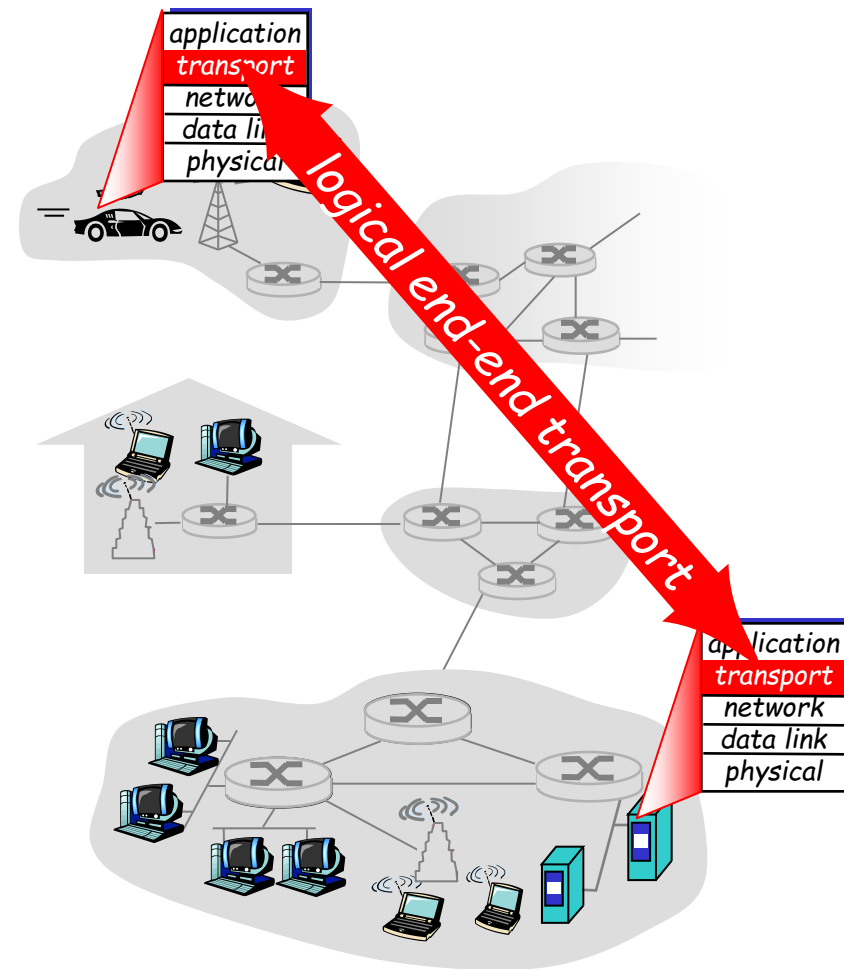# Ports, Services, Transport

Redes de Comunicações 1

Licenciatura em Engenharia de Computadores e Informática

# Transport services and protocols

- provide *logical communication* between application processes running on different hosts
- transport protocols run in end systems
  - sender side: breaks application messages into segments, passes to network layer
  - receiver side: reassembles segments into messages, passes to application layer
- two transport protocols available to applications:
  - TCP and UDP

application
transport
network
data link
physical

logical end-end transport

application
transport
network
data link
physical

Internet transport-layer protocols

□ reliable, in-order delivery: TCP
  ○ connection setup
  ○ flow control
  ○ congestion control

□ unreliable, unordered delivery: UDP
  ○ extension of "best-effort" IP

□ services not available:
  ○ delay guarantees
  ○ bandwidth guarantees

logical end-end transport
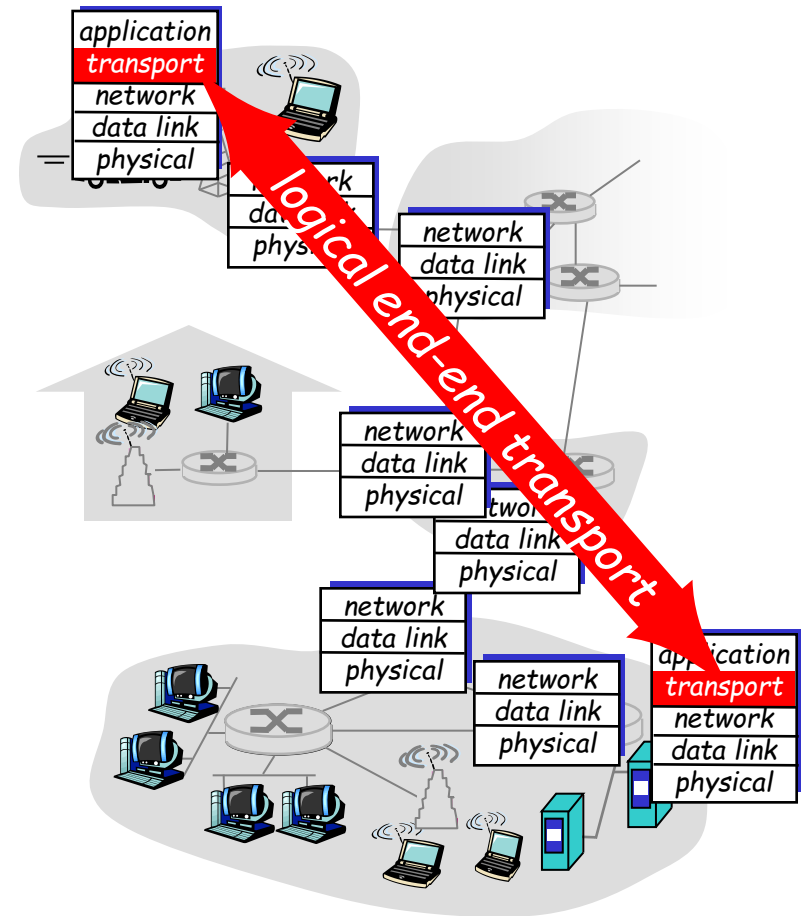
Reference Model

Upper		FTP	Telnet	HTTP
Transport		**UDP**	**TCP**	
Internet		IP		
Link		Ethernet	packet radio	ponto-a-ponto
Physical				

MULTIPLEXING AND DEMULTIPLEXING
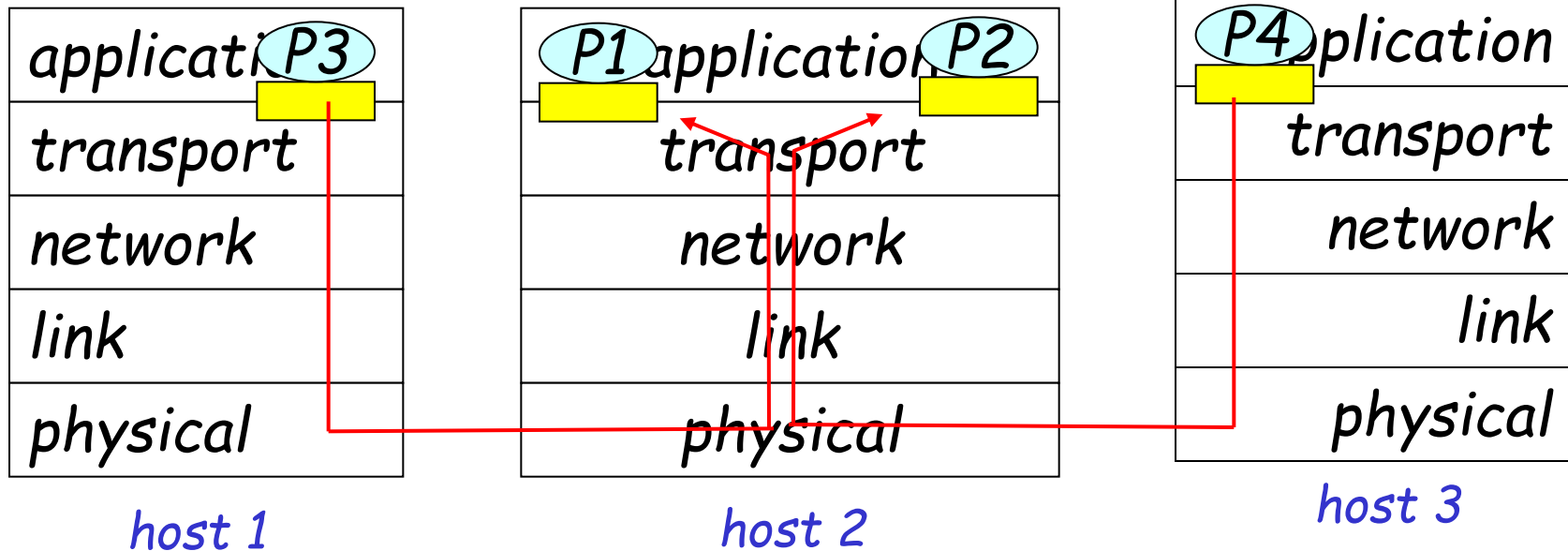
Multiplexing/demultiplexing

Demultiplexing at receiver host:

delivering received segments
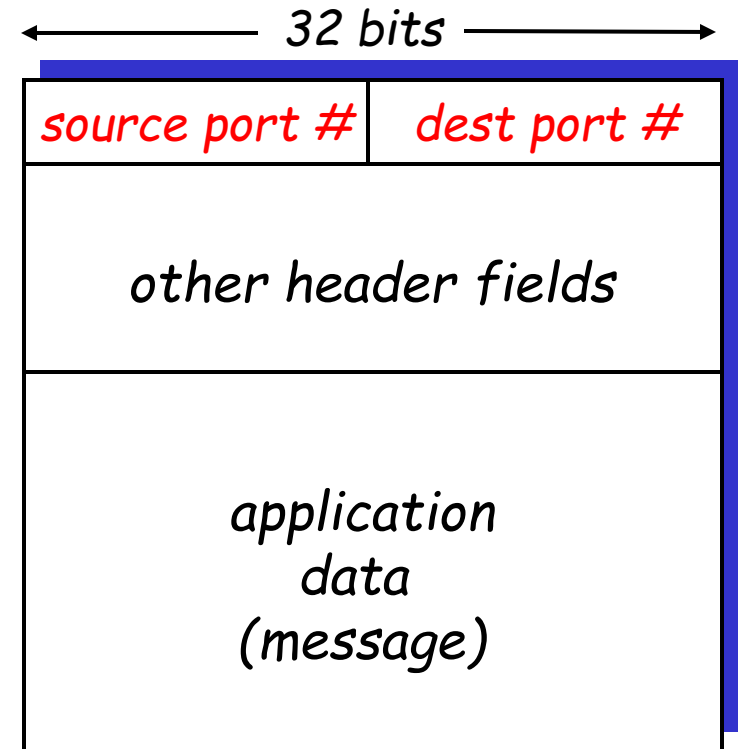to correct socket

