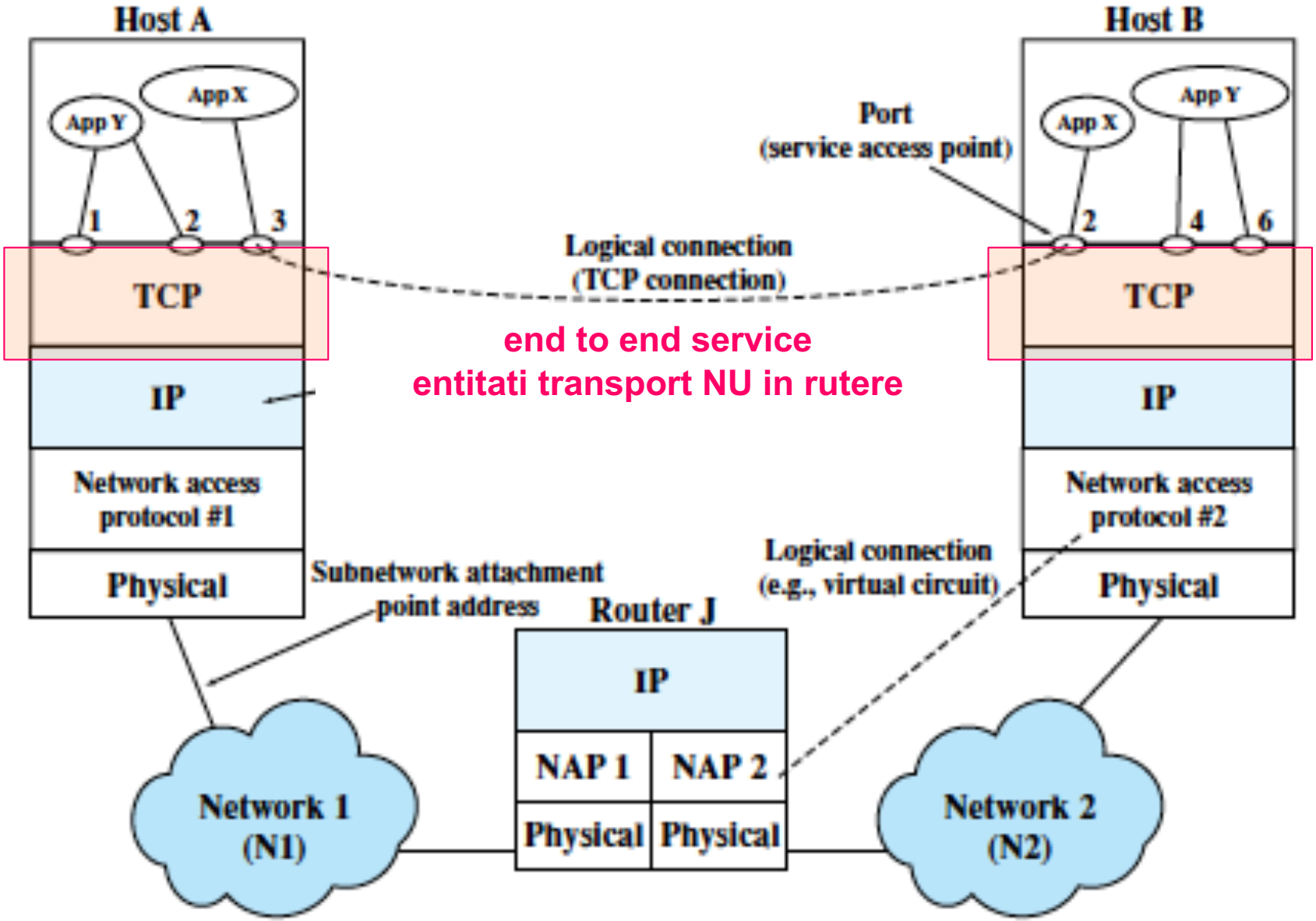
Nivelul transport

Nivelul transport in ierarhia TCP/IP

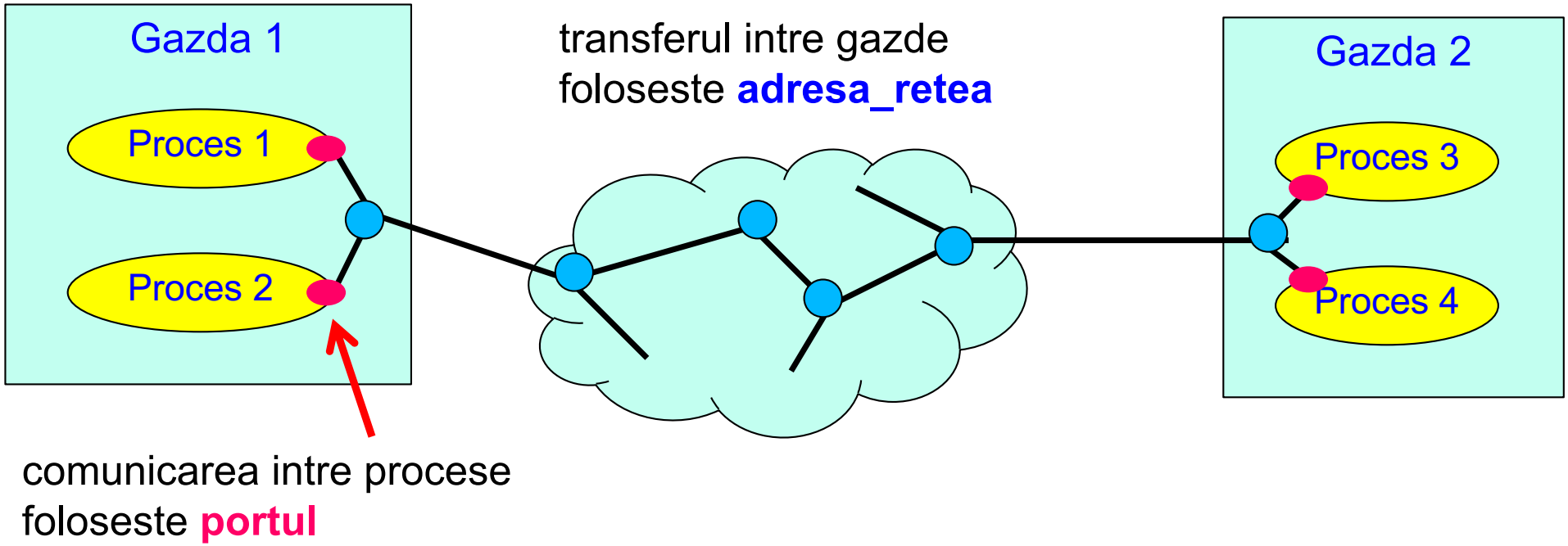


Comunicarea intre aplicatii

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

- **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 si 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.

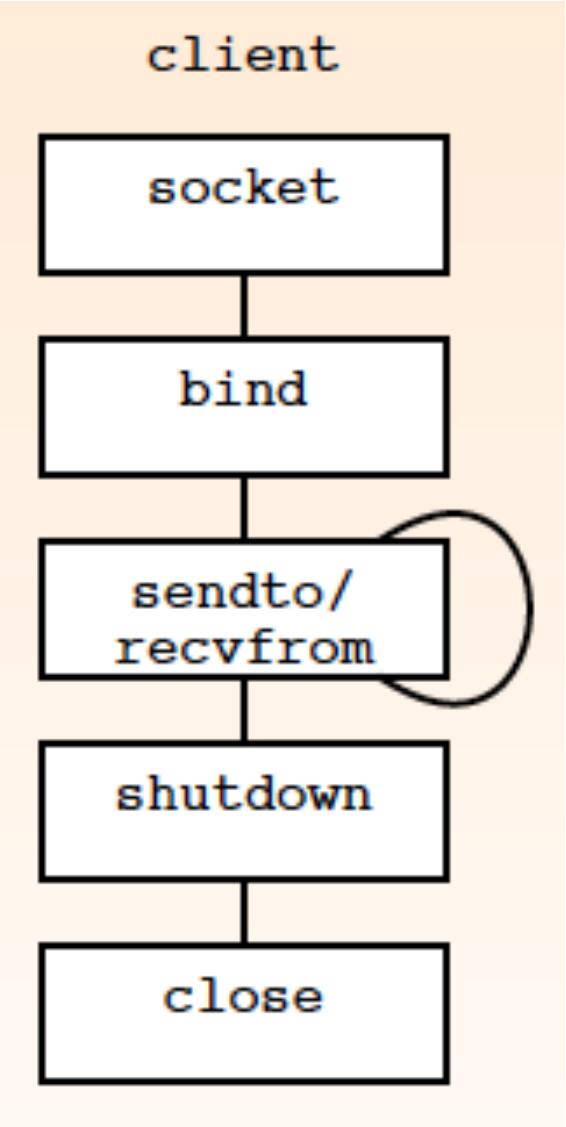
comm_type selecteaza tipul de comunicare

SOCK_DGRAM	- fara conexiune - datagrama
SOCK_STREAM	- orientat pe conexiune – flux de octeti

protocol specifica protocolul

IPPROTO_TCP	- TCP
IPPROTO_UDP	- UDP

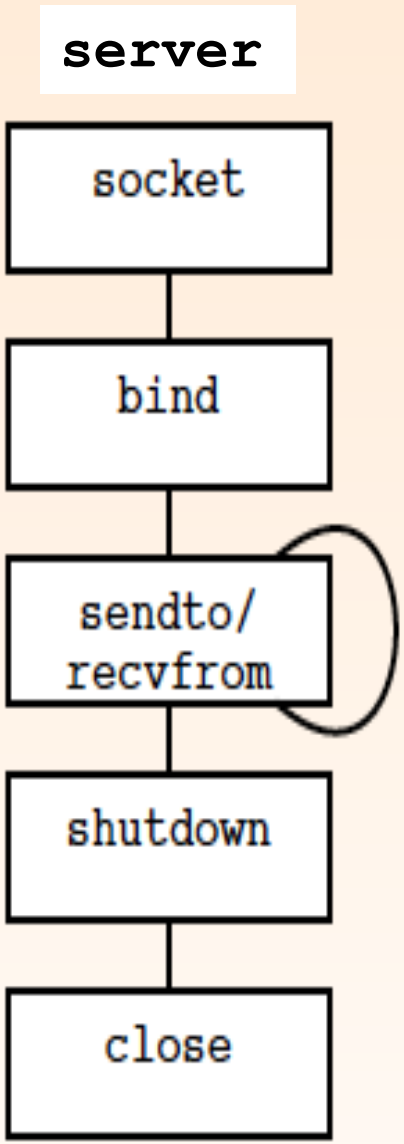
Serviciu fara conexiune



`server` este la fel!

bind - optional pentru initiator

Serviciu fara conexiune - server



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 !

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)

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
```

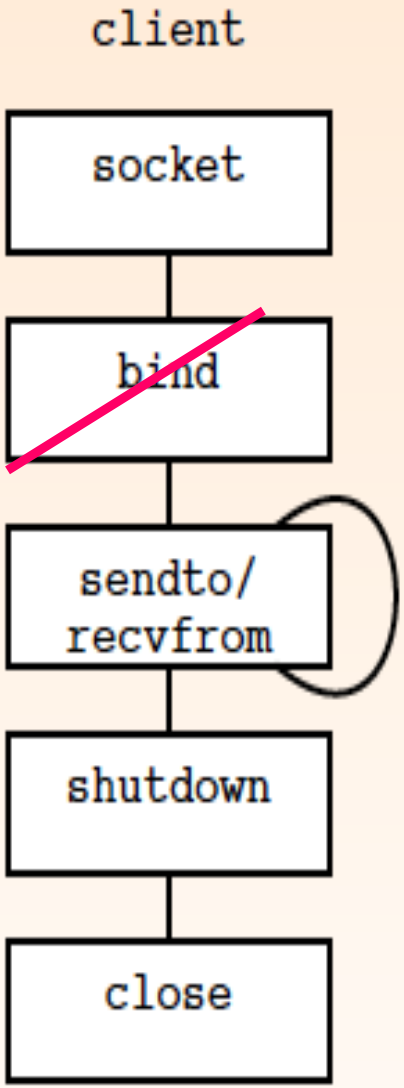
Exemplu client

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER PORT 12345 /* arbitrary, but client & server must agree */
#define BUF SIZE 4096 /* block transfer size */

int main(int argc, char **argv) {

    int c, s, bytes;
    char buf[BUF SIZE]; /* buffer for incoming file */
    struct hostent *h; /* info about server */
    struct sockaddr in channel; /* holds IP address */

    if (argc != 3) fatal("Usage: client server-name file-name");
    h = gethostbyname(argv[1]); /* look up host's IP address */
    if (!h) fatal("gethostbyname failed");

    s = socket(PF INET, SOCK STREAM, IPPROTO TCP);
    if (s <0) fatal("socket");

    memset(&channel, 0, sizeof(channel));
    channel.sin family= AF INET;
    memcpy(&channel.sin addr.s addr, h->h addr, h->h length);
    channel.sin port= htons(SERVER PORT);
    c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
    if (c < 0) fatal("connect failed");

    /* Connection is now established. Send file name including 0 byte at end. */
    write(s, argv[2], strlen(argv[2])+1);
    /* Go get the file and write it to standard output. */
    while (1) {
        bytes = read(s, buf, BUF SIZE); /* read from socket */
        if (bytes <= 0) exit(0); /* check for end of file */
        write(1, buf, bytes); /* write to standard output */
    }
}

fatal(char *string) {
    printf("%s\n", string);
    exit(1);
}
```

