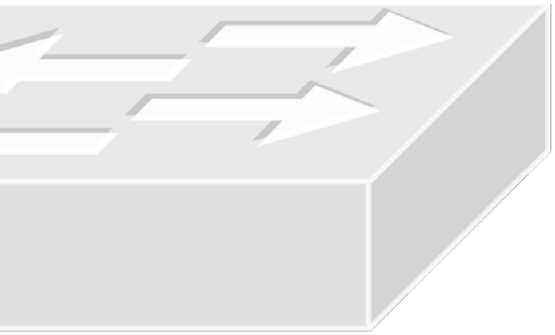
Reţele Locale de Calculatoare

TCP – Transmission Control Protocol



– curs 6 –

09.11.2009

11.11.2009

Universitatea POLITEHNICA Bucureşti

Nivelul Transport

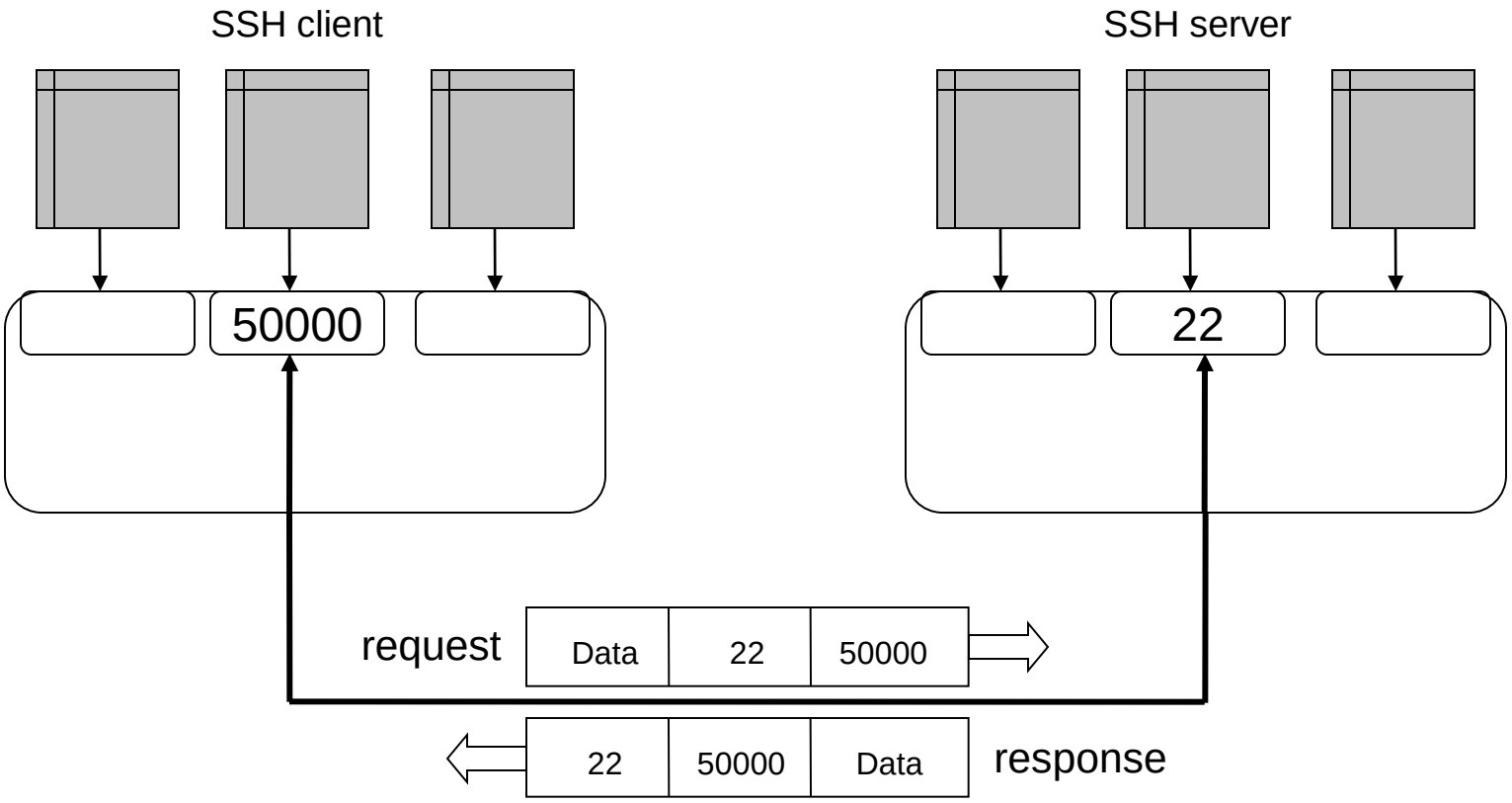
- Oferă servicii nivelului Sesiune
- Primește servicii de la nivelul Rețea

- Roluri
  - împărțirea datelor în **segmente**
  - crearea de **conexiuni**
  - un nou mecanism de adresare (**porturi**)
  - **controlul fluxului** (controlul congestiei)
  - siguranța transmisiei (**reliability**)

Nivelul Transport (cont.)

- Comunicație între procese
  - process-to-process delivery
  - nivelul Rețea - host-to-host delivery (comunicație între stații)
- Modelul client-server
  - server
    - proces “pasiv”
    - “ascultă” cereri de la clienți
  - client
    - procesul activ
    - inițiază o conexiune către server
    - solicită un anumit serviciu
  - adresare prin porturi
    - un proces server (un serviciu) = un port “listening”