Multiplexing at sender host:

gathering data from multiple
sockets, enveloping data with
header (later used for
demultiplexing)

▢ = socket ⬭ = process

host 1   host 2   host 3

How demultiplexing works

□ host receives IP datagrams
  ○ each datagram has source IP address, destination IP address
  ○ each datagram carries 1 transport-layer segment
  ○ each segment has source, destination port number

□ host uses IP addresses and port numbers to direct segment to appropriate socket

```
←——— 32 bits ———→

┌─────────────────┬─────────────────┐
│  source port #  │   dest port #   │
├─────────────────┴─────────────────┤
│                                   │
│        other header fields        │
│                                   │
├───────────────────────────────────┤
│                                   │
│            application            │
│               data                │
│            (message)              │
│                                   │
└───────────────────────────────────┘
```

TCP/UDP segment format

Connectionless demultiplexing

□ Create sockets with port numbers:

```
DatagramSocket mySocket1 = new
    DatagramSocket(12534);

DatagramSocket mySocket2 = new
    DatagramSocket(12535);
```
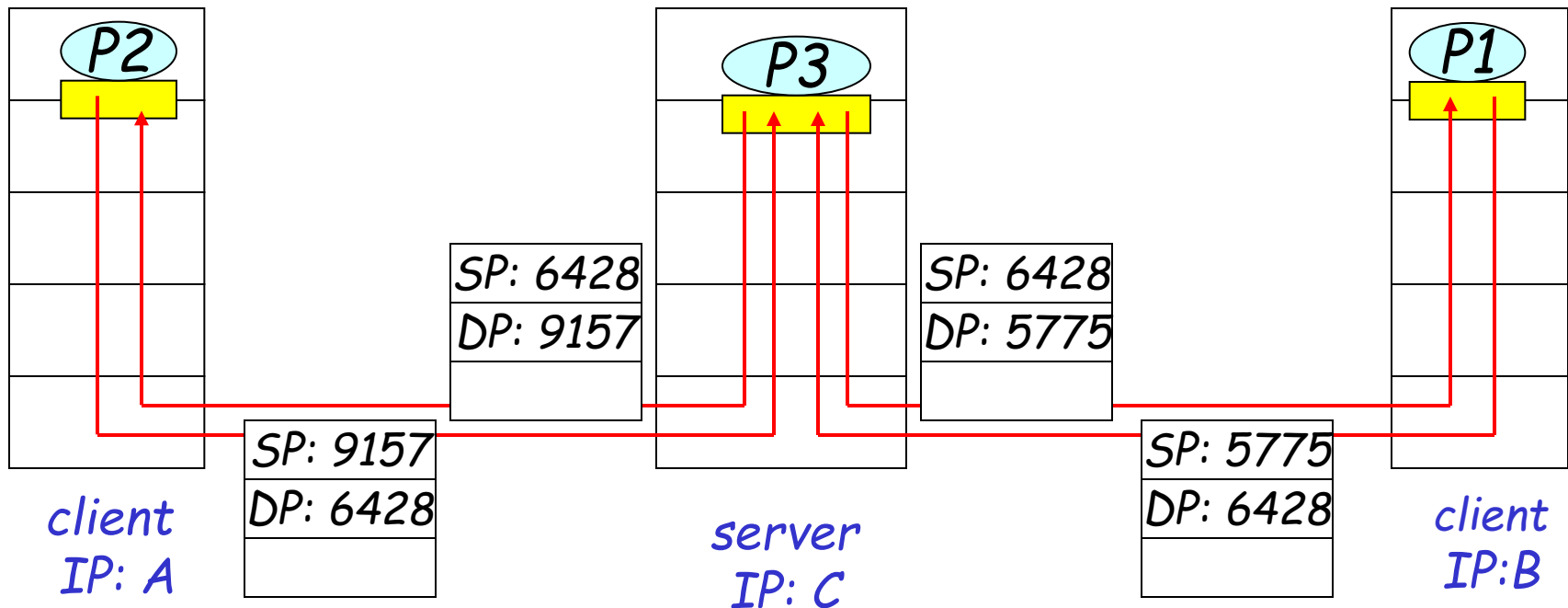
□ UDP socket identified by 2-tuple:

(destination IP address, destination port number)

□ When host receives UDP segment:
  ○ checks destination port number in segment
  ○ directs UDP segment to socket with that port number

□ IP datagrams with different source IP addresses and/or source port numbers are directed to the same socket

Connectionless demux (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```
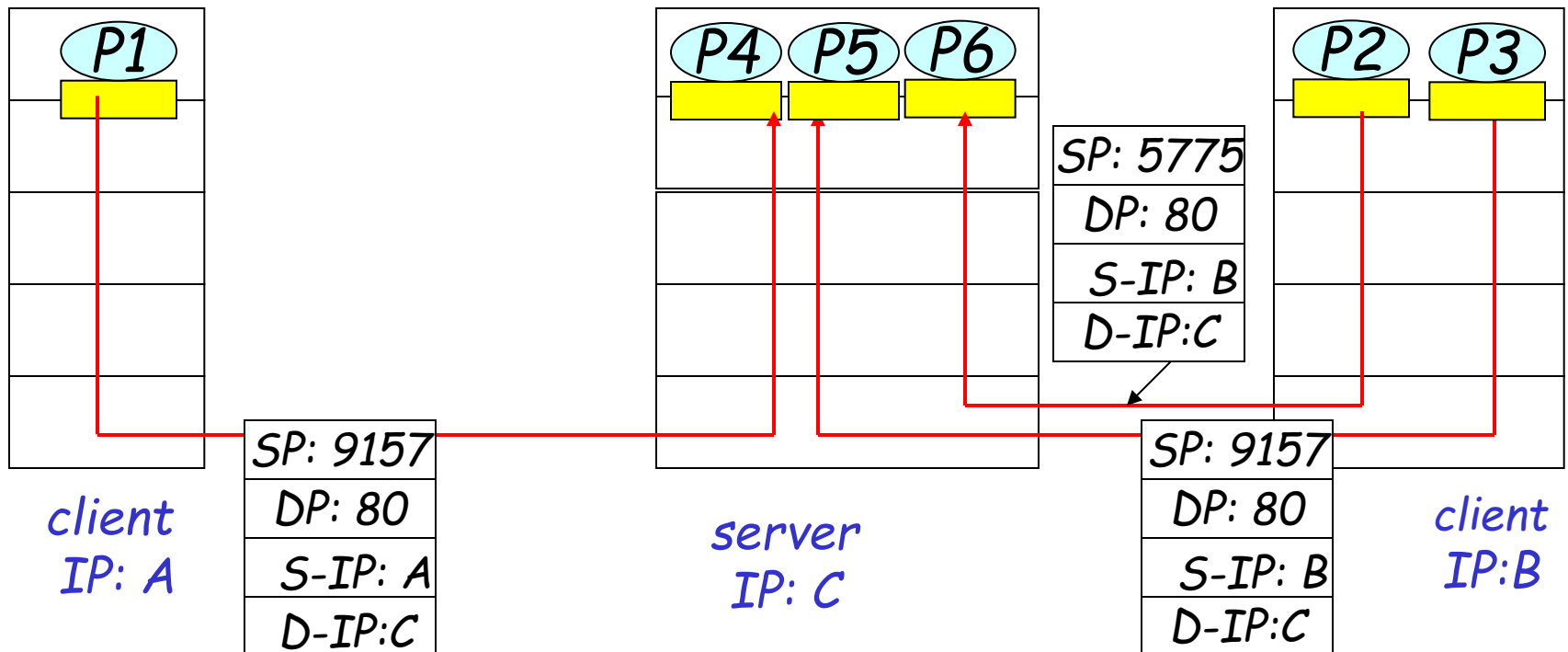

P2

P3

P1

SP: 6428
DP: 9157

SP: 6428
DP: 5775

SP: 9157
DP: 6428

SP: 5775
DP: 6428

client
IP: A

server
IP: C

client
IP:B

SP provides "return address"

