CMPE 150/L : Introduction to Computer Networks

Chen Qian

Computer Engineering

UCSC Baskin Engineering

Lecture 10

Midterm exam

- Midterm next Thursday
- Close book but one-side 8.5"x11" note is allowed (must use hand-writing!)
- Let me know by next Monday if you have any problem
- Sample midterm and sample question of Chapter 2&3

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

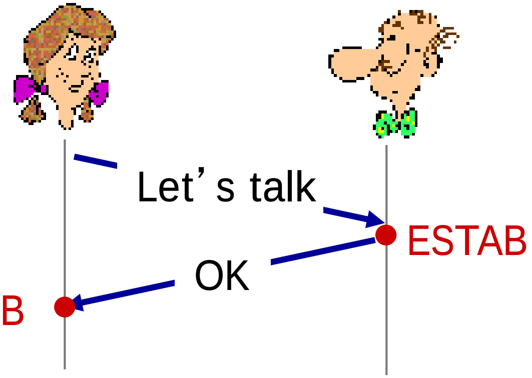
3.5 connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management

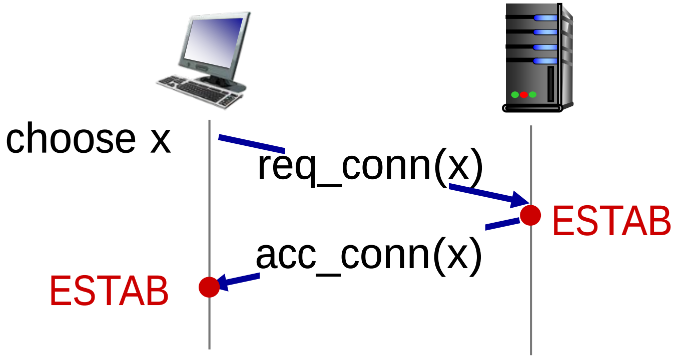
3.6 principles of congestion control

3.7 TCP congestion control

Agreeing to establish a connection

2-way handshake:



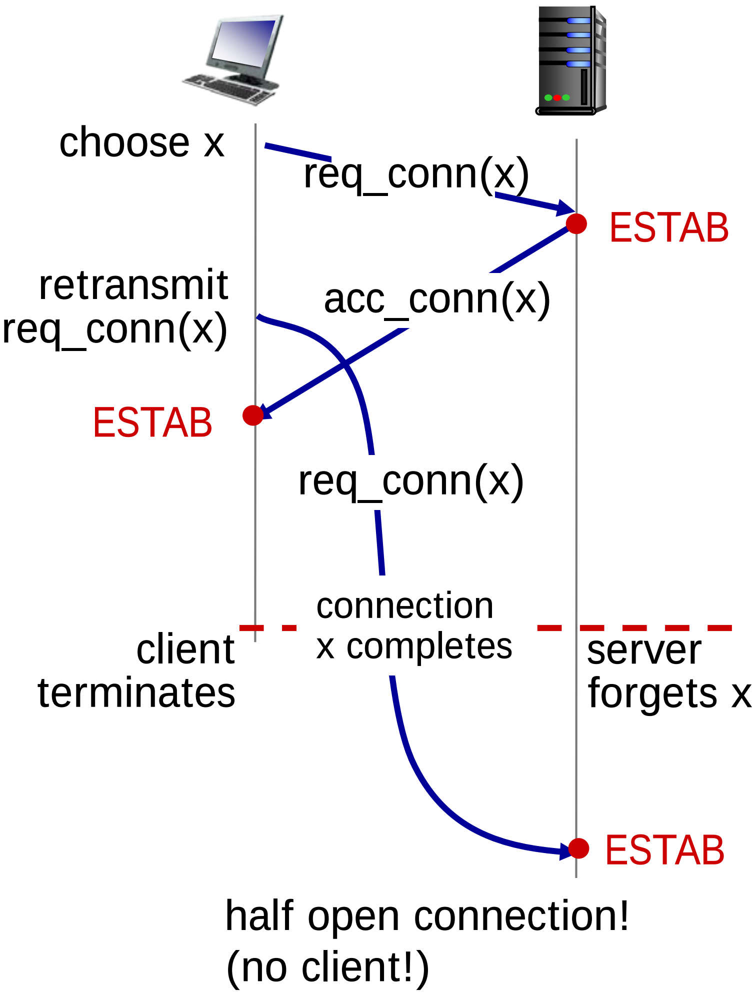


Q: will 2-way handshake always work in network?

- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can’t “see” other side

Agreeing to establish a connection

2-way handshake failure scenarios:



choose x

req_conn(x)

ESTAB

retransmit
req_conn(x)

acc_conn(x)

ESTAB

req_conn(x)

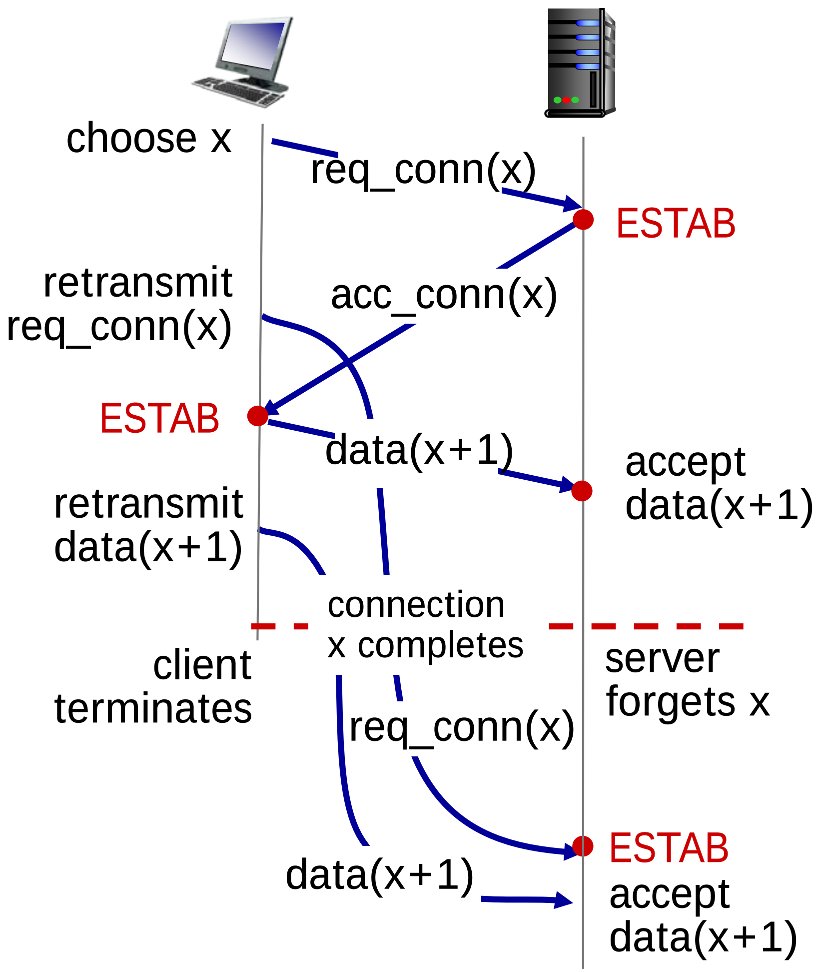
connection
x completes

client
terminates

server
forgets x

ESTAB

half open connection!
(no client!)



choose x

req_conn(x)

ESTAB

retransmit
req_conn(x)

acc_conn(x)

