



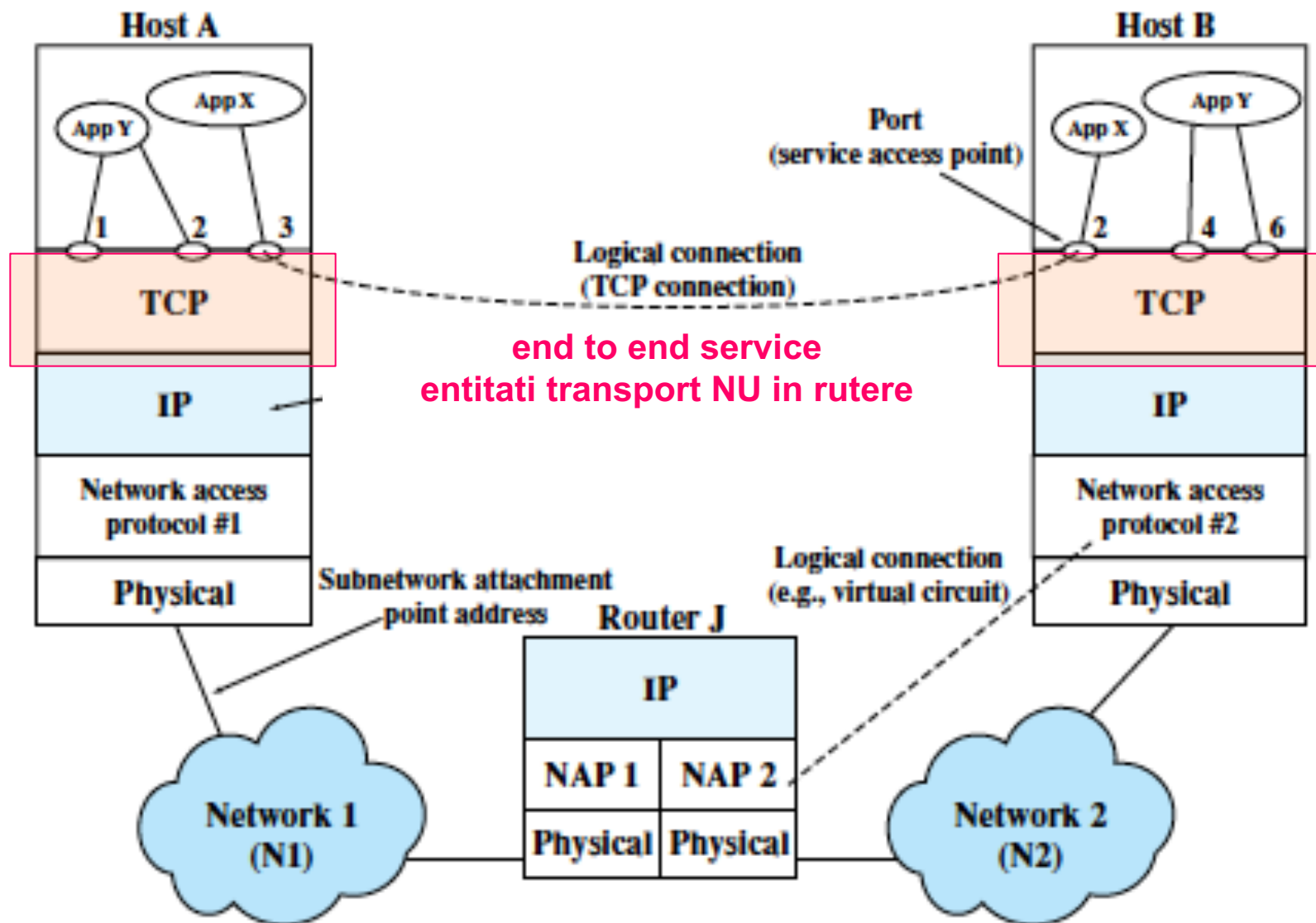
Nivelul transport



Cuprins

- Servicii si caracteristici
- socket API
- protocolul UDP
- protocolul TCP
 - gestiunea conexiunilor TCP
 - corectitudinea
 - controlul fluxului de date
 - mai multe numere de secventa
 - dimensiunea ferestrei receptorului
 - controlul congestiei, ceasurilor, time-out
- Real-time Transport Protocol
- TCP tranzactional

Nivelul transport in ierarhia TCP/IP

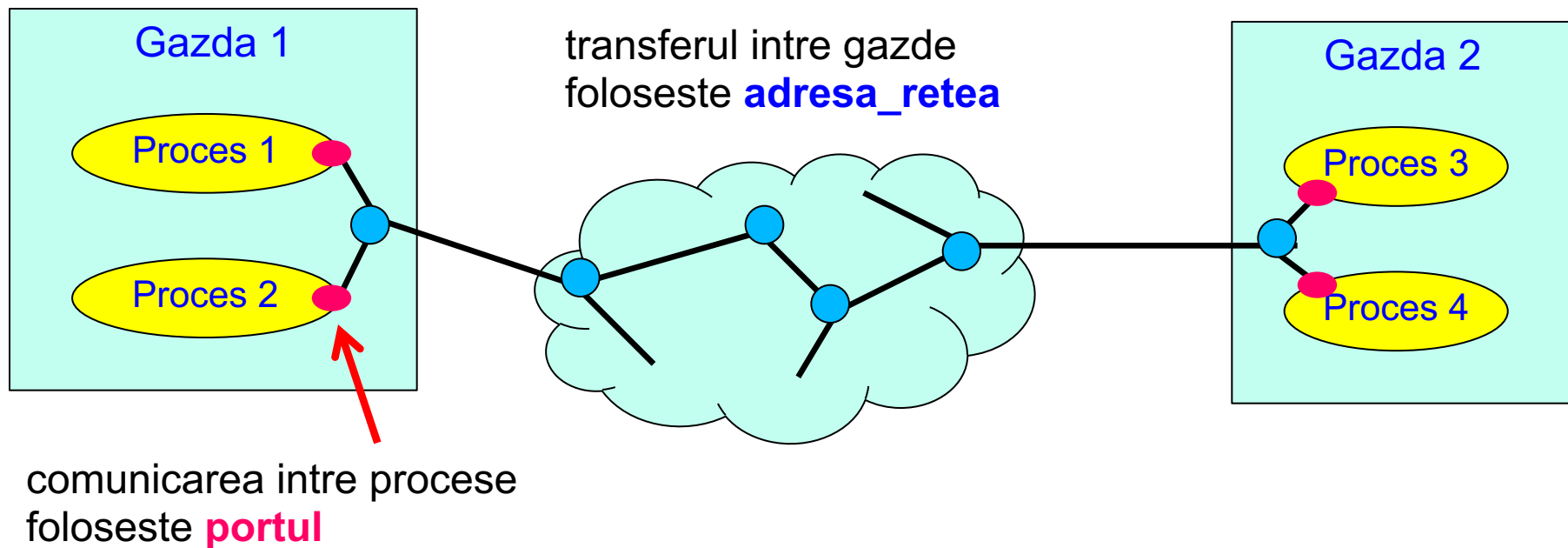


Comunicarea între aplicații

Reteaua asigură transmiterea **pachetelor** (datagrame) între **calculatoare** gazda

Transportul asigură comunicarea între aplicații

- legătura proceselor de aplicație cu **punctele** de acces la rețea
- **identificarea** unică a unui **punct de acces** prin **<adresa_rețea, port>**





Servicii ale nivelului Transport

- **Servicii furnizate**

- transfer de date între procese de aplicație, folosind rețele de diverse tipuri
- interfața uniformă cu utilizatorii

- **Caracteristici**

- două tipuri de servicii:
 - orientate pe conexiune (connection oriented) - TCP
 - fără conexiune (connectionless) - UDP



Socket interface

- Serviciile nivelului transport sunt accesibile ca **API** – Application Programming Interface
- Oferită ca bibliotecă utilizator sau funcții OS
 - API descrie cum se apelează aceste funcții
- **Socket API**
 - Originară din Berkeley BSD UNIX
 - Disponibilă și pe Windows, Solaris, etc.
 - **socket** = punctul în care procesul de aplicație se atașază la rețea
 - identificat prin descriptor - **număr** (ca la fișiere)
- Nu e standard de jure ci *standard de facto*



Creare socket

```
int socket(int family, int type, int protocol);
```

```
socket_descr = socket (protocol_family, comm_type, protocol)
```

- deschide un socket
- intoarce `socket_descriptor` folosit in apelurile urmatoare

`protocol_family` selecteaza familia de protocoale

PF_INET - protocoale **Internet**

PF_APPLETALK - protocoale AppleTalk etc.

`comm_type` selecteaza tipul de comunicare

SOCK_DGRAM - fara conexiune - datagrama

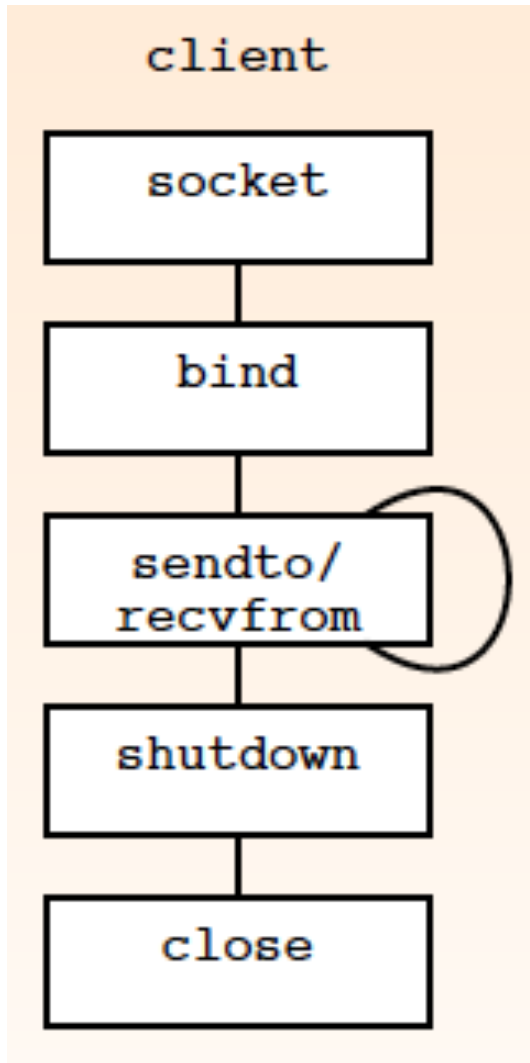
SOCK_STREAM - orientat pe conexiune – flux de octeti

`protocol` specifica protocolul

IPPROTO_TCP - TCP

IPPROTO_UDP - UDP

Serviciu fara conexiune



server este la fel!

bind - optional pentru initiator

Serviciu fara conexiune - server

server

socket

bind

sendto/
recvfrom

shutdown

close

Creaza socket si aloca-i resurse de sistem:

```
int s = socket (AF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

Asociaza un socket cu <port, adresa_IP>:

```
int bind (int socket_descriptor, struct sockaddr* local_address, int  
address_length)
```

```
bind(s, &addr, sizeof(addr));           // addr locala - vezi slide urmator !
```

Primește mesaje de la un socket aflat la distanță:

```
int recvfrom (int socket_descriptor, char* buffer_address, int  
buffer_length, int flags, struct sockaddr* sender_address, unsigned int  
sendaddress_length)
```

```
recvfrom(s, buf, BUFLen, 0, NULL, NULL);
```

Oprește trimitere sau/și recepție de date

```
shutdown (s, SHUT_RD / SHUT_RDWR / SHUT_WR)
```

Inchide socket - termina utilizarea socket si elibereaza resurse alocate

```
close (s)
```



Setare adresa

Format TCP/IP:

```
struct sockaddr_in { u_char  sin_len;    /* total length of address */
    u_char  sin_family;    /* family of the address */
    u_short sin_port;      /* protocol port number */
    struct in_addr sin_addr; /* IP address */
    char    sin_zero[8];   /* unused */
}
```

```
struct sockaddr_in serv_addr
memset ((char *) &serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;           // adrese pentru Internet
serv_addr.sin_addr.s_addr = INADDR_ANY;
                                           // foloseste adresa IP a masinii
serv_addr.sin_port = htons(portno);
                                           // converteste de la host la network byte order
```

Serviciu fara conexiune - client

client

socket

bind

sendto/
recvfrom

shutdown

close

Creaza socket și alocă-i resurse de sistem:

```
int s = socket (AF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

Trimite mesaj la un socket aflat la distanță:

```
int sendto (int socket_descriptor, char* data_address, int data_length, int flags,  
struct sockaddr* dest_address, int destaddress_length)
```

```
sendto (s, buf, BUFLen, 0, &addr, sizeof(addr));
```

