

# CSC 257/457 – Computer Networks

Fall 2017

MW 4:50 pm – 6:05 pm

CSB 601

# Announcement

---

Project 3 is out

CSC 457 seminar/survey paper

**Proposed topic write-up:**

**due on October 28th**

# **CHAPTER 3 (TRANSPORT LAYER)**

# Chapter 3

## Transport Layer

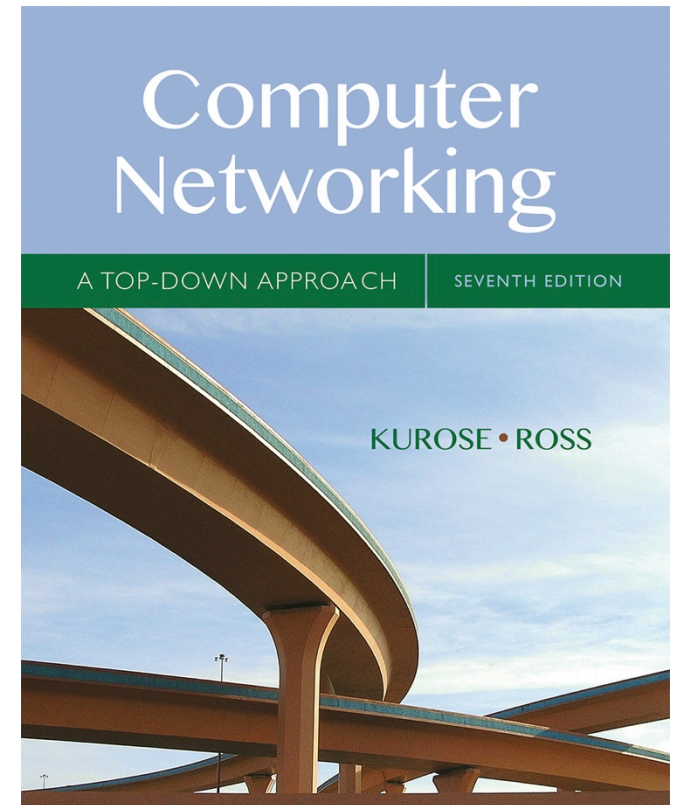
### A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

© All material copyright 1996-2016  
J.F Kurose and K.W. Ross, All Rights Reserved



## *Computer Networking: A Top Down Approach*

7<sup>th</sup> edition

Jim Kurose, Keith Ross

Pearson/Addison Wesley

April 2016

## TCP round trip time, 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

# TCP round trip time, timeout

---

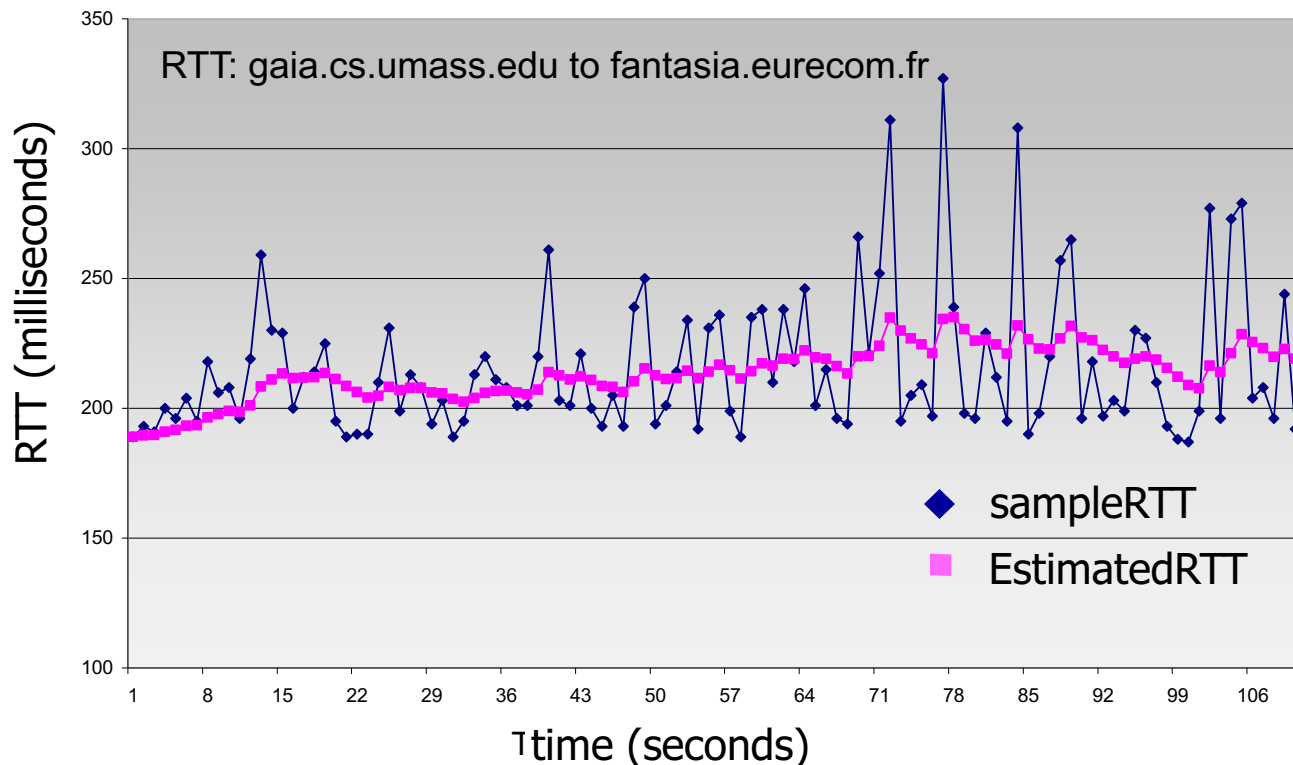
Q: how to estimate RTT?

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

# TCP round trip time, timeout

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

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value:  $\alpha = 0.125$




# TCP round trip time, timeout

---

- timeout interval: **EstimatedRTT** + “safety margin”
  - large variation in **EstimatedRTT** -> larger safety margin

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

 ↑ ↑  
estimated RTT                      “safety margin”

\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactivetcp.html](http://gaia.cs.umass.edu/kurose_ross/interactivetcp.html)



# TCP round trip time, timeout

- timeout interval: **EstimatedRTT** plus “safety margin”
  - large variation in **EstimatedRTT** -> larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:

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

(typically,  $\beta = 0.25$ )

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$



↑  
estimated RTT

↑  
“safety margin”

# Chapter 3 outline

---

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

# TCP reliable data transfer

---

- TCP creates rdt service on top of IP' s unreliable service
  - pipelined segments
  - cumulative acks
  - single retransmission timer
- retransmissions triggered by:
  - timeout events
  - duplicate acks

let' s initially consider simplified TCP sender:

- ignore duplicate acks
- ignore flow control, congestion control

# TCP sender events:

---

*data rcvd from app:*

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unacked segment
  - expiration interval: **TimeoutInterval**