Modelul client-server



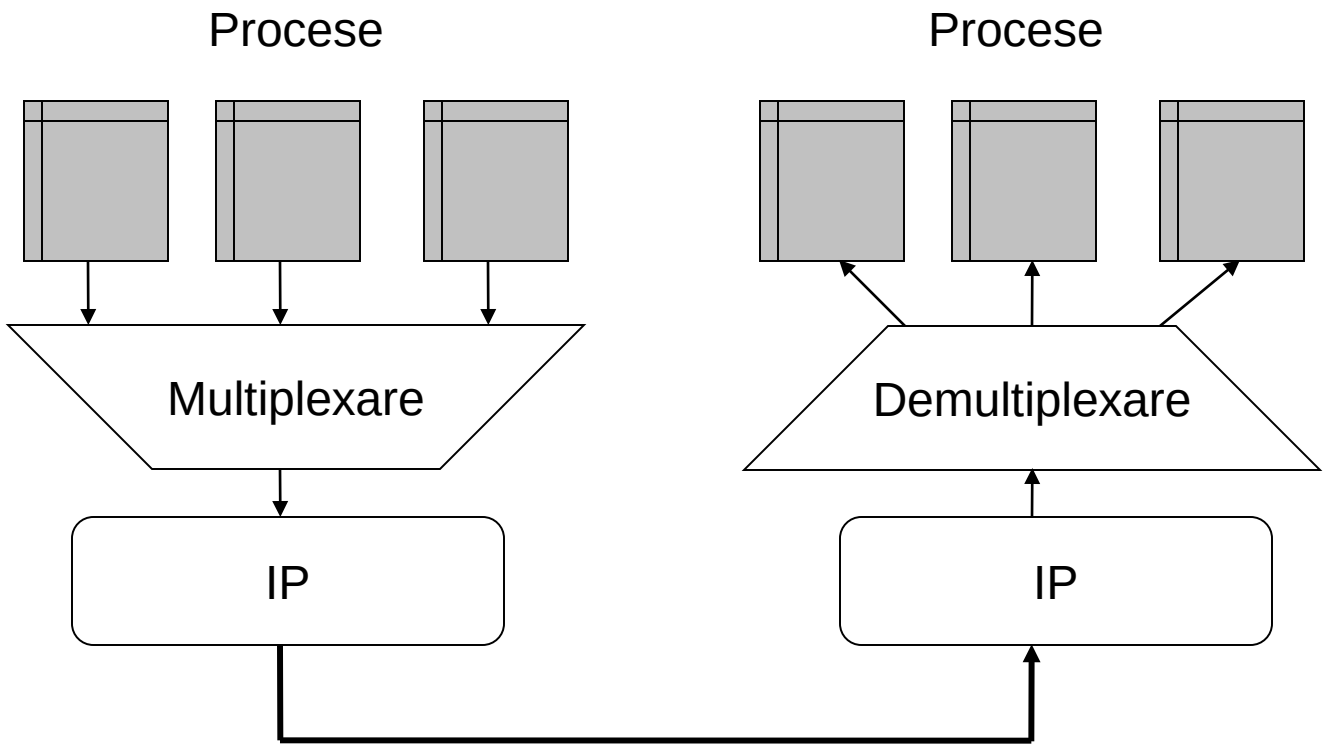
Porturi

- Sistemul de adresare folosit de nivelul Transport
- Asociat protocolului de nivel Transport
  - Port 100 UDP != Port 100 TCP

- Existența mai multor procese pe aceeași stație
  - mai multe servicii pe aceleși sistem
  - **multiplexare prin porturi**

- 16 biti → valori de la 0 la 65535
- Intervale de porturi (IANA)
  - Porturi rezervate (well-known): între 0 și 1023
    - SSH – 22, FTP – 21, Telnet – 23, SMTP – 25, HTTP – 80
  - Porturi înregistrate: între 1024 și 49151
    - Kazaa, RMI Registry, MySQL, etc.
  - Porturi dinamice (efemere): de la 49152 la 65535
    - testare locală

Porturi (cont.)



- O pereche formată dintr-o adresa IP si un port – socket
  - (211.42.121.13, 50000)

Socketi de rețea (Internet sockets)

- "The combination of an IP address and a port number is referred to as a socket." (Cisco)

- Datagram sockets: UDP

```
sockfd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

- Stream socket: TCP

```
sockfd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

- Asocierea socketului la o adresă şi un port (bind)

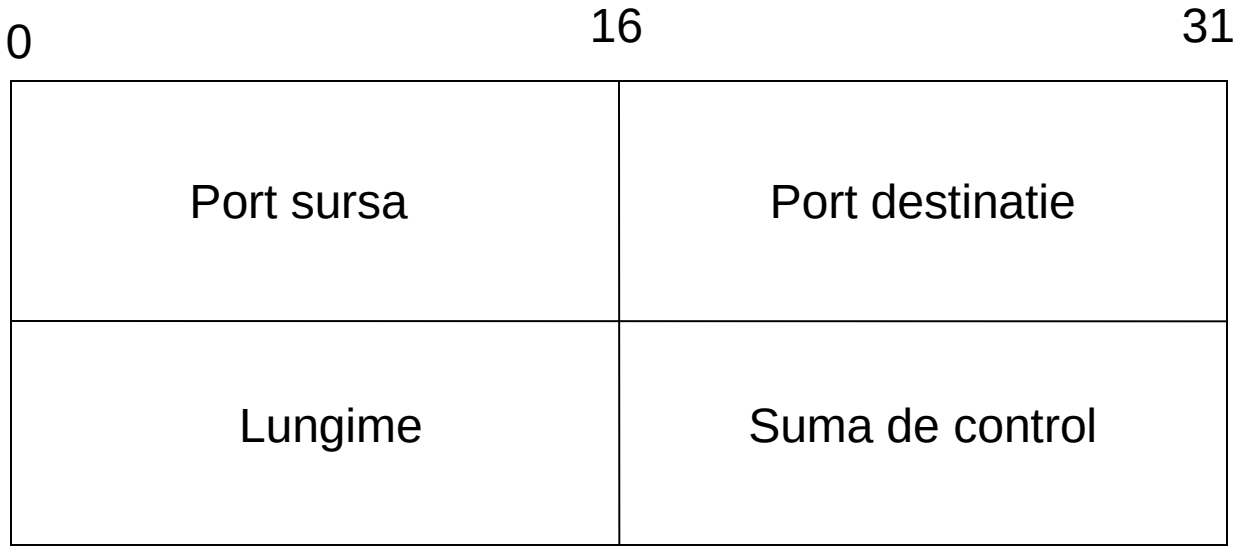
```
addr.sin_family = AF_INET;
addr.sin_port = htons(50000);
addr.sin_addr.s_addr = INADDR_ANY;
bind(sockfd, (struct sockaddr *) &addr, sizeof (addr));
```

UDP

- User Datagram Protocol
- Neorientat conexiune
- Nesigur (unreliable) (segmente pierdute)
- Fara controlul fluxului (segmente fără ordine)