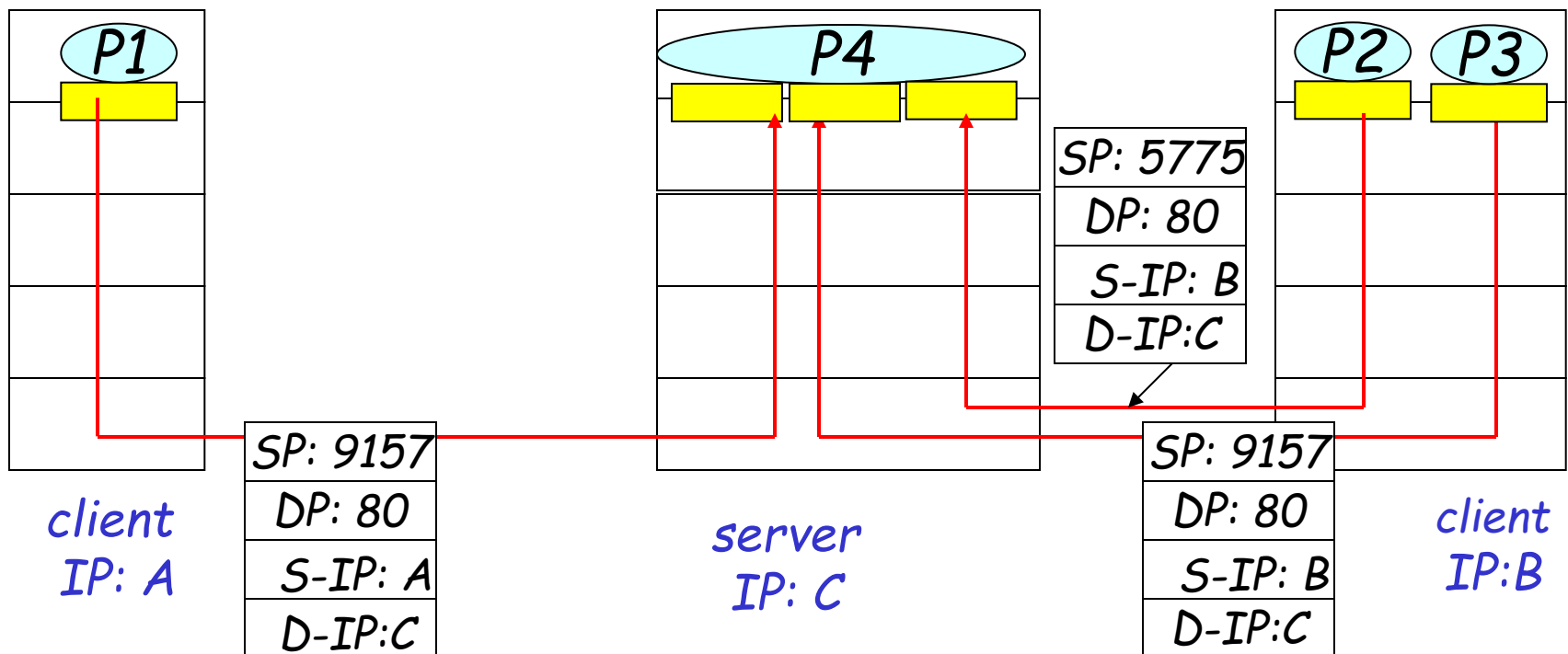# Connection-oriented demux

□ TCP socket identified by 4-tuple:
  ○ source IP address
  ○ source port number
  ○ destination IP address
  ○ destination port number

□ receiver host uses all four values to direct segment to appropriate socket

□ Server host may support many simultaneous TCP sockets:
  ○ each socket identified by its own 4-tuple

□ For example, Web servers have different sockets for each connecting client

Connection-oriented demux (cont)

P1

P4 P5 P6

P2 P3

SP: 5775
DP: 80
S-IP: B
D-IP:C

client
IP: A

SP: 9157
DP: 80
S-IP: A
D-IP:C

server
IP: C

SP: 9157
DP: 80
S-IP: B
D-IP:C

client
IP:B

Connection-oriented demux: Threaded Web Server

P1

P4

P2   P3

SP: 5775
DP: 80
S-IP: B
D-IP:C

SP: 9157
DP: 80
S-IP: A
D-IP:C

SP: 9157
DP: 80
S-IP: B
D-IP:C

client
IP: A

server
IP: C

client
IP:B

Allocated Ports

Decimal	Palavra Chave	Protocolo	Descrição
20	FTP-DATA	TCP	File Transfer Protocol (dados)
21	FTP-CONTROL	TCP	File Transfer Protocol (controlo)
23	TELNET	TCP	Terminal Connection
25	SMTP	TCP	Simple Mail Transport Protocol
67,68	BOOTP	UDP	Bootstrap Protocol
53	DNS	UDP/TCP	Domain Name System
69	TFTP	UDP	Trivial File Transfer Protocol
80	HTTP	TCP	Hypertext Transfer Protocol

UDP

User Datagram Protocol

☐ Provides a seamless service to transport data with the performance characteristics offered by IP

☐ Allows the exchange of data between applications, through a header and port identifier

☐ Allows the sending of data for multiple destinations (multi-point communications)

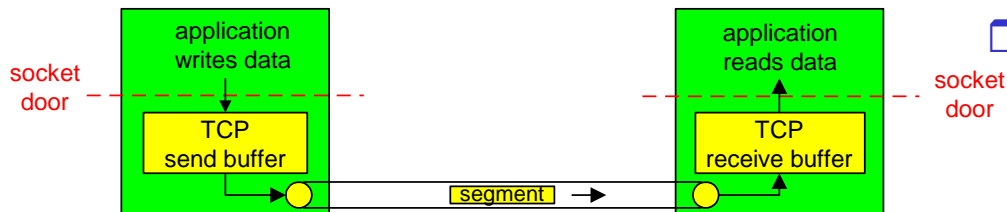| 0	16	31
Source Port	Destination Port	
Datagram Lenght	Checksum	
Data		
...		

☐ *Checksum:* datagrama UDP + *pseudoheader* IP (ID protocol IP, sender IP address, destination IP address, lenght IP datagram)

  ❖ Verify if the message was sent between the correct endpoints
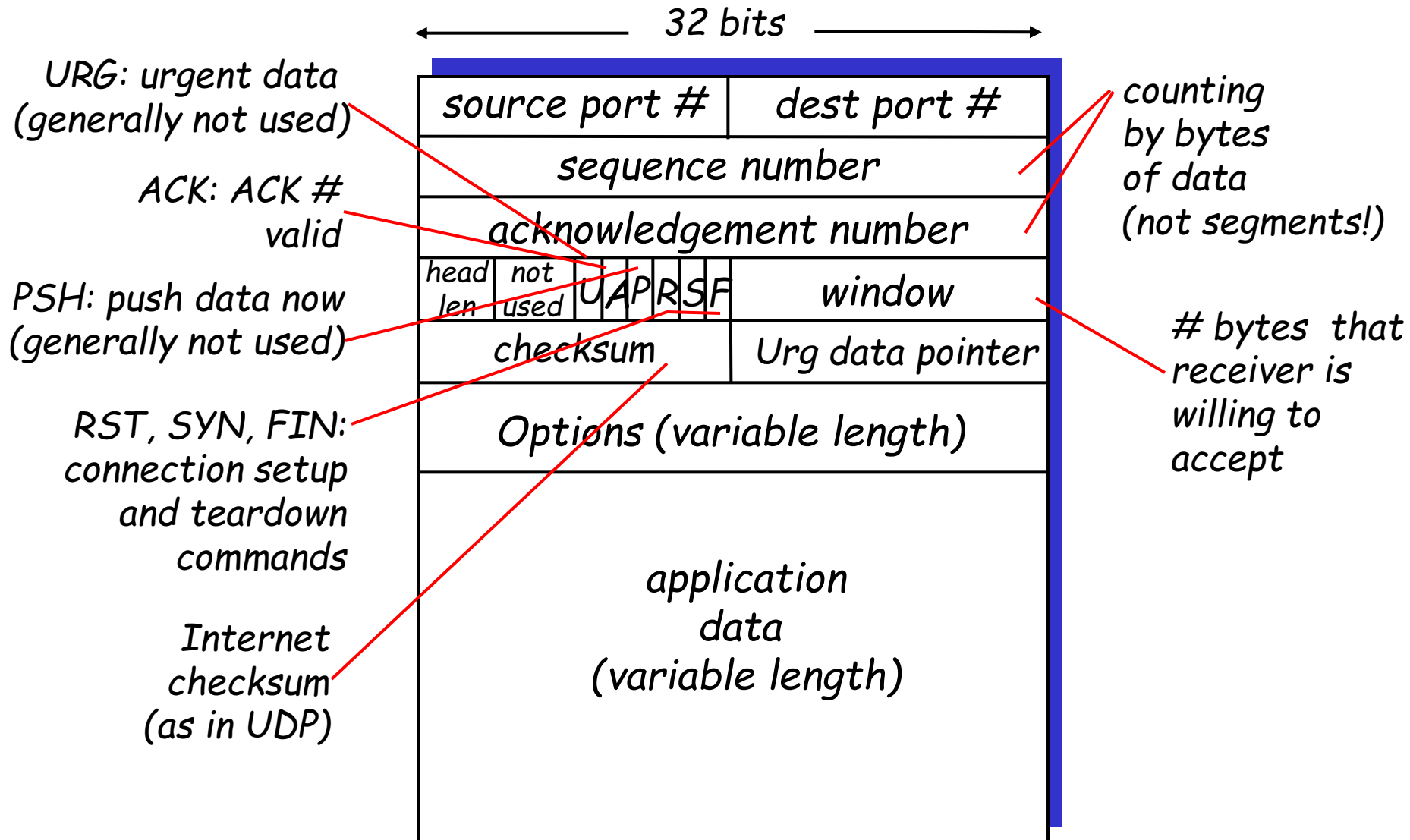
TCP

TCP: Overview   RFCs: 793, 1122, 1323, 2018, 2581

- □ point-to-point:
  - ○ one sender, one receiver
- □ reliable, in-order *byte stream:*
  - ○ no "message boundaries"
- □ pipelined:
  - ○ TCP congestion and flow control set window size
- □ *send & receive buffers*