# TCP sender events:

---

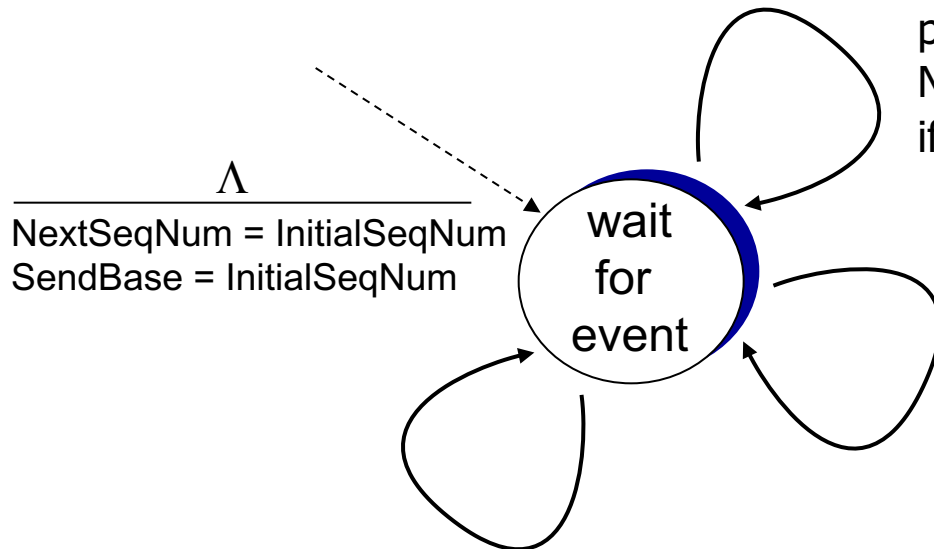
## *timeout:*

- retransmit segment that caused timeout
- restart timer

## *ack rcvd:*

- if ack acknowledges previously unacked segments
  - update what is known to be ACKed
  - start timer if there are still unacked segments

# TCP sender (simplified)



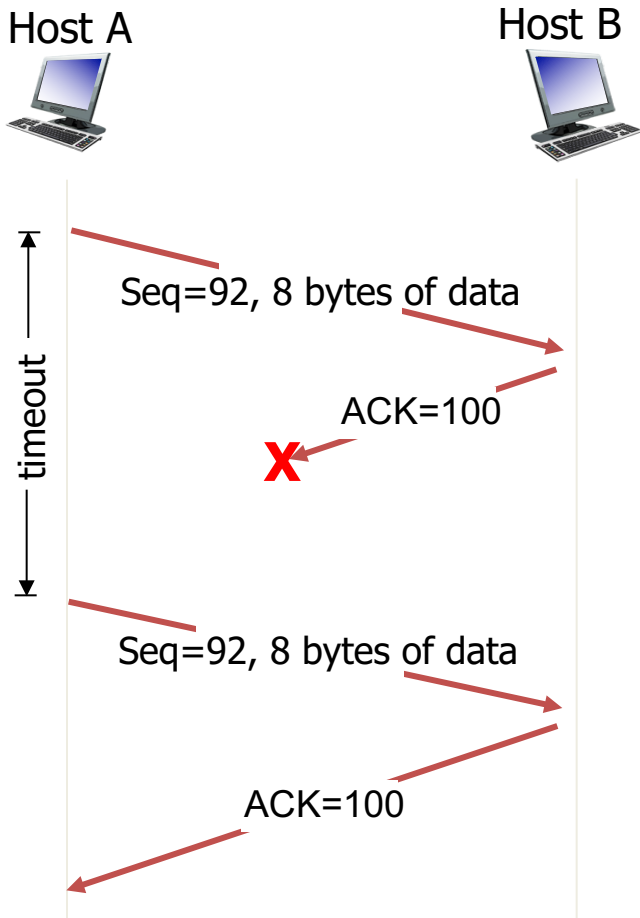
data received from application above  
create segment, seq. #: NextSeqNum  
pass segment to IP (i.e., “send”)  
NextSeqNum = NextSeqNum + length(data)  
if (timer currently not running)  
start timer

timeout  
retransmit not-yet-acked segment  
with smallest seq. #  
start timer

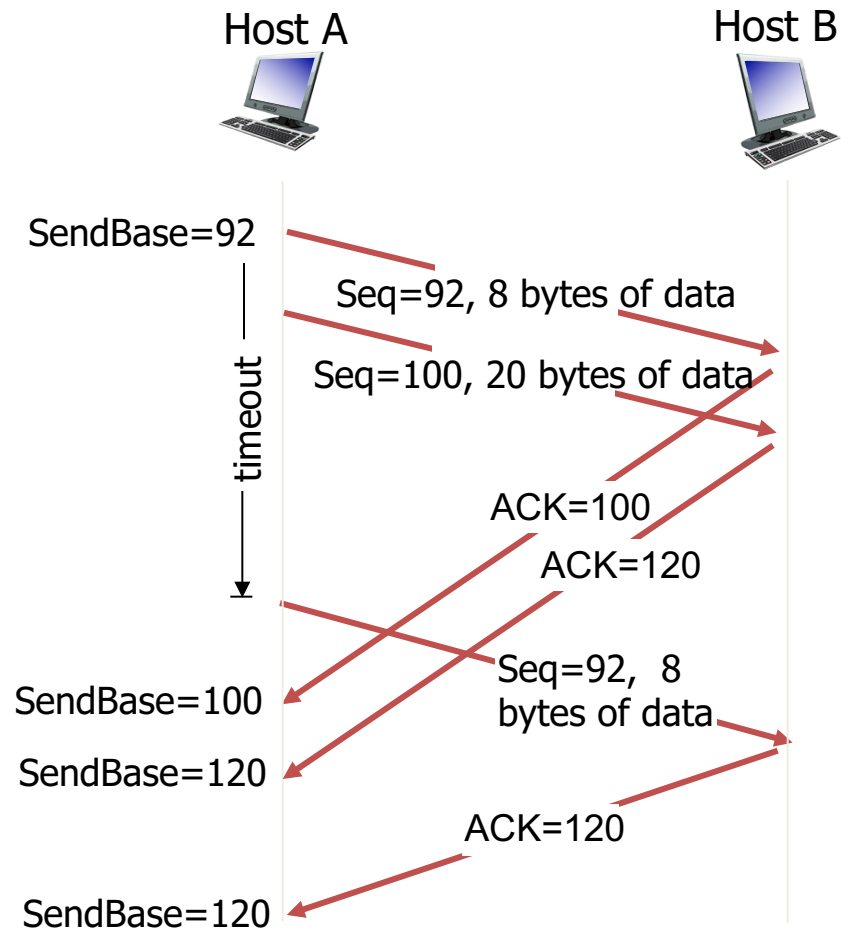
ACK received, with ACK field value y

```
if (y > SendBase) {  
    SendBase = y  
    /* SendBase-1: last cumulatively ACKed byte */  
    if (there are currently not-yet-acked segments)  
        start timer  
    else stop timer  
}
```