- Când se folosește UDP?
  - overhead mare indus de TCP
    - DNS, managementul rețelei (SNMP)
  - comunicații multimedia
    - controlul fluxului nu este foarte important
  - aplicația asigură controlul fluxului
  - rețele locale
    - “Brood pe UDP” :-)

Antetul UDP



Socketi UDP

- nu are sens folosirea connect (dar se poate)
  - nu se creează un canal virtual de comunicație”

- recvfrom și sendto

```
sendto(sockfd, buffer, BUF_SIZE, 0, (struct
    sockaddr*)&target_host_address, sizeof(struct sockaddr));

recvfrom(s, buffer, BUF_SIZE, 0, (struct sockaddr*)&host_address,
    &hst_addr_size);
```

TCP

- Transmission Control Protocol
- Orientat conexiune
  - circuit virtual în care are loc comunicația
- Protocol sigur (reliable)
  - datele ajung garantat la destinatie
  - datele ajung în ordine la destinație
    - numere de secventa si numere de confirmare
- Controlul fluxului
  - corelare sender și receiver
    - fereastră glisantă
- Controlul congestiei
- Controlul erorii
  - sumă de control

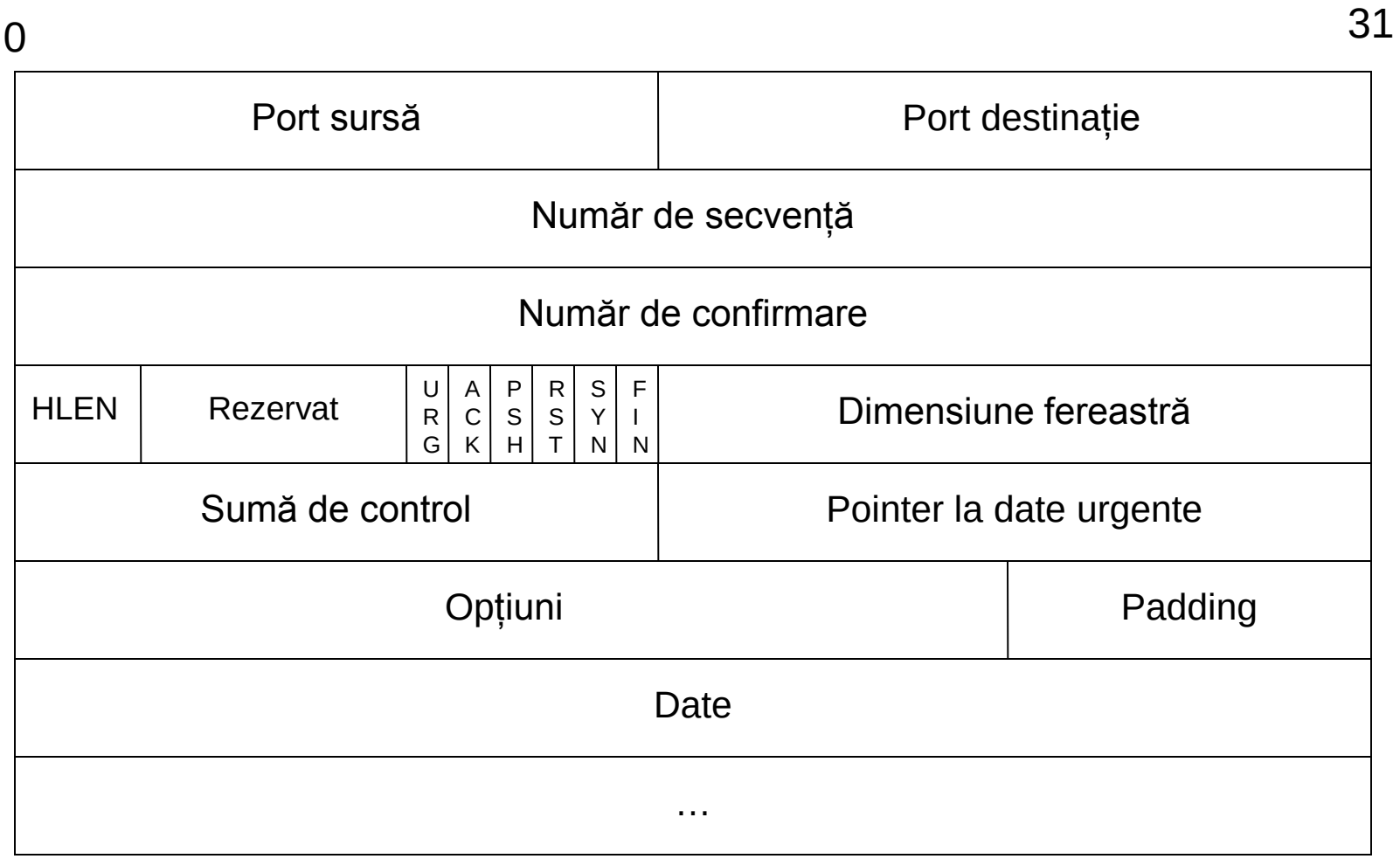
TCP (cont.)

– Transmisie de tip **flux de octeti** (**byte stream**)

– Folosire de timere
  - RTT – Round Trip Time
  - keep-alive timer

– TCP este folosit in 95% din comunicatiile din Internet
  - HTTP, FTP, SMTP, POP3, IMAP, SSH etc.