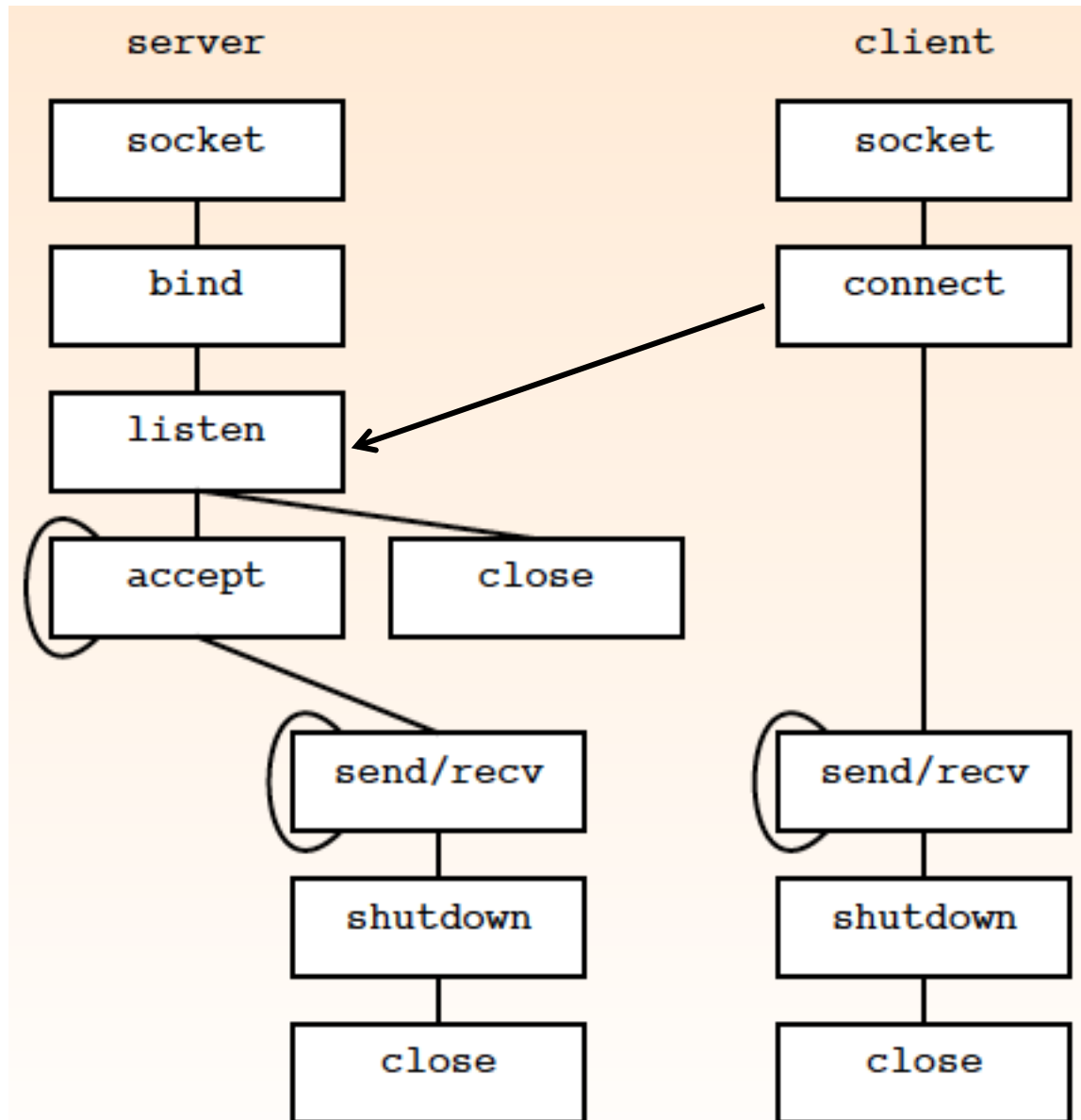
Oprește trimitere sau/și recepție de date

```
shutdown (s, SHUT_RD / SHUT_RDWR / SHUT_WR)
```

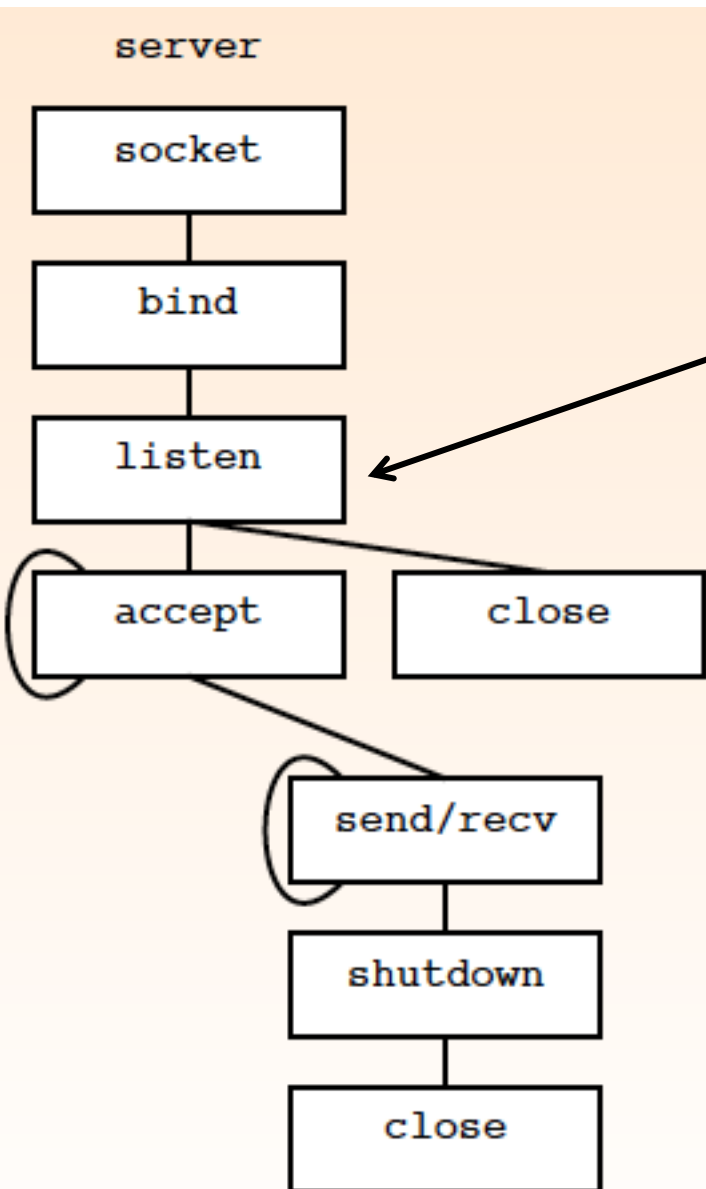
Termină utilizarea socket și eliberează resursele alocate

```
close (s)
```

Serviciu orientat pe conexiune



Serviciu orientat pe conexiune - server



1. Creaza socket:

```
int ls = socket (AF_INET, SOCK_STREAM,
                 IPPROTO_TCP);
```

2. Asociaza un socket cu <port, adresa_IP>:

```
bind(ls, &addr, sizeof(addr));
```

3. Asteapta cereri de conectare la socket de ascultare (declara nr max cereri asteapta):

```
int listen (int socket_descriptor, int queue_size)
```

```
listen(ls, 5);
```

4. repetat - Accepta o cerere de conectare de la un client; creaza un nou socket pentru conexiune:

```
int accept (int listen_socket_descriptor, struct sockaddr*
            client_socket_addr, int* client_addrlen)
```

```
int s = accept(ls, NULL, NULL);
```

5. repetat - Trimite / primește pe s etc.

Serviciu orientat pe conexiune - client

1. Creaza socket:

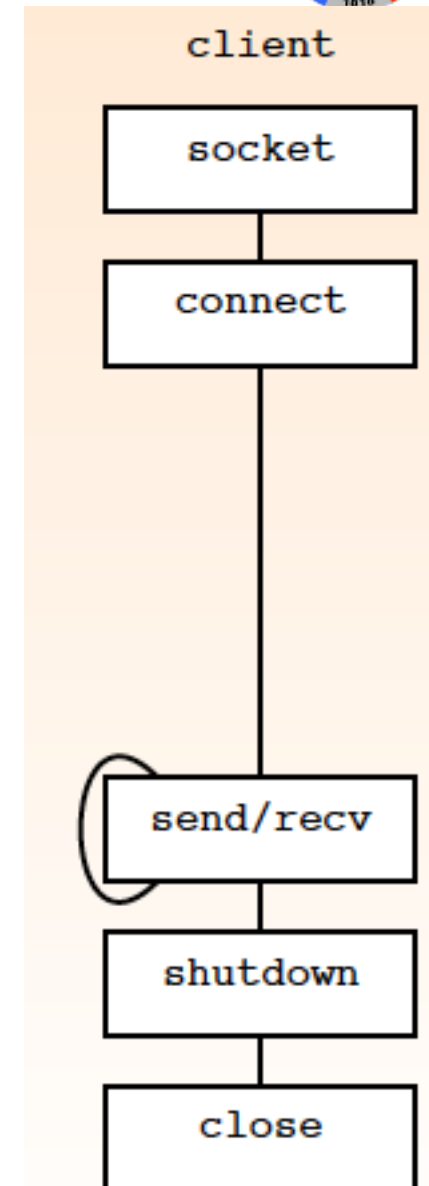
```
int s = socket (AF_INET, SOCK_STREAM,  
                IPPROTO_TCP);
```

2. Conecteaza un socket client (specificat prin descriptor) cu un socket server (precizat prin adresa):

```
int connect (int client_socket_descriptor, struct sockaddr*  
            server_socket_address, int  
            server_sockaddress_length)
```

```
connect(s, &addr, sizeof(addr));
```

3. repetat - Trimite / primește etc.



Transmisia de date cu TCP

send (s, buf, len)

`int send(int socket, const char* buf, int len, int flags);`

- Intoarce numarul de octeti trimisi
 - **Poate fi mai mic decat len !**

recv (s, buf, max_len)

`int recv(int socket, char* buf, int len, int flags);`

- Intoarce numarul de octeti primiti
 - **Poate fi mai mic decat max_len!**

`flags` indica optiuni speciale

MSG_OOB – trimite/primește date out-of-band

MSG_PEEK – livrează date primite, dar tratează ca necitite

Inchiderea conexiunii TCP

- Elibereaza resursele asociate conexiunii
- Informeaza capatul celalalt de inchiderea conexiunii
- API
 - **shutdown** (s, SHUT_RD/SHUT_RDWR/SHUT_WR)
`int shutdown (int socket, int how)`
 - opreste primirea – rejecteaza datele care sosesc
 - opreste transmiterea – ignora datele inca netrimise
 - ambele
 - **close** (s)
`int close (int socket);`
 - inchide socket (elibereaza structurile de date din kernel)



Protocoale de Transport

Doua potocoale majore

- UDP
- TCP

Aspecte discutate

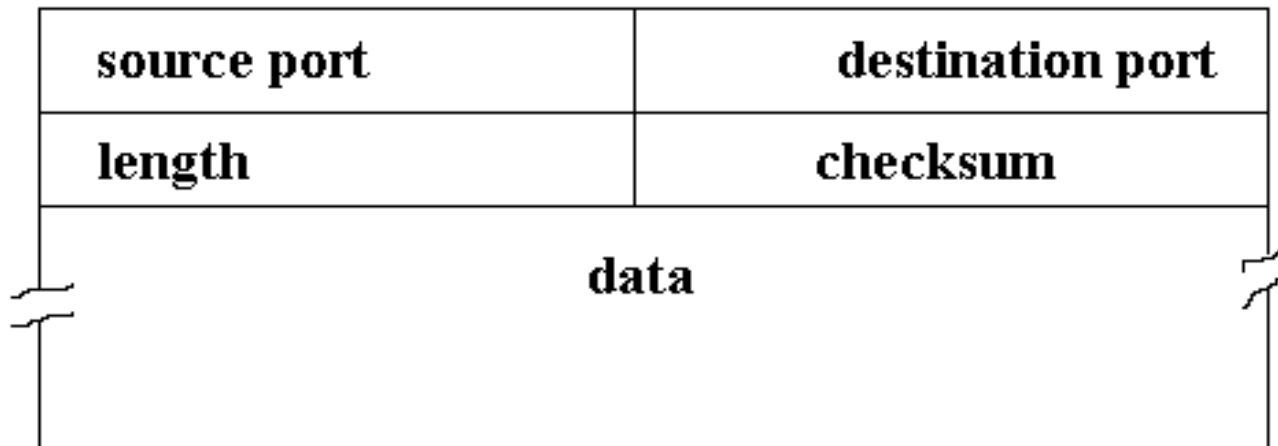
- Formatul datelor
- Adresare
- Functionare



UDP - User Datagram Protocol

- UDP livreaza datagrame utilizator - *user datagrams*
 - Livrare “Best effort” – datagramele pot fi pierdute, primite in alta ordine etc.
 - Sume de control pentru integritate
- Puncte de capat UDP = *protocol ports* sau *ports*
- UDP identifica *adresa Internet* si *numar port* pentru sursa si destinatie
- *Destination port* si *source port* pot diferi.

