă de la sursă cere ca pachetul să traverseze o listă specificată de* rutere şi în ordinea specificată, dar este permisă trecerea prin alte rutere pe drum
- *Înregistrează calea indică ruterelor de pe cale să-şi adauge adresele IP la câmpul opţiune.*
- *Amprentă de timp:* fiecare ruter înregistrează şi o amprentă de timp de 32 de biţi

MULȚUMESC!

Nivelul Transport (1)

Liviu P. Dinu

ldinu@fmi.unibuc.ro

Center for Computational Linguistics

nlp.unibuc.ro

Descriere

- Sarcina NT este de a transporta date de la maşina sursă la maşina destinaţie într-o manieră sigură şi eficace din punctul de vedere al costurilor, independent de reţeaua sau reţelele fizice utilizate.

- Scopul principal al nivelului transport este de a oferi servicii eficiente, sigure şi ieftine utilizatorilor, în mod normal procese aparţinând nivelului aplicaţie.

Precizari

- Hardware-ul şi/sau software-ul care se ocupă de toate acestea în cadrul nivelului transport poartă numele de **entitate de transport.**
- **Entitatea de transport** poate aparţine nucleului sistemului de operare, unui proces distinct, unei biblioteci legate de aplicaţiile de reţea sau poate fi găsită în cadrul plăcii de reţea.

Descriere

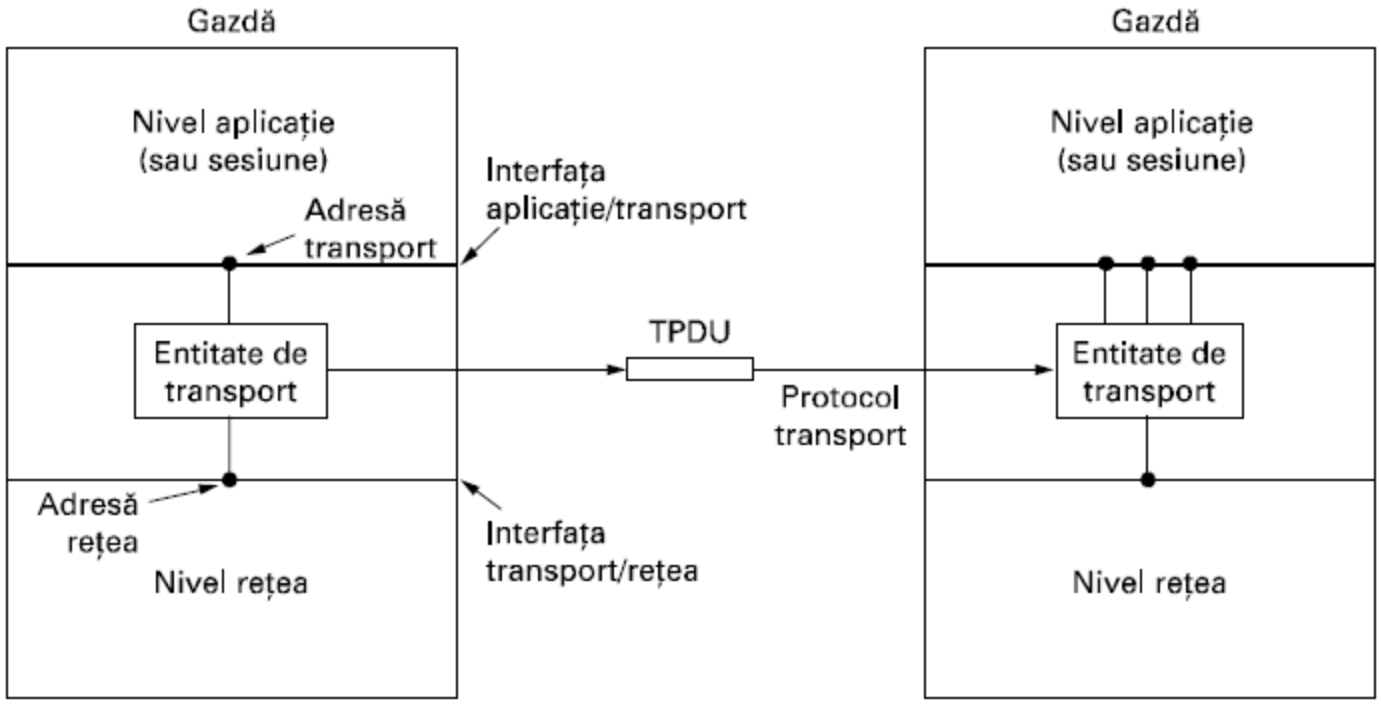


Fig. 6-1. Nivelurile rețea, transport și aplicație.

Servicii

- Serviciul orientat pe conexiune de la nivelul transport are multe asemănări cu cel de la nivel reţea.
- În ambele cazuri, conexiunile au trei faze: stabilirea conexiunii, transferul de date şi eliberarea conexiunii.
- Adresarea şi controlul fluxului sunt şi ele similare pentru ambele niveluri.
- Serviciul fără conexiune al nivelului transport este foarte asemănător cu cel al nivelului reţea

Descriere

- Nivelul transport face posibil ca serviciile de transport să fie mai sigure decât cele echivalente de la nivelul rețea.
- Pachetele pierdute sau incorecte pot fi detectate și corectate de către nivelul transport.
- Mai mult, primitivele serviciului de transport pot fi implementate ca apeluri către procedurile de bibliotecă, astfel încât să fie independente de primitivele de la nivelul reţea.

Primitive

- Pentru a permite utilizatorului să acceseze serviciile de transport, nivelul transport trebuie să ofere unele operaţii programelor aplicaţie, adică o interfaţă a serviciului transport.
- Diferențe intre transport si retea:

Serviciul rețea a fost conceput pentru a modela serviciile oferite de rețelele reale. Acestea pot pierde pachete, deci serviciile la nivel rețea sunt în general nesigure

Serviciile de transport (orientate pe conexiune) sunt sigure.

Retea vs Transport

- Serviciul reţea este folosit doar de entităţile de transport.
- Puţini utilizatori scriu ei înşişi entităţile de transport şi, astfel, puţini utilizatori sau programe ajung să vadă vreodată serviciile reţea aşa cum sunt ele.
- În schimb, multe programe (și programatori) folosesc primitivele de transport. De aceea, serviciul transport trebuie să fie uşor de utilizat.

Primitive

Primitiva	Unitatea de date trimisă	Explicații
LISTEN	(nimic)	Se blochează până când un proces încearcă să se conecteze
CONNECT	CONNECTION REQ.	Încearcă să stabilească conexiunea
SEND	DATE	Transmite informație
RECEIVE	(nimic)	Se blochează până când primește date trimise
DISCONNECT	DISCONNECTION REQ.	Trimisă de partea care vrea să se deconecteze

Fig. 6-2. Primitivele unui serviciu de transport simplu.

Exemplu

- Considerăm o aplicaţie cu un server şi un număr oarecare de clienţi la distanţă.
- La început, serverul apelează primitiva LISTEN, în general prin apelul unei funcţii de bibliotecă care face un apel sistem pentru a bloca serverul până la apariţia unei cereri client.
- Atunci când un client vrea să comunice cu serverul, el va executa un apel CONNECT.
- Entitatea de transport tratează acest apel blocând apelantul şi trimiţând un pachet la server.

Exemplu (2)

- Acest pachet încapsulează un mesaj către entitatea de transport de pe server.
- **TPDU (Transport Protocol Data Unit - unitate de date a protocolului de** transport) pentru toate mesajele schimbate între două entităţi de transport corespondente.
- TPDU-urile (schimbate la nivelul transport) sunt conţinute în pachete (utilizate de nivelul reţea).
- La rândul lor, pachetele sunt conţinute în cadre (utilizate la nivelul legătură de date).

Exemplu (3)

- Atunci când este primit un cadru, nivelul legătură de date prelucrează antetul cadrului şi dă conţinutul util nivelului reţea.
- Entitatea reţea prelucrează antetul pachetului şi pasează conţinutul util entităţii de transport
- Apelul CONNECT al clientului generează un TPDU de tip CONNECTION REQUEST care îi este trimis serverului.
- Atunci când acesta ajunge, entitatea de transport verifică dacă serverul este blocat într-un apel LISTEN (deci dacă aşteaptă o cerere de conexiune)

Exemplu

- În acest caz, deblochează serverul şi trimite înapoi clientului un TPDU CONNECTION ACCEPTED.
- Atunci când acest TPDU ajunge la destinaţie, clientul este deblocat şi conexiunea este stabilită.
- Acum pot fi schimbate date folosindu-se primitivele SEND şi RECEIVE.
- Cea mai simplă posibilitate este ca una din părţi să facă un apel RECEIVE (blocant) aşteptând ca cealaltă parte să execute un SEND.

Final

- Atunci când sosește un TPDU, receptorul este deblocat.
- El poate prelucra TPDU-ul și trimite o replică.
- Atâta vreme cât amândouă părţile ştiu cine este la rând să trimită mesaje şi cine este la rând să recepționeze, totul merge bine.
- Fiecare pachet de date trimis va fi confirmat.
- Pachetele care conţin TPDU-uri de control sunt de asemenea confirmate, implicit sau explicit.

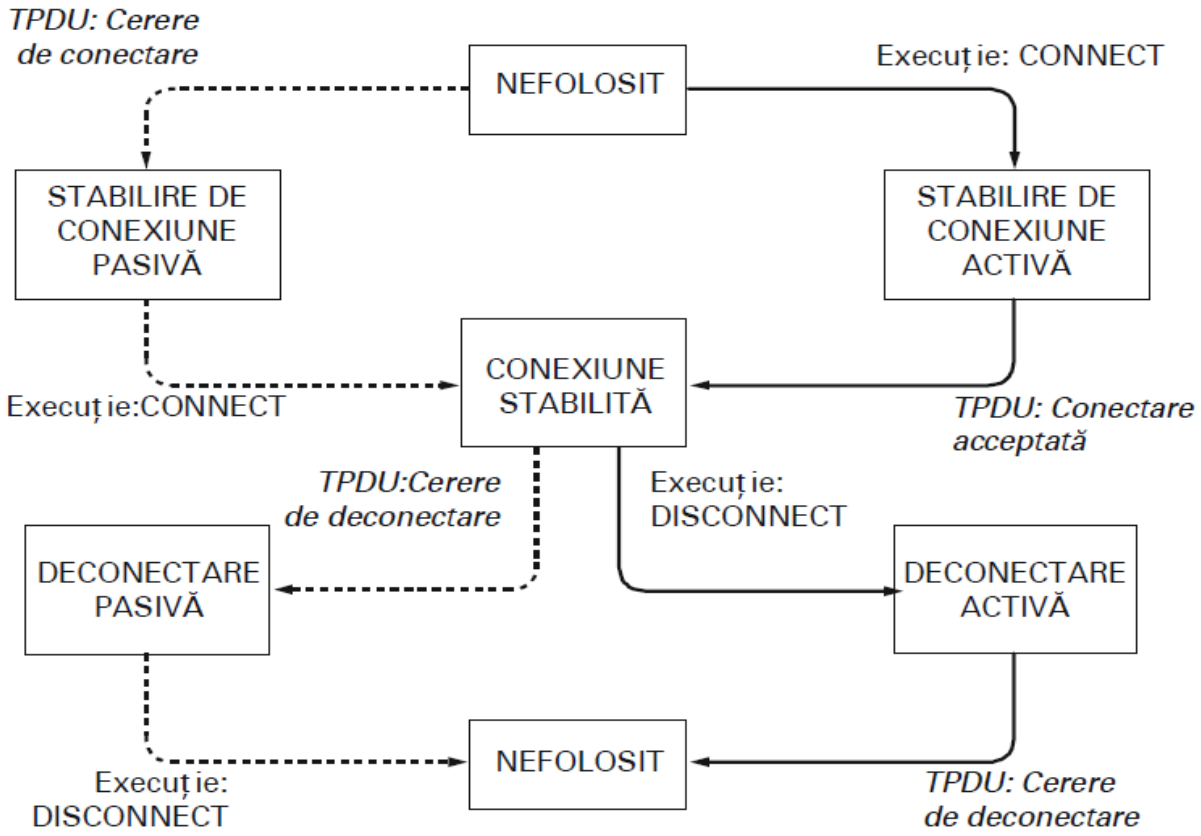


Fig. 6-4. Diagrama de stări pentru o schemă simplă de control al conexiunii. Tranzițiile etichetate cu italice sunt cauzate de sosirea unor pachete. Liniile continue indică secvența de stări a clientului. Liniile punctate indică secvența de stări a serverului.

Socluri Berkeley

Primitiva	Funcția
SOCKET	Creează un nou punct de capăt al comunicației
BIND	Atașează o adresă locală la un soclu
LISTEN	Anunță capacitatea de a accepta conexiuni; determină mărimea cozii
ACCEPT	Blochează apelantul până la sosirea unei cereri de conexiune
CONNECT	Tentativă (activă) de a stabili o conexiune
SEND	Trimite date prin conexiune
RECEIVE	Recepționează date prin conexiune
CLOSE	Eliberează conexiunea

Fig. 6-5. Primitivele pentru socluri TCP

Executie

- Primele patru primitive sunt executate, în această ordine, de către server.
- Primitiva SOCKET creează un nou capăt al conexiunii şi alocă spaţiu pentru el în tabelele entităţii de transport.
- În parametrii de apel se specifică formatul de adresă utilizat, tipul de serviciu dorit (de exemplu, flux sigur de octeți) și protocolul.
- Un apel SOCKET reușit întoarce un descriptor de fișier (la fel ca un apel OPEN) care va fi utilizat în apelurile următoare.

