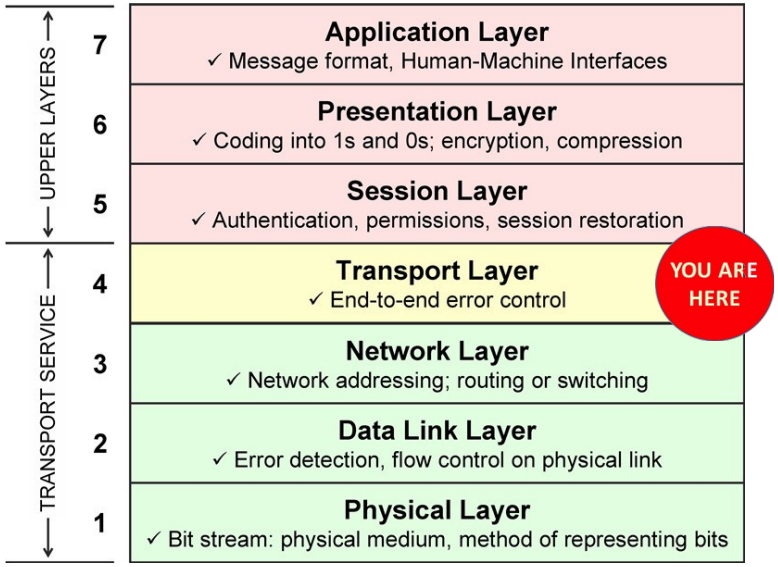
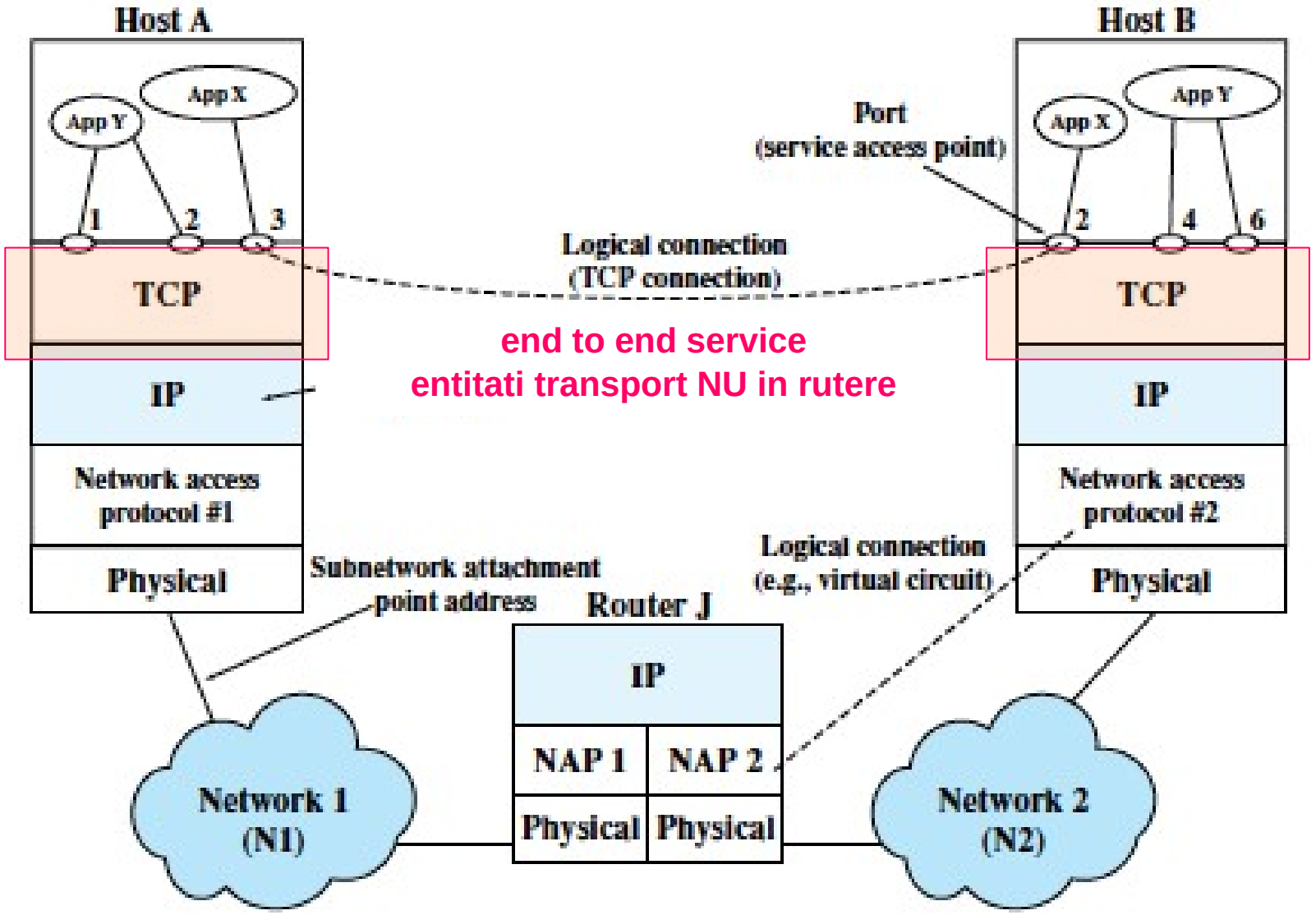
Nivelul transport

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 recceptorului
  - controlul congestiei, ceasurilor, time-out



Nivelul transport in ierarhia TCP/IP

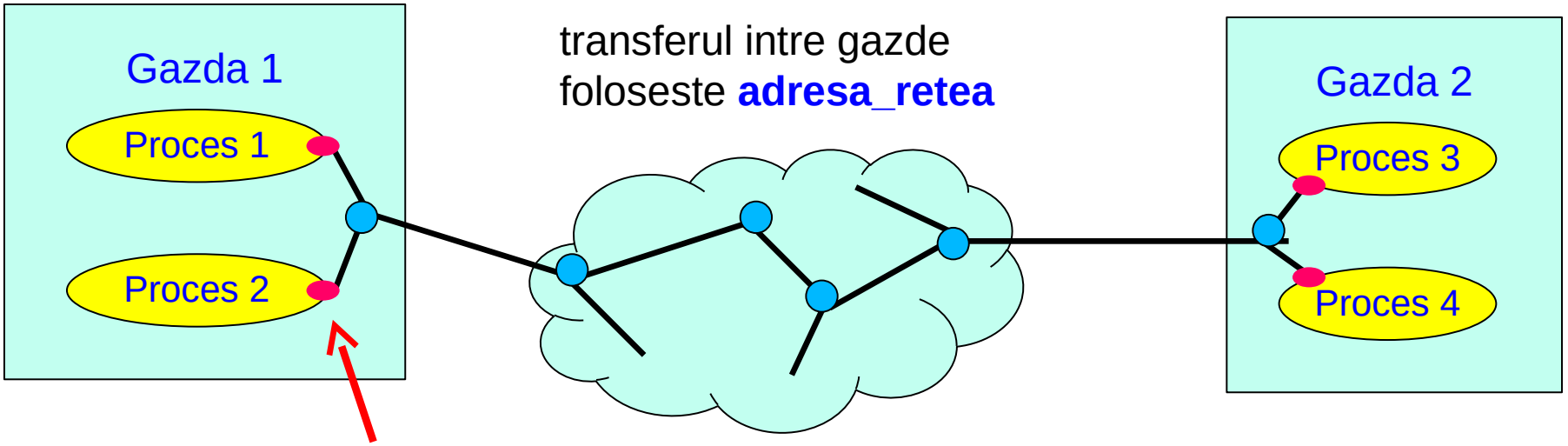


Comunicarea intre aplicatii

Reteaua asigura transmiterea pachetelor (datagrame) intre calculatoare gazda

Transportul asigura comunicarea intre aplicatii

- legatura proceselor de aplicatie cu punctele de acces la retea
- identificarea unica a unui punct de acces prin <**adresa_retea**, **port**>



Servicii ale nivelului Transport

- **Servicii furnizate**
  - transfer de date intre procese de aplicatie, folosind retele de diverse tipuri
  - interfaţa uniforma cu utilizatorii (Socket API)

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

Protocoale de Transport

Doua protocoale majore

- UDP
- TCP

Aspecte discutate

- Formatul datelor
- Adresare
- Functionare

UDP - User Datagram Protocol

- UDP livreaza datagrame utilizator - *user datagrams*
  - *Nu se realizeaza o conexiune*
  - Livrare “Best effort” – datagramele pot fi pierdute, primite in alta ordine etc.
  - Sume de control pentru integritate

TCP - Transmission Control Protocol

Orientat pe conexiune

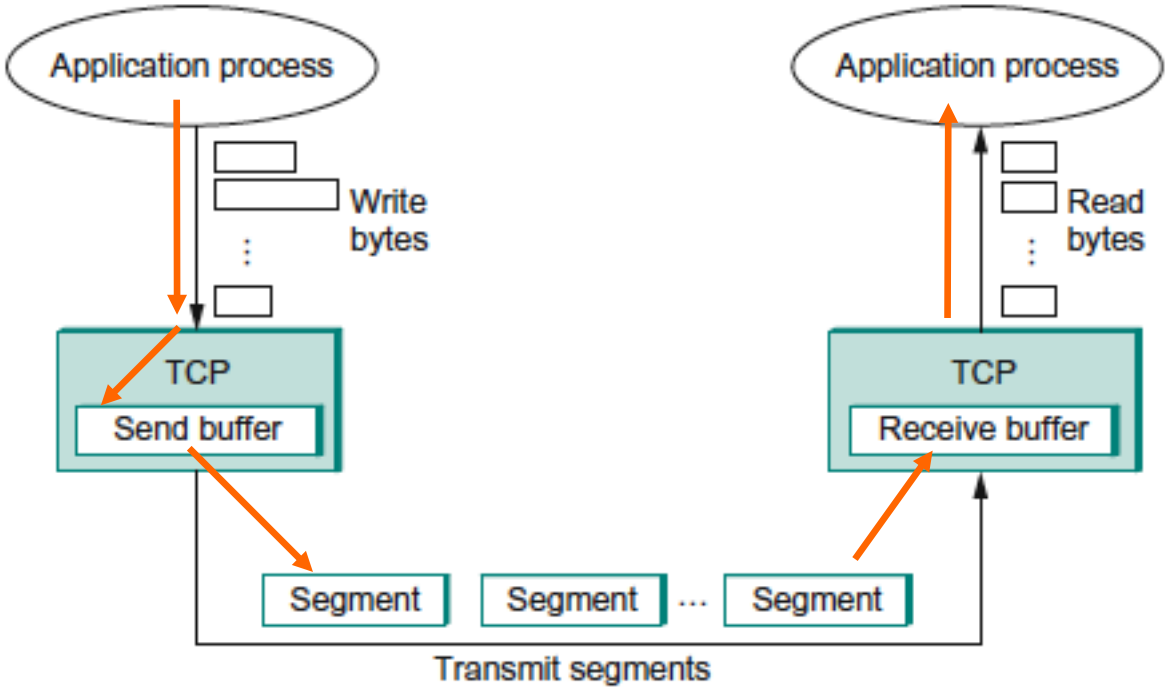
Livrare sigura pe retea nesigura

- ***Cel mai folosit protocol de transport***
  - Web
  - Email
  - SSH
  - Chat
  - Video streaming
  - Peer-to-peer