Antetul TCP - imagine



Antetul TCP – porturi

- Multiplexare prin porturi
  - process-to-process delivery
  - pot exista mai multe circuite virtuale între două stații
- Flux de comunicație (o conexiune) TCP
  - *<adresă IP sursă, port sursă, adresa IP destinație, port destinație>*
- Substituție port sursă – port destinație în pachetele de răspuns

Antetul TCP – numere de secvență

- Reprezentare pe 32 de biti
- Număr de secvență
  - indexul primului octet din segmentul TCP
  - în cadrul fiecărui segment
  - situație:
    - primul octet are numarul de secvență 1000
    - cel de-al 100-lea octet are numărul de secvență 1099
- Număr de confirmare
  - indexul urmatorului octet pe care receptorul se așteaptă să-l primească de la transmițător
  - confirmarea primirii datelor de pana la acest numar
  - nu este prezent în toate segmentele
    - activat de prezența câmpului (flag-ului) ACK

Antet TCP - Campuri de control

- Grup de 8 biți din antetul TCP
- Identifică diverse stări ale protocolului
- Mai mulți biți pot fi activi simultan

- **URG**
  - activare câmp “Pointer la date urgente”
    - offset până la ultimul octet de “date urgente”

- **PSH**
  - push function
  - pentru eficiență TCP folosește buffere de intrare și ieșire
  - golirea bufferelor – livrare imediată
    - transmiterea secvenței “login:” în rețea

Antet TCP – Campuri de control

- **RST**
  - resetarea conexiunii
  - invalidarea numerelor de secvență
- **ACK**
  - activare câmp “Număr de confirmare”
- **SYN**
  - protocolul de inițiere a conexiunii (handshake)
  - stabilirea/sincronizarea numerelor de secvență
- **FIN**
  - protocolul de încheiere a conexiunii
  - încheierea transmisiei de la FIN-sender

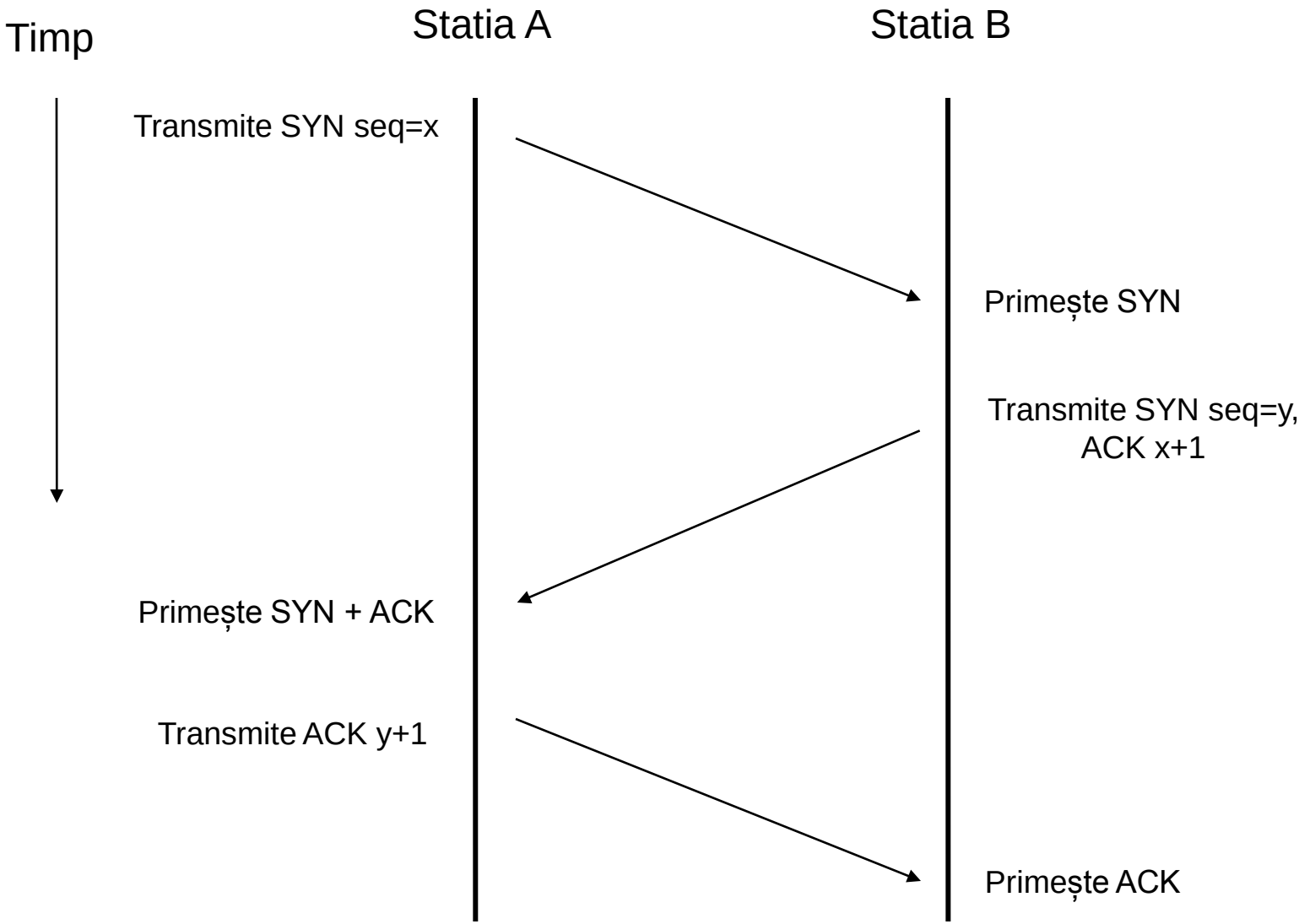
Antet TCP – Campuri de control (cont.)

- RFC 3168
  - introducerea câmpurilor CWR și ECE
- ECE
  - ECN Echo
- CWR
  - Congestion Window Reduced
  - s-a primit un segment cu ECE activat

Antet TCP – alte campuri

- **HLEN (Header Length)**
  - lungimea antetului TCP în cuvinte de 32 de octeți
  - maxim 15 (60 de octeti) –> 40 de octeți pentru opțiuni
- **Dimensiune fereastră**
  - spațiul pentru stocare date neconfirmate (receiver)
  - maxim 65535
  - opțiune de scalare a ferestrei
- **Sumă de control** (antet + date)
- **Opțiuni**
  - diverse opțiuni/extensii definite în RFC
    - specificarea MSS (Maximum Segment Size)
    - window scale

Inițierea conexiunii



Inițierea conexiunii (cont.)

- Clientul este entitatea activă – inițiază conexiunea

- Câmpul SYN activat
- ISN – Initial Sequence Number
  - numărul de secvență dintr-un segment cu SYN activat

- Protocolul de inițiere de conexiune - **3-way handshake**
- Primul pachet (SYN)
  - stabilirea ISN pentru comunicația de la client la server
- Al doilea pachet (SYN+ACK)
  - confirmarea primului pachet
  - stabilirea ISN pentru comunicația de la server la client
- Al treilea pachet (ACK)
  - confirmarea celui de-al doilea pachet
- Cele două ISN sunt generate aleator

Inițierea conexiunii - socketi

- Serverul este în starea listening (serv_sockfd)

```
listen(serv_sockfd, 5);
conn_sockfd = accept(serv_sockfd, (struct sockaddr *)
  &cli_addr, &cli_len);
```

- Un nou socket pentru intermedierea comunicației cu un clientul (conn_sockfd)
  - aceleași caracteristici cu socketul listener (IP, port)
  - demultiplexat pe baza peer-ului

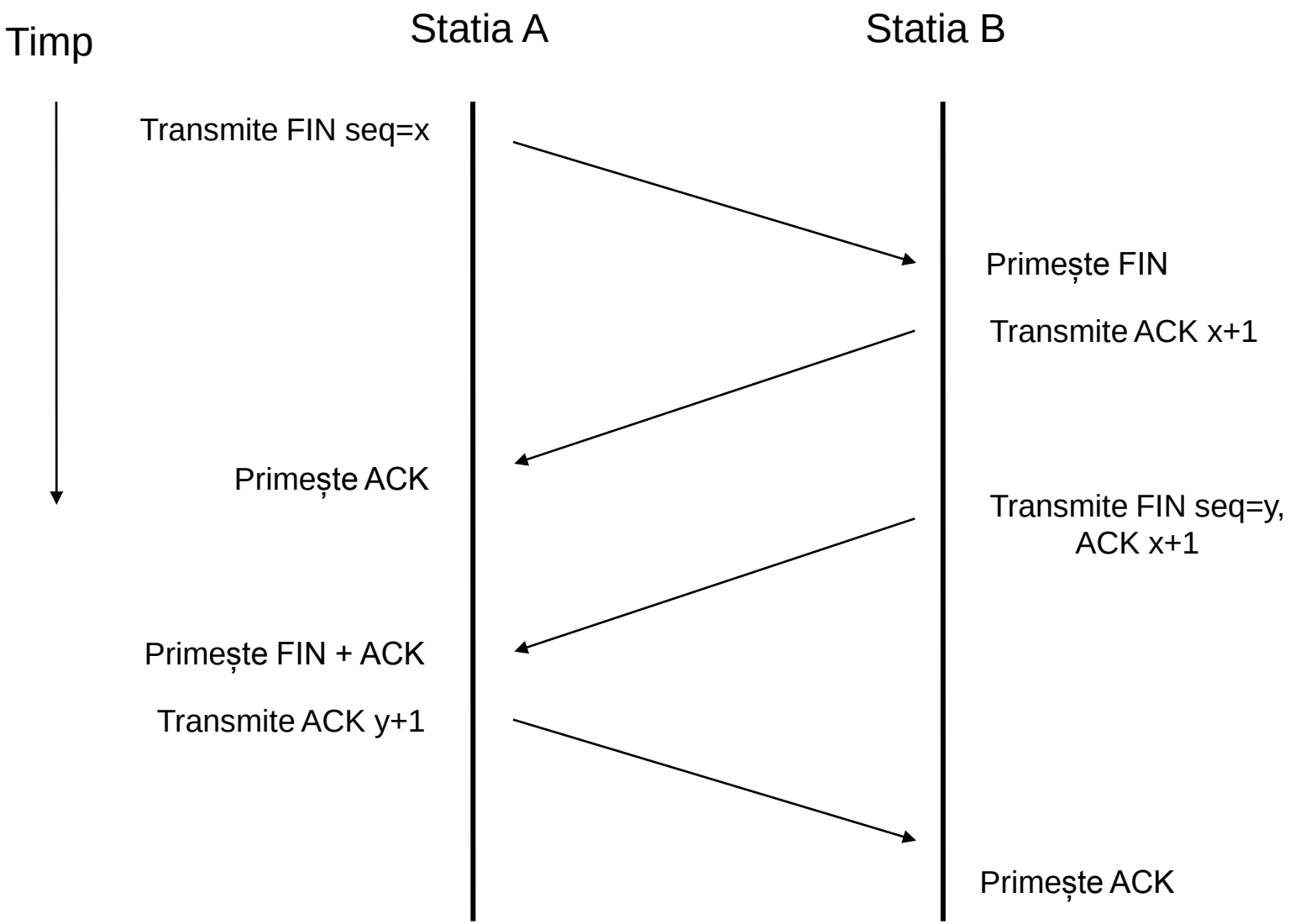
- Clientul inițiază conexiunea

```
connect (cli_sockfd, (struct sockaddr *) &serv_addr, sizeof (serv_addr));
```

Full duplex

- De ce sunt necesare două numere de secvență?
  - comunicația este full duplex
- O conexiune TCP - două canale virtuale de comunicatie
  - client → server
  - server → client
- Un socket - două buffere
  - buffer de citire/recepție
  - buffer de scriere/transmitere
  - SO_RCVBUF, SO_SNDBUF
- Este posibilă comunicație half-duplex prin închiderea unui capăt al conexiunii
  - Care capăt se închide? De scriere sau de citire?

Incheierea conexiunii



Incheierea conexiunii (cont.)

- Inițiat de oricare capăt al transmisiei
- Câmpul FIN activat

- Protocol de tipul **4-way handshake**
  - primul segment
    - câmpul FIN activ
  - al doilea segment este o confirmare a primului
  - conexiunea este pe jumătate închisă (HALF CLOSED)
    - comunicatia este intr-un singur sens
  - urmatoarele doua segmente închid conexiunea în celălalt sens

- Este posibil protocol de tipul 3-way handshake
  - cele doua entități închid conexiunea în același timp
  - al doilea și al treilea segment sunt “unite”

Încheierea conexiunii - socketi

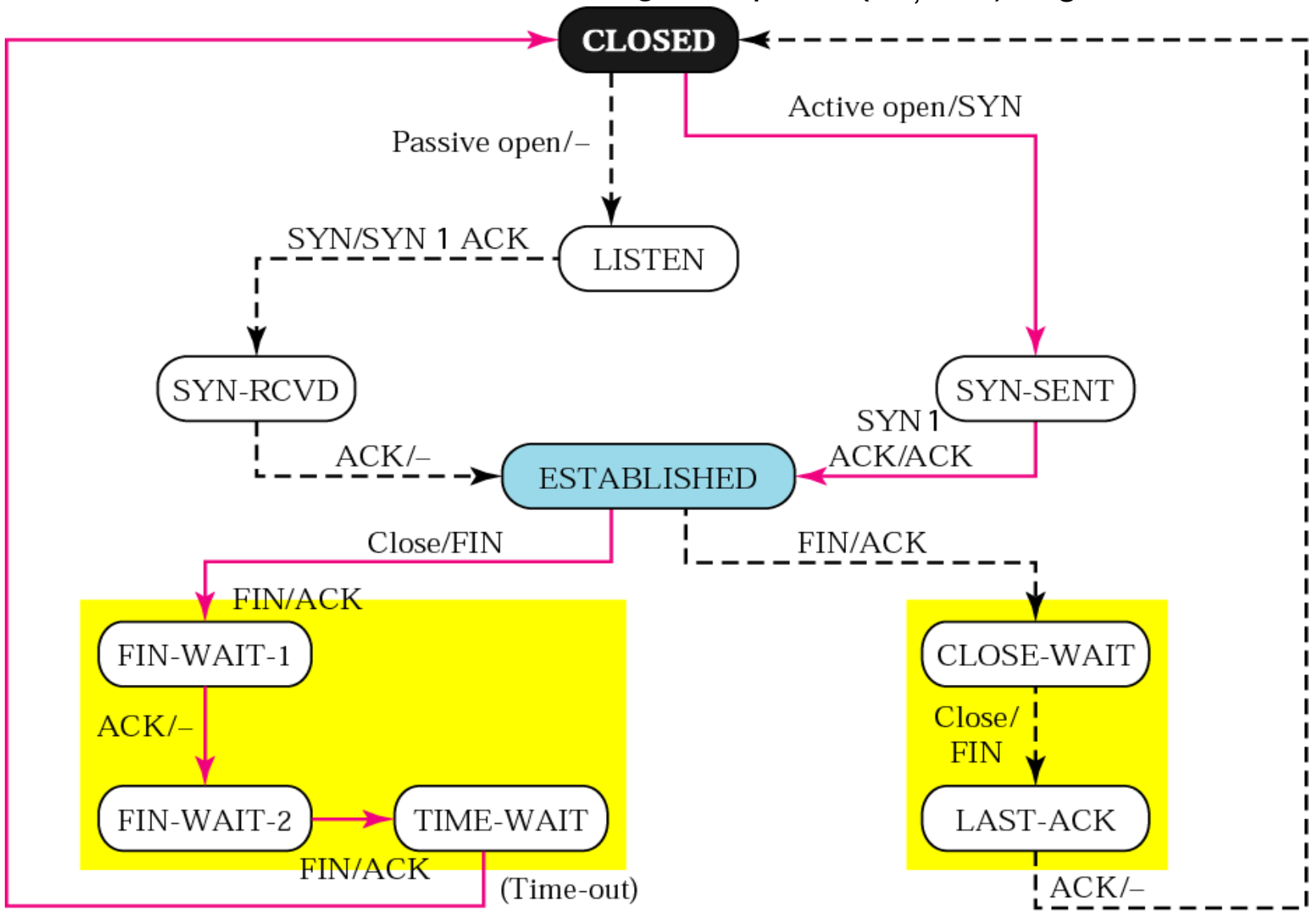
- Un capăt închide conexiunea (fie acesta clientul)

```
close (cli_sockfd);
```

- Celălalt capăt așteaptă sosirea de cereri
  - Poate solicita, de asemenea, închiderea conexiunii

```
n = read (conn_sockfd, buffer, BUF_LEN);
if (n == 0) { /* se inchide conexiunea */
    printf (“clientul %s a incheiat conexiunea\n”,
  inet_ntoa (cli_addr.sin_addr));
    close (conn_sockfd);
}
...
```

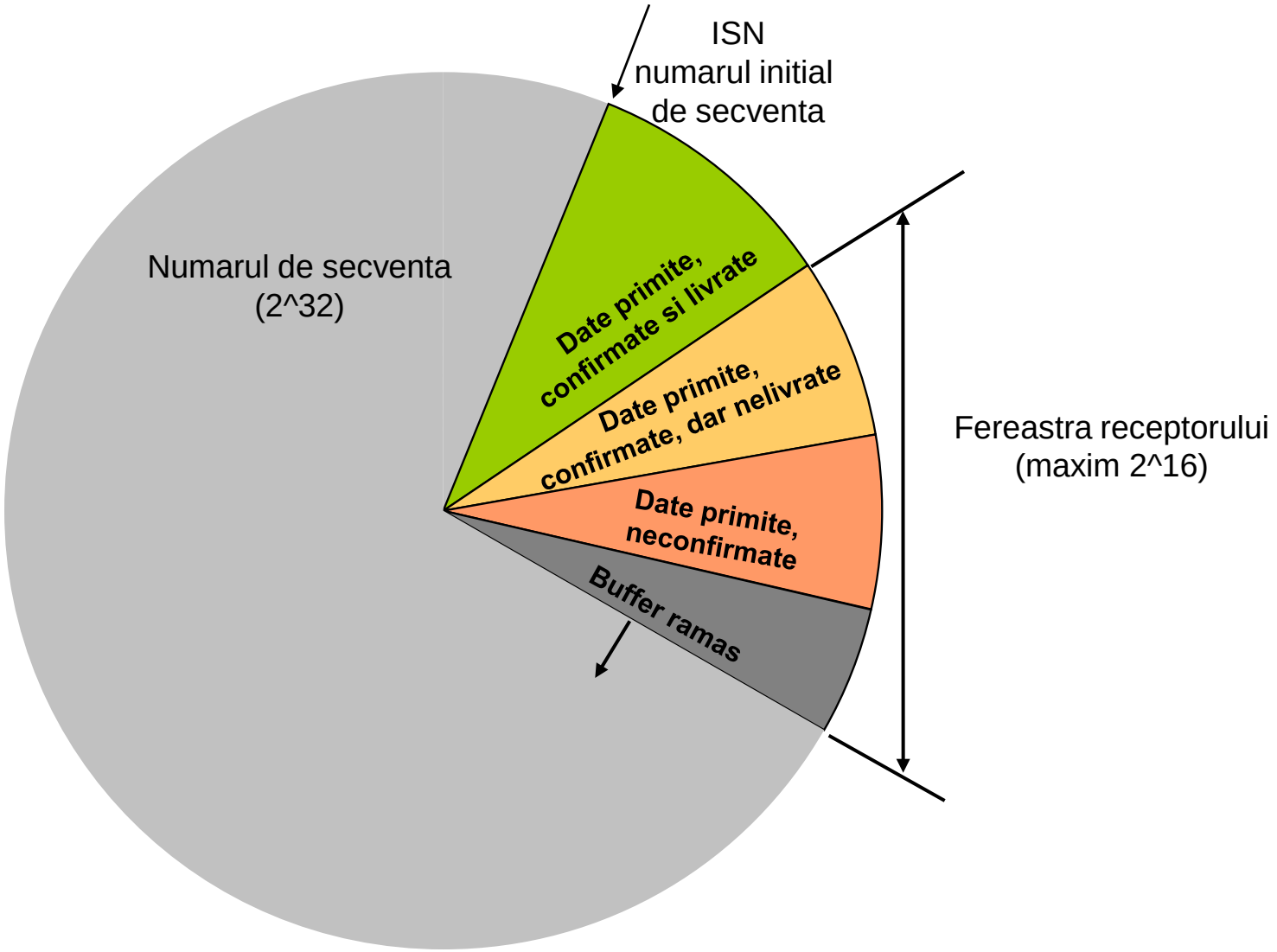
Diagrama de stări



Transmiterea de date

- După inţierea conexiunii
- Receptorul controlează transmisia (controlul fluxului)
- Transmitere date
  - exista date de transmis
  - nu se va depăși dimensiunea ferestrei anunțată de receiver
- Segment de confirmare pentru fiecare pachet de date
  - câmpul ACK activat
  - dimensiunea ferestrei receptorul (câți octeți poate primi)
- Receptorul confirma cel mai curent spatiu contiguu de date primit
  - unele date se pot pierde
  - pot sosi duplicate
- Timere pentru evitarea deadlock-urilor și a conexiunilor care nu mai răspund

Fereastră glisantă



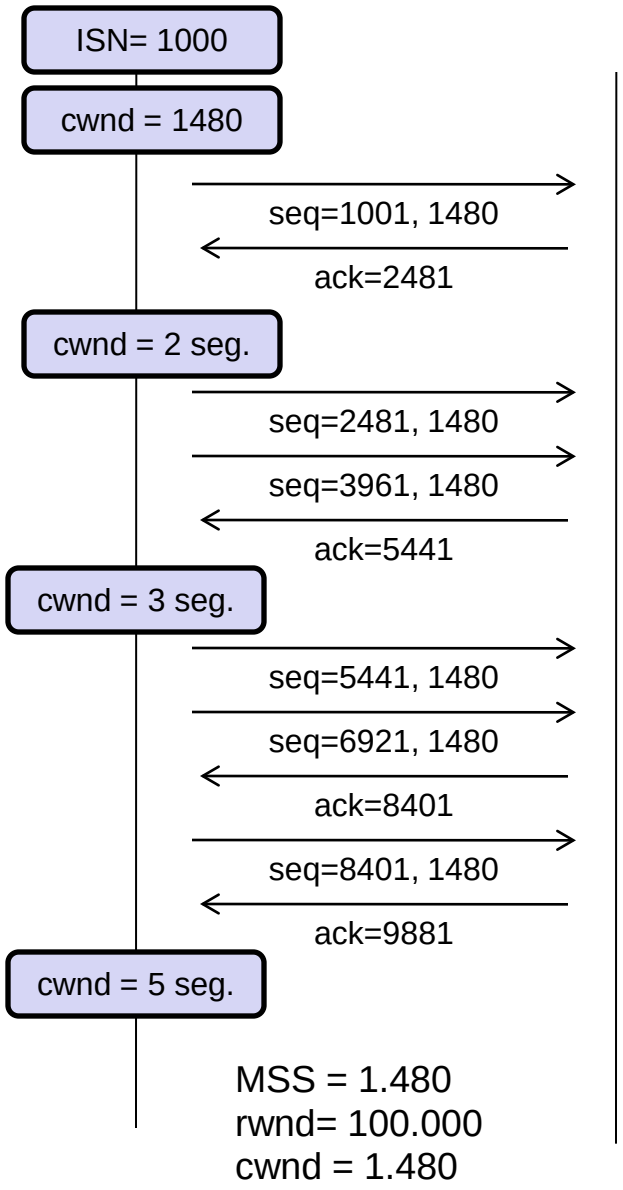
Fereastră glisantă (cont.)

- Roluri
  - eficiența comunicației
  - controlul fluxului – receptorul să nu fie încărcat
- Dimensiunea ferestrei transmițătorului este controlată de cea a receptorului
  - dimensiune mai mică în cazul unei congestii
- Exemplu de funcționare
  - transmițătorul primește un segment de confirmare cu ACK=1000 și WIN=1200
    - receptorul îi confirmă octetul cu numărul 1000
    - receptorul îi precizează dimensiunea fereastrei de 1200
  - transmițătorul poate transmite segmente cu numere de secvență până la ACK+WIN = 1000+1200 = 2200

Controlul congestiei

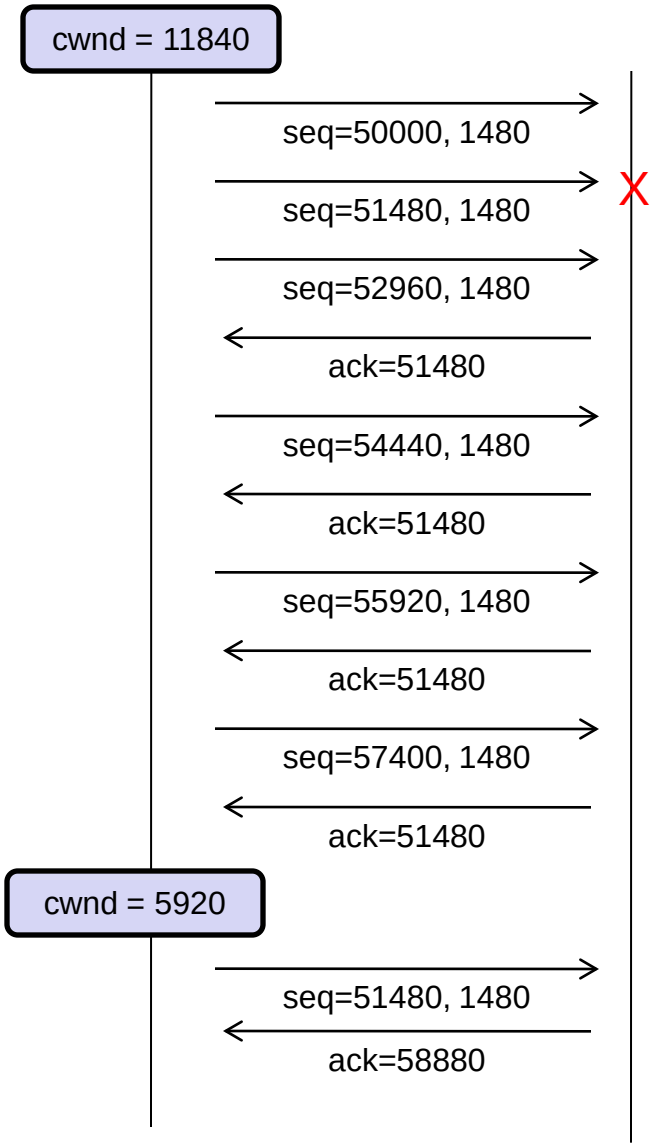
- Congestie - “aglomerarea” datelor (receptor sau ruter din circuitul virtual)
- Număr mare de algoritmi de control al traficului
- **Slow start**
  - transmițătorul controlează viteza de transmisie
  - viteza cu care receptorul transmite segmente de confirmare determină viteza de transmisie
- **Evitarea congestiei (congestion avoidance)**
  - folosit în paralel cu **Slow start**
  - segmente de confirmare piedute → reducerea dimensiunea ferestrei la jumătate
  - date retransmise → dimensiunea ferestrei crește
- **Fast retransmit**
  - prea multe segmente de confirmare la un singur pachet → segmentul sigur a ajuns
  - nu se așteaptă expirarea timerului de retransmisie
- **Fast recovery**
  - mai multe segmente de confirmare → segmentul a ajuns
  - nu se mai pune problema pune problema congestiei
  - se rulează algoritmul **Congestion avoidance** → retransmitere cu o fereastră mai mare

Slow Start



- Funcționarea sa este dictată fereastra receptorului (rwnd), precum şi de două variabile locale:
  - fereastra de congestie – congestion window (cwnd)
  - pragul de creştere – slow start threshold (ssthresh)
- Funcționare:
  - este inițializat la MSS sau fereastra receptorului
  - în general receptorul va trimite confirmare la fiecare două pachete
  - după fiecare rundă de transmisie fereastra de congestie se incrementează cu numărul de confirmări primite, până atinge valoarea ferestrei receptorului sau a pragului de creştere
  - la atingerea ssthresh conexiunea iese din slow start pentru respectivul sens

Fast Retransmit şi Fast Recovery



- Pentru fiecare segment trimis TCP aşteaptă un interval de timp fix (dependent de RTT). La expirarea acestui timp va iniția retransmiterea segmentului
- Pentru Fast Retransmit la primirea a 3 pachete de confirmare duplicate (4 confirmări identice) va considera segmentul pierdut
- În cazul unui segment pierdut pragul de congestie de la transmițător se înjumătățeşte:
  - `ssthresh=cwnd/2=11840/2=5920`
- Pentru Fast Recovery se consideră cele 3 pachete de confirmare duplicate, adăugând 3 segmente la cwnd:
  - `cwnd=ssthresh+3 seg.`

Timere TCP

- **MSL** (Maximum Segment Life)
  - timp de așteptare a unui segment
  - la inchiderea conexiunii, socketul este eliberat după timp 2*MSL
    - •transmiterea ultimului pachet ACK
    - •măsură de siguranta in cazul in care ultimul ACK se pierde
    - •bind: Address already in use
- **RTT** (Round Trip Time)
  - o medie a timpului între transmiterea unui segment și confirmarea acestuia
- **RTO** (Retransmission Timeout)
  - timer de primire a confirmării
- **Keepalive Timer**
  - o conexiune nu schimbă date (idle connection) → la un interval de timp dat se transmit segmente de testare a conexiunii (probe segments)
  - de obicei stabilit la 2 ore
  - în absența confirmării pentru un număr de segmente (de obicei 10) → intervale de 75 de secunde

Opţiuni TCP

- configurarea dimensiunii buffer-ului de transmisie/recepție

```
int window_size = 128 * 1024;/* 128 kilobytes */
setsockopt(sockfd, SOL_SOCKET, SO_SNDBUF, (char *)
    &window_size, sizeof(window_size));
```

- activarea/dezactivarea timer-ului de keepalive

```
int ka_value = 0; /* deactivate keepalive */
setsockopt(sockfd, SOL_SOCKET, SO_KEEPALIVE, (char *)
    &ka_value, sizeof(int)));
```

TCP advances

- TCP over wireless
  - TCP optimizat, în general, pentru transmisii de tip wired
  - pierderea unui pachet -> congestie
    - micșorarea dimensiunii ferestrei
- TCP offload engines
  - dispozitive hardware care implementează TCP
  - evitarea complexității TCP în software
  - creșterea vitezei de transmisie
  - probleme
    - integrarea în sistemele de calcul actuale
    - necesitatea alterării structurii sistemelor de operare

Cuvinte cheie

- nivelul transport
- process-to-process
- modelul client-server
- porturi
- socketi
- TCP
- flux de octeți
- conexiune
- URG, PSH, RST, ACK, SYN, FIN, ECE, CWR
- inițiere conexiune
- 3-way handshake
- ISN
- full-duplex
- încheiere conexiune
- 4-way handshake
- controlul congestiei
- fereastră glisantă
- controlul fluxului
- controlul congestiei
- slow start
- congestion avoindance
- fast retransmit/recovery
- TCP over wireless
- TCP offload engine