ESTAB

data(x+1)

accept
data(x+1)

retransmit
data(x+1)

connection
x completes

client
terminates

server
forgets x

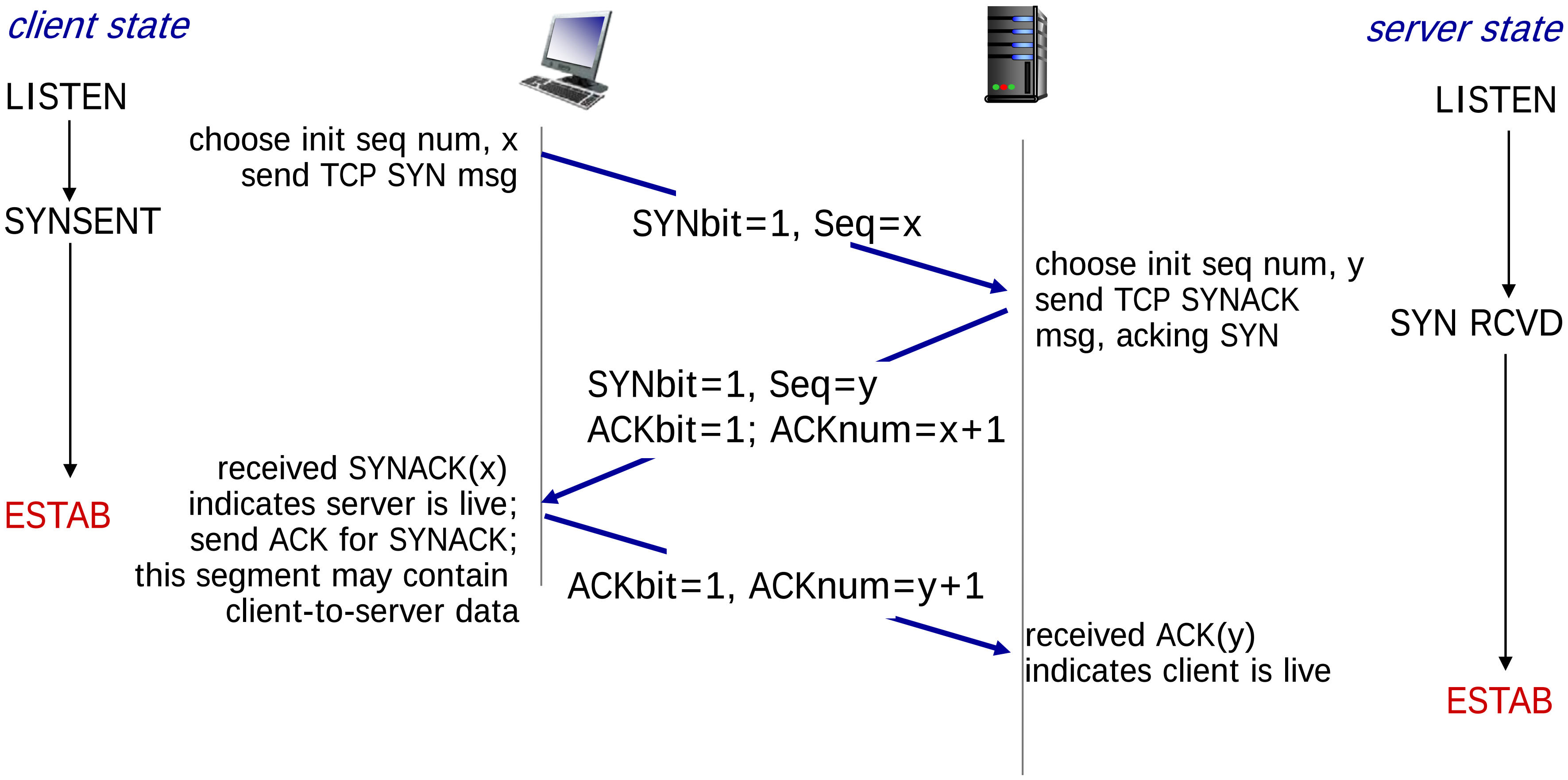
req_conn(x)

ESTAB

data(x+1)

accept
data(x+1)

TCP 3-way handshake



client state

LISTEN

SYNSENT

ESTAB

choose init seq num, x
send TCP SYN msg

SYNbit=1, Seq=x

server state

LISTEN

choose init seq num, y
send TCP SYNACK
msg, acking SYN

SYN RCVD

SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1

received SYNACK(x)
indicates server is live;
send ACK for SYNACK;
this segment may contain
client-to-server data

ACKbit=1, ACKnum=y+1

received ACK(y)
indicates client is live

ESTAB

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

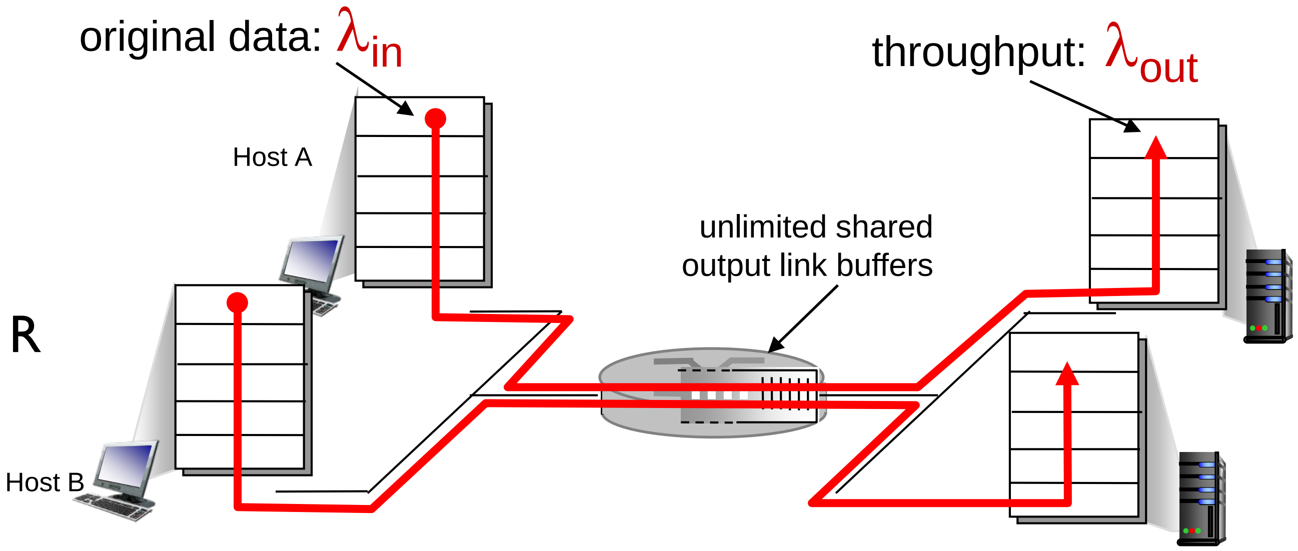
Principles of congestion control

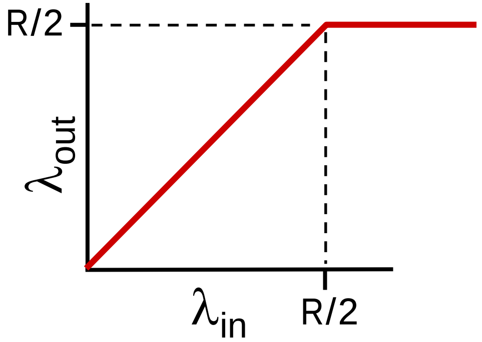
congestion:

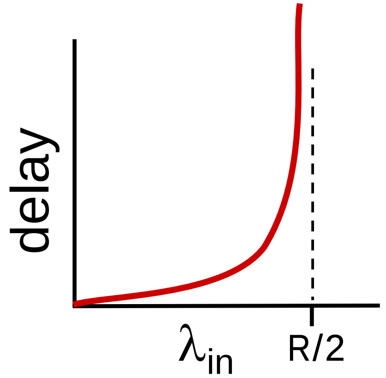
- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- output link capacity: R
- no retransmission