Antet UDP



checksum (calculat la fel ca la TCP) dar nefolosit

Nu control flux

Nu control erori

Nu retransmisie

Aplicatii care folosesc UDP

- DNS
- Voice over IP
- Online Games
- Se foloseste atunci cand:
 - Latența este foarte importantă
 - Livrarea tuturor datelor nu e necesara
 - Retransmisiile sunt implementate de aplicatii cand e nevoie (DNS)



TCP - Transmission Control Protocol

Livrare sigura pe retea nesigura (datagrame)

- ***Cel mai folosit protocol de transport***

Web

Email

SSH

Chat

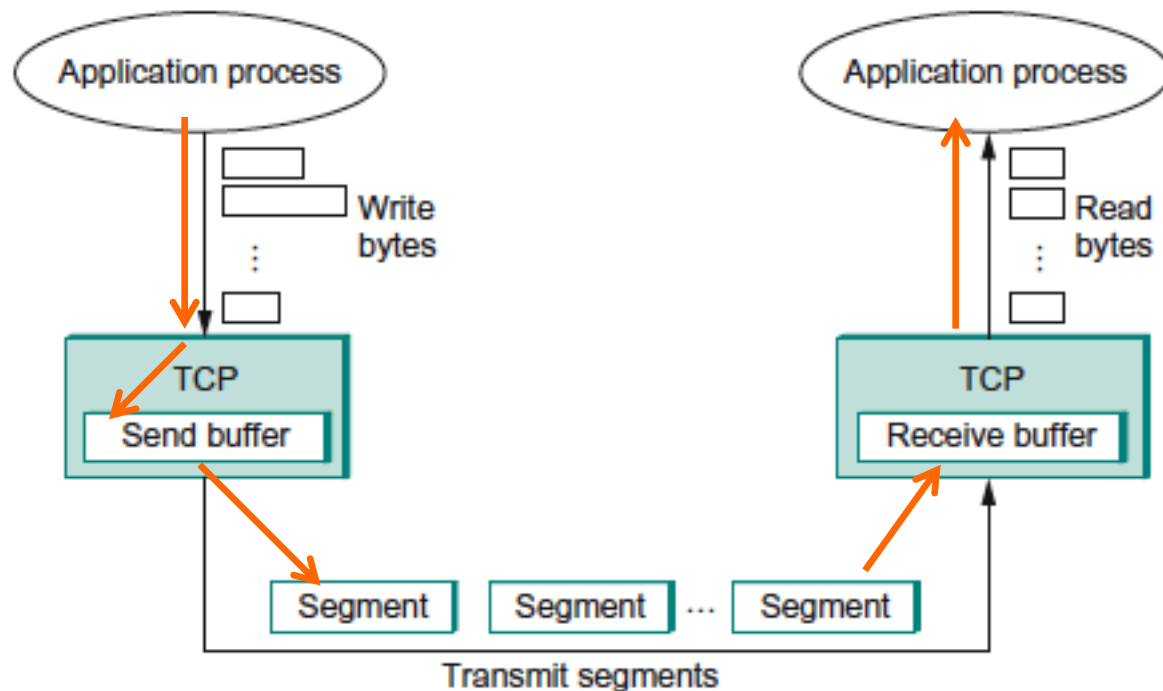
Video streaming

Peer-to-peer

Cateva porturi standard

Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial File Transfer Protocol
79	Finger	Lookup info about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

TCP este orientat pe flux de octeți

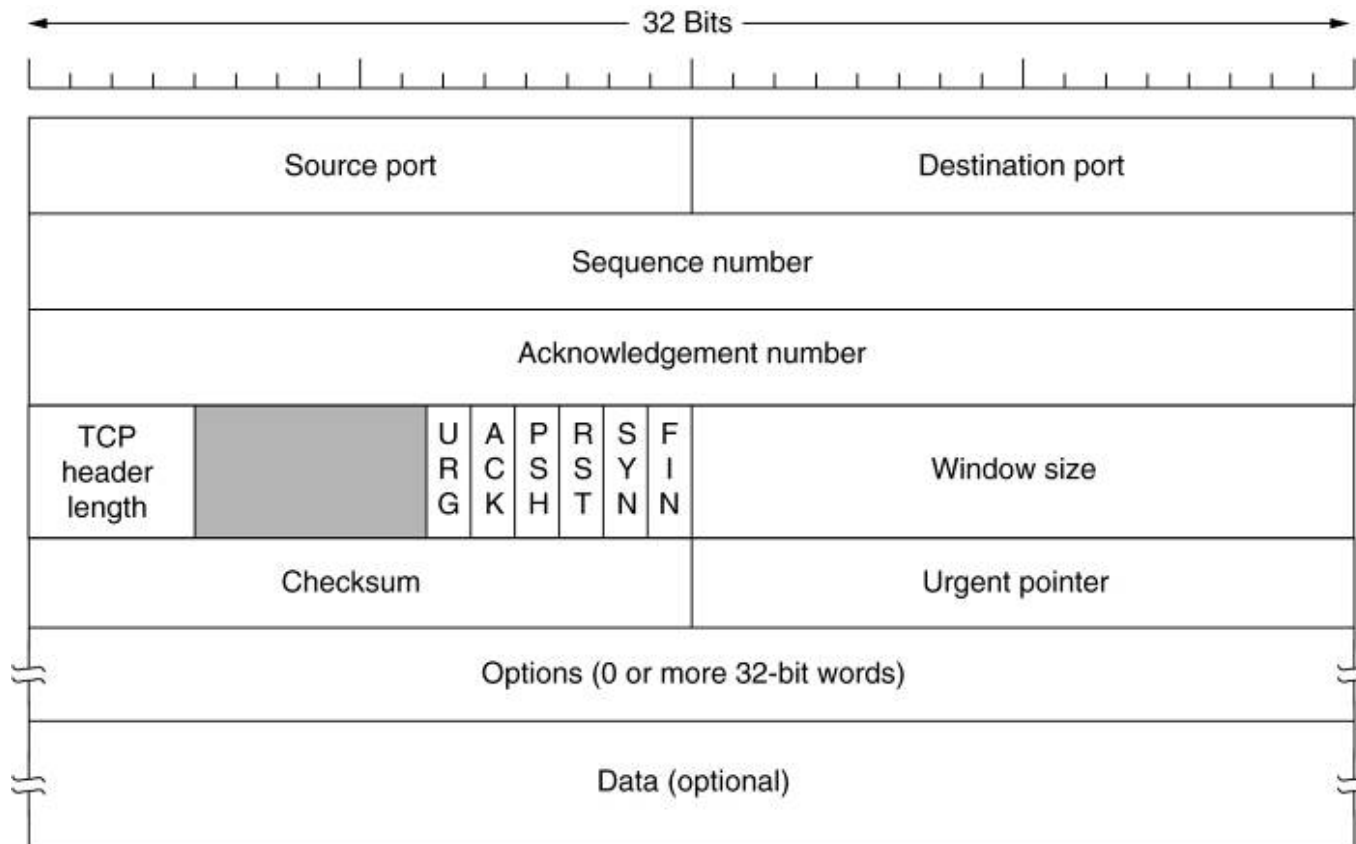


- Aplicația transmițătoare scrie octeți în conexiunea TCP
- TCP la sursă memorează octeții într-un buffer și transmite segmente
- TCP la destinație memorează segmentele într-un buffer
- Aplicația receptoare citește octeți (câți vrea!) din conexiunea TCP

Caracteristici

- Orientat pe conexiune
- Interfață **flux** (Stream)
 - transmisie și recepție siruri de octeți
- Face **controlul congestiei** adaptând viteza de transmisie la condițiile rețelei
- Garantează **transmisie în ordine și sigură** a datelor pe o conexiune
- **Full duplex**
- Stabilire sigură a conexiunii - three-way handshake
- Eliberare lină a conexiunii – fără pierdere de date

Antet segment TCP



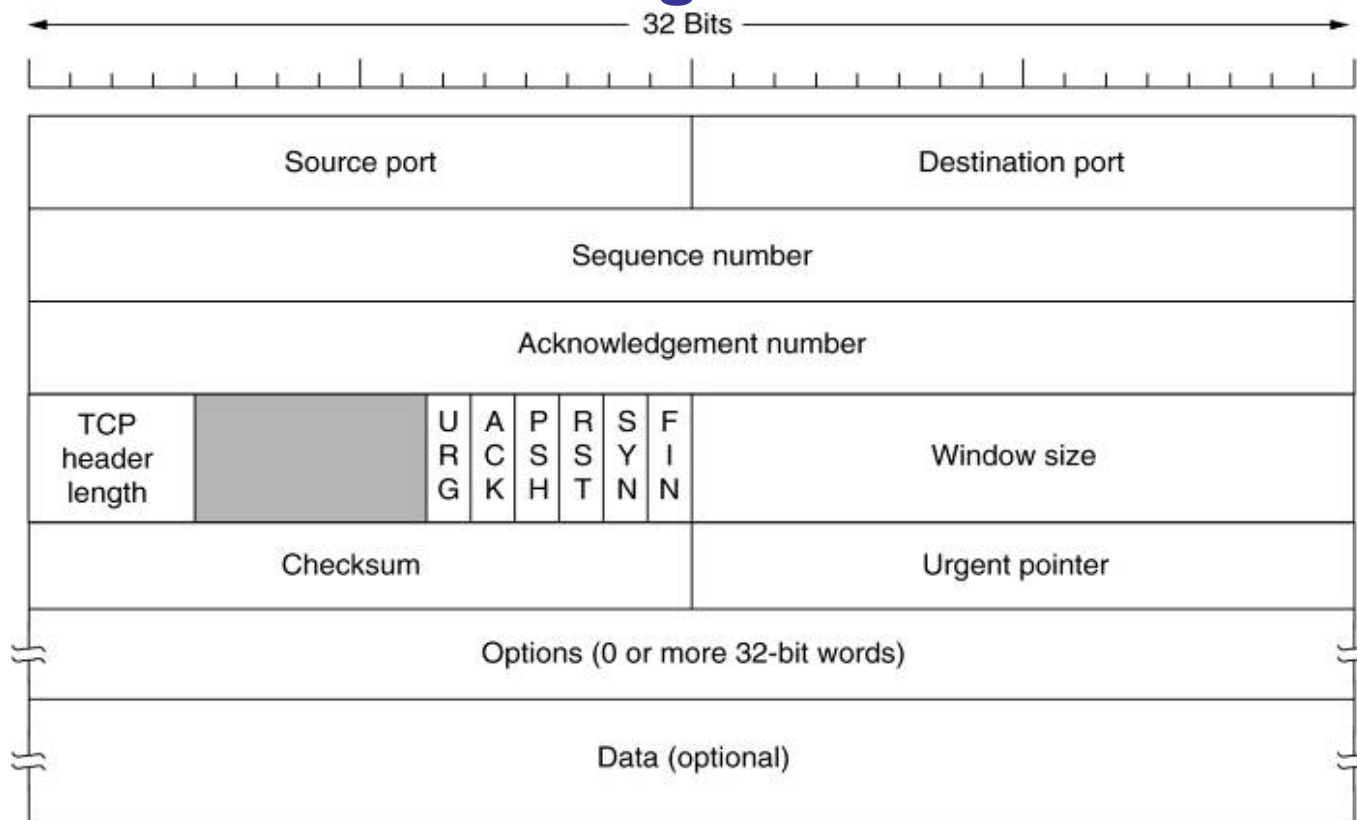
Sequence number – numărul **primului** octet din segment

Acknowledgement number – numărul **următorului** octet așteptat

Window size - numărul de **octeți** care pot fi trimiși, începând cu octetul confirmat

Urgent Pointer – **deplasamentul**, față de *Sequence number*, ptr. info. urgentă