- □ full duplex data:
  - ○ bi-directional data flow in same connection
  - ○ MSS: Maximum Segment Size; in general, MTU of attached link – (IP + TCP header lengths)
- □ connection-oriented:
  - ○ handshaking (exchange of control messages): initiates sender and receiver state before data exchange
- □ flow controlled:
  - ○ sender will not overwhelm receiver

socket door

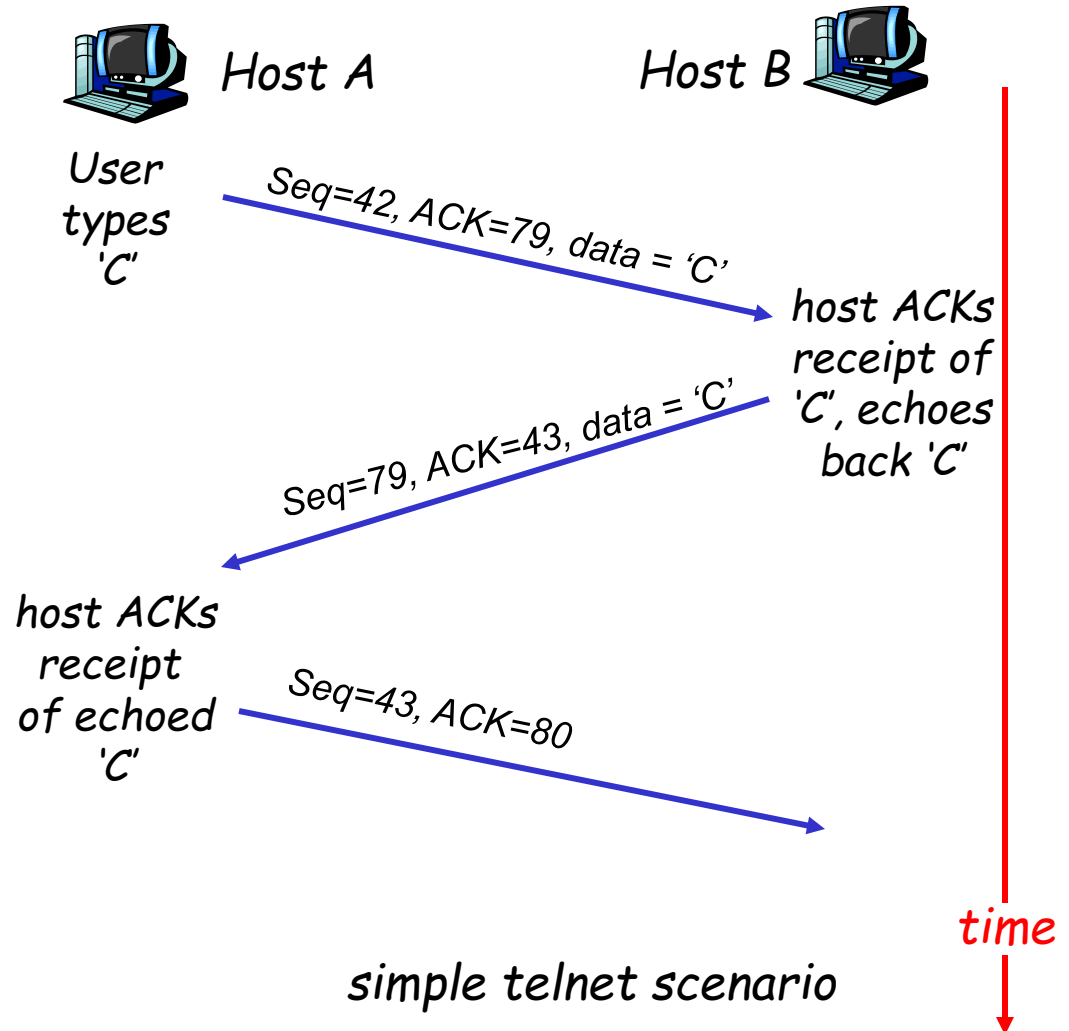application writes data

TCP send buffer

socket door

application reads data

TCP receive buffer

segment

TCP segment structure

32 bits

URG: urgent data
(generally not used)

ACK: ACK #
valid

PSH: push data now
(generally not used)

RST, SYN, FIN:
connection setup
and teardown
commands

Internet
checksum
(as in UDP)

source port #	dest port #
sequence number	
acknowledgement number	
head len	not used	U A P R S F	window
checksum	Urg data pointer
Options (variable length)	
application data (variable length)	

counting
by bytes
of data
(not segments!)

bytes that
receiver is
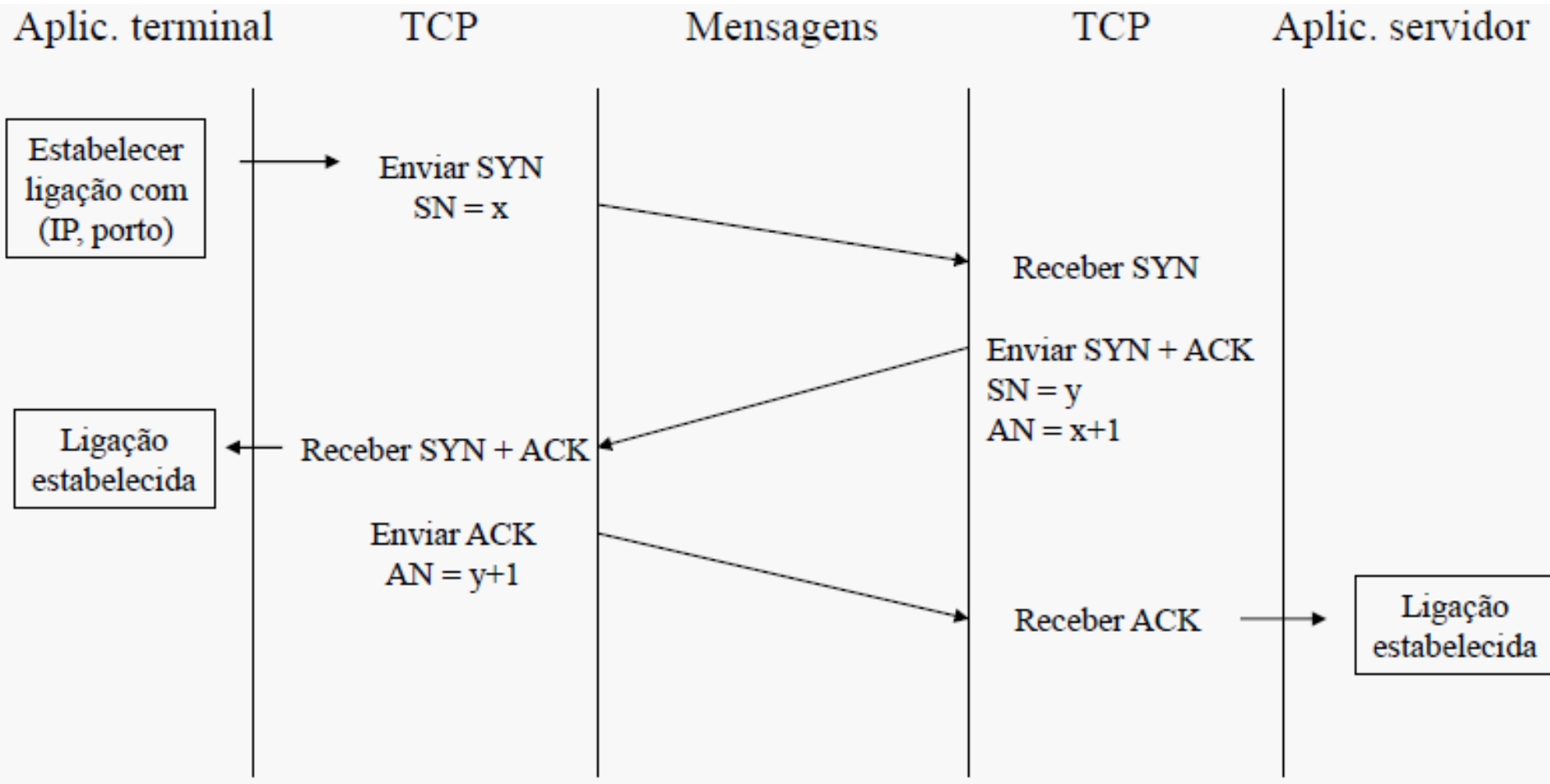willing to
accept

TCP seq. numbers and ACKs

Seq. number:
- byte stream "number" of first byte in segment's data

ACKs:
- seq. number of next byte expected from other side
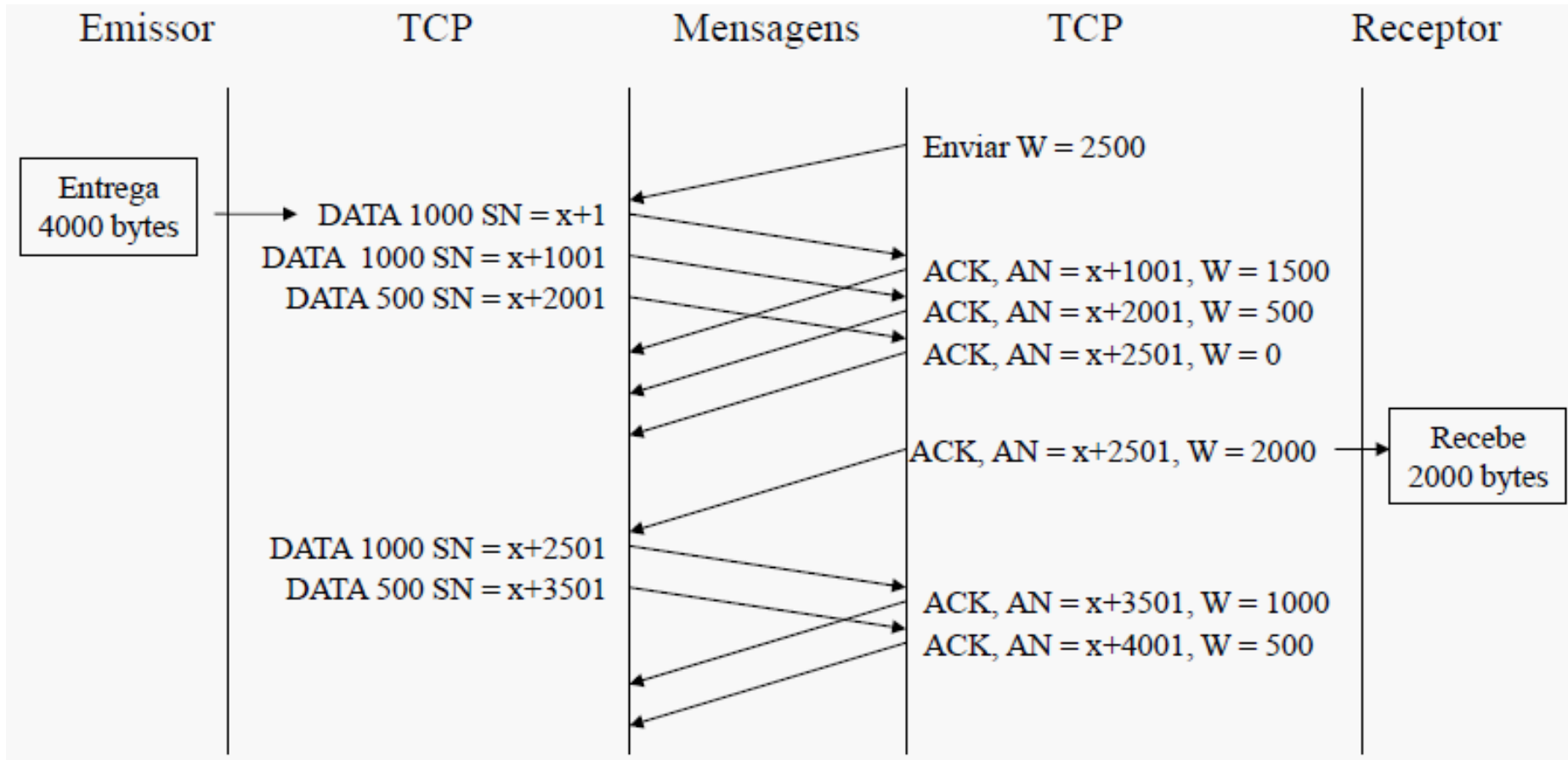- cumulative ACK

