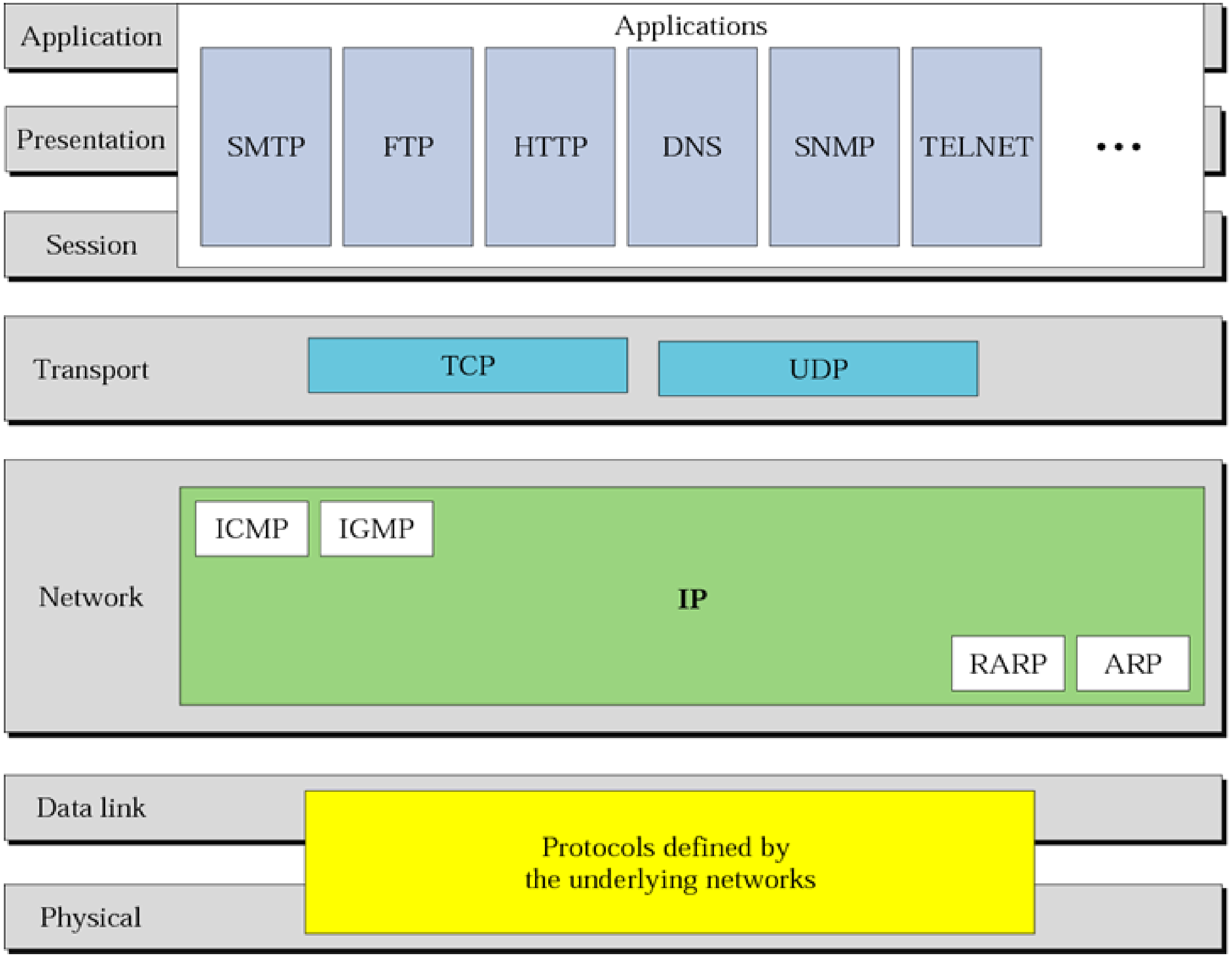
Department of Computer and IT Engineering
University of Kurdistan

Advanced Computer Networks

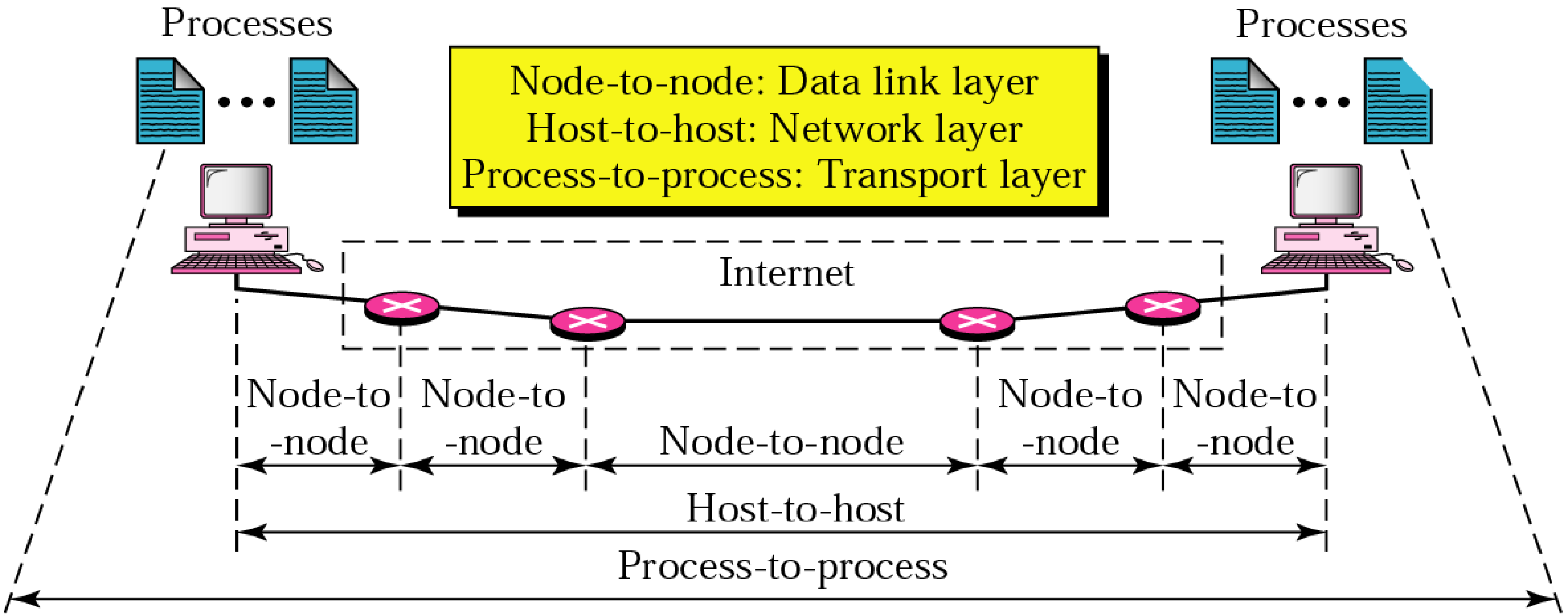
Transport Layer

By: Dr. Alireza Abdollahpouri

TCP/IP protocol suite



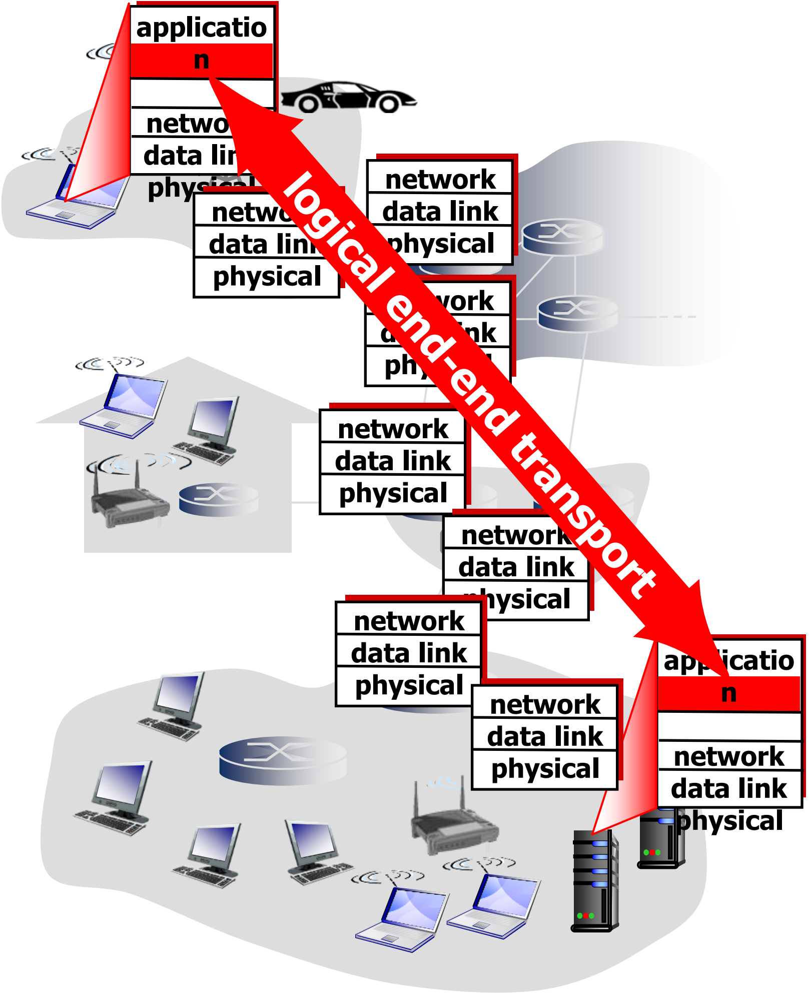
Transport Layer



The transport layer is responsible for process-to-process delivery.

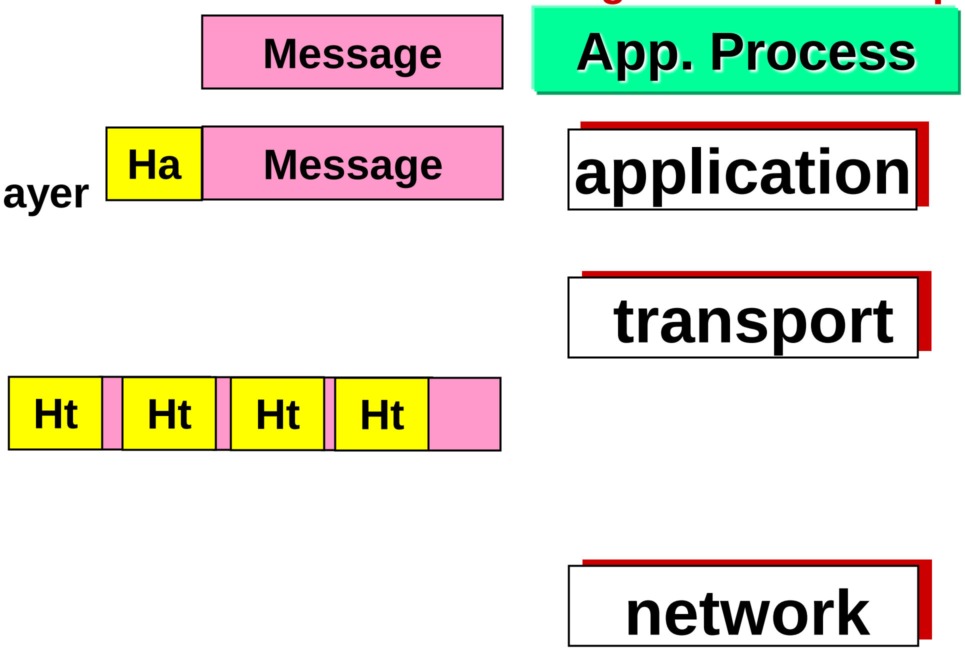
Transport Layer

- provide *logical communication* between application processes running on different hosts
- transport protocols run in end systems
  - sending side: breaks app messages into segments, passes to network layer
  - receiving side: reassembles segments into messages, passes to application layer
- more than one transport protocol available to applications.
  - Internet: TCP and UDP



Protocol layering and data

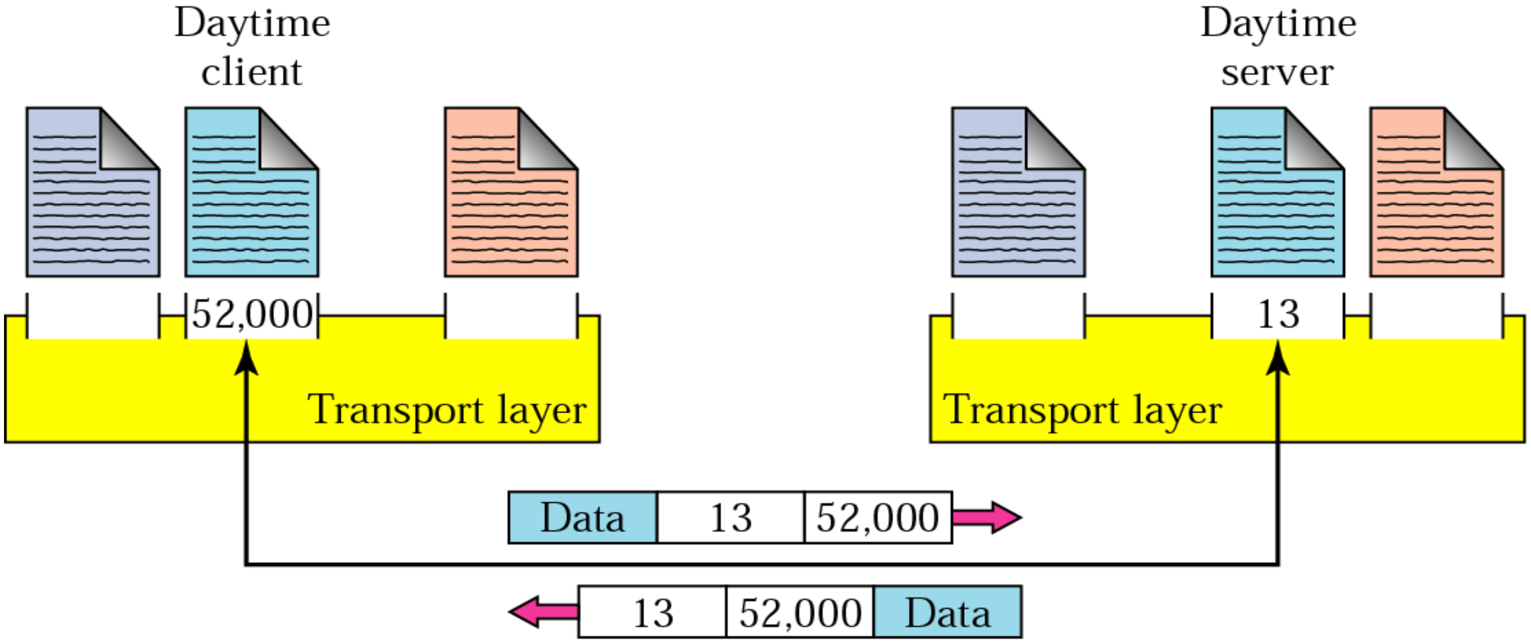
App. Process decides to send
a message to its counterpart



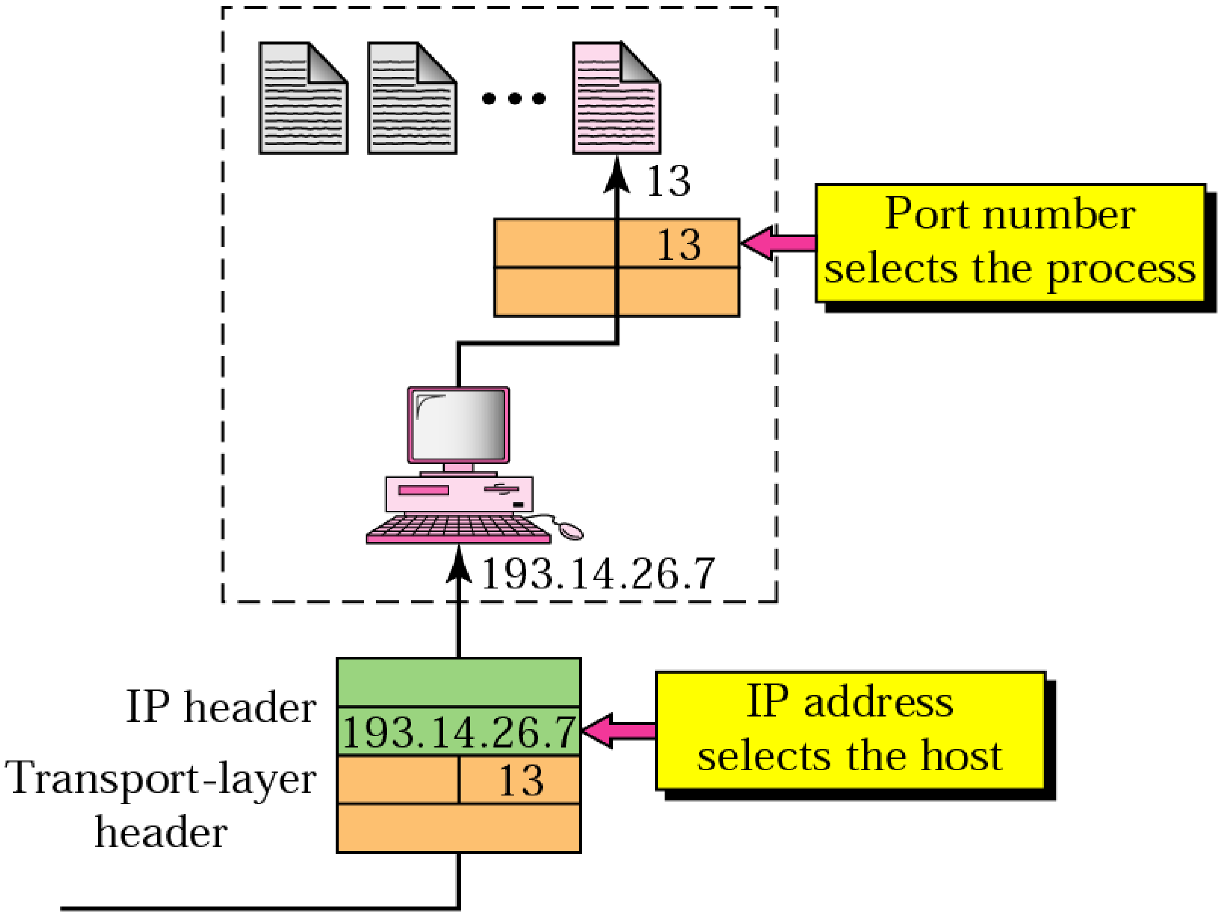
App. Layer adds its header,
sends the message to transport layer

Transport layer breaks down
the message into several parts,
add its header to each part
And makes segments.
It sends one-by-one segments
to network layer