Antet segment TCP



URG Urgent pointer valid

ACK Acknowledge Number valid

PSH - push information to user

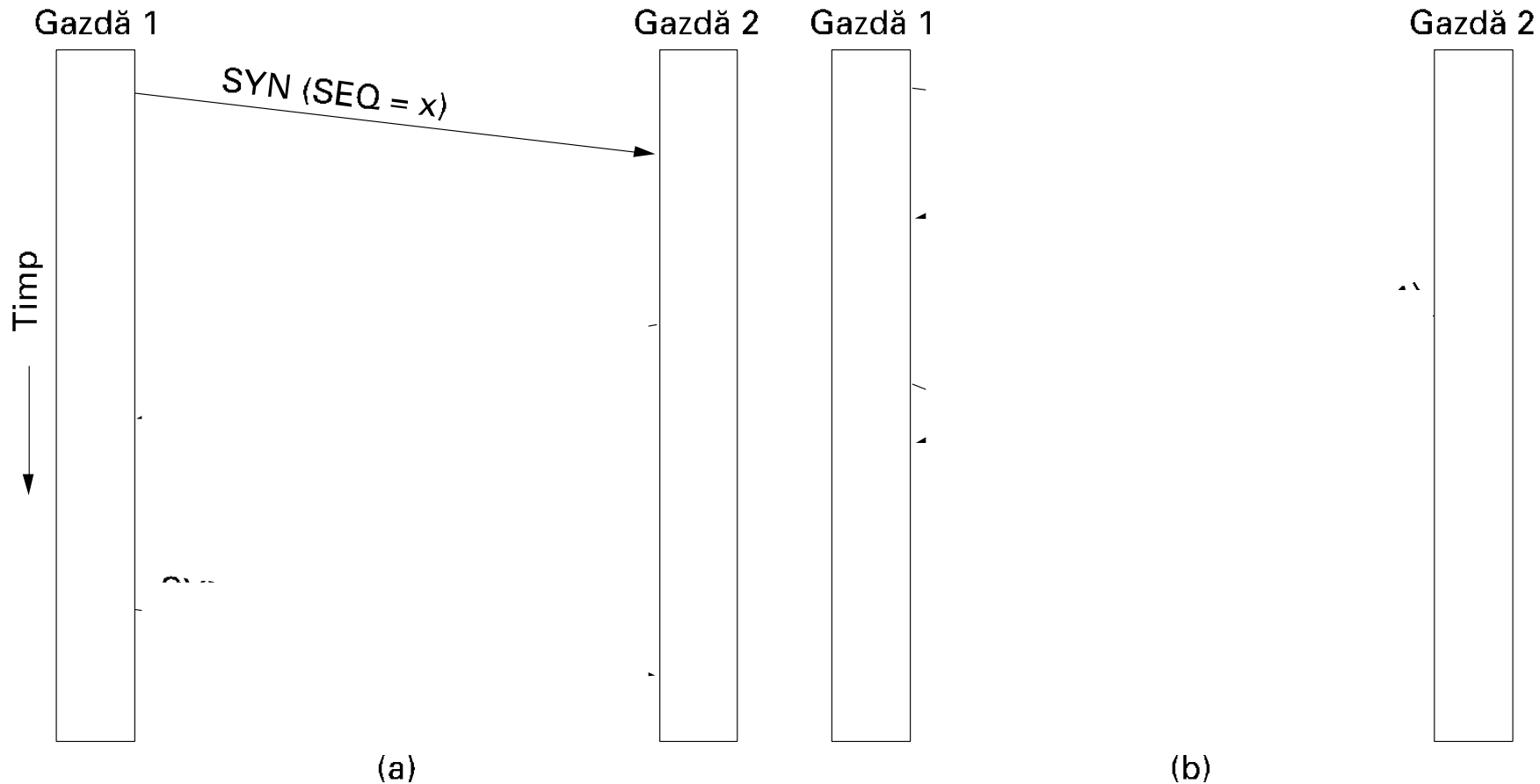
RST - close a connection due to an error

SYN - open connection

FIN - close a connection

Options: e.g. max TCP payload (implicit 536 octeti), selective repeat

Stabilirea conexiunii - Three way handshaking



(a) Cazul normal.

(b) Coliziune.

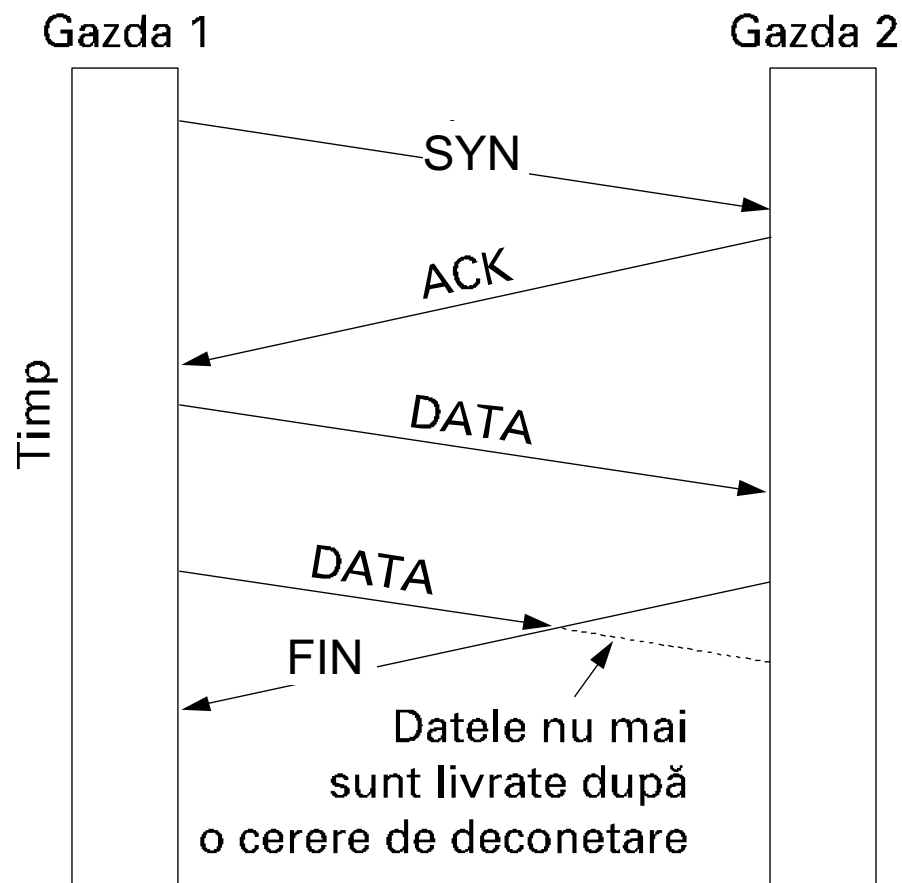
Rejectarea conexiunii

A**B****A****B**

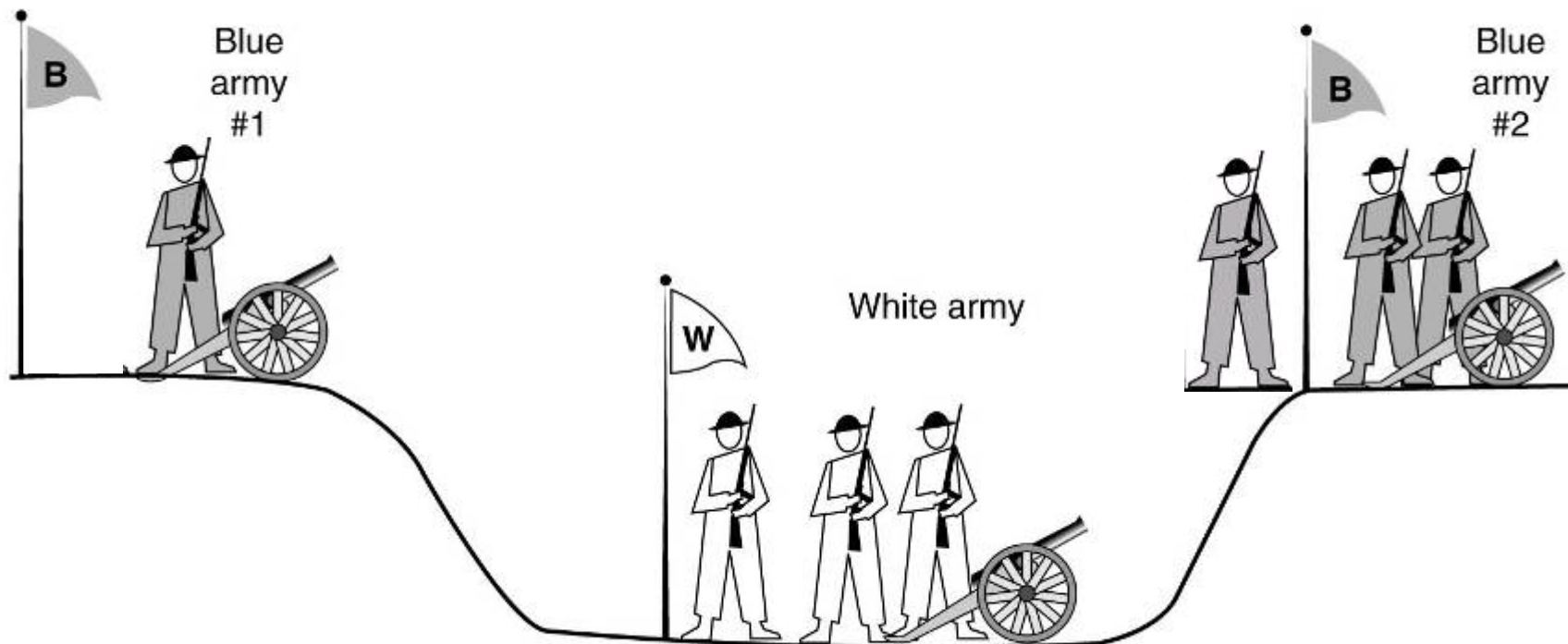
SYN intarziat
B accepta
A rejecteaza

A initiaza conexiune (SYN i) + SYN k intarziat
A refuza (RST)
B accepta SYN i – raspunde cu SYN j