Cateva porturi standard 0 - 65535

Port	Protocol	Use
20, 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 octeti



- 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 si receptie siruri de octeti
- 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

TCP

UDP

Socket interface

- Serviciile nivelului transport sunt accesibile ca API – Application Programming Interface
- Este un standard de facto pentru comunicarea intre procese pe internet
- Oferită ca bibliotecă de functii folosite pentru

dezvoltarea aplicatiilor Internet, pentru diverse

sisteme de operare

Socket API

  - Originară din Berkeley Software Distribution (BSD) UNIX, apoi adoptata ca standard POSIX.
  - Disponibilă si pe Windows, Solaris, etc.
  - socket = punctul în care procesul de aplicație se atașază la rețea. Este identificat prin descriptor (număr), in acelasi mod ca un fisier

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

Setare adresa

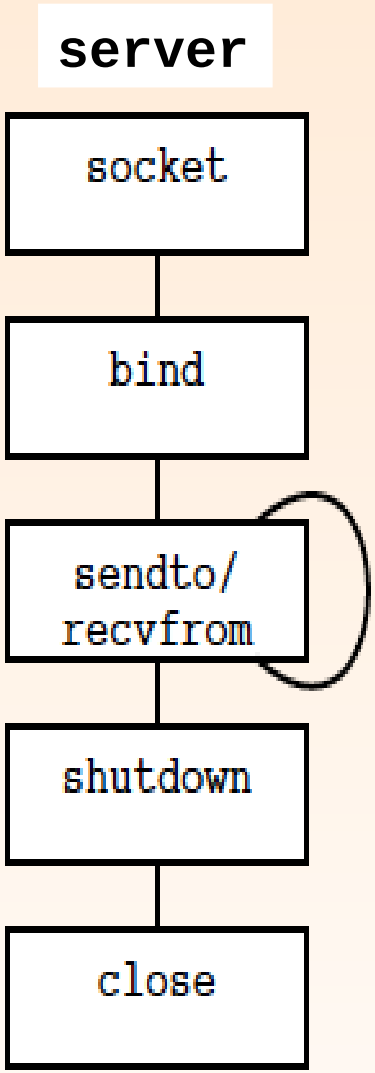
Format TCP/IP:

```
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 - server



Creeaza socket si aloca-i resurse de sistem:

*int s = **socket** (PF_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

Primeste 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);