Port numbers



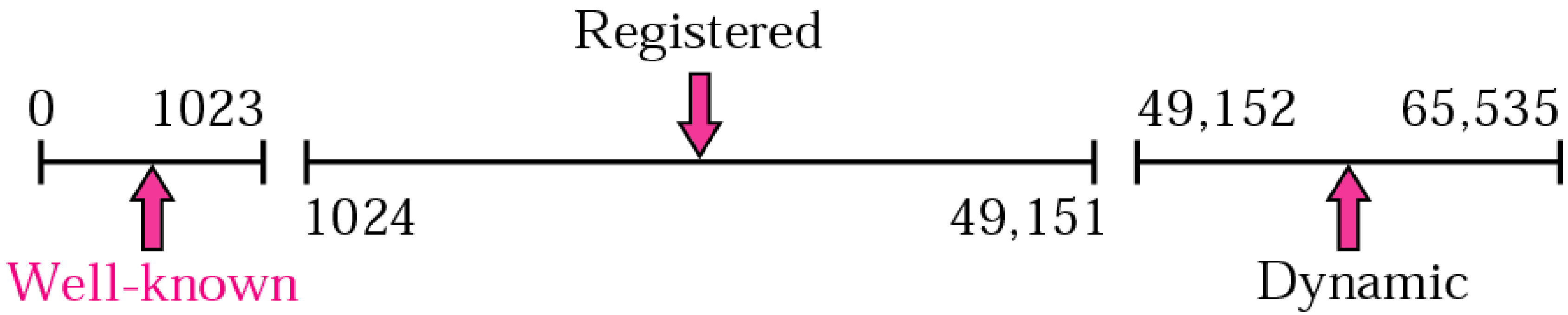
IP addresses versus port numbers



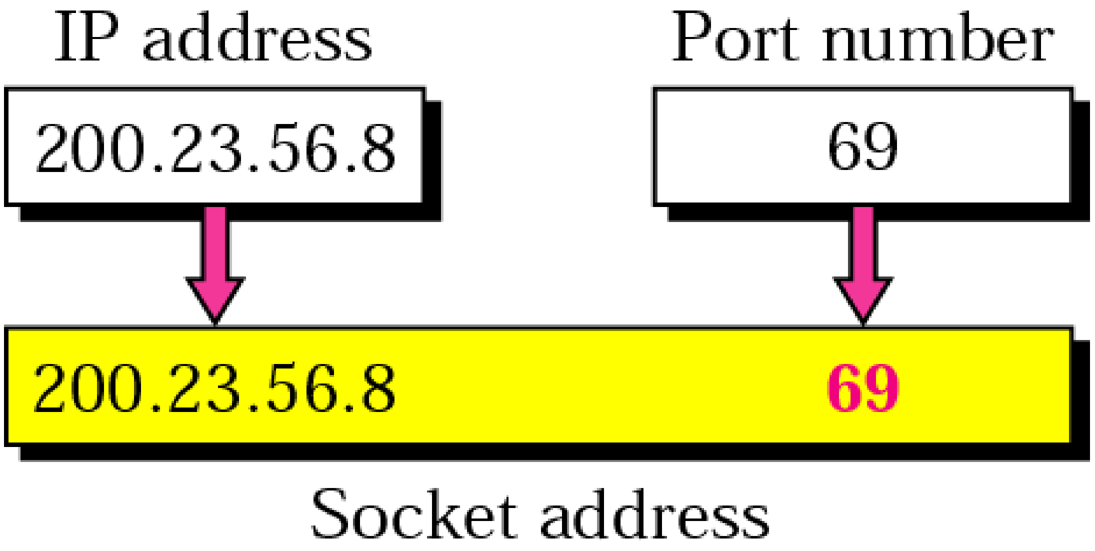
Port number selects the process

IP address selects the host

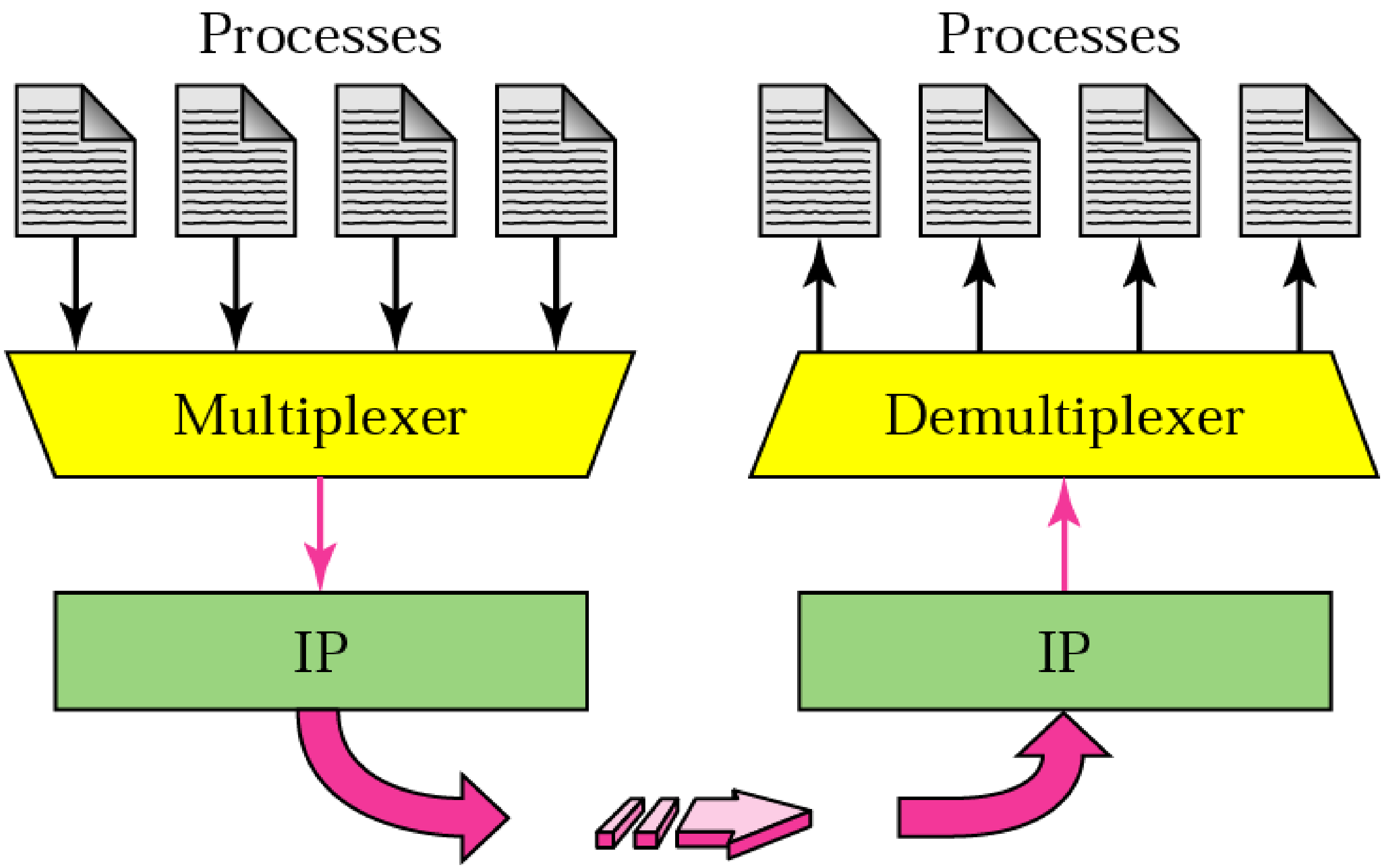
IANA ranges for port numbers



Socket address



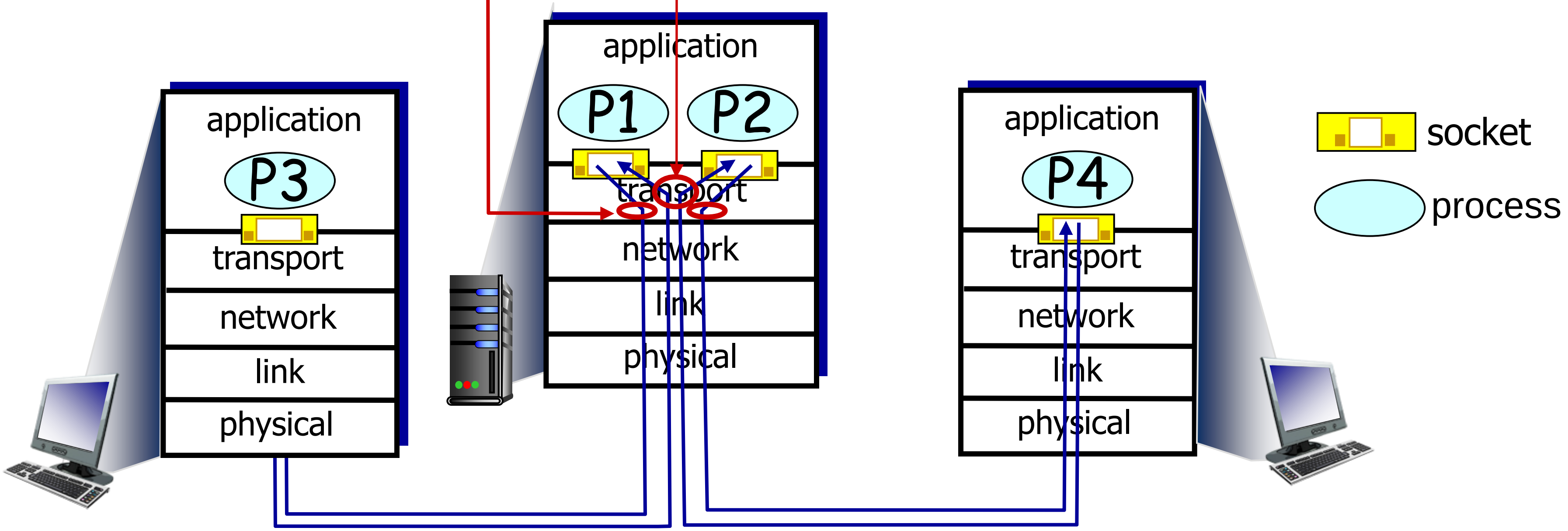
Multiplexing and demultiplexing



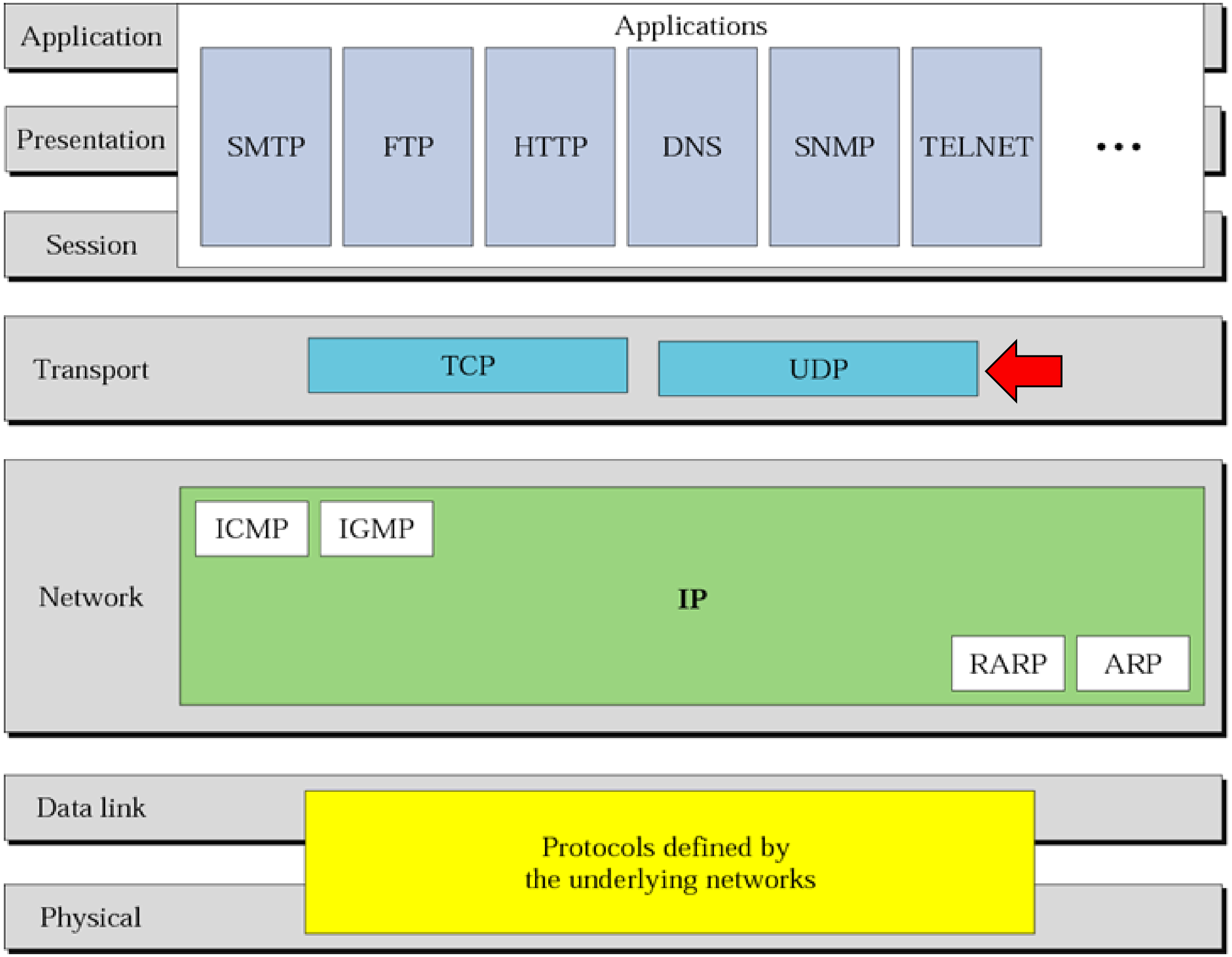
Multiplexing and demultiplexing

multiplexing at sender:
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:
use header info to deliver received segments to correct socket



TCP/IP protocol suite



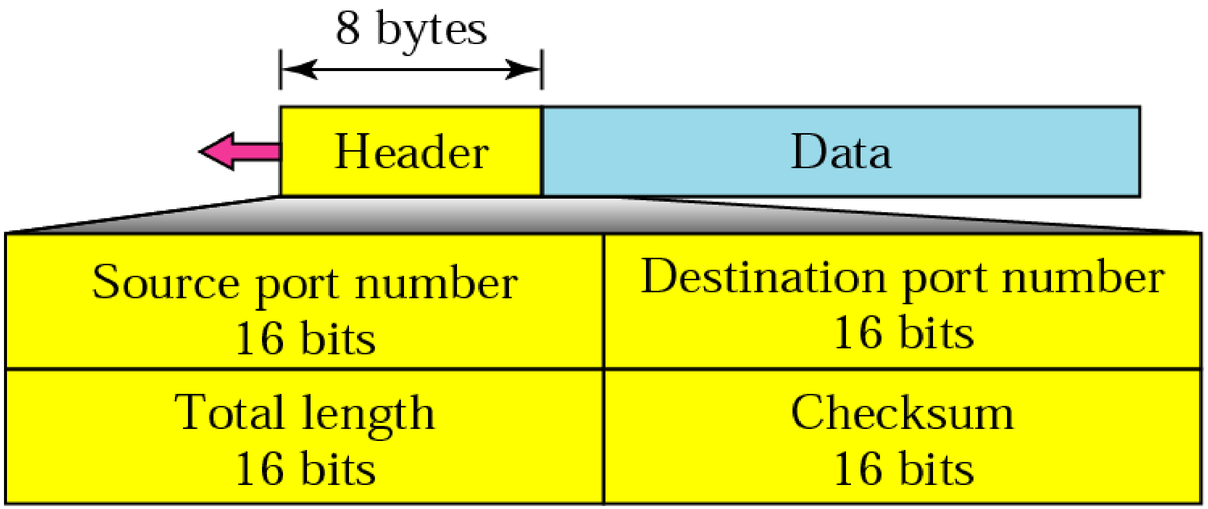
UDP

UDP is a connectionless, unreliable protocol that has no flow and error control. It uses port numbers to multiplex data from the application layer.

Some Well-known ports used by UDP

Port	Protocol	Description
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
53	Nameserver	Domain Name Service
67	Bootps	Server port to download bootstrap information
68	Bootpc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol

User datagram format


8 bytes

Header

Data

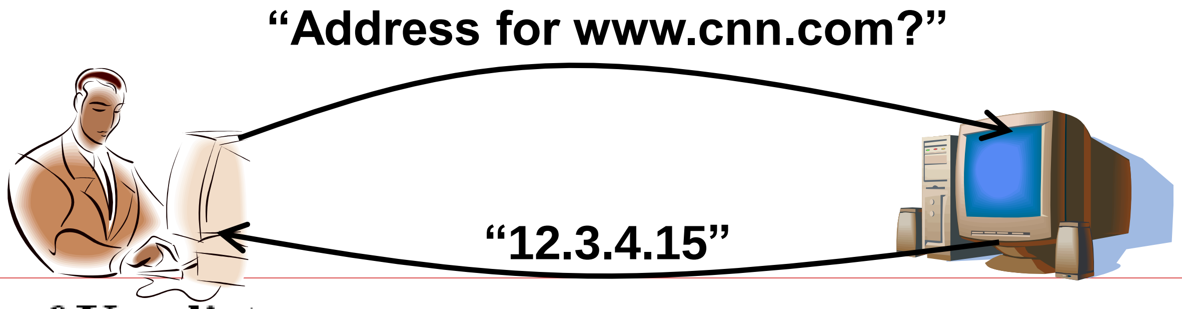
Source port number 16 bits	Destination port number 16 bits
Total length 16 bits	Checksum 16 bits

The calculation of checksum and its inclusion in the user datagram are optional.

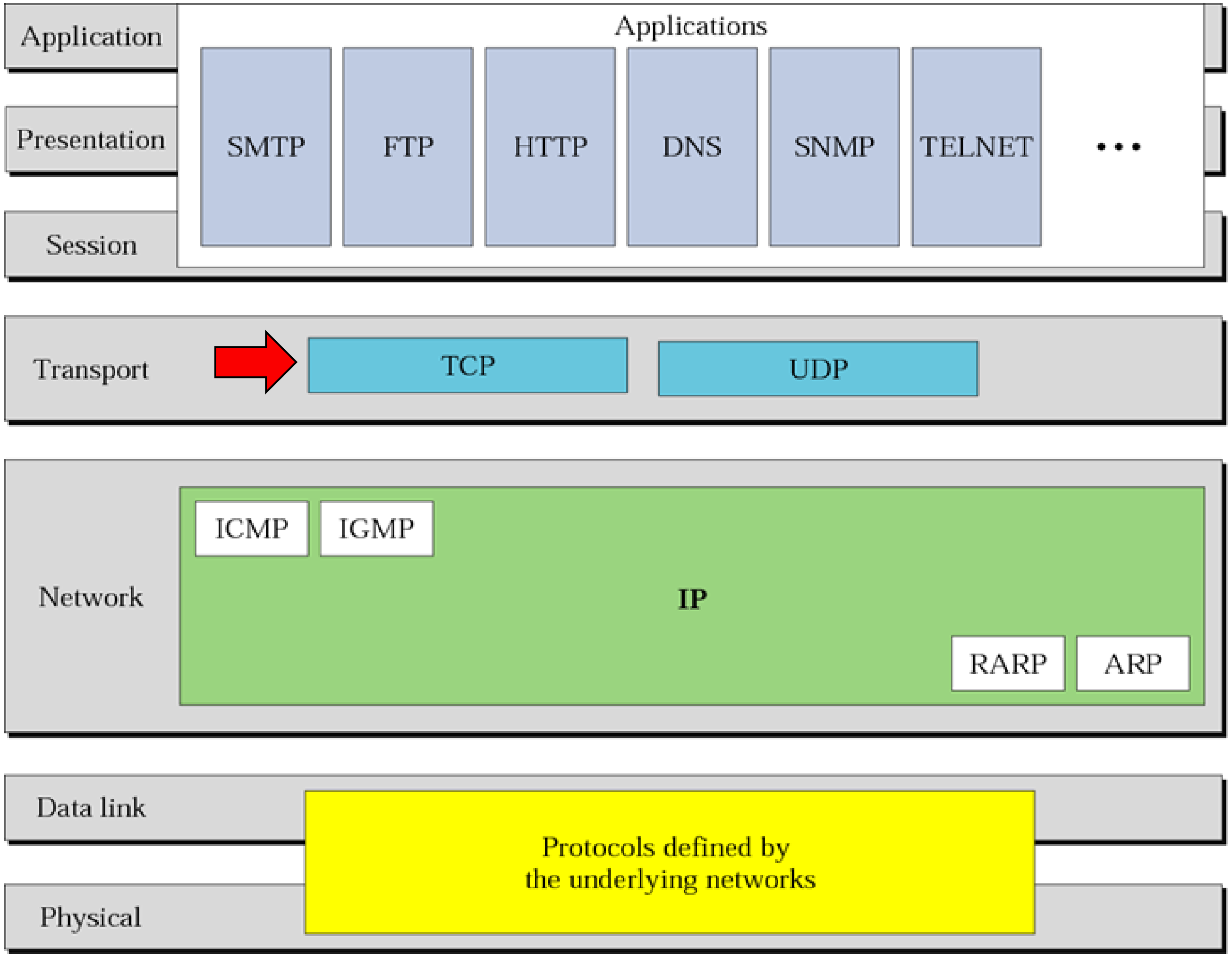
Popular Applications That Use UDP

- **Multimedia streaming**
  - Retransmitting lost/corrupted packets is not worthwhile
  - By the time the packet is retransmitted, it's too late
  - E.g., telephone calls, video conferencing, gaming

- **Simple query protocols like Domain Name System**
  - Overhead of connection establishment is overkill
  - Easier to have the application retransmit if needed



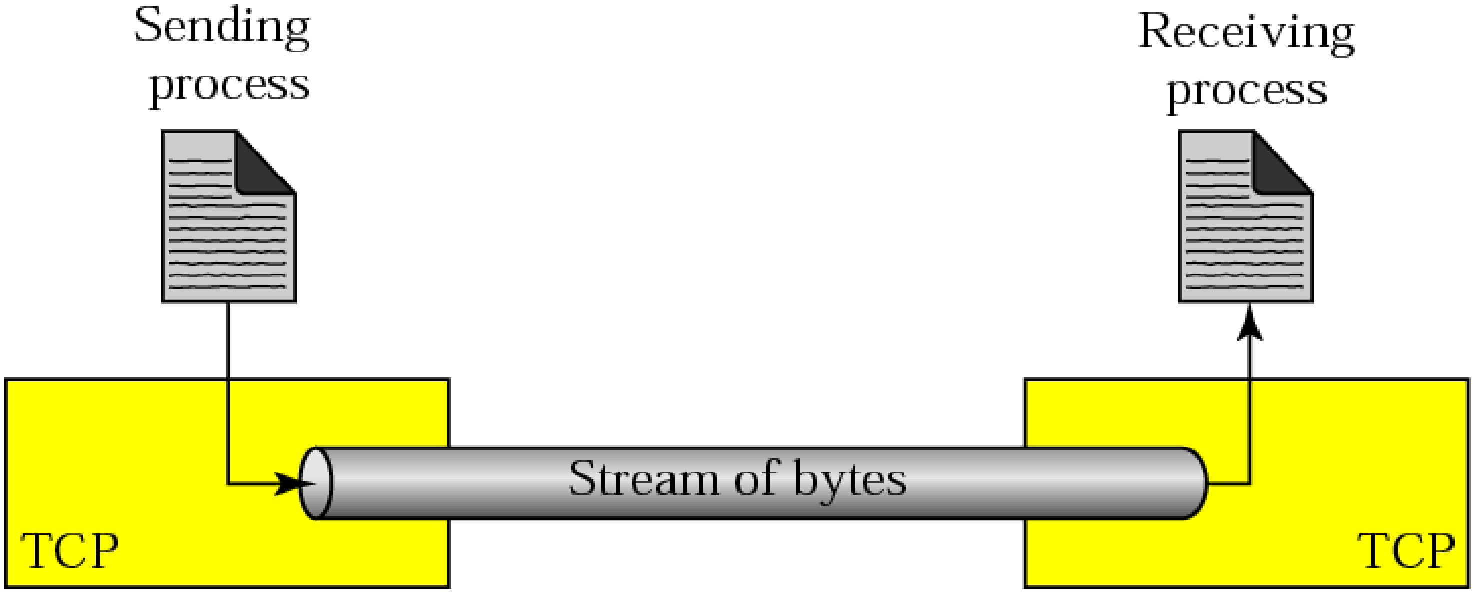
TCP/IP protocol suite



Transmission Control Protocol (TCP)

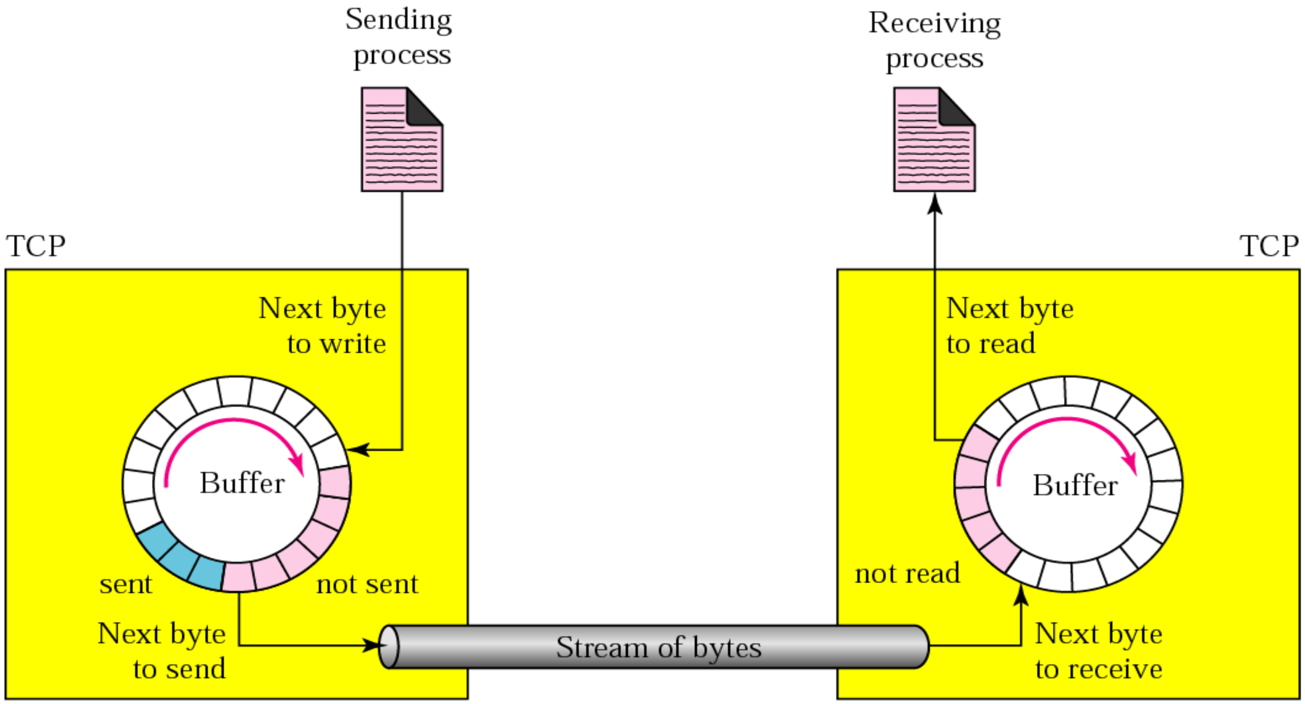
- **Connection oriented**
  - Explicit set-up and tear-down of TCP session
- **Stream-of-bytes service**
  - Sends and receives a stream of bytes, not messages
- **Reliable, in-order delivery**
  - Checksums to detect corrupted data
  - Acknowledgments & retransmissions for reliable delivery
  - Sequence numbers to detect losses and reorder data
- **Flow control**
  - Prevent overflow of the receiver’s buffer space
- **Congestion control**
  - Adapt to network congestion for the greater good

Stream delivery



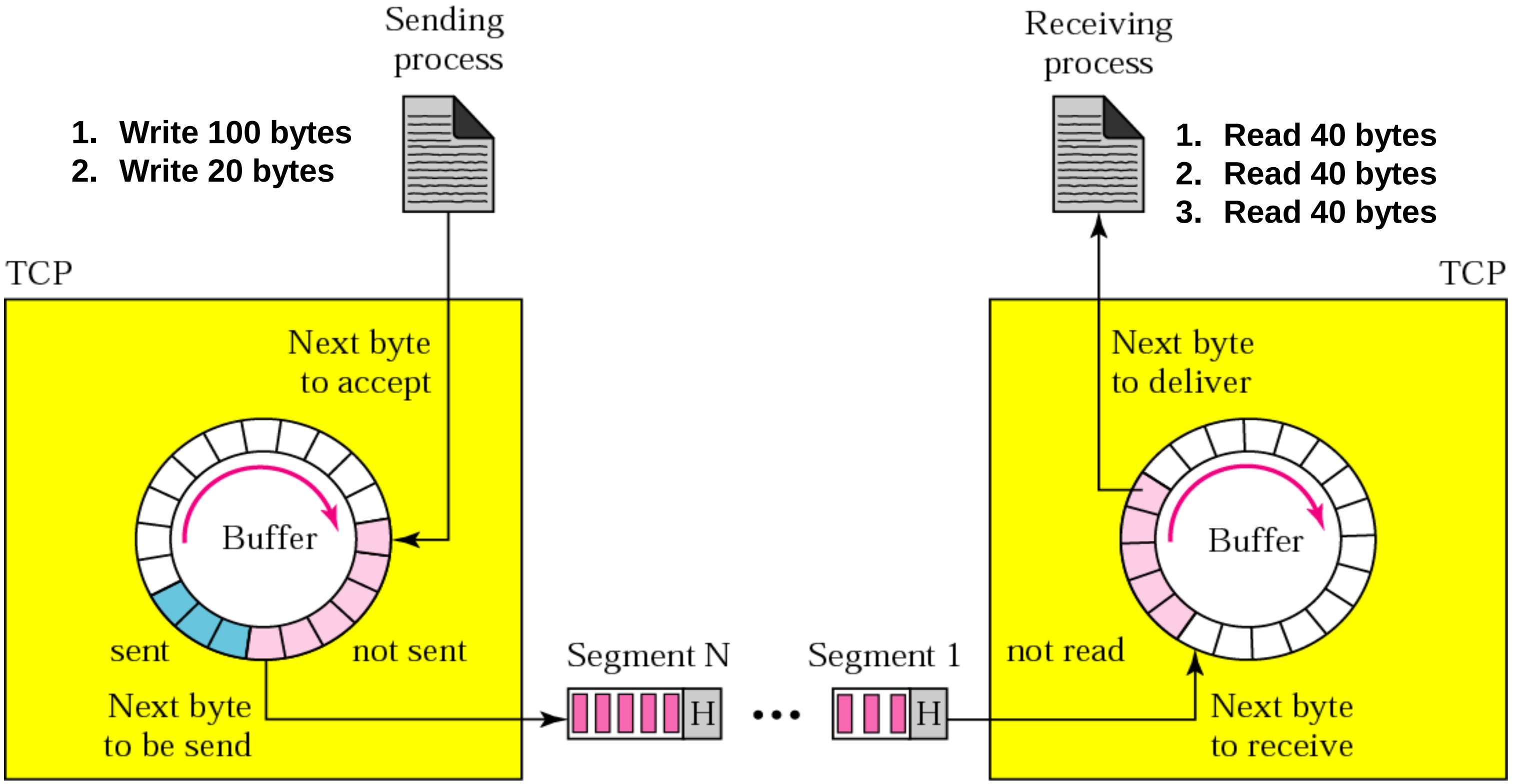
TCPمحیطی را فراهم می آورد که گویی دو پروسه به وسیله یک
لوله فرضی به همدیگر متصل شده اند

Sending and receiving buffers



TCPدر دو طرف فرستنده و گیرنده یک بافر برای ارسال و دریافت دارد

TCP segments



Sending process

1. Write 100 bytes
2. Write 20 bytes

Receiving process

1. Read 40 bytes
2. Read 40 bytes
3. Read 40 bytes

TCP

Next byte to accept

Buffer

sent

not sent

Next byte to be send

Segment N

Segment 1

TCP

Next byte to deliver

Buffer

not read

Next byte to receive

TCPجریانی از بایتها را در قالب سگمنت ها ارسال می کند
لایه بالاتر چیزی در موردسگمنتها و محدوده آنها نمیداند

Note:

The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with a randomly generated number.

Example of Byte numbering and sequence numbers

Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 10001. What are the sequence numbers for each segment if data is sent in five segments, each carrying 1000 bytes?

Solution

The following shows the sequence number for each segment:

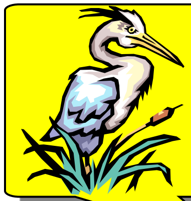
Segment 1 ➡ Sequence Number: 10,001 (range: 10,001 to 11,000)

Segment 2 ➡ Sequence Number: 11,001 (range: 11,001 to 12,000)

Segment 3 ➡ Sequence Number: 12,001 (range: 12,001 to 13,000)

Segment 4 ➡ Sequence Number: 13,001 (range: 13,001 to 14,000)

Segment 5 ➡ Sequence Number: 14,001 (range: 14,001 to 15,000)



Note:

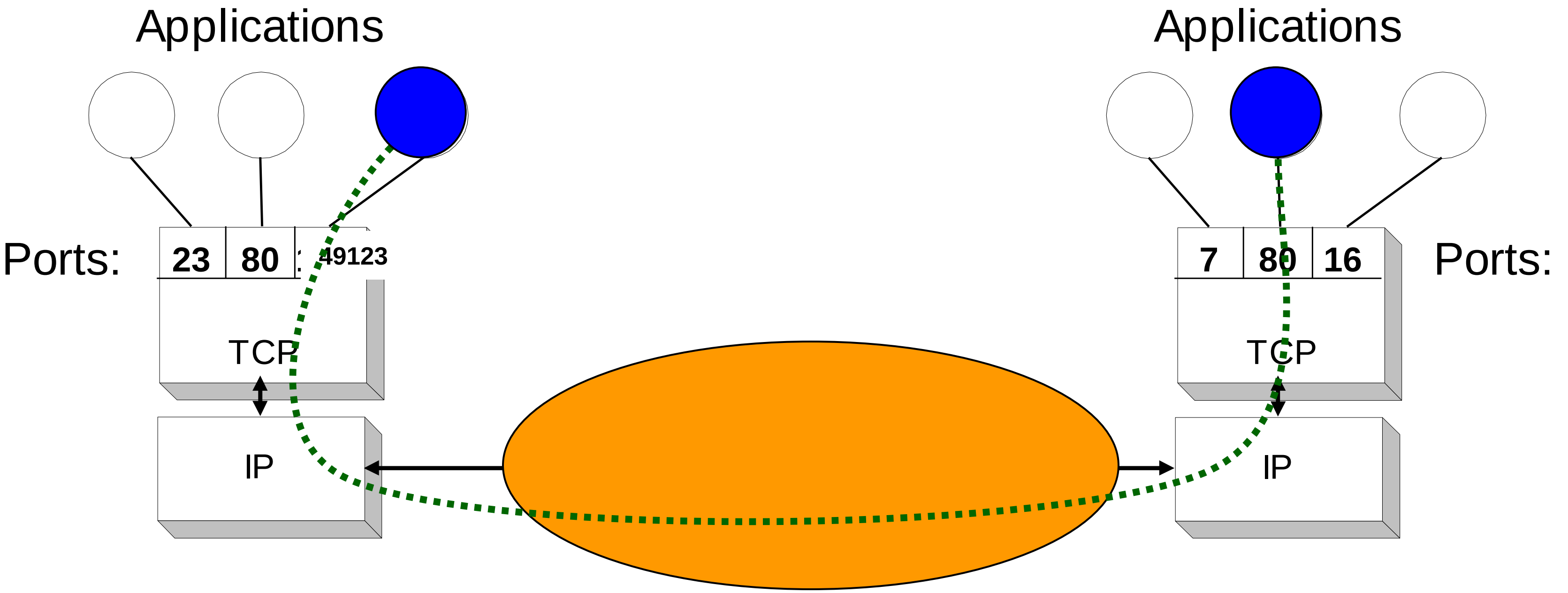
The value in the sequence number field of a segment defines the number of the first data byte contained in that segment.

Note:

The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive.

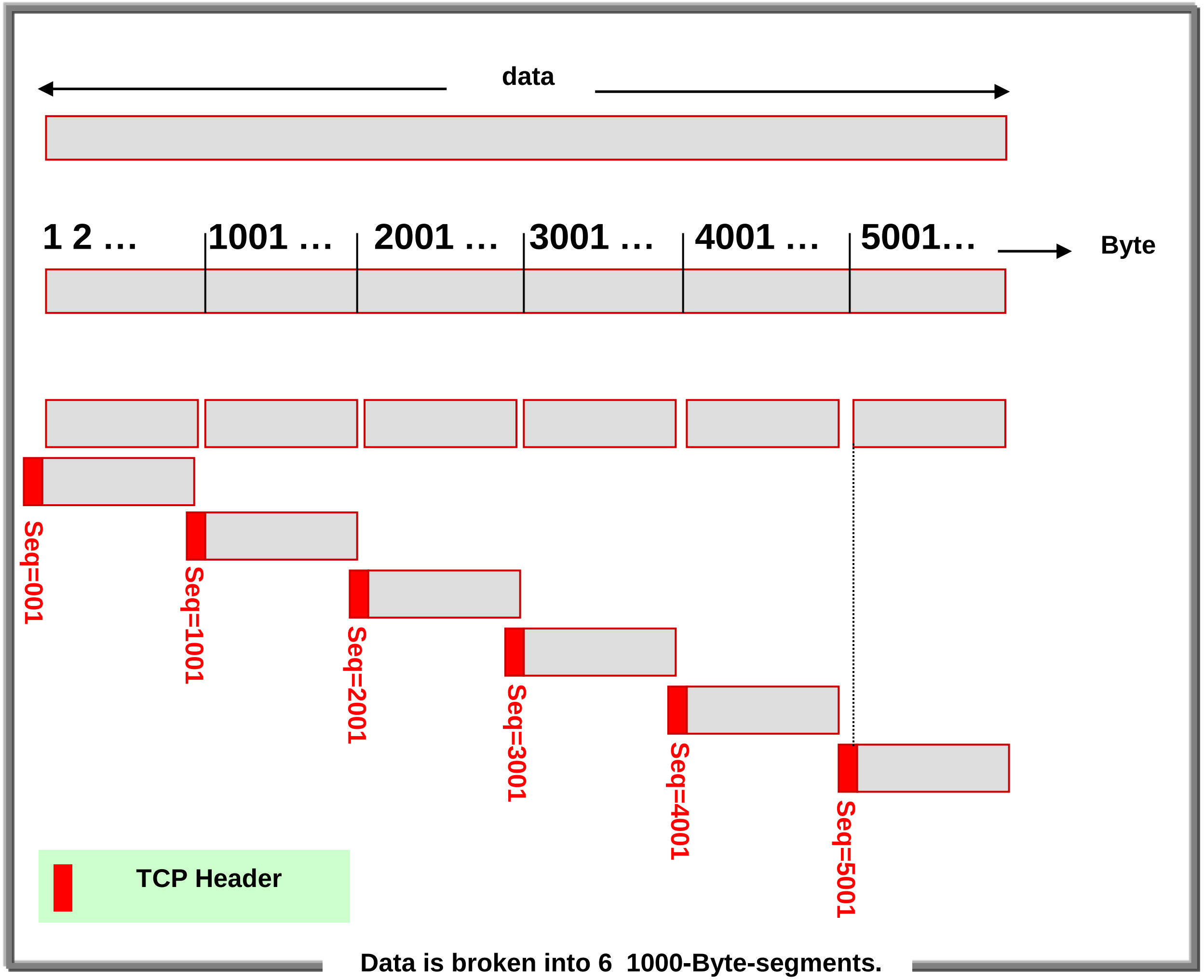
The acknowledgment number is cumulative.

TCP connection



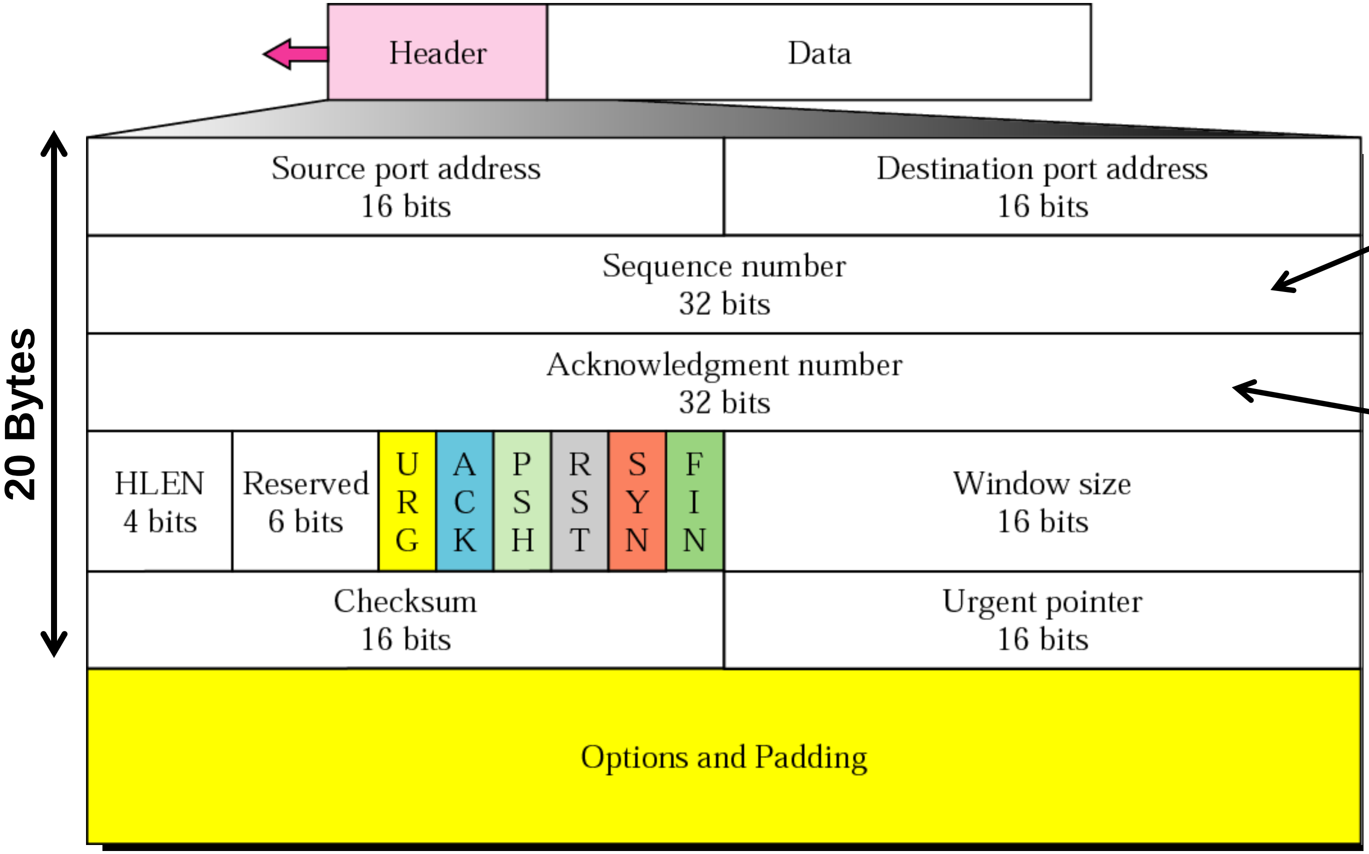
A pair <IP address, port number> **identifies one endpoint of a connection.**

Two pairs <client IP address, server port number> **and** <server IP address, server port number> **identify a TCP connection.**



Data is broken into 6 1000-Byte-segments.

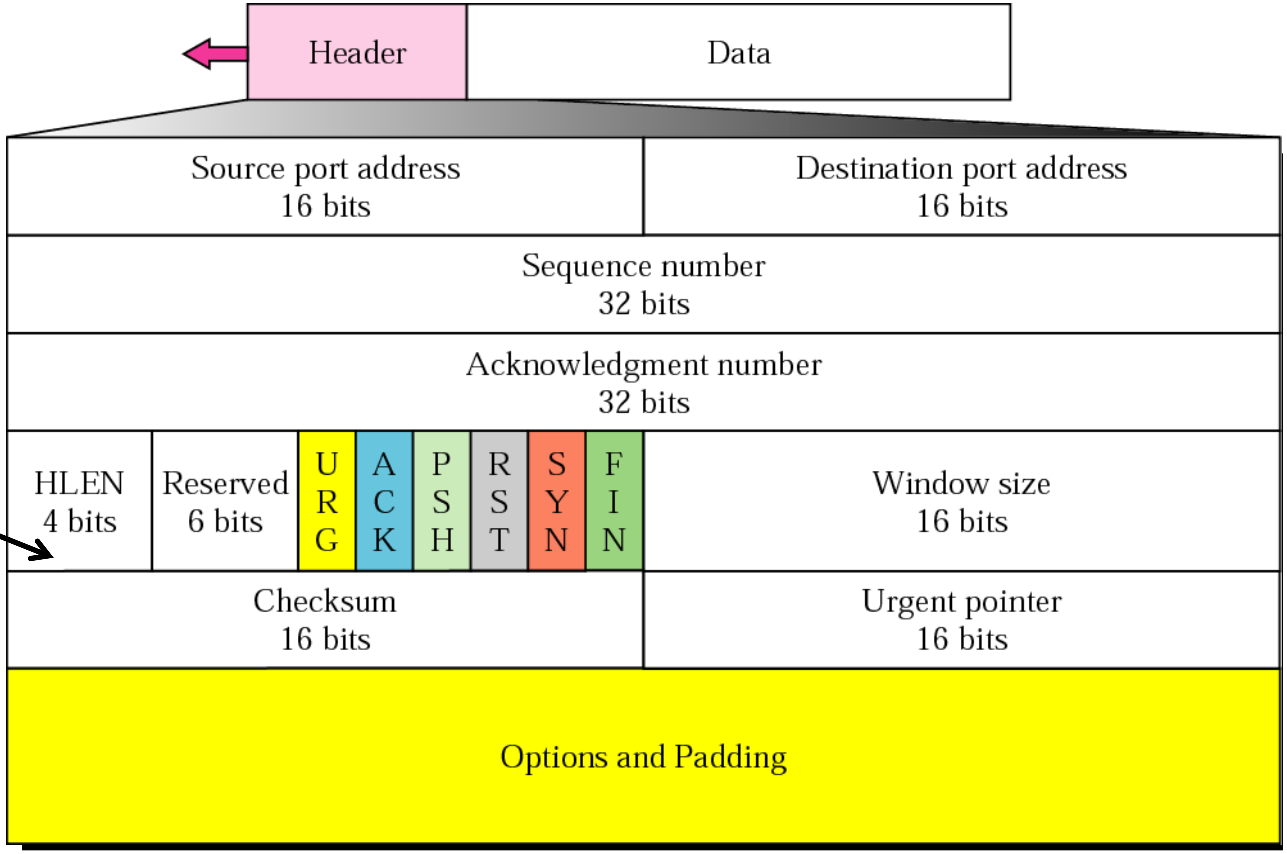
TCP Header



شماره ترتیب اولین بایتی
که در قسمت داده قرار
دارد

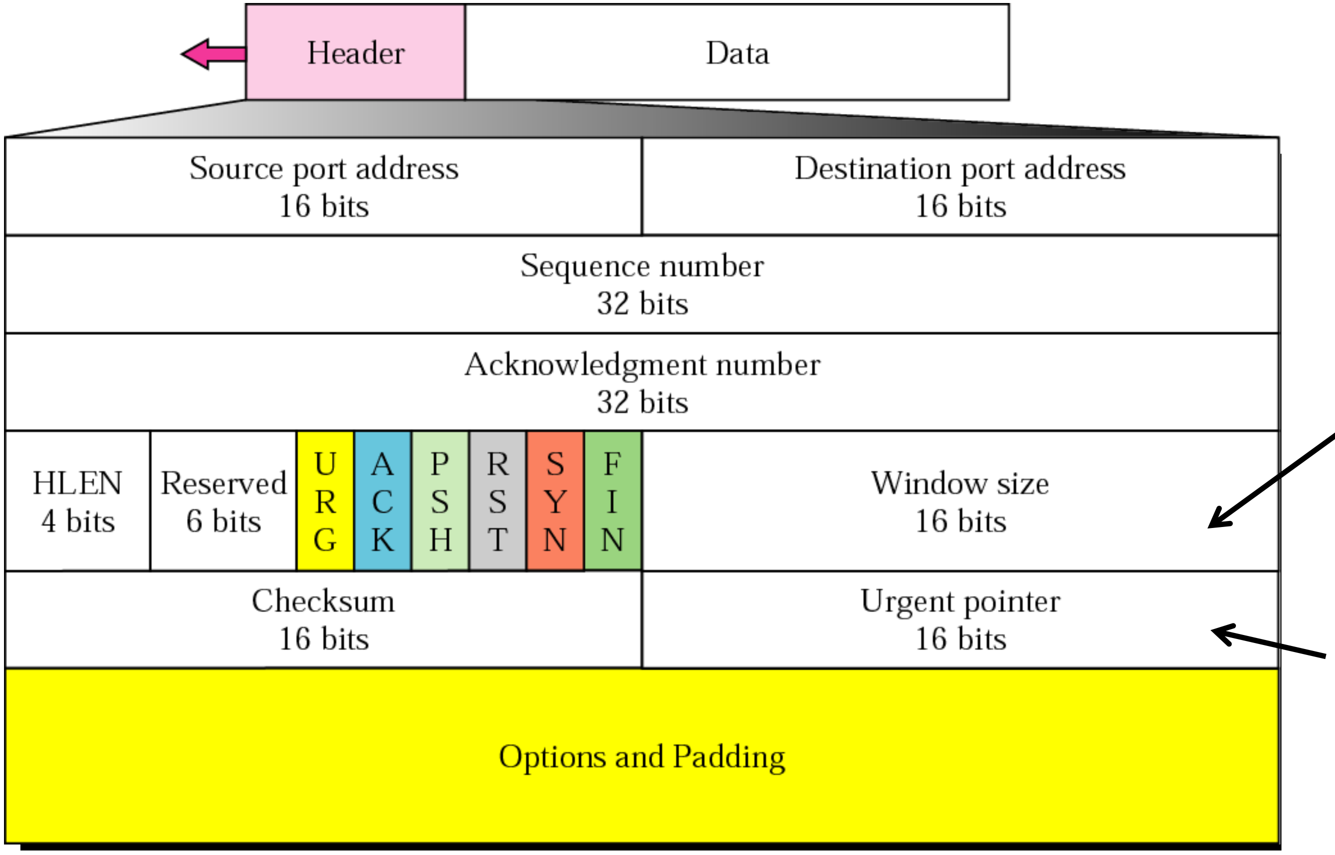
شماره ترتیب بایتی که
منتظر دریافت آن است

TCP Header



طول هدر بر حسب کلمه ٤ بایتی

TCP Header



Header

Data

Source port address
16 bits

Destination port address
16 bits

Sequence number
32 bits

Acknowledgment number
32 bits

HLEN
4 bits

Reserved
6 bits

URG

ACK

PSH

RST

SYN

FIN

Window size
16 bits

Checksum
16 bits

Urgent pointer
16 bits

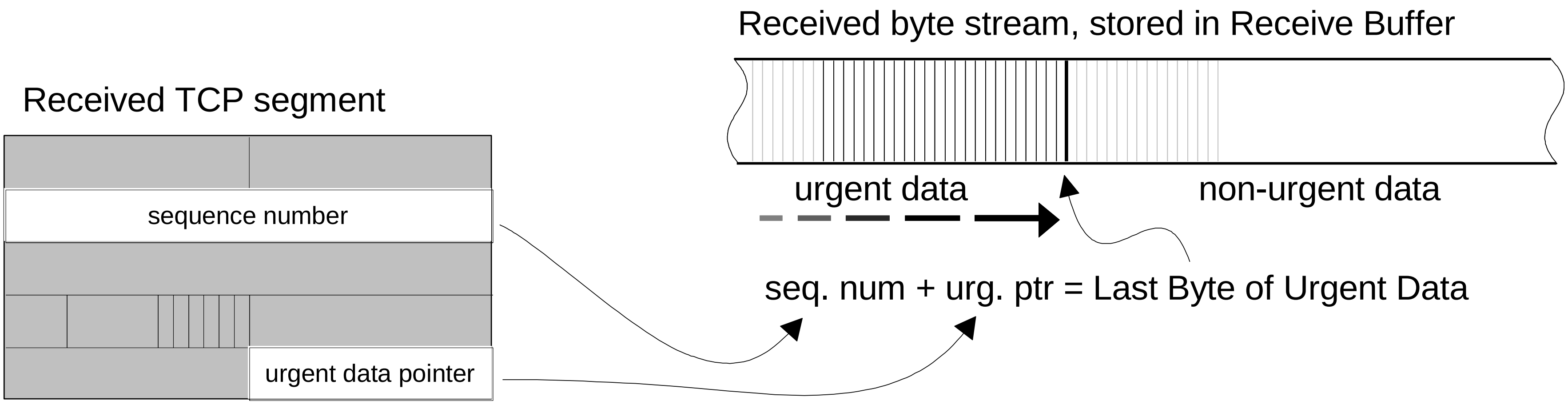
Options and Padding

اندازه پنجره دریافت- مورد
استفاده در مکانیسم کنترل جریان

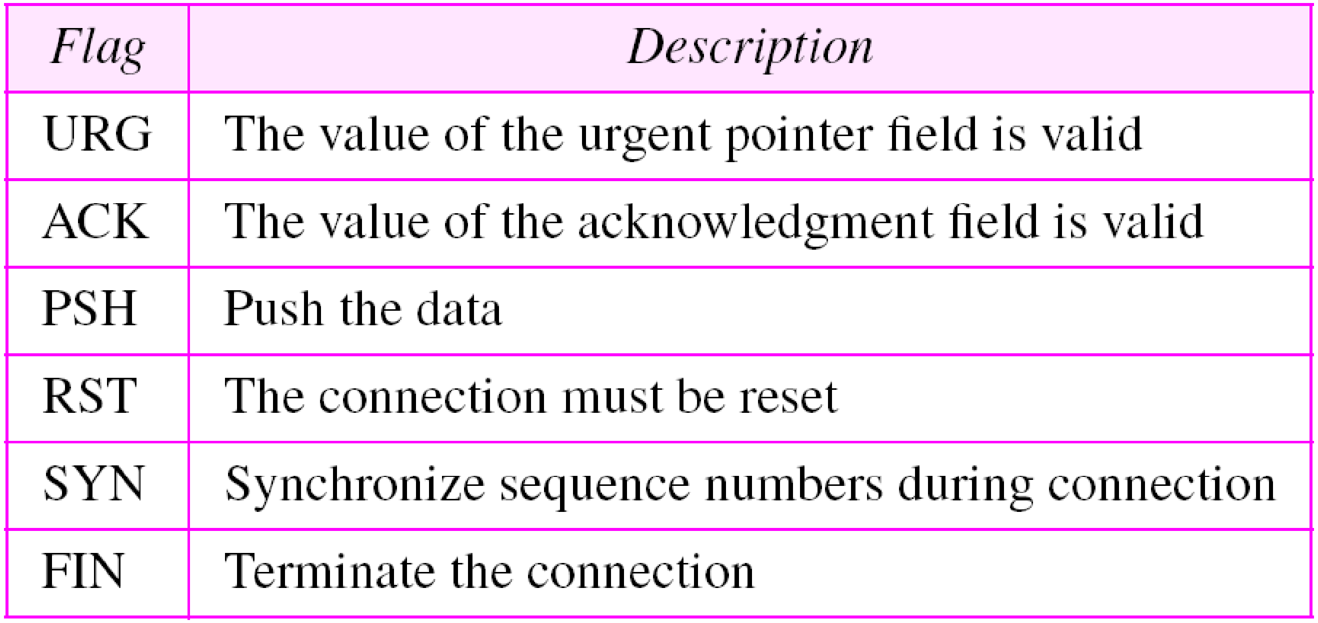
این فیلد به عنوان یک اشاره گر
موقعیت داده های اضطراری را
درون سگمنت معین میکند

TCP Header - Urgent Data Pointer

- Last byte of urgent data (LBUD) = sequenceNumber + urgentPointer
- First byte of urgent data never explicitly defined
- Any data in Receive buffer up to LBUD may be considered urgent


Received byte stream, stored in Receive Buffer
Received TCP segment
sequence number
urgent data pointer
urgent data
non-urgent data
seq. num + urg. ptr = Last Byte of Urgent Data

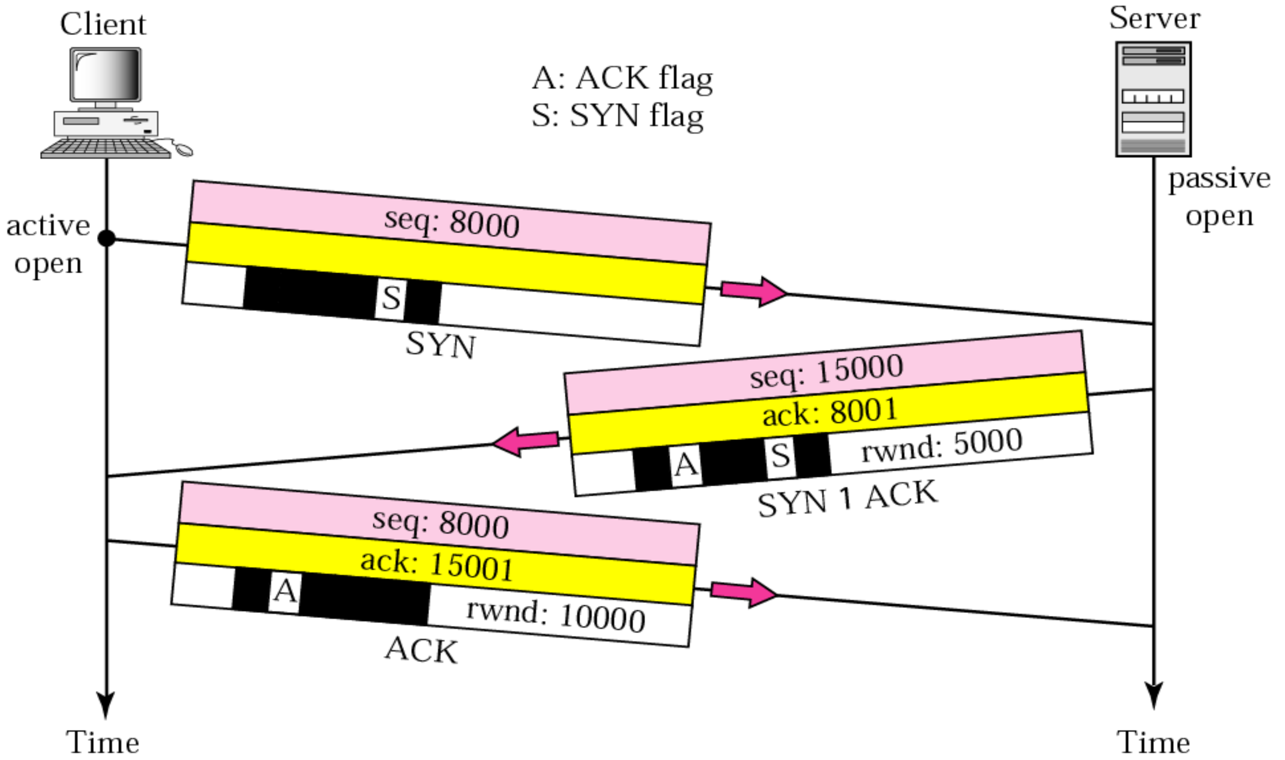
Description of flags in the control field



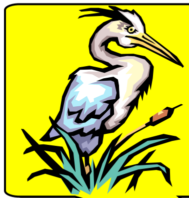
Flag	*Description*
URG	The value of the urgent pointer field is valid
ACK	The value of the acknowledgment field is valid
PSH	Push the data
RST	The connection must be reset
SYN	Synchronize sequence numbers during connection
FIN	Terminate the connection

بیتهای **SYNو FIN و ACK** برای برقراری و قطع اتصال استفاده می شوند

Connection establishment using *"three-way handshaking"*



A SYN segment cannot carry data, but it consumes one sequence number.

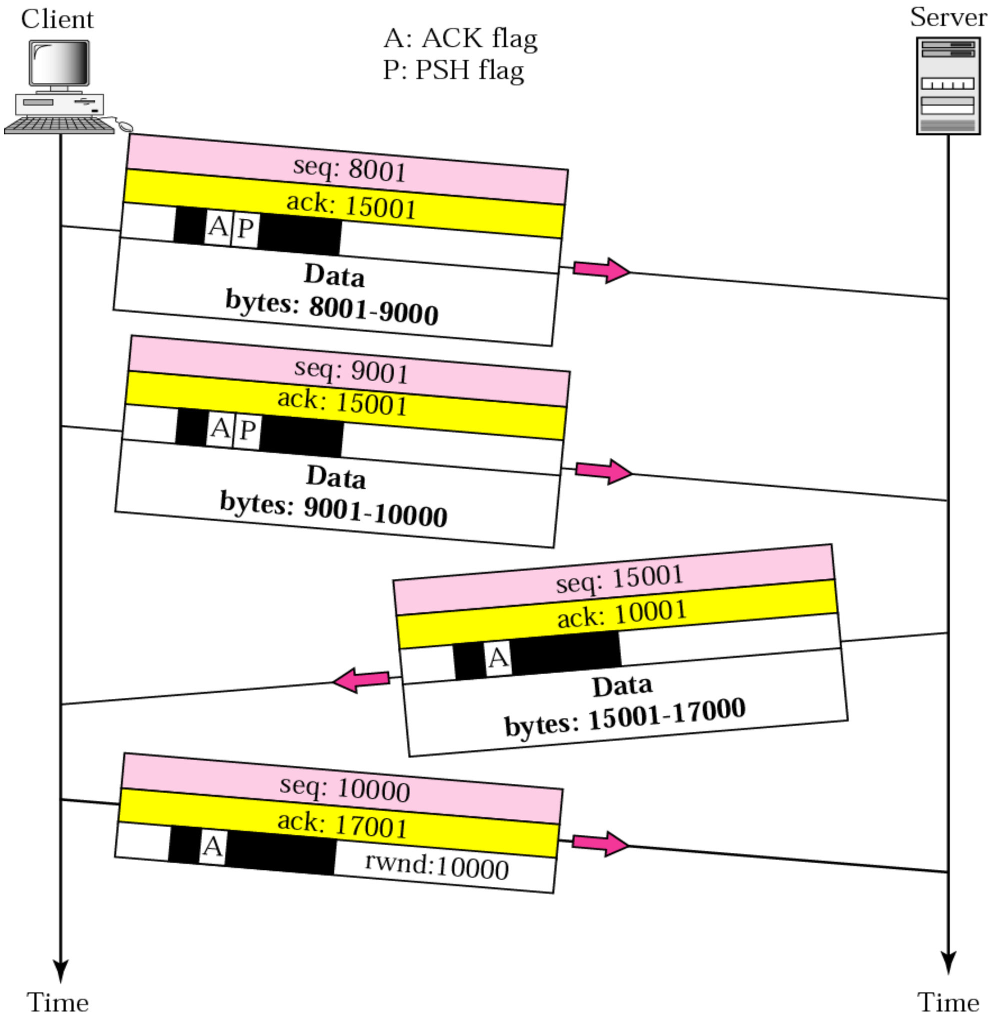


Note:

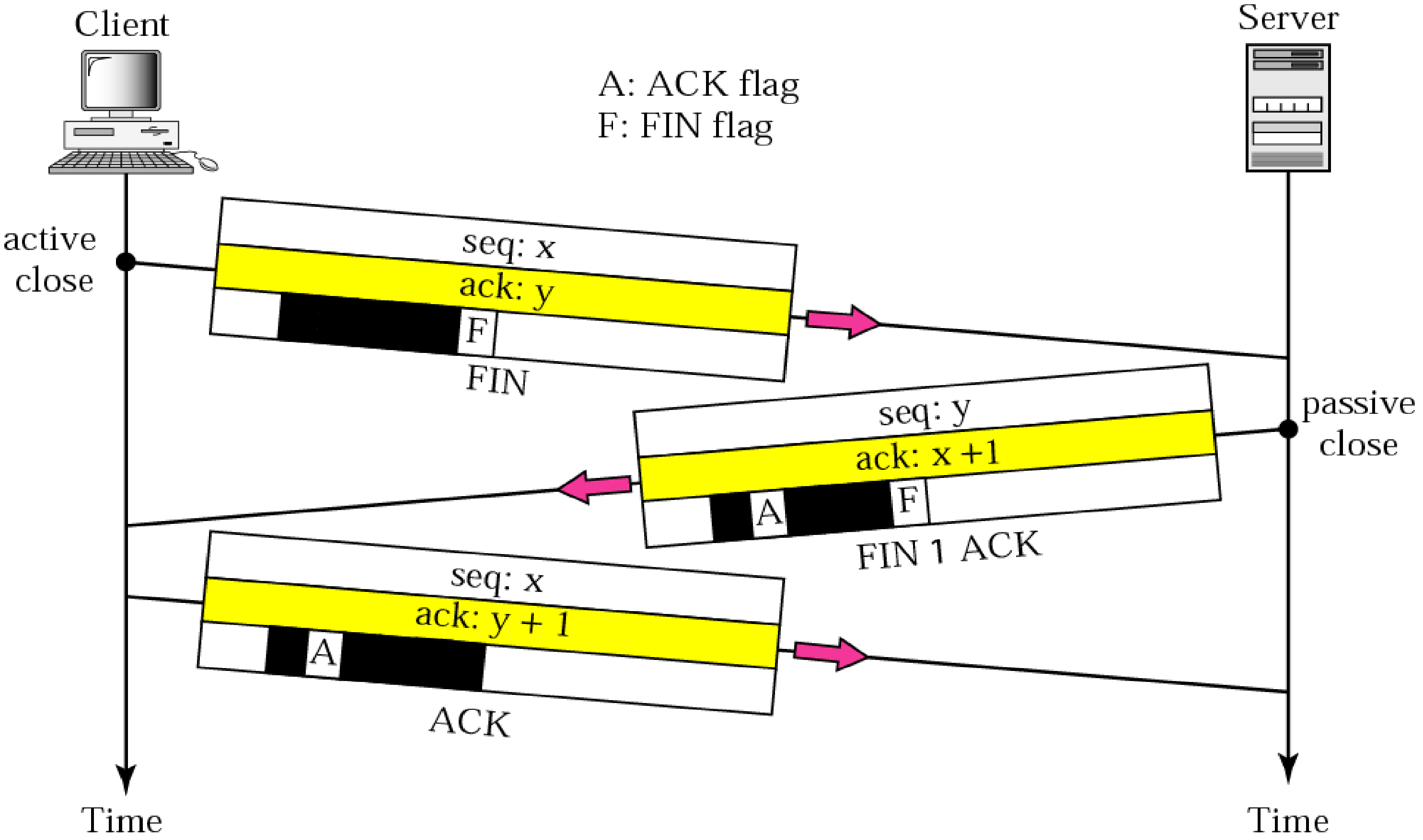
A SYN + ACK segment cannot carry data, but does consume one sequence number.