Deconectare abruptă cu pierdere de date

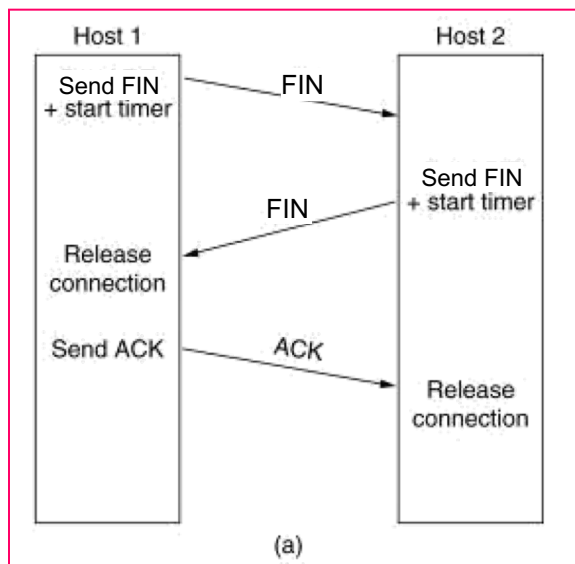


Problema celor doua armate

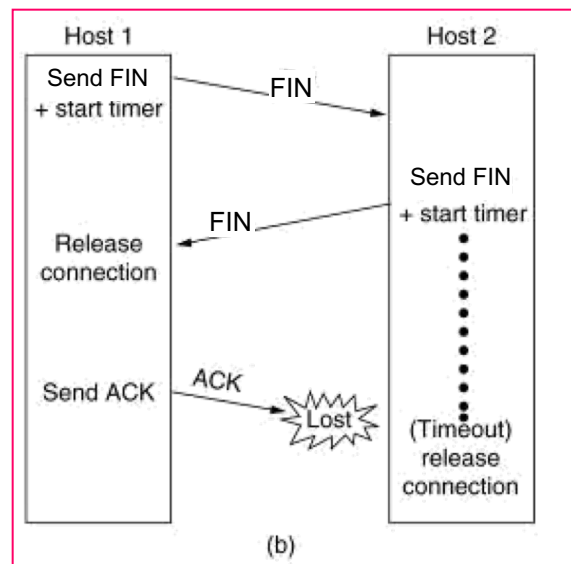


Patru scenarii de eliberarea conexiunii

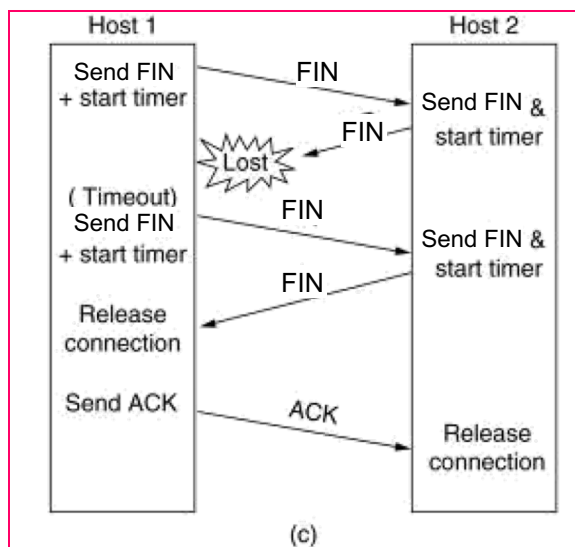
normal



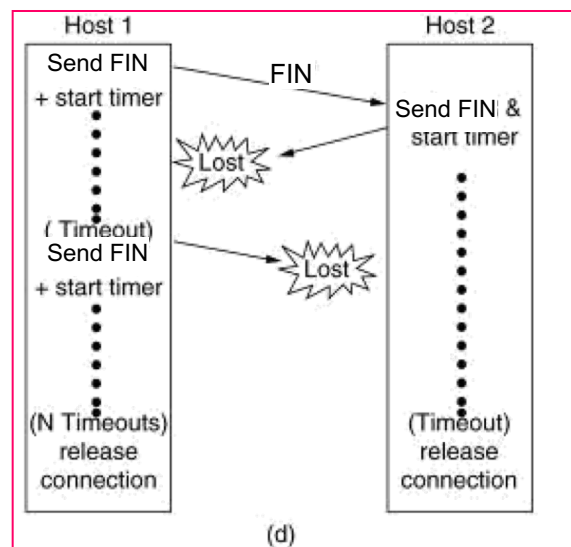
ACK final
pierdut



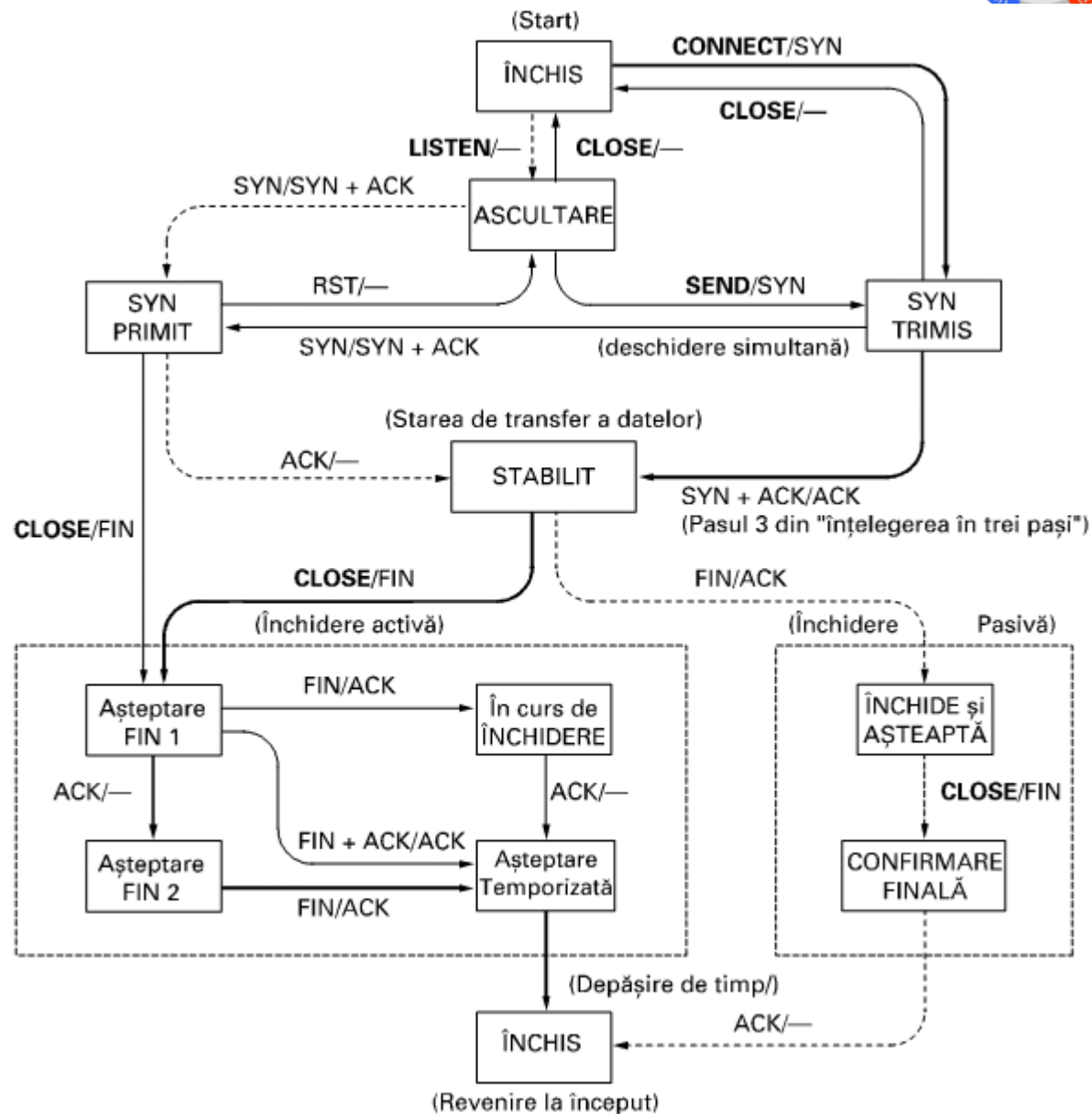
raspuns
pierdut



Raspuns si
FIN pierdute



- client (**activ**) cere conectarea
- server (**pasiv**) raspunde
- Linie groasa continua = cale normala pentru client
- Linie intrerupta = cale normala server
- Linii subtiri = evenimente exceptionale
- Fiecare tranzitie are eticheta **eveniment / actiune**

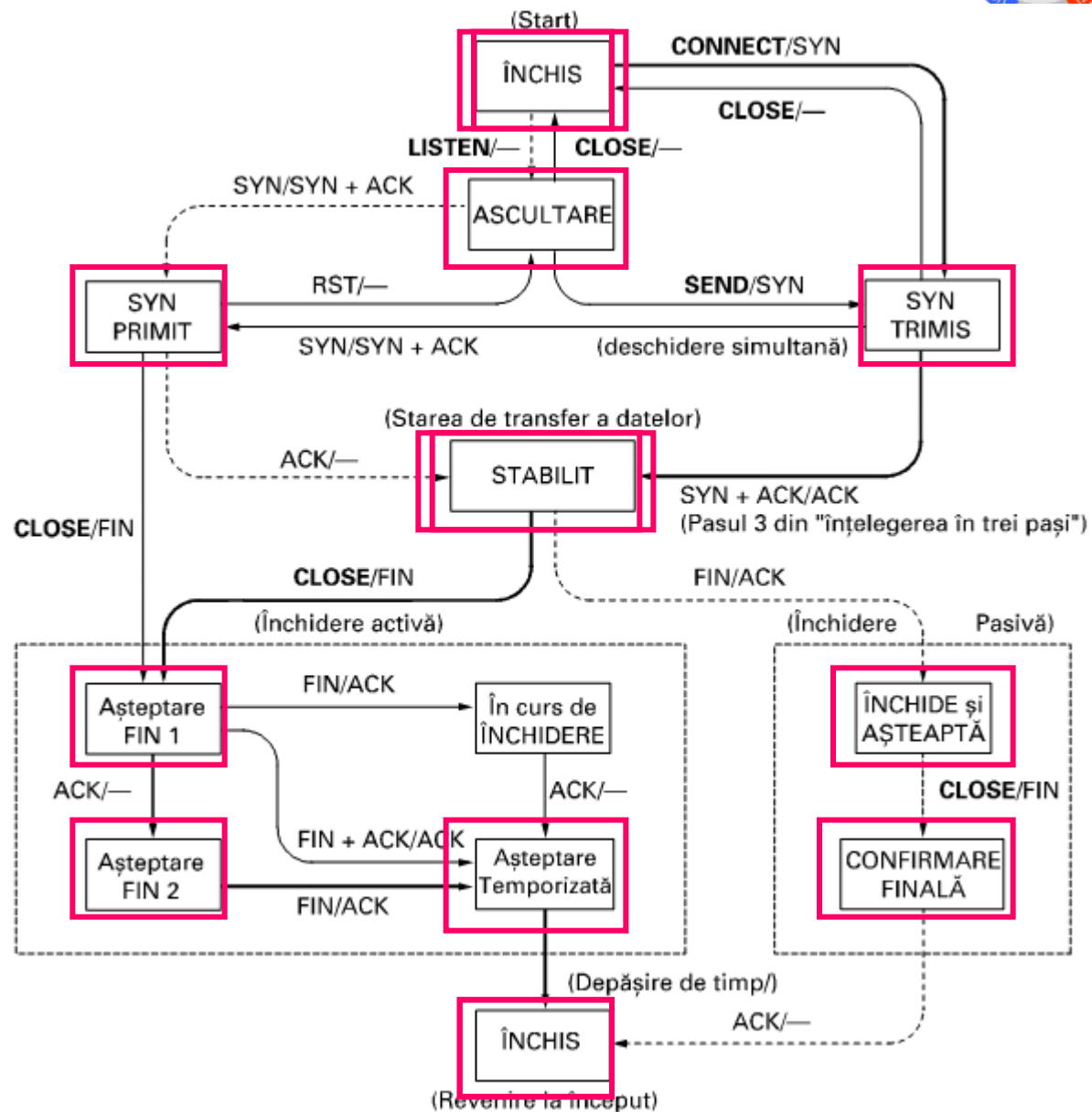


Client

Linie groasa continua

Server

Linie intrerupta



Corectitudinea segmentelor TCP

1 - Suma de control (checksum) din antet are 16 biti si include

1. antet
2. incarcatura segment TCP (date)
3. un pseudo-antet

adresele IP sursa si destinatie

protocolul (6 pentru TCP)

lungime segment TCP (include antetul)

Algoritm: la **transmisie**

aduna cuvinte de 16 biti in complement fata de 1

complementeaza rezultatul

scrie rezultatul in antet

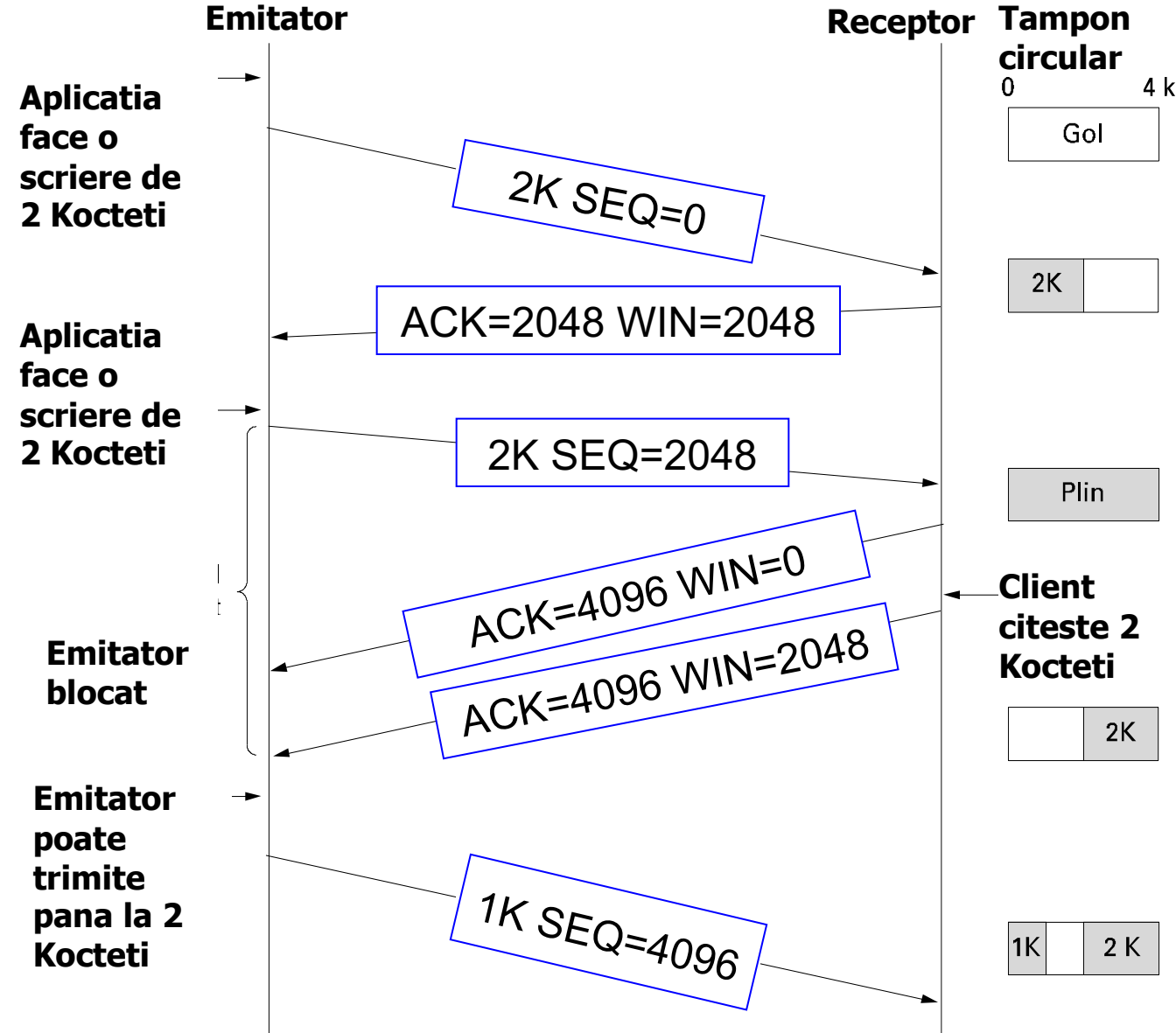
la **receptie**

aduna cuvinte de 16 biti – rezultatul trebuie sa fie zero

2 - **Acknowledgement** number

Corectia se face prin retransmisie

Controlul fluxului de date



Probleme dimensiuni câmpuri antet

Numere secvență de 32-biti

- Durata ciclu de numarare – depinde de viteza transmisie
 - 1 saptamana pentru 56kbps
 - 57 min pentru 10Mbps
 - 34 sec pentru 1Gbps (**sub 120 sec care este timp viata maxim in Internet**)

Problema: pot apare segmente diferite, cu acelasi numar de secventa?