Host A   Host B

User types 'C'

Seq=42, ACK=79, data = 'C'

host ACKs receipt of 'C', echoes back 'C'

Seq=79, ACK=43, data = 'C'

host ACKs receipt of echoed 'C'

Seq=43, ACK=80

time

simple telnet scenario

Establishment of a TCP Session

Termination of a TCP Session

Flow Control

Emissor   TCP   Mensagens   TCP   Receptor

Entrega 4000 bytes

Enviar $W = 2500$

DATA 1000 SN $= x+1$

DATA 1000 SN $= x+1001$

DATA 500 SN $= x+2001$

ACK, AN $= x+1001$, $W = 1500$

ACK, AN $= x+2001$, $W = 500$

ACK, AN $= x+2501$, $W = 0$

ACK, AN $= x+2501$, $W = 2000$

Recebe 2000 bytes

DATA 1000 SN $= x+2501$

DATA 500 SN $= x+3501$

ACK, AN $= x+3501$, $W = 1000$

ACK, AN $= x+4001$, $W = 500$
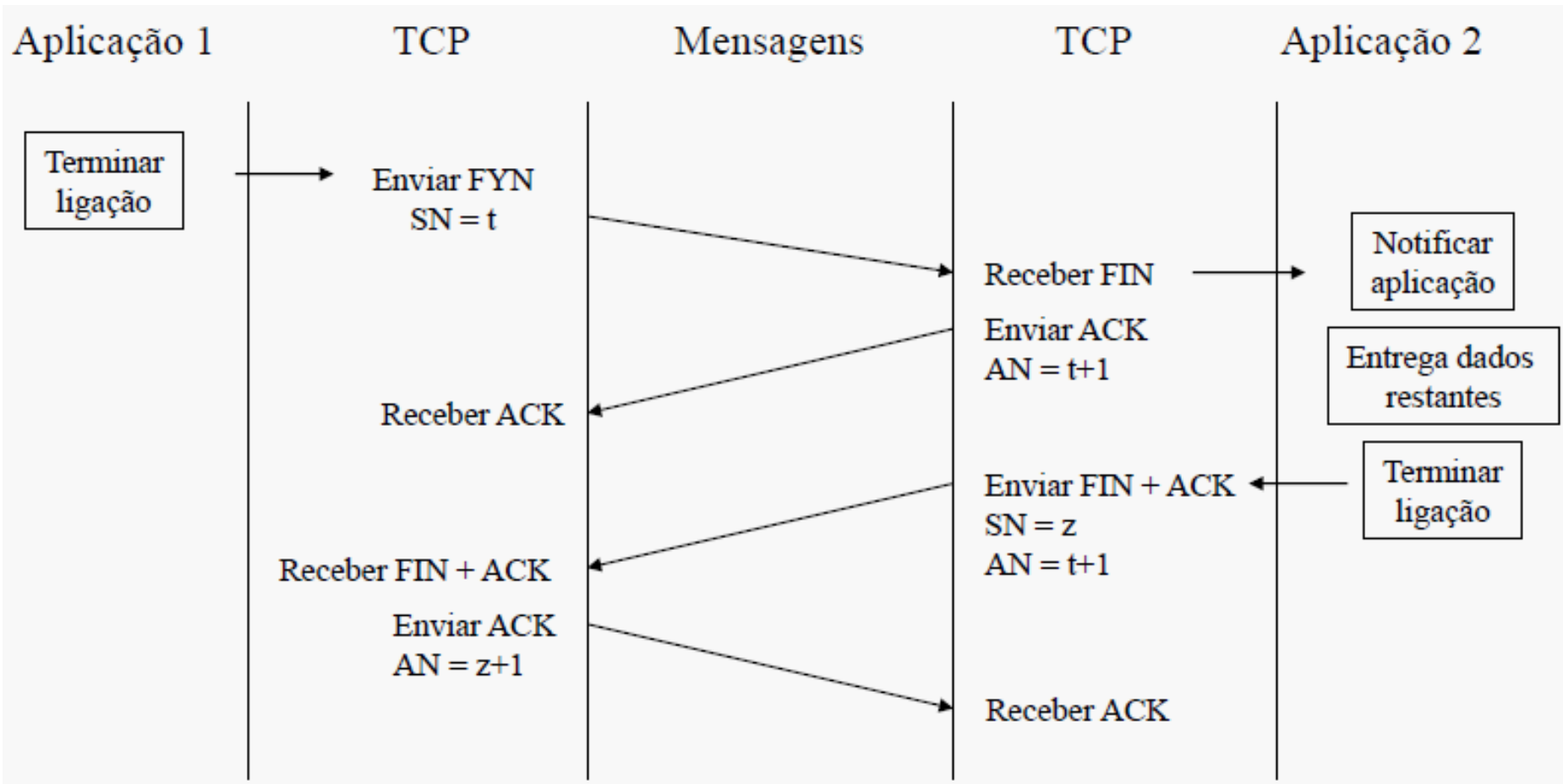
TCP Header Fields

□ *Sequence Number :* data already sent

□ *Acknowledge Number :* data already received

□ *Window :* receiver informs sender of how many octets is ready to receive

□ *Sequence Number* refers to transmission side, *Acknowledge Number* and *Window* refer to the opposite direction

Example (1)

□ Consider a TCP connection from A to B. In both stations, TCP:

  ○ Considers a reception buffer of 2000 bytes

  ○ Segments information is packets with a maximum of 1000 bytes

  ○ Station A chooses an initial Sequence Number of 1515, and Station B chooses an initial Sequence Number of 502

  ○ Station A sends a block of 5300 bytes and station B does not send data.

□ Draw the timing diagram of TCP segments exchanged, including establishment and termination.

Example (2)

- Consider a TCP connection between 2 stations A and B. They exchanged the following TCP segments. Which station is the client? What is the size (in bytes) of the data field in each segment, and the overall size of the data?

	Origem	Destino	SN	AN	W	Flags
1° segmento:	A	B	255	0	10000	SYN,
2° segmento:	B	A	4329	256	20000	SYN, ACK
3° segmento:	A	B	256	4330	10000	ACK
4° segmento:	B	A	4330	256	20000	ACK
5° segmento:	A	B	256	5000	10000	ACK
6° segmento:	B	A	5000	276	20000	FIN , ACK
7° segmento:	A	B	276	5201	10000	ACK
8° segmento:	A	B	276	5201	10000	FIN , ACK
9° segmento:	B	A	5201	277	1200	ACK

How to know that a packet shall be retransmitted?

After a specific time, the packet needs to be considered lost

Long time always?

Small time always?

Should depend on the delays in the network/path
Should depend on the round trip time

TCP Round Trip Time (RTT) and Timeout

Question: how to set TCP timeout value?

- larger than RTT
  - but RTT varies
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

Question: how to estimate RTT?