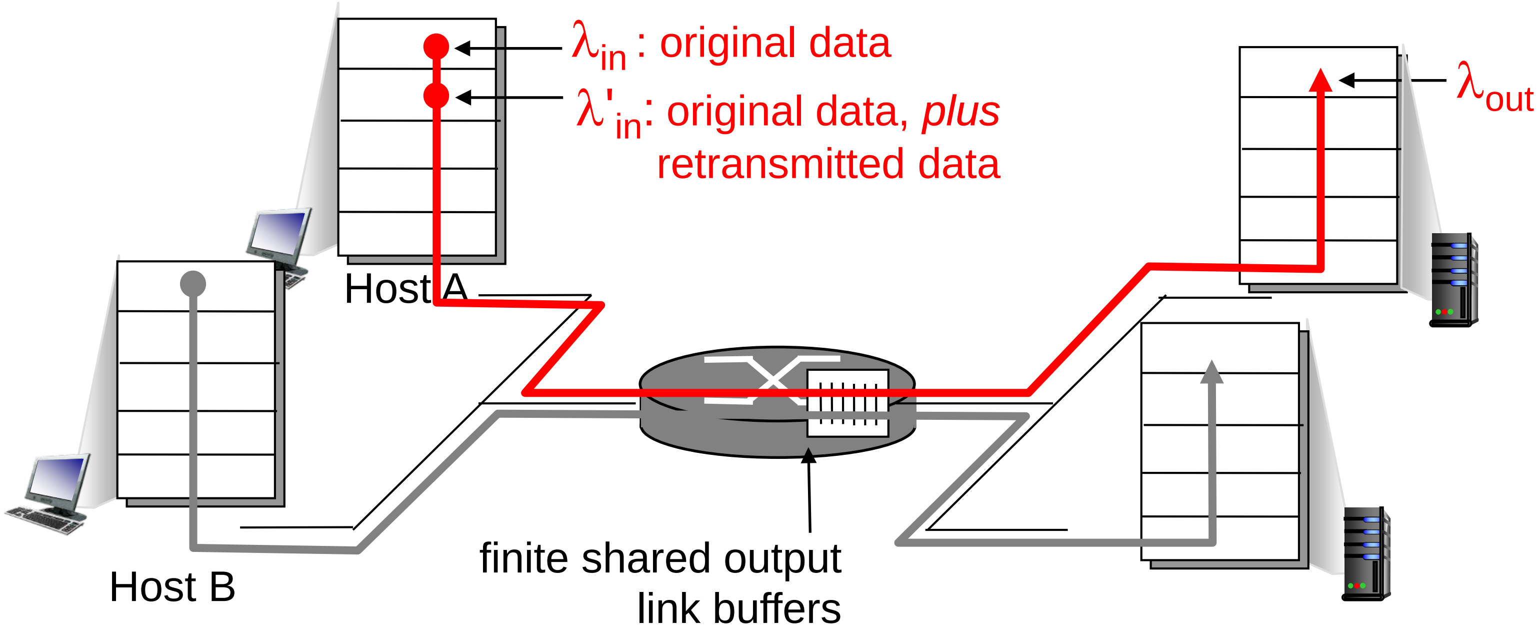




- maximum per-connection throughput: R/2
- large delays as arrival rate, λ_{in}, approaches capacity

Causes/costs of congestion: scenario 2

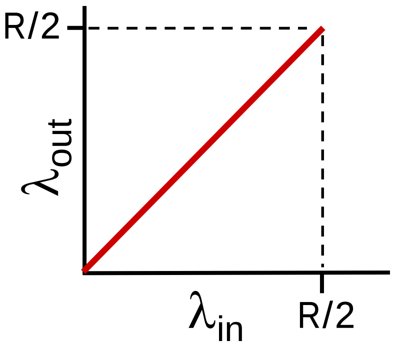
- one router, *finite* buffers
- sender retransmission of timed-out packet
  - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$

