



Nivelul transport



Servicii ale nivelului Transport

- **Servicii furnizate**
 - transfer de date eficient, sigur si cu raport cost performanta bun
 - interfața uniforma cu utilizatorii
- **Caracteristici**
 - servicii capăt la capăt (host to host)
 - doua tipuri de servicii:
 - orientate pe conexiune (connection oriented)
 - fără conexiune (connectionless)



Primitivele serviciului de transport ISO-OSI

Orientate pe conexiune

1. T-CONNECT.request (callee, caller, exp, quality, data)
2. T-CONNECT.indication (callee, caller, exp, quality, data)
3. T-CONNECT.response (quality, respondent, exp, data)
4. T-CONNECT.confirm (quality, respondent, exp, data)

5. T-DISCONNECT.request (data)

6. T-DISCONNECT.indication (reason, data)

7. T-DATA.request (data)

8. T-DATA.indication (data)

9. T-EXPEDITED-DATA.request (data)

10. T-EXPEDITED-DATA.indication (data)

Fara conexiune

11. T-UNITDATA.request (callee, caller, quality, data)
12. T-UNITDATA.indication (callee, caller, quality, data)

Întârzierea la stabilirea conexiunii
 Probabilitatea de insucces la stabilirea conexiunii
 Productivitatea
 Întârzierea la transfer
 Rata reziduală a erorilor
 Protecția
 Prioritatea
 Reziliența



Model simplificat Serviciu de Transport

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

Unități de date de transport - TPDU

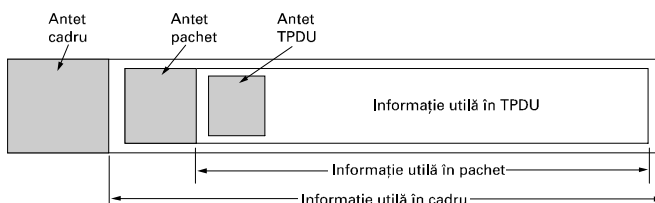
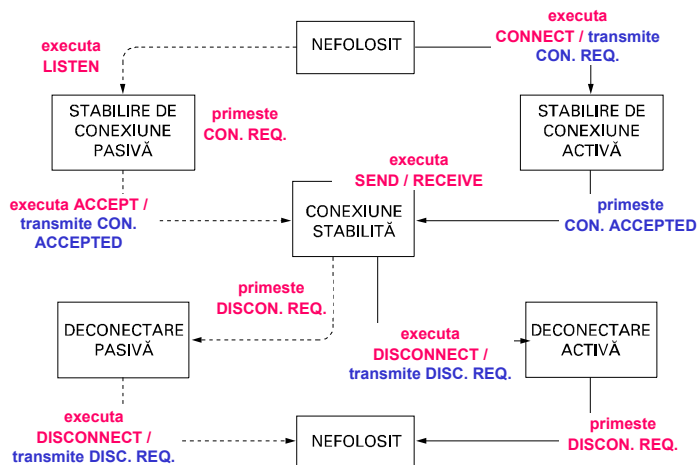




Diagrama de conectare / deconectare



Socket API

- Interfața pentru servicii de transport
- Oferit ca bibliotecă utilizator sau funcții OS
- Folosește descriptori (ca la fișiere)
- *Socket API* este
 - Originar din Berkeley BSD UNIX
 - Disponibil pe Windows, Solaris, etc.
- Nu e standard de jure ci *standard de facto*



Primitive de serviciu Socket API

- `socket_descriptor = socket` (`protocol_family`, `communication_type`, `protocol`)
- `close` (`socket_descriptor`)
- `bind` (`socket_descriptor`, `local_address`, `address_length`)
- `listen` (`socket_descriptor`, `queue_size`)
- `newsock = accept` (`original_socket_descriptor`, `client_socket_address`, `client_addreslen`)
- `connect` (`client_socket_descriptor`, `server_socket_address`, `server_sockaddress_length`)
- `sendto` (`socket_descriptor`, `data_address`, `data_length`, `flags`, `dest_address`, `destaddress_length`)
- `recvfrom` (`socket_descriptor`, `buffer_address`, `buffer_length`, `flags`, `sender_address`, `sendaddress_length`)
- `write` (`socket_descriptor`, `data_address`, `data_length`)
- `read` (`socket_descriptor`, `buffer_address`, `buffer_length`)



socket

`socket_descriptor = socket` (`protocol_family`, `communication_type`, `protocol`)

deschide un socket

intoarce `socket_descriptor` folosit in apelurile urmatoare

`protocol_family` selecteaza familia de protoacoale:

PF_INET - protoacoale Internet

PF_APPLETALK - protoacoale AppleTalk

`communication_type` selecteaza tipul de comunicare

SOCK_DGRAM – fara conexiune

SOCK_STREAM – orientat pe conexiune

`protocol` specifica protocolul

IPPROTO_TCP - TCP

IPPROTO_UDP - UDP



close

`close (socket_descriptor)`

Temina utilizarea descriptorului

bind

`bind (socket_descriptor, address, address_length)`

Leaga socket cu o adresa si un port

Pentru

server – leaga la port local

client – leaga la adresa si port la distanta (server)



Format adresa

Format generic:

```
struct sockaddr { u_char sa_len;      /* total length of address */
                  u_char sa_family;  /* family of the address */
                  char sa_data[14];  /* address */
}
```