An ACK segment, if carrying no data, consumes no sequence number.

Data transfer

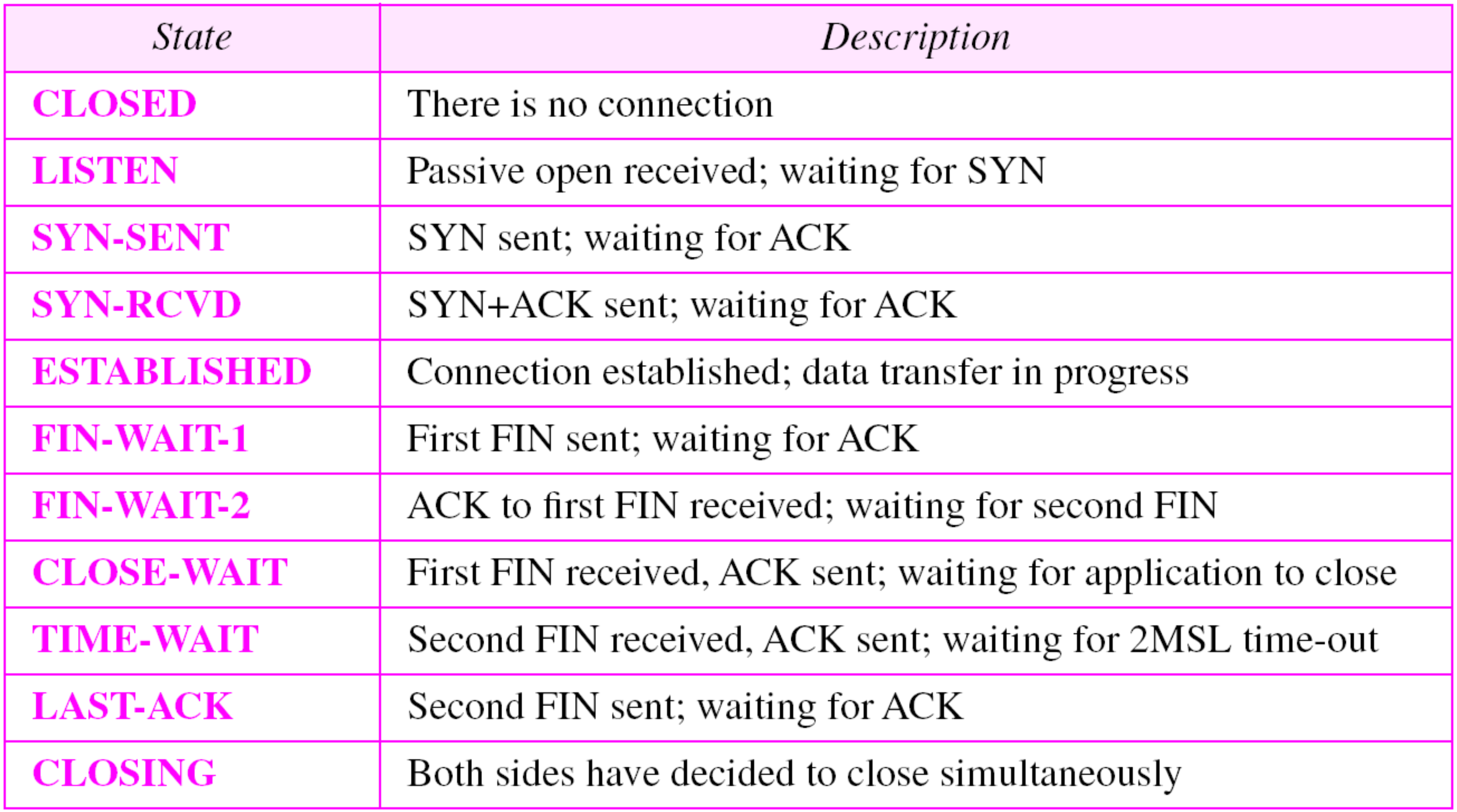


Connection termination



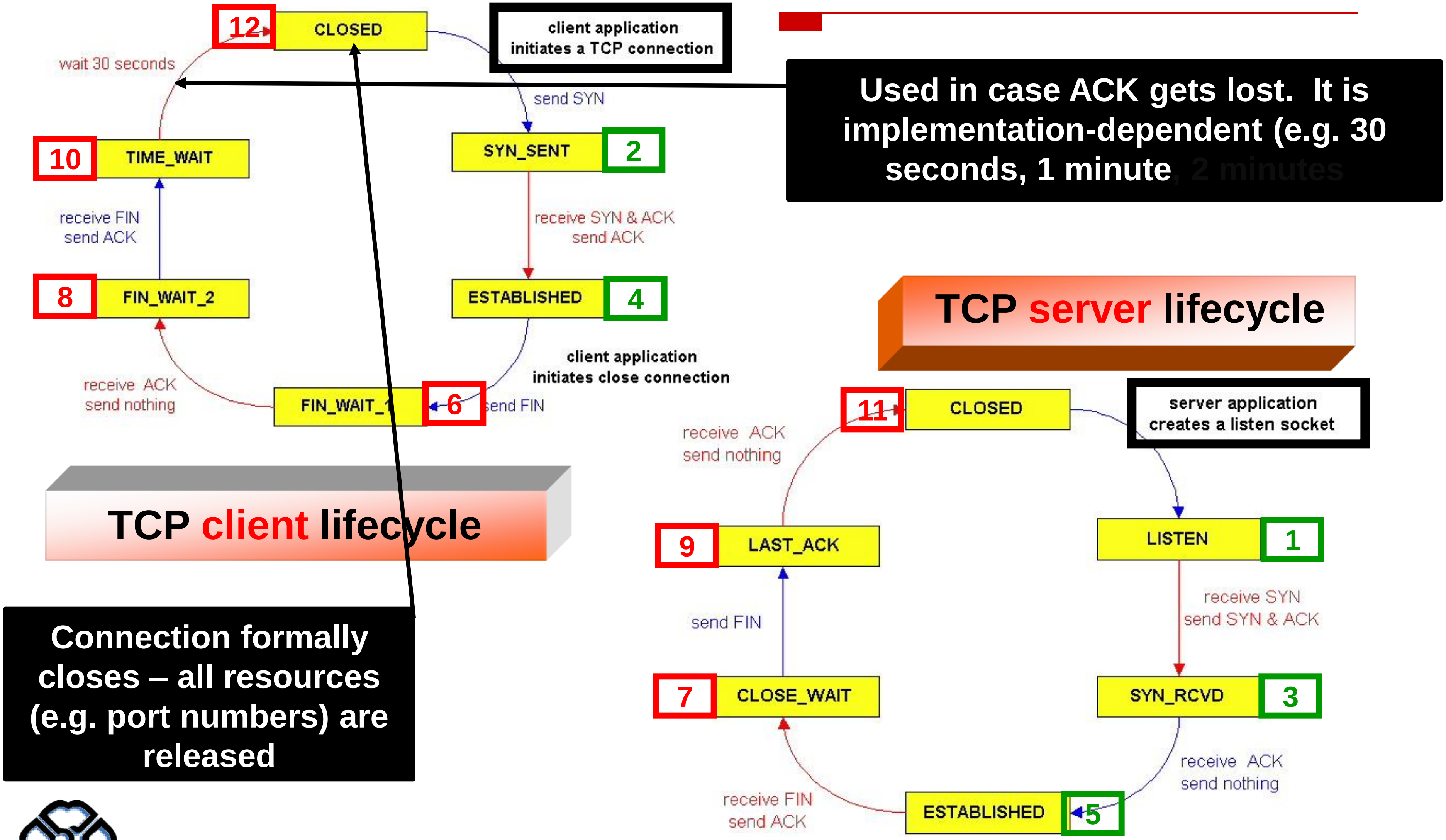
The FIN and (FIN+ACK) segments consume one sequence number if they do not carry data.

States for TCP



State	*Description*
CLOSED	There is no connection
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN+ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN received; waiting for second FIN
CLOSE-WAIT	First FIN received, ACK sent; waiting for application to close
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out
LAST-ACK	Second FIN sent; waiting for ACK
CLOSING	Both sides have decided to close simultaneously

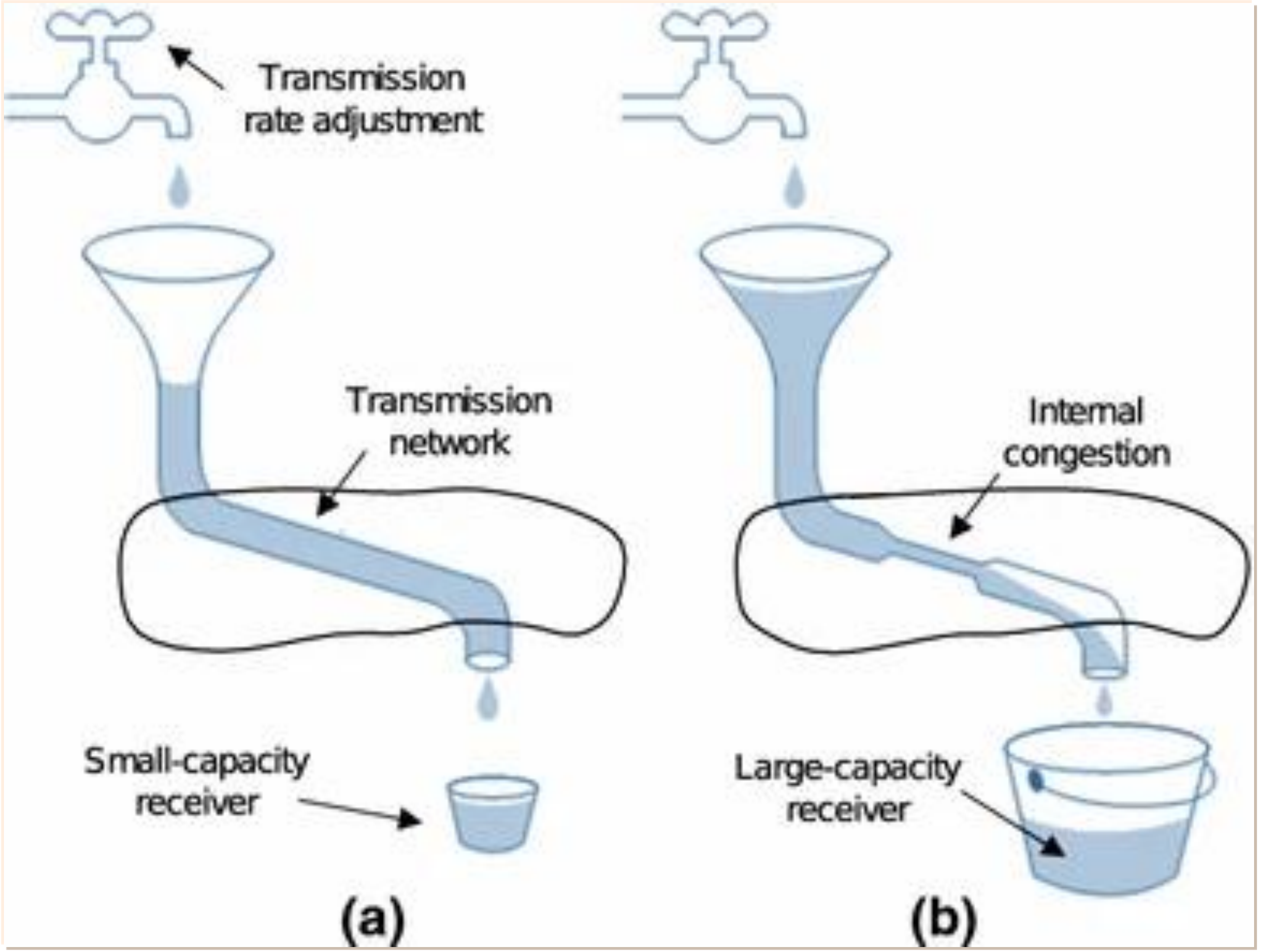
TCP states



Used in case ACK gets lost. It is implementation-dependent (e.g. 30 seconds, 1 minute, 2 minutes

Connection formally closes – all resources (e.g. port numbers) are released

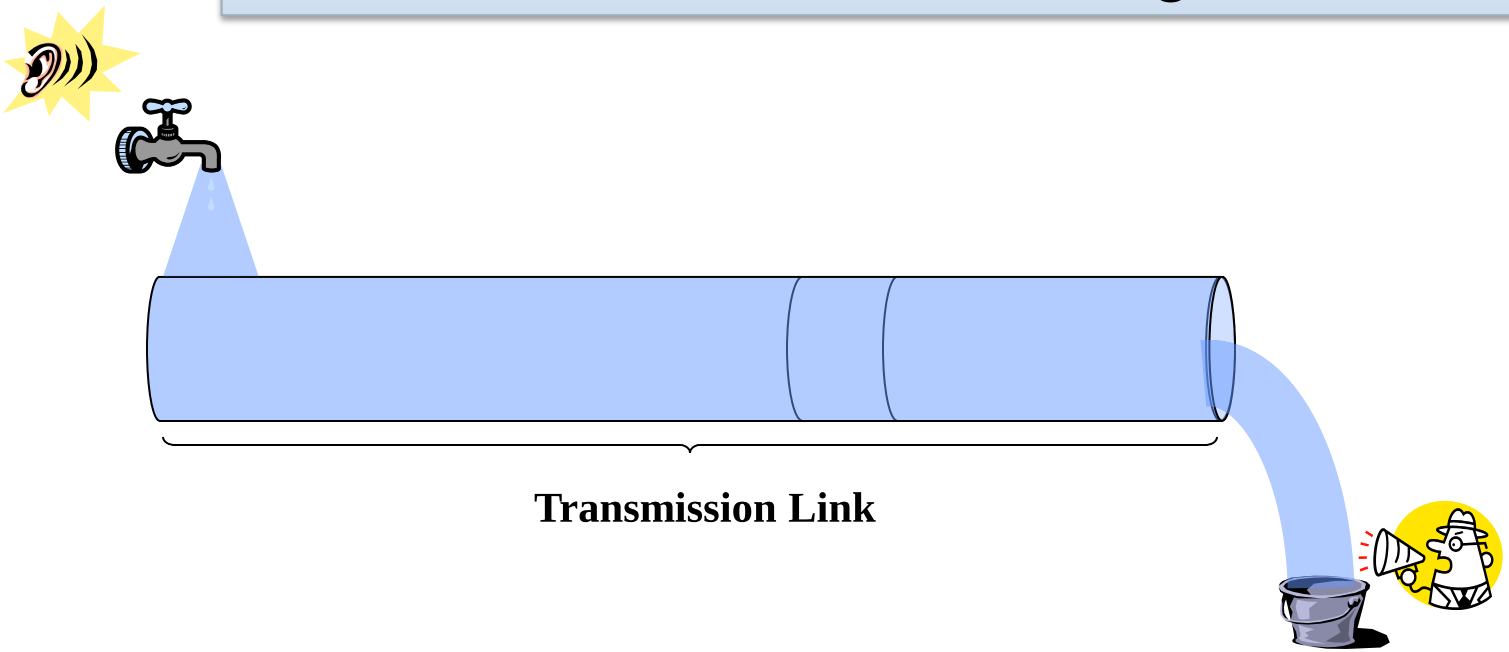
TCP flow control and congestion control



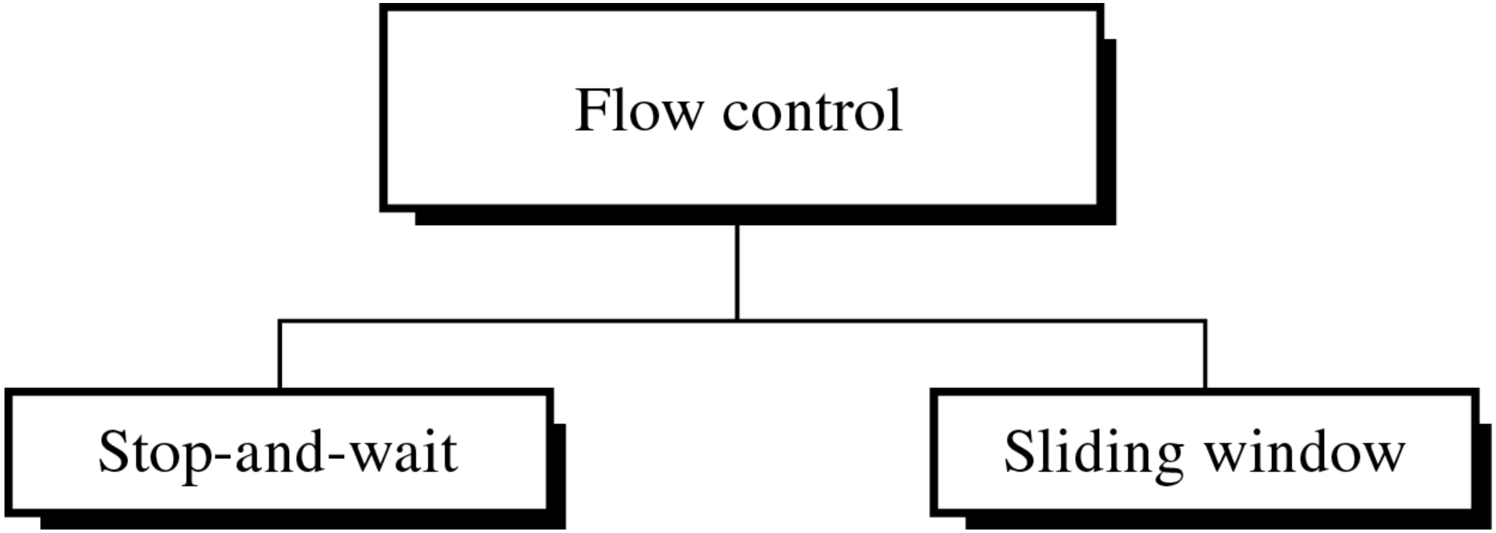
Flow control Congestion control

Flow Control

The process of managing the rate of data transmission between two nodes to prevent a fast sender from overwhelming a slow receiver



Categories of Flow Control



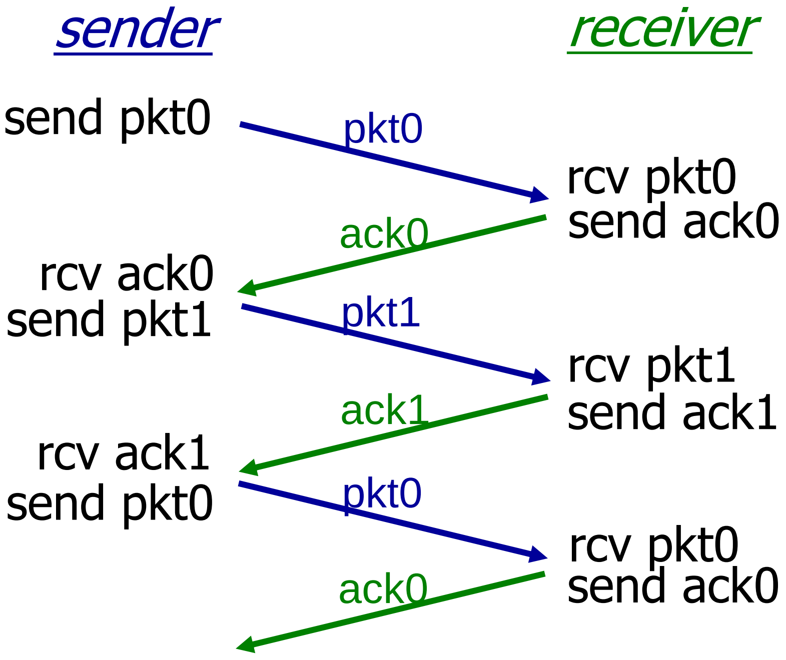
Send one frame at a time

Send several frames at a time

Stop-and-Wait Automatic Repeat reQuest

- Simplest flow and error control mechanism
- The sending device keeps a copy of the last frame transmitted until it receives an acknowledgement
  - identification of duplicate transmission (lost or delayed **ACK**)
- A damaged or lost frame is treated in the same way
- **Timers** introduced
- Positive ACK sent only for frames received safe & sound

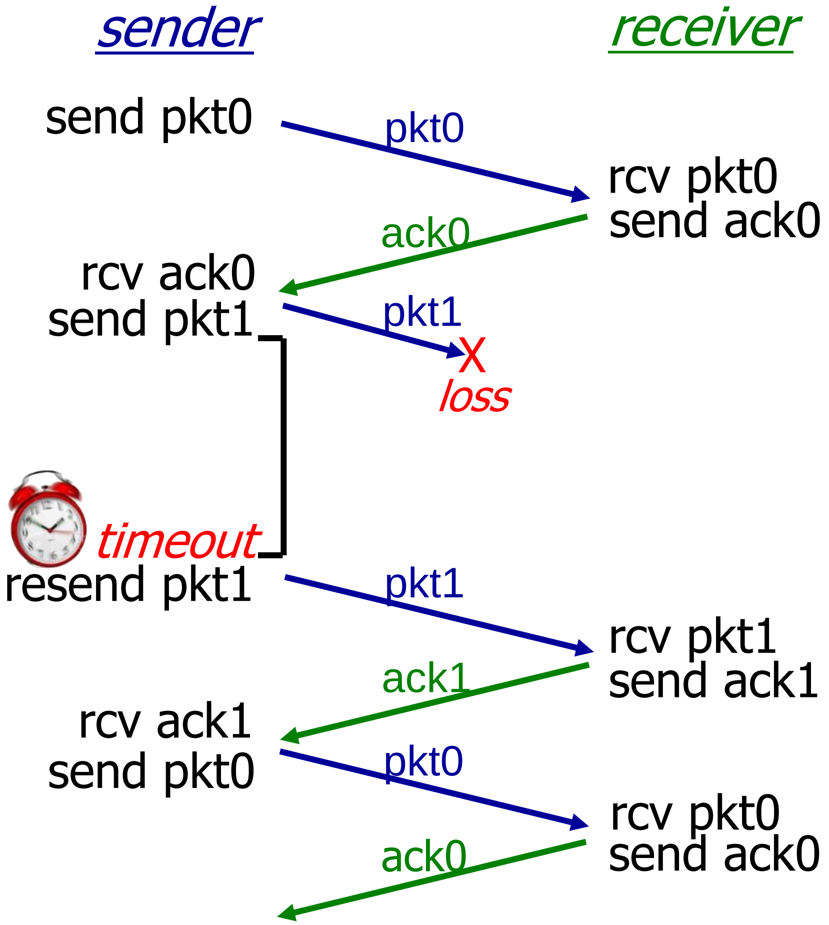
Stop-and-Wait



sender *receiver*

send pkt0
pkt0
rcv pkt0
send ack0
ack0
rcv ack0
send pkt1
pkt1
rcv pkt1
send ack1
ack1
rcv ack1
send pkt0
pkt0
rcv pkt0
send ack0
ack0

(a) no loss

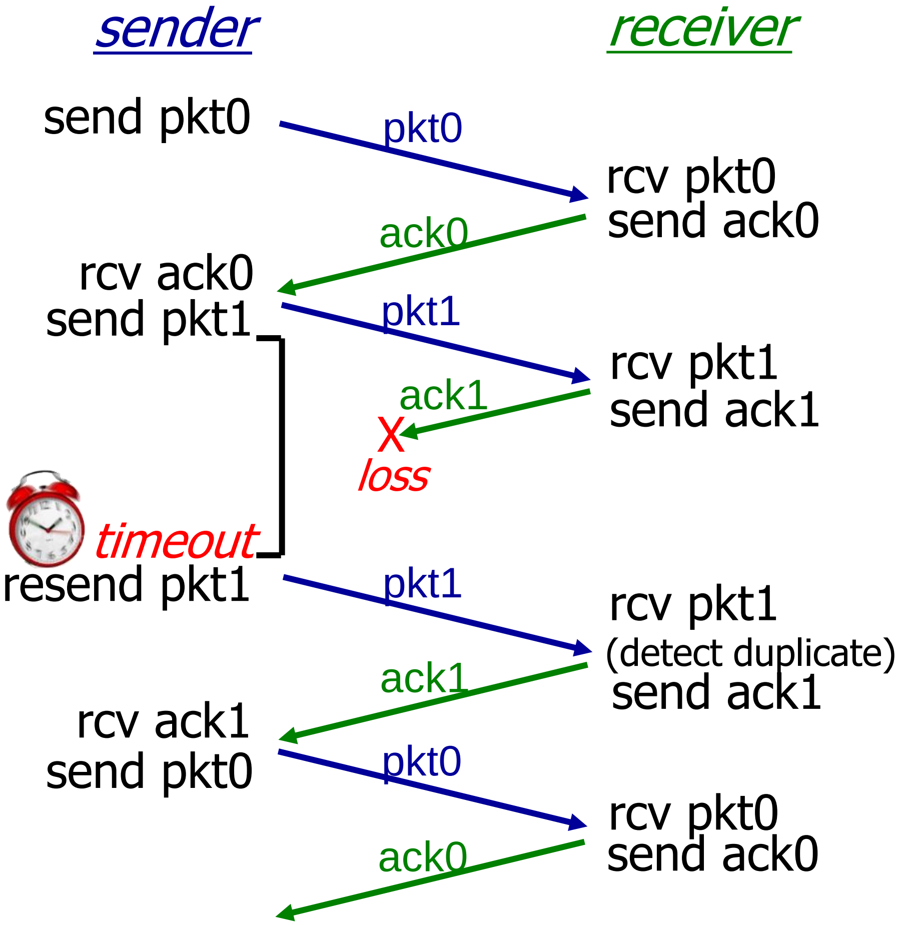


sender *receiver*

send pkt0
pkt0
rcv pkt0
send ack0
ack0
rcv ack0
send pkt1
pkt1
X
loss
timeout
resend pkt1
pkt1
rcv pkt1
send ack1
ack1
rcv ack1
send pkt0
pkt0
rcv pkt0
send ack0
ack0

(b) packet loss

Stop-and-Wait



sender

receiver

send pkt0

pkt0

rcv pkt0
send ack0

ack0

rcv ack0
send pkt1

pkt1

rcv pkt1
send ack1

ack1

X
loss

timeout
resend pkt1

pkt1

rcv pkt1
(detect duplicate)
send ack1

ack1

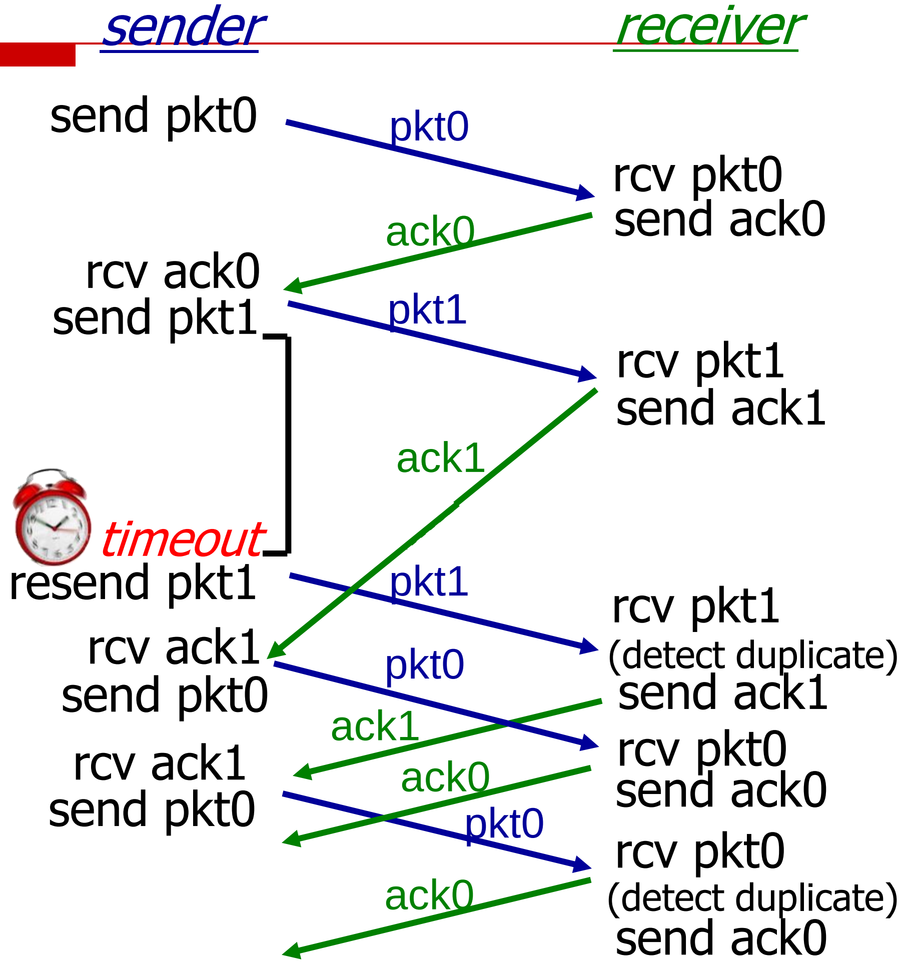
rcv ack1
send pkt0

pkt0

rcv pkt0
send ack0

ack0

(c) ACK loss



sender

receiver

send pkt0

pkt0

rcv pkt0
send ack0

ack0

rcv ack0
send pkt1

pkt1

rcv pkt1
send ack1

ack1

timeout
resend pkt1

pkt1

rcv ack1
send pkt0

pkt0

rcv pkt1
(detect duplicate)
send ack1

ack1

rcv pkt0
send ack0

ack0

rcv ack1
send pkt0

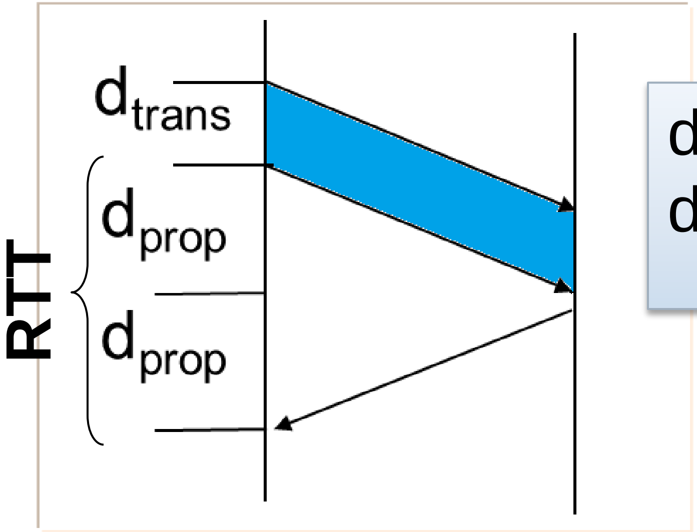
pkt0

rcv pkt0
(detect duplicate)
send ack0

ack0

(d) premature timeout/ delayed ACK

Performance of Stop-and-wait ARQ



d_{trans}= (Frame size)/Bandwidth

d_{ptop}= (Speed of signal)/ (Channel length)

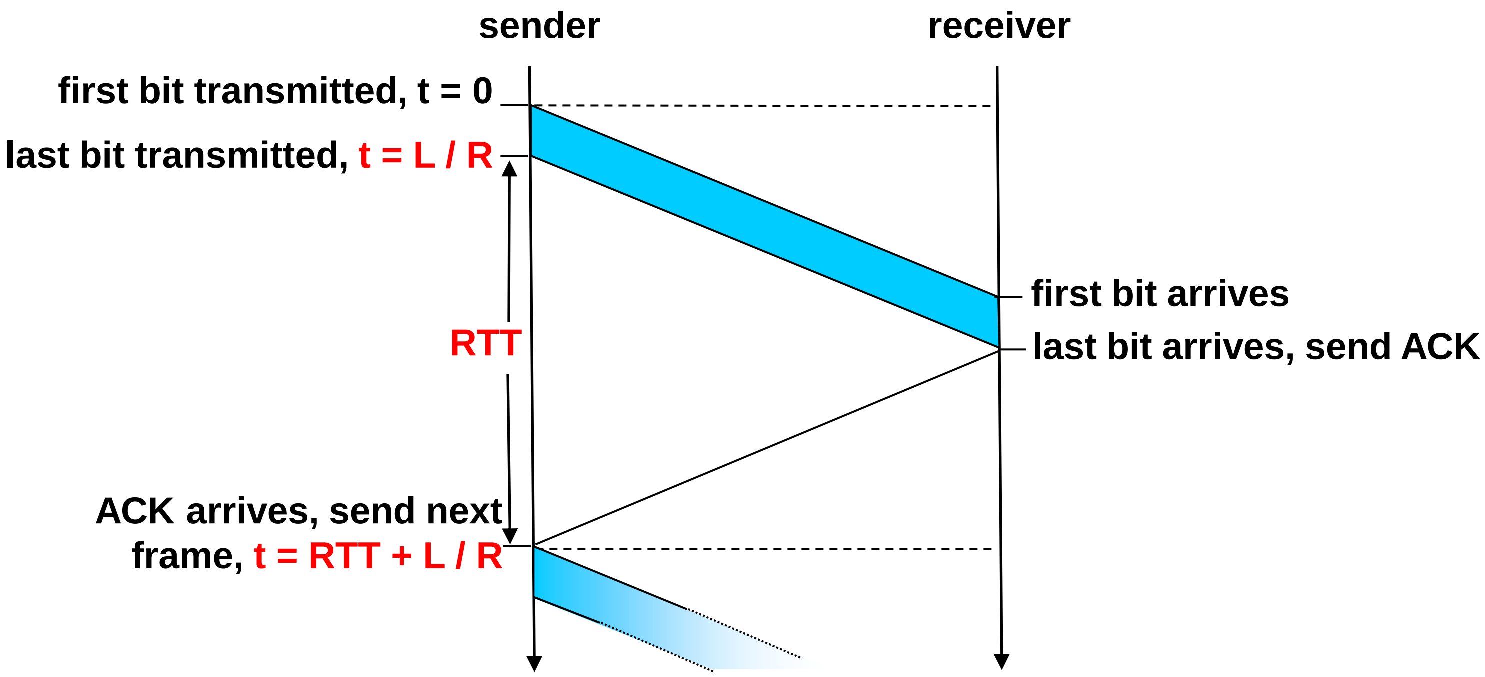
Utilization = d_{trans} /(d_{trans} + 2 d_{prop}), error free case

or

Utilization = (1-P_E)d_{trans} /(d_{trans} + 2 $d_{prop,}$) error case

Stop-and-wait operation

Example: 1 Gbps link, 15 ms prop. delay, 8000 bit frame:



sender
receiver
first bit transmitted, t = 0
last bit transmitted, t = L / R
RTT
first bit arrives
last bit arrives, send ACK
ACK arrives, send next frame, t = RTT + L / R

$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\text{bps}} = 8\,\text{microseconds}$$

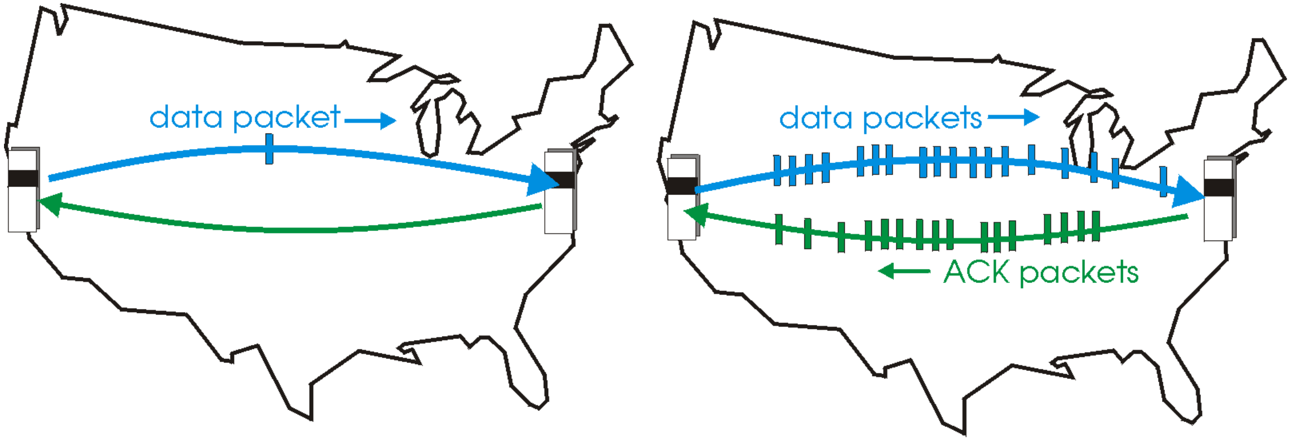
$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

very low

Sliding window (Pipelined) protocols

Pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged frames

- range of sequence numbers must be increased
- buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- Two generic forms of pipelined protocols:

  Go-Back-N , *Selective repeat*

Sliding Window Protocols

Go Back *n*

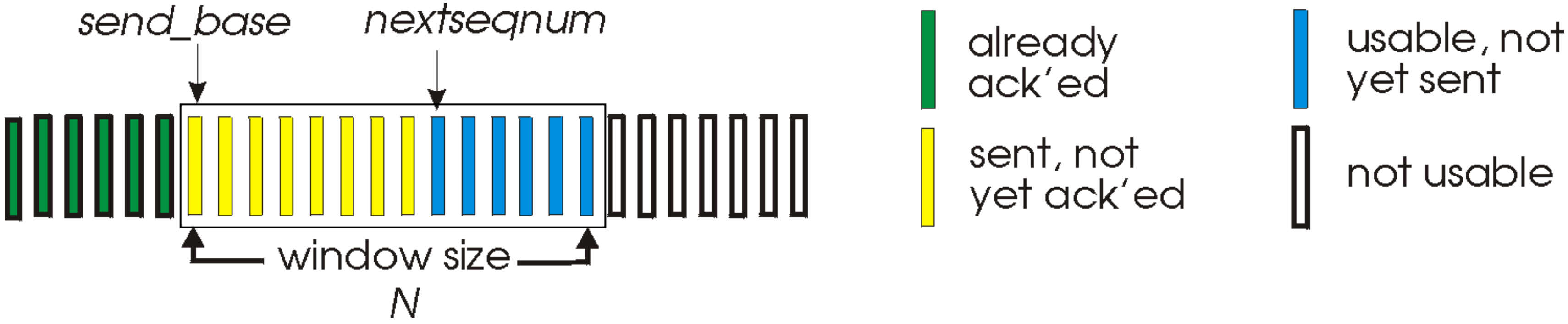
از بسته خراب شده به بعد مجددا ارسال شوند

Selective Repeat

فقط بسته خراب شده مجددا ارسال شود

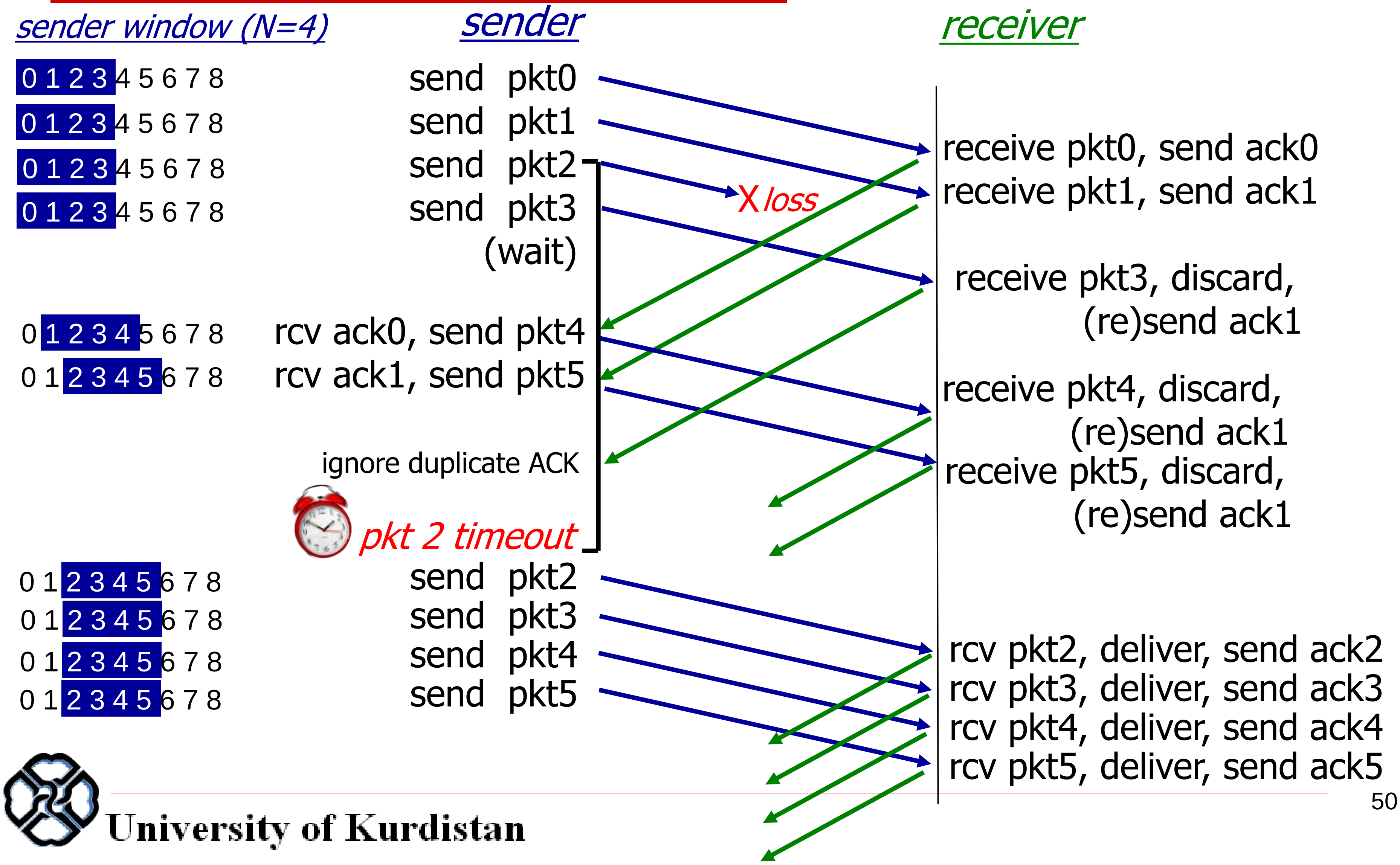
Go-Back-N: sender

- k-bit seq # in pkt header
- “window” of up to N, consecutive unack’ed pkts allowed



- ACK(n): ACKs all pkts up to, including seq # n - *“cumulative ACK ”*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher seq # pkts in window

GBN in action



sender window (N=4)

0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8

sender

send pkt0
send pkt1
send pkt2
send pkt3
(wait)

rcv ack0, send pkt4
rcv ack1, send pkt5

ignore duplicate ACK

pkt 2 timeout

send pkt2
send pkt3
send pkt4
send pkt5

X *loss*

receiver

receive pkt0, send ack0
receive pkt1, send ack1

receive pkt3, discard,
(re)send ack1

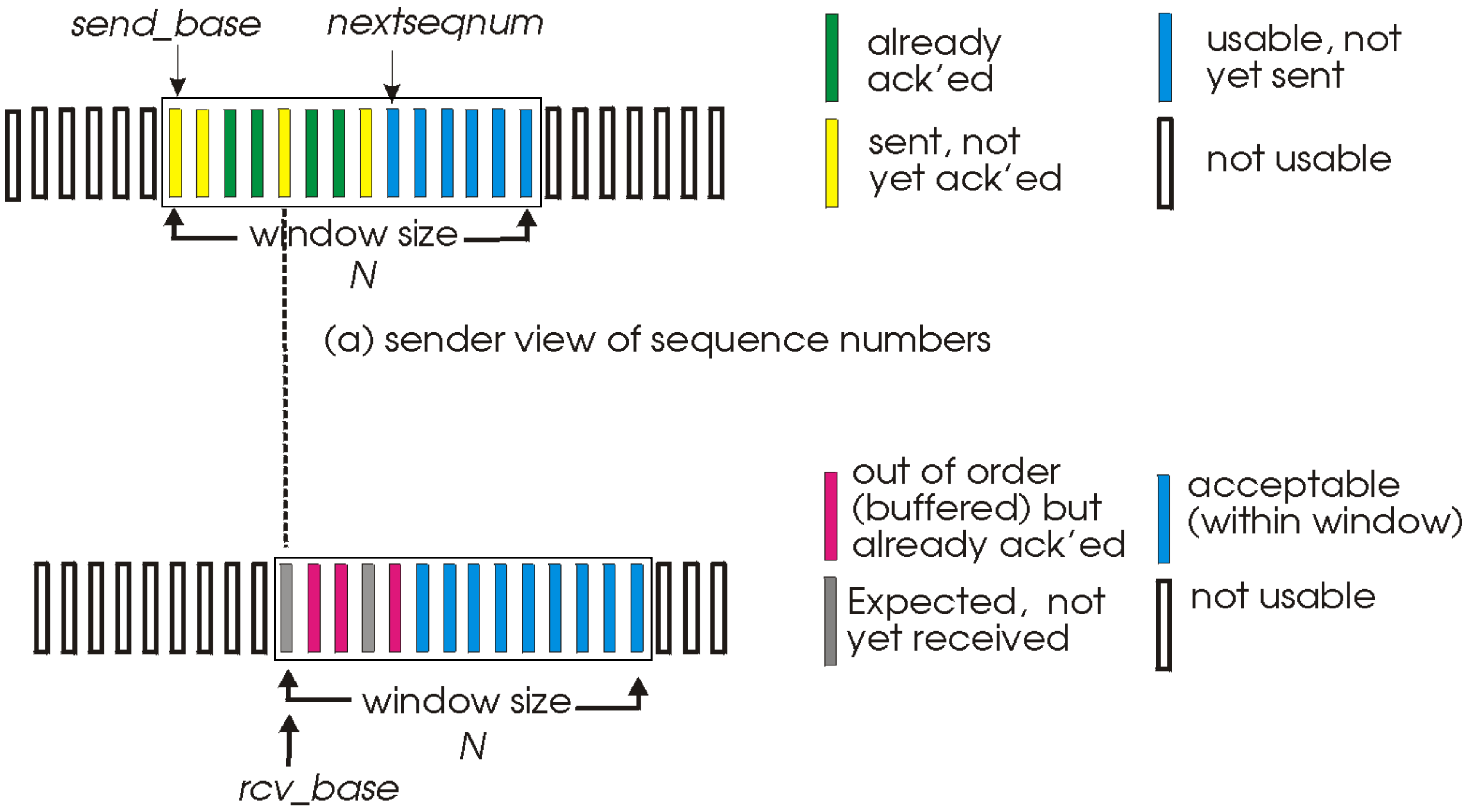
receive pkt4, discard,
(re)send ack1
receive pkt5, discard,
(re)send ack1

rcv pkt2, deliver, send ack2
rcv pkt3, deliver, send ack3
rcv pkt4, deliver, send ack4
rcv pkt5, deliver, send ack5

Selective repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - *N* consecutive seq #’s
  - limits seq #s of sent, unACKed pkts

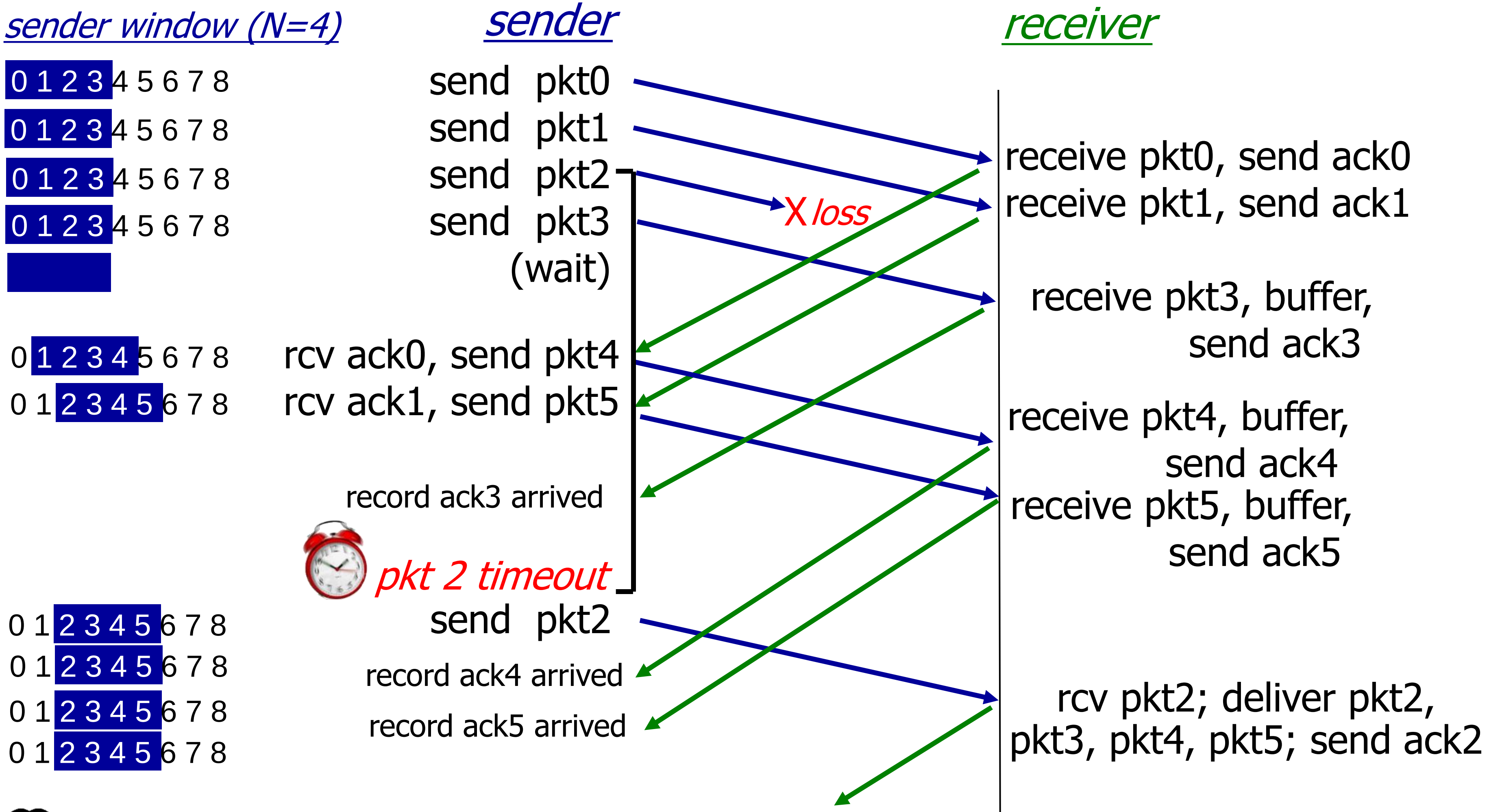
Selective repeat: sender, receiver windows



(a) sender view of sequence numbers

(b) receiver view of sequence numbers

Selective repeat in action



sender window (N=4)

sender

receiver

0 1 2 3 4 5 6 7 8 send pkt0

0 1 2 3 4 5 6 7 8 send pkt1

0 1 2 3 4 5 6 7 8 send pkt2

0 1 2 3 4 5 6 7 8 send pkt3

(wait)