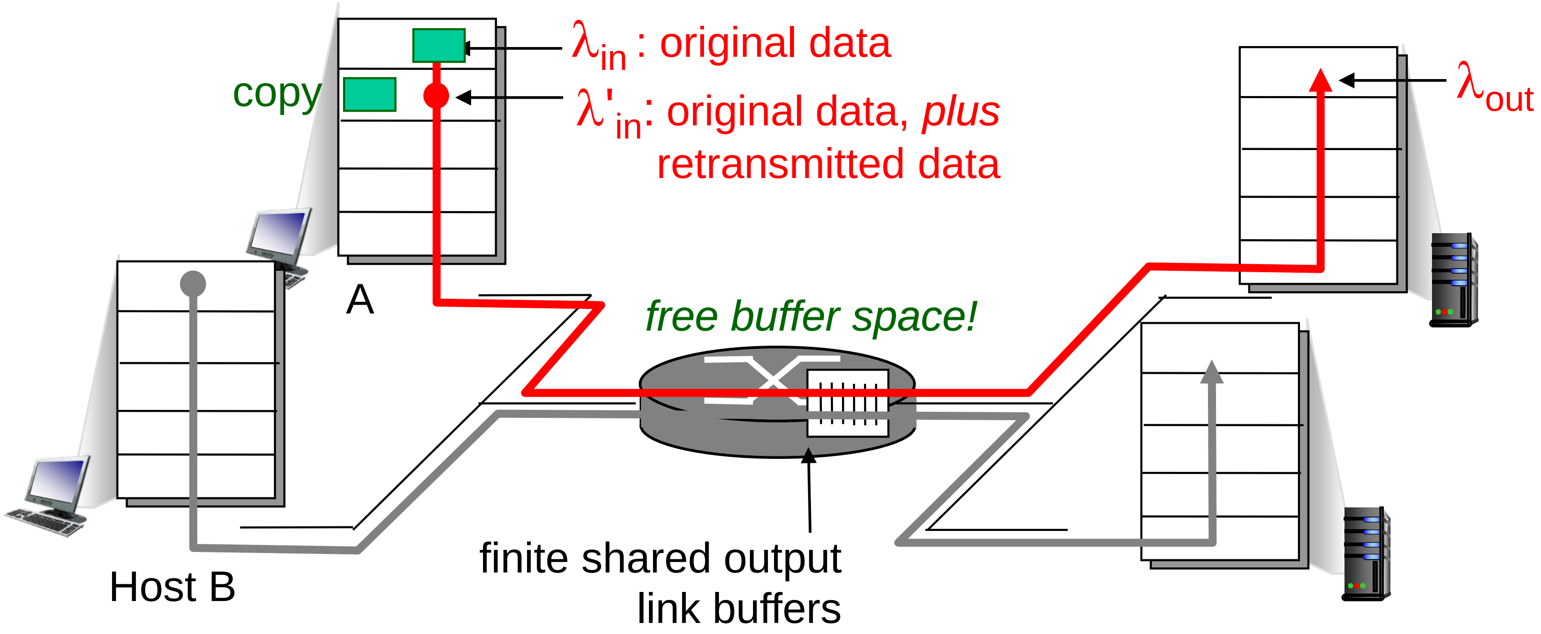


Causes/costs of congestion: scenario 2

idealization: perfect knowledge

- sender sends only when router buffers available





λ_{in} : original data

λ'_{in}: original data, *plus* retransmitted data

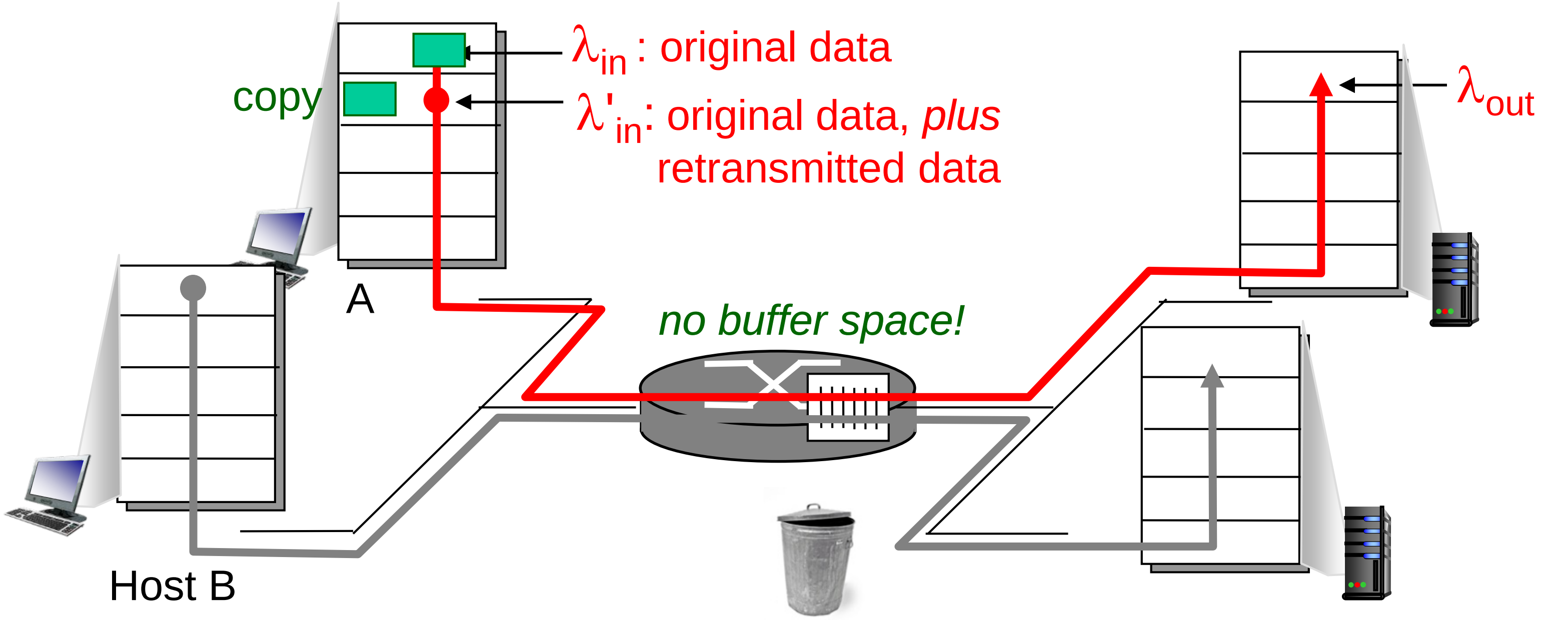
λ_{out}

finite shared output link buffers

Causes/costs of congestion: scenario 2

Idealization: known loss
packets can be lost, dropped at router due to full buffers

- sender only resends if packet *known* to be lost

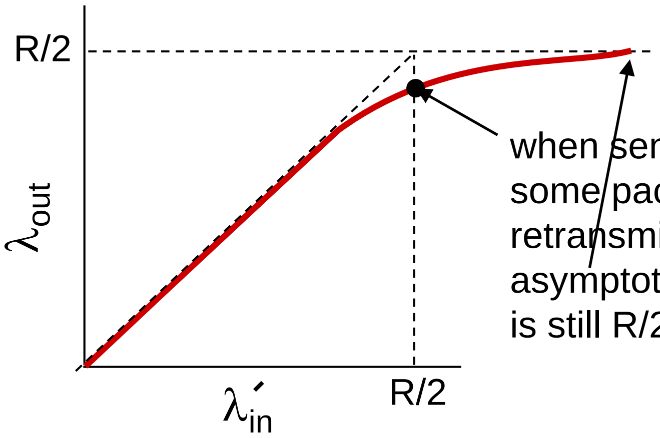


Causes/costs of congestion: scenario 2

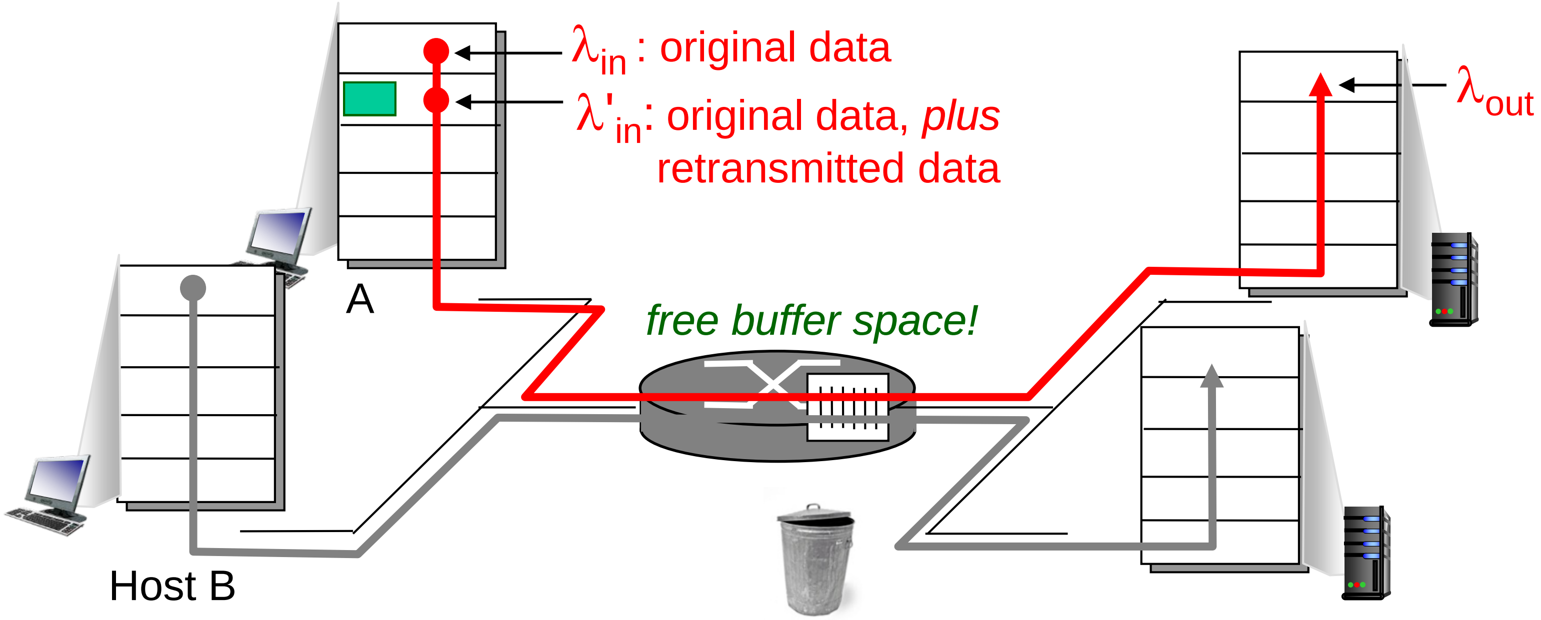
Idealization: known loss

packets can be lost, dropped at router due to full buffers

- sender only resends if packet *known* to be lost



when sending at R/2, some packets are retransmissions but asymptotic goodput is still R/2 (why?)



λ_{in} : original data

λ'_{in}: original data, *plus* retransmitted data

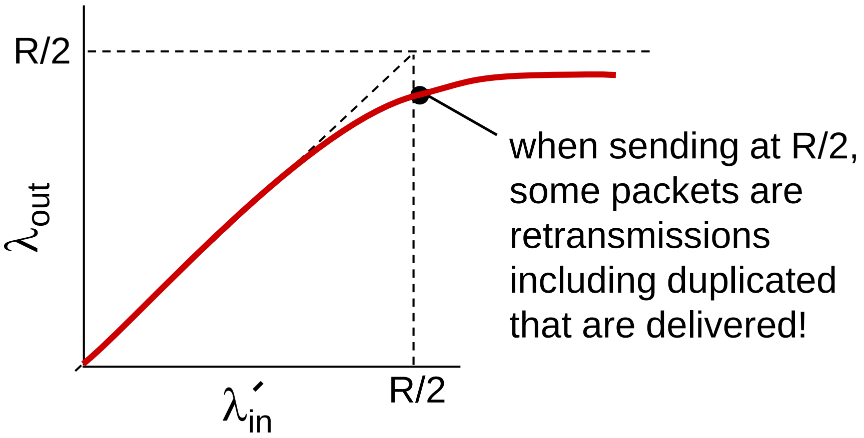
λ_{out}

free buffer space!

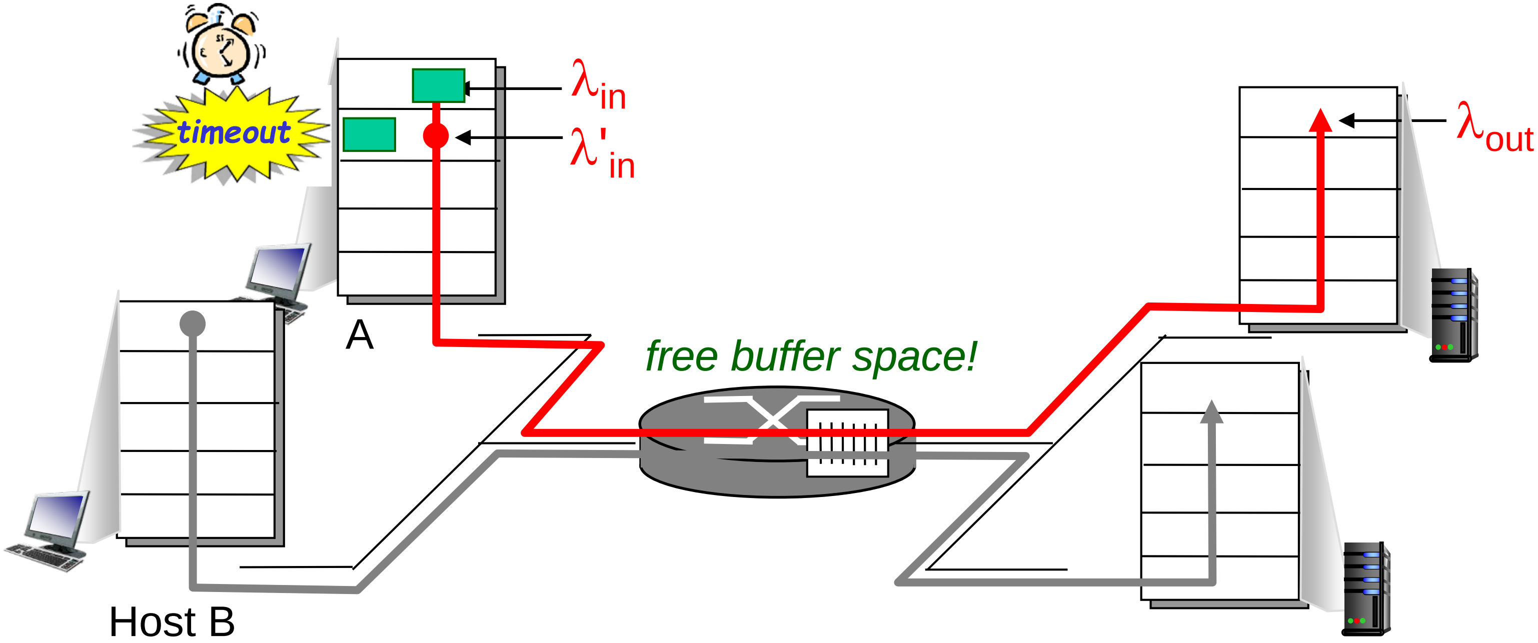
Causes/costs of congestion: scenario 2

Realistic: duplicates

- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending *two* copies, both of which are delivered



when sending at R/2, some packets are retransmissions including duplicated that are delivered!



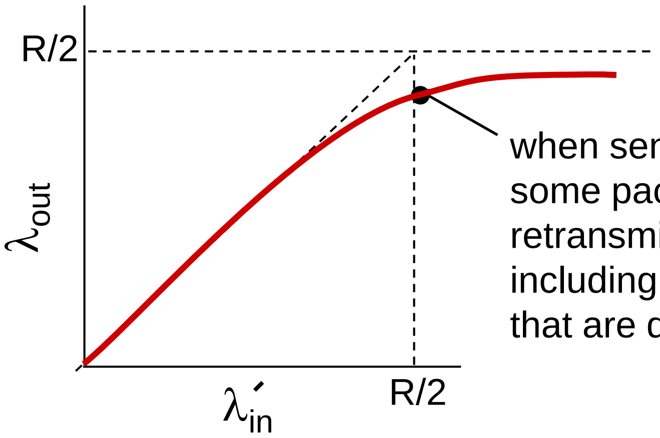
free buffer space!

- Throughput:
  - Data rate at the receiver
- Goodput:
  - Rate at the receiver for data without duplicate!

Causes/costs of congestion: scenario 2

Realistic: duplicates

- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending *two* copies, both of which are delivered



when sending at R/2, some packets are retransmissions including duplicated that are delivered!

"costs" of congestion:

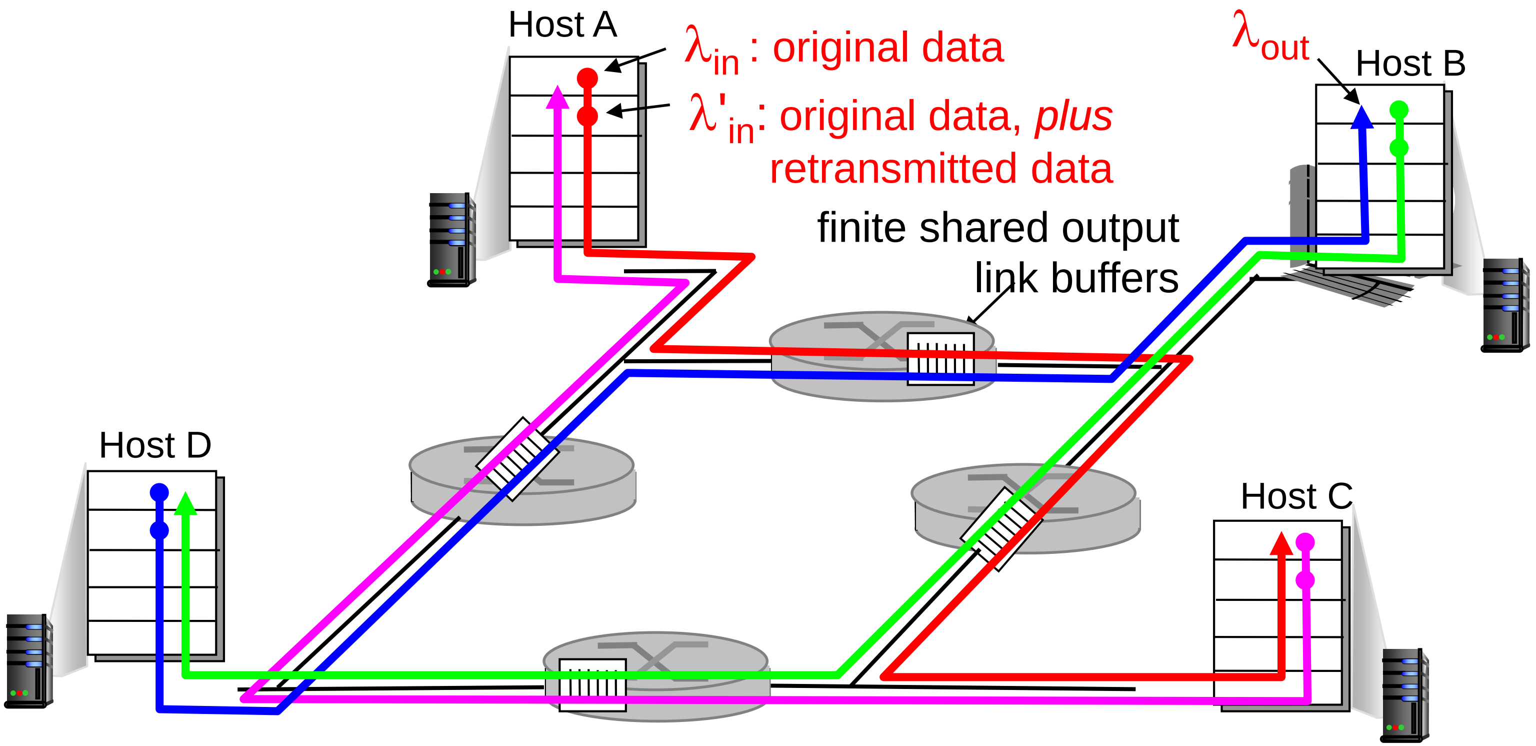
- more work (retrans) for given "goodput"
- unneeded retransmissions: link carries multiple copies of pkt
  - decreasing goodput

Causes/costs of congestion: scenario 3

- four senders
- multihop paths
- timeout/retransmit

Q: what happens as λ_{in} and λ_{in}' increase ?

A: as red λ_{in}' increases, all arriving blue pkts at upper queue are dropped, blue throughput → 0



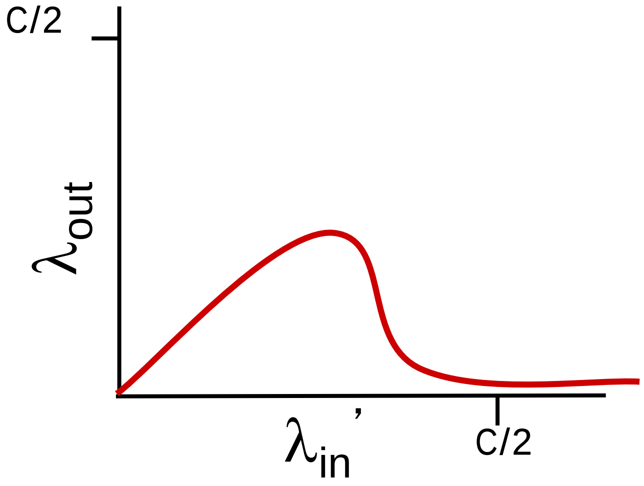
λ_{in} : original data

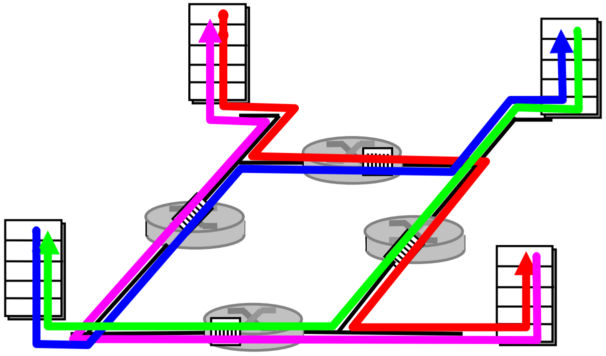
λ'_{in}: original data, *plus* retransmitted data

finite shared output link buffers

λ_{out}

Causes/costs of congestion: scenario 3





another "cost" of congestion:

- when packet dropped, any "upstream transmission capacity used for that packet was wasted!

Approaches towards congestion control

two broad approaches towards congestion control:

end-end congestion control:

- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

network-assisted congestion control:

- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate for sender to send at

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

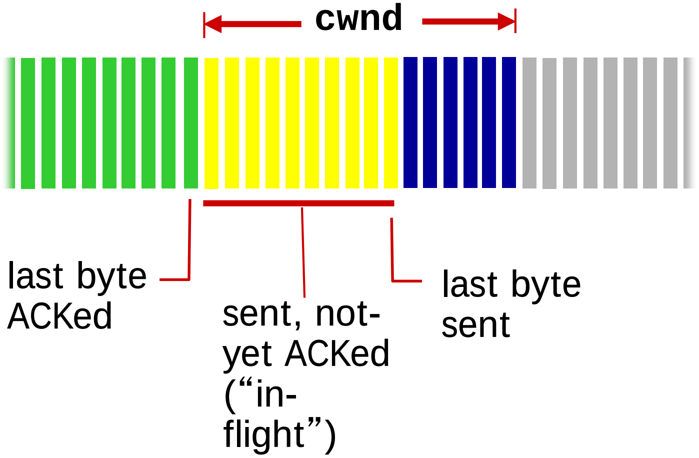
3.4 principles of reliable data transfer

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: details



sender sequence number space

cwnd

last byte ACKed

sent, not-yet ACKed (“in-flight”)

last byte sent

- sender limits transmission:

$$\texttt{LastByteSent-LastByteAcked} \leq \texttt{cwnd}$$

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

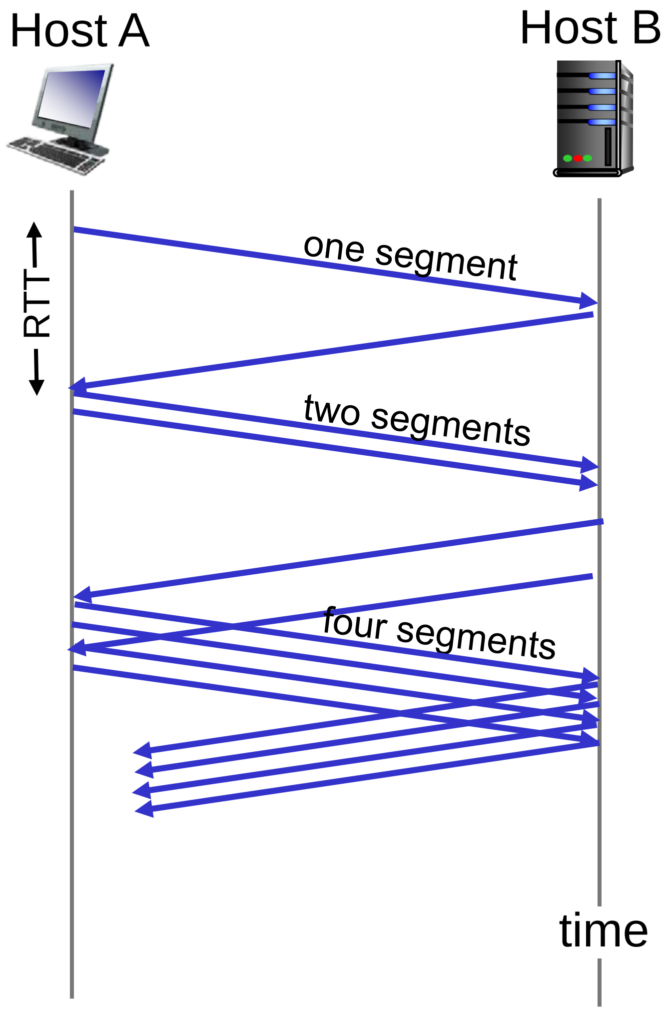
TCP sending rate:

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

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

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

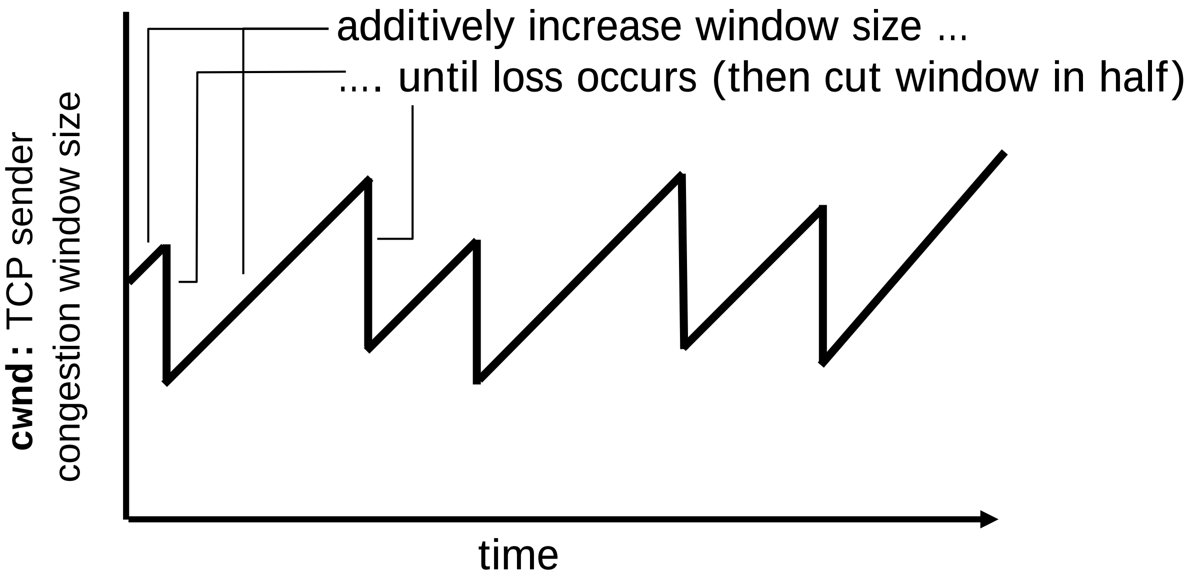
After cwnd reaching the threshold

- Congestion avoidance algorithm:
- Additive increase multiplicative decrease (AIMD)

TCP congestion control: AIMD

- *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



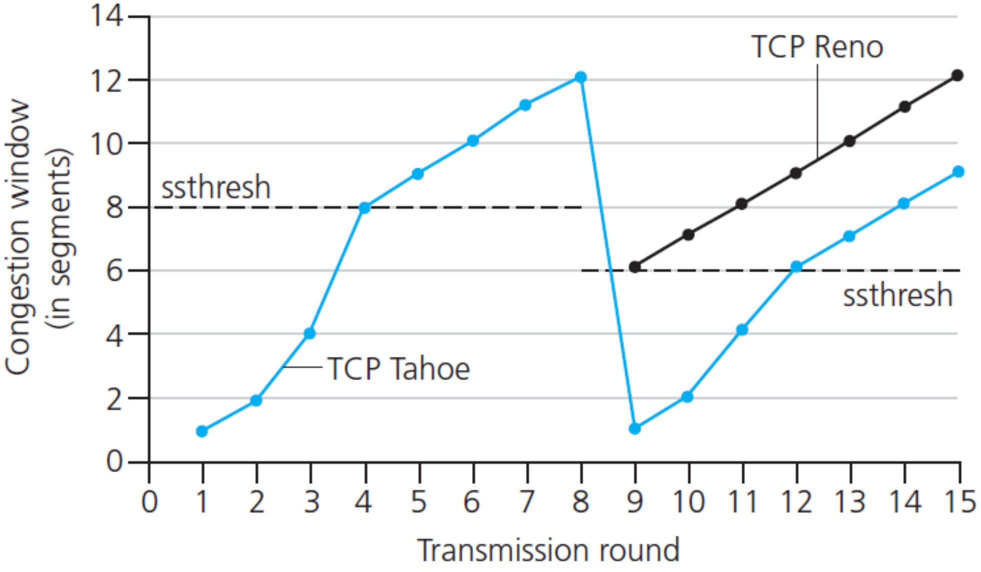
additively increase window size ...

.... until loss occurs (then cut window in half)

TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

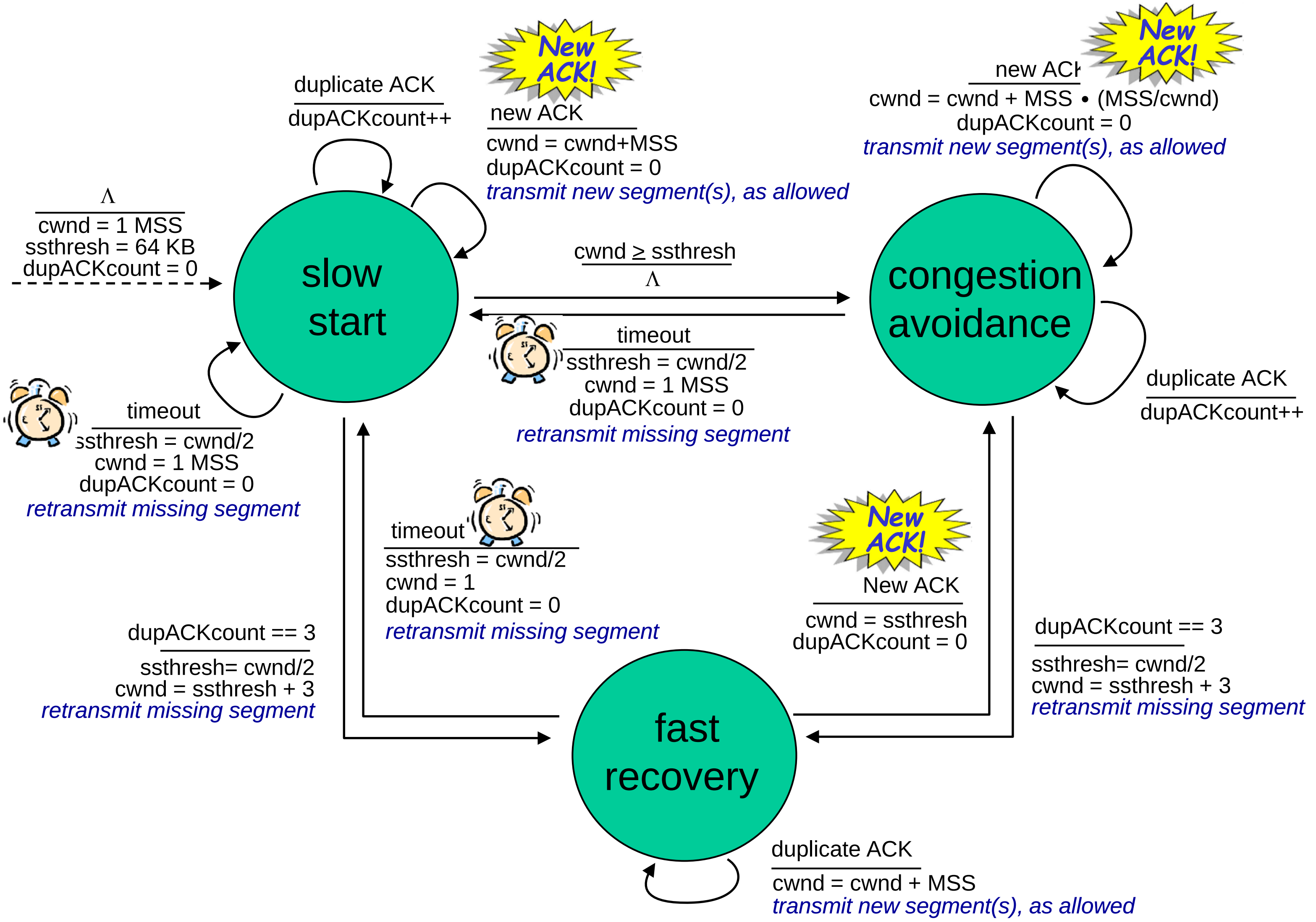
A: when **`cwnd`** gets to 1/2 of its value before timeout.



Implementation:

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

Summary: TCP Congestion Control



Λ
cwnd = 1 MSS
ssthresh = 64 KB
dupACKcount = 0

duplicate ACK
dupACKcount++

new ACK
cwnd = cwnd+MSS
dupACKcount = 0
transmit new segment(s), as allowed

new ACK
cwnd = cwnd + MSS • (MSS/cwnd)
dupACKcount = 0
transmit new segment(s), as allowed

cwnd ≥ ssthresh
Λ

timeout
ssthresh = cwnd/2
cwnd = 1 MSS
dupACKcount = 0
retransmit missing segment

duplicate ACK
dupACKcount++

timeout
ssthresh = cwnd/2
cwnd = 1 MSS
dupACKcount = 0
retransmit missing segment

timeout
ssthresh = cwnd/2
cwnd = 1
dupACKcount = 0
retransmit missing segment

New ACK
cwnd = ssthresh
dupACKcount = 0

dupACKcount == 3
ssthresh= cwnd/2
cwnd = ssthresh + 3
retransmit missing segment

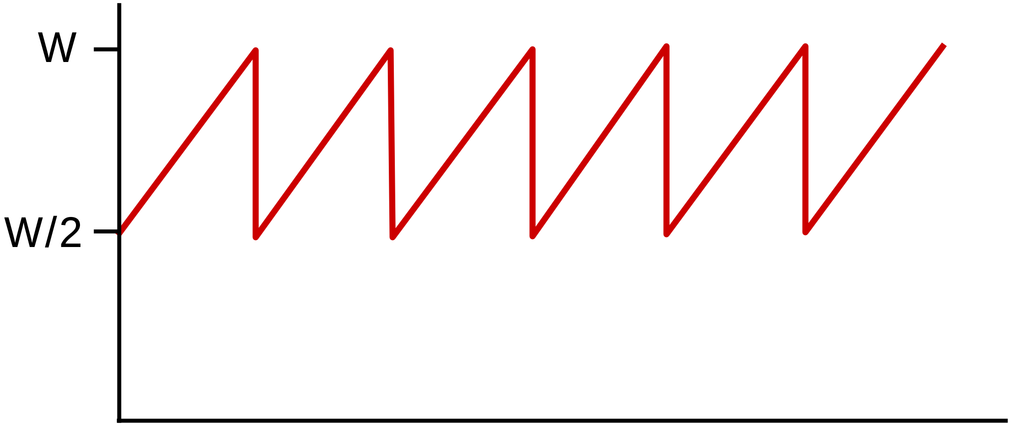
dupACKcount == 3
ssthresh= cwnd/2
cwnd = ssthresh + 3
retransmit missing segment

duplicate ACK
cwnd = cwnd + MSS
transmit new segment(s), as allowed

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 ¾ W
  - avg. thruput is 3/4W per RTT

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



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]:

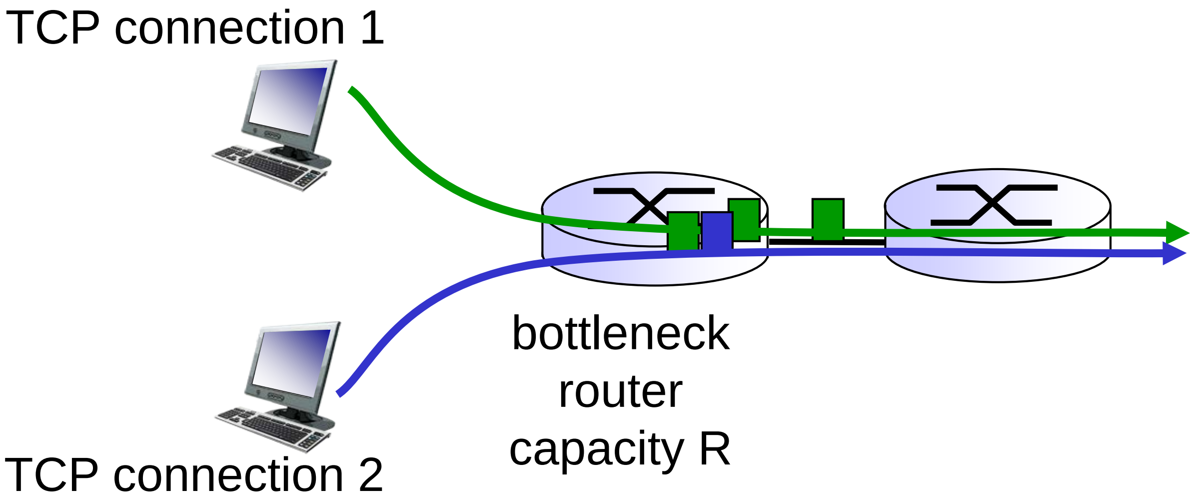
$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT}\sqrt{L}}$$

➜ 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

TCP Fairness

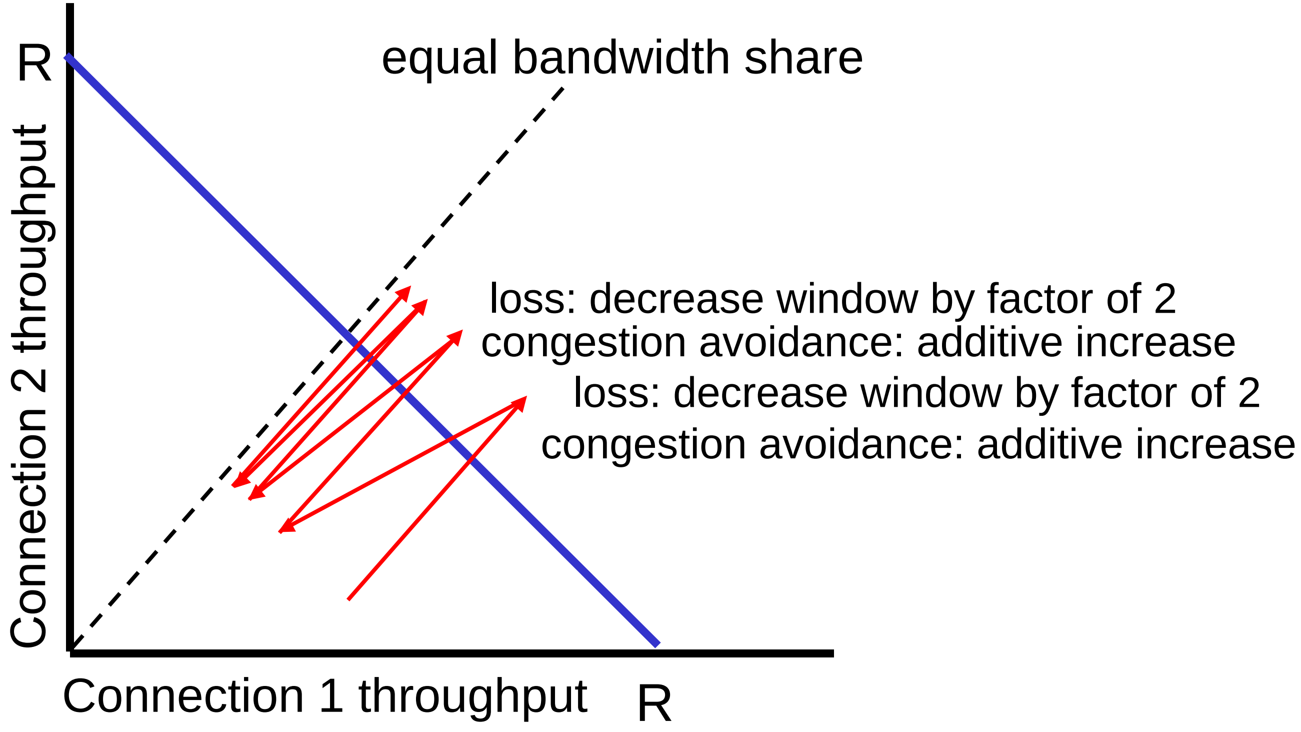
fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

two competing sessions:

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



Van Jacobson

- One of the key designers of TCP congestion control
- https://www.youtube.com/watch?v=QP4A6L7CEqA
- 1:40-9:20

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 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2

Chapter 3: summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
  - UDP
  - TCP

next:

- leaving the network "edge" (application, transport layers)
- into the network "core"

Next class

- Midterm covers every slide until here.
- Please read Chapter 4.1-4.2 of your textbook BEFORE Class