Exemplu server

```
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER PORT 12345 /* arbitrary, but client & server must agree */
#define BUF SIZE 4096 /* block transfer size */
#define QUEUE SIZE 10

int main(int argc, char *argv[]) {

    int s, b, l, fd, sa, bytes, on = 1;
    char buf[BUF SIZE]; /* buffer for outgoing file */
    struct sockaddr in channel; /* holds IP address */

    /* Build address structure to bind to socket. */
    memset(&channel, 0, sizeof(channel)); /* zero channel */
    channel.sin family = AF INET;
    channel.sin addr.s addr = htonl(INADDR ANY);
    channel.sin port = htons(SERVER PORT);
    /* Passive open. Wait for connection. */
    s = socket(AF INET, SOCK STREAM, IPPROTO TCP); /* create socket */
    if (s < 0) fatal("socket failed");

    setsockopt(s, SOL SOCKET, SO REUSEADDR, (char *) &on, sizeof(on));
    b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
    if (b < 0) fatal("bind failed");

    l = listen(s, QUEUE SIZE); /* specify queue size */
    if (l < 0) fatal("listen failed");

    /* Socket is now set up and bound. Wait for connection and process it. */
    while (1) {
        sa = accept(s, 0, 0); /* block for connection request */
        if (sa < 0) fatal("accept failed");

        read(sa, buf, BUF SIZE); /* read file name from socket */
        /* Get and return the file. */
        fd = open(buf, O RDONLY); /* open the file to be sent back */
        if (fd < 0) fatal("open failed");
        while (1) {
            bytes = read(fd, buf, BUF SIZE); /* read from file */
            if (bytes <= 0) break; /* check for end of file */
            write(sa, buf, bytes); /* write bytes to socket */
        }
        close(fd); /* close file */
        close(sa); /* close connection */
    }
}
```

Serviciu fara conexiune - client



Creaza socket și aloca-i resurse de sistem:

int s = ***socket*** *(AF_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));

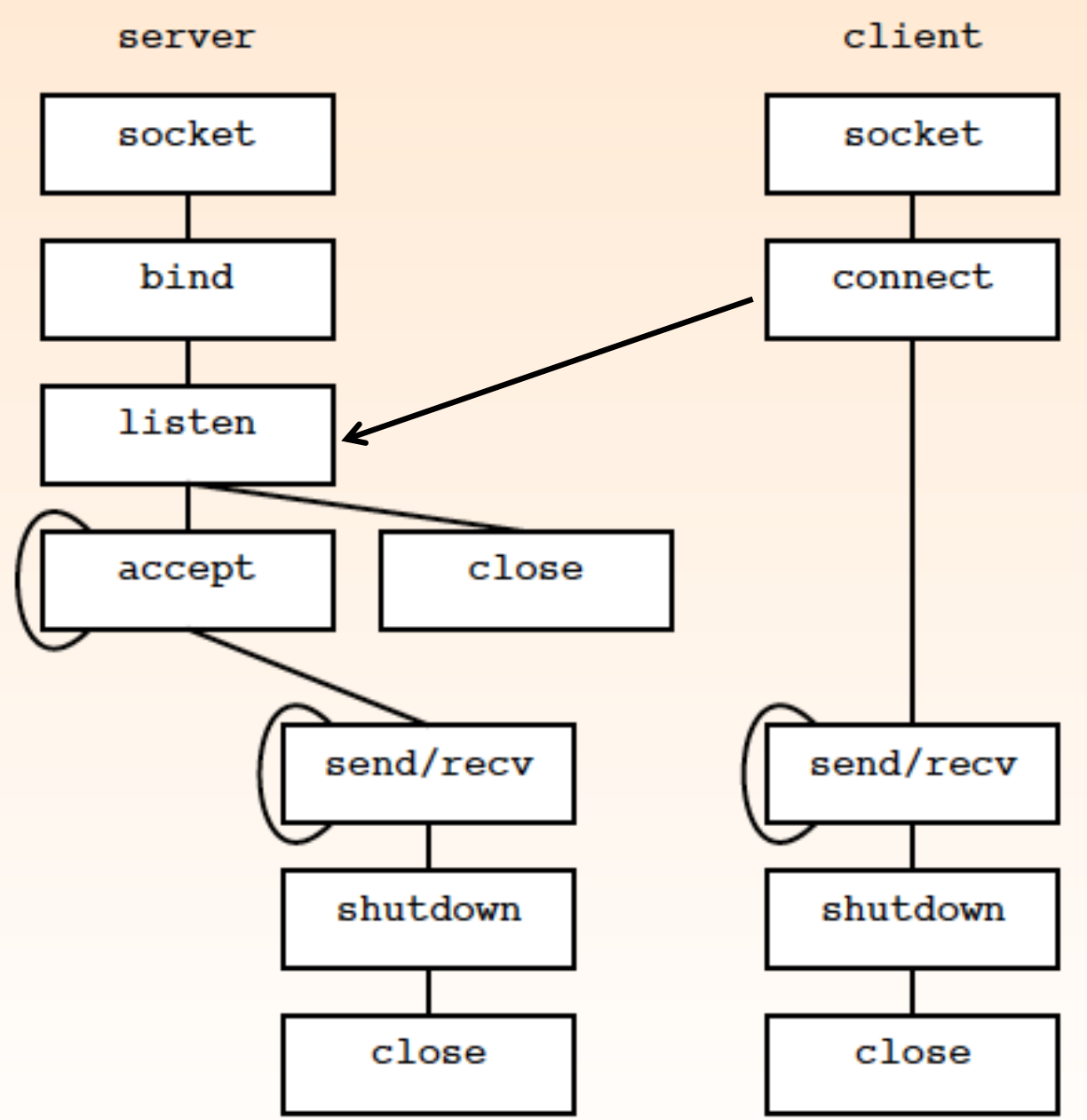
Opreste trimitere sau/și receptie de date

shutdown (s, SHUT_RD / SHUT_RDWR / SHUT_WR)

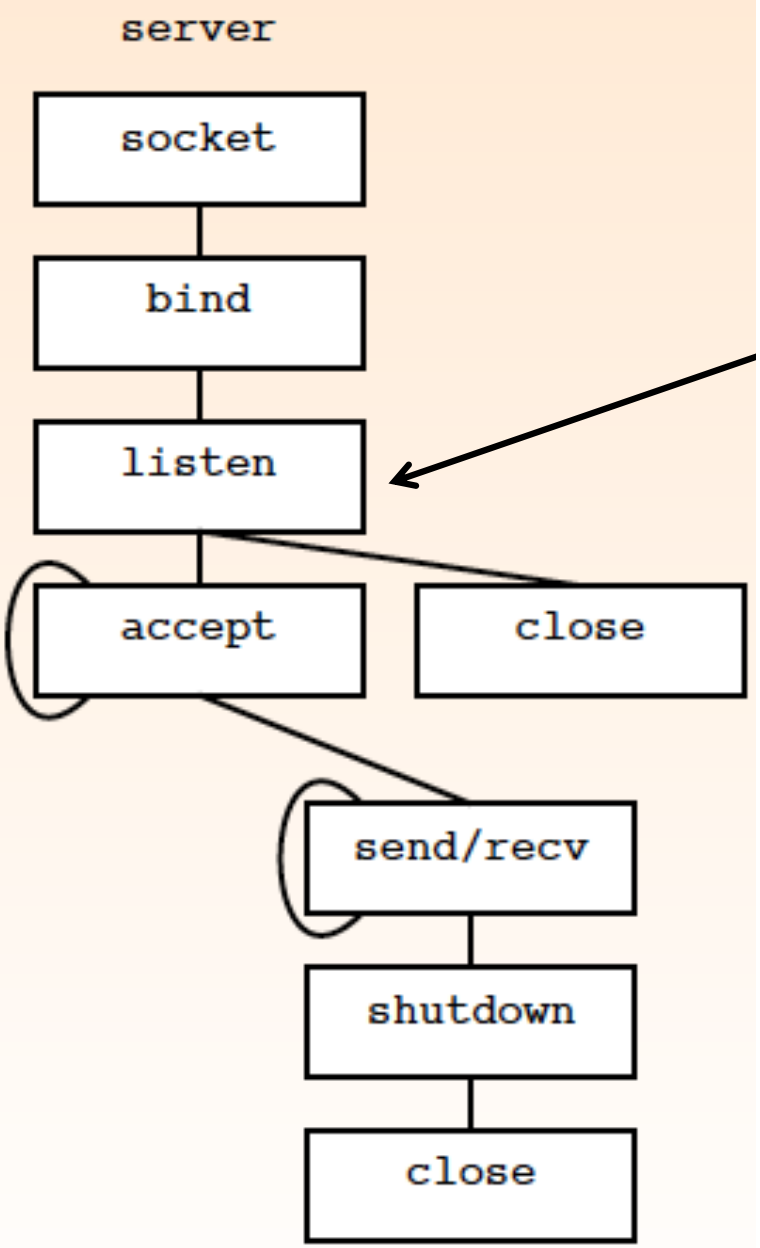
Termina utilizarea socket si elibereaza resursele alocate

close (s)

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 / primeste pe s etc.

Serviciu orientat pe conexiune - client

1. Creaza socket:

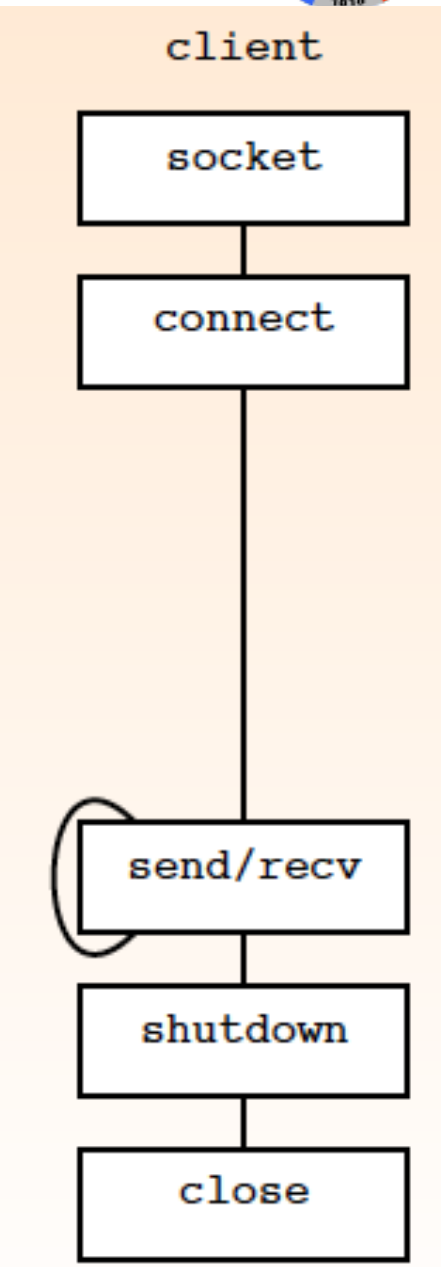
int s = ***socket*** *(AF_INET, SOCK_STREAM, IPPROTO_TCP);*

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 / 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)