# TCP: retransmission scenarios

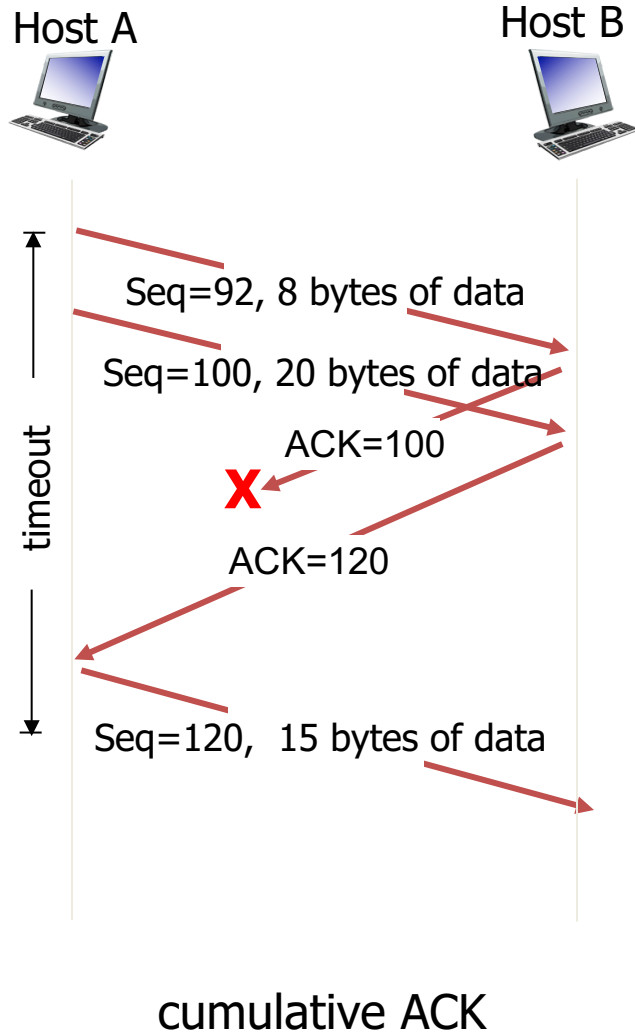


lost ACK scenario



premature timeout

# TCP: retransmission scenarios





# TCP ACK generation [RFC 1122, RFC 2581]

<i>event at receiver</i>	<i>TCP receiver action</i>
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	<b>delayed ACK</b> . Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	<b>immediately send</b> single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expected seq. # . Gap detected	immediately send <b>duplicate ACK</b> , indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	<b>immediate send ACK</b> , provided that segment starts at lower end of gap

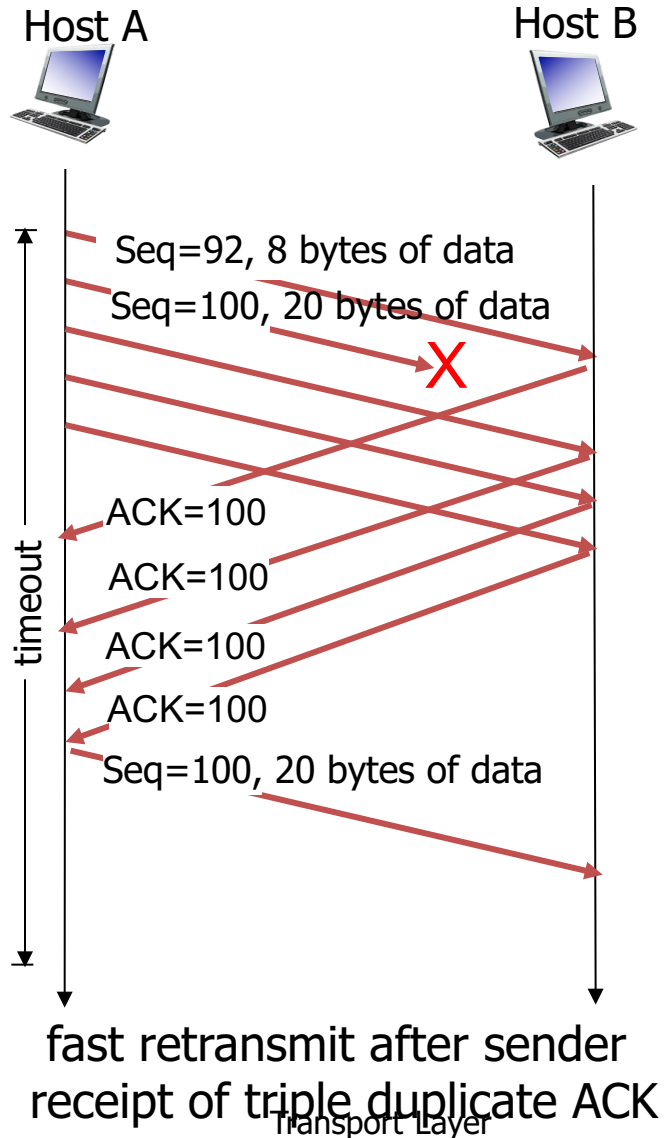
# TCP fast retransmit

---

if sender receives 3 ACKs for same data (“triple duplicate ACKs”), resend unacked segment with smallest (“triple duplicate ACKs”),

- likely that unacked segment lost, so don't wait for timeout

# TCP fast retransmit



# Chapter 3 outline

---

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

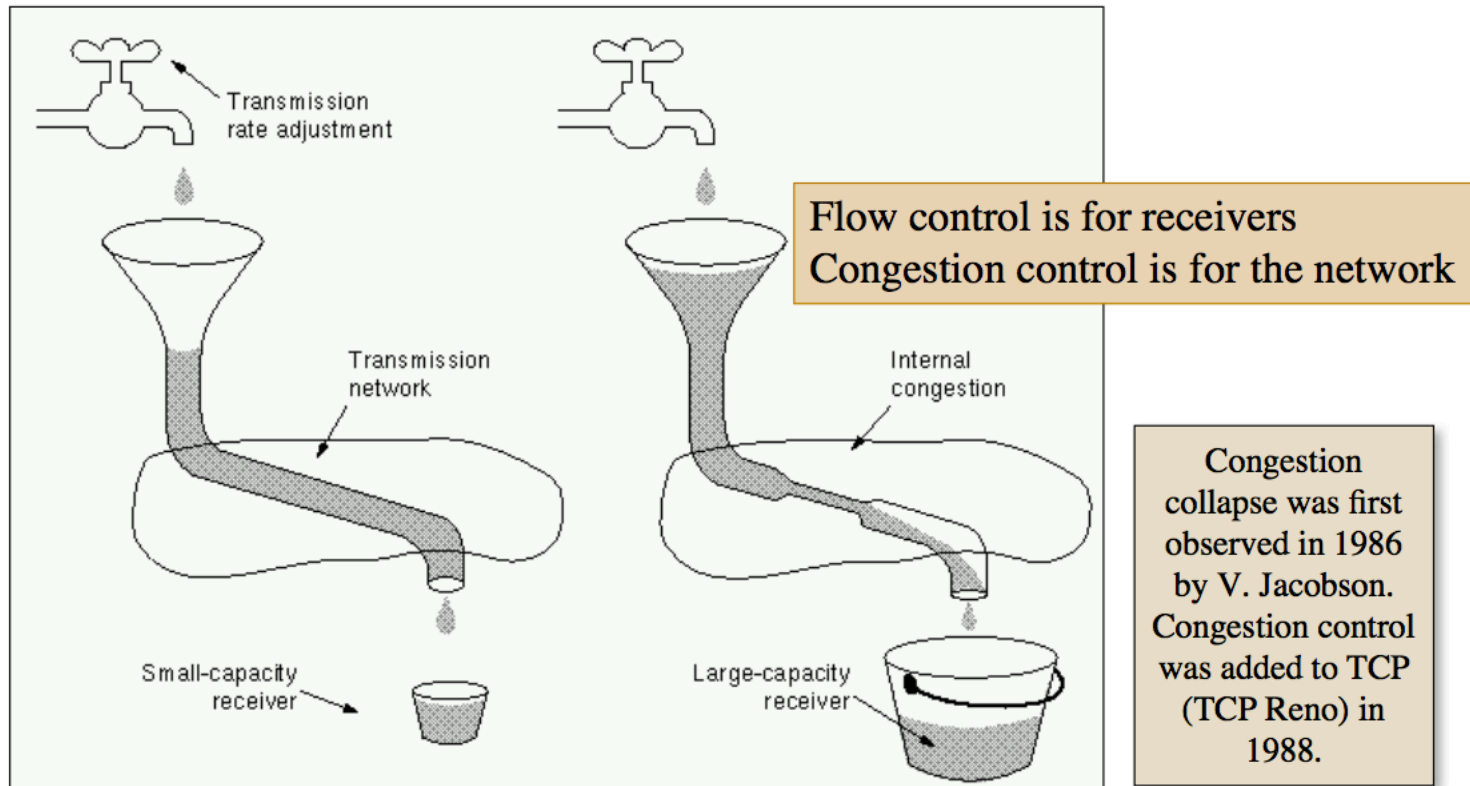
3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