Soluție

- Folosire opțiuni TCP (RFC 1323)
 - TCP Timestamps
 - asociaza o amprenta de timp fiecarui segment
 - rezolva numere de secventa duplicate



Probleme dimensiuni câmpuri antet (2)

Fereastra receptor (camp **Window size** de 16 biti – echivalent 64 KB)

Transmitere **500 Kb** pe legatura **1 Gbps** ocupa 500 μ sec

La intarziere de **20 ms** pe sens confirmarea se primeste dupa **40 ms** =>
ocupare canal pe un ciclu complet este mică - **1.25%**

Ocupare completă in ambele direcții: produs **bandwidth*delay** =
 $= 1 \text{ Gbps} * 40 \text{ ms} = 40 \text{ milioane biti}$

Condiții - fereastra receptor ar trebui sa fie \geq **bandwidth*delay**

- camp **Window size** nu se poate mari

Soluție

- Folosire **opțiuni** TCP (RFC 1323)
 - **Window Scale** – factor de scalare a câmpului **Window size** de până la **2^{14}** ori
 - ➔ ferestre de până la **2^{30}** octeți



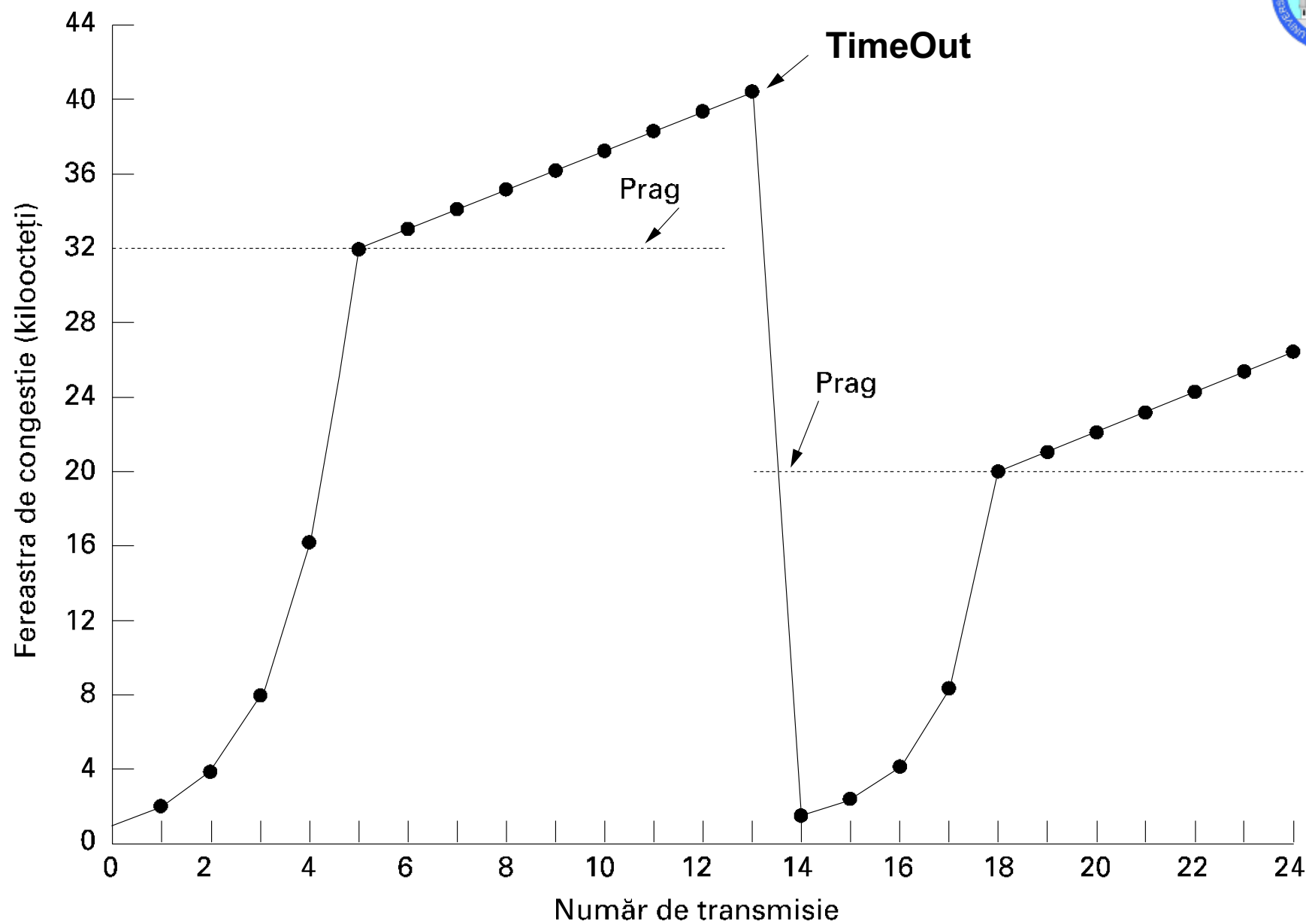
Controlul congestiei

- Fluxul de date transmis pe o conex. TCP limitat de minimul dintre:
 - dimensiunea fereastrei receptorului
 - capacitatea rețelei (**fereastra de congestie**)
- Algoritm de **stabilire fereastra de congestie**
 - transmite un segment de **dimensiune maximă** pe conexiunea stabilită
 - dubleaza volumul de date – rafală de segmente - (creștere exponențială) la fiecare transmisie confirmată la timp
 - la **primul timeout** opreste procedeul si fereastra ramane la valoarea ultimei transmisii confirmate la timp (fara timeout)

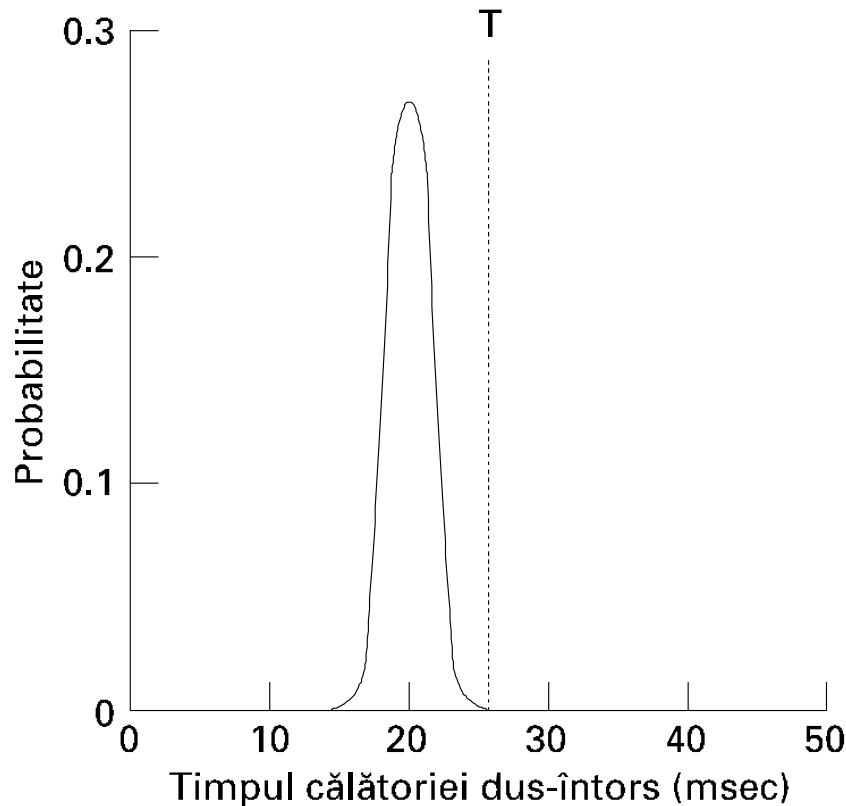


Controlul congestiei

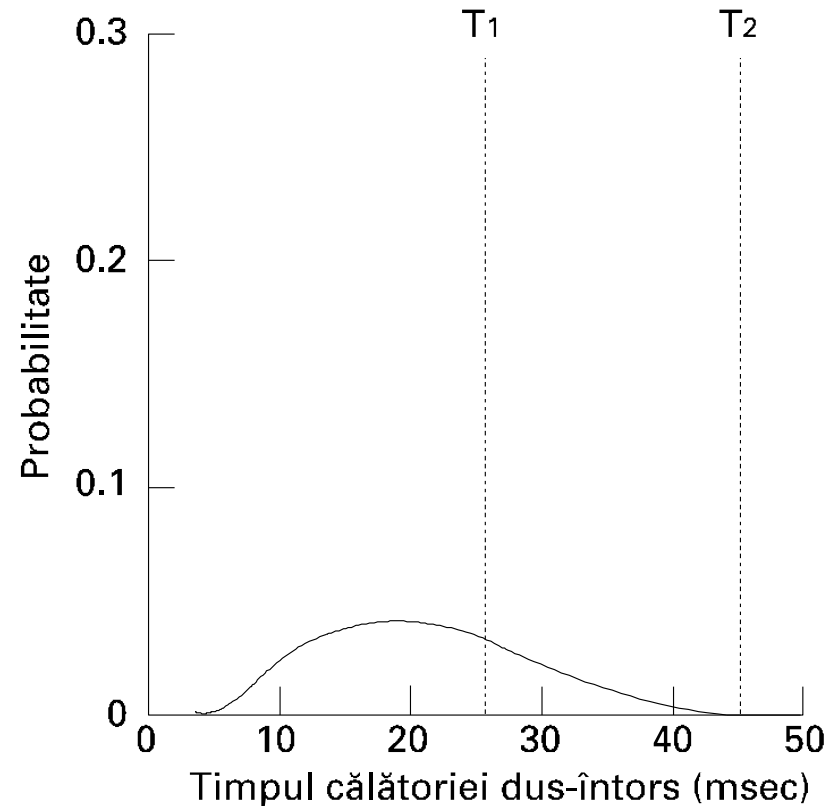
- Algoritmul de **control al congestiei**
 - folosește un **prag** (threshold)
 - la un timeout pragul setat la jumătate din fereastra de congestie
 - se aplica procedeul de creștere (exponentială) a ferestrei de congestie până se atinge pragul
 - peste prag se aplica o creștere liniară (cu câte un segment de dimensiune maximă o dată)



Gestiunea ceasurilor in TCP



(a)



(b)

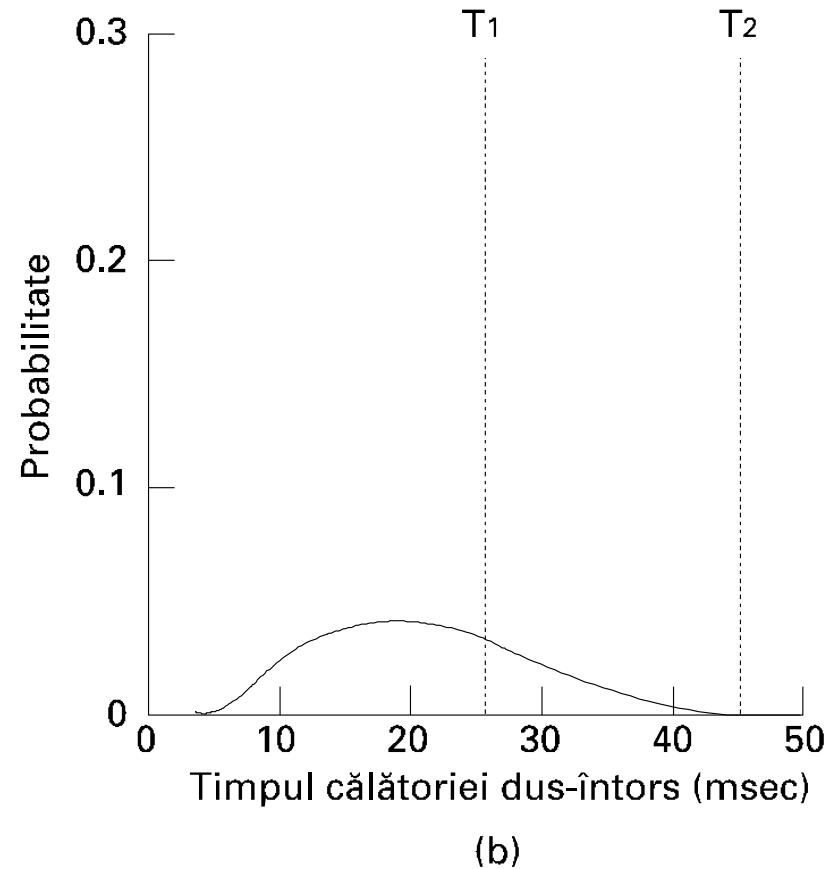
(a) Densitatea de probabilitate a timpilor de sosire ACK in nivelul **legatura de date**.

(b) Densitatea de probabilitate a timpilor de sosire ACK pentru **TCP**.

Gestiunea ceasurilor in TCP (2)

Setare proasta – performante slabe:

- Prea lung (T_2) – transmitatorul asteapta mult ptr retransmisie
- Prea scurt (T_1) – trafic inutil generat de transmitator



Stabilire time-out

- **Timeout** diferit la fiecare conexiune - setat dinamic
- Se folosesc metode empirice
- Transmisorul alege *Retransmission TimeOut* (**RTO**) pe baza *Round Trip Time* (**RTT**)

M este timpul masurat pana la primirea ack

$$\mathbf{RTT = \alpha * RTT + (1 - \alpha) * M \quad \text{cu } \alpha = 7/8}$$

$$\mathbf{RTO = \beta * RTT \quad \text{cu } \beta = 2}$$

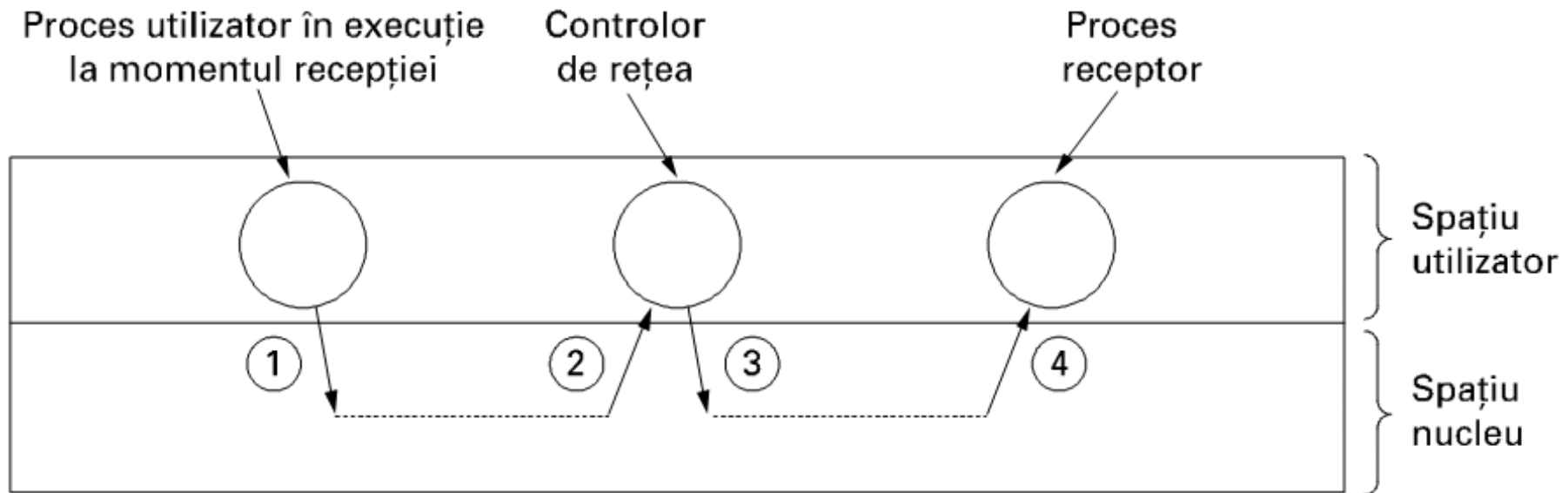
- Alegere dupa *deviatia standard* (*DS*);

D aproximeaza DS

$$\mathbf{D = \alpha * D + (1 - \alpha) * |RTT - M|}$$

$$\mathbf{RTO = RTT + 4 * D}$$

Proiectarea pentru performanța (optional)



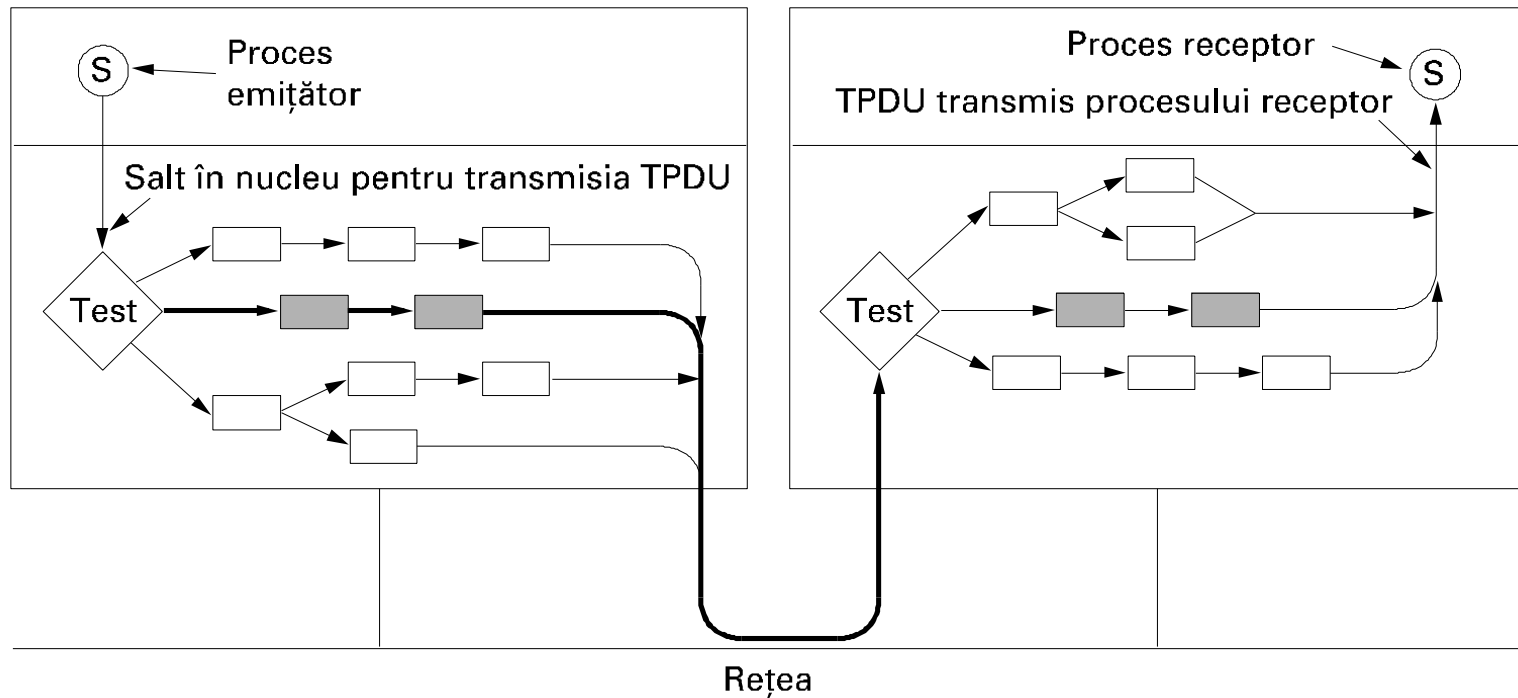
Problema: Multe **schimbări de context user – kernel** pentru a manevra pachete cu un Controlor de rețea în spațiul utilizatorului.

Controlorul assemblează segmentele pentru a le furniza apoi procesului receptor

Soluția: **reducerea** numărului schimbări de context - acumularea mesajelor sosite, în memorie tampon și livrarea în grup către utilizator.

Similar, gruparea mesajelor de transmis, în memorie tampon și trimiterea lor grupată

Prelucrare rapida TPDU (optional)



Calea rapida intre transmitator si receptor este cu line groasa. Pasii sunt reprezentati cu gri.

Test caz normal:

starea = ESTABLISHED
 nu se incerca inchiderea conexiunii,
 TPDU normal (nu out-of-band),
 suficient spatiu la receptor

} → alege fast path

Prelucrare rapida TPDU (2)

Source port		Destination port	
Sequence number			
Acknowledgement number			
Len	Unused		Window size
Checksum		Urgent pointer	

(a)

(a) Antet TCP.

VER.	IHL	TOS	Total length		
Identification					Fragment offset
TTL		Protocol	Header checksum		
Source address					
Destination address					

(b)

(b) Antet IP.

Transmitator

Pastreaza un **prototip** de mesaj in entitatea de transport – campuri nemodificate in unitati de date consecutive; la fel pentru pachet IP

Campurile gri sunt luate din **prototip** fara modificari. Celelalte se calculeaza pentru fiecare segment

Receptor

- Localizeaza inregistrarea conexiunii din TPDU intr-o **tabela** hash
- Testeaza pentru cazul normal (similar cu transmisia)
- Actualizeaza inregistrarea conexiunii (starea curenta)
- Copiaza datele la utilizator si calculeaza suma de control
- Transmite confirmarea

Prelucrare Fast TPDU (timer management)

“Timing wheel.”

1 slot = 1 clock tick

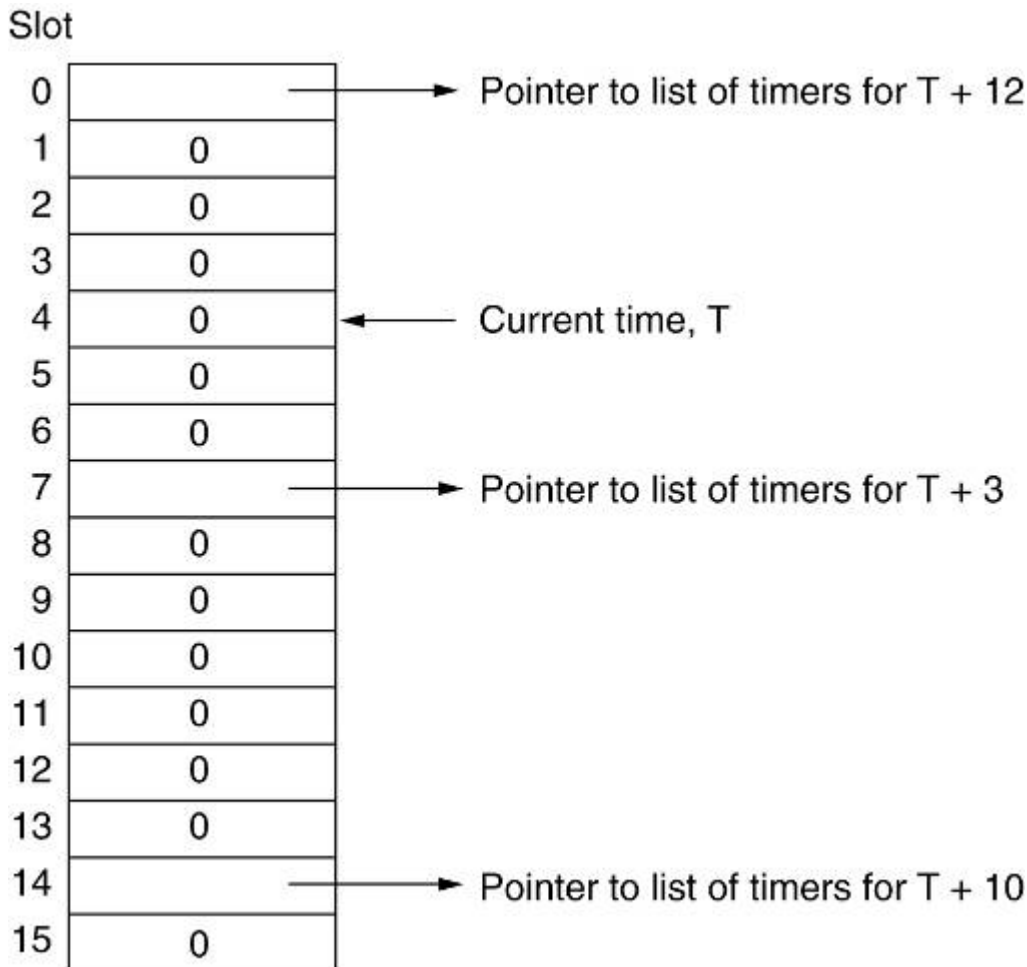
Time current $T=4$

Programare time-out peste 7 tick-uri
→ insereaza eveniment in lista de la slot 11

Anulare → cauta in lista de la slot corespunzator si elimina eveniment

La fiecare **Clock tick**, un pointer avaneaza cu un slot, circular

Daca slot nevid, proceseaza toate evenimentele





The Real-Time Transport Protocol

Folosit pentru aplicatii multimedia de timp real

- muzica sau video la cerere, videoconferinte etc.

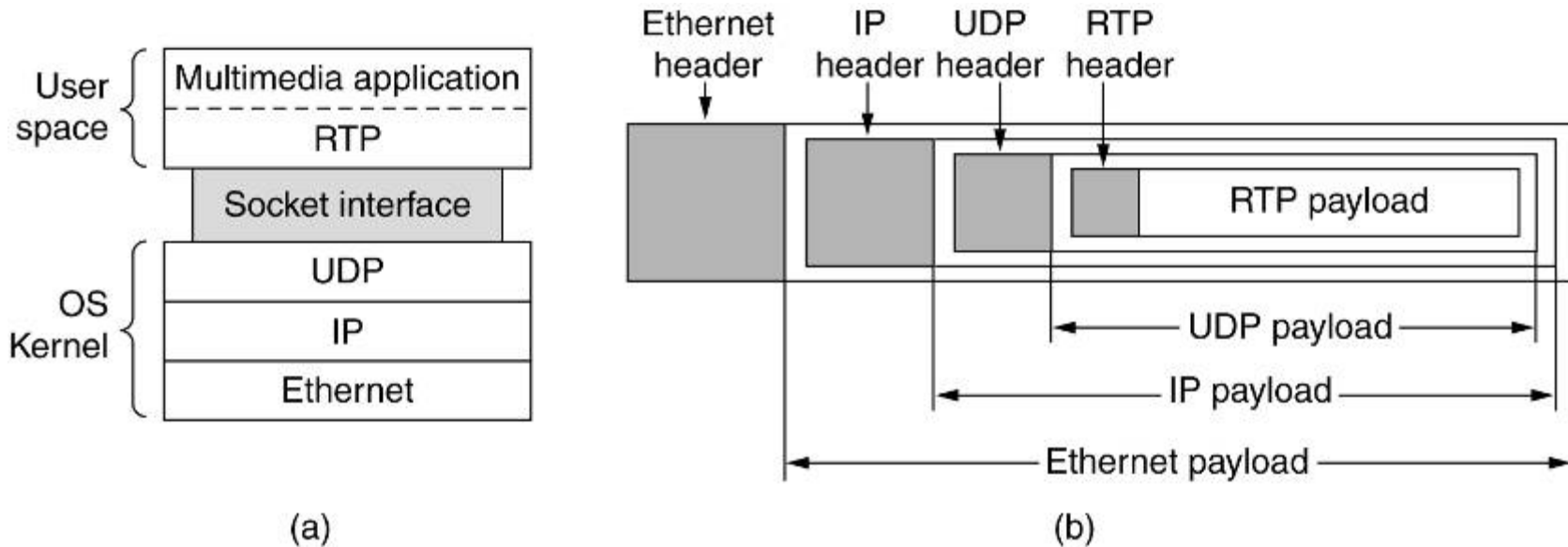
Funcție principală: multiplexare fluxuri RTP și transmiterea lor ca un **singur șir** de pachete **UDP**

La recepție, RTP livrează aplicației datele multimedia

Fără retransmiterea pachetelor

- UDP nu asigură corectitudinea transmisiei
- RTP atribuie un număr de **secvență** fiecărui pachet
- **tratarea** unui pachet pierdut se face **la aplicație**: receptorul poate “interpola” pachetul absent sau ignora eroarea

The Real-Time Transport Protocol (2)



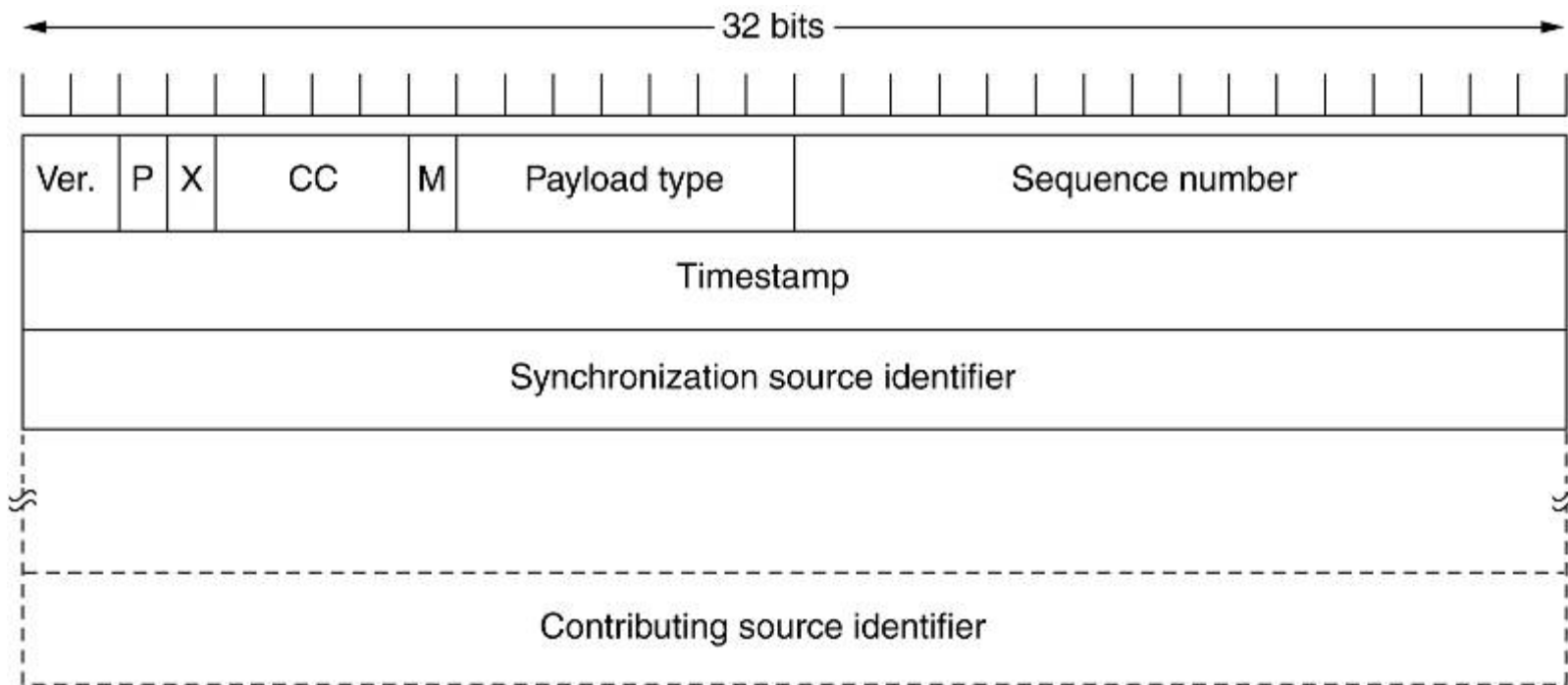
(a) Pozitia RTP in stiva de protocoale.

(b) Format pachet.

Formatul datelor din "RTP payload" este specific aplicatiilor

RTP permite definirea unor **profile** (ex. single audio stream) si, pentru fiecare profil, a mai multor **formate**

Antet RTP



Timestamp – amprenta de timp relativa la inceputul fluxului

- reduce efectul intarzierilor variabile
- permite sincronizarea intre stream-uri

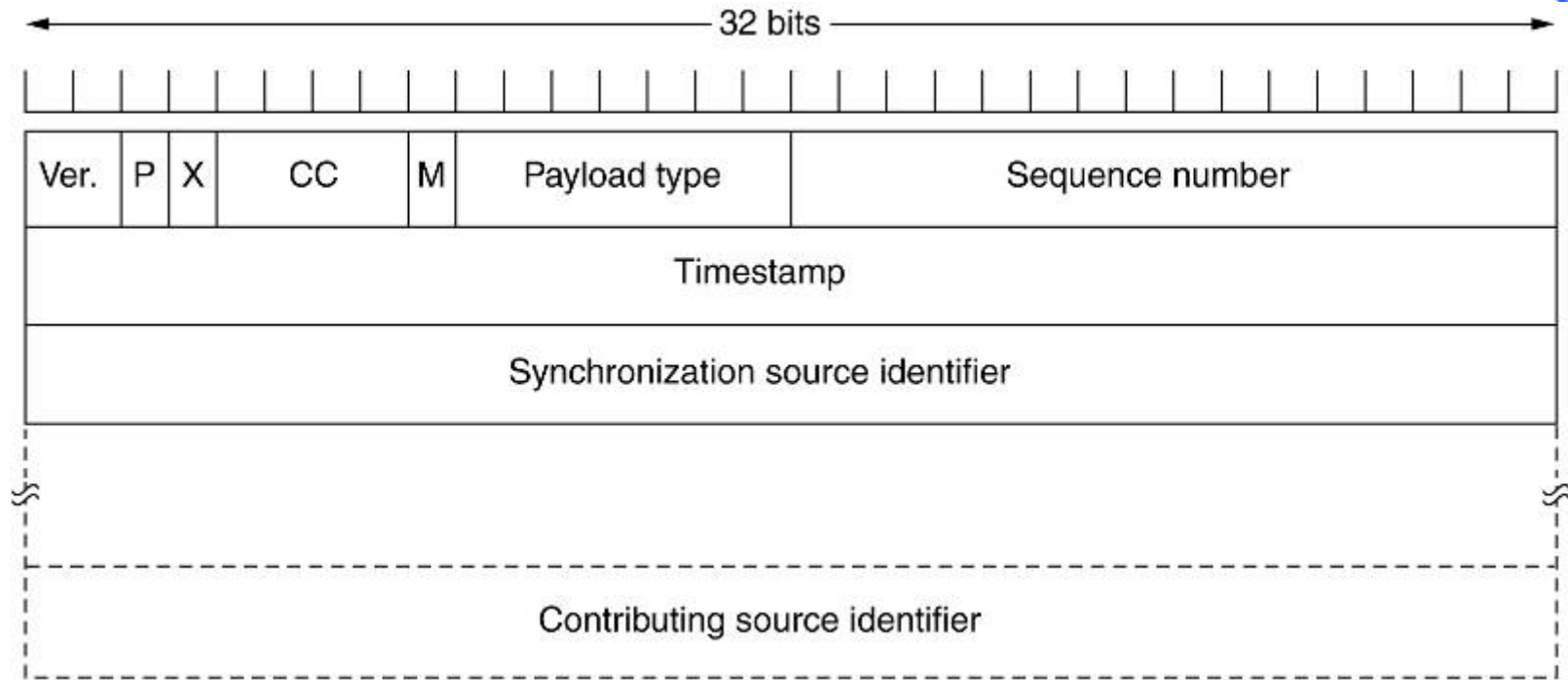
Synchro – identifica sursa de sincronizare (ex. microfon, camera video)

- receptorul grupeaza pachetele dupa sursa, pentru redare

Contrib – lista surselor care au contribuit la continutul curent; folosit pentru mixere;

- ex. la audio-conf, lista vorbitorilor ale caror discursuri sunt in pachet curent

Antet RTP



P – padding - pachet extins la multiplu de 4 octeti

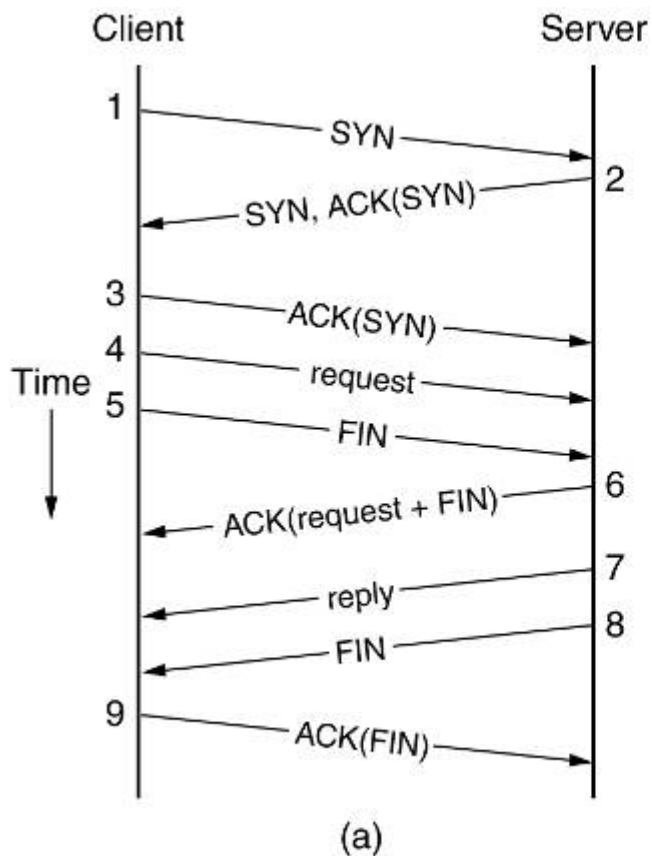
X – extension - antet extins (primul cuvant da lungimea)

CC – no. antete surse contributoare

M – mark (specific aplicatiei. Ex. start video frame)

Payload type – e.g. MP3

TCP Tranzactional



(a) RPC – Remote Procedure Call - folosind TPC normal.

(b) RPC folosind T/T

Permite transfer de date odata cu stabilirea conexiunii

Studiu individual

A. S. Tanenbaum Rețele de calculatoare, ed 4-a, BYBLOS 2003

6.1 SERVICIILE OFERITE DE NIVELUL TRANSPORT

6.2.1 Adresarea

6.2.2 Stabilirea conexiunii

6.2.3 Eliberarea conexiunii

6.2.4 Controlul fluxului și memorarea temporară (buffering)

6.4.1 Introducere în UDP

6.4.3 Protocolul de transport în timp real

6.5.3 Protocolul TCP

6.5.4 Antetul segmentului TCP

6.5.5 Stabilirea conexiunii TCP

6.5.6 Eliberarea conexiunii TCP

6.5.7 Modelarea administrării conexiunii TCP

6.5.8 Politica TCP de transmisie a datelor

6.5.9 Controlul congestiei în TCP

6.5.10 Administrarea contorului de timp în TCP

6.5.12 TCP Tranzacțional

6.6.4 Prelucrarea rapidă a TPDU-urilor



Studiu individual

A. S. Tanenbaum Computer networks, 5-th ed. PEARSON 2011

6.1 THE TRANSPORT SERVICE

6.2.1 Addressing

6.2.2 Connection Establishment

6.2.3 Connection Release 6.2.4 Error Control and Flow Control

6.3.2 Regulating the Sending Rate

6.4.1 Introduction to UDP

6.4.3 Real-Time Transport Protocols

6.5.3 The TCP Protocol

6.5.4 The TCP Segment Header

6.5.5 TCP Connection Establishment

6.5.6 TCP Connection Release

6.5.7 TCP Connection Management Modeling

6.5.8 TCP Sliding Window

6.5.9 TCP Timer Management

6.5.10 TCP Congestion Control

6.6.4 Fast Segment Processing