- **SampleRTT:** measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **SampleRTT** will vary, want estimated RTT "smoother"
  - to average several recent measurements, and not used just the last **SampleRTT**

TCP Round Trip Time (RTT) and Timeout

$$EstimatedRTT = (1- \alpha)*EstimatedRTT + \alpha*SampleRTT$$

❒ Exponential weighted moving average
❒ Influence of past sample decreases exponentially fast
❒ Typical value: $\alpha = 0.125$

EstimatedRTT after the Kth ACK
SampleRTT of Kth ACK

Example RTT estimation:

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr
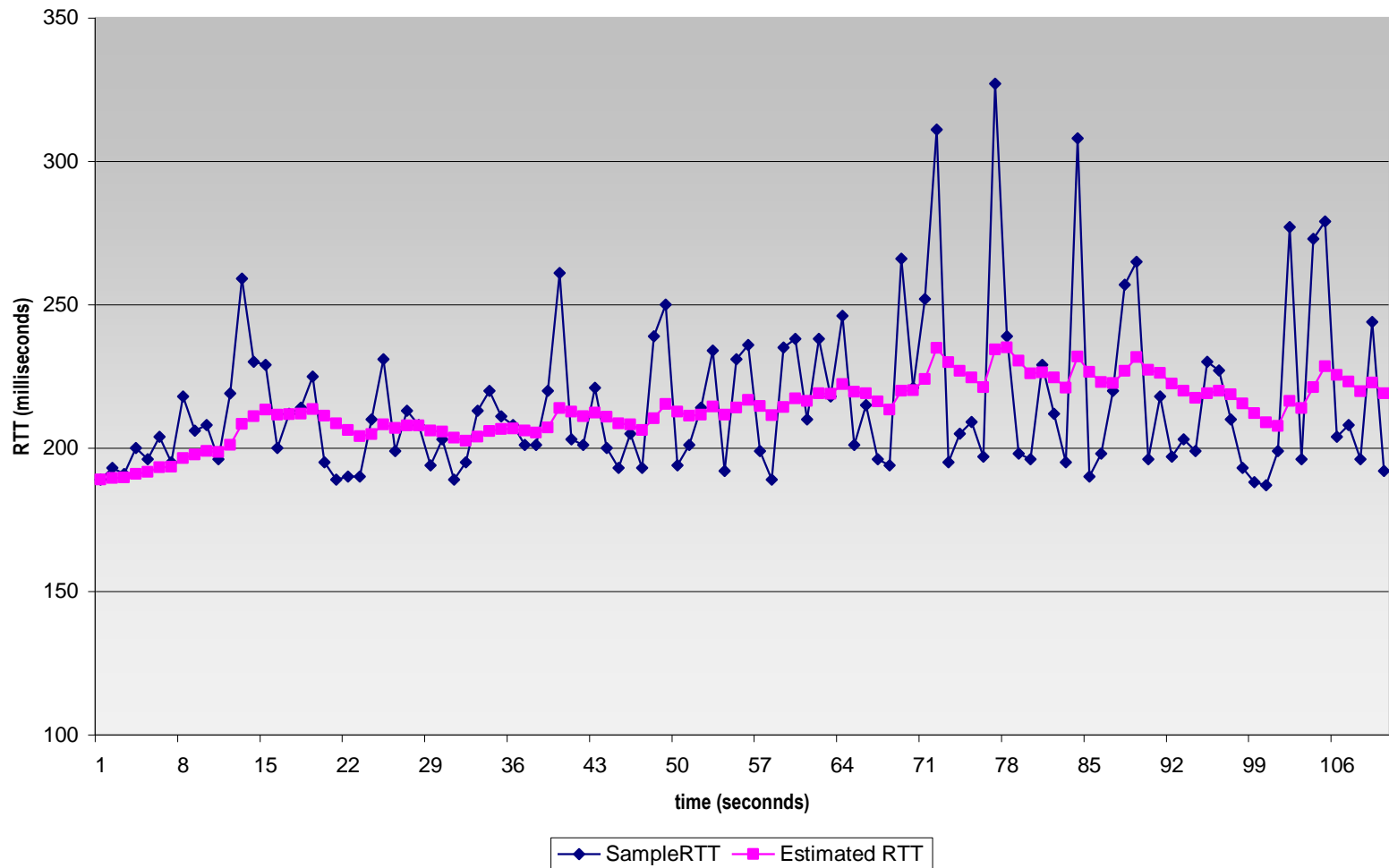
TCP Round Trip Time and Timeout

Setting the timeout

☐ **EstimatedRTT** plus "safety margin"
  ○ large variation in **EstimatedRTT** -> larger safety margin

☐ first estimate of how much SampleRTT deviates from EstimatedRTT:

$$DevRTT = (1-\beta)*DevRTT + \beta*|SampleRTT-EstimatedRTT|$$

$$(typically, \beta = 0.25)$$

Then set timeout interval:

$$TimeoutInterval = EstimatedRTT + 4*DevRTT$$

βxRTT

IETF RFC 793 proposes $1.3 < \beta < 2.0$

TCP Retransmissions

TCP Slow Start

□ **When connection begins, increase rate exponentially until first loss event:**

  ○ double `CongWin` every RTT

  ○ done by incrementing `CongWin` for every ACK received

□ Summary: initial rate is slow but ramps up exponentially fast

Host A   Host B

RTT

one segment

two segments

four segments
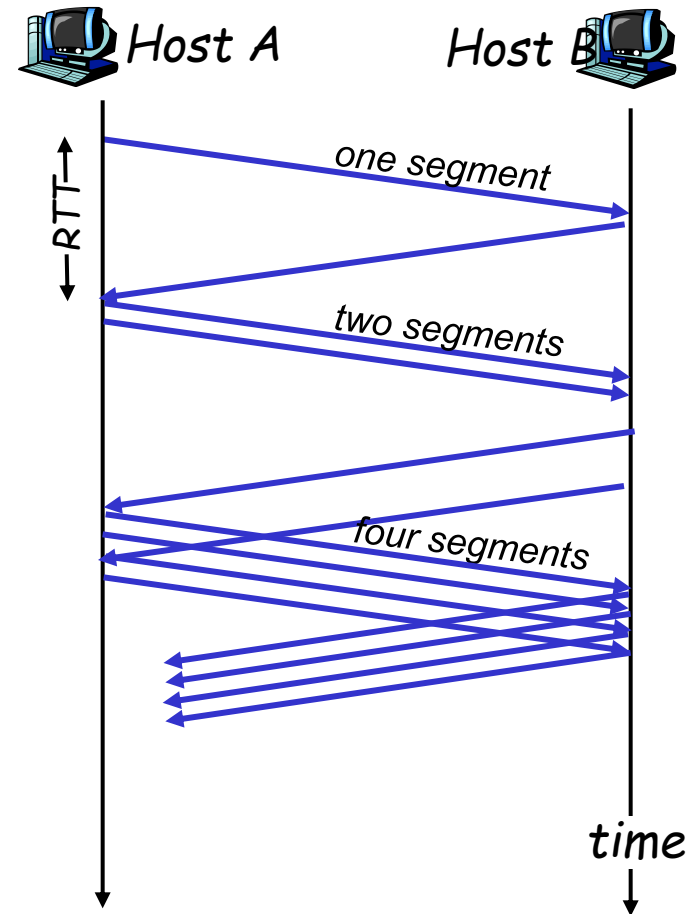
time

TCP Slow Start

□ When connection begins, `CongWin` = 1 packet

□ Each time an ACK is received, `CongWin` is incremented by 1 segment of maximum size

□ *When connection begins, increase rate exponentially fast until first loss event*
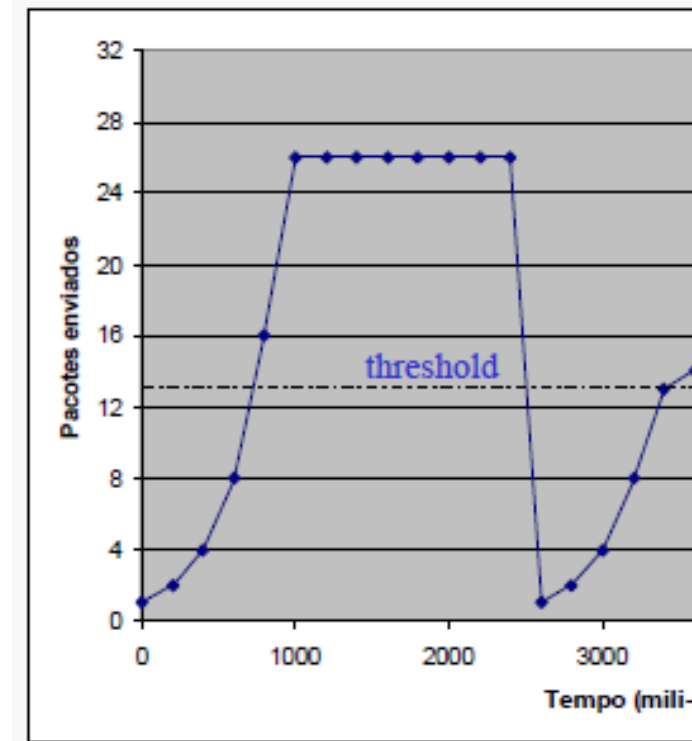
TCP Congestion avoidance

□ When the rate is larger, increase the window more carefully

  ○ **Congestion avoidance**
  ○ Window increases linearly to avoid losses
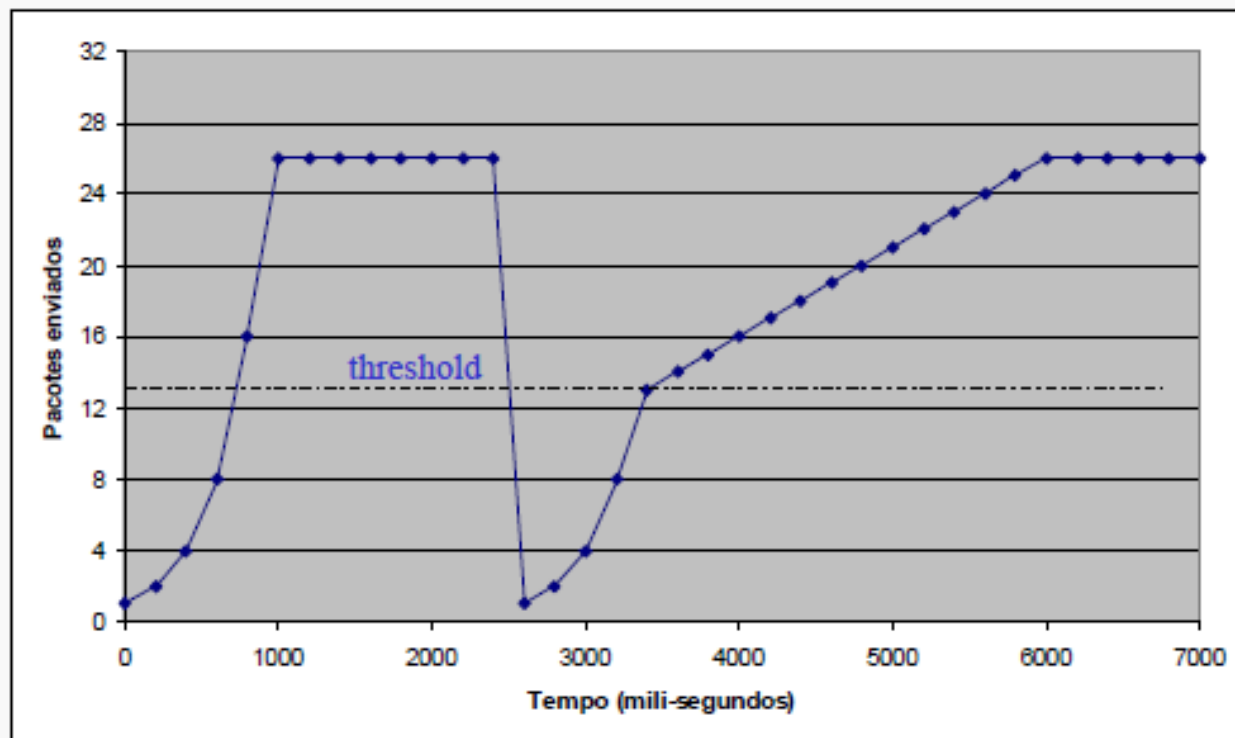
Fast Retransmission mechanism

r Detect lost segments via duplicate ACKs

  o Sender often sends many segments back-to-back

  o If segment is lost, there will likely be many duplicate ACKs.