# Let's go back to the origin!

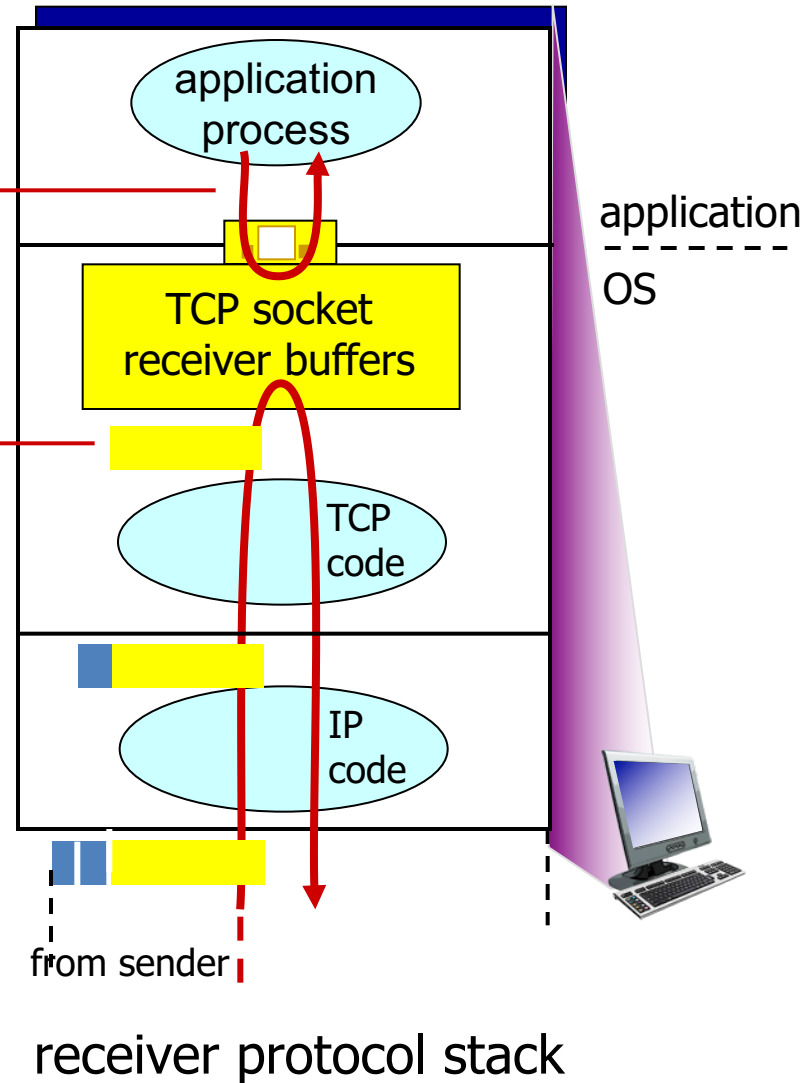


From Computer Networks, A. Tanenbaum

# TCP flow control

application may remove data from TCP socket buffers ....

... slower than TCP receiver is delivering (sender is sending)



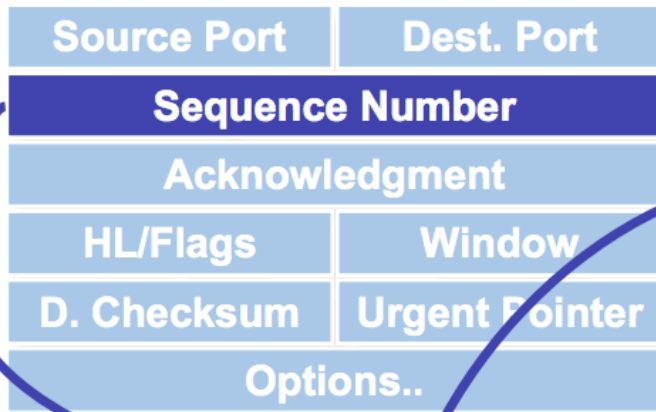
## *flow control*

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

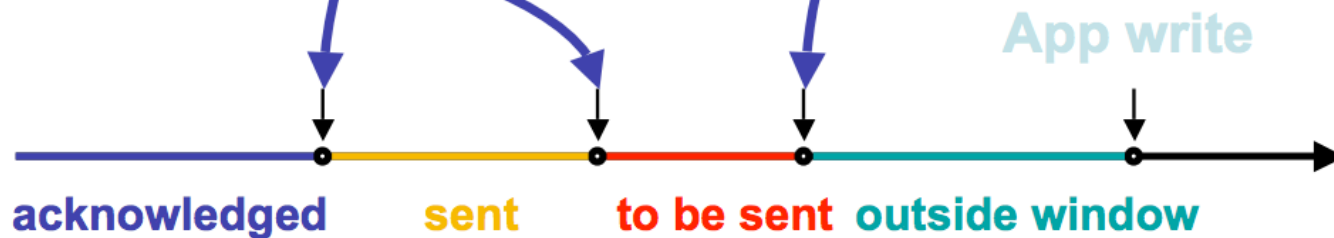
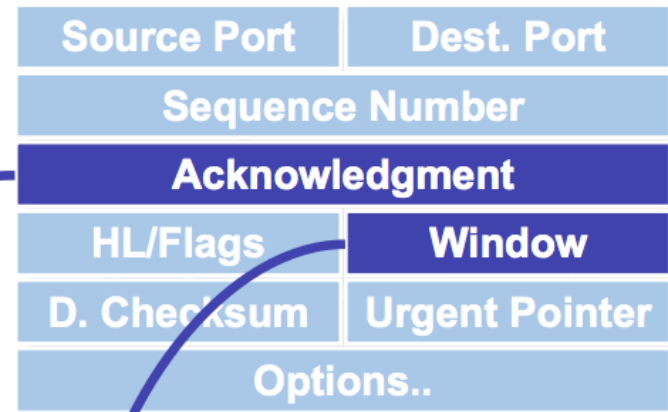
# Flow control