Socluri

- Soclurile nou create nu au încă nici o adresă.
- Ataşarea unei adrese se face utilizând primitiva BIND.
- Odată ce un server a ataşat o adresă unui soclu, clienţii se pot conecta la el.
- Motivul pentru care apelul SOCKET nu creează adresa direct este că unor procese le pasă de adresa lor (de exemplu, unele folosesc aceeaşi adresă de ani de zile şi oricine cunoaşte această adresă), în timp ce altele nu.

LISTEN

- Apelul LISTENalocă spaţiu pentru a reţine apelurile primite în cazul când mai mulţi clienţi încearcă să se conecteze în acelaşi timp.
- Spre deosebire de modelul din primul exemplu, aici LISTEN nu mai este un apel blocant
- Pentru a se bloca şi a aştepta un apel, serverul execută o primitivă ACCEPT.
- Atunci când soseşte un TPDU care cere o conexiune, entitatea de transport creează un nou soclu cu aceleaşi proprietăţi ca cel iniţial şi întoarce un descriptor de fişier pentru acesta.

ACCEPT

- Serverul poate atunci să creeze un nou proces sau fir de execuţie care va gestiona conexiunea de pe noul soclu şi să aştepte în continuare cereri de conexiune pe soclul iniţial.
- ACCEPT returnează un descriptor normal de fişier, care poate fi folosit pentru citirea și scrierea în mod standard, la fel ca pentru fișiere.

CLIENT

- In cazul client, soclul trebuie creat folosind o primitivă SOCKET, dar primitiva BIND nu mai este necesară, deoarece adresa folosită nu mai este importantă pentru server.
- Primitiva CONNECT blochează apelantul şi demarează procesul de conectare.
- Când acesta s-a terminat (adică atunci când TPDU-ul corespunzător a fost primit de la server), procesul client este deblocat şi conexiunea este stabilită.

CLIENT (2)

- Atât clientul cât şi serverul pot utiliza acum primitivele SEND şi RECEIVE pentru a transmite sau recepţiona date folosind o conexiune duplex integral.
- Se pot folosi şi apelurile de sistem READ şi WRITE standard din UNIX, dacă nu sunt necesare opţiunile speciale oferite de SEND şi RECV
- Eliberarea conexiunii este simetrică. Atunci când ambele părţi au executat primitiva CLOSE, conexiunea este eliberată..

PROTOCOALELE DE TRANSPORT

- Serviciul transport este implementat prin intermediul unui **protocol de transport folosit de cele** două entităţi de transport.
- Caracteristici asemănătoare pentru protocoalele de transport și pentru cele de legătură de date:
  - Amândouă trebuie să se ocupe de controlul erorilor, de secvențiere și de controlul fluxului

Diferente

- Diferenţe le sunt datorate deosebirilor majore dintre mediile în care operează protocoalele.
- La nivelul legăturii de date, cele două rutere comunică direct printr-un canal fizic, în timp ce la nivelul transport acest canal fizic este înlocuit de întreaga subreţea.
- În cazul legăturii de date, pentru un ruter nu trebuie specificat cu care alt ruter vrea să comunice, deoarece fiecare linie specifică în mod unic o destinaţie.
- În schimb, în cazul nivelului transport este necesară adresarea explicită.

Diferente

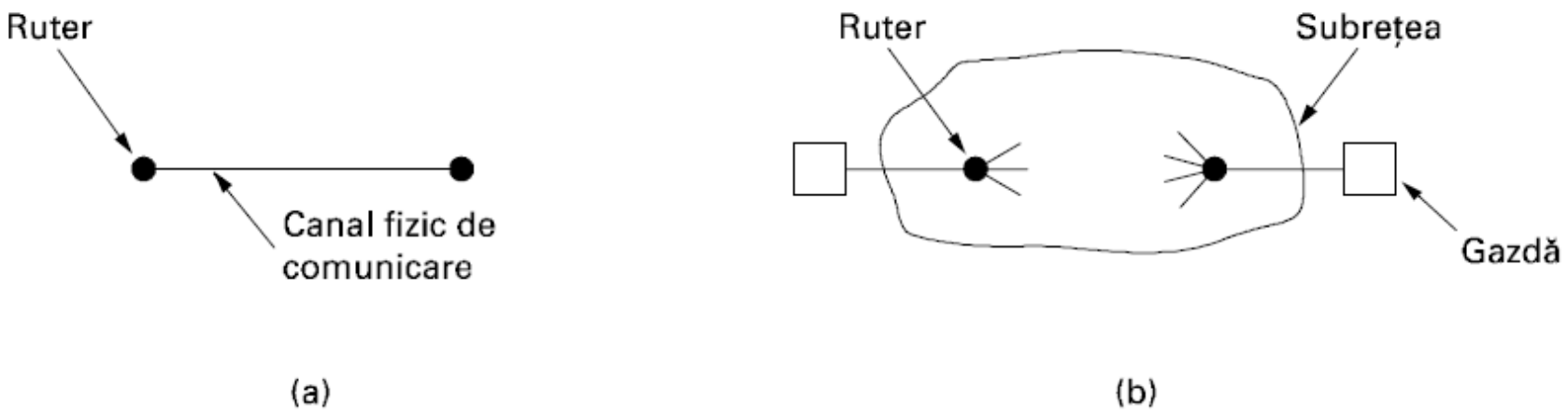


Fig. 6-7. (a) Mediul pentru nivelul legătură de date. (b) Mediul pentru nivelul transport.

Diferente

- O altă diferenţă între nivelurile legătură de date şi transport, care generează multe probleme, este existenţa potenţială a unei capacităţi de memorare a subreţelei.
- Atunci când un ruter trimite un cadru (nivel legătură de date), acesta poate să ajungă sau poate să se piardă.
- Pentru nivelul transport, stabilirea iniţială a conexiunii este mult mai complicată.
- O ultimă diferenţă între nivelurile legătură de date şi transport este una de dimensionare. Folosirea tampoanelor şi controlul fluxului sunt necesare la amândouă nivelurile, dar prezenţa unui număr mare de conexiuni în cazul nivelului transport necesită o abordare diferită.

Adresarea

- Atunci când un proces aplicaţie (de exemplu, un proces utilizator) doreşte să stabilească o conexiune cu un proces aflat la distanţă, el trebuie să specifice cu care proces doreşte să se conecteze.
- Metoda folosită în mod normal este de a defini adrese de transport la care procesele pot să aștepte cereri de conexiune

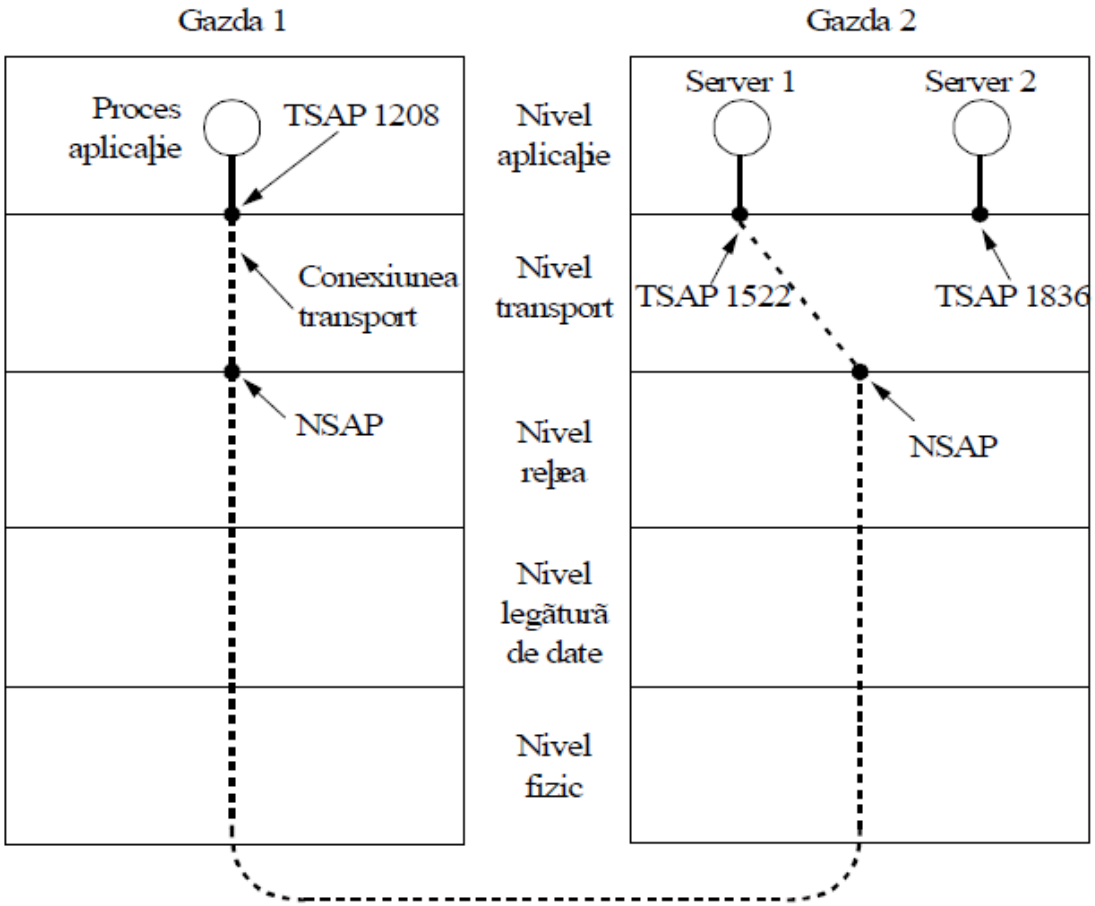


Fig. 6-8. TSAP, NSAP și conexiunile la nivel transport.

Stabilirea conexiunii

- Problema complicată.
- Suficient ca o entitate de transport să trimită numai un TPDU CONNECTION REQUEST şi să aştepte replica CONNECTION ACCEPTED?
- NU!
- Reţeaua poate pierde, memora sau duplica pachete

Retea congestionata

- O subreţea poate fi atât de congestionată încât confirmările ajung greu înapoi, şi, din această cauză, fiecare pachet ajunge să fie retransmis de câteva ori.
- Putem presupune că subreţeaua foloseşte datagrame şi fiecare pachet urmează un traseu diferit.
- Unele pachete pot să întâlnească o congestie locală de trafic şi să întârzie foarte mult, ca şi cum ar fi fost memorate de subreţea un timp şi eliberate mai târziu.

Scenariu negru

- Un utilizator stabileşte o conexiune cu o bancă şi trimite un mesaj cerând transferul unei sume de bani în contul unei alte persoane în care nu poate avea încredere în totalitate, şi apoi eliberează conexiunea.
- Din nefericire, fiecare pachet din acest scenariu este duplicat şi memorat în subreţea.

COSMAR

- După ce conexiunea a fost eliberată, pachetele memorate ies din subreţea şi ajung la destinatar, cerând băncii să stabilească o nouă conexiune, să facă transferul (încă o dată) şi să elibereze conexiunea.

- Banca nu poate să ştie că acestea sunt duplicate, ea trebuie să presupună că este o tranzacţie independentă şi va transfera banii încă o dată.

Duplicate

- Punctul crucial al problemei este existența duplicatelor întârziate.
- Nu exista solutie, doar aproximatii.
- Solutii:
- Adrese de transport valabile doar pentru o singură utilizare. În această abordare, ori de câte ori este necesară o adresă la nivel transport, va fi generată una nouă.
- După ce conexiunea este eliberată, adresa nu mai este folosită.

DUPLICATE DUPLICATE

- Atribuirea pentru fiecare conexiune a unui identificator (adică, un număr de secvenţă incrementat pentru fiecare conexiune stabilită), ales de cel care iniţiază conexiunea, şi pus în fiecare TPDU, inclusiv în cel care iniţiază conexiunea.
- După ce o conexiune este eliberată, fiecare entitate de transport va completa o tabelă cu conexiunile care nu mai sunt valide, reprezentate ca perechi (entitate de transport, identificator conexiune).
- Ori de câte ori apare o cerere de conexiune se va verifica în tabelă că ea nu aparţine unei conexiuni care a fost eliberată anterior.