r If sender receives 3 more ACKs for the same data, it supposes that segment after ACKed data was lost:

  o fast retransmission mechanism: when 3 duplicate ACKs are received, resend segment before timer expires

Duplicate ACKs before timeout: network is not too congested!

Duplicate ACKs and fast retransmission

Sender   Receiver

Segment 1
Segment 2
Segment 3   ACK 2
Segment 4   ACK 3
                   ACK 3
Segment 5
Segment 6   ACK 3
                   ACK 3
Retranmission
of segment 3
                   ACK 7
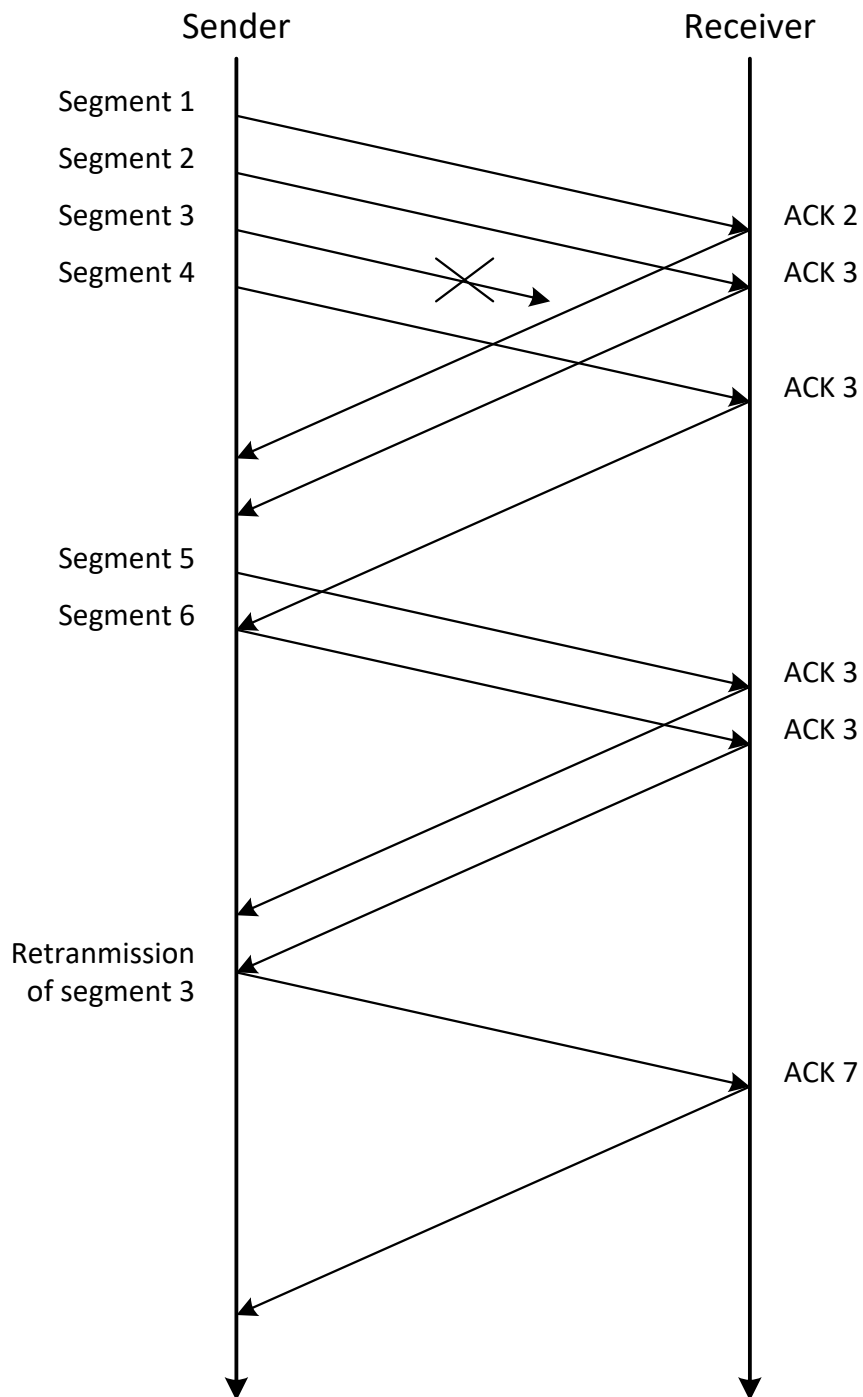
❑ Duplicate ACKs
  ❖ Sender receives 3 duplicate ACKs of segment 3

❑ Fast retransmission
  ❖ Sender retransmits segment 3 before Timeout of segment 3 expires

Refinement: inferring loss

□ After 3 dup ACKs:
  ○ `CongWin` is cut in half
  ○ window then grows linearly

□ <u>But</u> after timeout event:
  ○ `CongWin` instead set to 1 MSS;
  ○ window then grows exponentially
  ○ to a threshold, then grows linearly

Philosophy:

❑ *3 dup ACKs indicates network capable of delivering some segments*
❑ *timeout indicates a "more alarming" congestion scenario*

TCP Tahoe versus TCP Reno

TCP Tahoe:

- Congestion detection based only on Timeouts
- Slow start at beginning and when timeout occurs