prevents receiver's buffer overflow

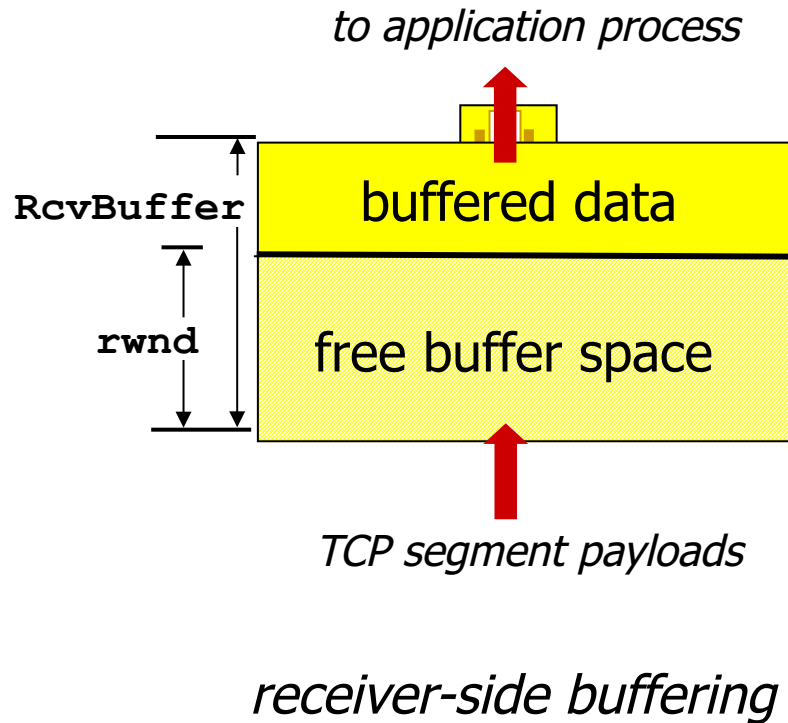
## Packet Sent



## Packet Received



# TCP flow control





# Chapter 3 outline

---

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

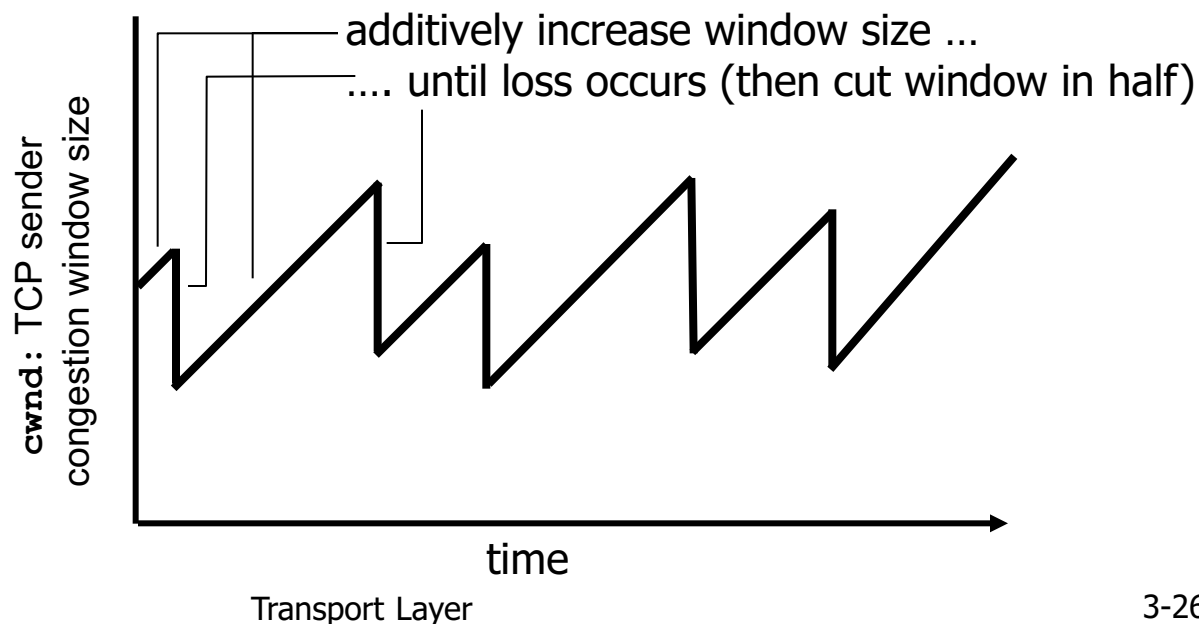
3.6 principles of congestion control

3.7 TCP congestion control

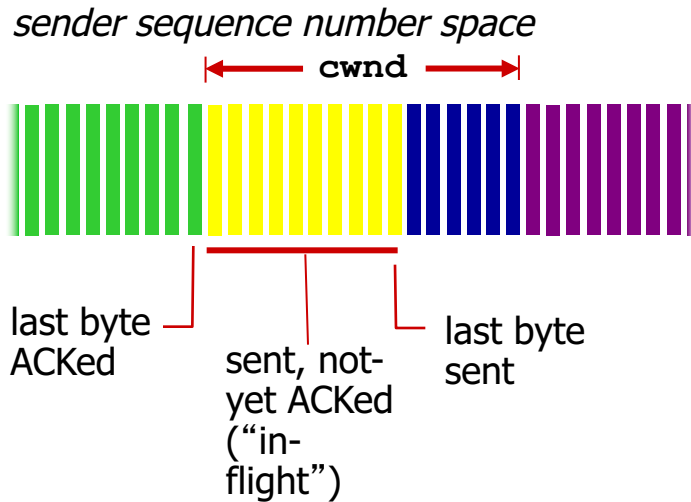
# TCP congestion control: additive increase multiplicative decrease

- *approach*: sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase*: increase **cwnd** by 1 MSS every RTT until loss detected
  - *multiplicative decrease*: cut **cwnd** in half after loss

AIMD saw tooth behavior: probing for bandwidth



# TCP Congestion Control: details



- sender limits transmission:

$$\text{LastByteSent} - \text{LastByteAked} \leq \text{cwnd}$$

*TCP sending rate:*

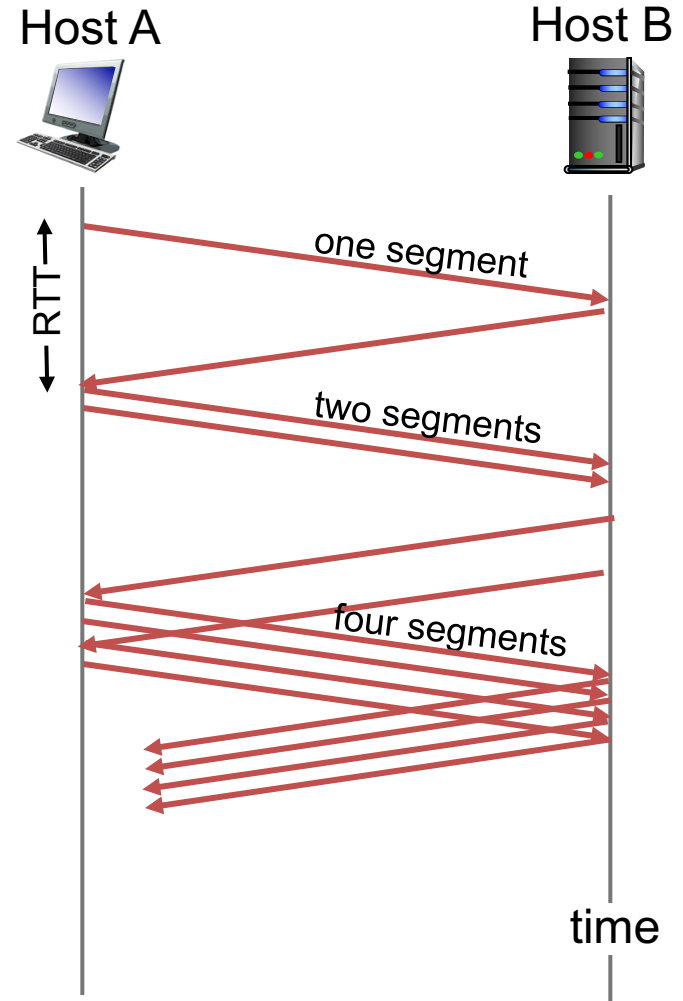
- *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

- **cwnd** is dynamic, function of perceived network congestion

# TCP Slow Start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received
- *summary*: initial rate is slow but ramps up exponentially fast



# TCP: detecting, reacting to loss

---

- loss indicated by timeout:
  - **cwnd** set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold, then grows linearly
- loss indicated by 3 duplicate ACKs: **TCP Reno**
  - dup ACKs indicate network capable of delivering some segments
  - **cwnd** is cut in half window then grows linearly
- **TCP Tahoe** always sets **cwnd** to 1 (timeout or 3 duplicate acks)