DEFECT

- Necesită ca fiecare entitate de transport să menţină informaţia despre conexiunile precedente un timp nedefinit.
- Dacă o maşină cade şi îşi pierde datele din memorie, ea nu va mai şti care identificatori de conexiune au fost deja utilizaţi

Solutie alternativa

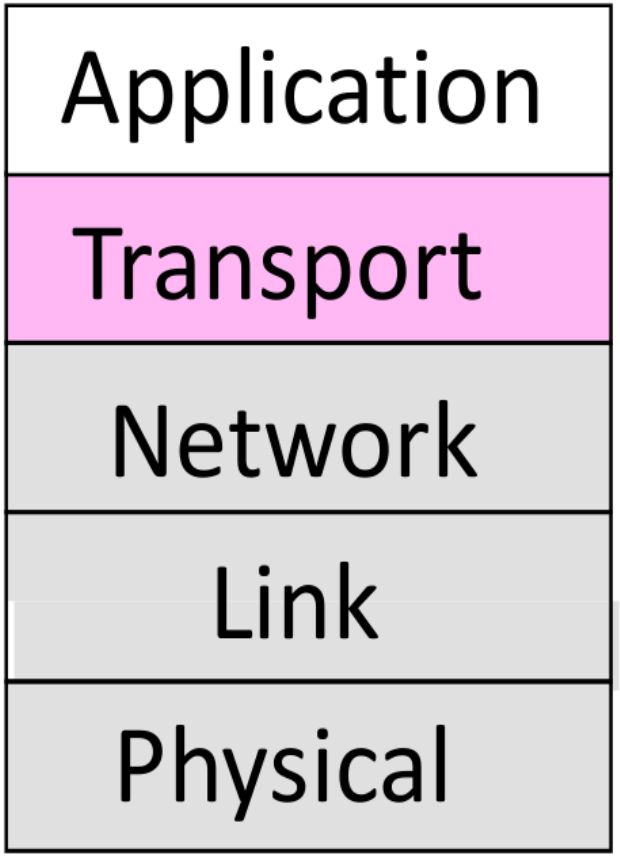
- Inventam un mecanism care să elimine pachetele îmbătrânite.
- Dacă suntem siguri că nici un pachet nu poate să supravieţuiască mai mult de un anume interval de timp cunoscut, durata de viaţă a pachetelor poate fi limitată la un maxim cunoscut, folosind:
  - Restricţii în proiectarea subreţelei
  - Adăugarea unui contor al nodurilor parcurse în fiecare pachet
  - Adăugarea unei amprente de timp la fiecare pachet

MULȚUMESC!

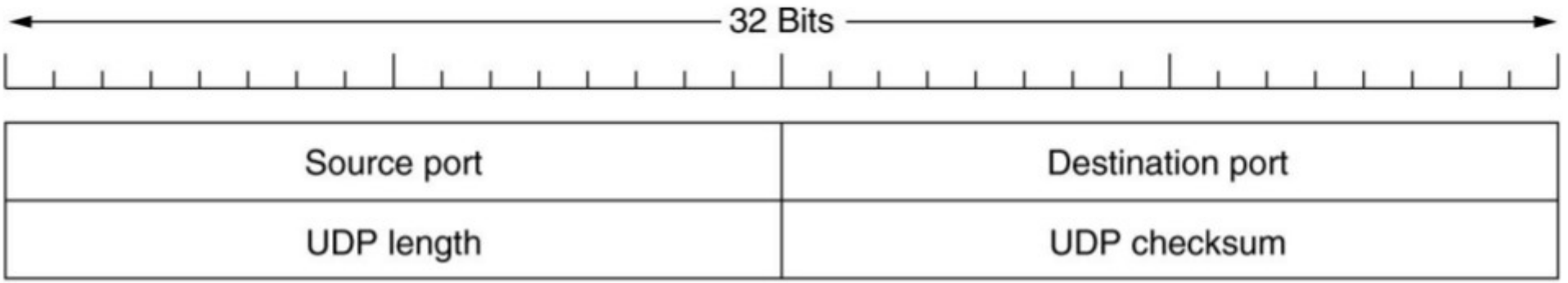
Nivelul Transport (2)

Liviu P. Dinu
ldinu@fmi.unibuc.ro
Center for Computational Linguistics
nlp.unibuc.ro

Transport layer – UDP / TCP



User Datagram Protocol (UDP)



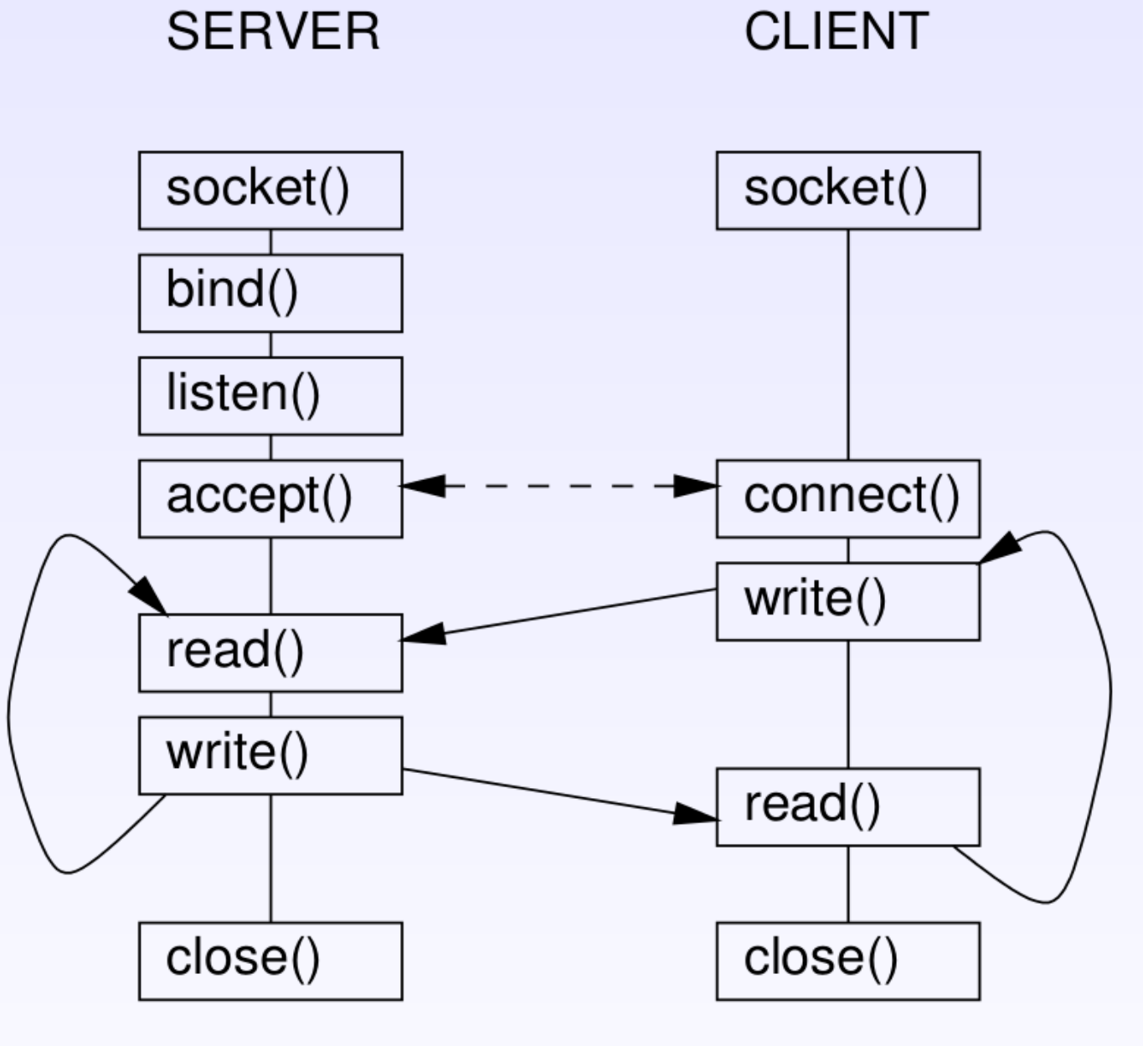
- UDP is simple:
  - no flow control
  - no error control
  - no retransmissions
- UDP packets cannot be larger than 65 K
  - that is the maximum IP packet size
- checksum (16 bits) for reliability

Socket API

SOCKET, BIND, LISTEN, ACCEPT, CONNECT, SEND, RECEIVE, CLOSE

- The client and server each bind a transport-level address and a name to the locally created socket.
- The server must listen to its socket, thereby telling the kernel that it will subsequently wait for connections from clients.
- After that, the server can accept or select connections from clients.
- The client connects to the socket.
- it needs to provide the transport-level address by which it can locate the server.
- Now the client and server communicate through send/receive operations on their respective sockets.
- also standard read / write operations can be used

Socket API (2)


SERVER
CLIENT
socket()
bind()
listen()
accept()
read()
write()
close()
socket()
connect()
write()
read()
close()

UDP client

```
1 DatagramSocket  socket = new DatagramSocket();
2 byte[] data = "Hello Server".getBytes();
3 DatagramPacket packet = new DatagramPacket(data, data.length,
4                                            HOST, PORT);
5 socket.send(packet);
6 socket.setSoTimeout(1000);
7 packet.setData(new byte[PACKETSIZE]);
8 socket.receive(packet);
9 System.out.println(new String(packet.getData()));
```

UDP server

```
1  DatagramSocket socket = new DatagramSocket(port);
2  while (true){
3          DatagramPacket packet =
4                  new DatagramPacket(new byte[PACKETSIZE], PACKETSIZE);
5          socket.receive(packet);
6          System.out.println(packet.getAddress() +
7                  " " + packet.getPort() + ": " + new String(packet.getData()));
8          socket.send(packet);
9  }
10
```

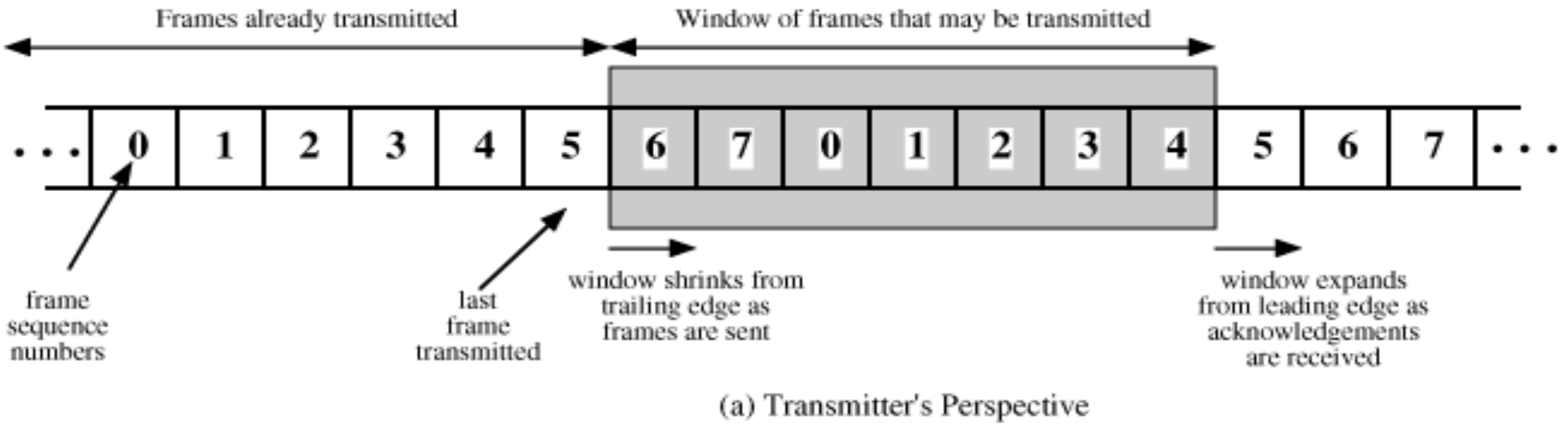
UDP – why? when?

1. finer application-level control over what data is sent and when (e.g. real-time, skype, VoIP, DNS)
2. no connection establishment and no connection state
4. small packet header overhead
5. good in client-server situations (short interactions)
6. simple code

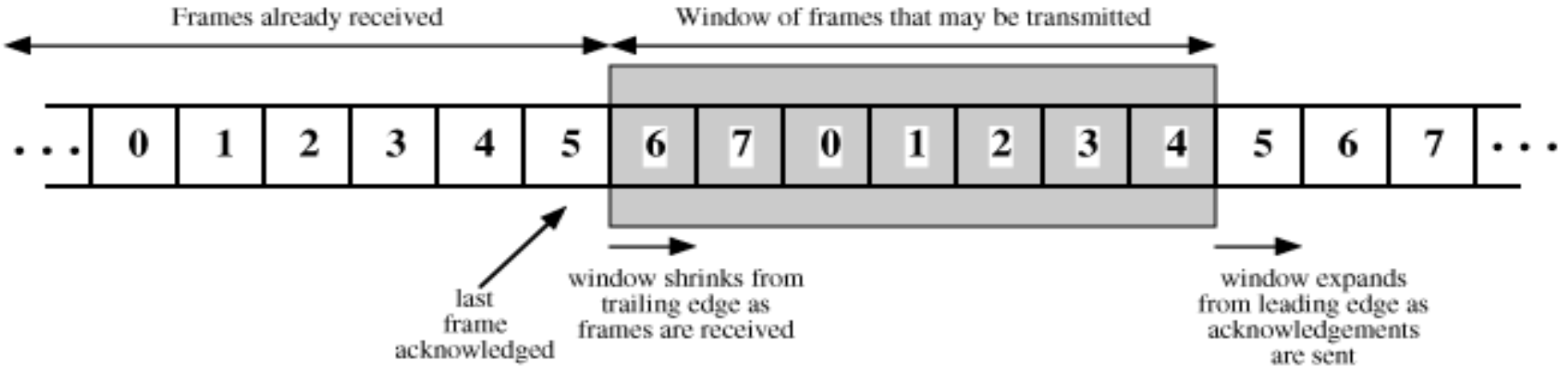
TCP

- sliding window for reliability (timeout)
- flow control for slow receivers
- based on connections
- reliable bytestream service (order)

Sliding window – recap



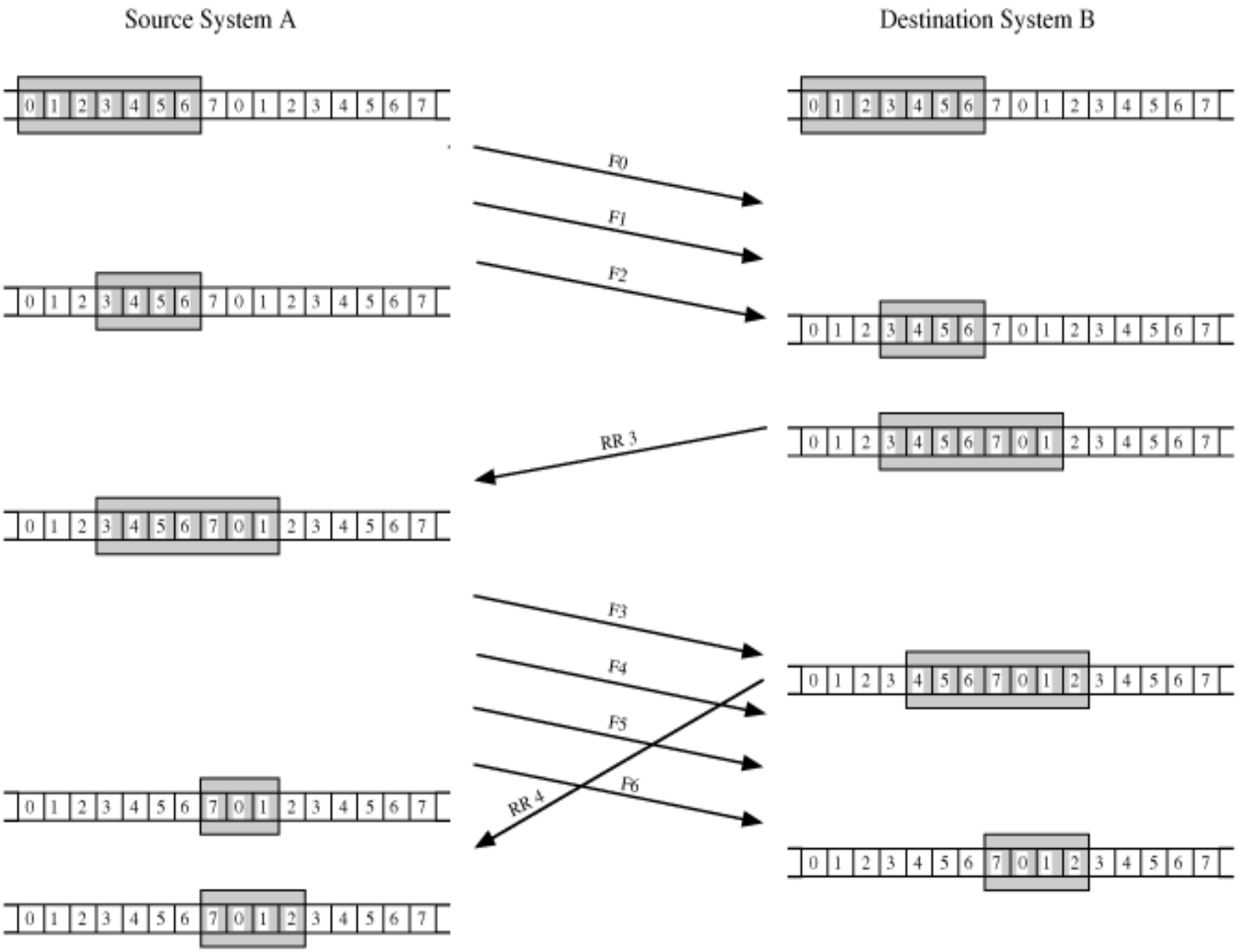
(a) Transmitter's Perspective



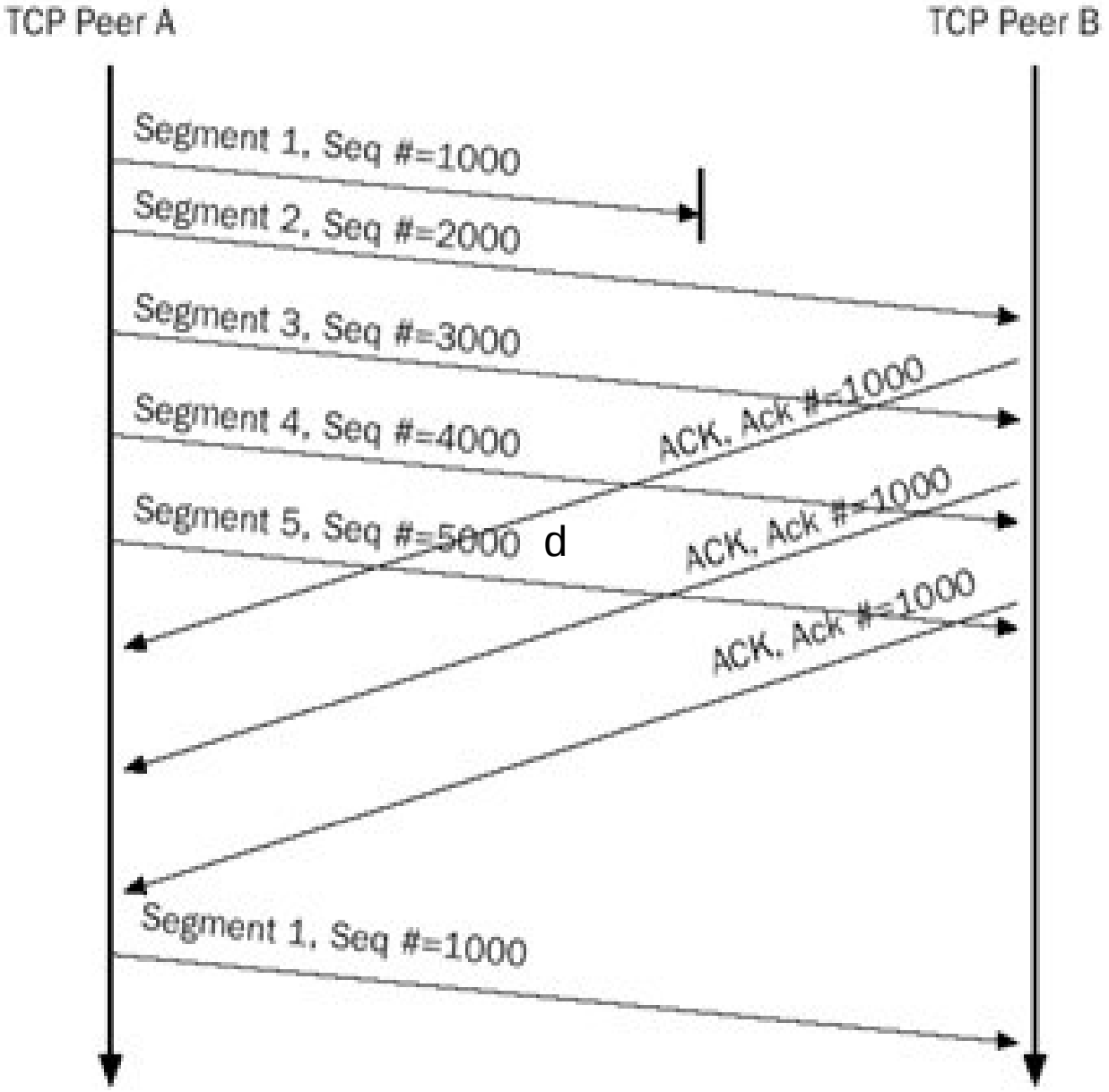
(b) Receiver's Perspective

Timeout for each frame

Sliding window – recap



TCP 3-ACKs


TCP Peer A
TCP Peer B
Segment 1, Seq #=1000
Segment 2, Seq #=2000
Segment 3, Seq #=3000
Segment 4, Seq #=4000
Segment 5, Seq #=5000
d
ACK, Ack #=1000
ACK, Ack #=1000
ACK, Ack #=1000
Segment 1, Seq #=1000

TCP - timeout

- Timeout should be “just right”
  - Too long wastes network capacity
  - Too short leads to spurious resends
  - But what is “just right”?
- Easy to set on a LAN (Link)
  - Short, fixed, predictable RTT (timp tur-retur)
- Hard on the Internet (Transport)
  - Wide range, variable RTT

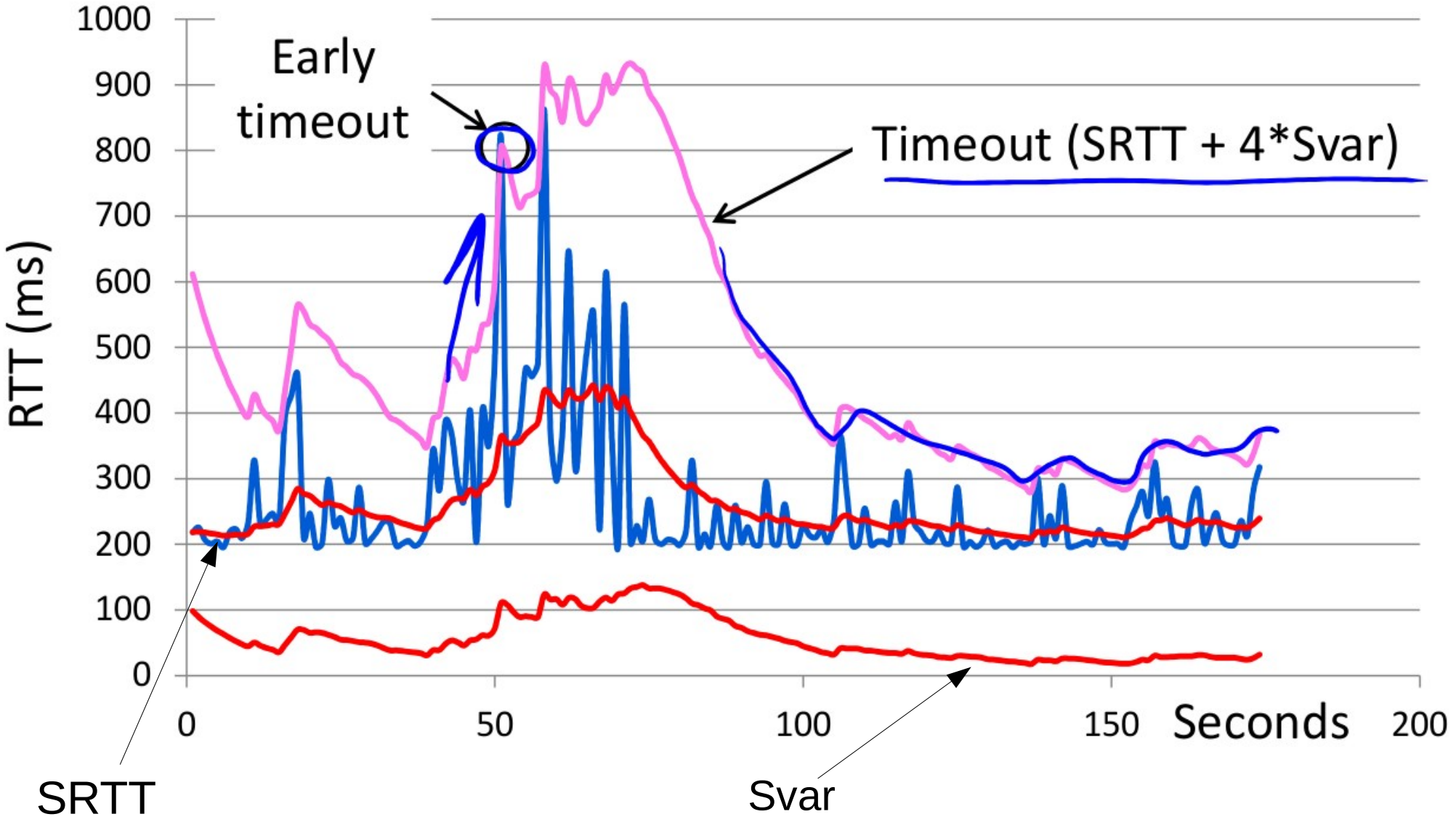
TCP - adaptive timeout (2)

$$SRTT_{N+1} = 0.9*SRTT_{N} + 0.1*RTT_{N+1}$$

$$Svar_{N+1} = 0.9*Svar_{N} + 0.1* \mid RTT_{N+1} - SRTT_{N+1} \mid$$

$$\text{TCP Timeout}_{N} = SRTT_{N} + 4*Svar_{N}$$

Real data

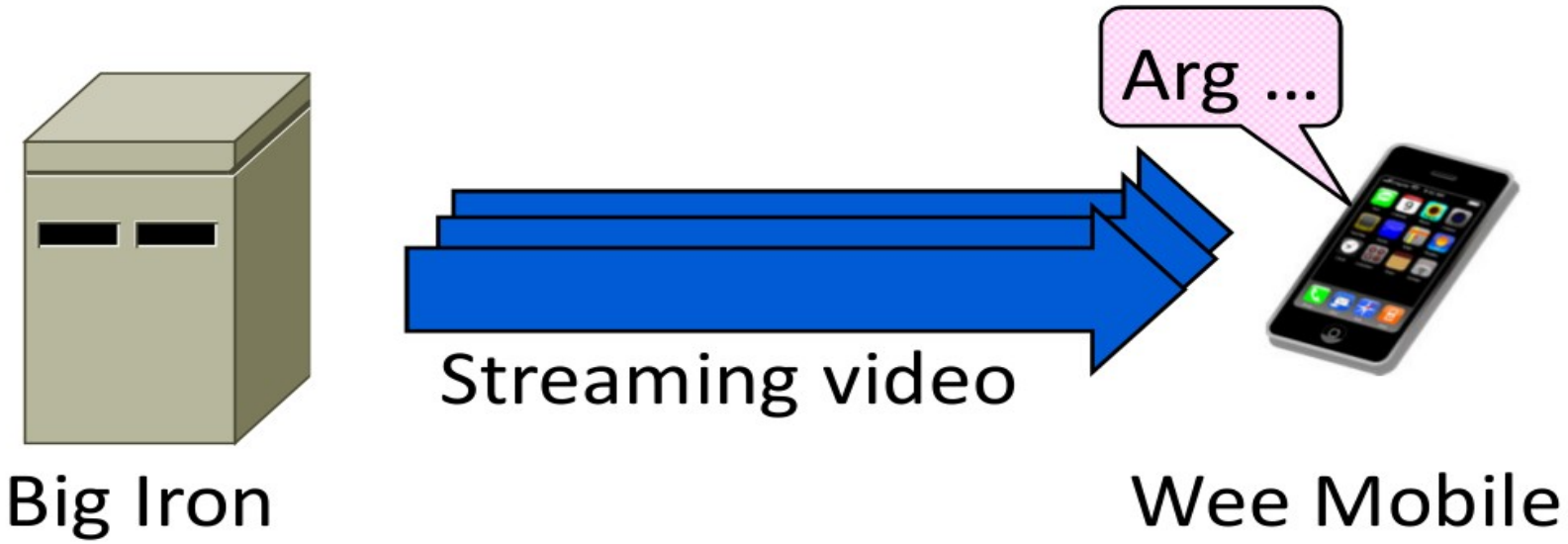

1000
900
800
700
600
500
400
300
200
100
0
RTT (ms)
Early timeout
Timeout (SRTT + 4*Svar)
0
50
100
150
200
Seconds
SRTT
Svar

Questions?

Flow control

Problem: Sliding window uses pipelining to keep the network busy

What if the receiver is overloaded?

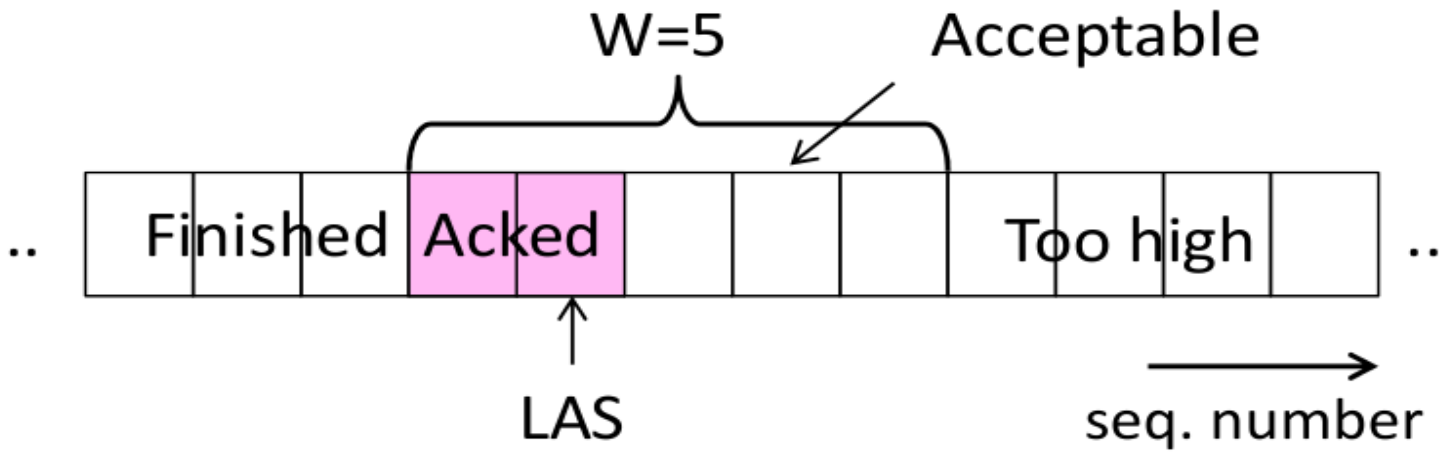


Flow control (2)

- Consider receiver with sliding window of W buffers

  LAS = LAST ACK SENT, Application layer pulls in-order data from buffer with recv() call

- Suppose the next two segments arrive but app does not call recv()
  - LAS rises, but we can’t slide window!

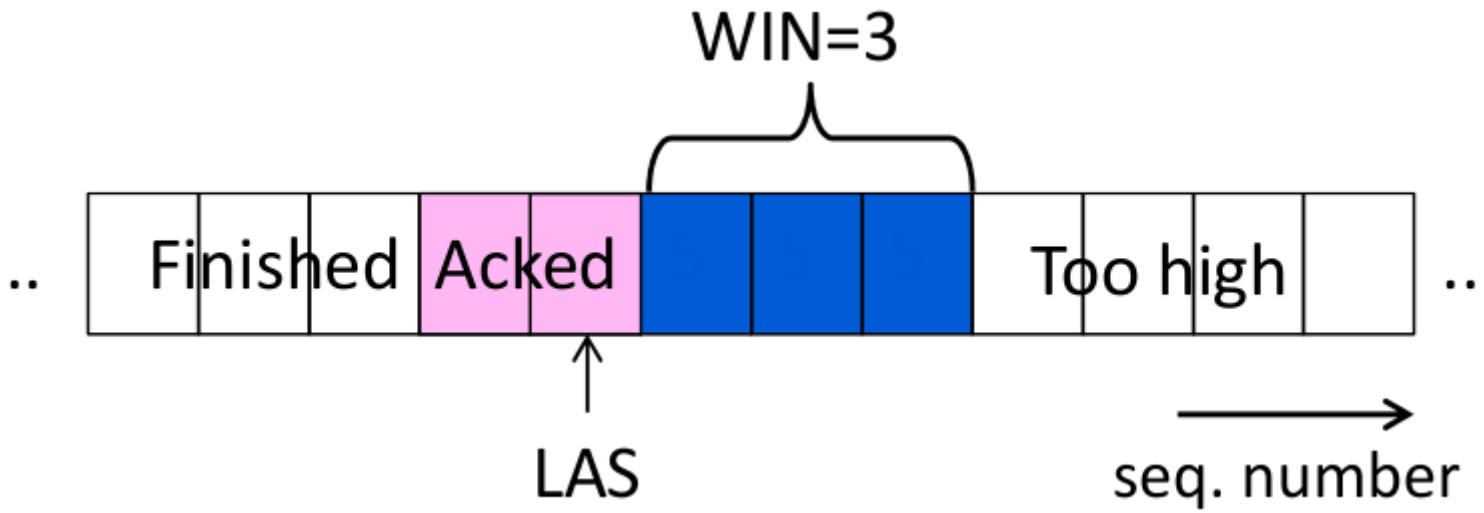


Flow control (3)

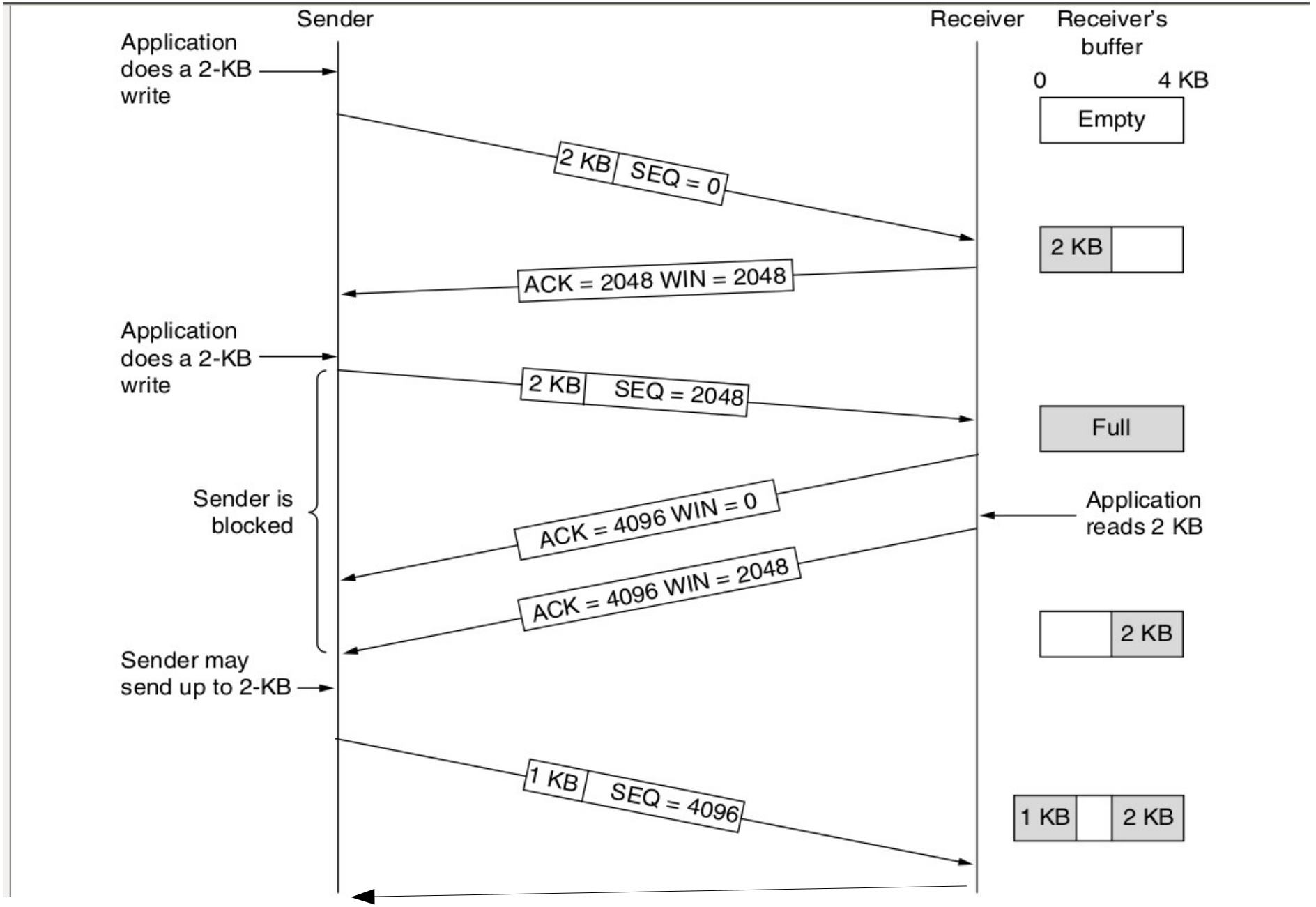
Solution: Avoid loss at receiver by telling sender the available buffer space

WIN= #Acceptable = W – LAS

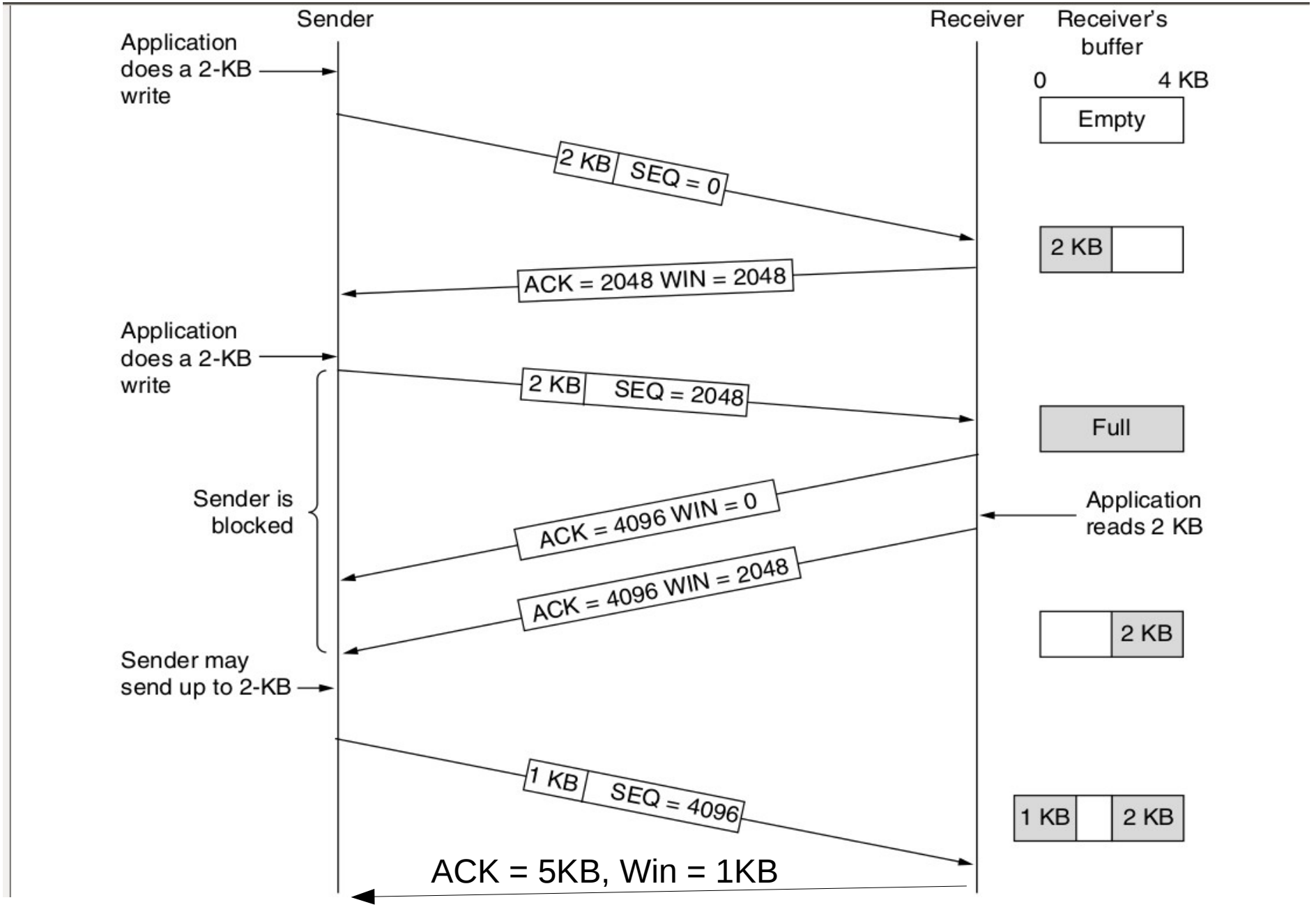
- Sender uses the lower of the sliding window and flow control window (WIN) as the effective window size



Flow control (4)



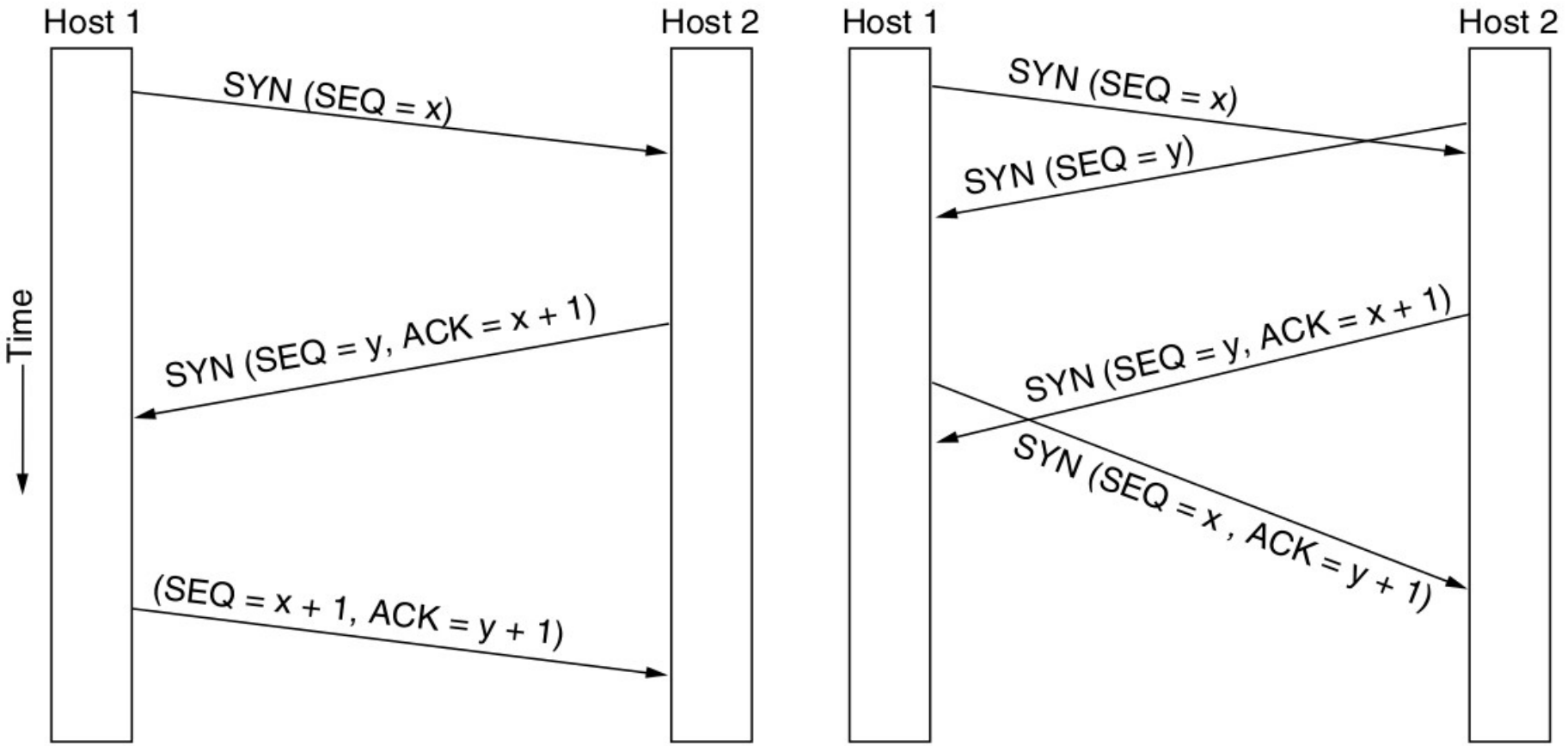
Flow control (4)



Questions?

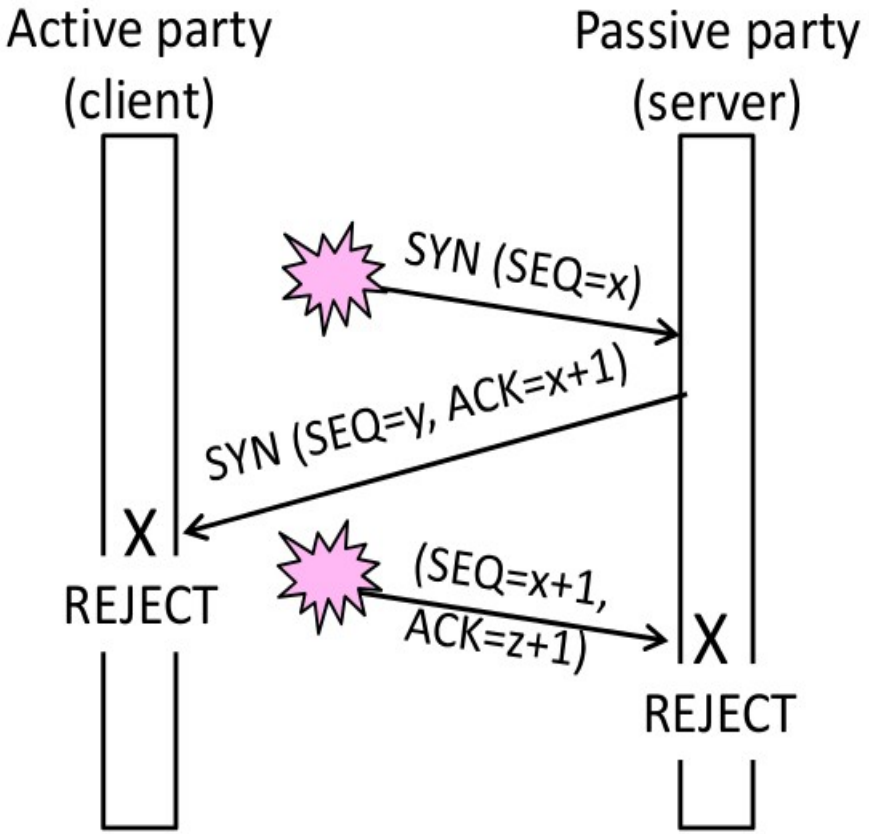
TCP - connection esablishment

- three way handshake
- initial sequence numbers are chosen through a clock based scheme (clock tick is 4 μs)

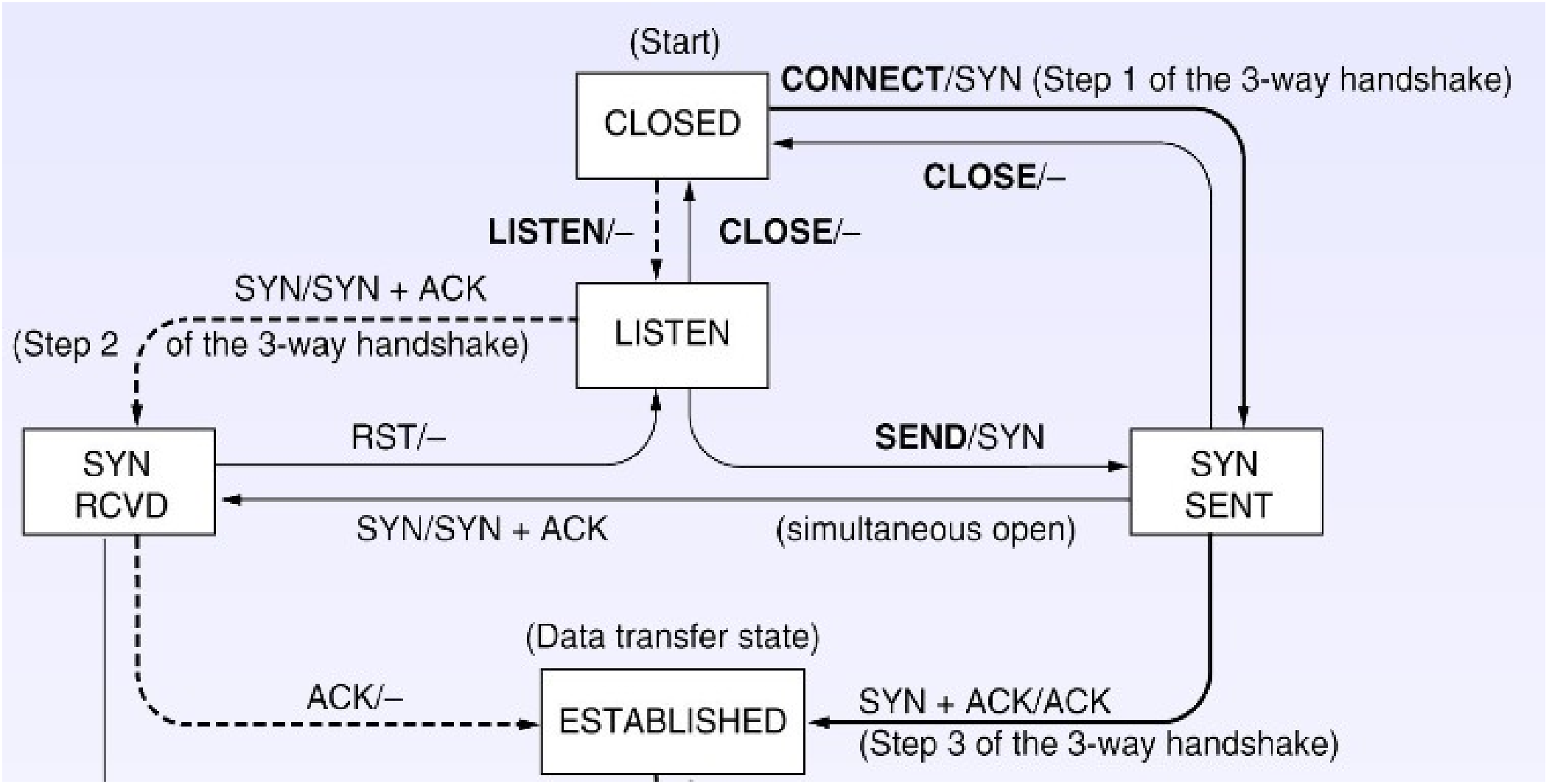


TCP - connection establishment (2)

- Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
- Connection will be cleanly rejected on both sides



Connect – FSM



CLOSE

TCP – connection release

Two steps:

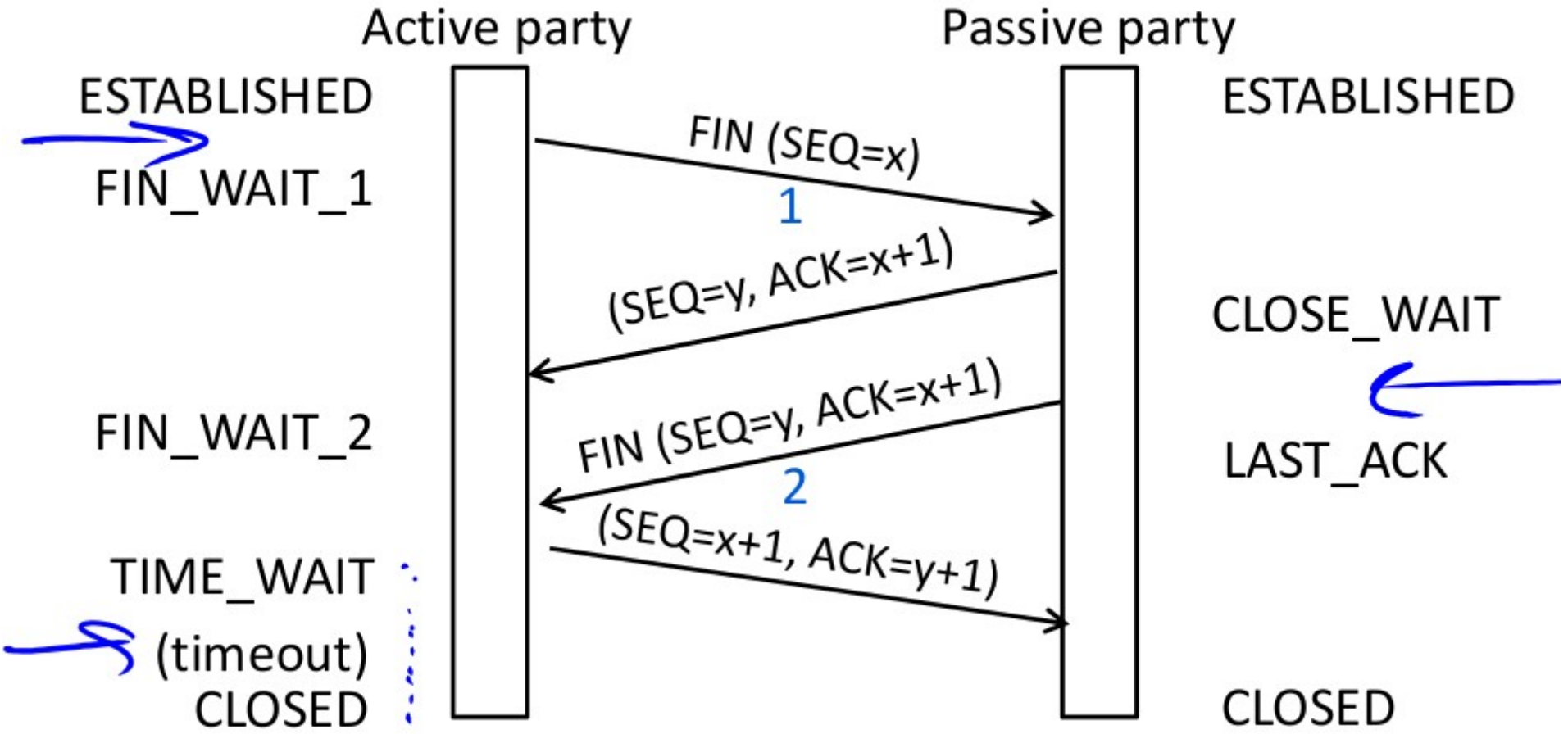
– Active sends FIN(x), passive ACKs

– Passive sends FIN(y), active ACKs

– FINs are retransmitted if lost

• Each FIN/ACK closes one direction of data transfer

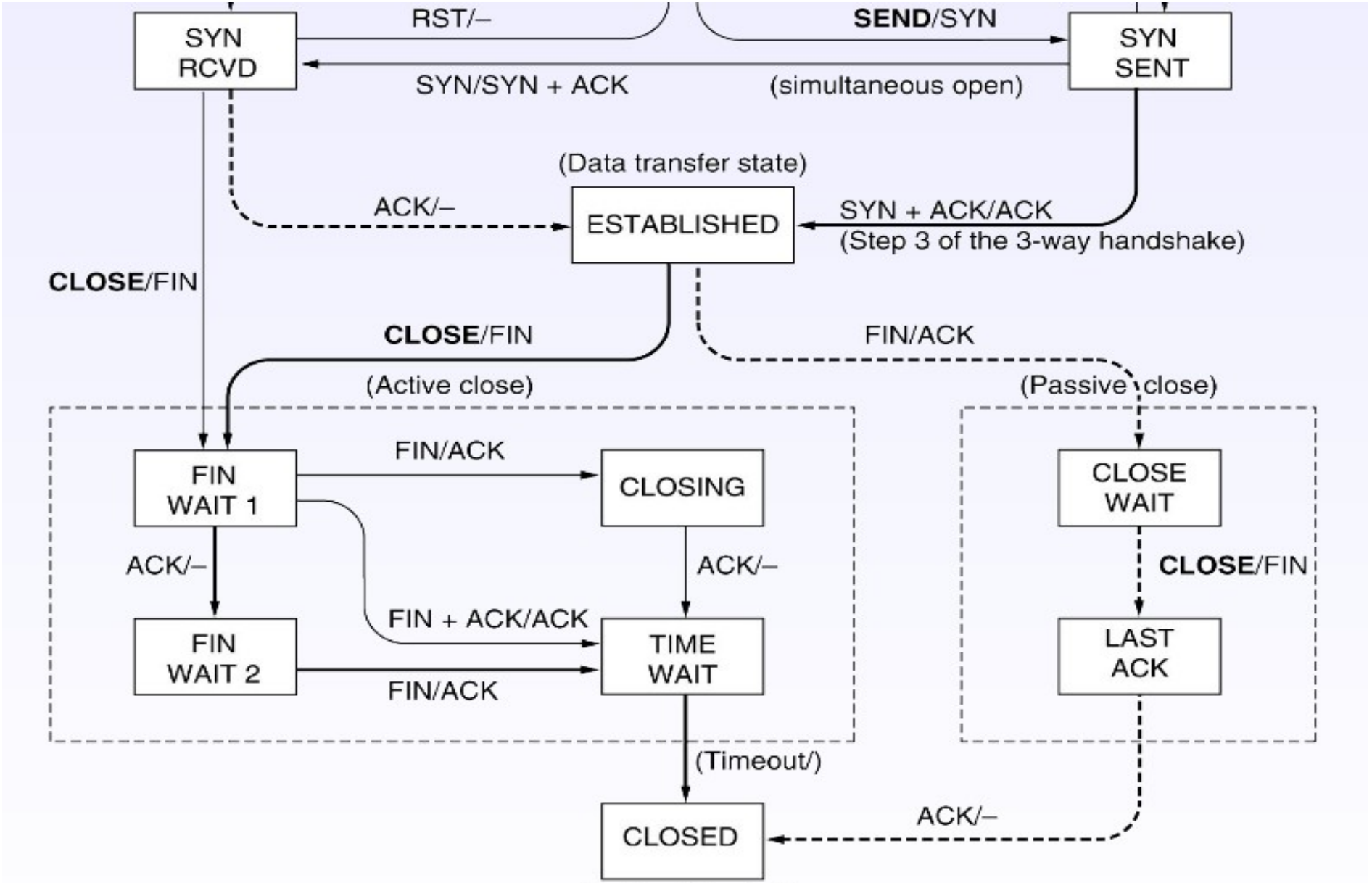
TCP – connection release (2)



TCP – connection release (3)

- We wait a long time (two times the maximum segment lifetime of 60 seconds) after sending all segments and before completing the close
  - ACK might have been lost, in which case FIN will be resent for an orderly close
  - Could otherwise interfere with a subsequent connection

Release - FSM


SYN RCVD
RST/–
SEND/SYN
SYN SENT
SYN/SYN + ACK
(simultaneous open)
(Data transfer state)
ACK/–
ESTABLISHED
SYN + ACK/ACK
(Step 3 of the 3-way handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
(Active close)
(Passive close)
FIN WAIT 1
FIN/ACK
CLOSING
CLOSE WAIT
ACK/–
ACK/–
CLOSE/FIN
FIN + ACK/ACK
FIN WAIT 2
TIME WAIT
LAST ACK
FIN/ACK
(Timeout/)
CLOSED
ACK/–

TCP server

```
1  String clientSentence;
2  String modifiedSentence;
3  ServerSocket welcomeSocket = new ServerSocket(PORT);
4  while (true) {
5          Socket connectionSocket = welcomeSocket.accept();
6          BufferedReader inFromClient =
7                  new BufferedReader(new
8                          InputStreamReader(
9                                  connectionSocket.getInputStream()));
10         DataOutputStream outToClient =
11                 new DataOutputStream(
12                         connectionSocket.getOutputStream());
13         clientSentence = inFromClient.readLine();
14         System.out.println("Received: " + clientSentence);
15         modifiedSentence = "hello " + clientSentence + "!" + '\n';
16         outToClient.writeBytes(modifiedSentence);
17 }
```

TCP client

```
1  String sentence = "world";
2  String modifiedSentence;
3  BufferedReader inFromUser =
4            new BufferedReader(
5                     new InputStreamReader(System. in ));
6  Socket clientSocket = new Socket("localhost", PORT);
7  DataOutputStream outToServer =
8            new DataOutputStream(
9                     clientSocket.getOutputStream());
10 BufferedReader inFromServer =
11           new BufferedReader(
12                    new InputStreamReader(
13                              clientSocket.getInputStream()));
14 sentence = inFromUser.readLine();
15 outToServer.writeBytes(sentence + '\n');
16 modifiedSentence = inFromServer.readLine();
17 System.out.println("Dă la servăr: " + modifiedSentence);
18 clientSocket.close();
```

Not covered here

To read about:

- TCP congestion control
- Additive Increase Multiplicative Decrease (for TCP)
  - TCP Ack Clocking
  - TCP slow start
  - TCP fast recovery

Questions?

MULȚUMESC!

Nivelul Transport (3)

Liviu P. Dinu

ldinu@fmi.unibuc.ro

Center for Computational Linguistics

nlp.unibuc.ro

TCP

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

Cerinta

- Maşina care suportă TCP dispune de o entitate de transport TCP, fie ca proces utilizator, fie ca procedură de bibliotecă, fie ca parte a nucleului.
- O entitate TCP acceptă fluxuri de date utilizator de la procesele locale, le împarte în fragmente care nu depăşesc 64K octeţi, şi expediază fiecare fragment ca o datagramă IP separată.

Sosire

- Când datagramele IP conţinând informaţie TCP sosesc la o maşină, ele sunt furnizate entităţii TCP, care reconstruieşte fluxul original de octeţi.
- Nivelul IP nu oferă nici o garanţie că datagramele vor fi livrate corect, astfel că este sarcina TCPului să detecteze eroarea şi să efectueze o retransmisie atunci când situaţia o impune.
- Datagramele care ajung (totuşi) la destinaţie pot sosi într-o ordine eronată; este, de asemenea, sarcina TCP-ului să le reasambleze în mesaje respectând ordinea corectă (de secvenţă).

Modelul serviciului TCP

- Serviciul TCP este obţinut prin crearea atât de către emiţător, cât şi de către receptor, a unor puncte finale (socluri-sockets).
- Fiecare soclu are un număr de soclu (adresă) format din adresa IP a maşinii gazdă şi un număr de 16 biţi, local gazdei respective, numit **port.**
- Pentru a obţine o conexiune TCP, trebuie stabilită explicit o conexiune între un soclu de pe maşina emiţătoare şi un soclu de pe maşina receptoare.

Conexiuni

- Un soclu poate fi folosit la un moment dat pentru mai multe conexiuni.
- Două sau mai multe conexiuni se pot termina la același soclu.
- Conexiunile sunt identificate prin identificatorii soclurilor de la ambele capete, adică *(soclu 1, soclu 2)*.
- *Nu este folosit nici un alt număr sau identificator* de circuit virtual.

<=256

- Numerele de port mai mici decât 256 se numesc **porturi general cunoscute și sunt rezervate serviciilor** standard.
- Orice proces care doreşte să stabilească o conexiune cu o maşină gazdă pentru a transfera un fișier utilizând FTP, se poate conecta la portul 21 al mașinii destinație pentru a contacta demonul său FTP.
- Portul 23 este folosit pentru a stabili o sesiune de lucru la distanţă utilizând TELNET.
- Lista porturilor general cunoscute se găseşte la *www.iana.org*

Porturi

Port	Protocol	Utilitate
21	FTP	Transfer de fișiere
23	Telnet	Login la distanță
25	SMTP	E-mail
69	TFTP	Protocol de transfer de fișiere trivial
79	Finger	Căutare de informații despre un utilizator
80	HTTP	World Wide Web
110	POP-3	Acces prin e-mail la distanță
119	NNTP	Știri USENET

Fig. 6-27. Câteva porturi asignate.

Protocolul TCP

- Caracteristică importantă a TCP: fiecare octet al unei conexiuni TCP are propriul său număr de secvenţă, reprezentat pe 32 biţi
- Entităţile TCP de transmisie şi de recepţie interschimbă informaţie sub formă de segmente.
- Un **segment TCP constă dintr-un antet de exact 20 de octeţi (plus o parte opţională) urmat de zero sau** mai mulţi octeţi de date.

Limite

- Limite care restricţionează dimensiunea unui segment:
  - fiecare segment, inclusiv antetul TCP, trebuie să încapă în cei 65.535 de octeţi de informaţie utilă IP.
  - fiecare reţea are o **unitate maximă de transfer sau MTU (Maximum Transfer Unit), deci fiecare segment** trebuie să încapă în acest MTU.
- În realitate, MTU este în general de 1500 octeţi (dimensiunea informaţiei utile din Ethernet), definind astfel o limită superioară a dimensiunii unui segment.

Baza

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

Baza +Probleme

- Dacă cronometrul emiţătorului depăşeşte o anumită valoare înaintea primirii confirmării, emiţătorul retransmite segmentul neconfirmat.
- Probleme:
- Segmentele pot ajunge într-o ordine arbitrară
- Segmentele pot întârzia pe drum un interval de timp suficient de mare pentru ca emiţătorul să detecteze o depăşire a cronometrului şi să le retransmită.

Probleme

- Retransmisiile pot include porţiuni de mesaj fragmentate altfel decât în transmisia iniţială, ceea ce impune o tratare atentă, astfel încât să se ţină evidenţa octeţilor primiţi corect.

Antetul segmentului TCP

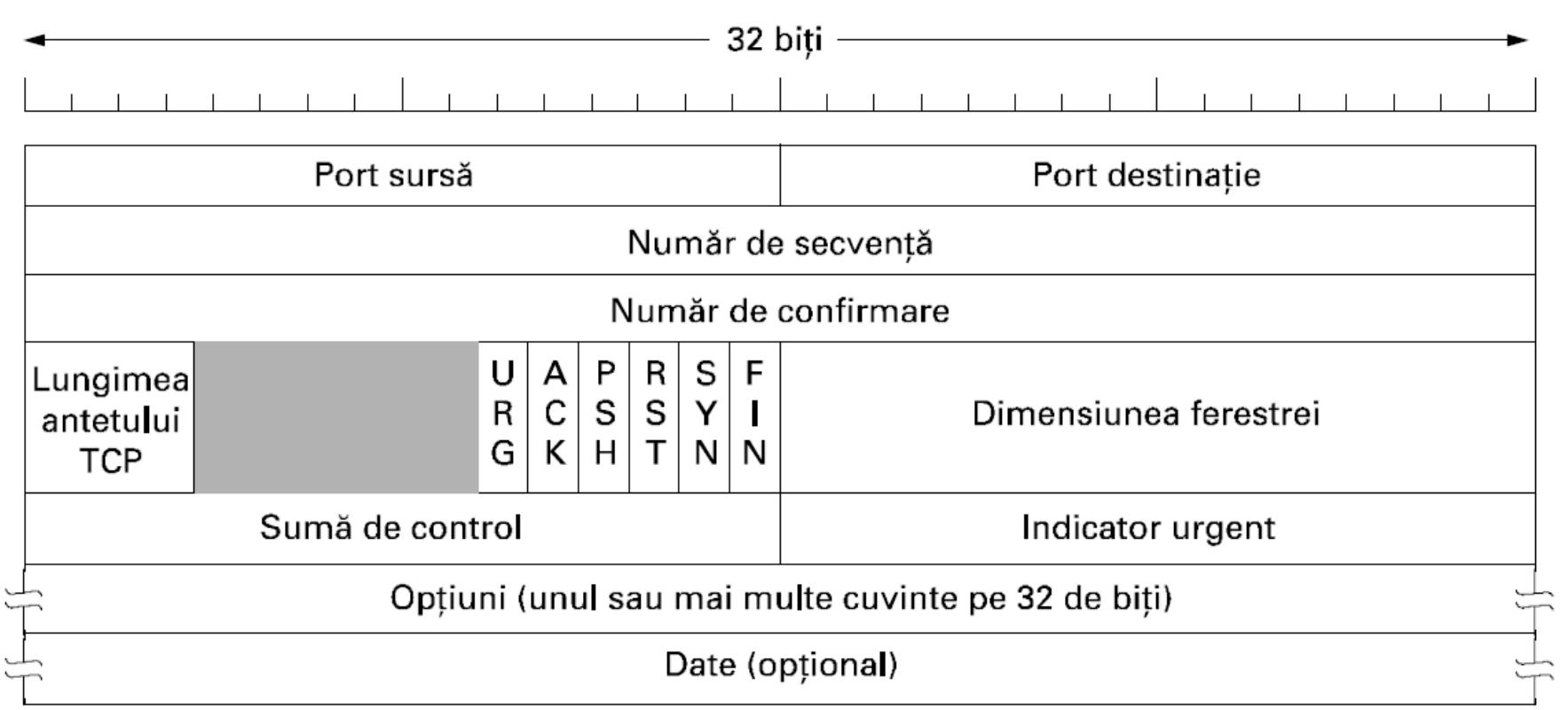


Fig. 6-29. Antetul TCP.

Descriere

- Fiecare segment începe cu un antet format dintr-o structură fixă de 20 de octeţi.
- Antetul fix poate fi urmat de un set de opţiuni asociate antetului.
- În continuarea opţiunilor, dacă ele există, pot urma până la 65.535 - 20 - 20 = 65.495 de octeţi de date, unde primul 20 reprezintă antetul IP, iar al doilea antetul TCP.

Campuri

- Câmpurile *Port sursă și Port destinație* identifică punctele finale ale conexiunii.
- Un port formează împreună cu adresa IP a maşinii sale un unic punct de capăt (eng.: end point) de 48 de biţi.
- Conexiunea este identificată de punctele de capăt ale sursei și destinației.
- *Număr de secvenţă şi Număr de confirmare* au semnificaţia standard. Cel din urmă indică octetul următor aşteptat şi nu ultimul octet recepţionat în mod corect. Ambele câmpuri au lungimea de 32 de biți, deoarece într-un flux TCP fiecare bit de informație este numerotat.

Campuri (2)

- *Lungimea antetului TCP indică numărul de cuvinte de 32 de biţi care sunt conţinute în antetul* TCP. In realitate indica începutul datelor din segment, măsurat în cuvinte de 32 de biţi.
- Câmp de şase biţi care este neutilizat
- Sase indicatori de câte un bit.
- URG este poziţionat pe 1 dacă Indicatorul Urgent este valid. Este folosit pentru a indica deplasamentul în octeţi faţă de numărul curent de secvenţă la care se găseşte informaţia urgentă

Campuri...

- Bitul *ACK este poziţionat pe 1 pentru a indica faptul că Numărul de confirmare este valid.*
  - *În cazul* în care *ACK este poziţionat pe 0, segmentul în discuţie nu conţine o confirmare şi câmpul Număr de confirmare este ignorat.*
- *PSH indică informaţia FORŢATĂ.*
  - *Receptorul este rugat respectuos să livreze aplicaţiei* informaţia respectivă imediat ce este recepţionată şi să nu o memoreze în aşteptarea umplerii tampoanelor de comunicaţie
- *RST este folosit pentru a desfiinţa o conexiune care a devenit inutilizabilă datorită defecţiunii* unei maşini sau oricărui alt motiv

Campuri...

- *SYN este utilizat pentru stabilirea unei conexiuni.*
  - *Cererea de conexiune conţine SYN = 1* şi *ACK = 0 pentru a indica faptul că acel câmp suplimentar de confirmare nu este utilizat.*
  - *Răspunsul* la o astfel de cerere conţine o confirmare, având deci *SYN = 1 şi ACK = 1.*
- *FIN este folosit pentru a încheia o conexiune.*
  - *El indică faptul că emiţătorul nu mai are nici*o informaţie de transmis

Campuri...

- *Fereastră* indică numărul de octeţi care pot fi trimişi, începând de la octetul confirmat.
  - Un câmp *Fereastră* de valoare 0 este perfect legal şi spune că octeţii până la *Număr de confirmare - 1 inclusiv au fost* recepţionaţi, dar receptorul dorește o pauză
- *Sumă de control,* este calculată pentru antet, informaţie şi pseudo-antet.
- Pseudo-antetul conţine adresele IP ale maşinii sursă şi destinaţie, de 32 de biţi fiecare, numărul de protocol pentru TCP (6) şi numărul de octeţi al segmentului TCP (incluzând şi antetul).

Campuri...

- Câmpul *Opţiuni a fost proiectat pentru a permite adăugarea unor facilităţi suplimentare neacoperite* de antetul obişnuit.
- Cea mai importantă opţiune este aceea care permite fiecărei maşini să specifice încărcarea maximă de informaţie utilă TCP pe care este dispusă să o accepte.
- În timpul iniţializării conexiunii, fiecare parte anunţă dimensiunea maximă acceptată şi aşteaptă de la partener aceeaşi informaţie.
- Câştigă cel mai mic dintre cele două numere.

ELEMENTE DE PERFORMANŢĂ

- Când sunt interconectate sute sau mii de calculatoare, au loc adesea interacţiuni complexe cu consecinţe nebănuite, care conduc în mod frecvent la performanţe slabe fără ca cineva să ştie de ce.
- Cinci aspecte de performanţă ale reţelei:
- Probleme de performanţă.
- Măsurarea performanţei reţelei.
- Proiectarea de sistem pentru performanţe mai bune.
- Prelucrarea rapidă TPDU.
- Protocoale pentru reţele viitoare de mare performanţă.

Probleme de performanţă

- Unele probleme de performanţă, cum este congestia, sunt cauzate de supraîncărcarea temporară a resurselor.
- Performanţele se degradează de asemenea în cazul unei dezechilibrări structurale a balanţei resurselor.
  - Dacă o linie de comunicaţie de un gigabit este ataşată la un terminal PC de performanţă scăzută, procesorul slab nu va putea prelucra suficient de repede pachetele care sosesc, unele din acestea pierzându-se. Aceste pachete vor fi retransmise în cele din urmă, adăugând întârzieri, consumând din lărgimea de bandă şi în general reducând performanţele.

Probleme de performanţă...

- Supraîncărcarea poate fi iniţiată în mod sincron.
- Performanţele pot fi slabe datorită unei proaste reglări a sistemului.
  - De exemplu, dacă o maşină are suficientă memorie şi putere de prelucrare, dar nu a fost alocat suficient spaţiu pentru tampoane, vor apărea aglomerări şi se vor pierde TPDU-uri. Similar, dacă algoritmul de planificare nu acordă o prioritate suficient de mare prelucrării TPDU-urilor care sunt recepţionate, unele din ele se vor pierde.

Potrivirea de timp

- Potrivirea corectă a intervalelor de limită de timp. Atunci când este trimis un TPDU, în mod normal se poziţionează un contor pentru a evita pierderea TPDU-lui.
  - Dacă limita de timp este prea scurtă, se vor produce retransmisii inutile, obturând cablurile.
  - Dacă limita de timp este prea lungă, se vor introduce întârzieri inutile după pierderea unui TPDU.

Măsurarea performanţelor reţelei

- Ciclul de bază utilizat pentru îmbunătăţirea performanţelor reţelei conţine următorii paşi:
  - Măsurarea performanţelor şi a parametrilor relevanţi ai reţelei.
  - Încercarea de a înţelege ceea ce se petrece.
  - Modificarea unuia din parametri.
- Aceşti paşi se repetă până la atingerea unor performanţe suficient de bune sau până când este clar că şi ultima îmbunătăţire posibilă a fost pusă în aplicare

Masurarea

- Cel mai simplu mod de măsurare este iniţializarea unui contor la începutul unei activităţi şi observarea timpului necesar pentru îndeplinirea acelei sarcini.
- Alte măsurători sunt făcute cu contoare care înregistrează frecvenţa de apariţie a unor evenimente (de exemplu, numărul de TPDU-uri pierdute).
- Unii pot fi interesaţi să afle valorile unor mărimi, ca de exemplu numărul de octeţi prelucraţi într-un anume interval de timp.

Capcane

- Măsurarea performanţelor şi a parametrilor reţelei ascunde multe capcane potenţiale
- Dimensiunea testului trebuie să fie suficient de mare
- Testele trebuie să fie reprezentative
- Utilizarea ceasurilor cu granularitate mare trebuie făcută cu atenţie

Capcane

- Nu trebuie să se petreacă ceva neaşteptat în timpul măsurătorilor
- Lucrul cu memoria ascunsă poate distruge măsurătorile
- Trebuie înţeles ceea ce se măsoară
- Atenţie la extrapolarea rezultatelor

Proiectarea de sistem pentru performanţe superioare

- Reguli rezultate din experienţă:

- Regula #1: Viteza procesorului este mai importantă decât viteza reţelei.
- Regula #2: Reducerea numărului de pachete pentru a reduce supraîncărcarea datorată programelor.
- Regula #3: Minimizarea numărului de comutări de context.

Reguli

- Regula #4: Minimizarea numărului de copieri.
- Regula #5: Oricând se poate cumpăra mai multă lărgime de bandă, dar niciodată o întârziere mai mică.
- Regula #6: Evitarea congestiei este preferabilă eliminării congestiei.
- Regula #7: Evitarea întârzierilor.

MULȚUMESC!