X *loss*

receive pkt0, send ack0

receive pkt1, send ack1

receive pkt3, buffer, send ack3

0 1 2 3 4 5 6 7 8 rcv ack0, send pkt4

0 1 2 3 4 5 6 7 8 rcv ack1, send pkt5

receive pkt4, buffer, send ack4

receive pkt5, buffer, send ack5

record ack3 arrived

pkt 2 timeout

0 1 2 3 4 5 6 7 8 send pkt2

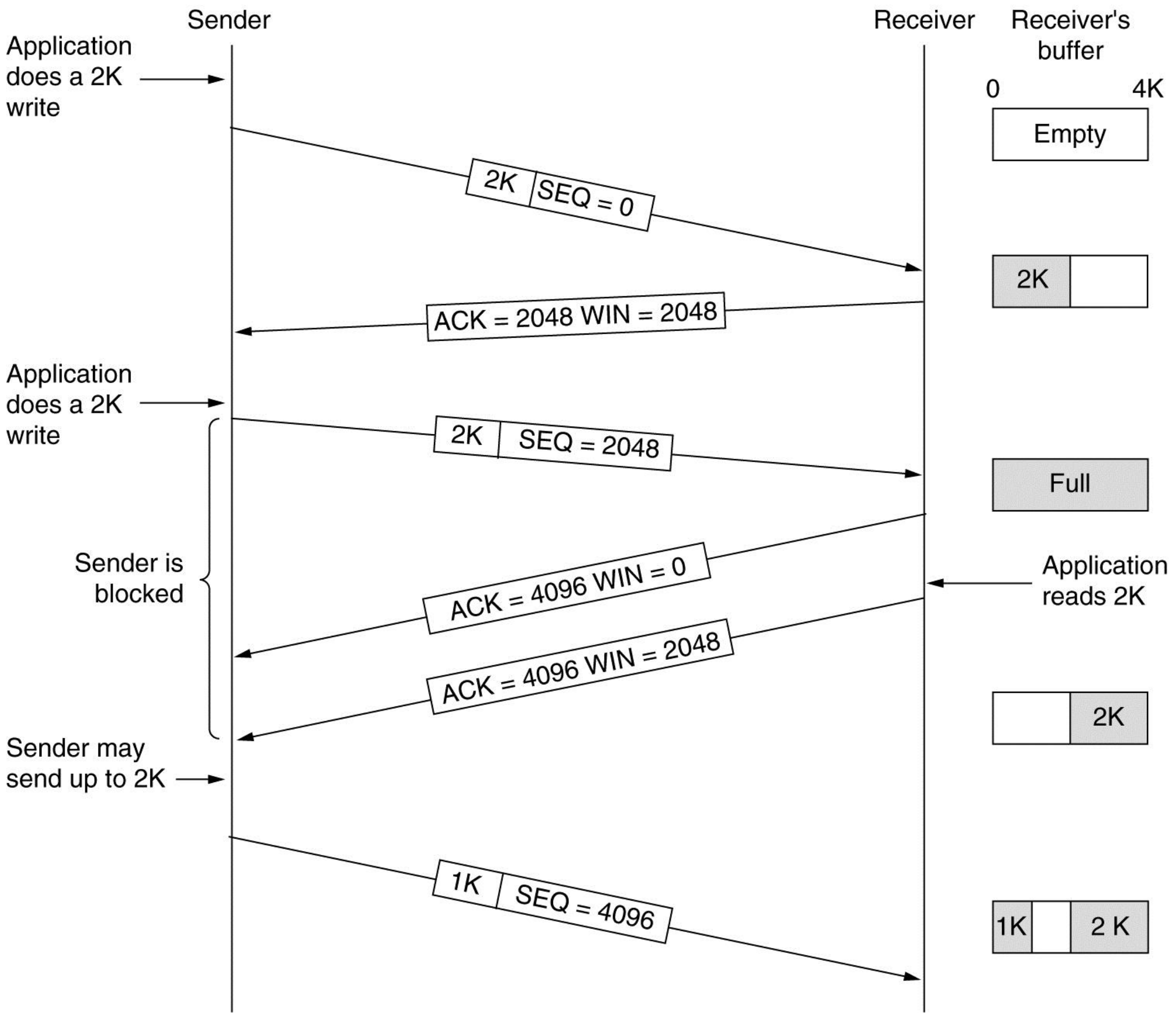
0 1 2 3 4 5 6 7 8 record ack4 arrived

0 1 2 3 4 5 6 7 8 record ack5 arrived

0 1 2 3 4 5 6 7 8

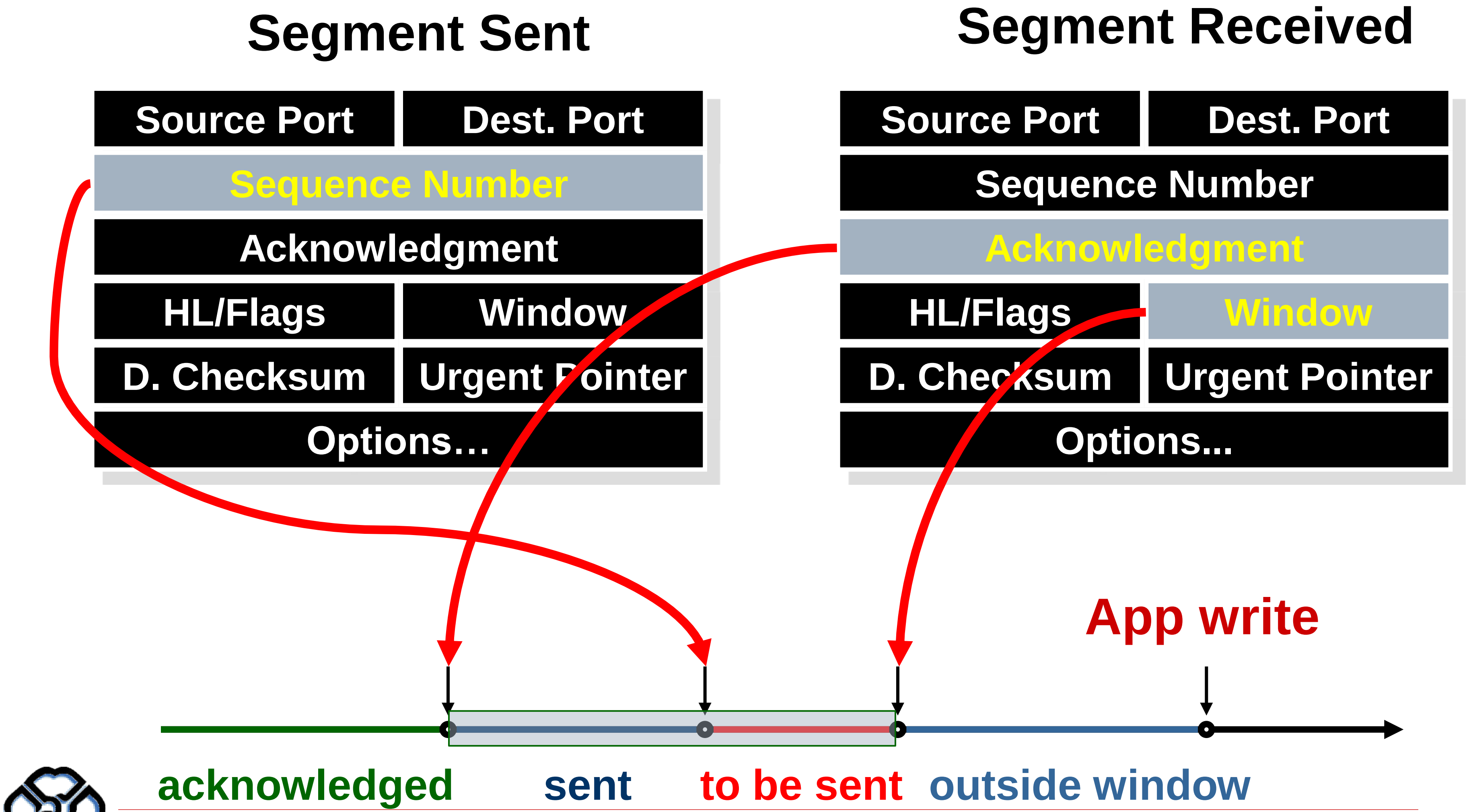
rcv pkt2; deliver pkt2, pkt3, pkt4, pkt5; send ack2

TCP Flow control



مقدار درج شده در فیلد **Window Size** درهر بسته **TCP**، به طرف مقابل تفهیم می‌کند که از بایتی که شماره آن در فیلد **Acknowledgement Number** درج شده، به اندازه چند بایت حق ارسال داده دارد. درج عدد صفر در فیلد **Window Size** کاملاً قانونی و معتبر است و در حقیقت بیان می‌کند که اگر چه تا بایت شماره **Acknowledgement Number-۱**، دریافت شده و لیکن به دلیل کمبود شدید فضای بافر، تا اطلاع ثانوی نمی‌تواند پذیرای داده بیشتری باشد. گیرنده بعداً می‌تواند به فرستنده مجوز ارسال بدهد: این کار با فرستادن یک بسته که درآن فیلد **Acknowledgement Number** همان مقدار قبلی را دارد و فیلد **Window Size** آن غیر صفر است، صورت می‌گیرد.

Window Flow Control: Sender Side



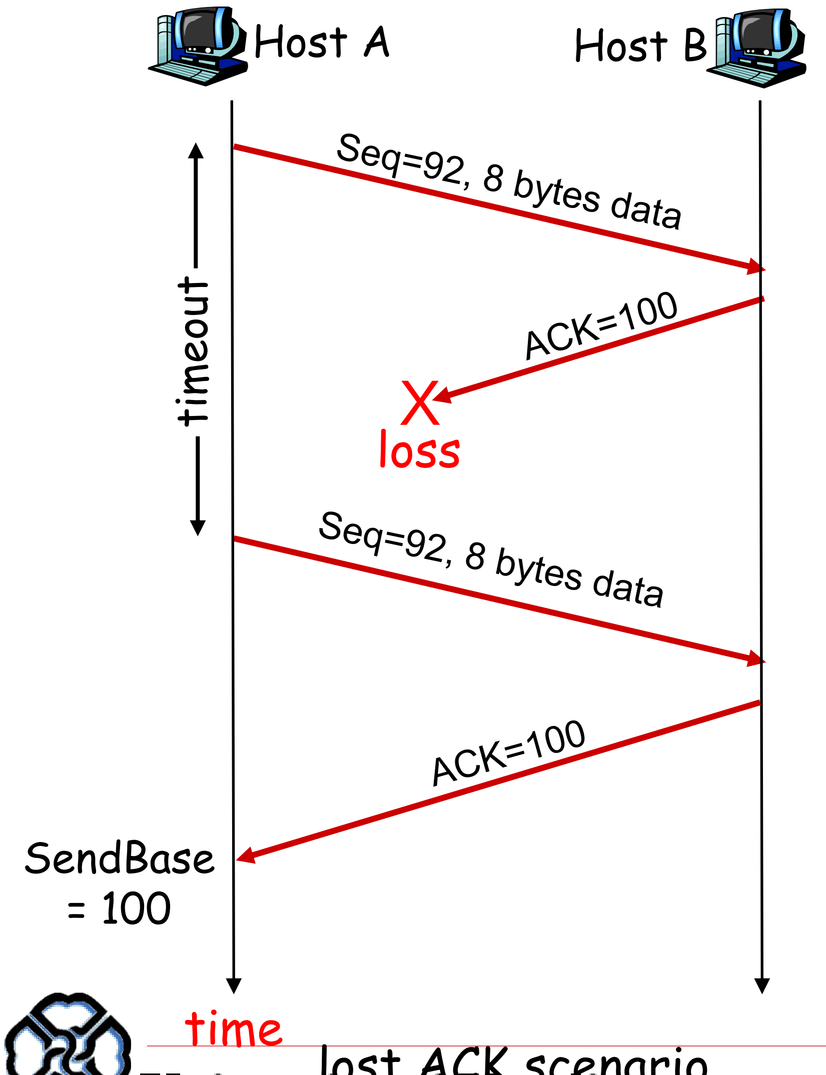
Segment Sent

Source Port	Dest. Port
Sequence Number	
Acknowledgment	
HL/Flags	Window
D. Checksum	Urgent Pointer
Options…	

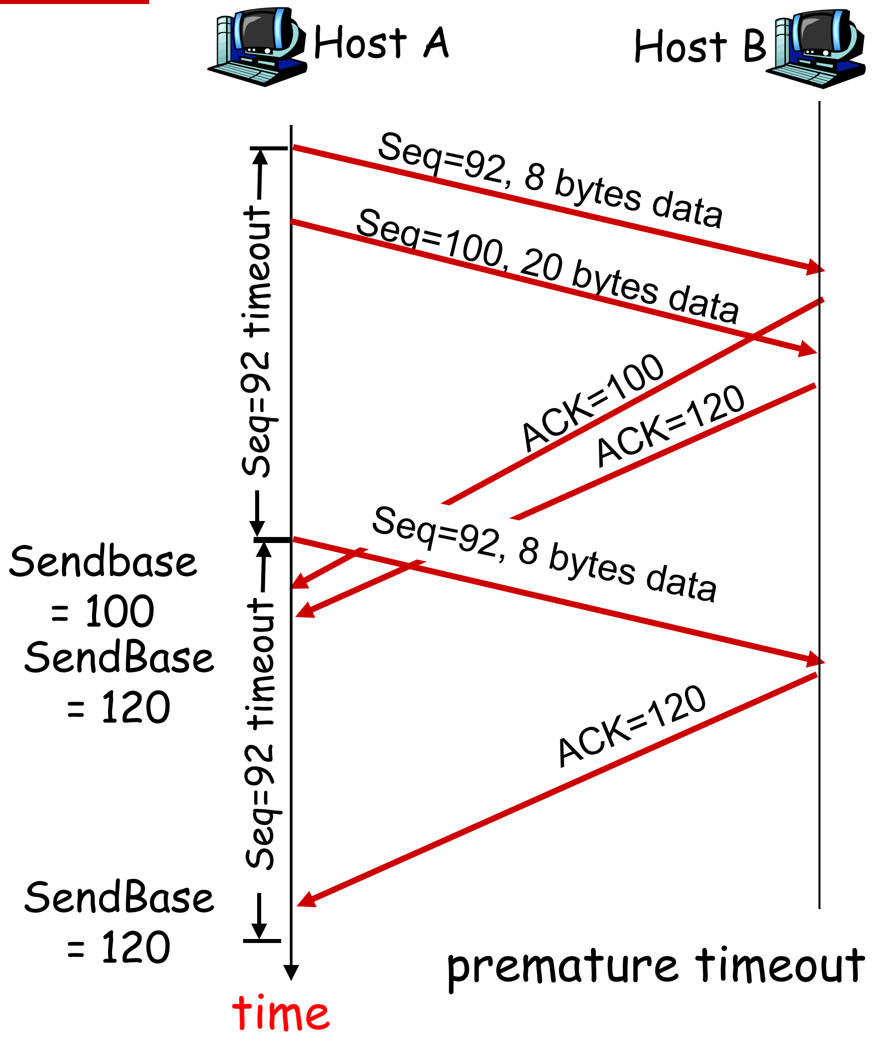
Segment Received

Source Port	Dest. Port
Sequence Number	
Acknowledgment	
HL/Flags	Window
D. Checksum	Urgent Pointer
Options...	

TCP: retransmission scenarios



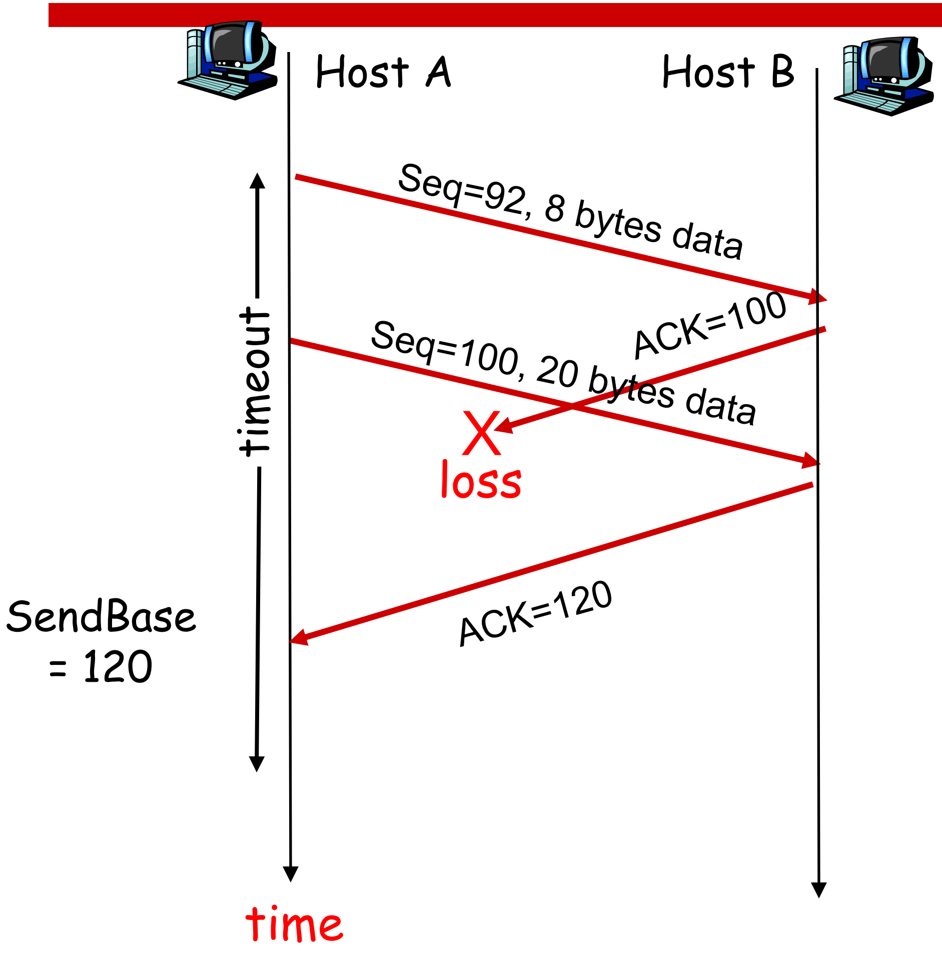
lost ACK scenario



premature timeout

University of Kurdistan

TCP retransmission scenarios (more)



Cumulative ACK scenario

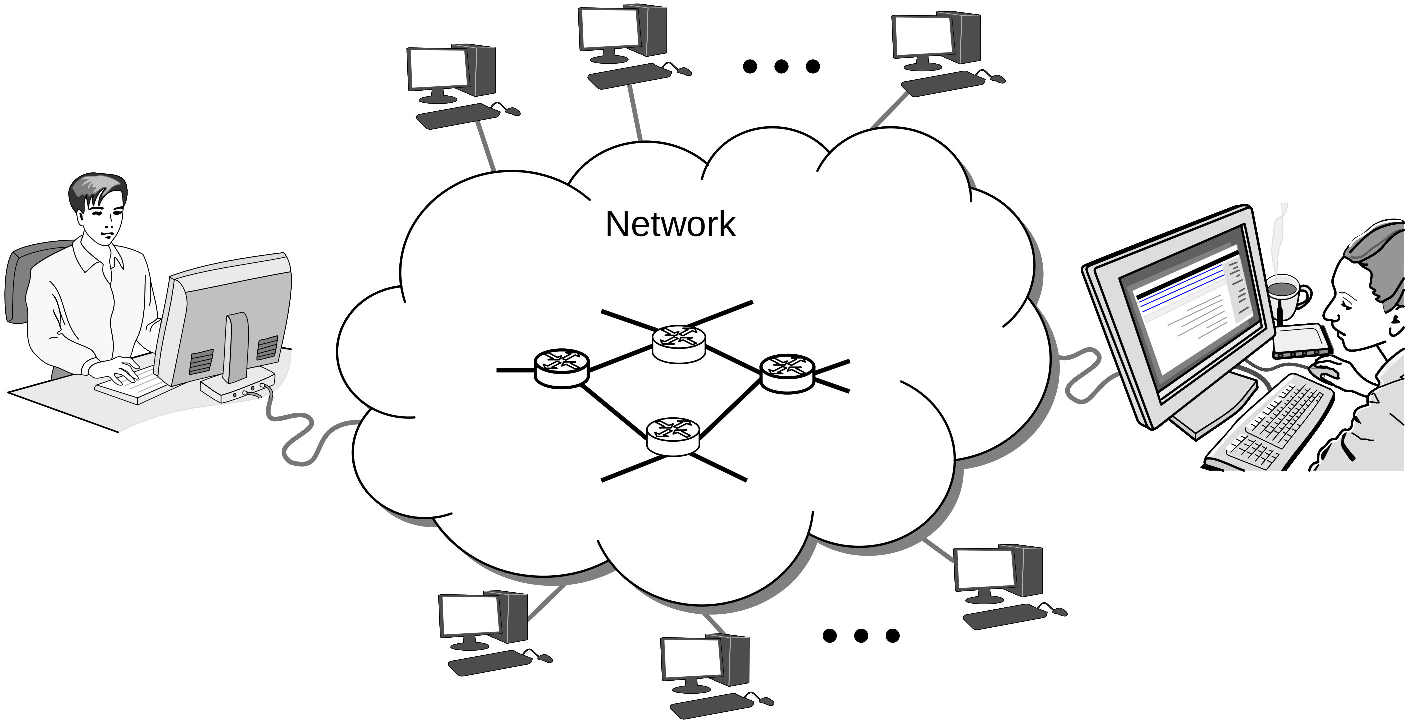
TCP Congestion Control

- TCP limit sending rate as a function of perceived network congestion
  - little traffic – increase sending rate
  - much traffic – reduce sending rate

- Congestion algorithm has three major “components”:
  - additive-increase, multiplicative-decrease (AIMD)
  - slow-start
  - reaction to timeout events

Network Conceptual Model

Many sources and many receivers …

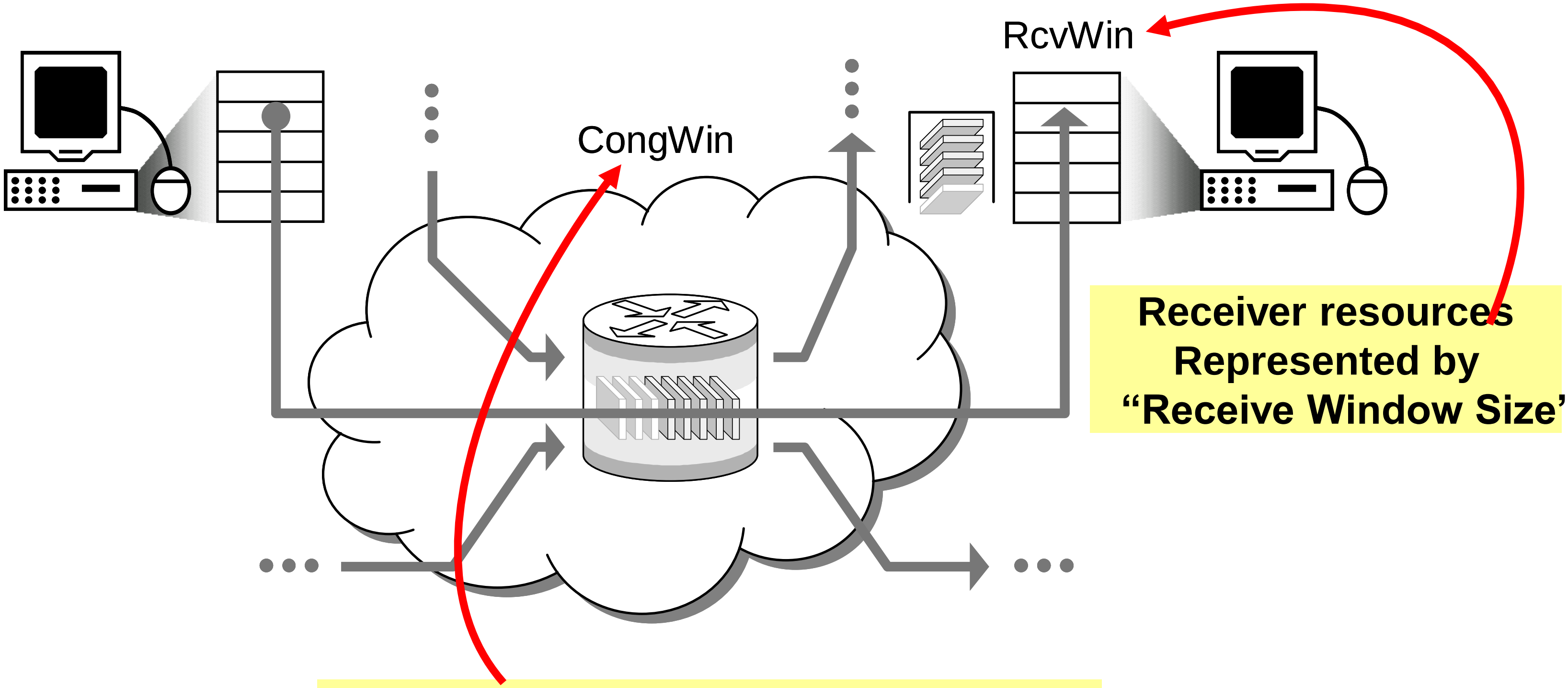


(a)

We don't know when sources will start/end their sessions; also their datarates are variable

Simplified Network Model

The entire network is abstracted as a single router – "black box"

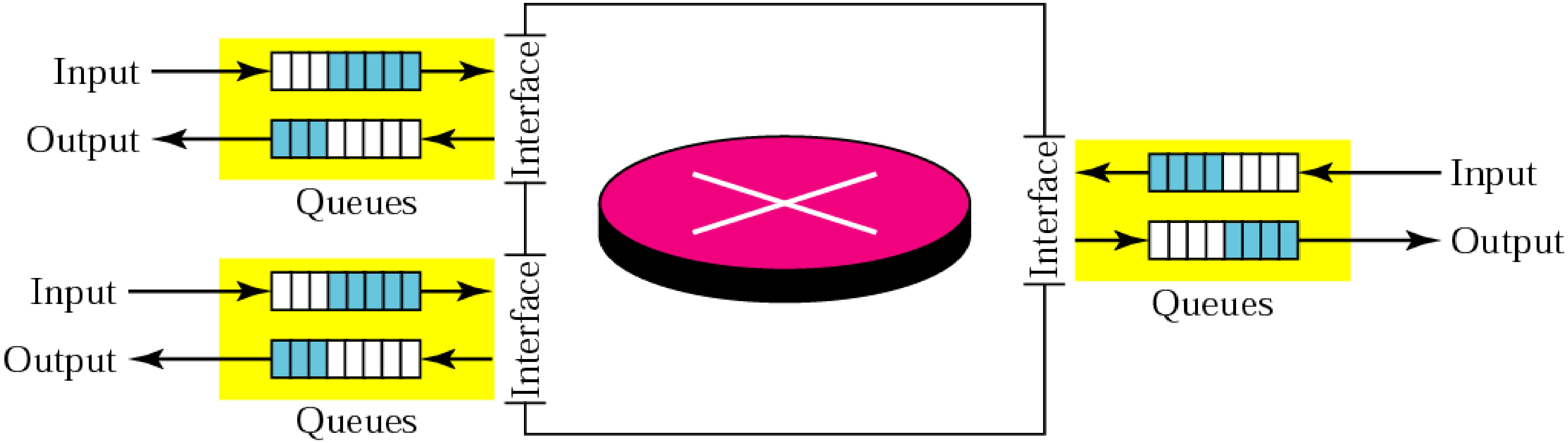


Receiver resources Represented by "Receive Window Size"

(b)

Network resources Represented by "Congestion Window Size"

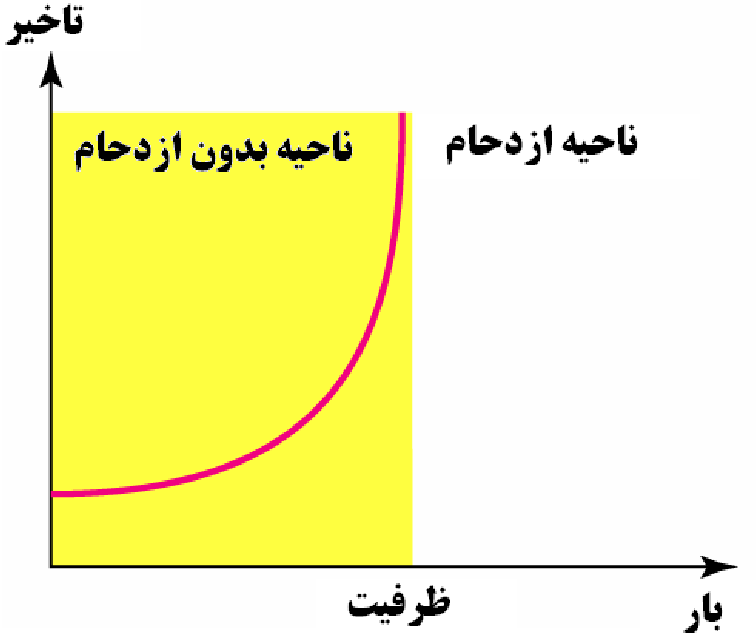
Router queues

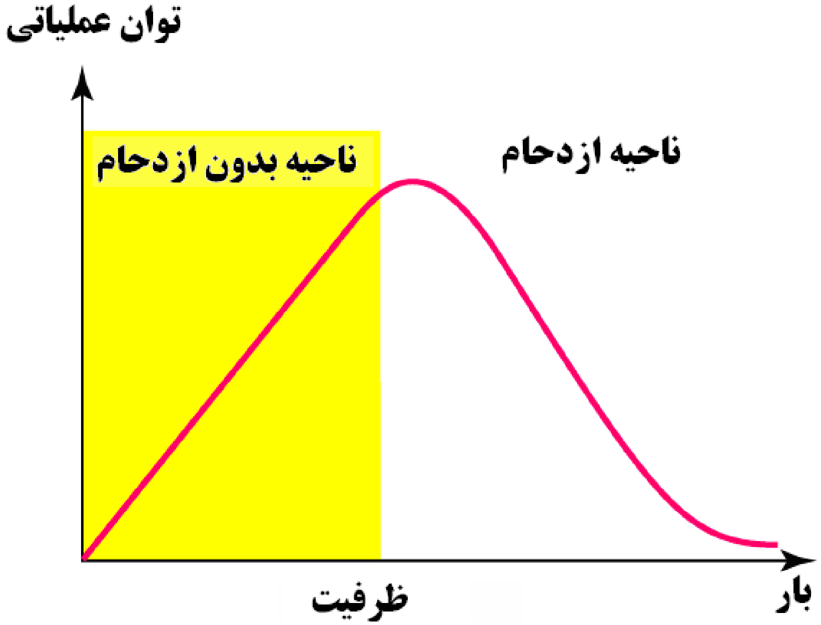


اگرنرخ ورودی بسته ها، از سرعت پردازش بسته ها در داخل مسیریاب بیشتر باشد، صفهای ورودی طولانی خواهند شد.

اگرنرخ حرکت بسته ها در صفهای خروجی کمتر از نرخ پردازش آنها باشد، صفهای خروجی طولانی خواهند شد.