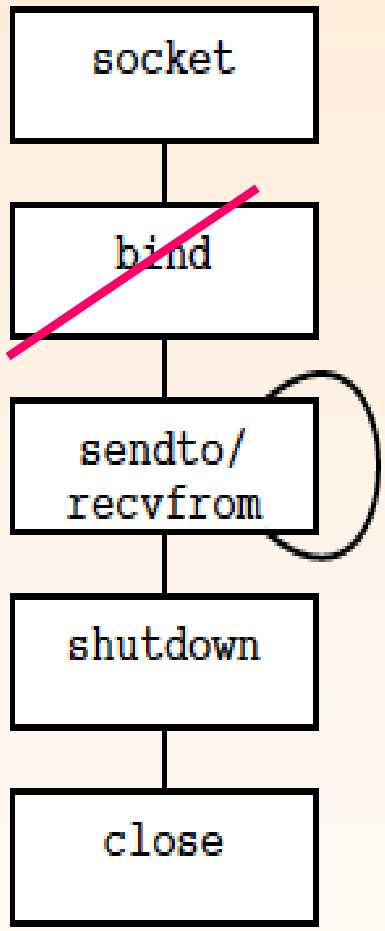
Oprește trimitere sau/și recepție de date

shutdown (s, SHUT_RD / SHUT_RDWR / SHUT_WR)

Inchide socket - termina utilizarea socket si elibereaza resurse alocate

close (s)

Serviciu fara conexiune - client



Creeaza socket și aloca-i resurse de sistem:

*int s = **socket** (PF_INET, SOCK_DGRAM, PPROTO_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));

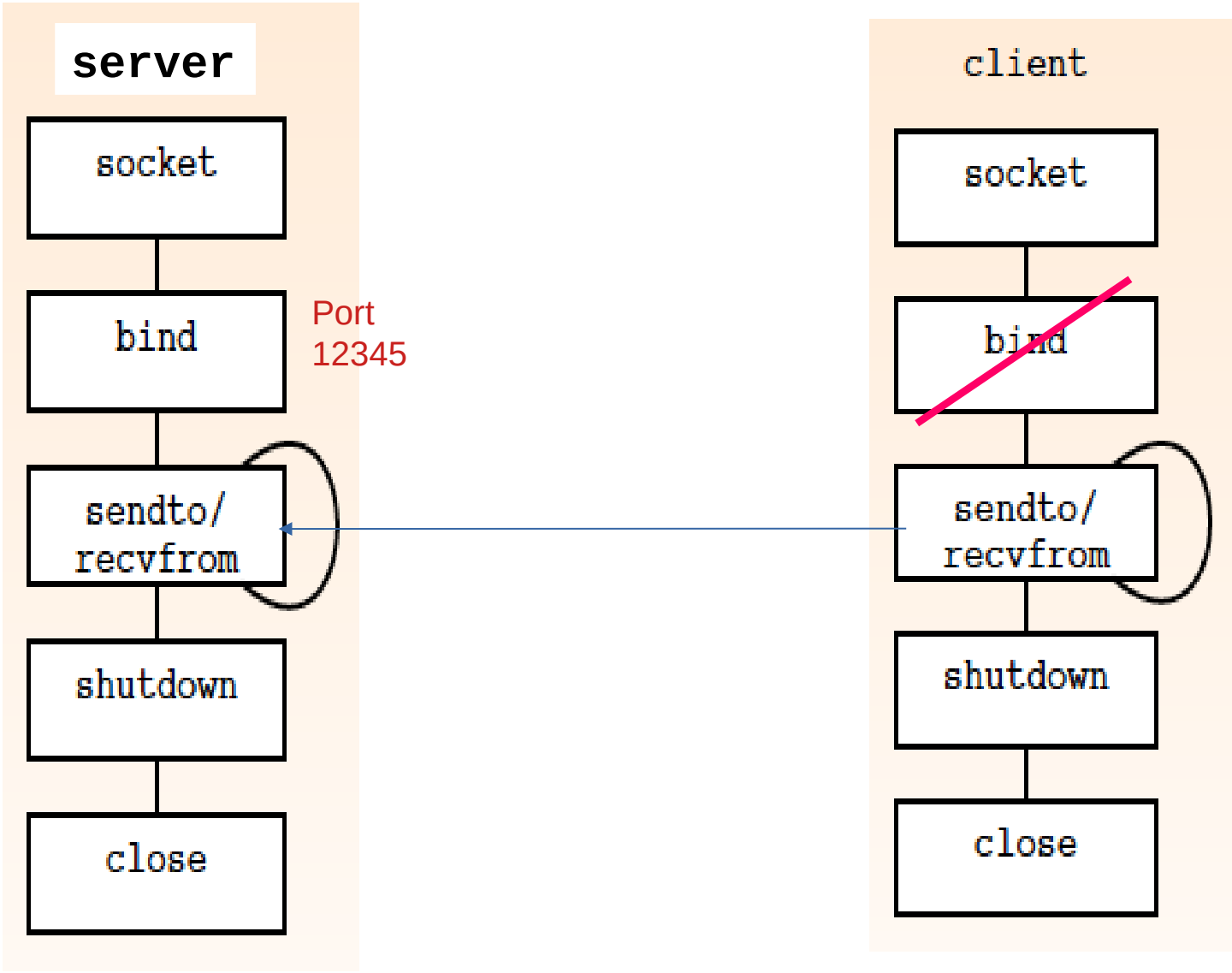
Opreste trimitere sau/și receptie de date

shutdown (s, SHUT_RD / SHUT_RDWR / SHUT_WR)

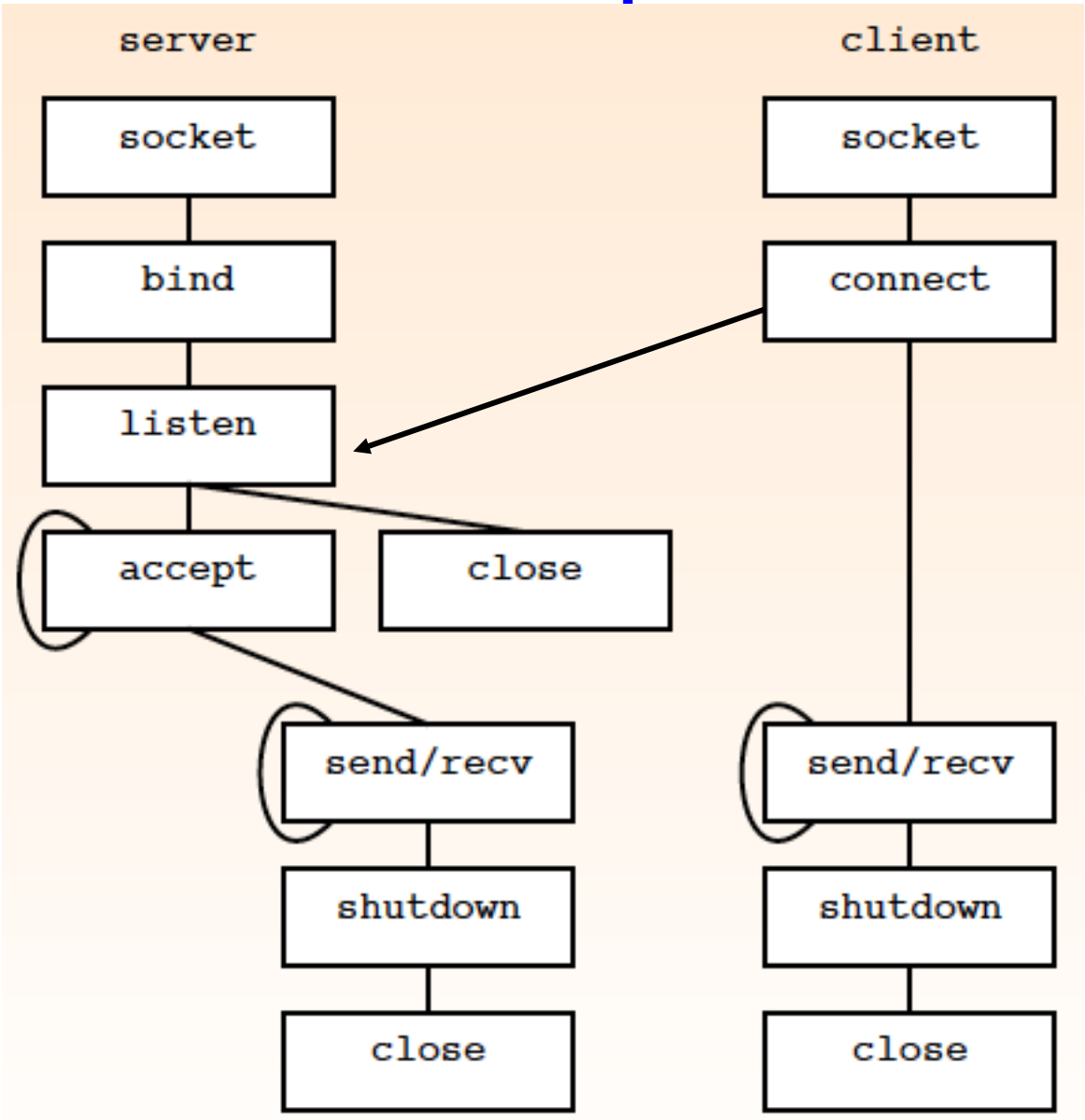
Termina utilizarea socket si elibereaza resursele alocate

close (s)

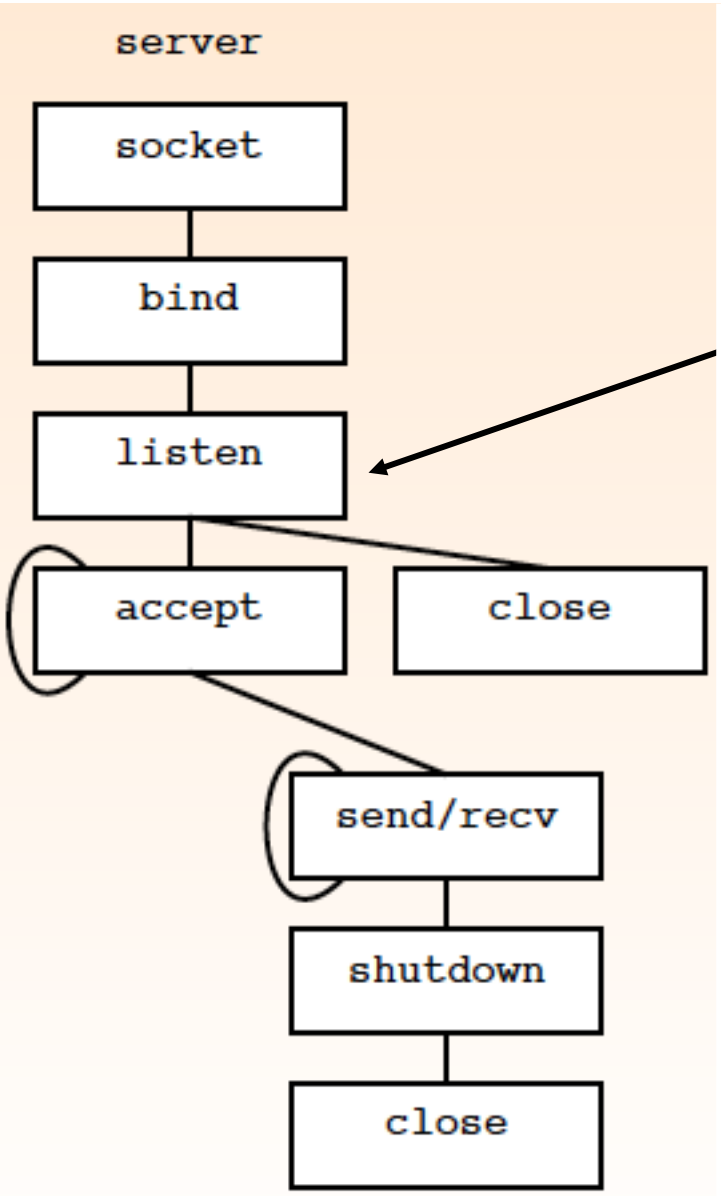
UDP


server
socket
bind
Port
12345
sendto/
recvfrom
shutdown
close
client
socket
bind
sendto/
recvfrom
shutdown
close

Serviciu orientat pe conexiune



Serviciu orientat pe conexiune - server



1. Creeaza socket:

*int ls = **socket** (PF_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 in asteptare):

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 / primeste pe s etc.

Serviciu orientat pe conexiune - client

1.Creaza socket:

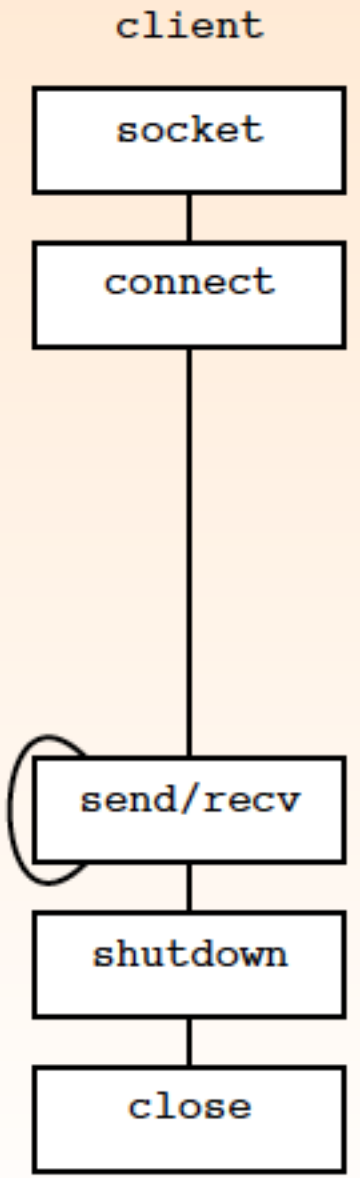
int s = ***socket*** *(PF_INET, SOCK_STREAM, IPPROTO_TCP);*