TCP Reno

- Congestion detection based on Timeouts and 3 duplicated ACKs
- When timeout occurs
  - TCP Tahoe
- When 3 duplicated ACKs
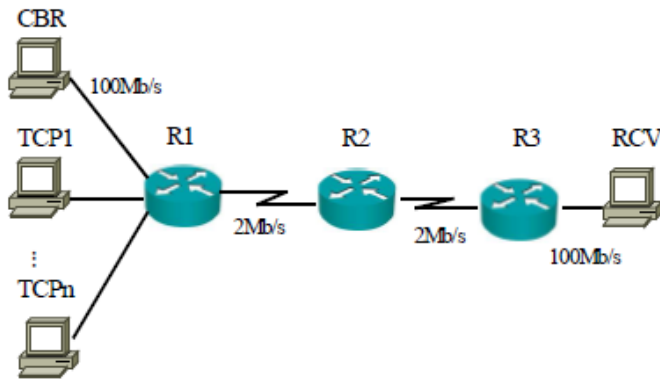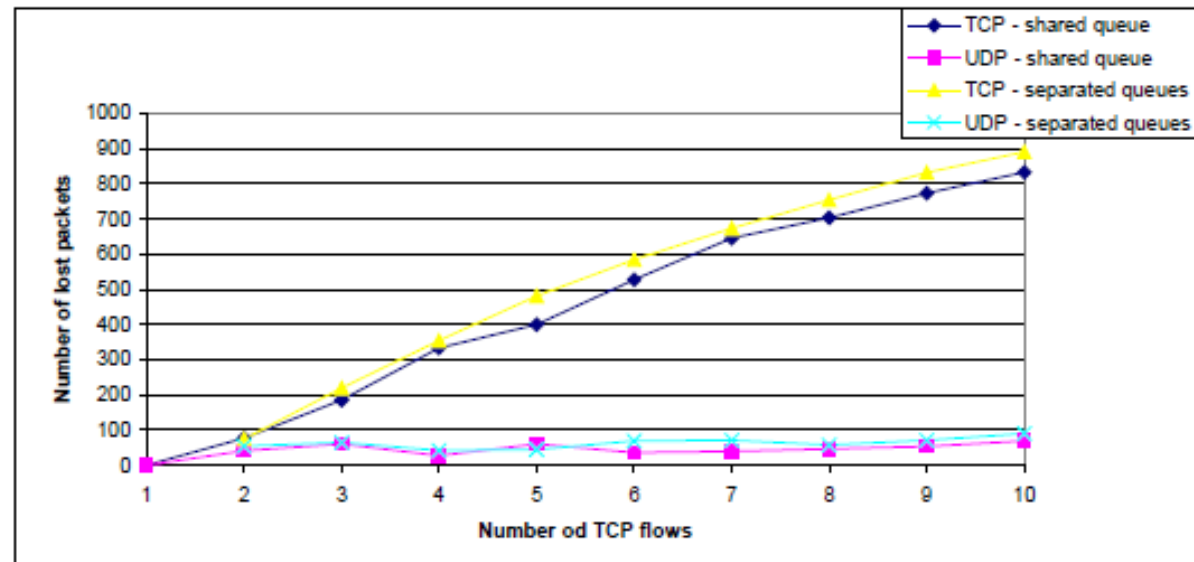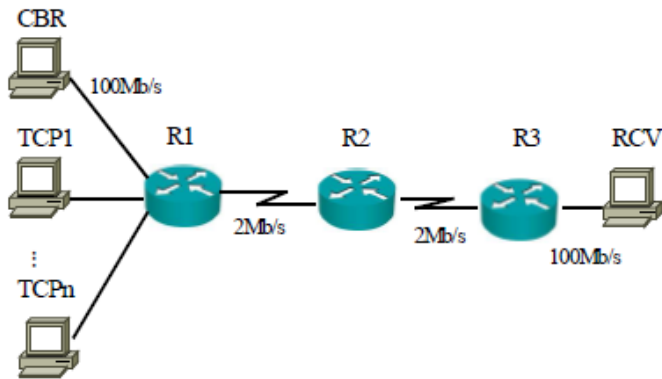 - Fast retransmission
  - Fast recovery
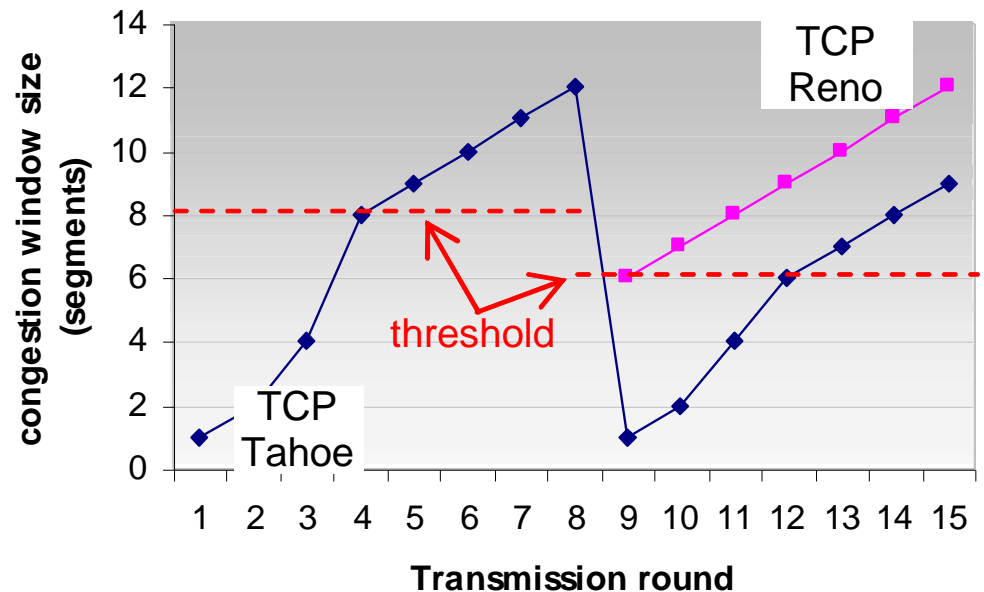
UDP and TCP coexistence
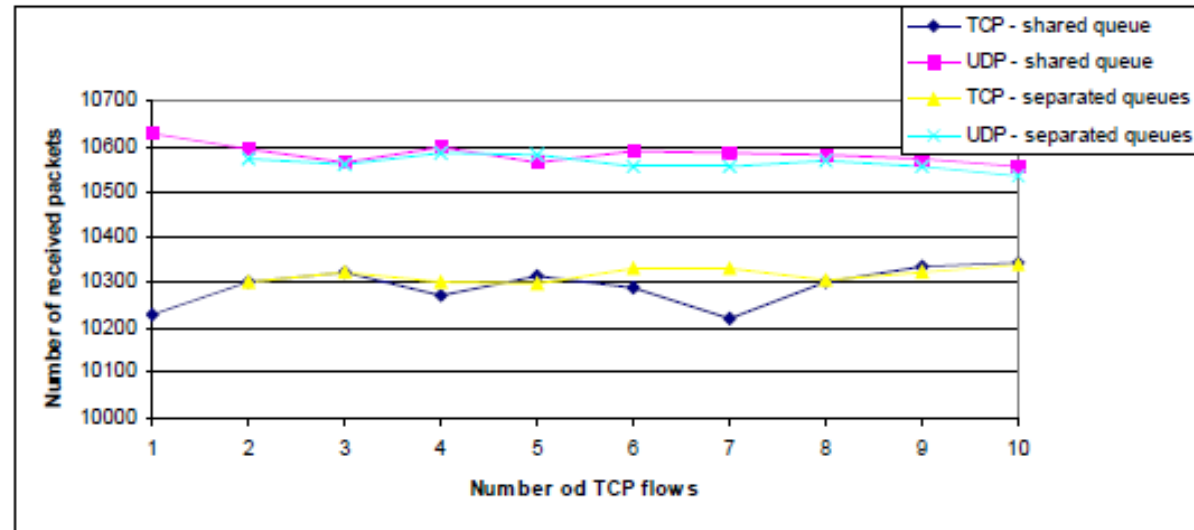
Figure 2. Number of received TCP and UDP packets for both simulation scenarios

Queue size [packet]	60
Each virtual queue size when traffic was separated [packet]	30
Bit rate of CBR source [Mbit/s]	1
Version of TCP	SACK
Packet size [B]	1000

UDP and TCP coexistence

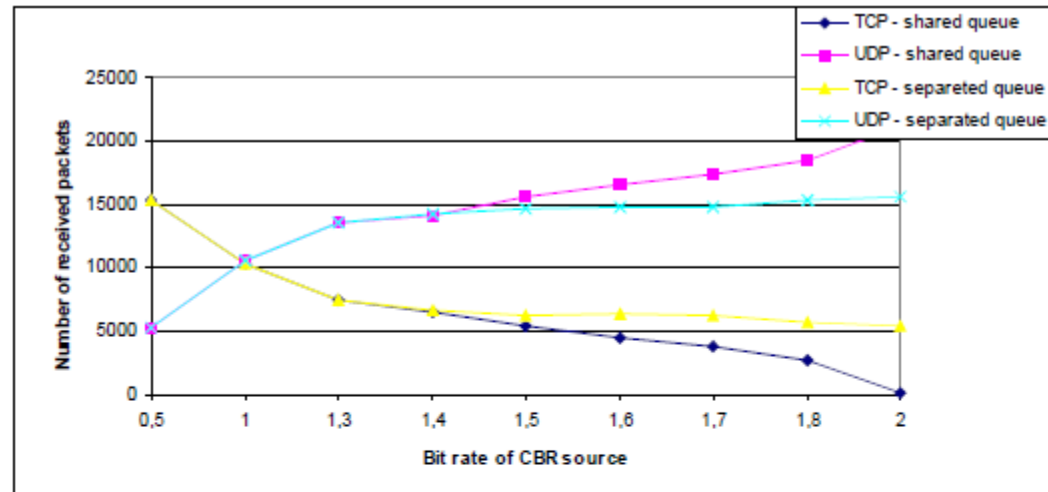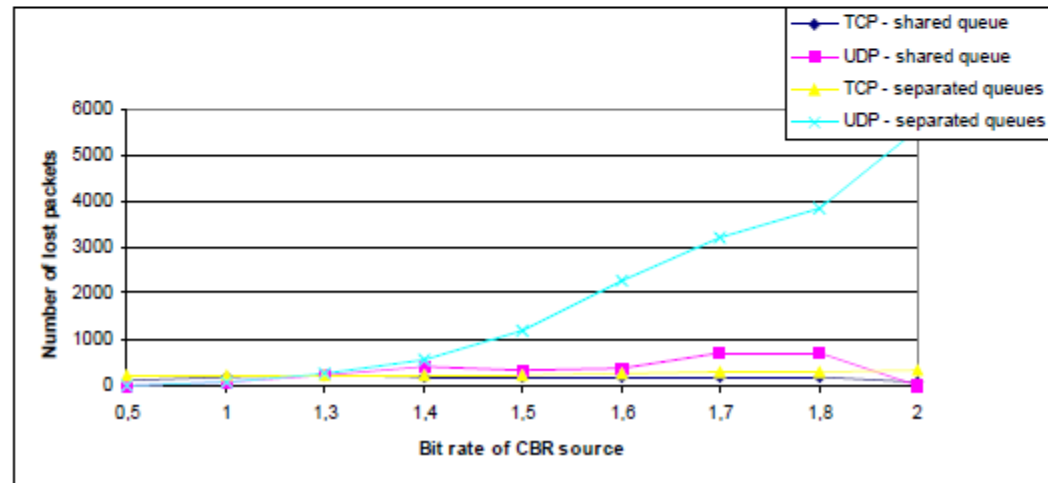|	
Queue size [packet]	60
Each virtual queue size when traffic was separated [packet]	30
Bit rate of CBR source [Mbit/s]	1
Version of TCP	SACK
Packet size [B]	1000

Figure 4. Number of received TCP and UDP packets for both simulation scenarios

IPTV: Reliable UDP (R-UDP)

□ Sent in multicast UDP to the SetTopBox
□ Losses found
  ○ Losses information sent in unicast UDP (packet sequence numbers)
  ○ Retransmission sent in unicast UDP
□ Buffer in the application
  ○ Can go up to 8 sec (normal is 1 sec)
  ○ If retransmission arrives before play-out time, it is included in the play-out
  ○ STB in places with real-time visualizations (such as football games)
    • Very small buffers

IPTV: Instant Channel Change (ICC)

□ Normal channel change
  ○ Multicast join to the server, all routers will have to start sending the multicast flow
  ○ Buffers have to fill to start the play-out (at the rate of the multicast flow)

□ ICC
  ○ Parallel to the multicast join to the server, there is a message to the distribution server
  ○ This server maintains the last 8 sec of all channels
  ○ Sends by unicast stream the latest information at the fastest path rate
  ○ Fills the buffer more quickly and play-out the video (40 msec)

□ When multicast process is done, normal multicast will be received

Bibliography to study

☐ J. Kurose, K. Ross, "Computer Networking: A Top-Down Approach", Addison-Wesley, 4th Edition

  ○ Chapter 3 "Transport Layer"