Protocoale de Transport

Doua potocoale majore

- UDP
- TCP

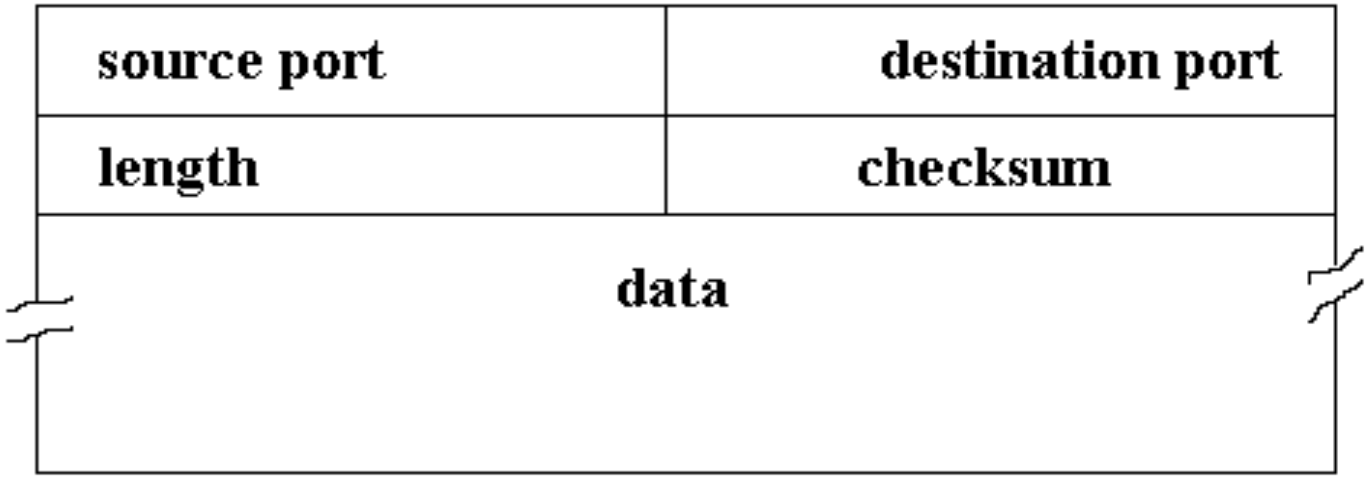
Aspecte discutate

- 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

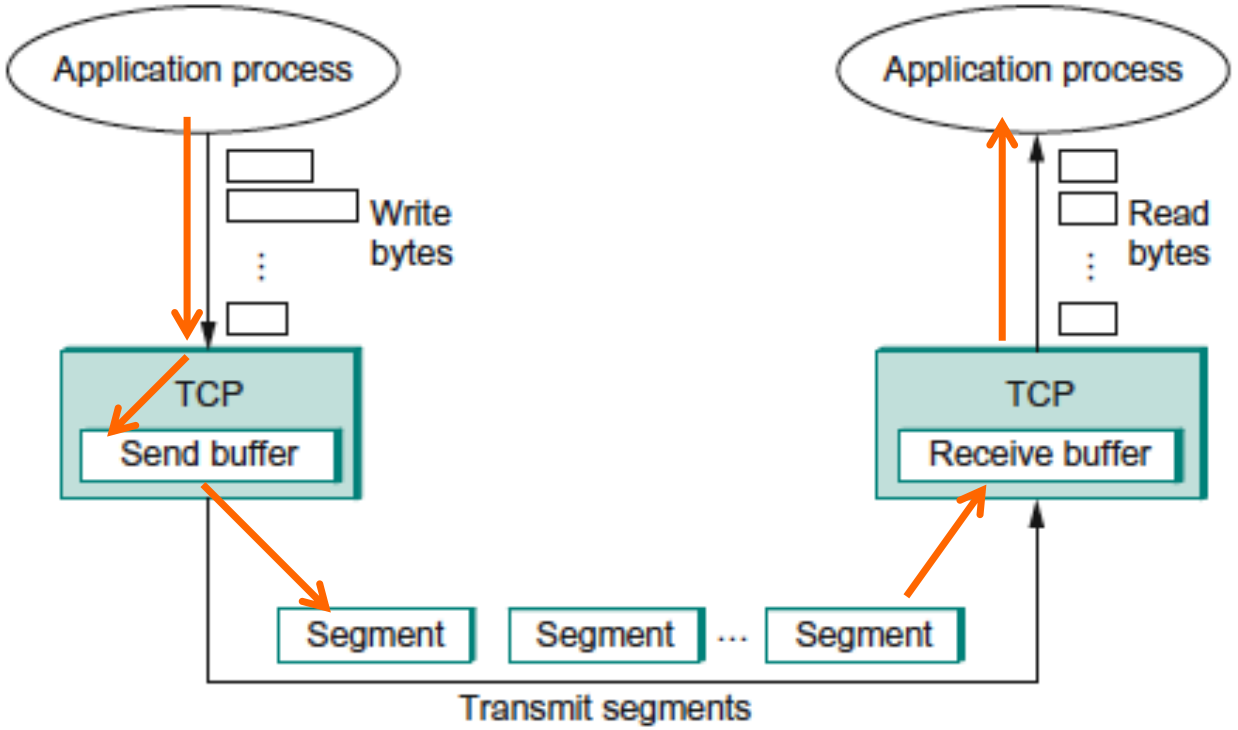
Livrare sigura pe retea nesigura (datagrame)

- ***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 octeti

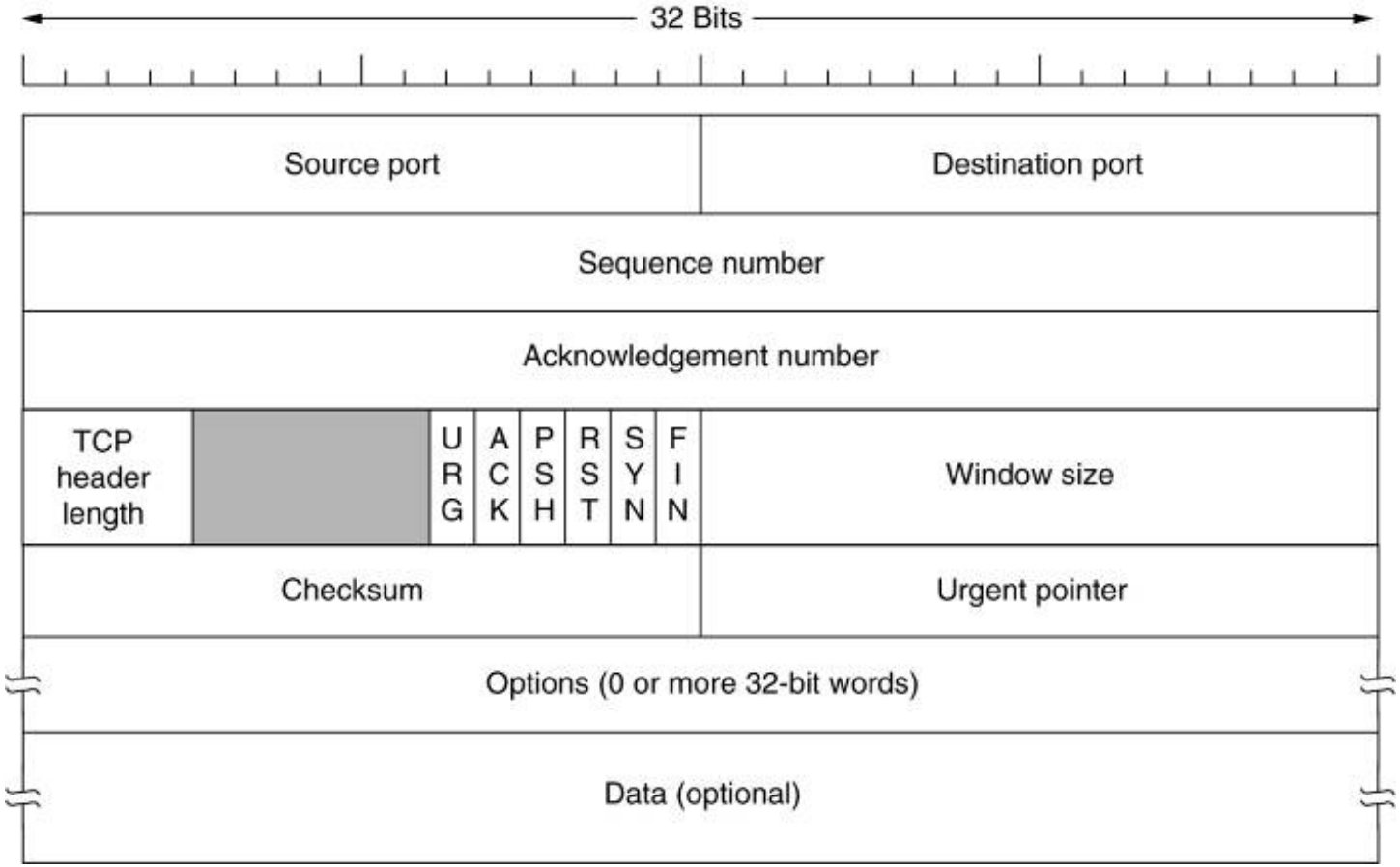


- Aplicația transmițătoare scrie octeți în conexiunea TCP
- TCP la sursă memorează octeții într-un buffer și transmite segmente
- TCP la destinație memorează segmentele într-un buffer
- Aplicația receptoare citește octeți (câți vrea!) din conexiunea TCP

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

Antet segment TCP



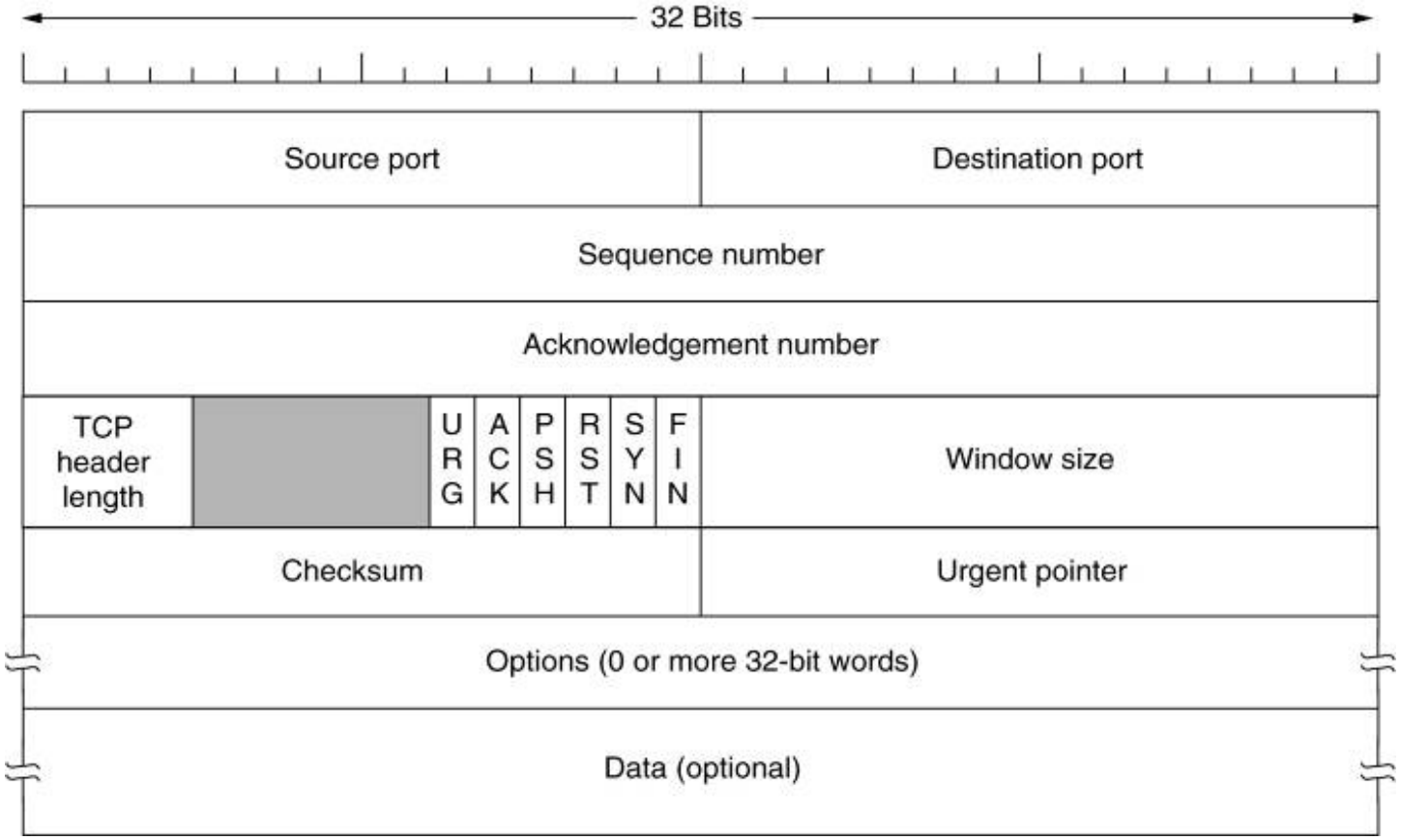
Sequence number – numarul primului octet din segment

Acknowledgement number – numarul urmatorului octet asteptat

Window size - numărul de octeţi care pot fi trimişi, începând cu octetul confirmat