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 */
}
```



listen

listen (socket_descriptor, queue_size)

Folosit de server: asteapta cereri de conexiune
queue_size numar maxim cereri in asteptare

accept

newsock = **accept** (original_socket_descriptor, client_socket_address,
client_addresslen)

Folosit de server: accepta urmatoarea cerere de conectare
newsock capatul la server al noii conexiuni
original_socket_descriptor ramane neschimbat si continua sa primeasca noi
cereri de conectare
client_socket_address adresa clientului
client_addresslen lungimea adresei



connect

connect (client_socket_descriptor, server_socket_address,
server_sockaddress_length)

Folosit de client la conectarea cu serverul
De obicei pe servicii cu conexiune (TCP)
Serverul este implicit la urmatoarele operatii pentru servicii cu conexiune
(send, recv)

send, recv

send (socket_descriptor, data_address, data_length, flags)

recv (socket_descriptor, data_address, data_length, flags)

Folosite pentru transmitere/receptie servicii cu conectare
flags indica optiuni speciale
MSG_OOB – trimite/primește date out-of-band
MSG_PEEK – livrează date primite, dar tratează ca necitite



Exemplu client folosind TCP

```
int main(int argc, char *argv[]) {
    int sockfd, portno, n;    struct sockaddr_in serv_addr;    struct hostent *server;    char buffer[256];

    portno = atoi(argv[2]);
    if( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) error("ERROR opening socket");
    if( (server = gethostbyname(argv[1])) == NULL ) { fprintf(stderr, "ERROR, no such host\n"); exit(0); }
    memset((char *) &serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    memcpy((char *)&serv_addr.sin_addr.s_addr, (char *)server->h_addr, server->h_length);
    serv_addr.sin_port = htons(portno);

    if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) error("ERROR
    connecting");

    //trimit mesajul din buffer
    if( (n = send(sockfd, buffer, strlen(buffer), 0)) < 0) error("ERROR writing to socket");
    //astept raspuns
    memset(buffer, 0, 256);
    if( (n = recv(sockfd, buffer, 255, 0)) < 0) error("ERROR reading from socket");
    printf("[SRV]%s\n", buffer);
    close(sockfd);
    return 0;
}
```

<http://people.cs.uchicago.edu/~mark/51081/labs/LAB6/sock.html>



Exemplu server folosind TCP

```
int main(int argc, char *argv[]) {
    int sockfd, newsockfd, portno, clien; char buffer[256]; struct sockaddr_in serv_addr, cli_addr; int n;

    if( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) error("ERROR opening socket");
    portno = atoi(argv[1]);
    memset((char *) &serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY; // foloseste adresa IP a masinii
    serv_addr.sin_port = htons(portno);

    if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(struct sockaddr)) < 0) error("ERROR on
    binding");
    listen(sockfd, MAX_CLIENTS);
    // accepta o conexiune de la un client
    clien = sizeof(struct sockaddr_in);
    if( (newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clien)) < 0) error("ERROR on
    accept");
    // astept mesaj
    memset(buffer, 0, 256);
    if( (n = recv(newsockfd, buffer, 255, 0)) < 0) error("ERROR reading from socket");
    printf("Here is the message: %s\n", buffer);
    //... trimit raspunsul din buffer
    n = send(newsockfd, buffer, strlen(buffer), 0);
    close(sockfd); close(newsockfd); return 0;
}
```



Exemplu client folosind UDP

```
int main(int argc, char *argv[]) {
    int sd, rc, i; struct sockaddr_in cliAddr, remoteServAddr; struct hostent *server;
    if( (server = gethostbyname(argv[1])) == NULL ) { printf("%s: unknown host '%s'\n", argv[0], argv[1]);
    exit(1); }
    printf("%s: -> '%s' (IP : %s) \n", argv[0], h->h_name, inet_ntoa(*(struct in_addr *)h->h_addr_list[0]));

    remoteServAddr.sin_family = h->h_addrtype;
    memcpy((char *) &remoteServAddr.sin_addr.s_addr, h->h_addr_list[0], h->h_length);
    remoteServAddr.sin_port = htons(REMOTE_SERVER_PORT);

    if( (sd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) { printf("%s: cannot open socket \n", argv[0]); exit(1); }
    cliAddr.sin_family = AF_INET;
    cliAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    cliAddr.sin_port = htons(0); // alegere automata a unui port disponibil

    if( (rc = bind(sd, (struct sockaddr *) &cliAddr, sizeof(cliAddr))) < 0 ) {
        printf("%s: cannot bind port\n", argv[0]); exit(1);
    }
    for(i=2; i<argc; i++) {
        rc = sendto(sd, argv[i], strlen(argv[i])+1, 0, (struct sockaddr *) &remoteServAddr,
        sizeof(remoteServAddr));
        if(rc<0) { printf("%s: cannot send data %d \n", argv[0], i-1); close(sd); exit(1); }
    }
    return 0;}

```



Exemplu server folosind UDP