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 / primeste 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/primeste date out-of-band

  MSG_PEEK – livreaza date primite, dar trateaza 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

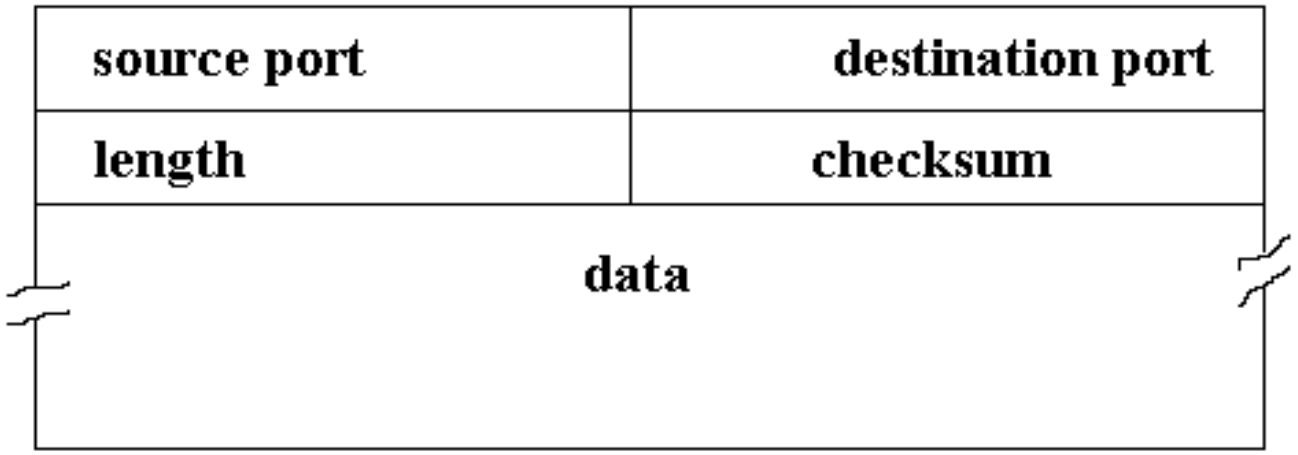
    •opreste transmiterea – ignora datele inca netrimise

    •ambele
  - close (s)

    int close (int socket);

    •inchide socket (elibereaza structurile de date din kernel)

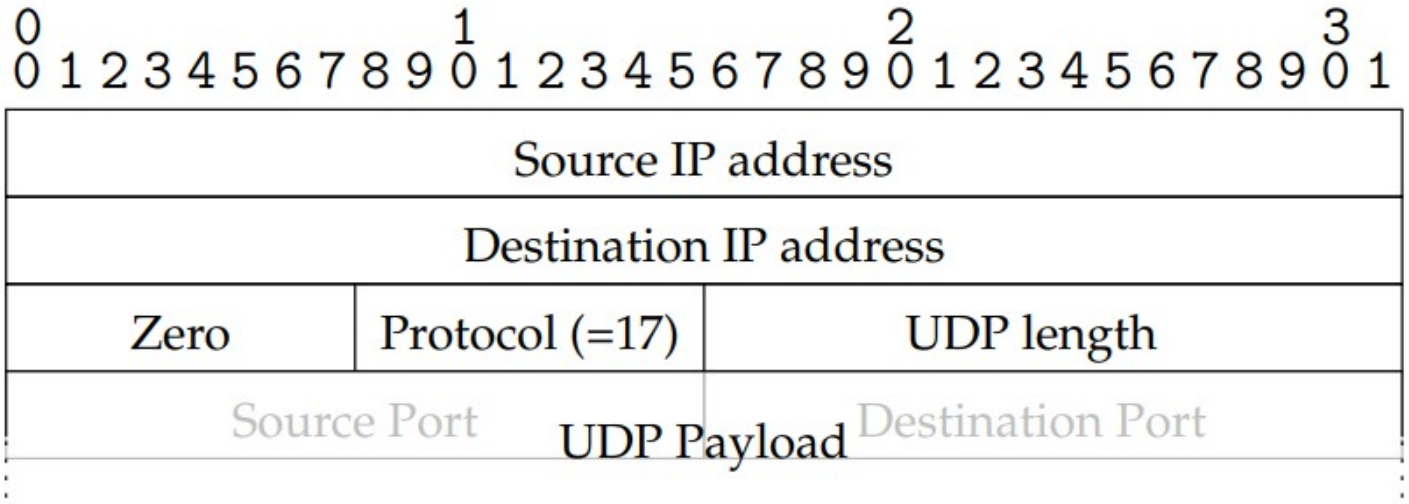
Antet UDP



NU ASIGURA: control flux, control erori, retransmisie

- UDP identifica adresa Internet si numar port pentru sursa si destinatie
- *Destination port* si *source port* pot fi diferite.

Calcul checksum (pseudo-antet IP)



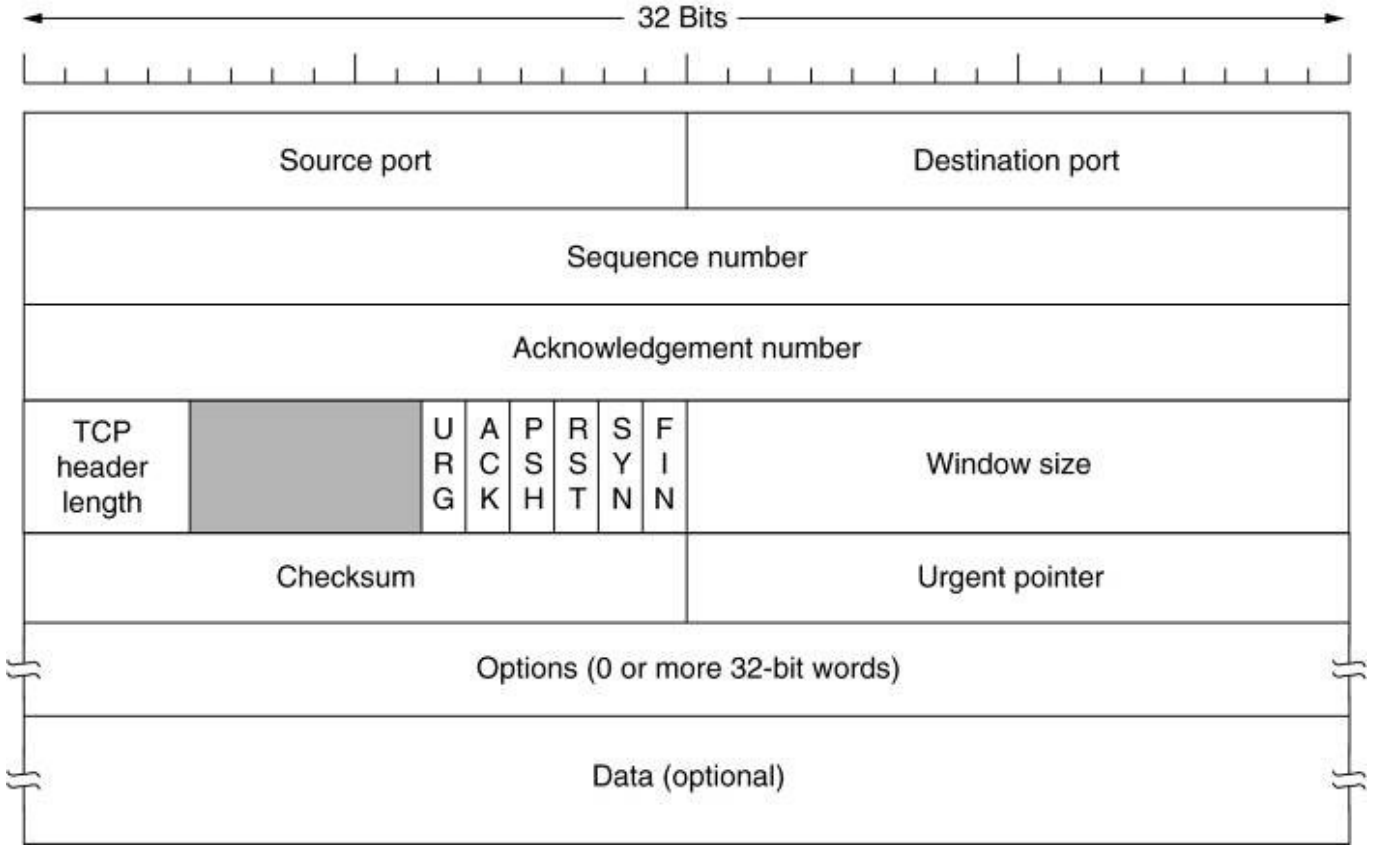
Checksum:

- Include antet, data si un pseudo-antet IP care contine:
  - Adresele IP pentru sursa si destinatie
  - Numar protocol UDP (17) si lungimea segment
- Algoritmul aduna toate cuvintele in complement fata de 1 de 16 biti si retine complementul fata de 1 al sumei
- Receptorul face calculul pe intregul segment, inclusiv checksum, iar rezultatul trebuie sa fie 0

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)

Antet segment TCP



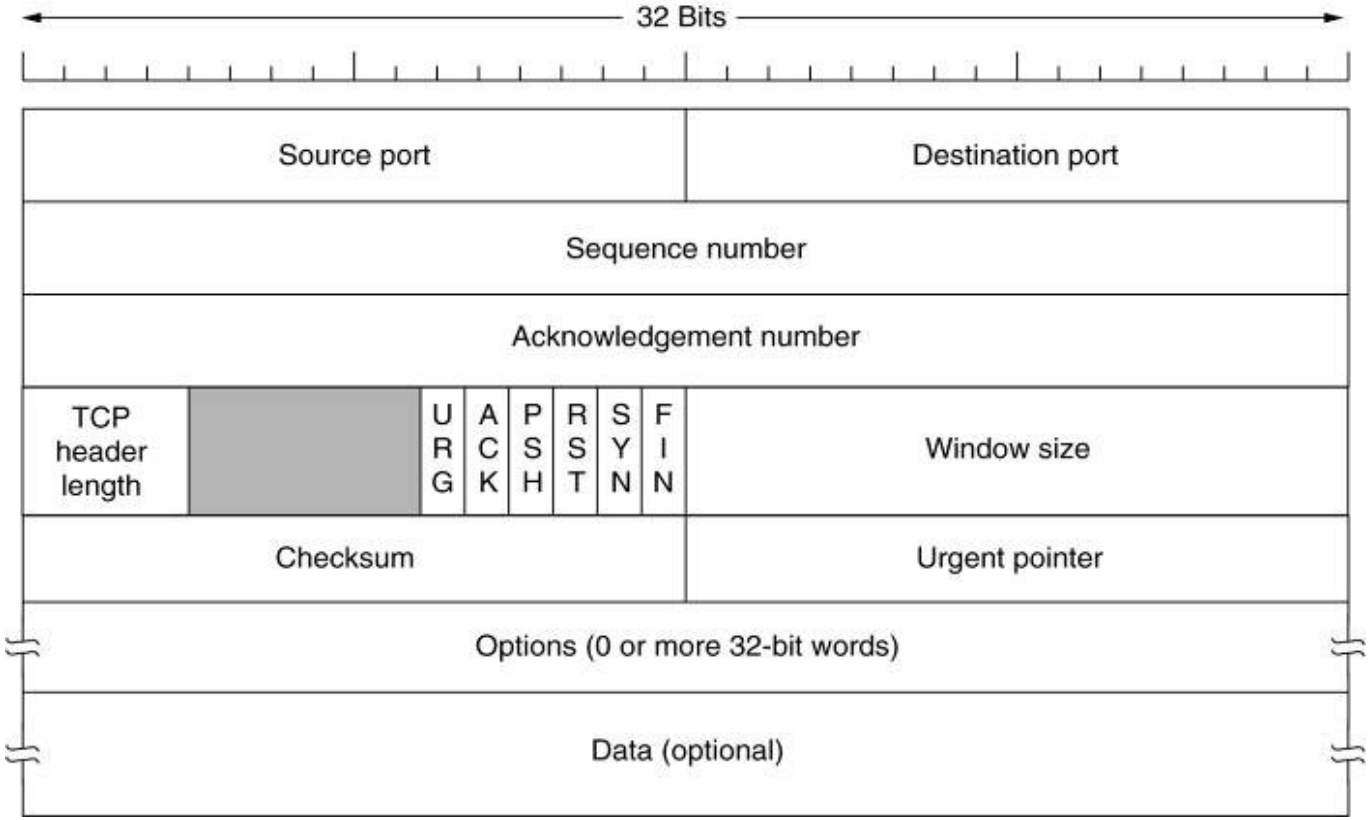