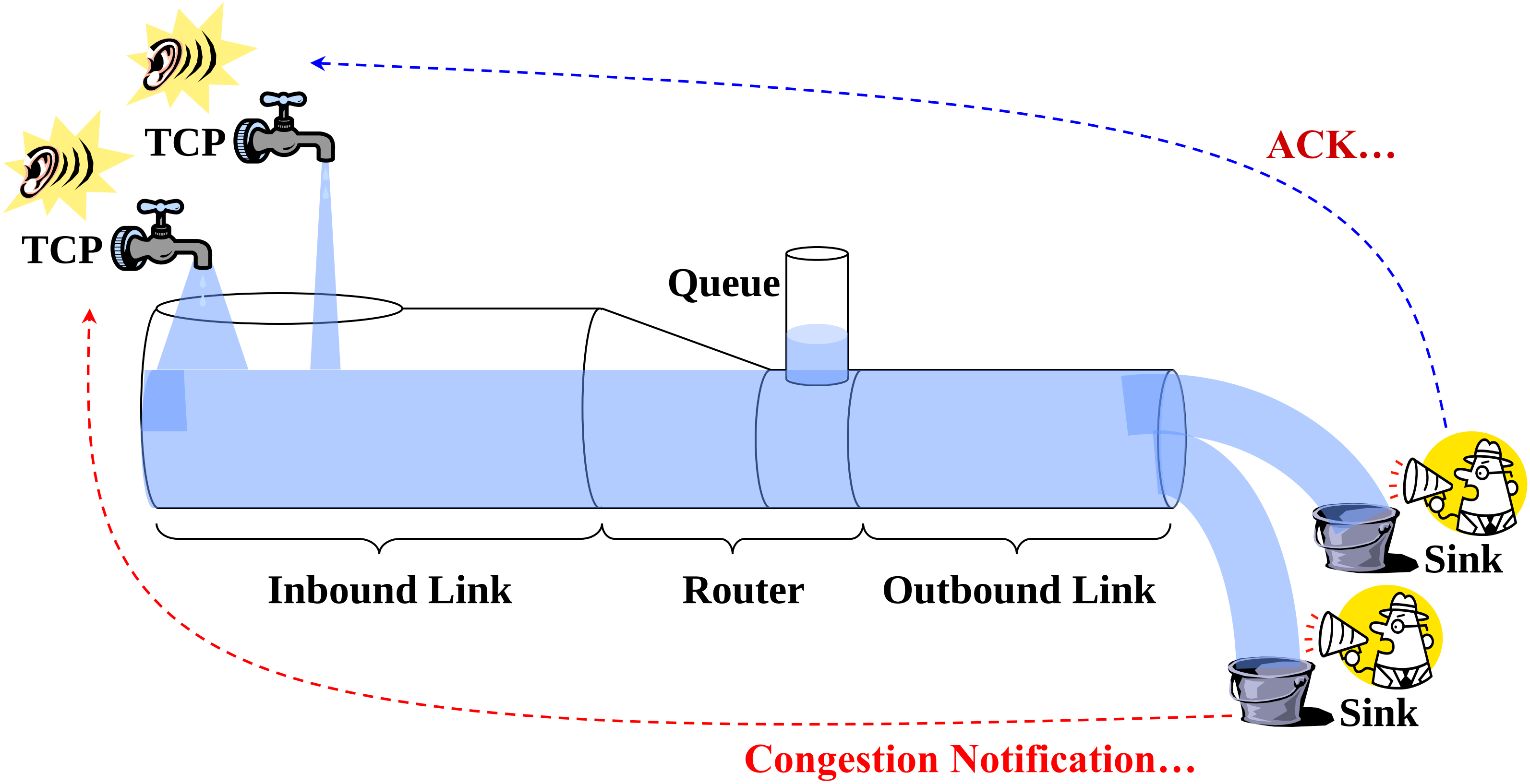
در صورت پر شدن بافر، مسیریاب مجبور به دور ریختن بسته میشود.



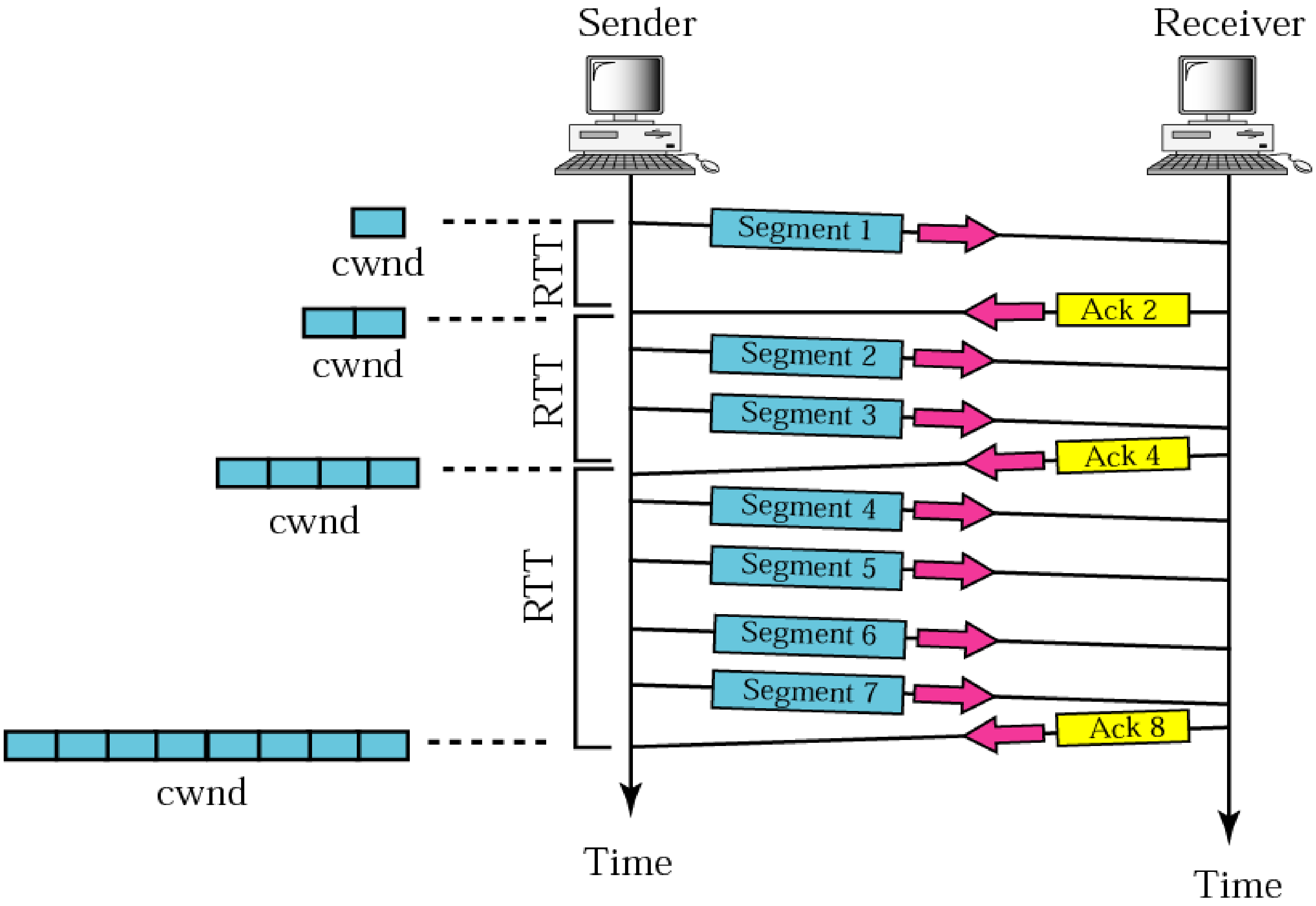


هنگامی که بار بیش از ظرفیت میگردد، تاخیر به سمت بینهایت میرود. در این حالت بسته ها به مقصد نمیرسند و صفها طولانی و طولانیتر خواهند شد. از طرف دیگر، فرستنده بسته ها نیز به دلیل اینکه پیام تصدیقی از جانب گیرنده دریافت نمیکند، اقدام به ارسال مجدد بسته ها نموده و به بدتر شدن شرایط کمک مینماید.

TCP in action

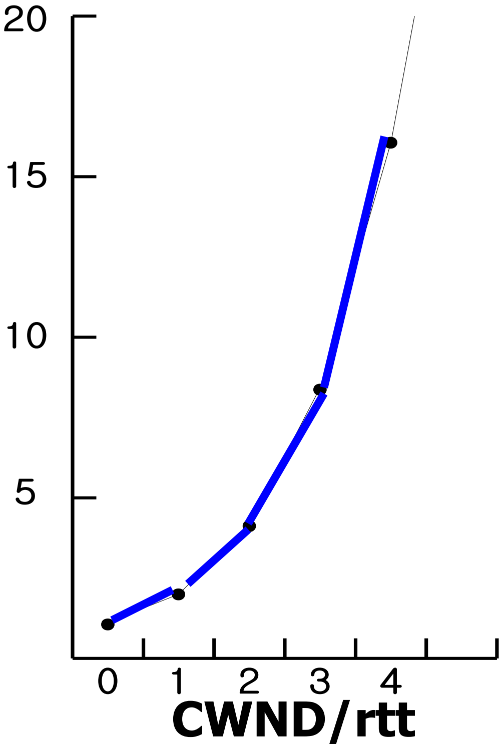


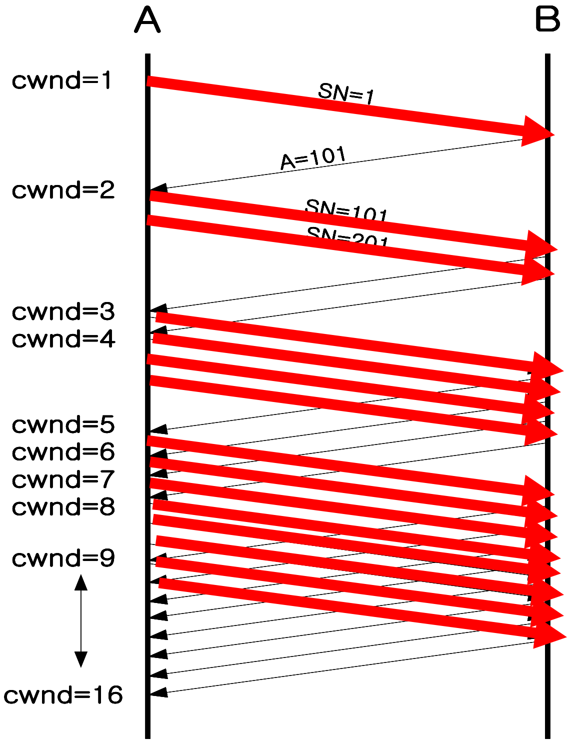
Slow start, exponential increase



Slow Start

- If CWND is less than or equal to SSTHRESTH : Slow start
- Slow start dictates that CWND start at one segment, and be incremented by one segment every time an ACK is received

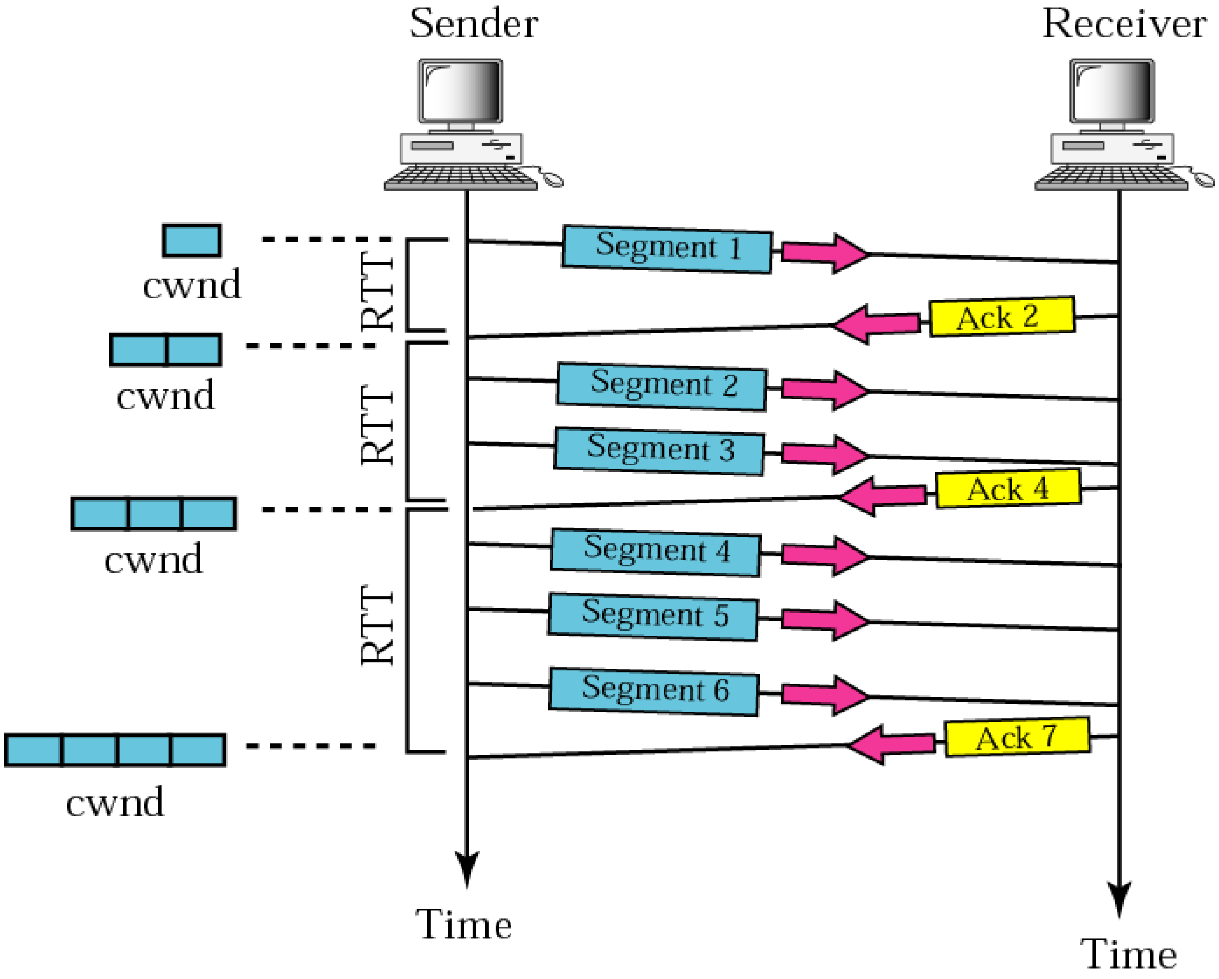




Note:

In the slow start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.

Additive increase



Note:

In the congestion avoidance algorithm the size of the congestion window increases additively until congestion is detected.

Note:

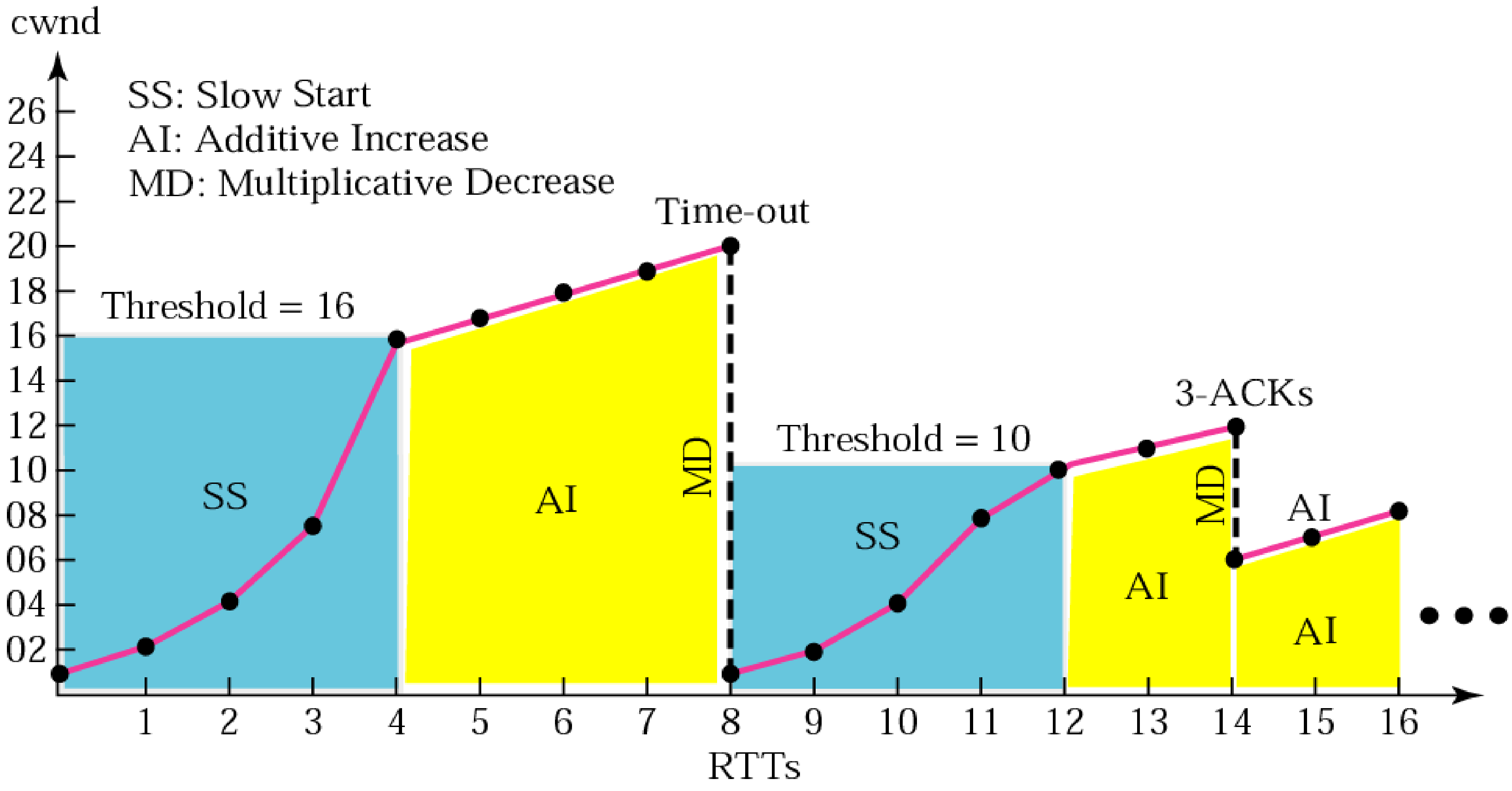
Most implementations react differently to congestion detection:

- ***If detection is by time-out, a new slow start phase starts.***
- ***If detection is by three ACKs, a new congestion avoidance phase starts.***

Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires

Congestion Control example



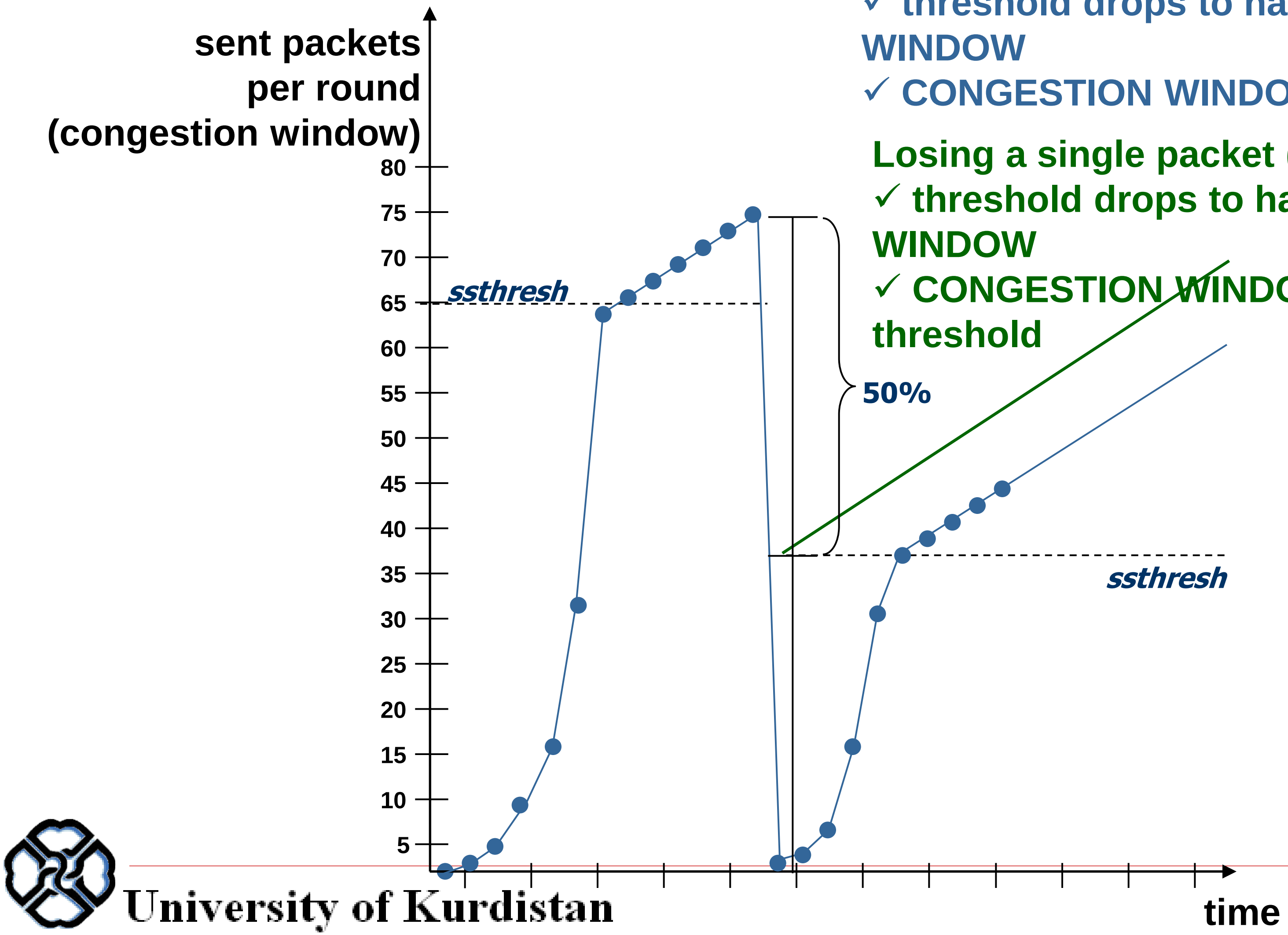
TCP Congestion Control

Losing a single packet (TCP Tahoe):

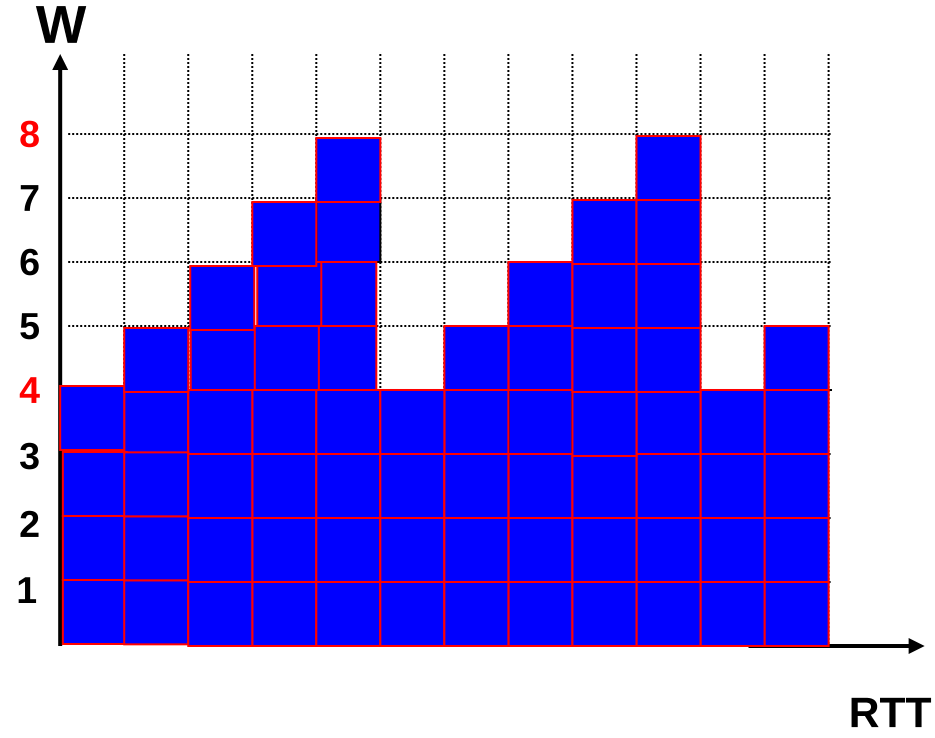
- ✓ threshold drops to halve CONGESTION WINDOW
- ✓ CONGESTION WINDOW back to 1

Losing a single packet (TCP Reno):

- ✓ threshold drops to halve CONGESTION WINDOW
- ✓ CONGESTION WINDOW back to new threshold



TCP Behavior



Calculate “average packet loss rate” and “average throughput”

TCP Round Trip Time and Timeout

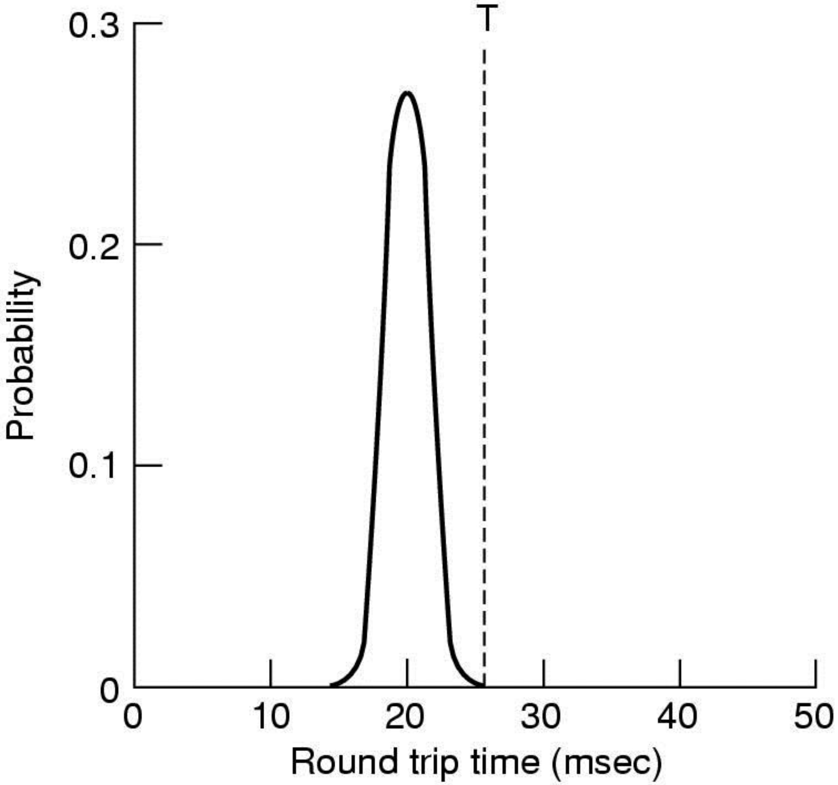
Q: how to set TCP timeout value?

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

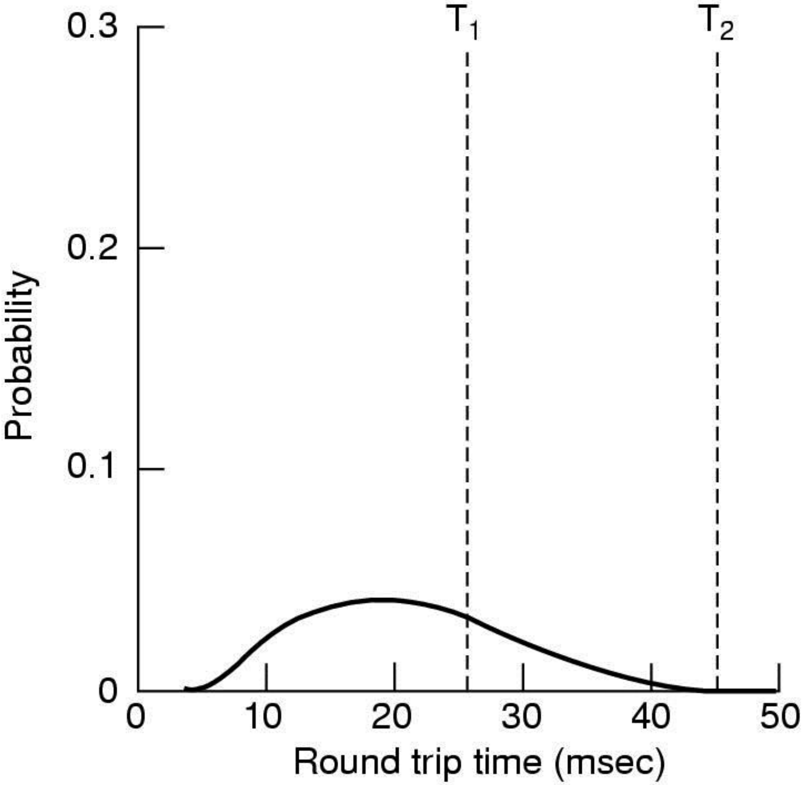
Q: how to estimate RTT?

- **`SampleRTT`**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **`SampleRTT`** will vary, want estimated RTT “smoother”
  - average several recent measurements, not just current **`SampleRTT`**

TCP timer management



Data link layer



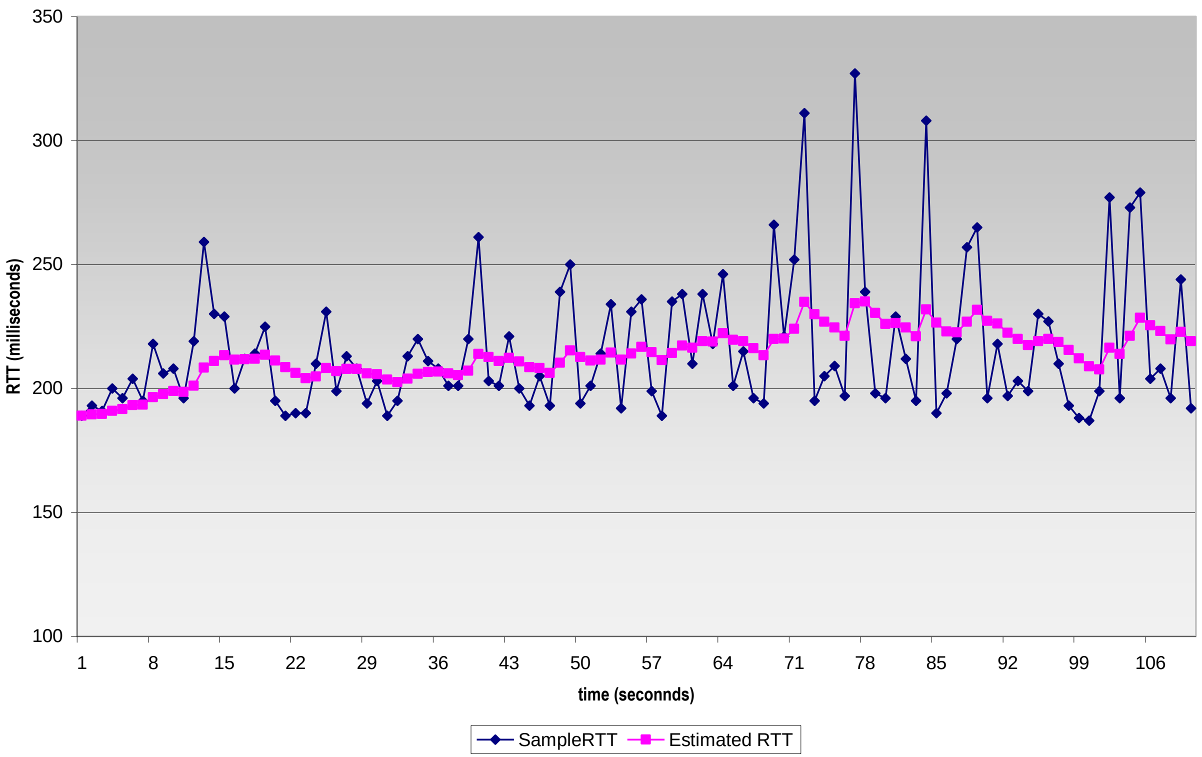
Transport layer

TCP Round Trip Time and Timeout

```
EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT
```

- typical value: $\alpha = 0.125$

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:

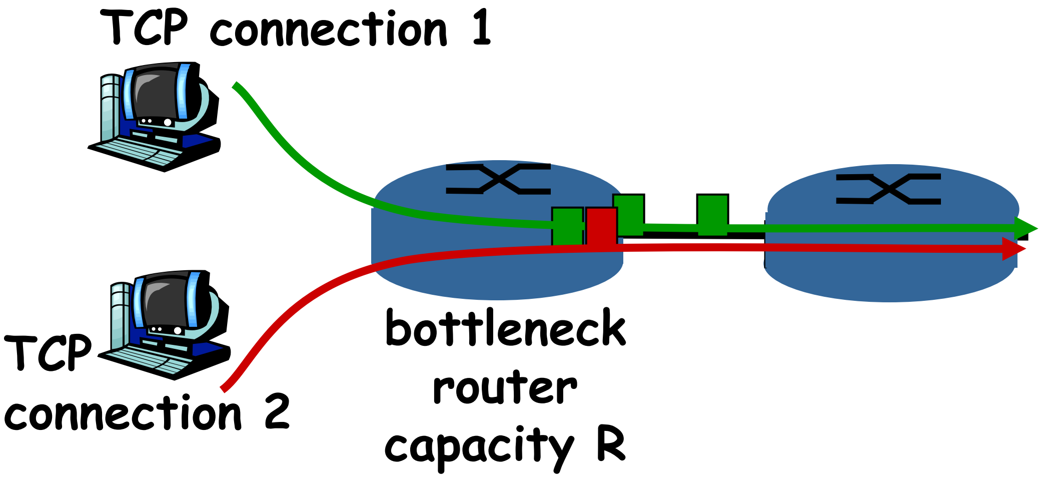
```
DevRTT = (1-β)*DevRTT +β*|SampleRTT-EstimatedRTT|

(typically, β = 0.25)
```

Then set timeout interval:

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

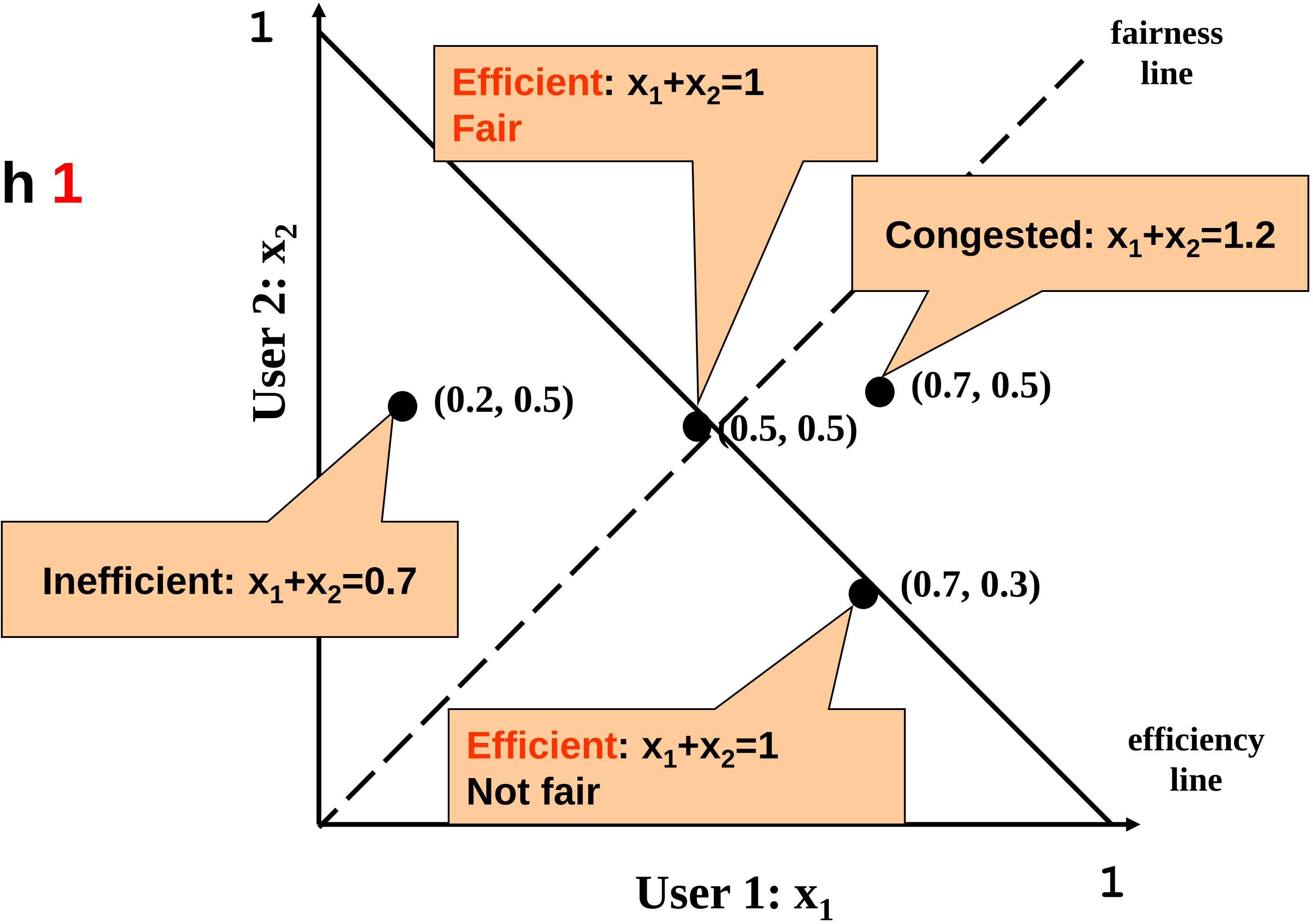
TCP Fairness

Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Example

- Total bandwidth 1



Efficient: $x_1+x_2=1$
Fair

Congested: $x_1+x_2=1.2$

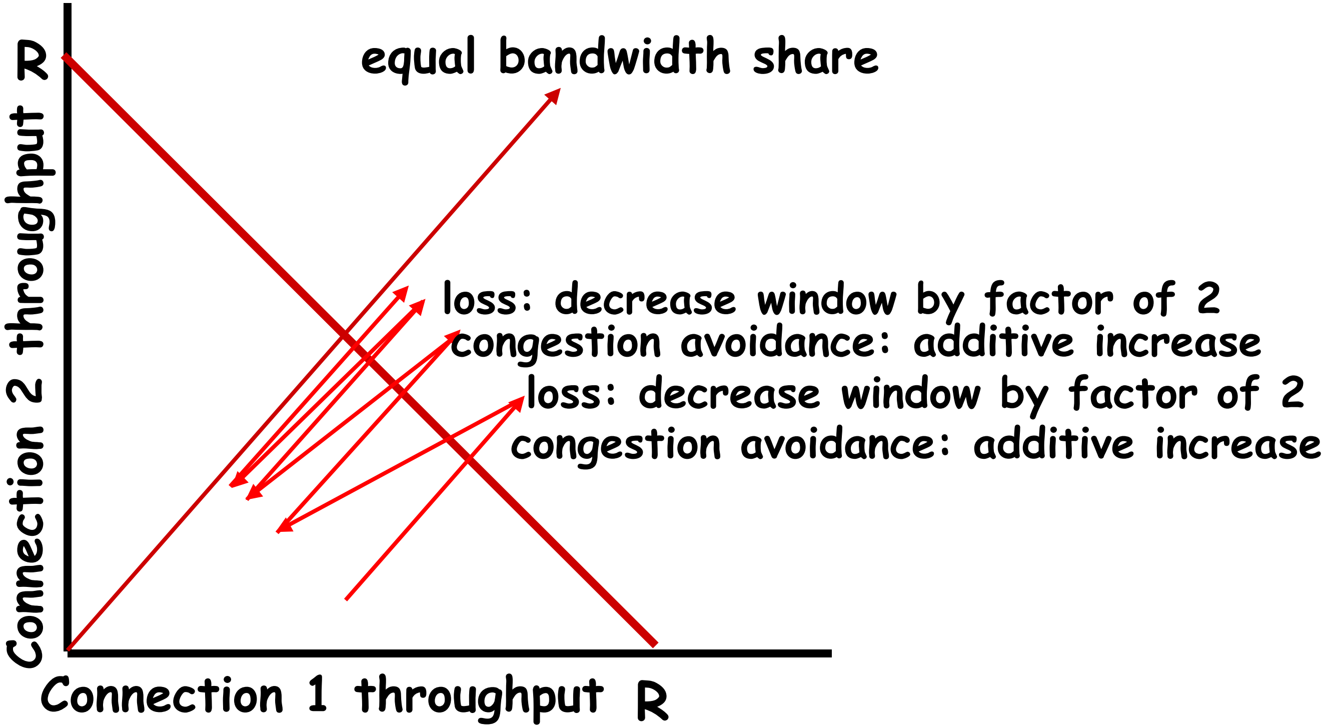
Inefficient: $x_1+x_2=0.7$

Efficient: $x_1+x_2=1$
Not fair

Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally


R
equal bandwidth share
loss: decrease window by factor of 2
congestion avoidance: additive increase
loss: decrease window by factor of 2
congestion avoidance: additive increase
Connection 2 throughput
Connection 1 throughput
R

Delay modeling

Q: How long does it take to receive an object from a Web server after sending a request?

- **Latency** is the time the client when initiates a TCP connection until receiving the complete object.

Key components of Latency are:

1) TCP connection establishment, 2) data transmission delay, 3) slow start

Notation, assumptions:

- one link between client and server of rate R
- amount of sent data depends only on CongWin (large RcvWin)
- all protocols headers and non-file segments are ignored
- file send has integer number of MSSs
- large initial Threshold
- no retransmissions (no loss, no corruption)
- MSS is **S** bits
- object size is **O** bits
- **R** bps is the transmission rate
- Latency lower bound with no congestion window constraint = 2RTT (TCP Conn) + O/R

Congestion Window size:

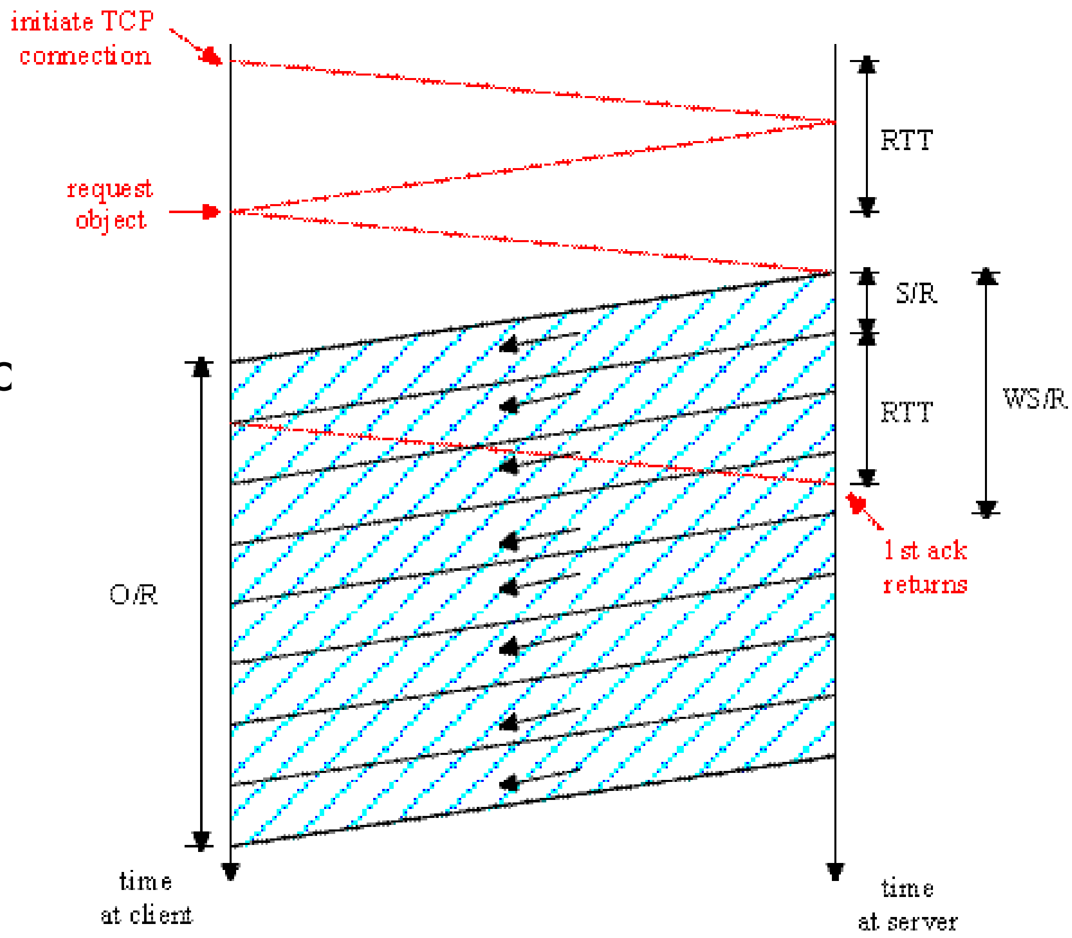
- First assume: fixed congestion window, W segments
- Then dynamic window, modeling slow start

Fixed congestion window (1)

First case:

WS/R > RTT + S/R: server receives ACK for 1st segment in 1st window before 1st window's worth c data sent where W=4.

Segments arrive periodically from server every S/R seconds and ACKs arrive periodically at server every S/R seconds



delay = 2RTT + O/R

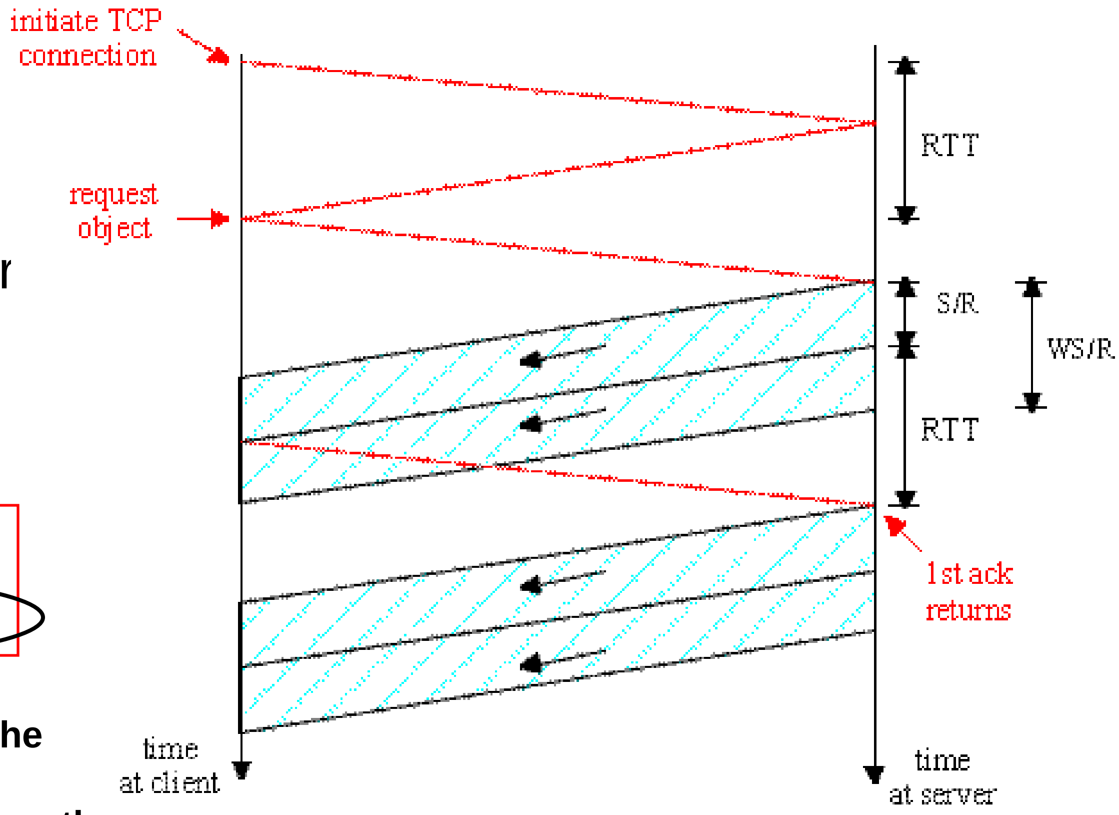
Fixed congestion window (2)

Second case:

- WS/R < RTT + S/R: server waits for ACK after sending all window's segments where W=2.

delay = 2RTT + O/R
+ (K-1)[S/R + RTT - WS/R]

*** K = # windows of data that cover the object or K=O/WS**
*** Additional stalled state time between the transmission of each of the windows. For K-1 periods (server not stalled when transmitting last window) with each period lasting RTT-(W-1)S/R**



TCP Delay Modeling: Slow Start (1)

Now suppose window grows according to slow start

Will show that the delay for one object is:

$$\boxed{Latency = 2RTT + \frac{O}{R} + P\left[RTT + \frac{S}{R}\right] - (2^P - 1)\frac{S}{R}}$$

- *P* is the number of times TCP idles at server:

$$P = \min\{Q, K-1\}$$

- Q is the number of times the server idles if the object were of infinite size.

- K is the number of windows that cover the object.

TCP Delay Modeling: Slow Start (2)

Delay components:

- 2 RTT for connection estab and request
- O/R to transmit object
- time server idles due to slow start

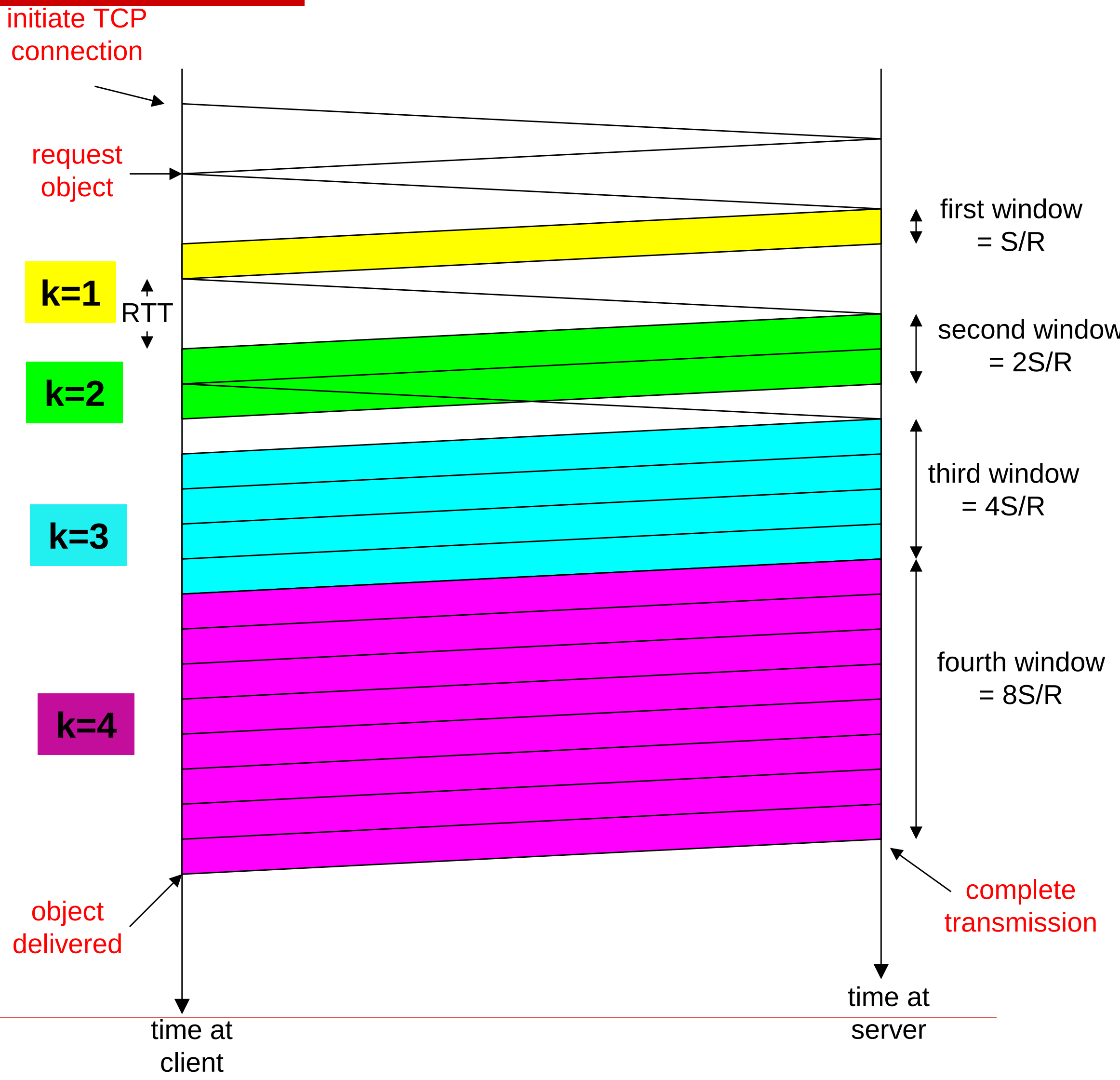
Server idles:

P = min{K-1,Q} times

Example:

- O/S = 15 segments in object
- K = 4 windows
- Q = 2
- P = min{K-1,Q} = 2

Server idles P=2 times



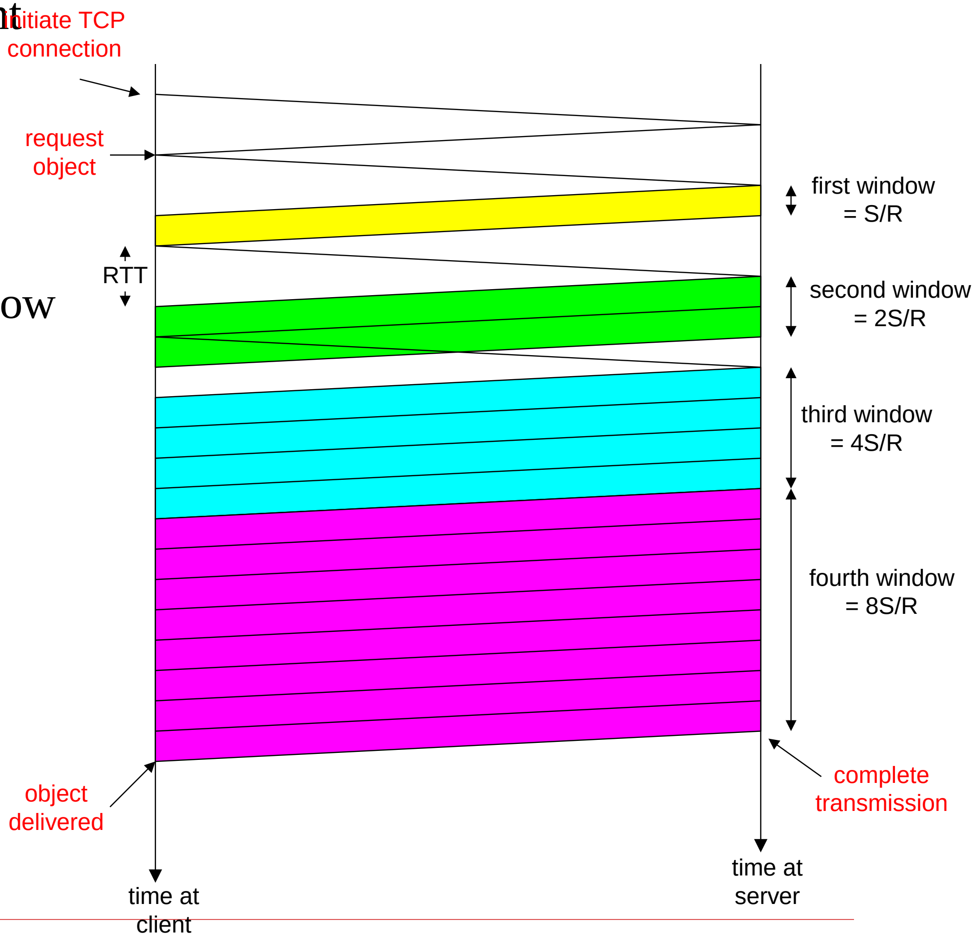
TCP Delay Modeling (3)

$$\frac{S}{R} + RTT = \text{time from when server starts to send segment until server receives acknowledgement}$$

$$2^{k-1}\frac{S}{R} = \text{time to transmit the kth window}$$

$$\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]^{+} = \text{idle time after the } k\text{th window}$$

$$\begin{aligned}\text{delay} &= \frac{O}{R} + 2RTT + \sum_{p=1}^{P} idleTime_p \\ &= \frac{O}{R} + 2RTT + \sum_{k=1}^{P}\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right] \\ &= \frac{O}{R} + 2RTT + P\left[RTT + \frac{S}{R}\right] - (2^{P}-1)\frac{S}{R}\end{aligned}$$



Questions