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

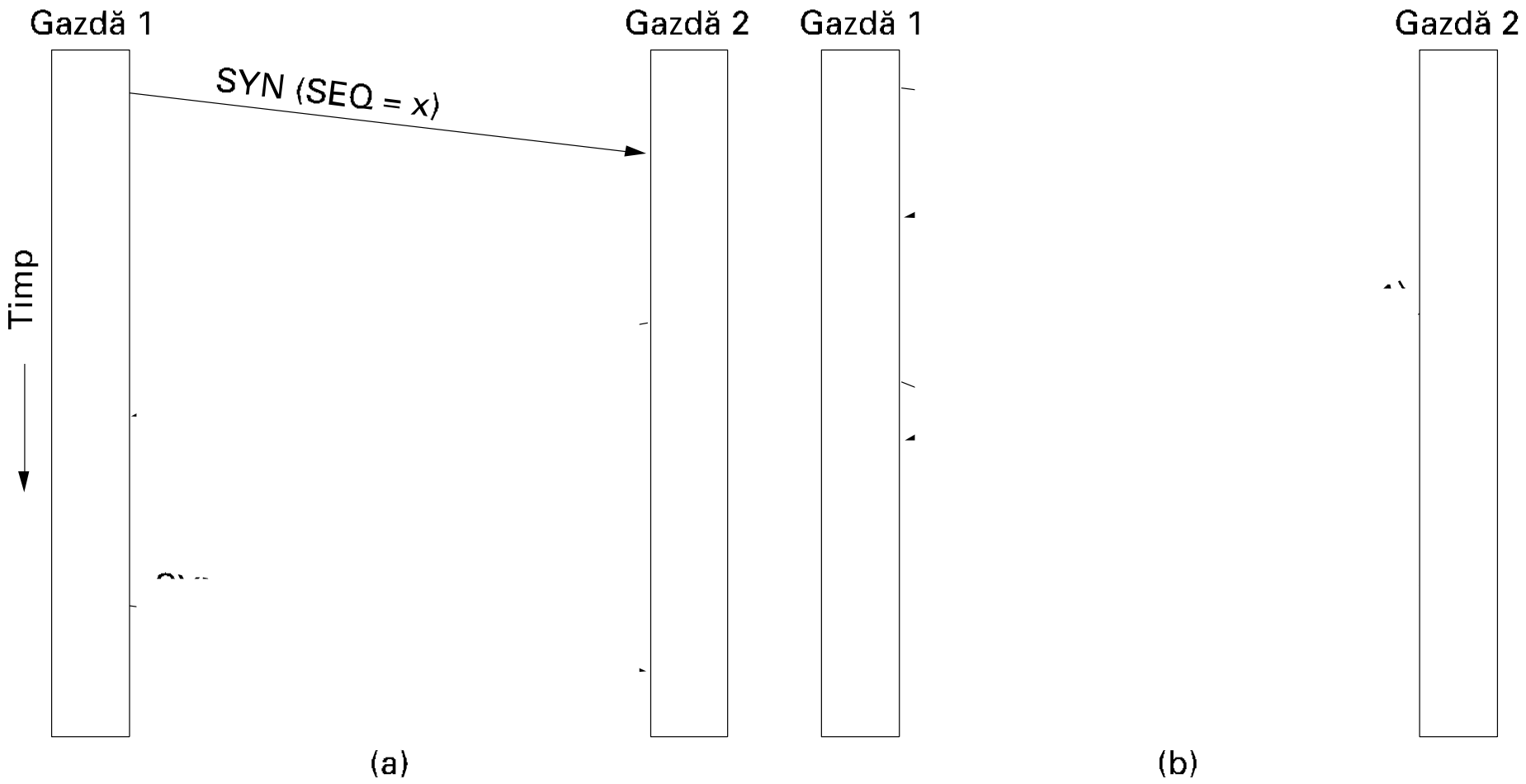
RST - close a connection due to an error

SYN - open connection

FIN - close a connection

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

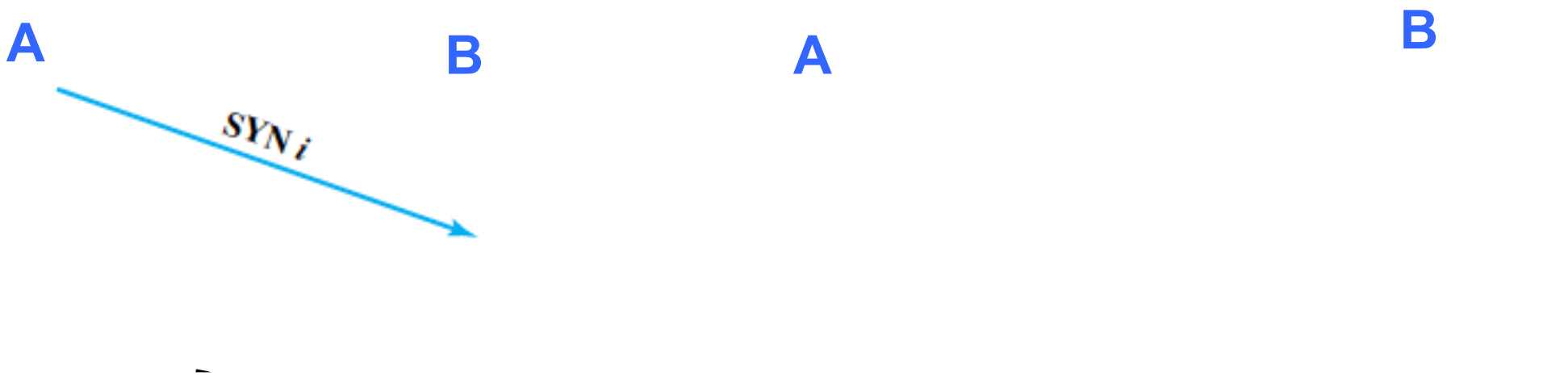
Stabilirea conexiunii - Three way handshaking



(a) Cazul normal.

(b) Coliziune.

Rejectarea conexiunii



SYN intarziat

B accepta

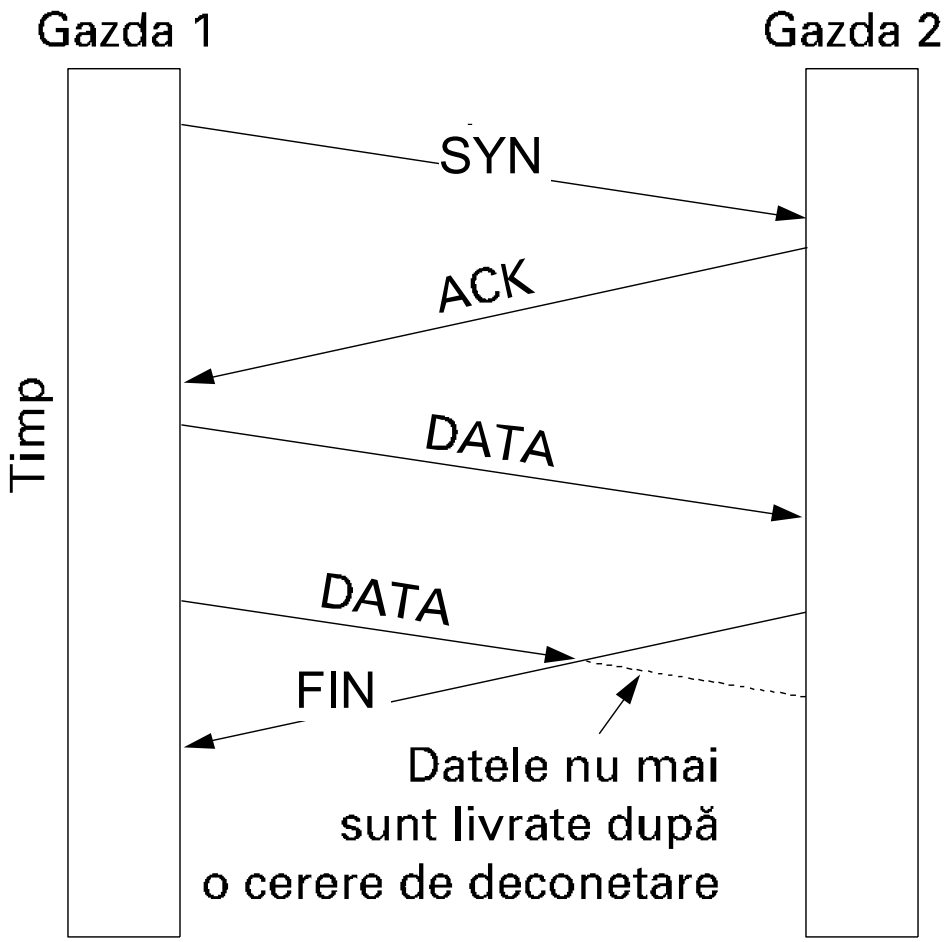
A rejecteaza

A initiaza conexiune (SYN i) + SYN k intarziat

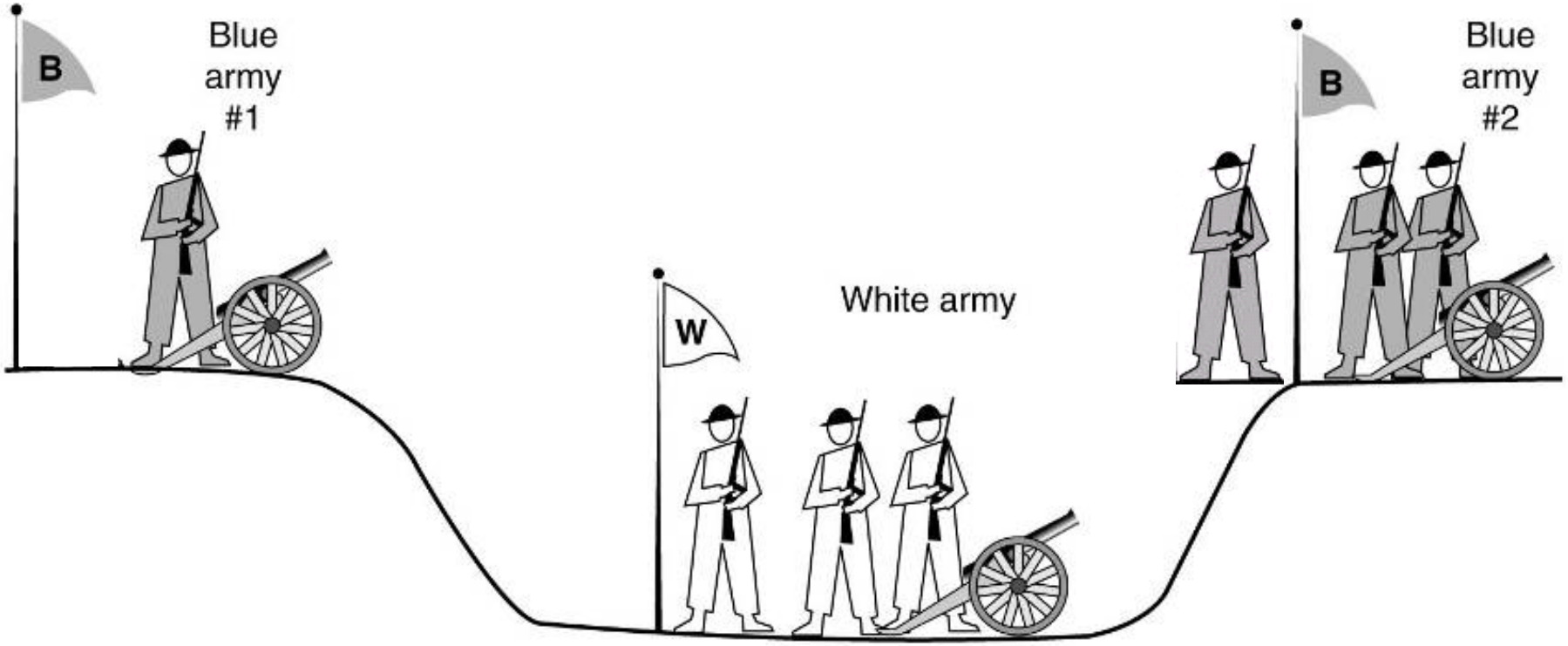
A refuza (RST)

B accepta SYN i – raspunde cu SYN j

Deconectare abrupta cu pierdere de date

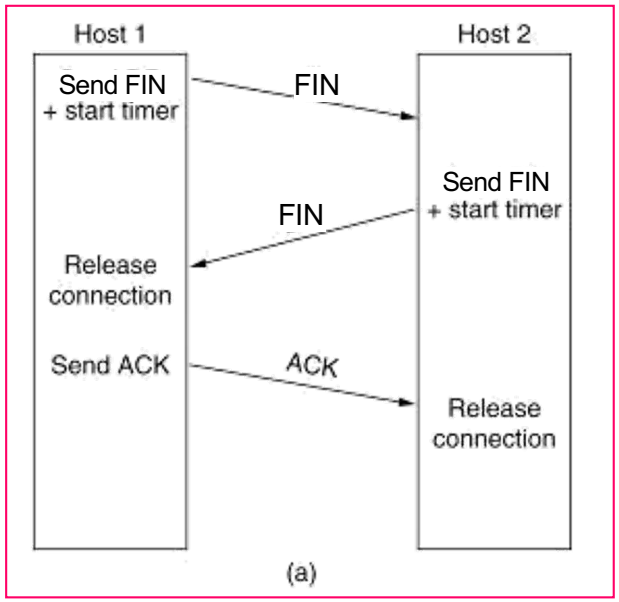


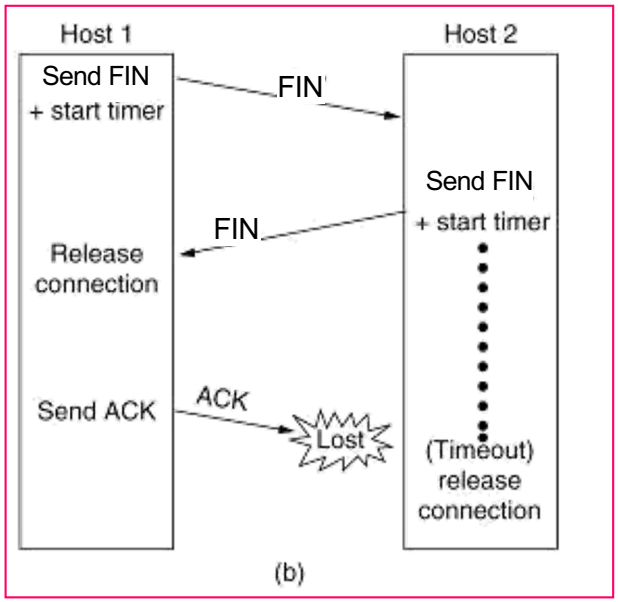
Problema celor doua armate



Patru scenarii de eliberarea conexiunii

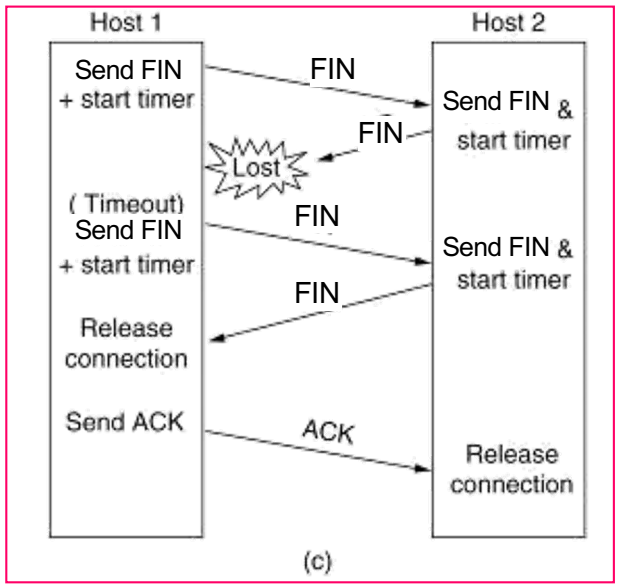
normal

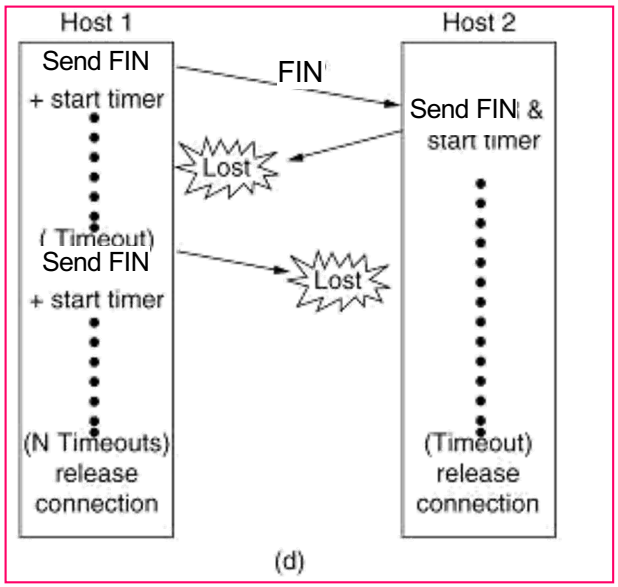




ACK final pierdut

raspuns pierdut

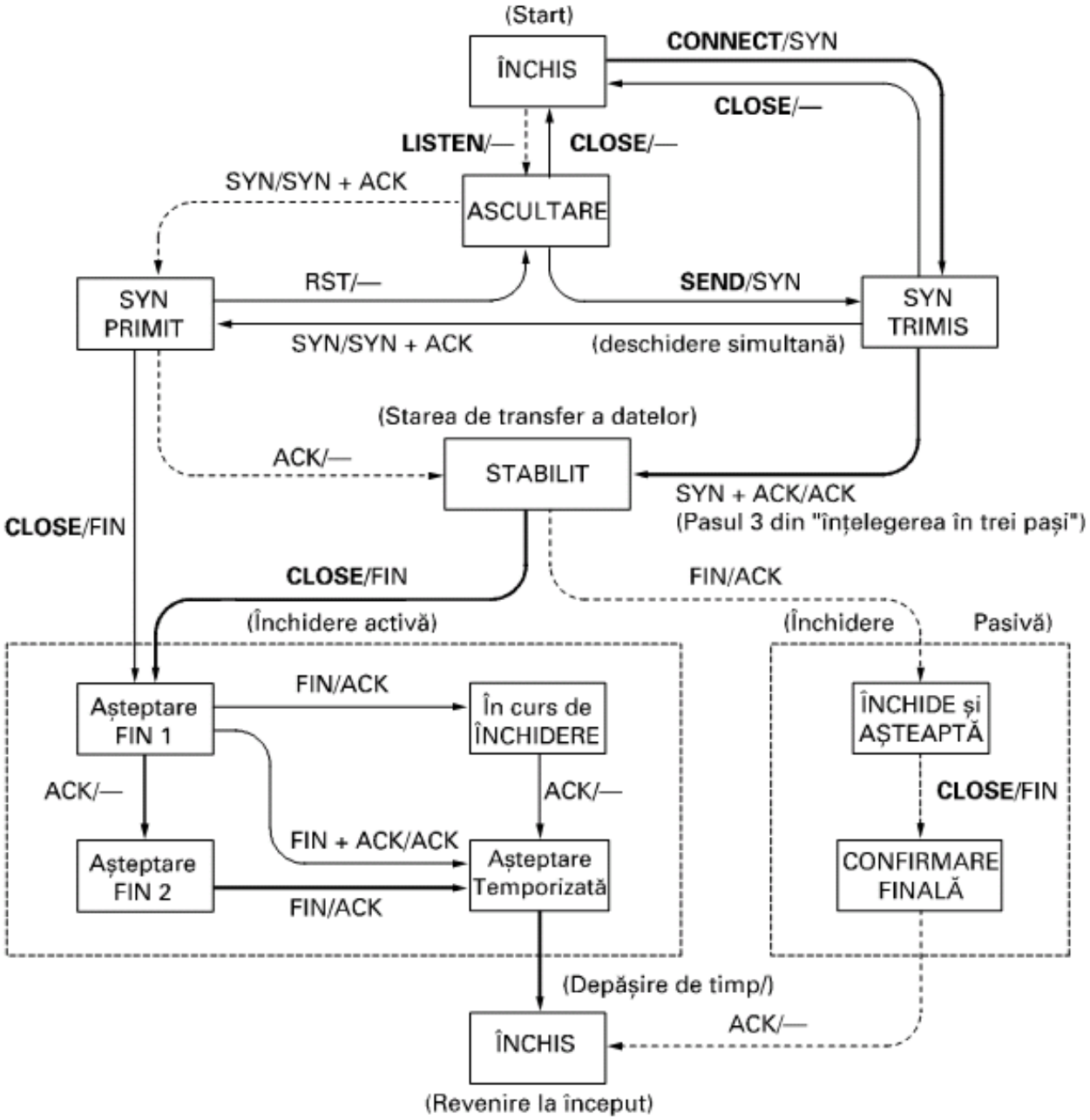




Raspuns si FIN pierdute

Gestiunea Conexiunilor TCP (mașina de stări)

- 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

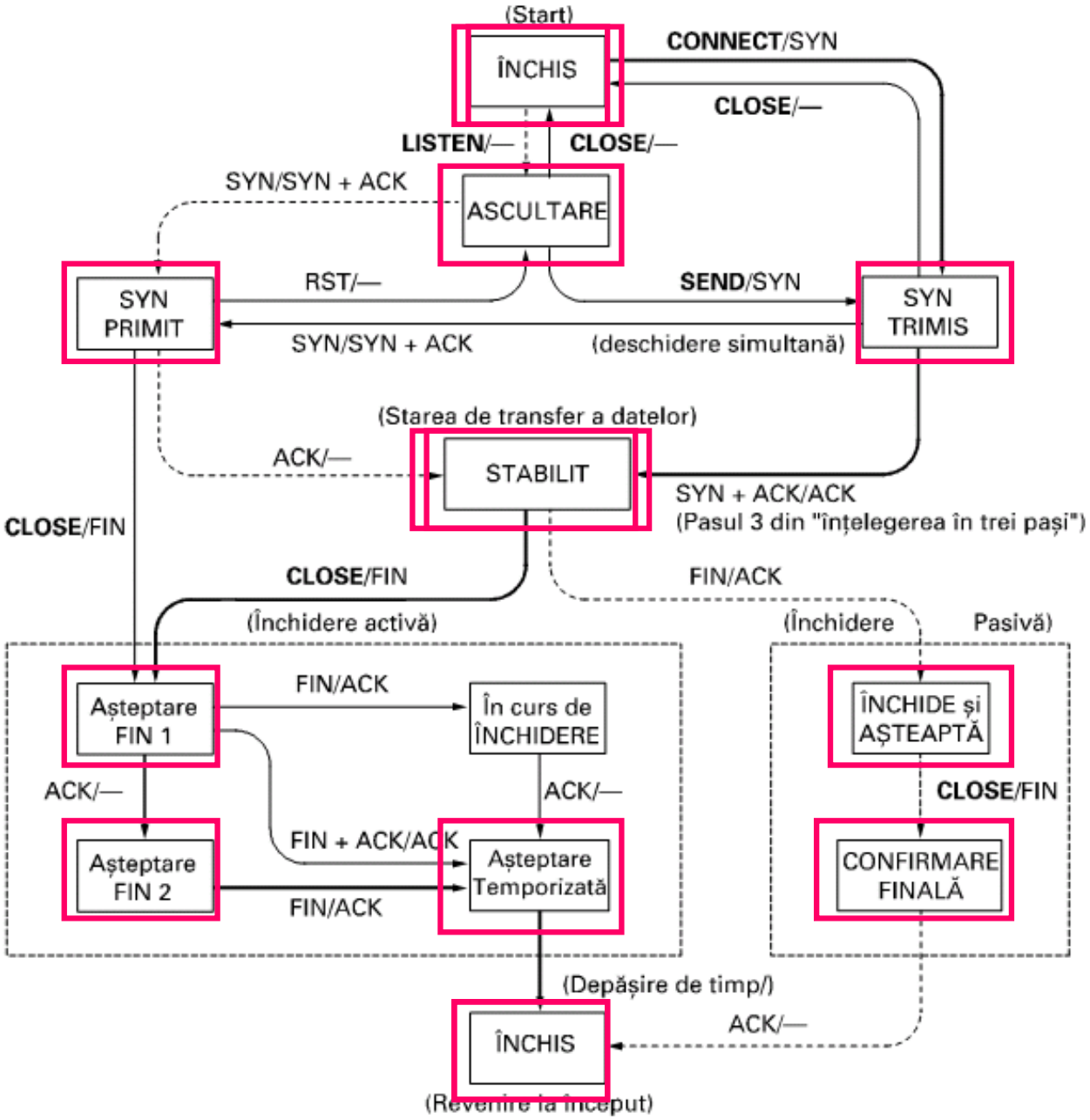


Client

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

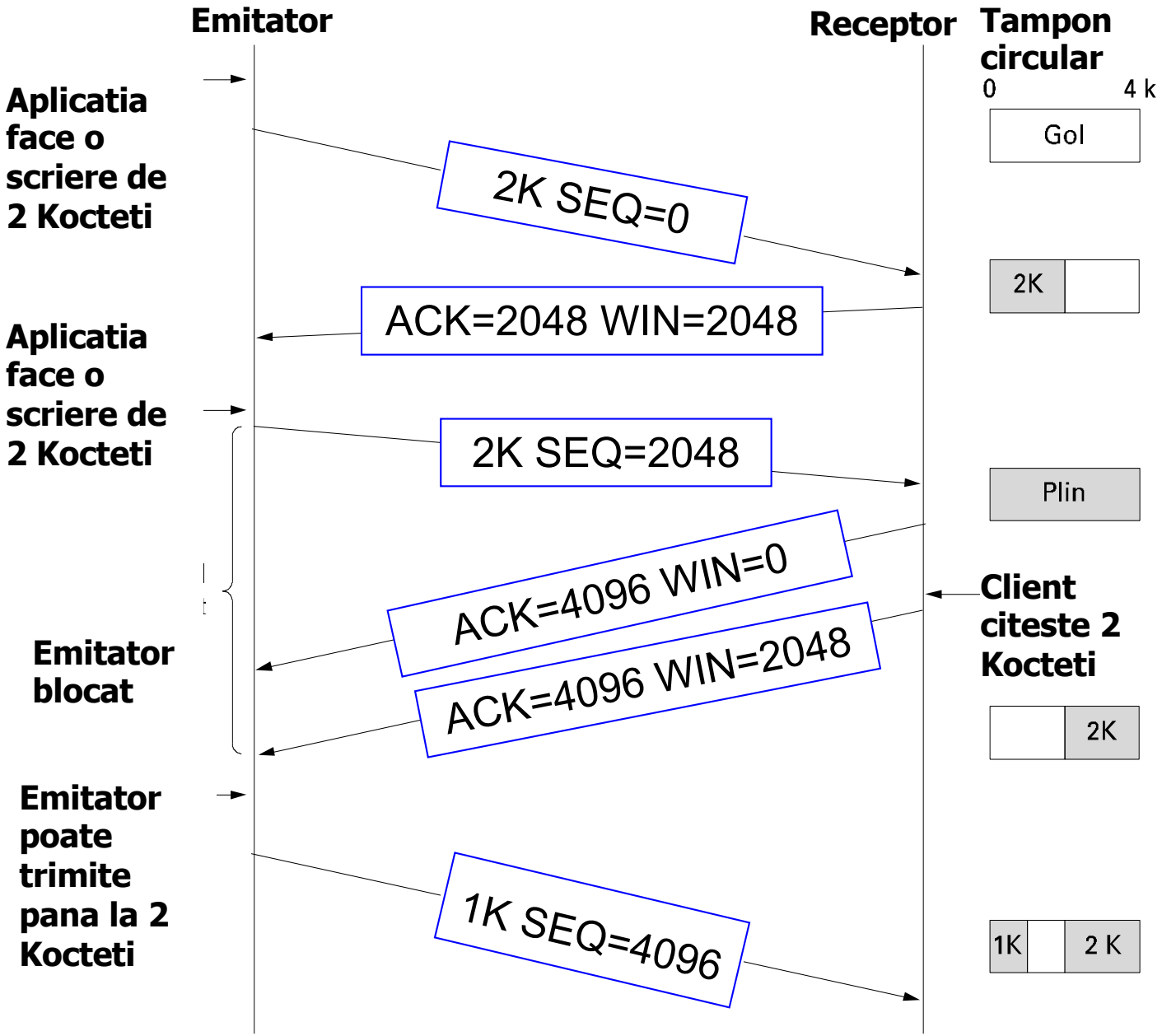
la **receptie**

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

2 - Acknowledgement number

Corectia se face prin retransmisie

Controlul fluxului de date



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 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 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

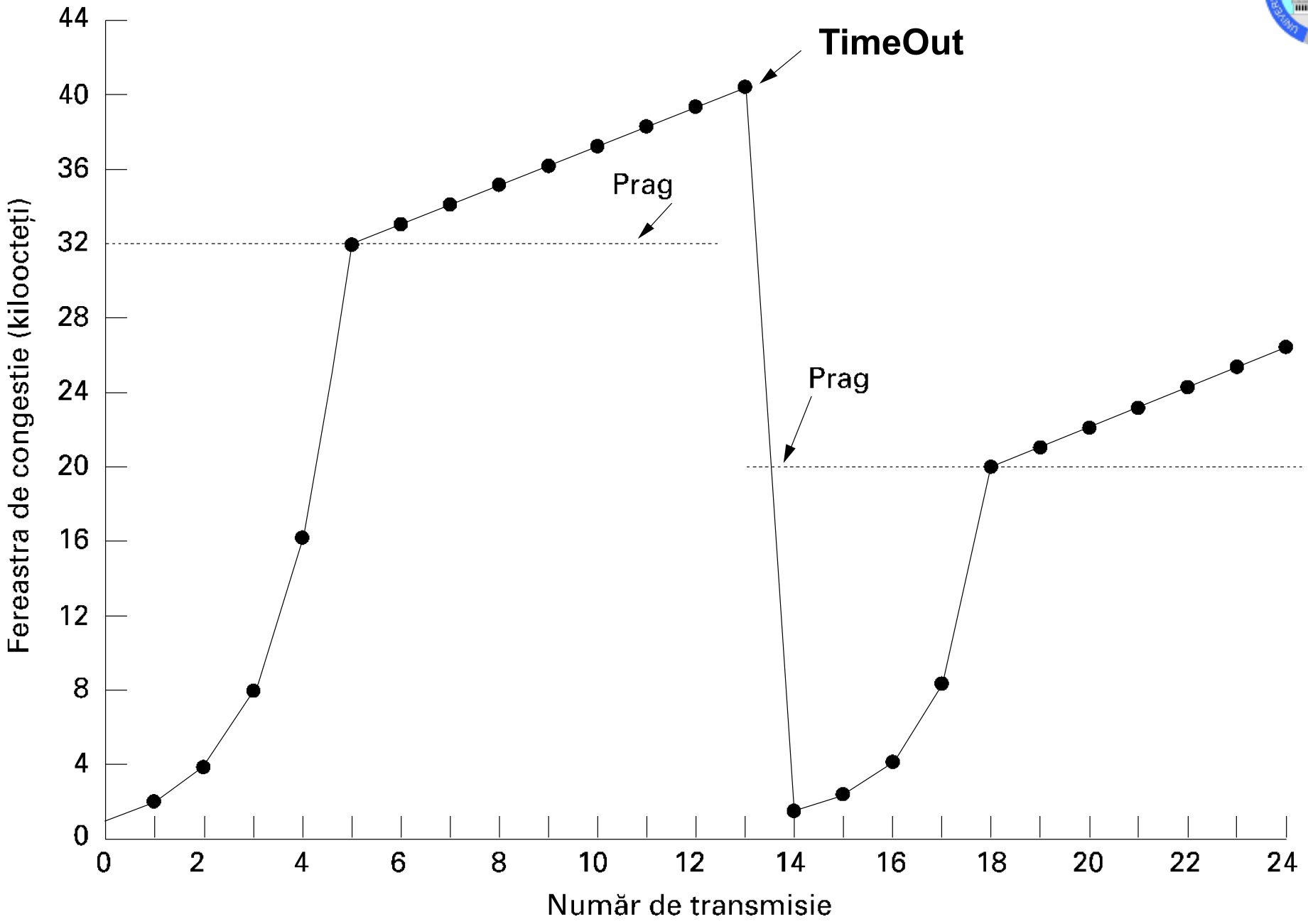
    ➔ ferestre de până la 2**30 octeți

Controlul congestiei

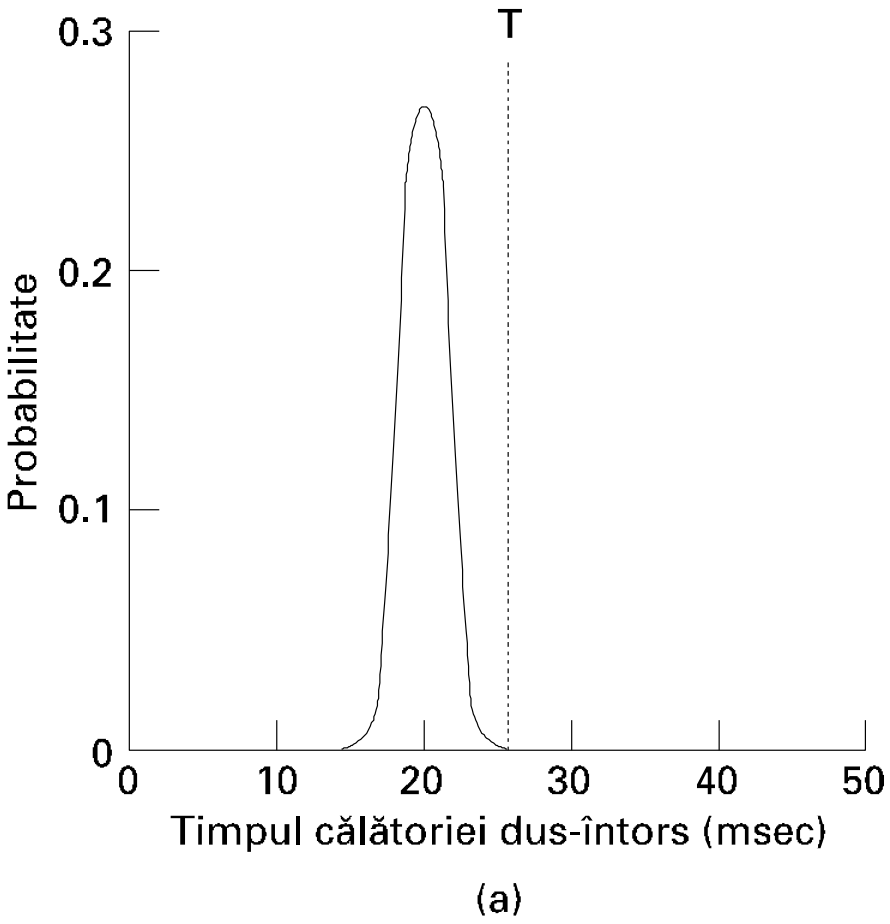
- Fluxul de date transmis pe o conex. TCP limitat de minimul dintre:
  - dimensiunea fereastrei receptorului
  - capacitatea retelei (**fereastra de congestie**)
- Algoritm de stabilire fereastra de congestie
  - transmite un segment de dimensiune maximă pe conexiunea stabilita
  - dubleaza volumul de 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)

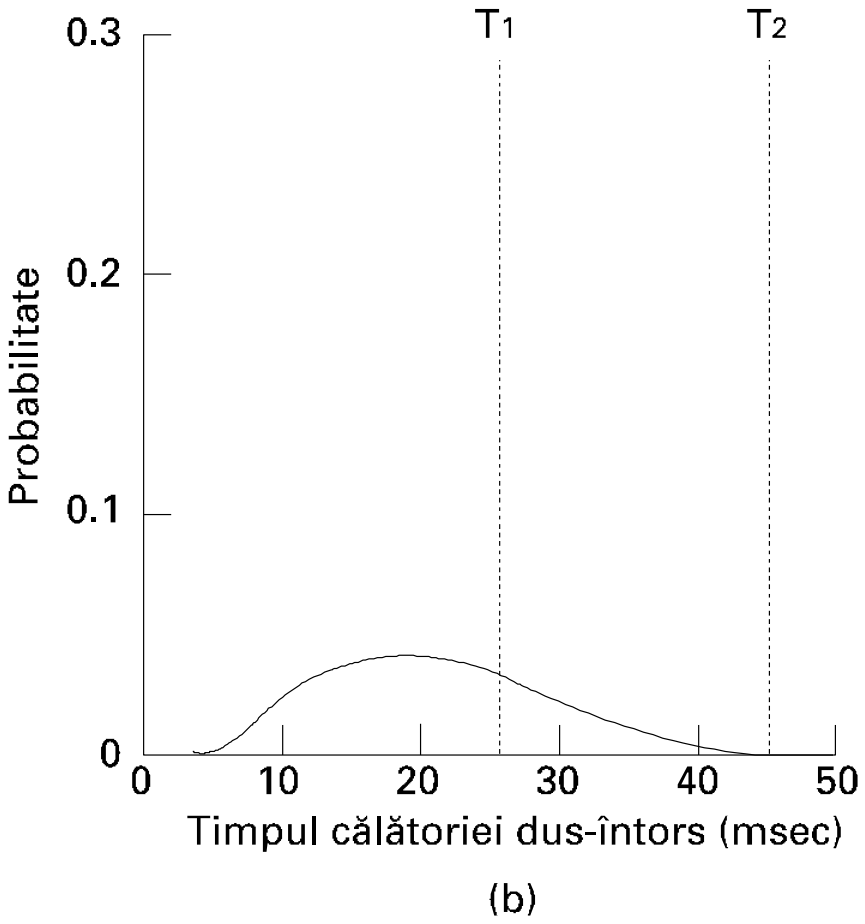
Controlul congestiei

- Algoritmul de control al congestiei
  - foloseste un **prag** (threshold)
  - la un timeout pragul 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)


TimeOut
Prag
Prag
Fereastra de congestie (kiloocteți)
Număr de transmisie
0
4
8
12
16
20
24
28
32
36
40
44
0
2
4
6
8
10
12
14
16
18
20
22
24

Gestiunea ceasurilor in TCP





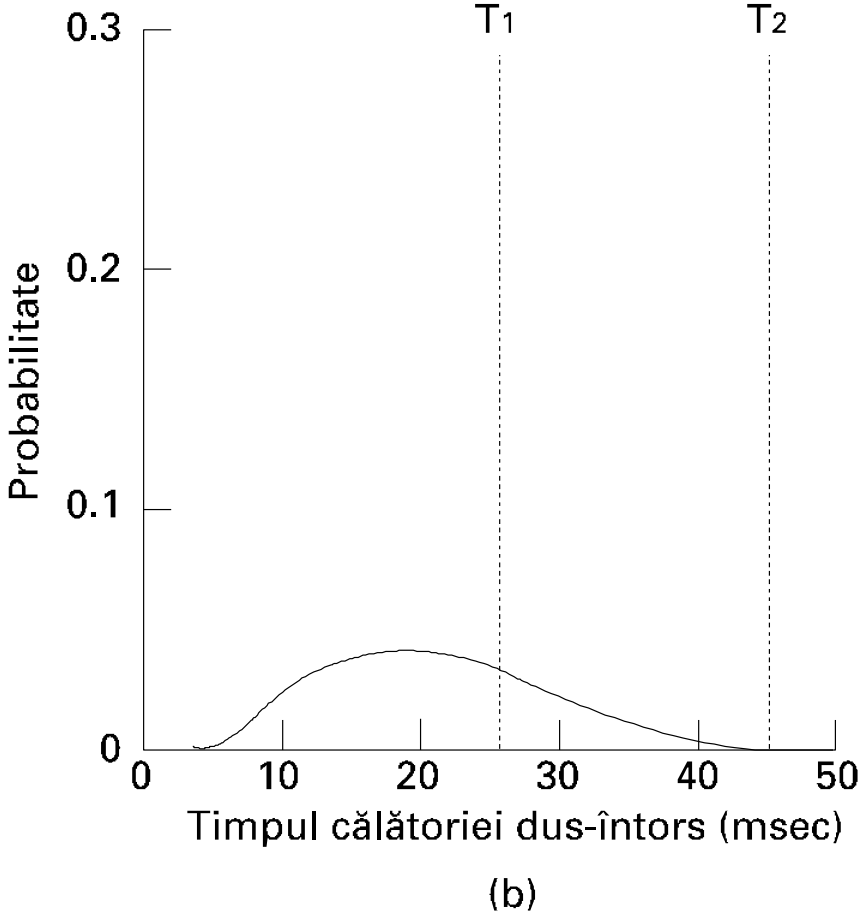
(a) Densitatea de probabilitate a timpilor de sosire ACK in nivelul legatura de date.

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

Gestiunea ceasurilor in TCP (2)

Configurare incorecta – 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

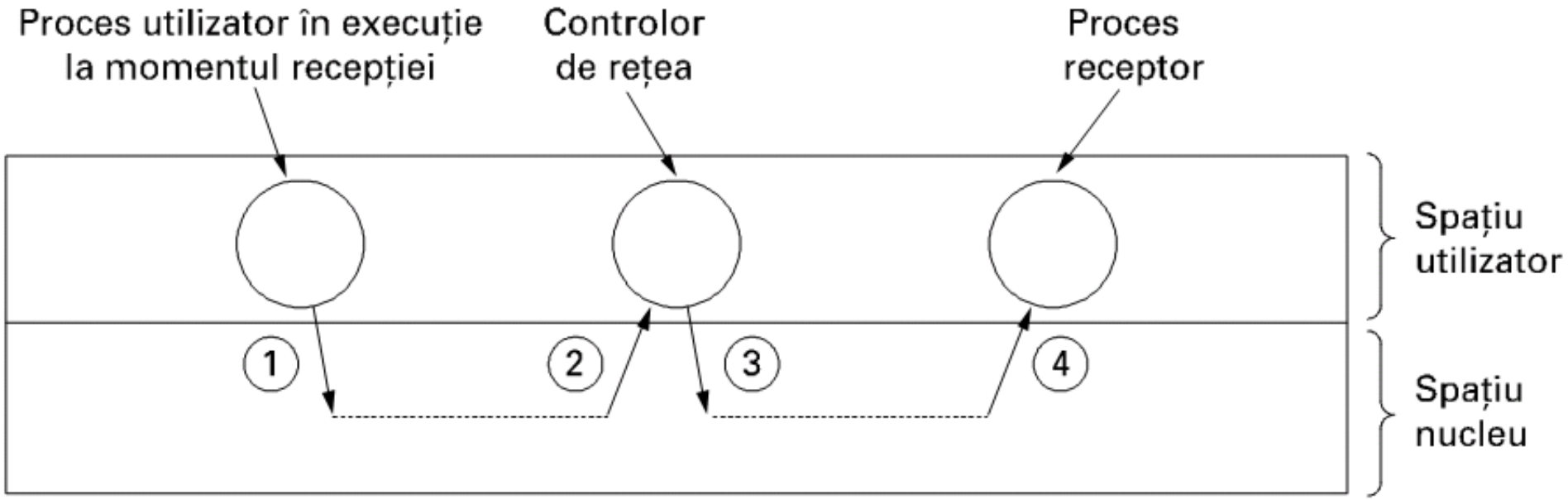
- Alegere dupa *deviatia standard (DS);*

  D aproximeaza DS

  D = α*D + (1- α)*|RTT-M|

  RTO = RTT + 4*D

Proiectarea pentru performanta



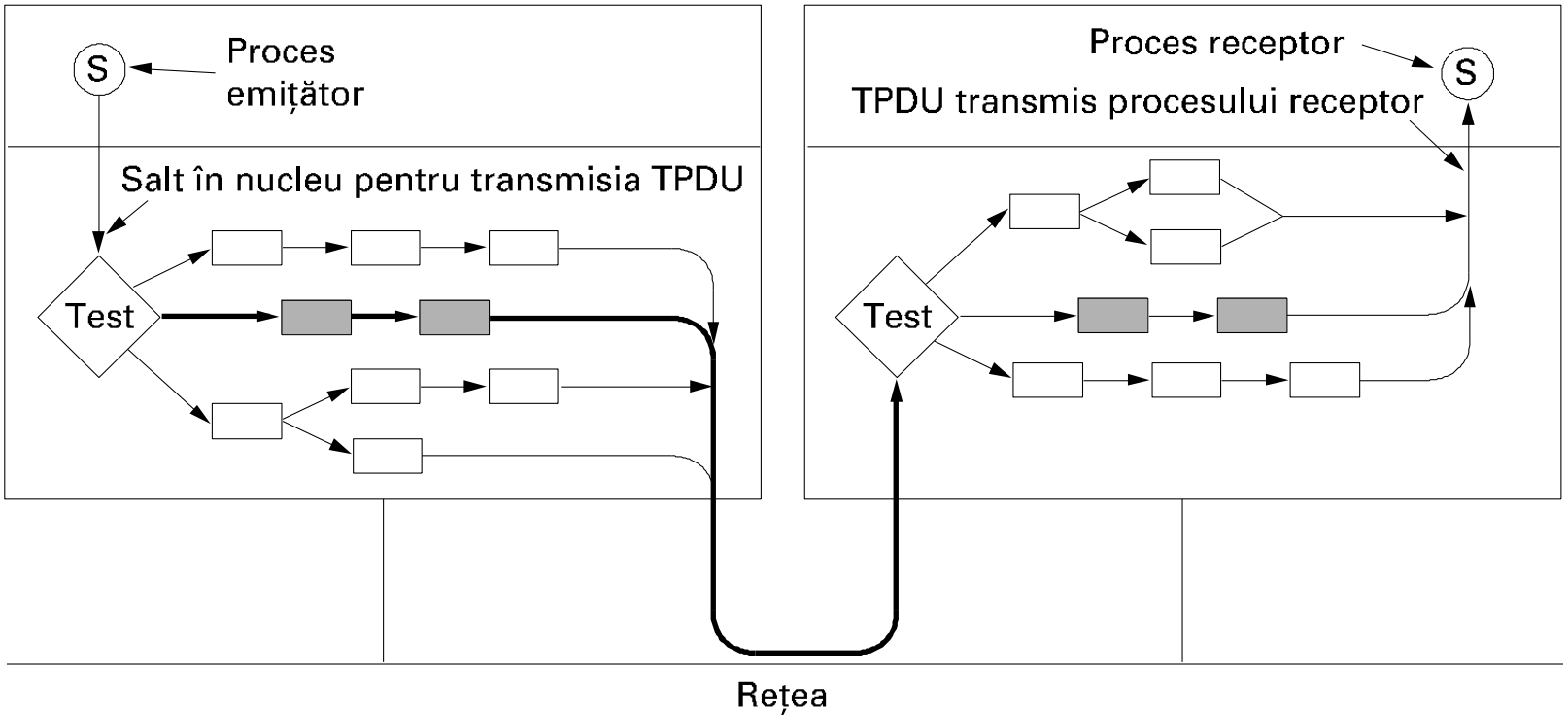
Problema: Multe schimbari de context user – kernel pentru a manevra pachete cu un Controlor de retea in spatiul utilizatorului.

Controlorul asambleaza segmentele pentru a le furniza apoi procesului receptor

Solutia: reducerea numar schimbari de context - acumularea mesajelor sosite, in memorie tampon si livrarea in grup catre utilizator.

Similar, gruparea mesajelor de transmis, in memorie tampon si trimiterea lor grupata

Prelucrare rapida TPDU



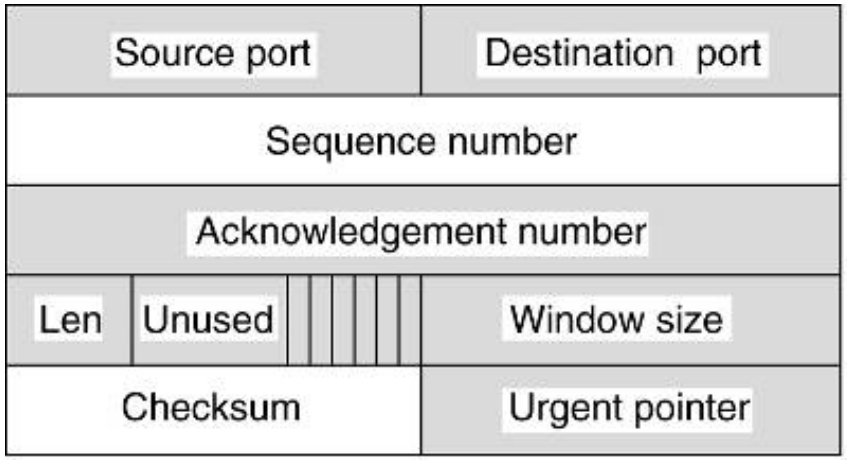
Calea rapida intre transmitator si receptor este cu line groasa. Pasii sunt reprezentati cu gri.

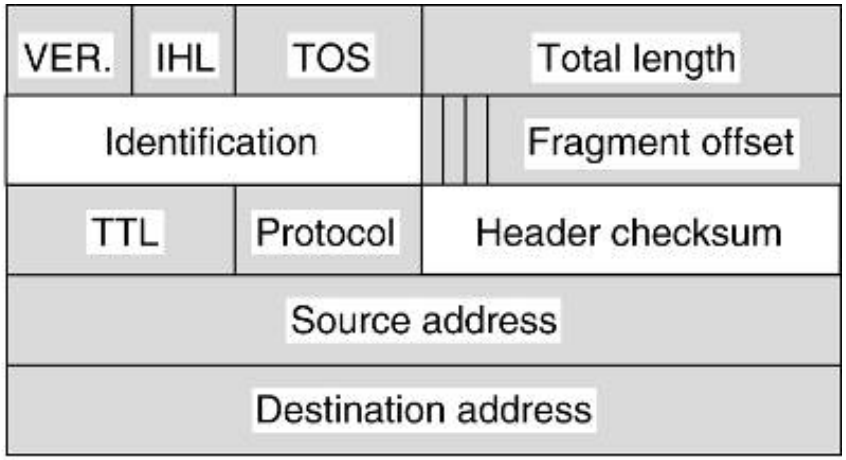
Test caz normal:

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

} –> alege fast path

Prelucrare rapida TPDU (2)





(a) Antet TCP.

(b) Antet IP.

Transmitator

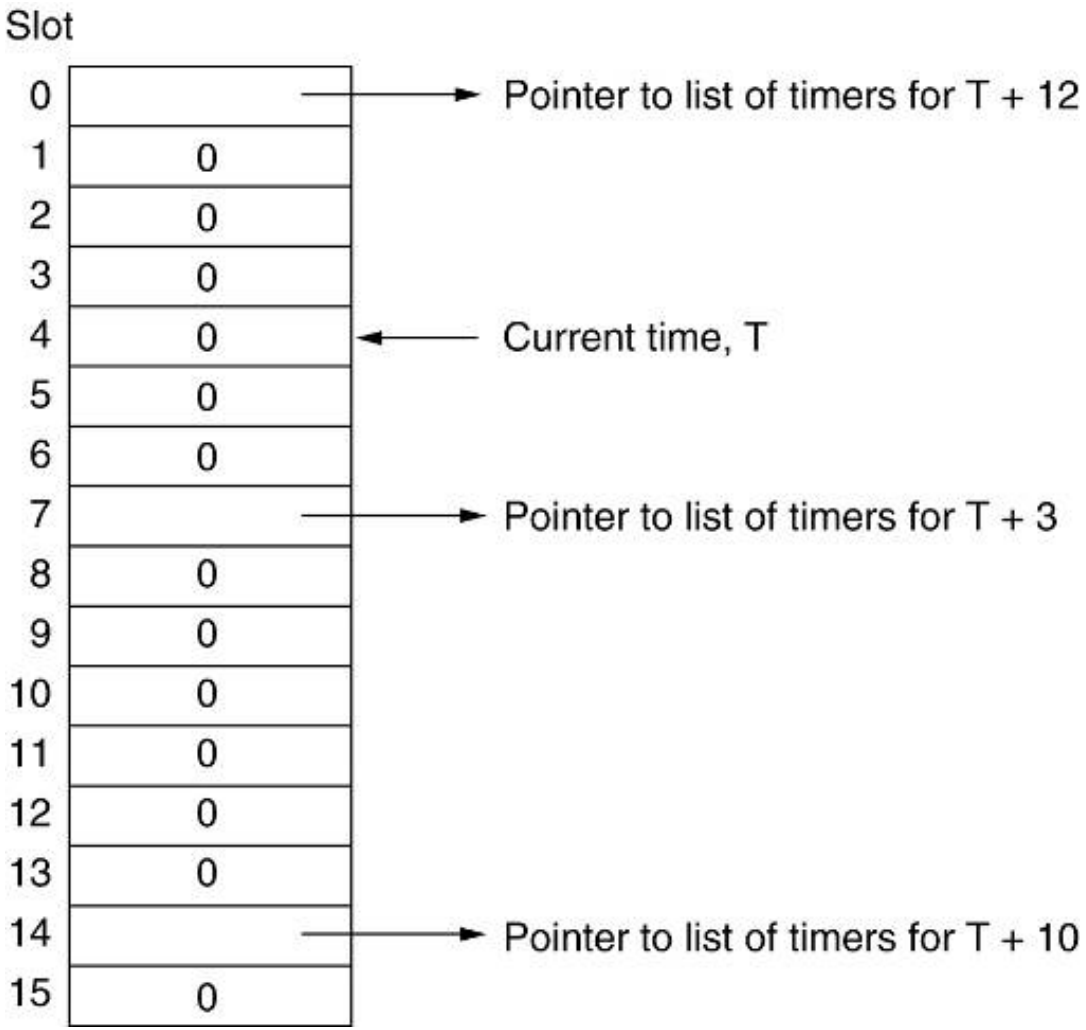
Pastreaza un prototip de mesaj in entitatea de transport – campuri nemodificate in unitati de date consecutive; la fel pentru pachet IP

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

Timp curent 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.

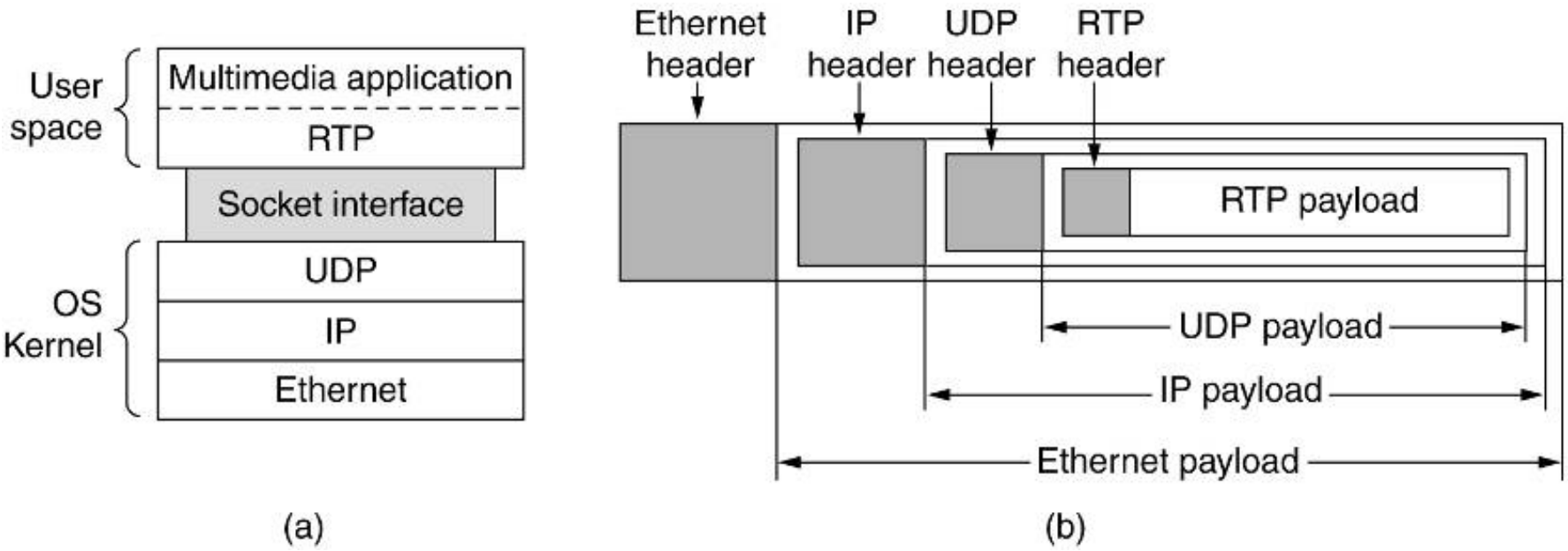
Functie principala: multiplexare fluxuri RTP si transmiterea lor ca un singur șir de pachete UDP

La receptie, RTP livreaza aplicatiei datele multimedia

Fara retransmiterea pachetelor

- UDP nu asigura corectitudinea transmisiei
- RTP atribuie un numar de secventa fiecarui pachet
- tratarea unui pachet pierdut se face la aplicatie: receptorul poate “interpola” pachetul absent sau ignora eroarea

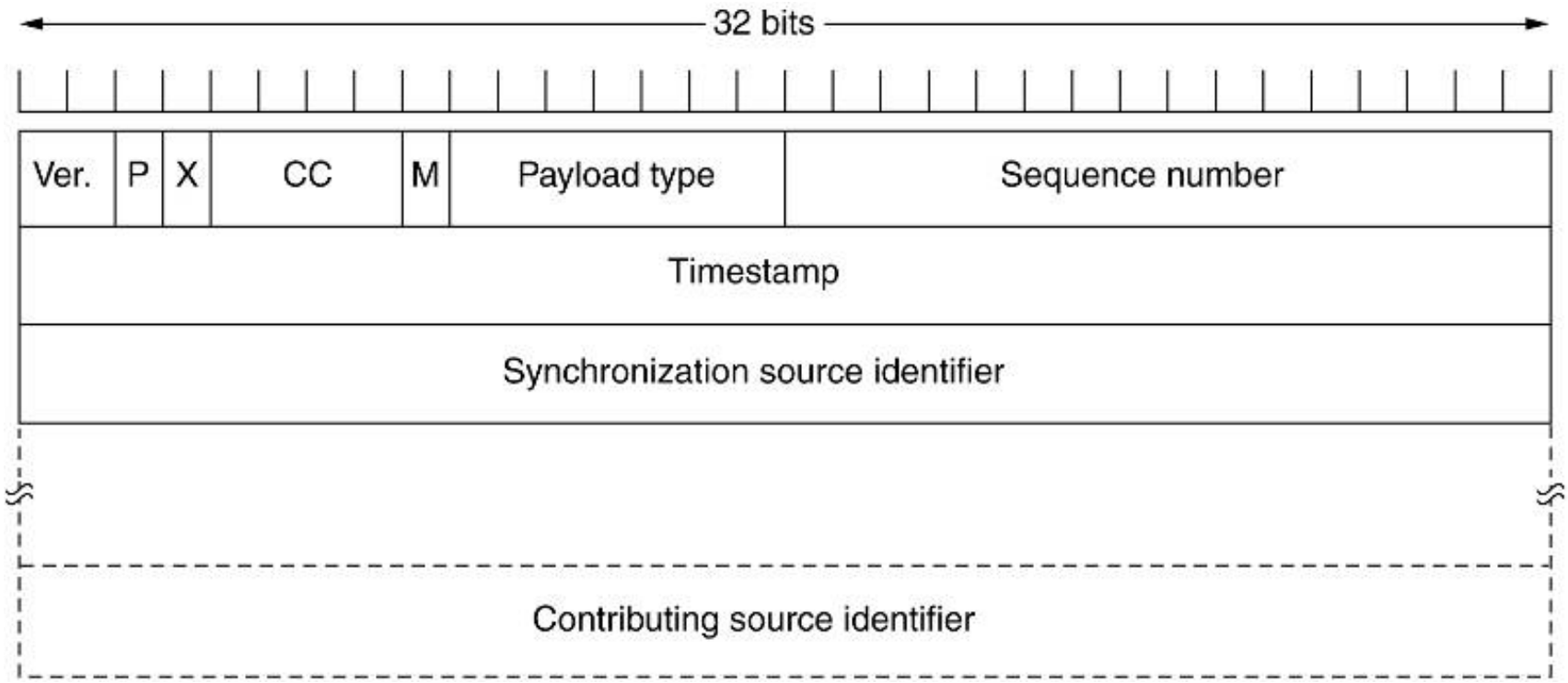
The Real-Time Transport Protocol (2)



(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

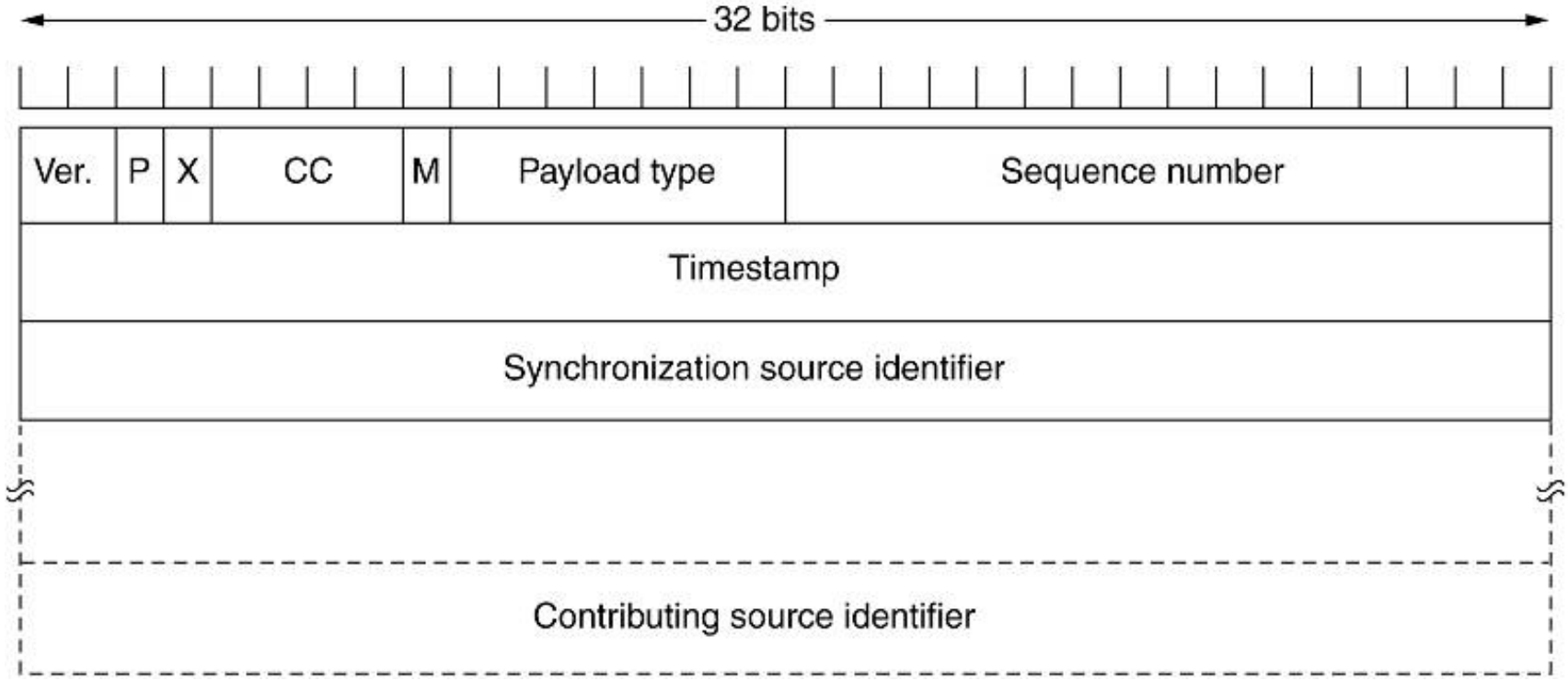
Synchro – identifica sursa de sincronizare (ex. microfon, camera video)

• 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

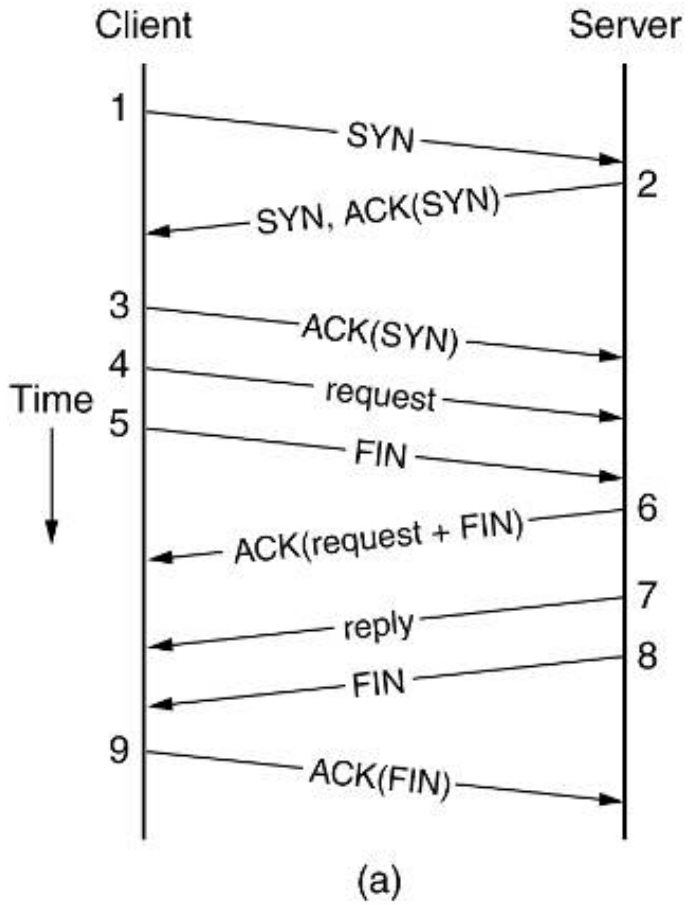
X – extension - antet extins (primul cuvant da lungimea)

CC – no. antete surse contributoare

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

Payload type – e.g. MP3

TCP Tranzactional



(a)

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

(b) RPC folosind T/T

Permite transfer de date odata cu stabilirea conexiunii