Sequence number – numarul primului octet din segment

Acknowledgement number – numarul urmatorului octet asteptat

Window size - numărul de octeţi care pot fi trimişi, începând cu ultimul octet

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

Antet segment TCP



URG Urgent pointer valid

ACK Acknowledge Number valid

PSH - push information to user

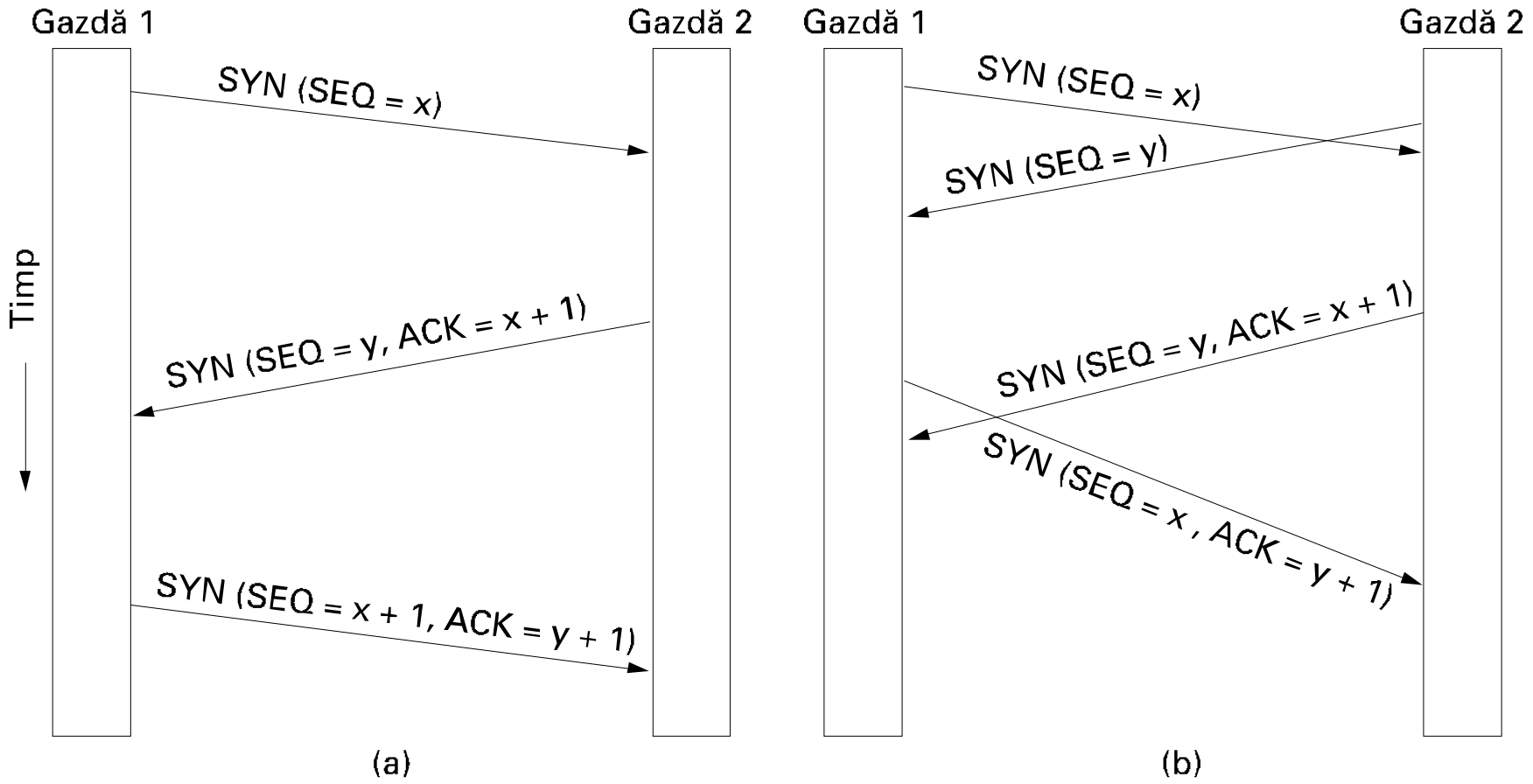
RST - close a connection due to an error

SYN - open connection

FIN - close a connection

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

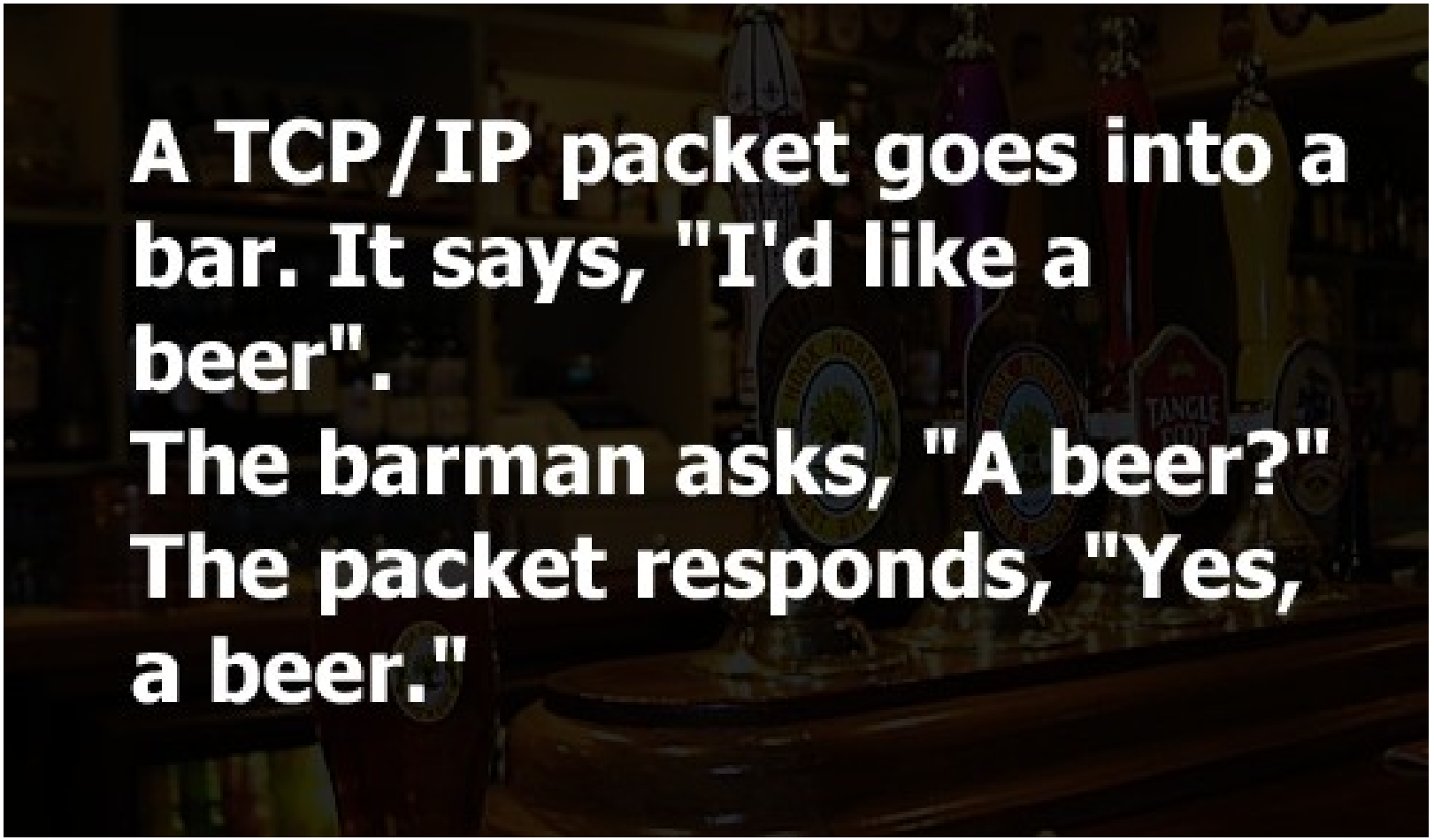
Stabilirea conexiunii - Three way handshake



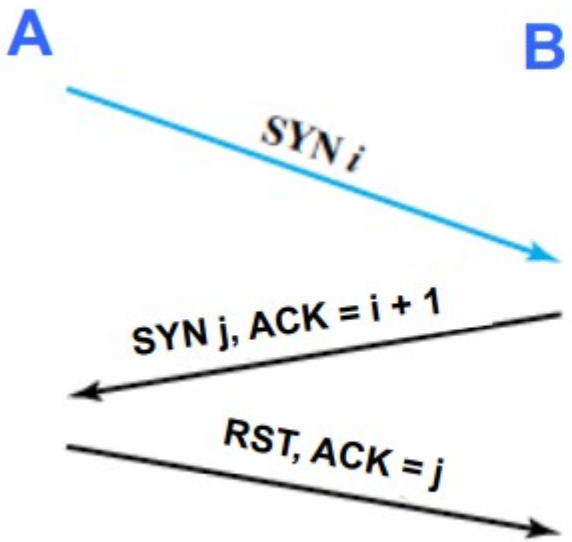
(a) Cazul normal.

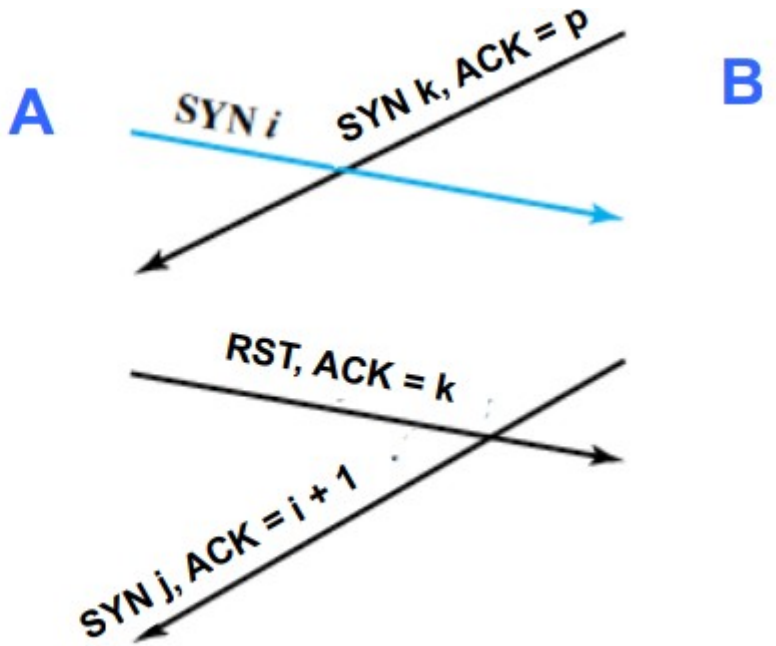
(b) Coliziune.

Stabilirea conexiunii - Three way handshaking



Rejectarea conexiunii





SYN intarziat

B accepta

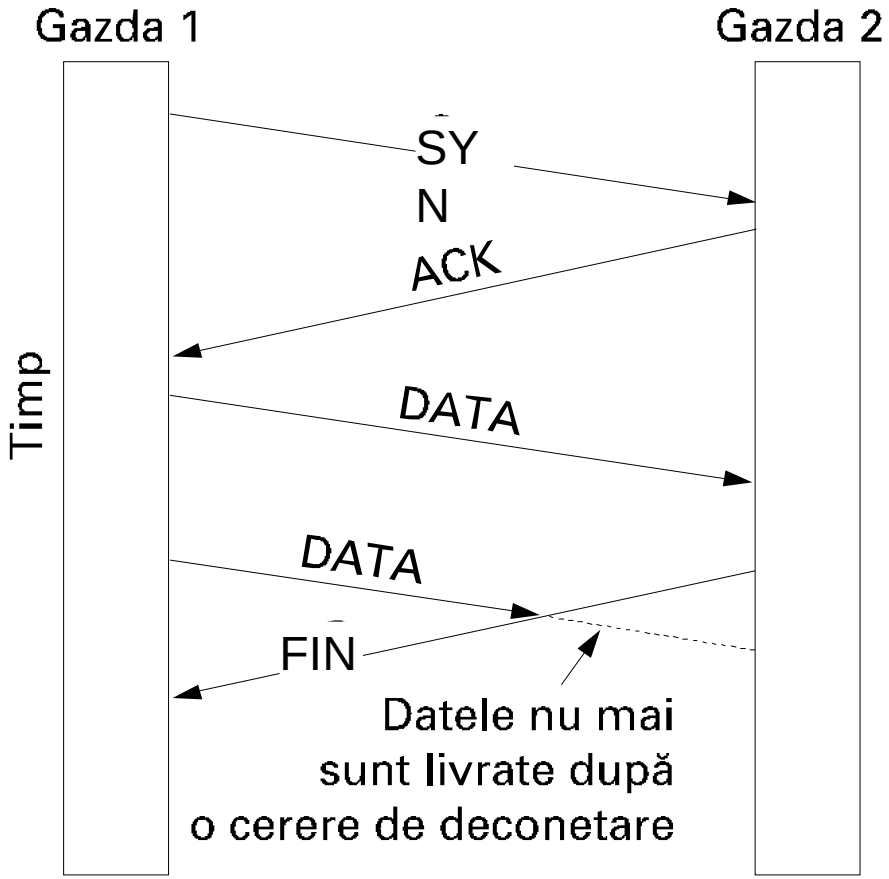
A rejecteaza

A initiaza conexiune (SYN i) + SYN k intarziat

A refuza (RST)

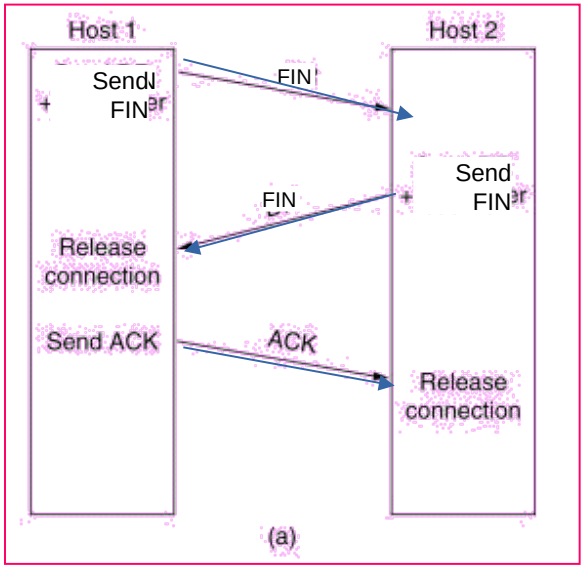
B accepta SYN i – raspunde cu SYN j

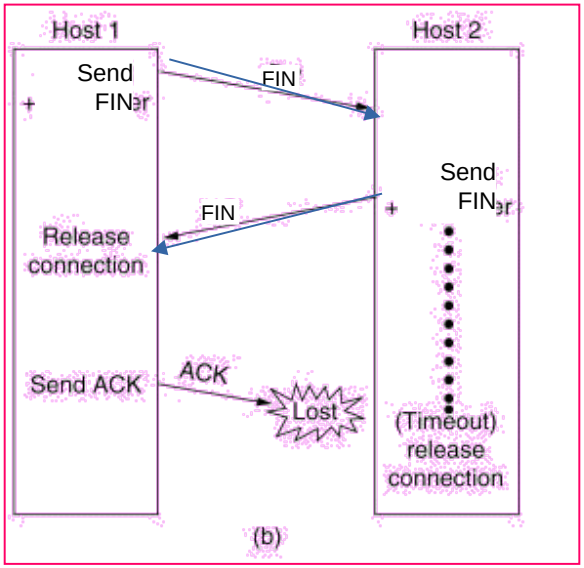
Deconectare abrupta cu pierdere de date



Patru scenarii de eliberarea conexiunii

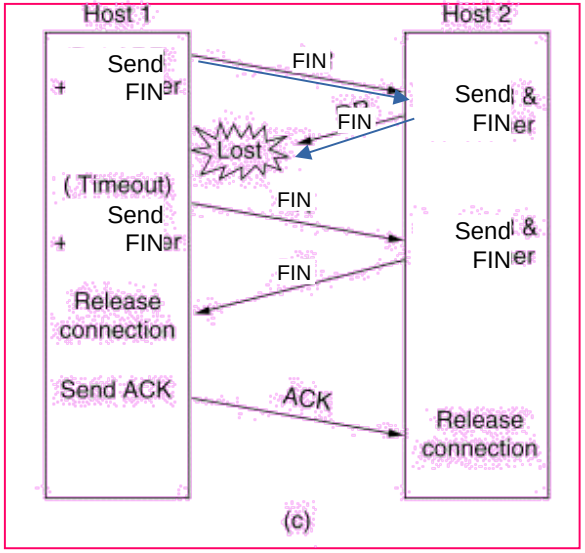
normal

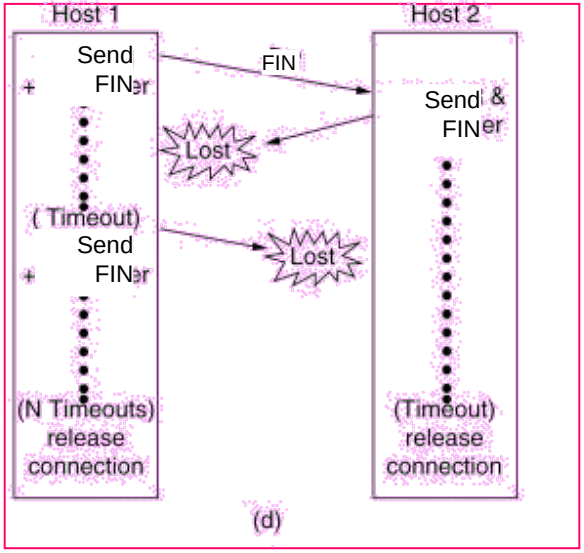




ACK final pierdut

raspuns pierdut





Raspuns si FIN pierdute

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

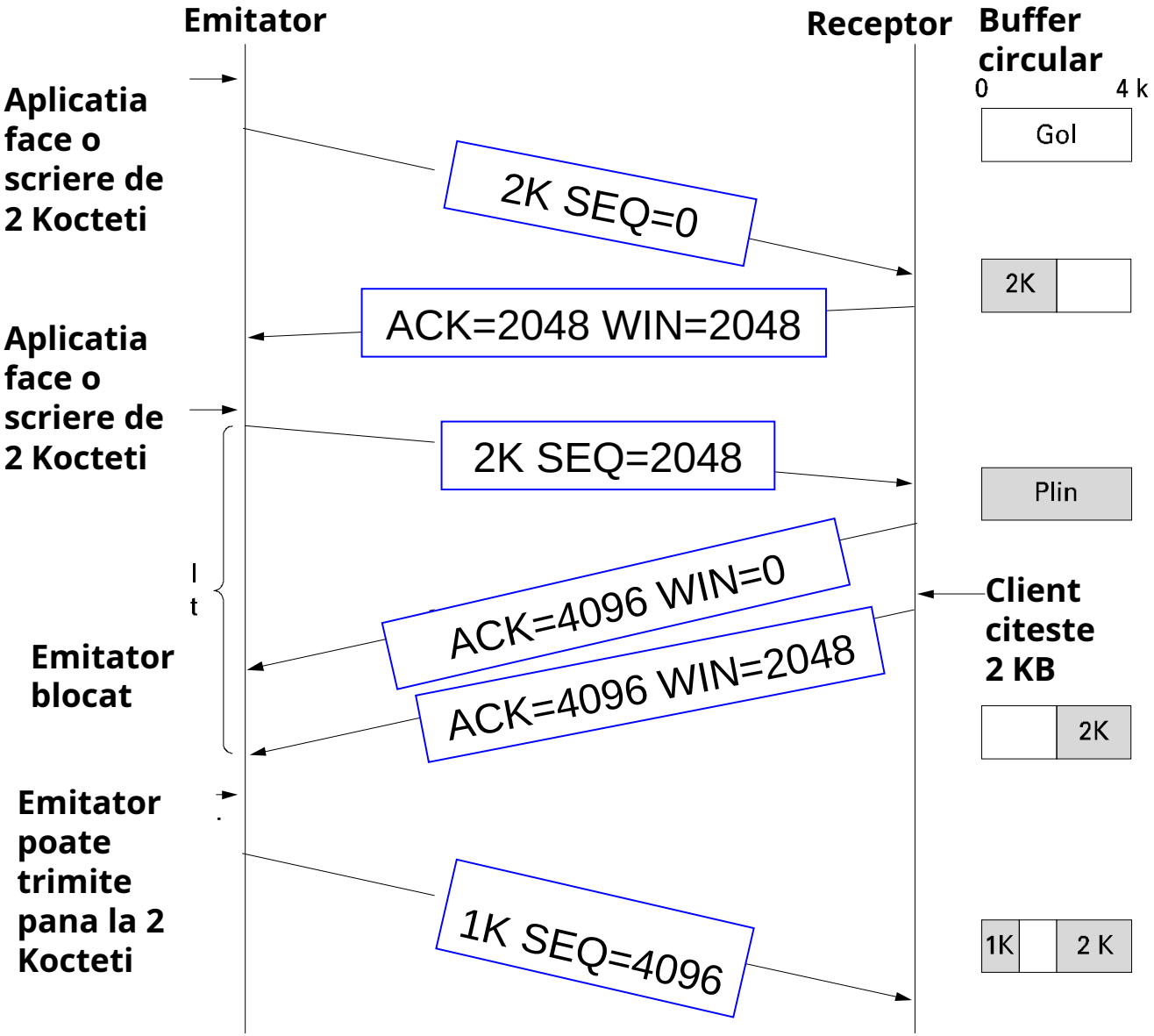
la **receptie**

- aduna cuvinte de 16 biti – rezultatul trebuie sa fie zero

2 - Acknowledgement number

Corectia se face prin retransmisie

Controlul fluxului de date



Receptorul specifica fereastra de receptie disponibila

Un emitator blocat poate trimite:

- date urgente
- un segment de 1 octet ptr a afla fereastra (daca anuntul precedent al receptorului s-a pierdut)

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 sa apara 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 µsec

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 Gbps * 40 ms = 40 milioane biti

Condiții - fereastra receptor ar trebui sa fie >= 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

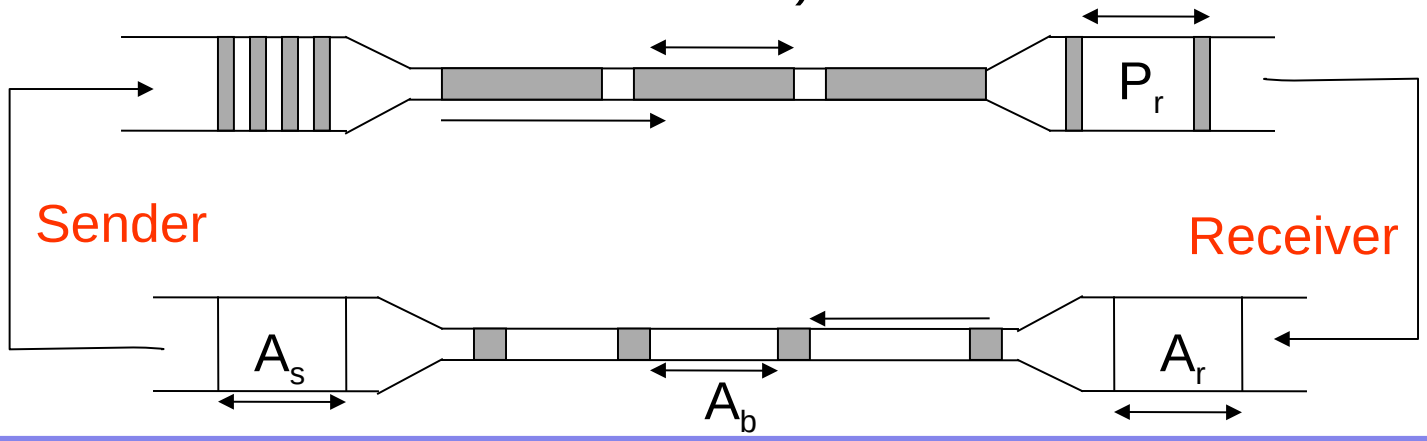
Controlul congestiei

Controlul congestiei

- Fluxul de date transmis pe o conexiune TCP limitat de minimul dintre:
  - dimensiunea fereastrei receptorului - *rwnd*
  - capacitatea retelei (**fereastra de congestie**) - *cwnd*
- Dimensiunea ferestrei glisante ar trebui sa fie *min(rwnd,cwnd)*
- Algoritm de stabilire fereastra de congestie
  - transmite un segment de dimensiune maximă pe conexiunea stabilita
  - 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)

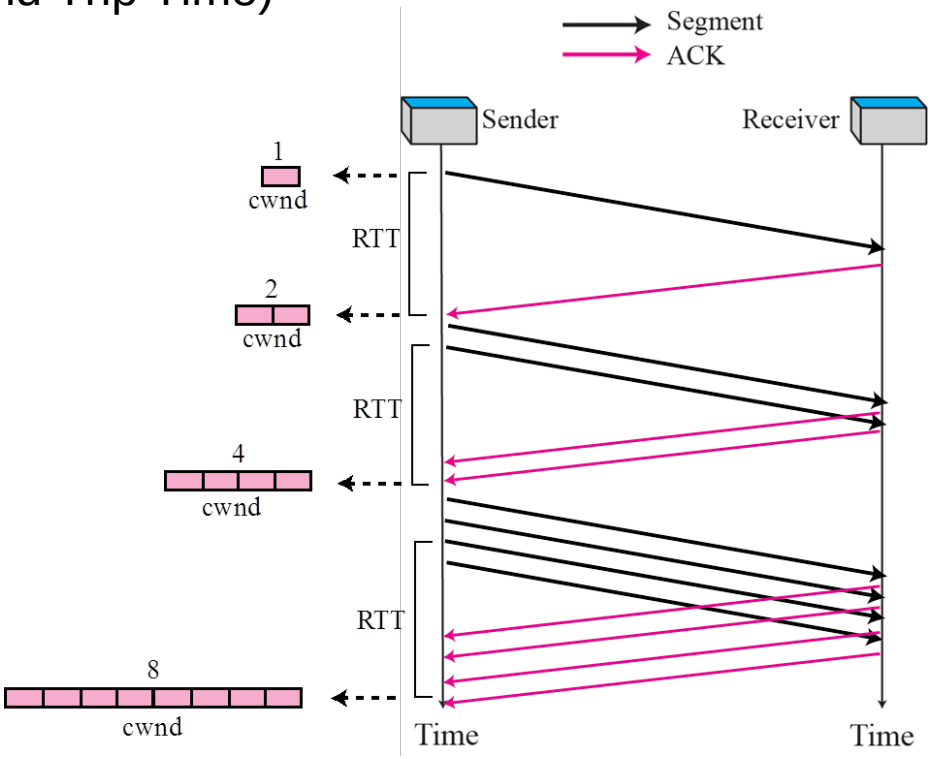
Controlul congestiei

- Algoritmul de control al congestiei
  - foloseste un **prag** (threshold)
  - la un timeout pragul este setat la jumatate din fereastra de congestie
  - se aplica procedeul de crestere (exponentiala) a fereastrei de congestie pana se atinge pragul
  - peste prag se aplica o crestere liniara (cu cate un segment de dimensiune maximă o dată)



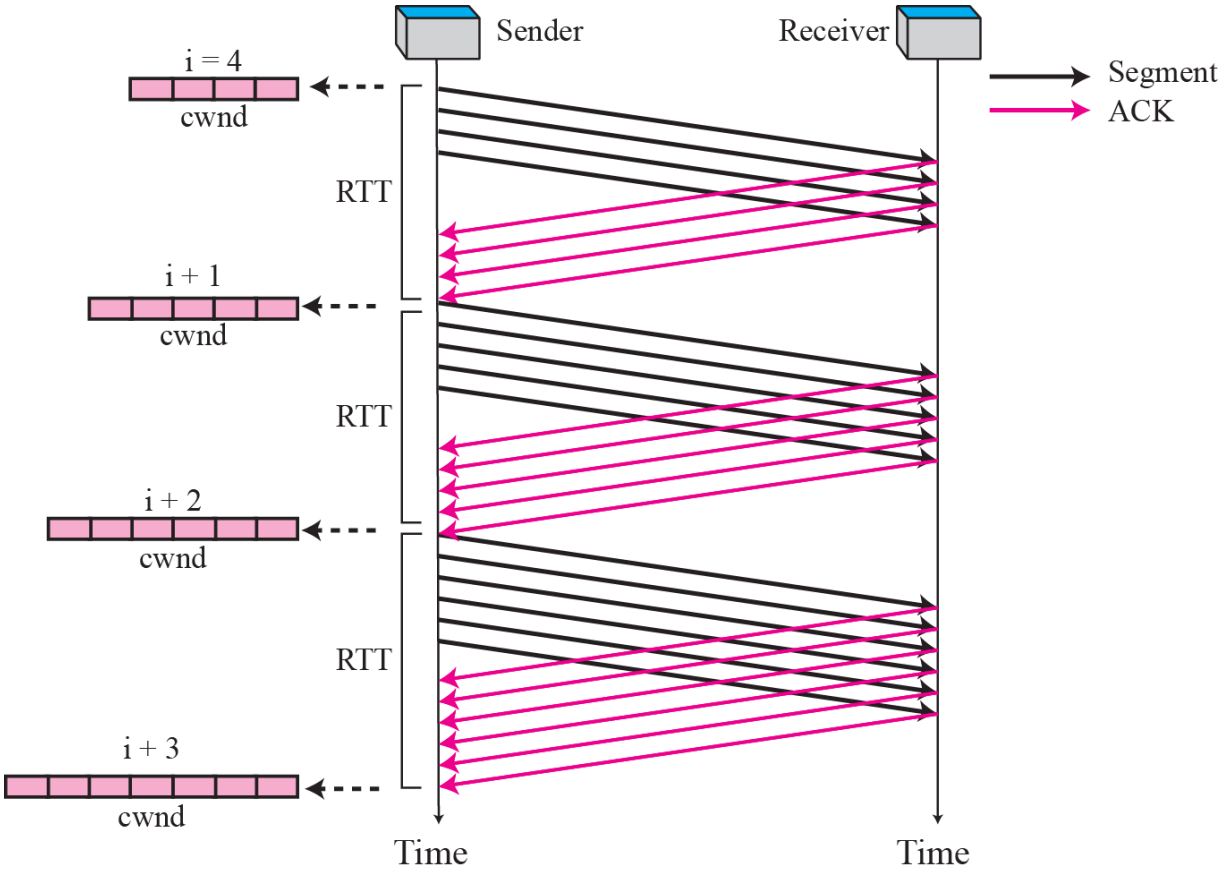
(1)TCP Slow Start

- Se doreste cresterea rapida a ratei de transmisie
- Se incepe cu trimiterea unui singur pachet si fereastra de congestie creste exponential pana la primul pachet pierdut
- Rata de transfer este foarte lenta la inceput dar creste exponential la fiecare RTT (Round Trip Time)



(2) Evitarea congestiei (crestere liniara)

- Cand dimensiunea ferestrei de congestie a ajuns la nivelul pragului pentru “slow start”, dimensiunea ferestrei de congestie va mai creste incremental pana se declanseaza un timeout.

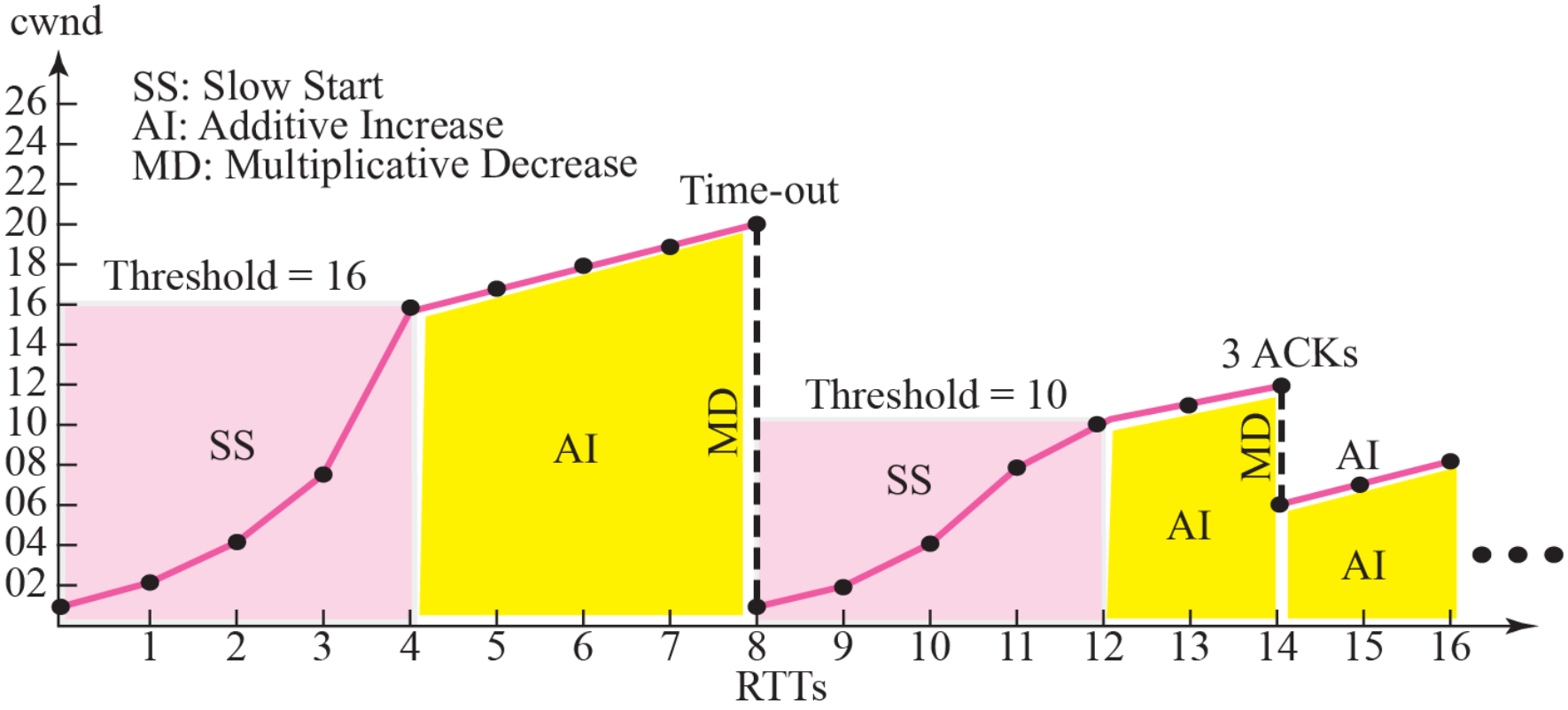


(3) Evitarea congestiei (scadere multiplicativa)

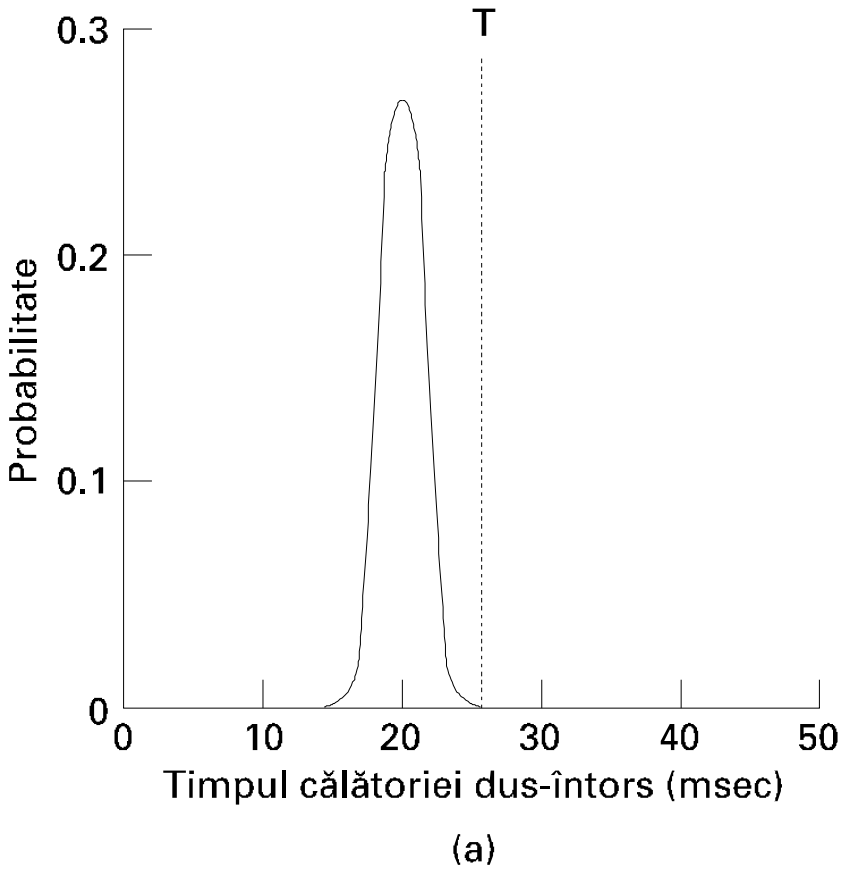
- In cazul in care se detecteaza o congestie pot fi luate una din urmatoarele doua actiuni:
  - Daca a avut loc un timeout se reia fereastra de congestie de la valoarea 1 si se reporneste etapa de Slow Start
  - Daca au fost primite 3 ACK-uri duplicate, este posibil sa nu fie atat de mare congestia, deci se reduce valoarea ferestrei de congestie la jumatate din valoarea curenta a ferestrei si se porneste etapa de crestere liniara.

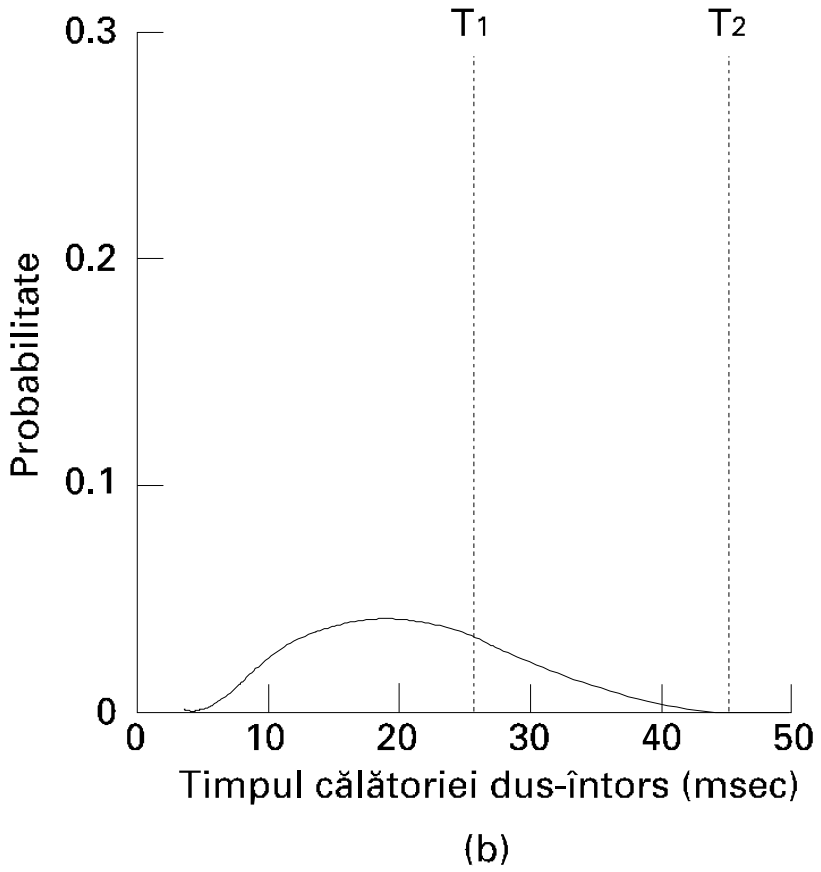
Comportament tipic TCP

(The TCP Sawtooth)



Gestiunea ceasurilor in TCP





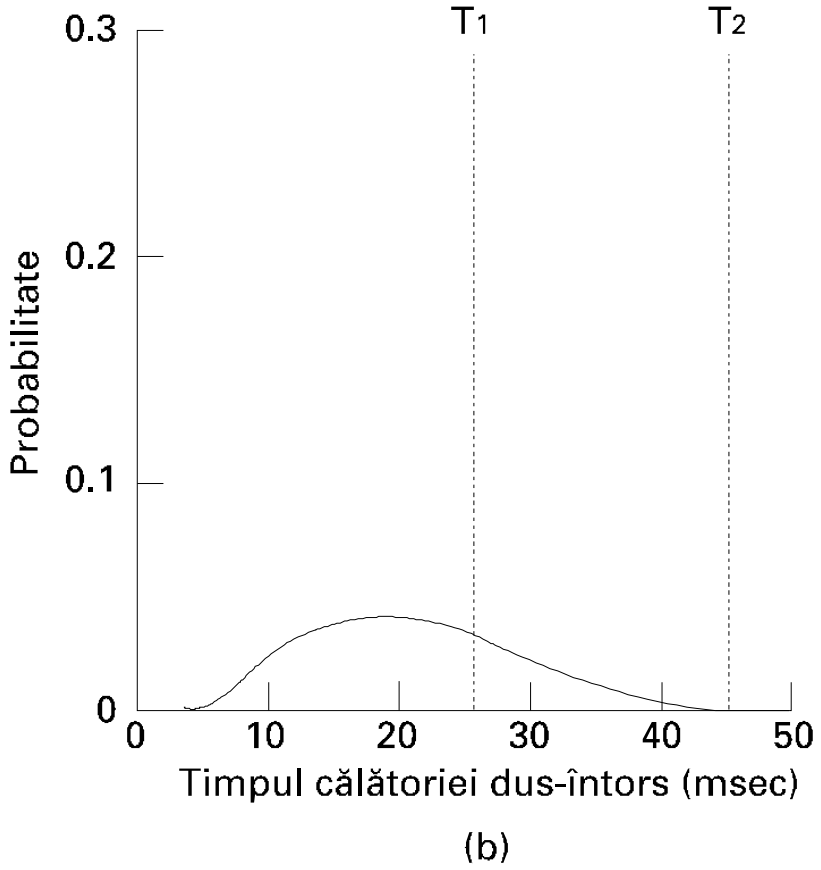
(a) Densitatea de probabilitate a timpilor de sosire ACK in nivelul legatura de date.

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

Gestiunea ceasurilor in TCP (2)

Setare proasta – performante slabe:

- Prea lung (T2) – transmitatorul asteapta mult ptr retransmisie
- Prea scurt (T1) – trafic inutil generat de transmitator



Stabilire time-out

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

  M este timpul masurat pana la primirea ack

  RTT = α*RTT + (1- α)*M cu α = 7/8

  RTO = β*RTT cu β = 2

- Alegere dupa *deviatia standard (DS);*

  D aproximeaza DS

  D = α*D + (1- α)*|RTT-M|

  RTO = RTT + 4*D

  Valoarea lui **α** este dependenta de implementare , dar de obicei este ales 3/4