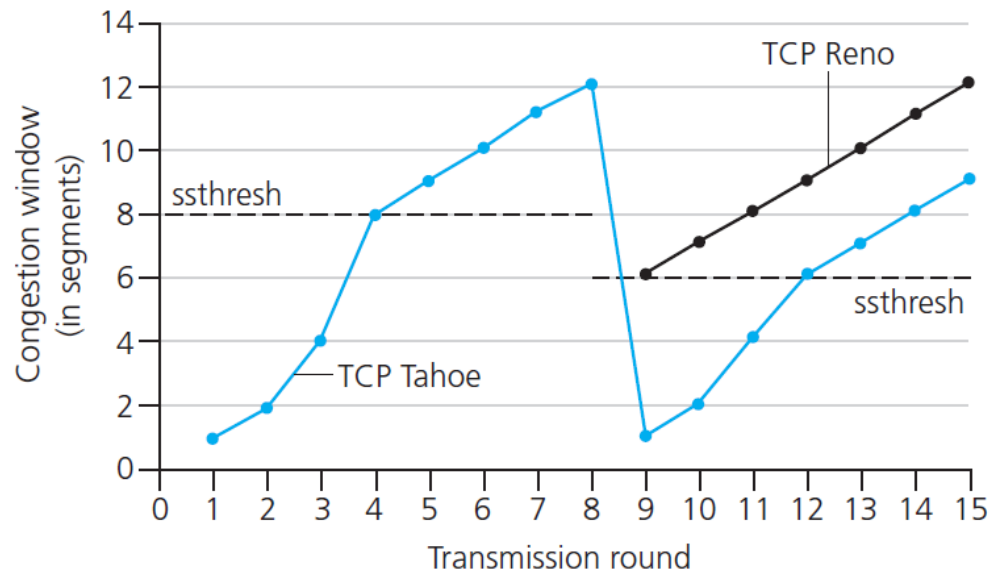
# TCP: switching from slow start to CA

**Q:** when should the exponential increase switch to linear?

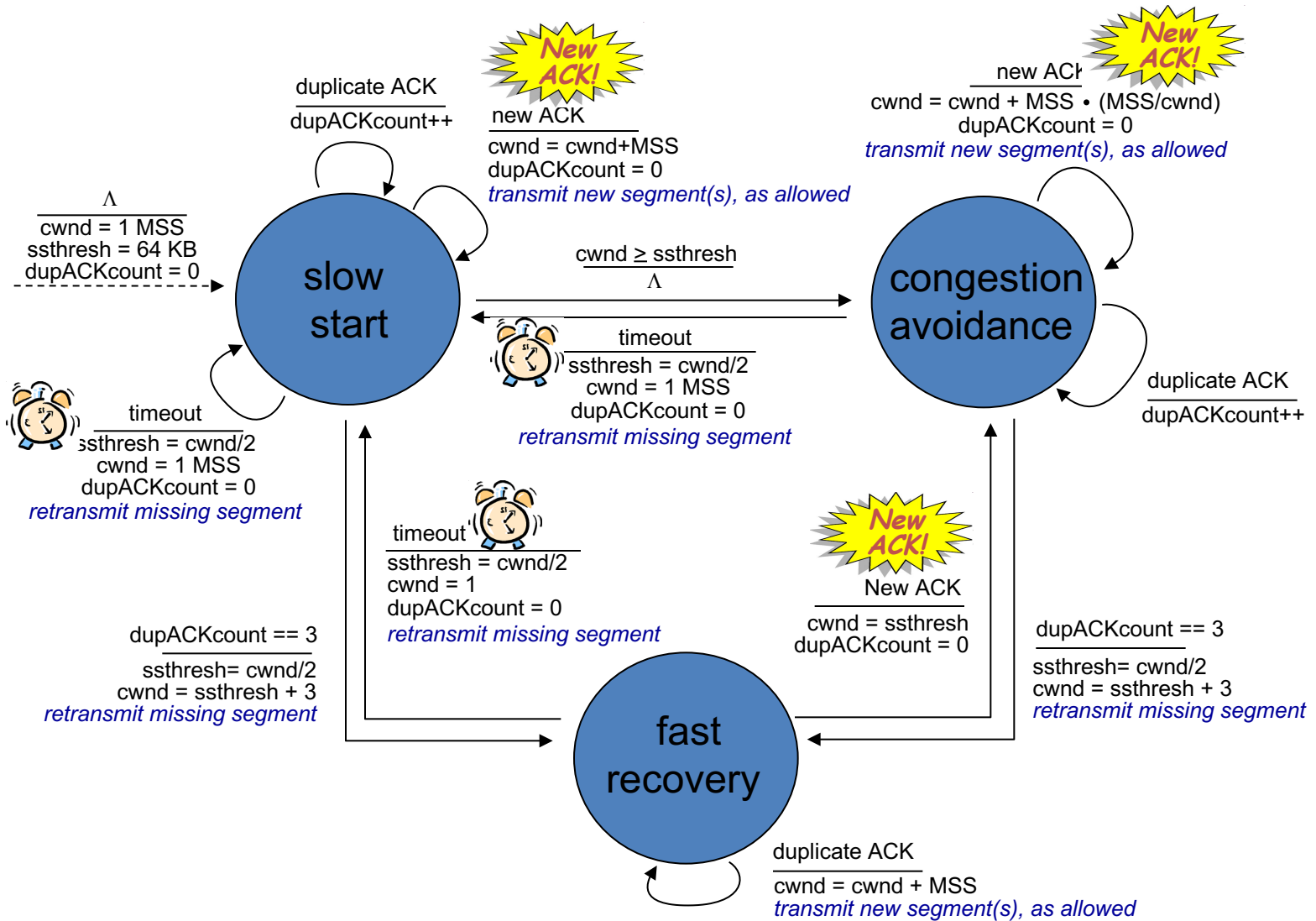
**A:** when **cwnd** gets to  $1/2$  of its value before timeout.

## Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to  $1/2$  of **cwnd** just before loss event



# Summary: TCP Congestion Control

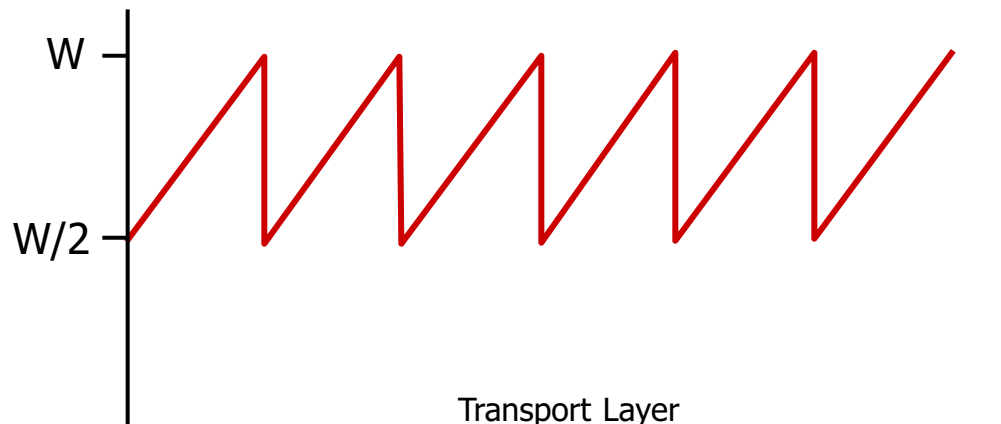


# TCP throughput

- avg. TCP thruput as function of window size, RTT?
  - ignore slow start, assume always data to send
- $W$ : window size (measured in bytes) where loss occurs
  - avg. window size (# in-flight bytes) is  $\frac{3}{4} W$

$$\text{avg TCP thruput} = \frac{3}{4} \frac{W}{\text{RTT}} \text{ bytes/sec}$$

- avg. throughput is  $\frac{3}{4}W$  per RTT





# TCP Futures: TCP over “long, fat pipes”

---

- example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- requires  $W = 83,333$  in-flight segments
- throughput in terms of segment loss probability,