```
int main(int argc, char *argv[]) {
    int sd, rc, n, cliLen; struct sockaddr_in cliAddr, servAddr; char msg[MAX_MSG];

    if( (sd=socket(PF_INET, SOCK_DGRAM, 0)) < 0 ) { printf("%s: cannot open socket \n", argv[0]); exit(1); }
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY); // orice adresa pt server
    servAddr.sin_port = htons(LOCAL_SERVER_PORT); // portul pentru server

    if( (rc = bind(sd, (struct sockaddr *) &servAddr, sizeof(servAddr))) < 0 ) {
        printf("%s: cannot bind port number %d \n", argv[0], LOCAL_SERVER_PORT); exit(1);
    }
    printf("%s: waiting for data on port UDP %u\n", argv[0], LOCAL_SERVER_PORT);
    while(1) {
        /* server infinite loop */
        memset(msg, 0, MAX_MSG); /* init buffer */
        cliLen = sizeof(cliAddr); /* receive message */
        if( (n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) &cliAddr, &cliLen)) < 0 ) {
            printf("%s: cannot receive data \n", argv[0]); continue;
        }
        printf("%s: from %s:UDP%u : %s \n", argv[0], inet_ntoa(cliAddr.sin_addr), ntohs(cliAddr.sin_port), msg);
    }
    return 0;}

```



Tipuri de servicii oferite de rețea – ISO 8073

Tip rețea	Descriere
A	Rata erori reziduale acceptabilă Rata erori semnalate acceptabilă
B	Rata erori reziduale acceptabilă Rata erori semnalate neacceptabilă
C	Rata erori reziduale neacceptabilă Rata erori semnalate neacceptabilă

Clase protocoale de transport

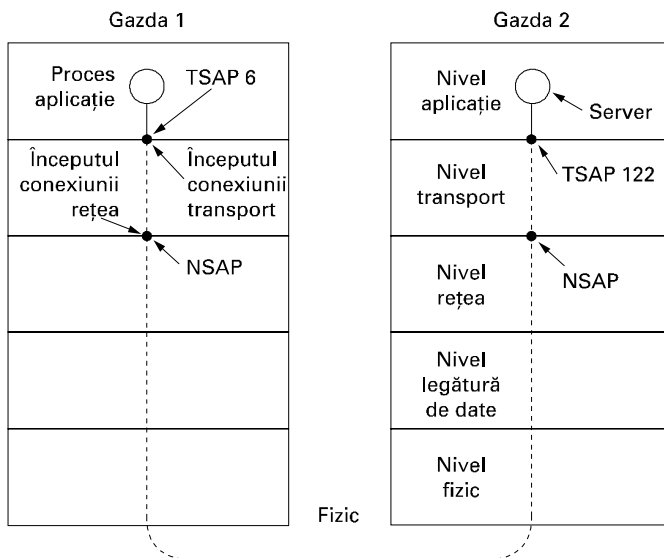
Clasa 0 (cu rețea A):	Clasa simplă
Clasa 1 (cu rețea B):	recuperare erori de bază (N-resets)
Clasa 2 (cu rețea A):	Multiplexare
Clasa 3 (cu rețea B):	Recuperare erori, multiplexare, control flux
Clasa 4 (cu rețea C):	detectie și recuperare erori



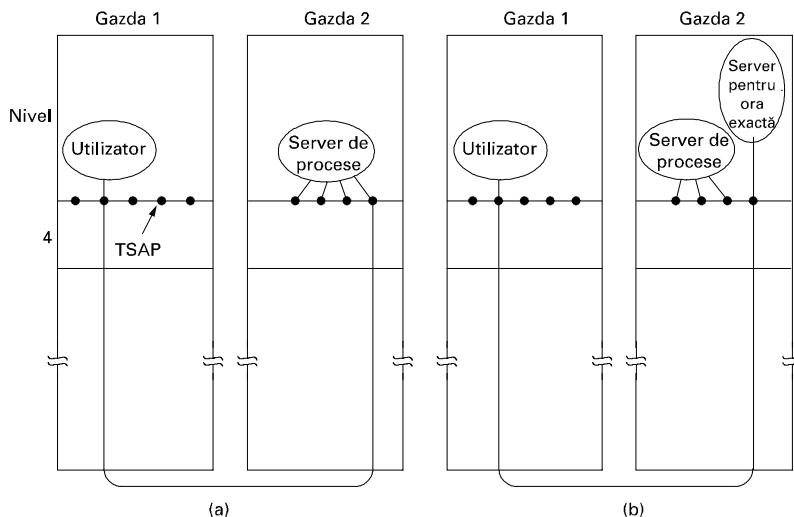
Elementele Protocoalelor de Transport

- Adresare
- Stabilire Conexiune
- Eliberare Conexiune
- Control flux și memorare temporară
- Multiplexare
- Recuperare avarii

Adresare

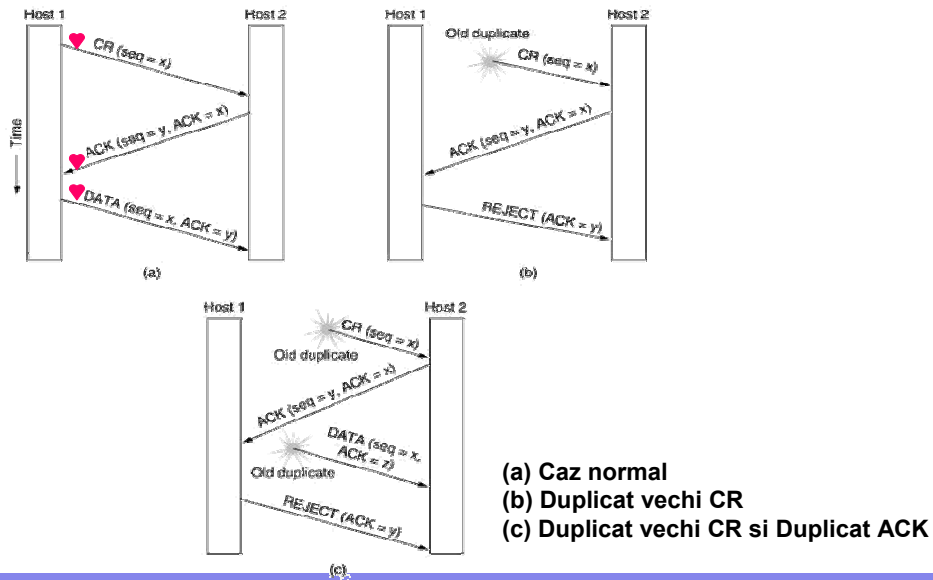


Conectare initiala



Un proces **Utilizator** in Gazda 1 stabileste o conexiune cu un **Server pentru ora exactă** in Gazda 2, cu ajutorul unui **Server de procese**.

Stabilirea conexiunii - Three way handshaking

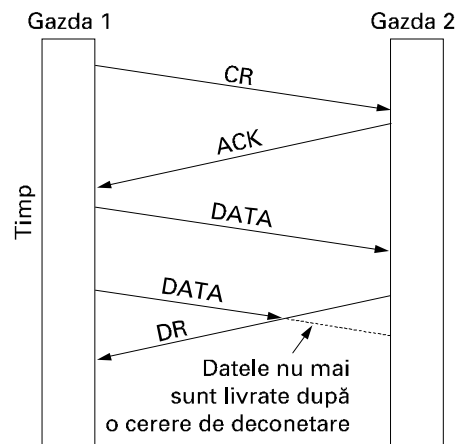


02.05.2008

Protocoloale de comunicație - Curs 9

21

Deconectare abruptă cu pierdere de date

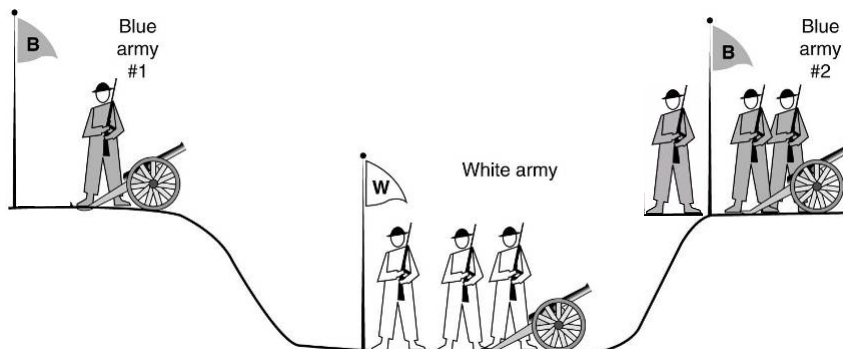


02.05.2008

Protocoloale de comunicație - Curs 9

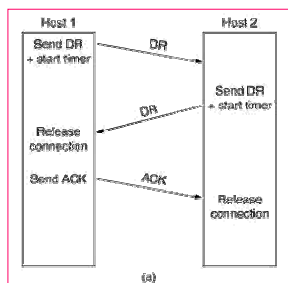
22

Problema celor doua armate

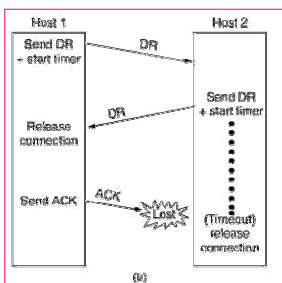


Patru scenarii de eliberarea conexiunii

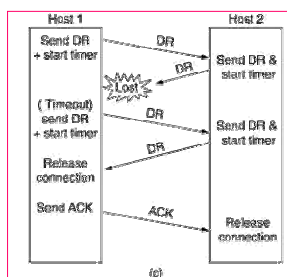
normal



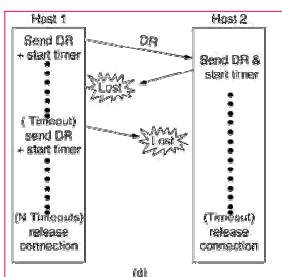
ACK final pierdut



raspuns pierdut



Raspuns si DR pierdute





Control flux si memorare tampon

A	Mesajul	B	Comentarii
1 →	< cere 8 tamponane>	→	A cere 8 tamponane
2 ←	<ack=15, buf=4>	←	B îi acordă tamponane numai de la 0 la 3
3 →	<seq = 0, data = m0>	→	A mai are 3 tamponane libere
4 →	<seq = 1, data = m1>	→	A mai are 2 tamponane libere
5 →	<seq = 2, data = m2>	...	Mesaj pierdut, dar A crede că mai are un singur tampon liber
6 ←	<ack = 1, buf = 3>	←	B confirmă 0 și 1 și permite 2-4
7 →	<seq = 3, data = m3>	→	A mai are tamponane
8 →	<seq = 4, data = m4>	→	A nu mai are tamponane libere și trebuie să se oprească
9 →	<seq = 2, data = m2>	→	A retransmite la expirarea intervalului de timp
10 ←	<ack = 4, buf = 0>	←	Toate mesajele sunt confirmate, dar A este în continuare blocat
11 ←	<ack = 4, buf = 1>	←	A poate să îl trimită acum pe 5
12 ←	<ack = 4, buf = 2>	←	B a mai găsit un tampon
13 →	<seq = 5, data = m5>	→	A mai are un tampon liber
14 →	<seq = 6, data = m6>	→	A este blocat din nou
15 ←	<ack = 6, buf = 0>	←	A este blocat în continuare
16 ...	<ack = 6, buf = 4>	←	Posibilă interblocare



TCP - Transmission Control Protocol

- *Cel mai folosit protocol de transport*
- Livrare sigura pe retea nesigura (datagrame)

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

Cateva porturi

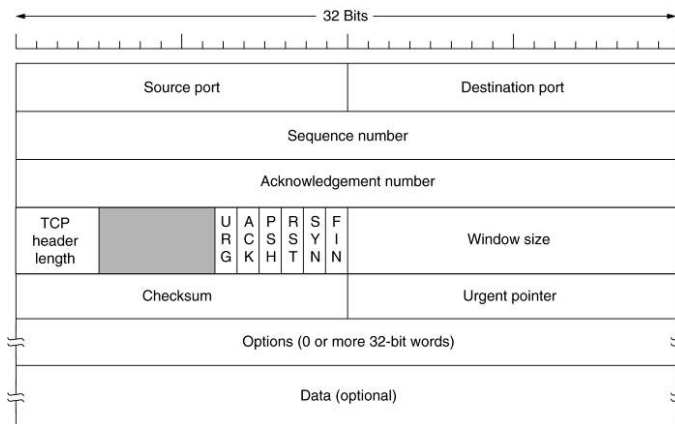


Caracteristici

- Orientat pe conexiune
- Punct la punct
- Sigur
- Full duplex
- Interfata flux (Stream)
- Three-way handshake
- Eliberare lina a conexiunii



Antet segment TCP



URG Urgent pointer valid

SYN - open connection

ACK Acknowledge Number valid

FIN - close a connection

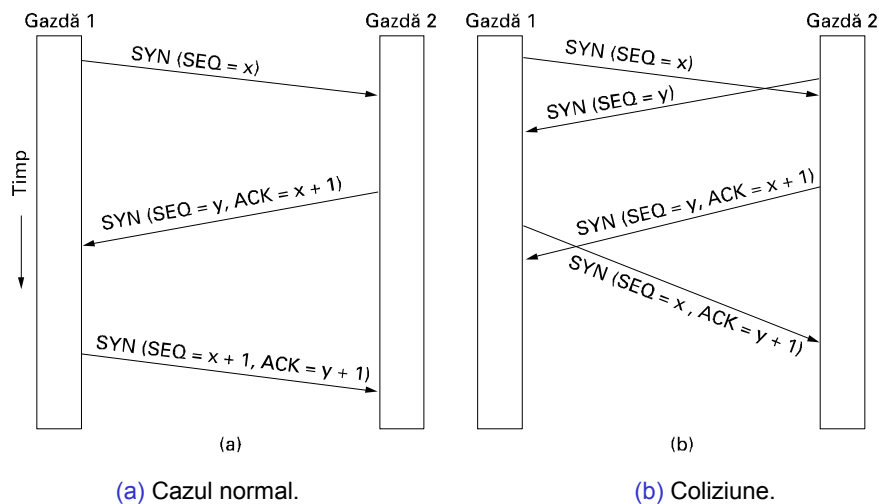
PSH - push information to user

Options: e.g. max TCP payload, selective repeat

RST - close a connection due to an error



Stabilire Conexiune TCP



Management Conexiune TCP

Stare	Descriere
CLOSED (ÎNCHIS)	Nici o conexiune nu este activă sau în așteptare
LISTEN (ASCULTARE)	Serverul așteaptă recepționarea unui apel
SYN RCVD (Recepție SYN)	S-a recepționat o cerere de conexiune; aștept ACK
SYN SENT (Transmisie SYN)	Aplicația a început deschiderea unei conexiuni
ESTABLISHED (STABILIT)	Starea normală de transfer a datelor
FIN WAIT 1 (Așteptare FIN 1)	Aplicația a anunțat că termină
FIN WAIT 2 (Așteptare FIN 2)	Partenerul este de acord cu eliberarea conexiunii
TIMED WAIT (Așteptare Temporizată)	Se așteaptă „moartea” tuturor pachetelor
CLOSING (În curs de ÎNCHIDERE)	Ambele părți încearcă simultan închiderea
CLOSE WAIT (ÎNCHIDE și AȘTEAPTĂ)	Partenerul a inițiat eliberarea conexiunii
LAST ACK (CONFIRMARE FINALĂ)	Se așteaptă „moartea” tuturor pachetelor

Stările folosite în managementul conexiunii TCP



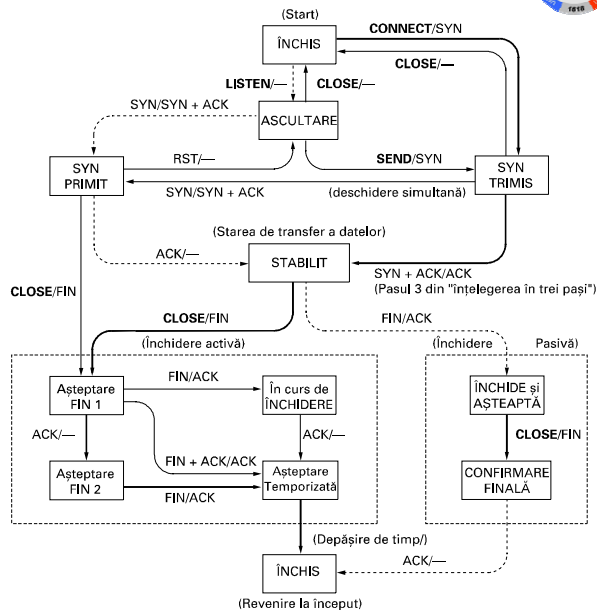
Stabilirea și desființarea Conexiunilor TCP

• Linie groasa continua = cale normala pentru client

• Linie groasa intrerupta = cale normala server

• Linii subtiri = evenimente exceptionale

• Fiecare tranzitie are eticheta eveniment/actiune



02.05.2008

Protocoloale de comunicație - Curs 9

31



Corectitudinea segmentelor TCP

1. Suma de control (checksum) din antet - include

antet + incarcatura segment TCP

pseudo-antet (vezi figura)

adresele IP sursa si destinatie (separarea nivelelor?!)

protocolul (6 pentru TCP)

lungime segment TCP (include antetul)

Algoritm: la transmisie aduna cuvinte de 16 biti in complement fata de 1

complementeaza rezultatul

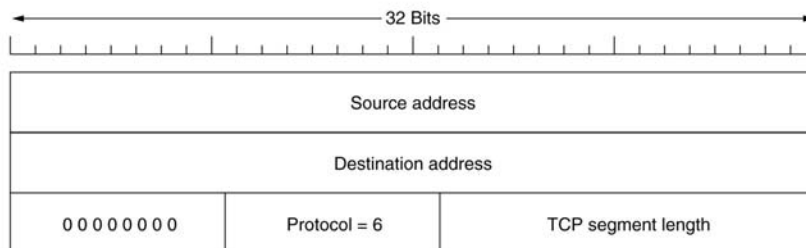
scrie rezultatul in antet

la receptie

aduna cuvinte de 16 biti – rezultatul trebuie sa fie zero

2. Acknowledgement number

urmatorul octet asteptat



02.05.2008

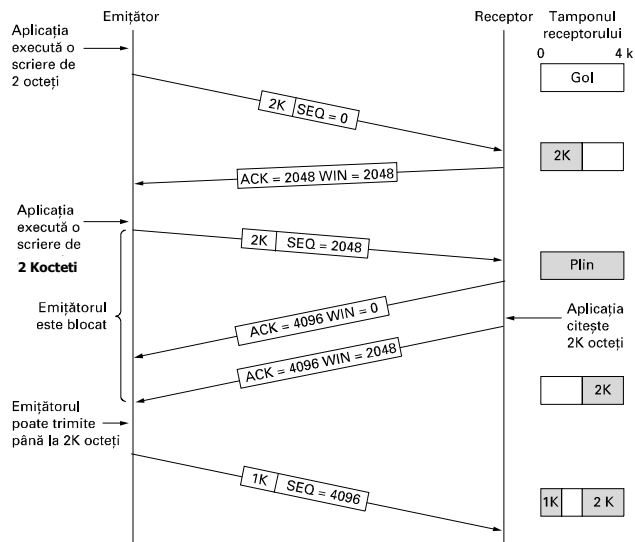
Protocoloale de comunicație - Curs 9

32



Controlul fluxului de date

Receptorul specifica fereastra de receptie



Un emitor blocat poate trimite:

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



Probleme dimensiuni campuri antet

Numere secventa de 32-biti

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

Problema: cum se diferentiaza segmentele cu acelasi numar de secventa?

Fereastra receptor

Transmitere 500 Kb pe legatura 1 Gbps ocupa 500 μ sec

La intarziere 20 ms confirmarea se primeste dupa 40 ms => ocupare canal 1.25%

Ocupare completa in ambele directii: produs $\text{bandwidth} \times \text{delay} = 40$ milioane biti

→ fereastra receptor $\geq$ acest produs

Problema: cum se poate mari dimensiunea ferestrei?

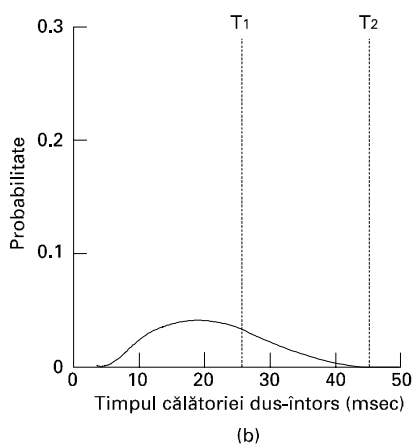
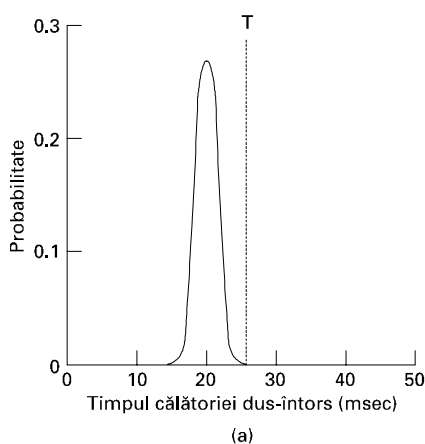
Solutii

- Folosire optiuni TCP (RFC 1323)
 - TCP Timestamps – rezolva numere de secventa duplicate
 - Window Scale – factor de scalare a campului Window size cu pana la 2^{14} pozitii binare → ferestre de pana la 2^{30} octeti

Controlul congestiei

- Fluxul de date transmis pe o conex. TCP limitat de minimul dintre:
 - dimensiunea fereastră receptorului
 - capacitatea rețelei (**fereastră de congestie**)
- Algoritm de stabilire fereastră de congestie
 - transmite un segment de dimensiune maxima pe conexiunea stabilita
 - se dubleaza dimensiunea (rafala de segmente) la fiecare transmisie confirmata la timp
 - la primul timeout se opreste procedeeul si fereastră ramane la valoarea ultimei transmisii confirmate la timp (fara timeout)
- Algoritmul de control al congestiei
 - foloseste un **prag** (threshold)
 - la un timeout pragul setat la jumătate din fereastră 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 maxim o data)

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.



Stabilire time-out

- Setare proasta – performante slabe:
 - Prea lung – transmitatorul asteapta mult ptr retransmisie
 - Prea scurt – trafic inutil generat de transmitator
- Timeout diferit la fiecare conexiune - setat dinamic
- Transmitatorul alege *retransmission timeout* (RTO) pe baza *round trip time* (RTT)

M este timpul masurat pana la primirea ack

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

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

- Alegere dupa *deviatia standard* (DS);

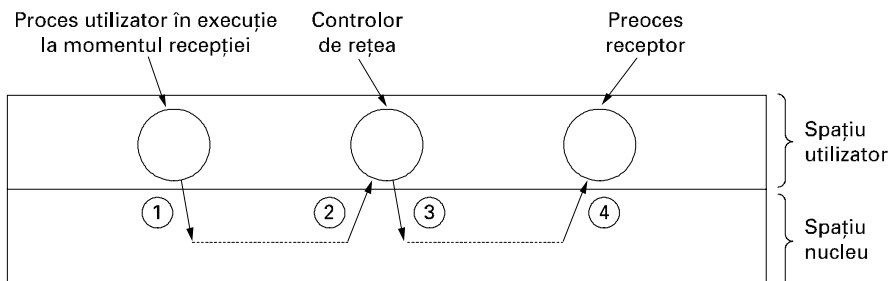
D aproximeaza DS

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

$$RTO = RTT + 4 * D$$



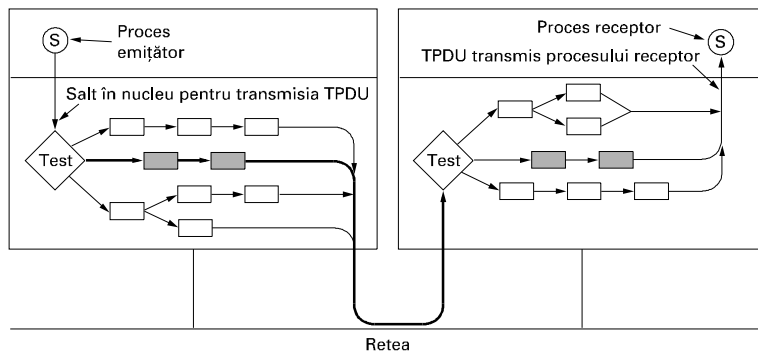
Proiectarea pentru performanta



Patru schimbări de context pentru a manevra un pachet cu un manager de rețea în spațiul utilizatorului.



Prelucrare Fast TPDU



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

Test caz normal: starea = ESTABLISHED,

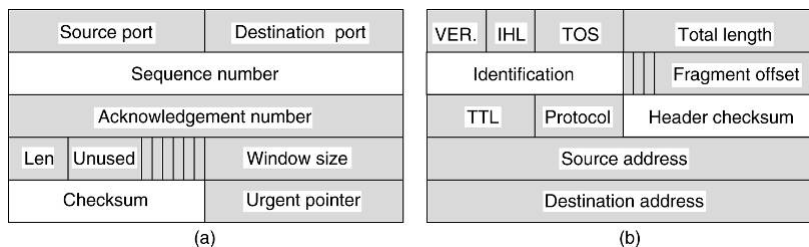
nu se incearca inchiderea conexiunii,

TPDU normal, suficient spatiu la receptor

→ alege **fast path**



Prelucrare Fast TPDU (2)



(a) Antet TCP. (b) Antet IP.

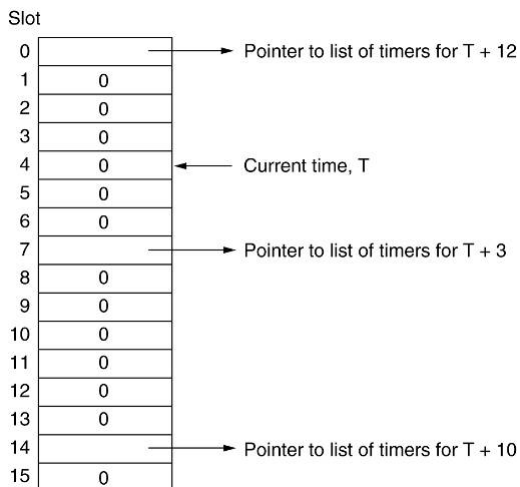
In ambele cazuri, campurile gri sunt luate din **prototip** fara modificari.

Receptor

- Localizeaza inregistrarea conexiunii din TPDU intr-o tabela hash
- Testeaza pentru cazul normal (similar cu transmisia)
- Actualizeaza inregistrarea conexiunii
- Copiază datele la utilizator și calculează suma de control
- Transmite confirmarea



Prelucrare Fast TPDU (timer management)



“Timing wheel.”

1 slot = 1 clock tick

Time 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

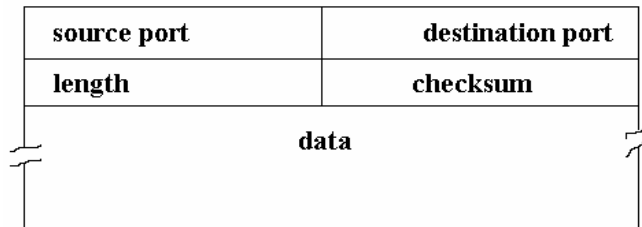
Daca slot nevid, proceseaza toate evenimentele



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 nefolosit (calculat la fel ca la TCP)

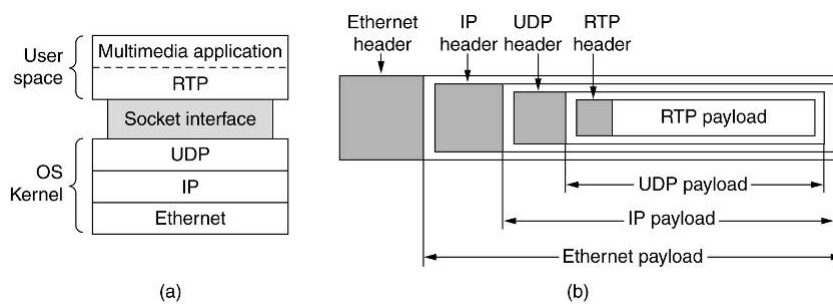
Nu control flux

Nu control erori

Nu retransmisie

Utilizat in aplicatii client-server (DNS)

The Real-Time Transport Protocol



(a) Pozitia RTP in stiva de protocoale.

(b) Format pachet.

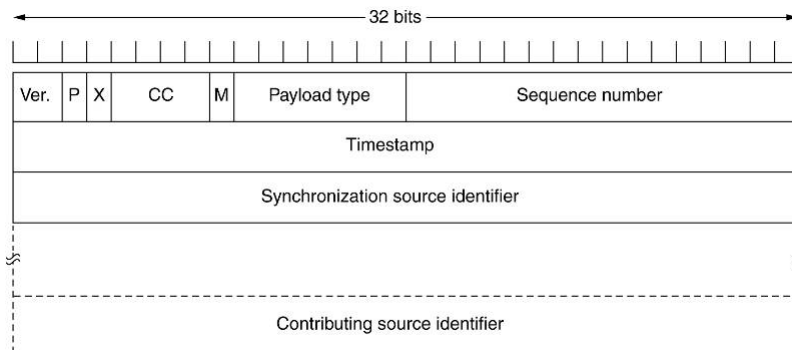
Funcție principală: multiplexare fluxuri RTP in flux UDP

Unicast sau multicast

Fara retransmitere – receptorul “interpoleaza” packetele anse



Antet RTP



P – pachet extins la multiplu de 4 octeti

X – antet extins

CC – no. surse multiplexate

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

Payload type – e.g. MP3

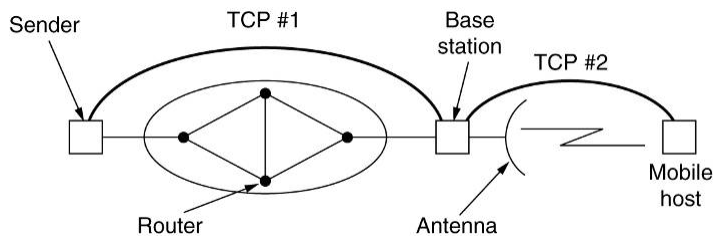
Timestamp – timpul primei probe din pachet relativ la inceputul fluxului (receptorul sa foloseasca proba la timpul potrivit)

Synchro – identifica fluxul (pentru multiplexare de-multiplexare)

Contrib – lista fluxurilor mixate



TCP si UDP fara fir



Congestie:

Retea cu fir – incetineste transmitia

Retea fara fir – accelereaza

Indirect TCP:

Imparte conexiunea TCP in doua

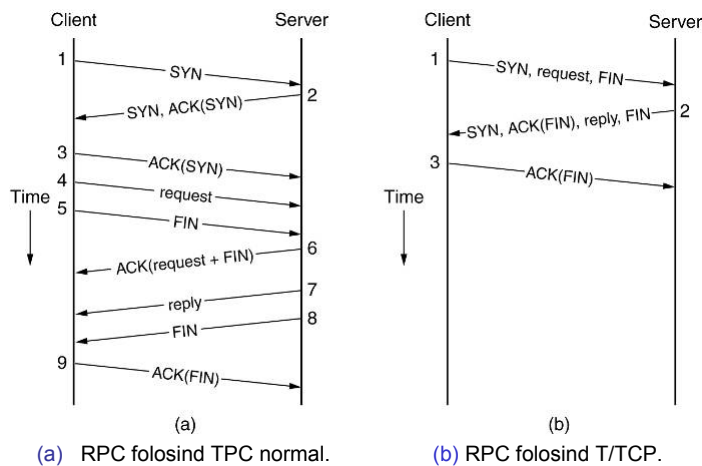
Violeaza semantica TCP – implica *Base station* in TCP

Modifica nivelul retea in Base station:

“Snooping agent”: - retransmite segmente fara stirea sursei
- cere retransmisii din gazde mobile



TCP Tranzactional



- Permite transfer de date la “connection setup”



Sumar

- Serviciile, primitivele de serviciu, diagrama de conectare/deconectare
- Socket API – Exemple client-server TCP si UDP
- Elementele Protocoloalelor de Transport
 - Adresare, Stabilire Conexiune, Eliberare Conexiune, Control flux si memorare temporara, Multiplexare, Recuperare avarii
- TCP - Transmission Control Protocol
- Management Conexiune TCP
- Controlul congestiei
- Gestiunea ceasurilor in TCP
- Proiectarea pentru performanta - Fast TPDU
- UDP - User Datagram Protocol
- Real-Time Transport Protocol
- TCP si UDP fara fir
- TCP Tranzactional