L [Mathis 1997]:

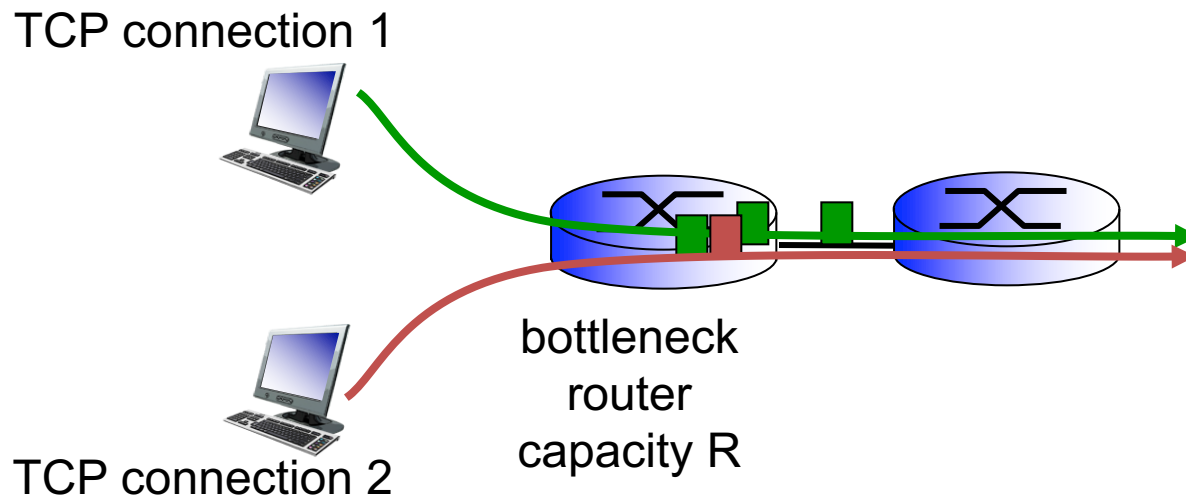
$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT} \sqrt{L}}$$

→ to achieve 10 Gbps throughput, need a loss rate of  $L = 2 \cdot 10^{-10}$  – *a very small loss rate!*

- new versions of TCP for high-speed (RFC 3649)

# TCP Fairness

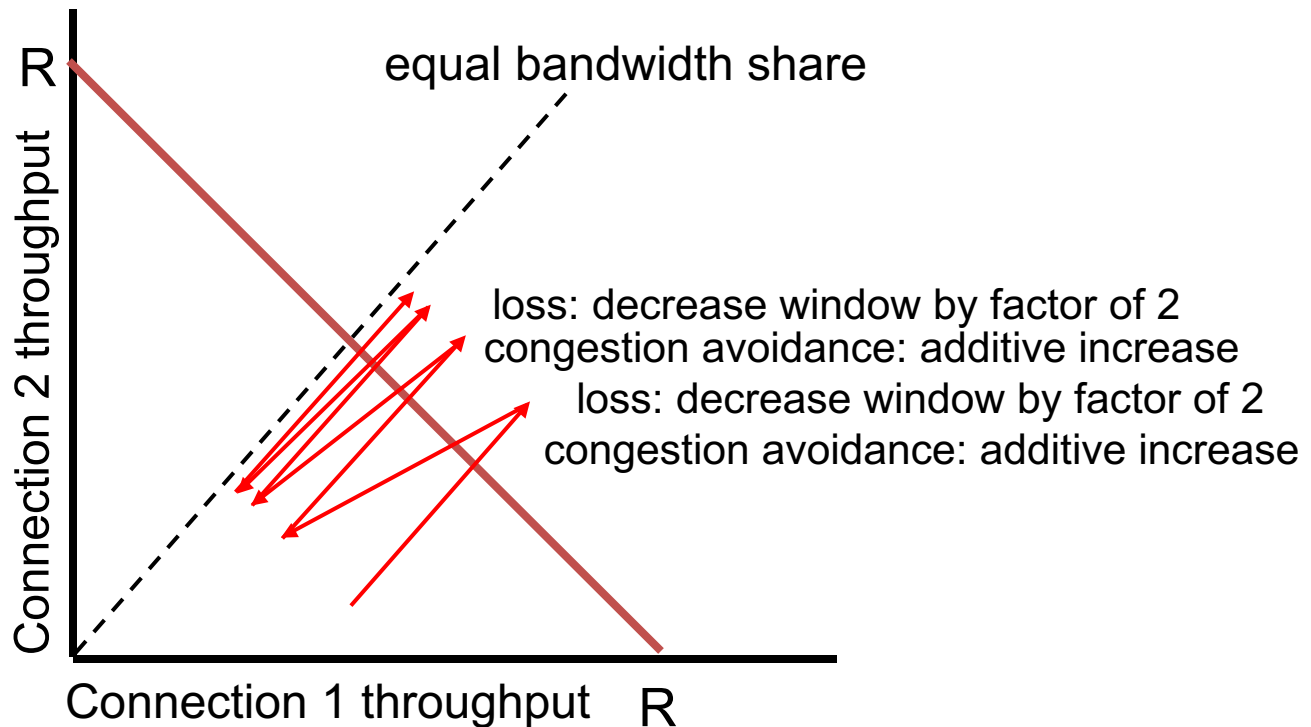
*fairness goal:* if  $K$  TCP sessions share same bottleneck link of bandwidth  $R$ , each should have average rate of  $R/K$



# Why is TCP fair?

two competing sessions:

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



# Fairness (more)

---

## *Fairness and UDP*

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss

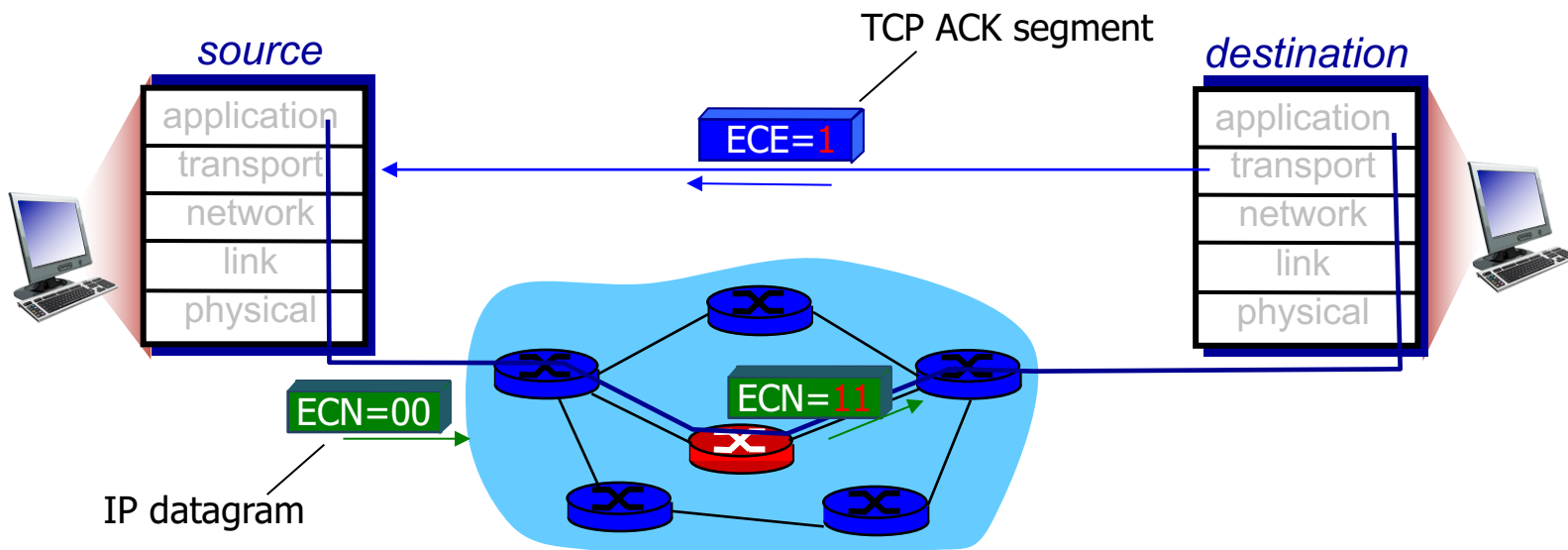
## *Fairness, parallel TCP connections*

- application can open multiple parallel connections between two hosts
- web browsers do this
- e.g., link of rate  $R$  with  $g$  existing connections:
  - new app asks for 1 TCP, gets rate  $R/10$
  - new app asks for 11 TCPs, gets  $R/2$

# Explicit Congestion Notification (ECN)

*network-assisted congestion control:*

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
- congestion indication carried to receiving host
- receiver (seeing congestion indication in IP datagram) sets ECE bit on receiver-to-sender ACK segment to notify sender of congestion



# Chapter 3 outline

---

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

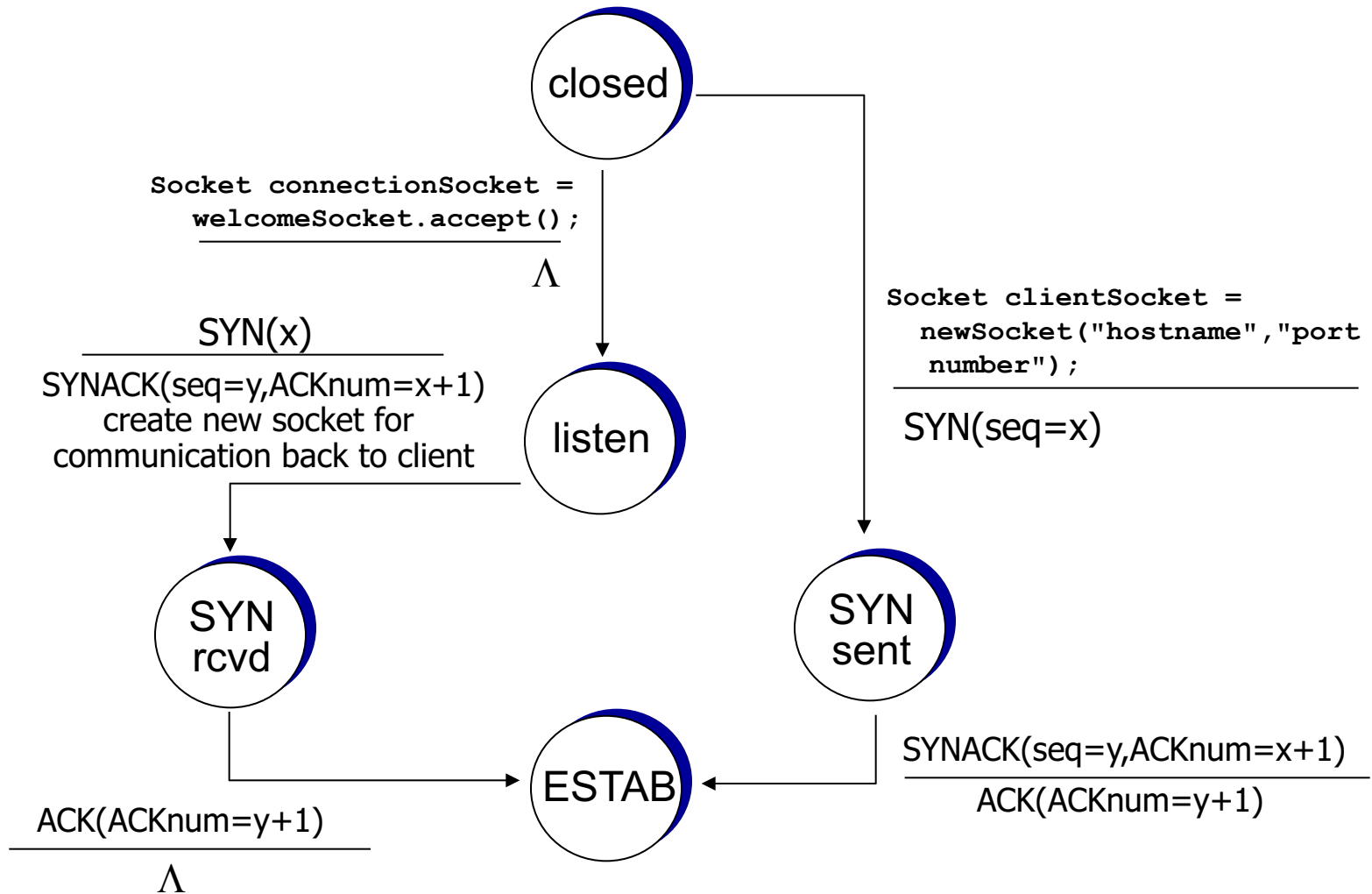
3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

# TCP 3-way handshake: FSM



## TCP: closing a connection

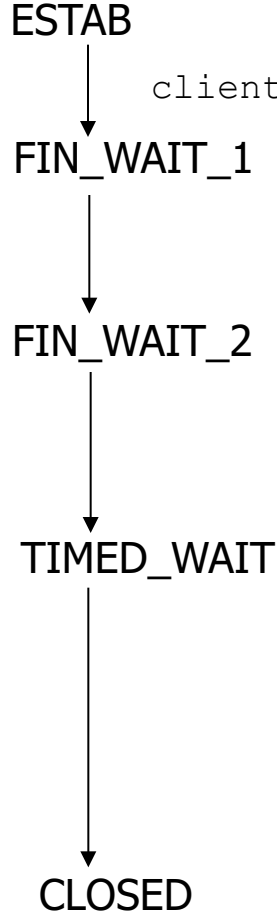
---

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled



# TCP: closing a connection

*client state*

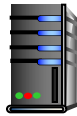


`clientSocket.close()`

can no longer send but can receive data

wait for server close

timed wait for  $2 * \text{max segment lifetime}$



FINbit=1, seq=x

ACKbit=1; ACKnum=x+1

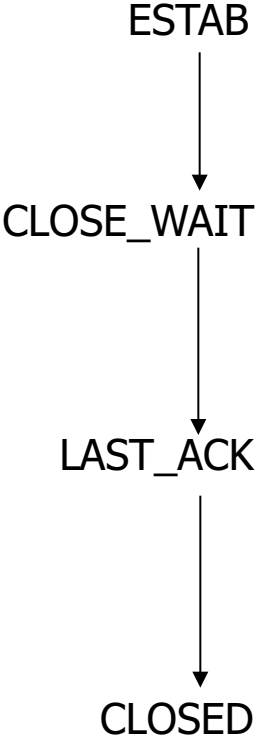
FINbit=1, seq=y

ACKbit=1; ACKnum=y+1

can still send data

can no longer send data

*server state*



# Acknowledgement

- Most of the slides in this presentation are taken from the slides provided by the authors of the textbook.
  
- Thanks to YouTube for providing many useful videos.
  - If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
  - If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR