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
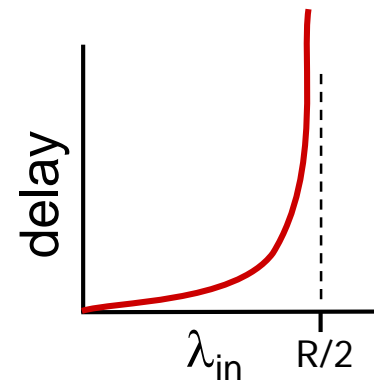
congestion:

- 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
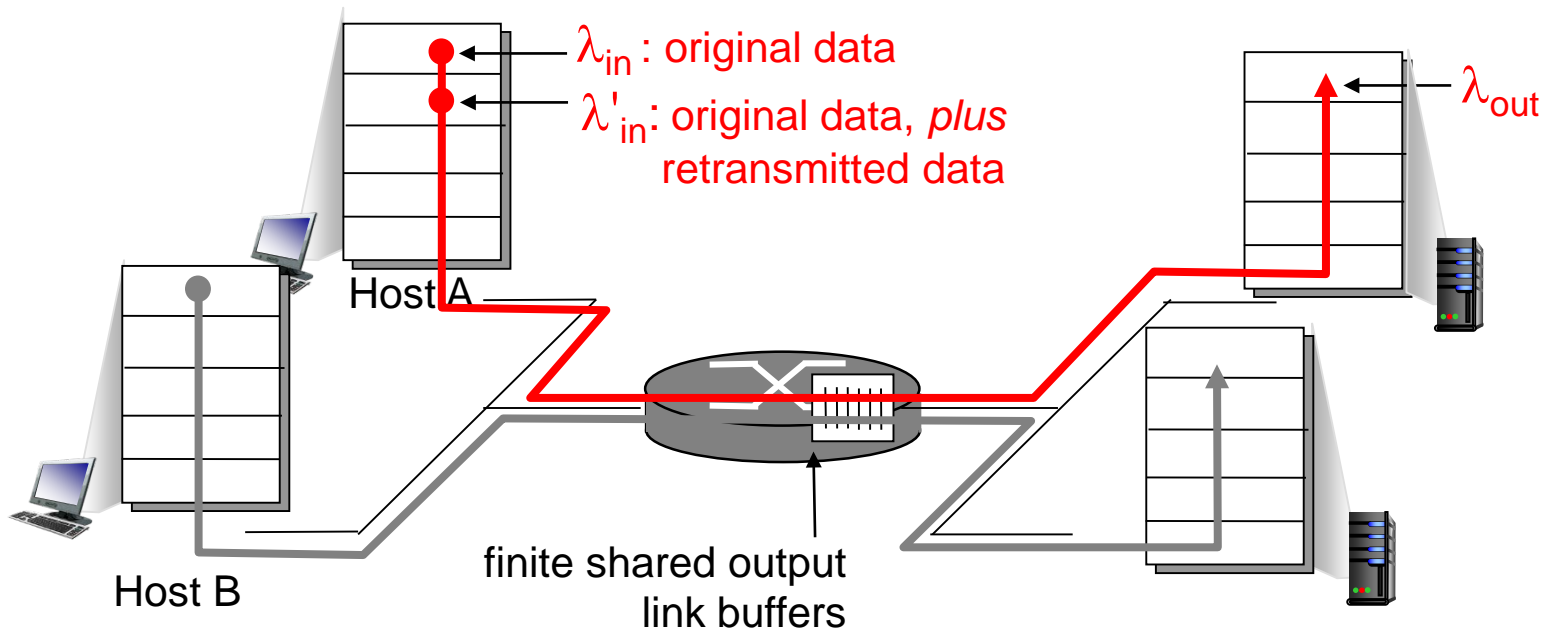- no retransmission

original data: λ_{in}

throughput: λ_{out}

Host A

unlimited shared output link buffers

Host B
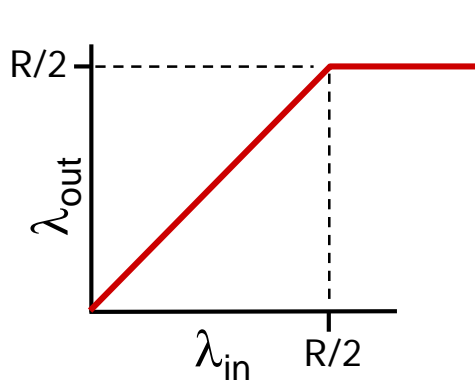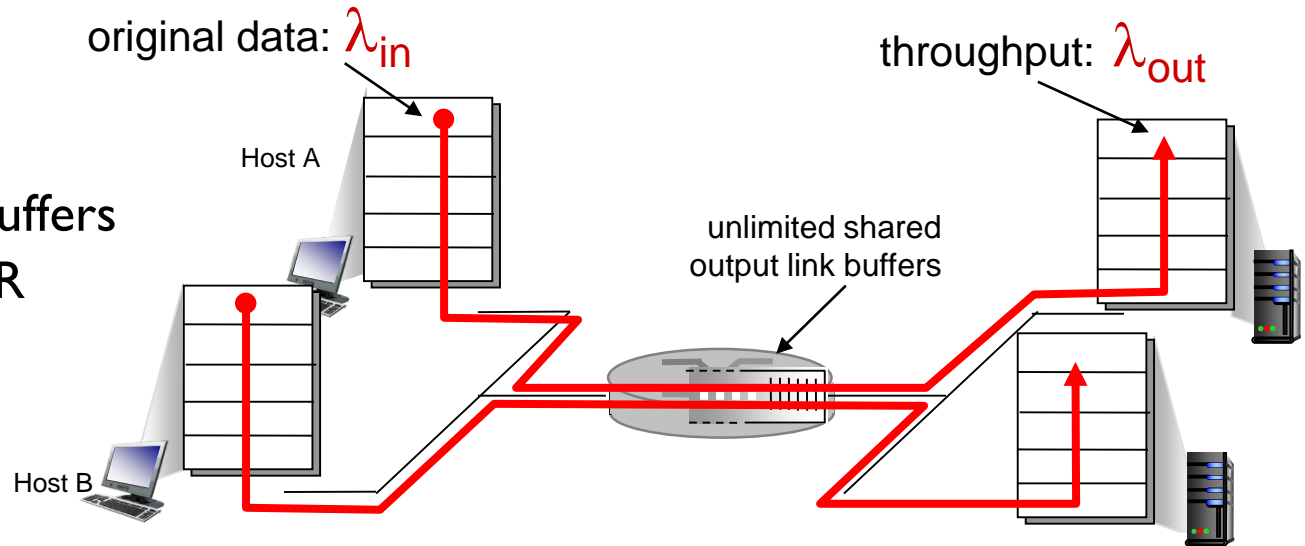
- maximum per-connection throughput: R/2

- large delays as arrival rate, λ_{in}, approaches capacity

Causes/costs of congestion: scenario 2

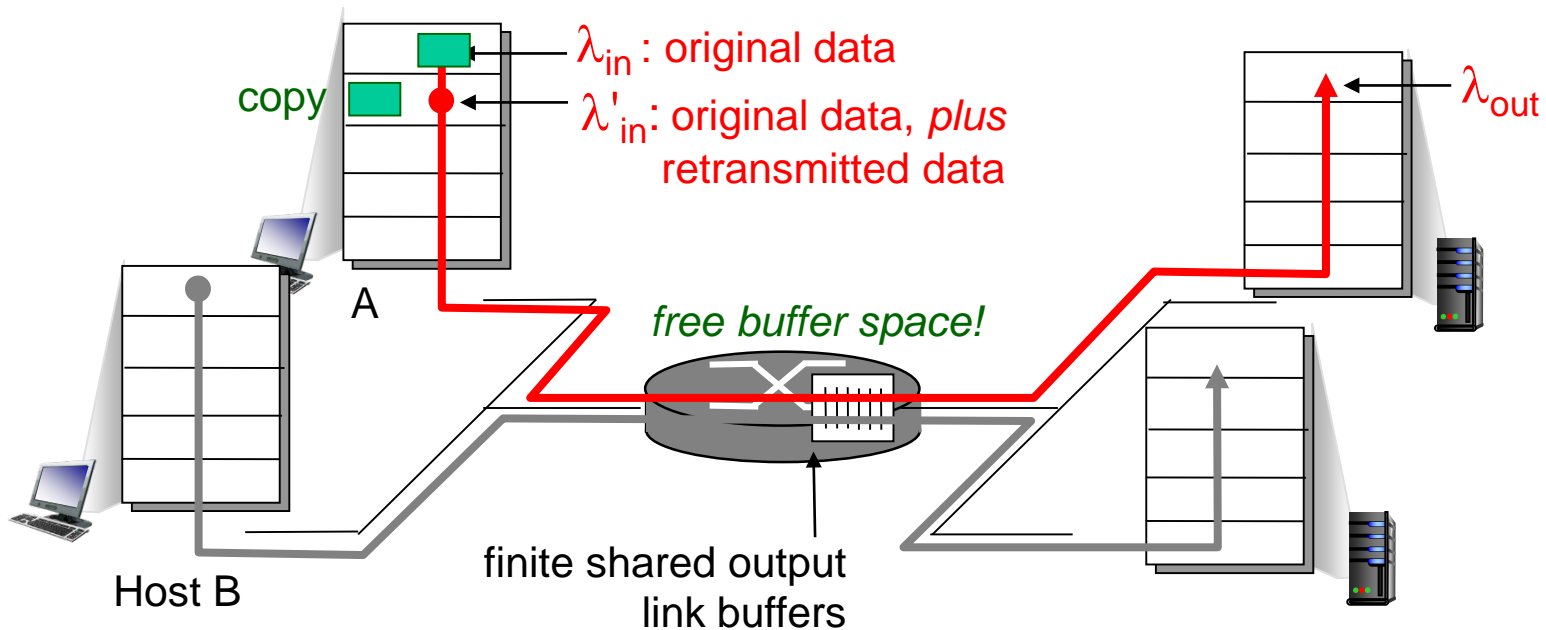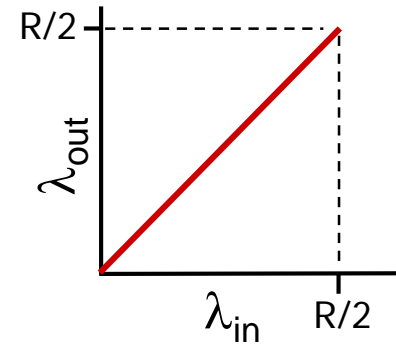- one router, *finite* buffers

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

λ_{in} : original data

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

λ_{out}

Host A

Host B

finite shared output link buffers

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}

copy

A

Host B

free buffer space!

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

λ_{in} : original data

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

λ_{out}
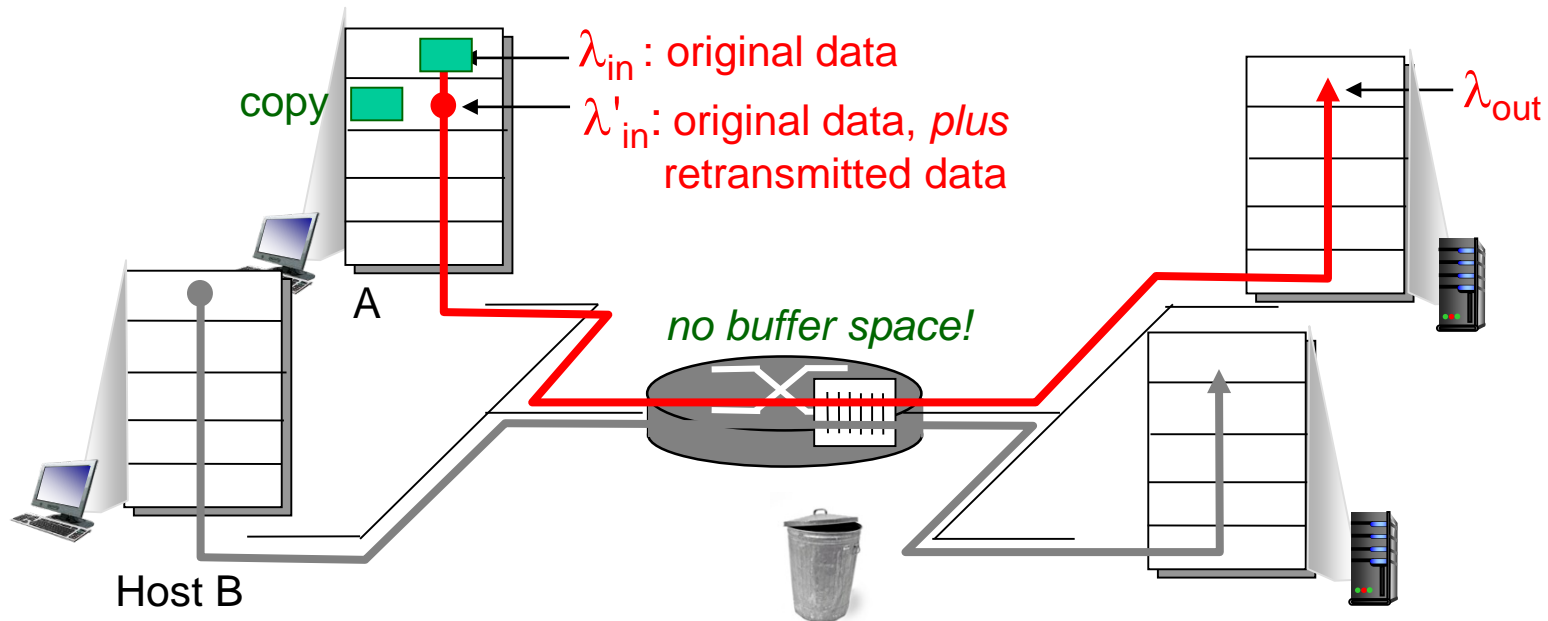
copy

no buffer space!

A

Host B

Causes/costs of congestion: scenario 2

Idealization: known loss

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

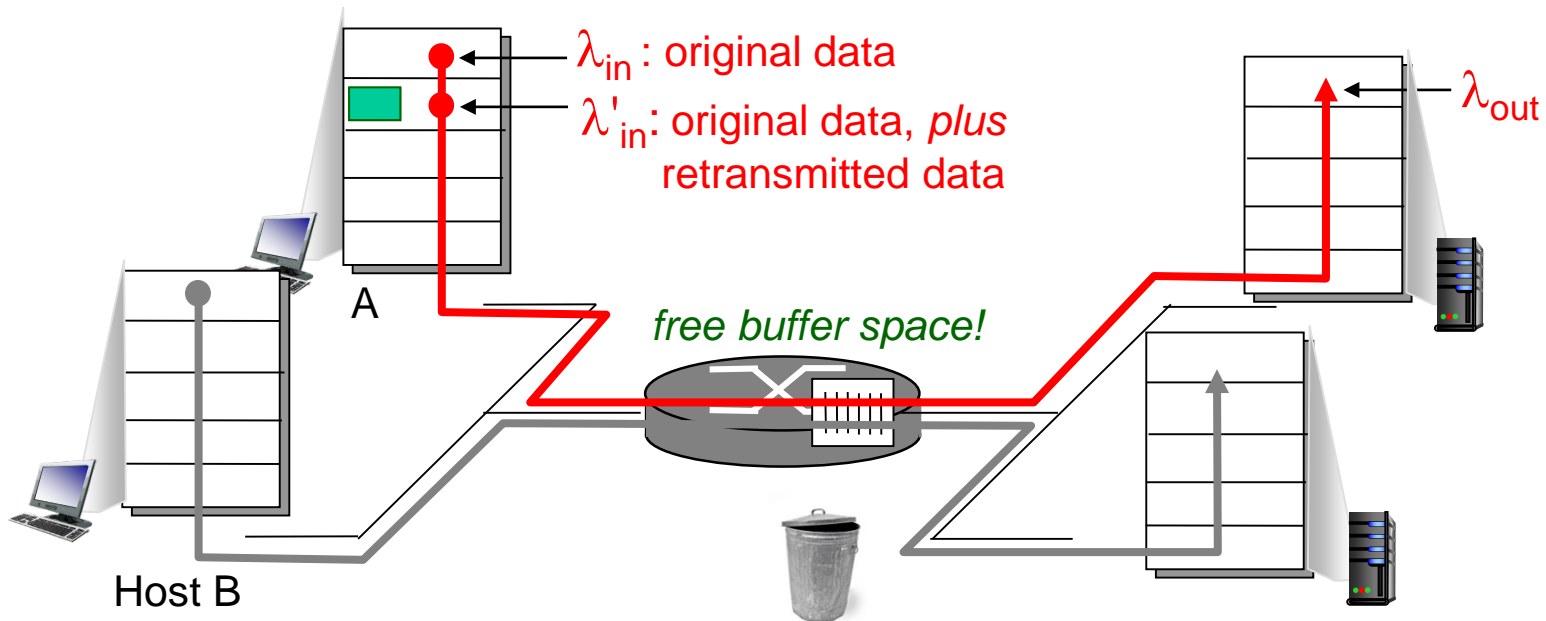- sender only resends if packet *known* to be lost
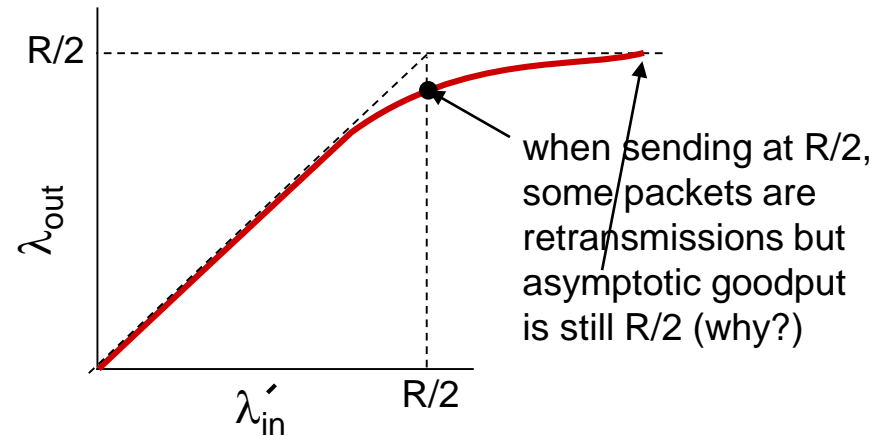

when sending at R/2, some packets are retransmissions but asymptotic goodput is still R/2 (why?)

λ_{in} : original data

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

λ_{out}

A

free buffer space!

Host B

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!

λ_{out} vs λ'_{in}

timeout

λ_{in}

λ'_{in}

A

λ_{out}

free buffer space!

Host B

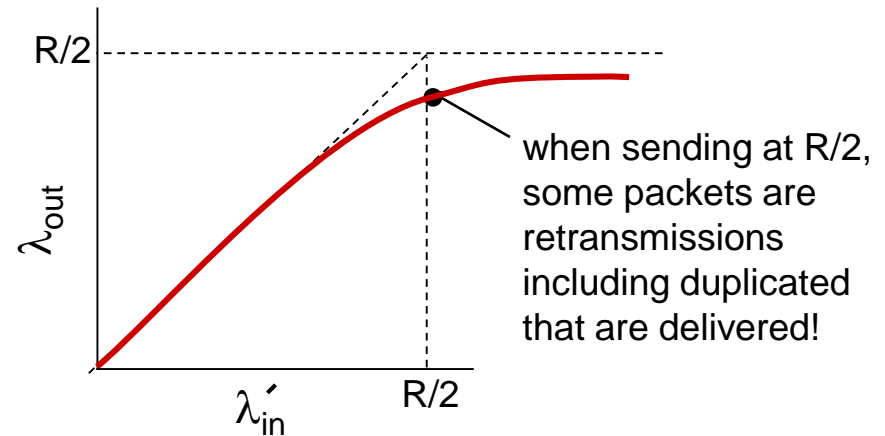# Causes/costs of congestion: scenario 2

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!
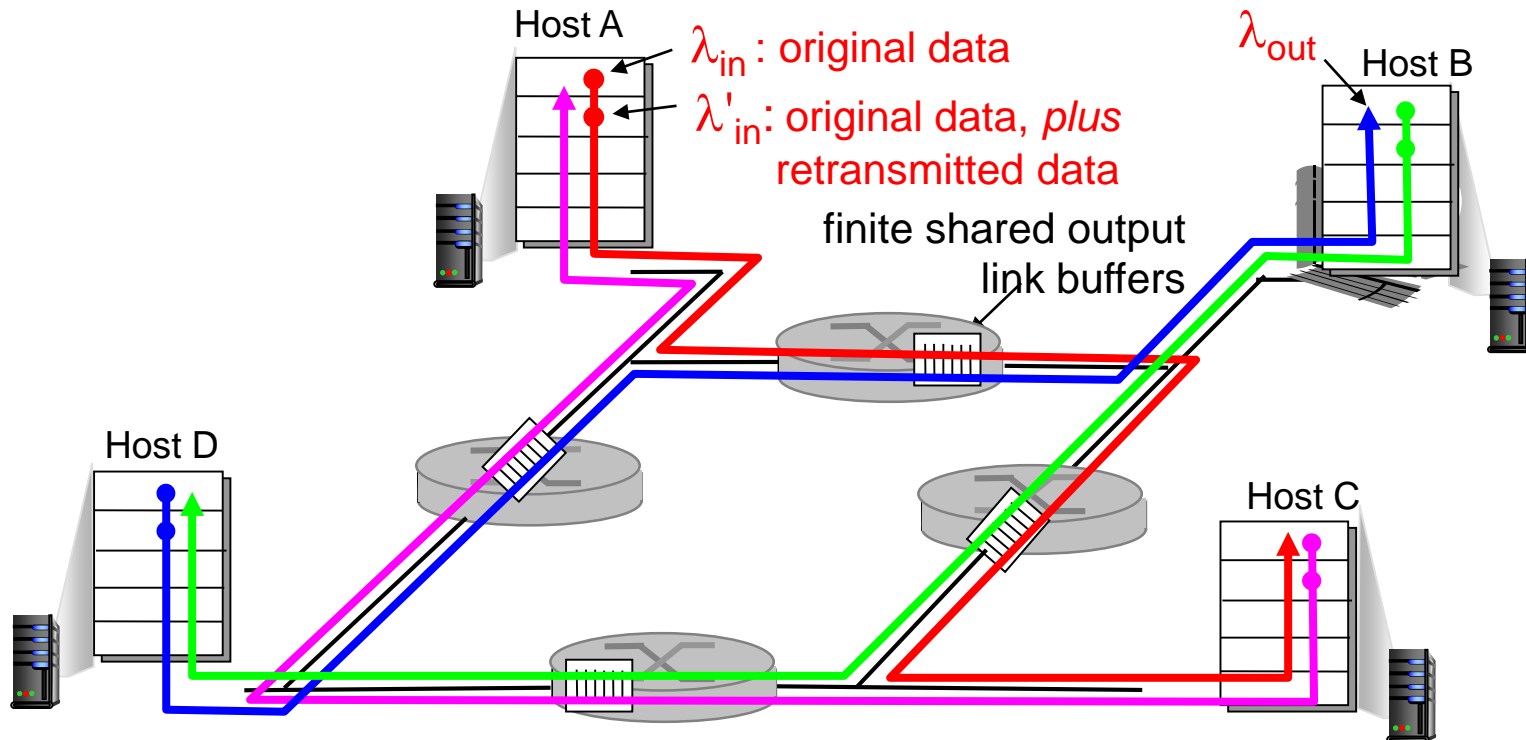
"costs" of congestion:

- 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

Host A

λ_{in} : original data

λ_{in}': original data, *plus* retransmitted data

λ_{out}

Host B

finite shared output link buffers

Host D

Host C

Causes/costs of congestion: scenario 3
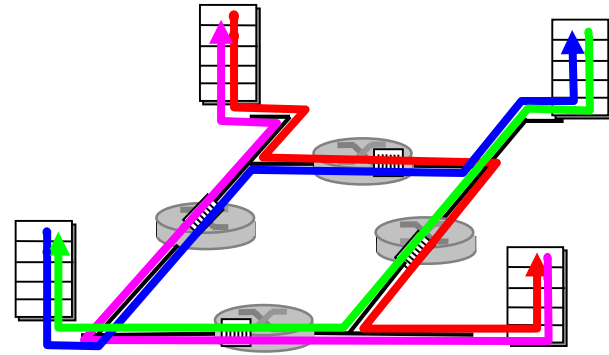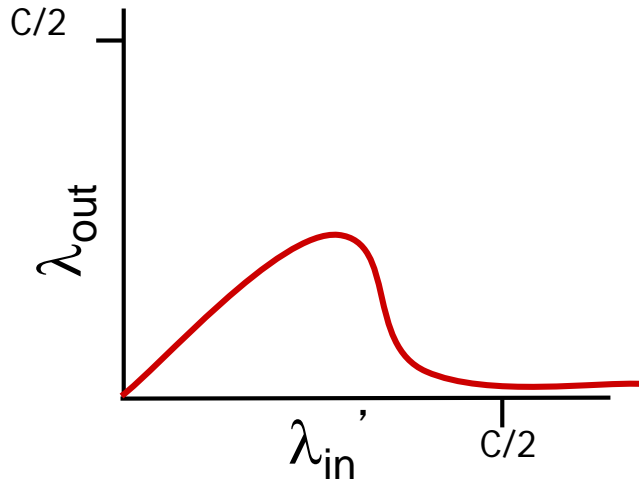
another "cost" of congestion:

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

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

additively increase window size ...
.... until loss occurs (then cut window in half)

`cwnd:` TCP sender congestion window size

time
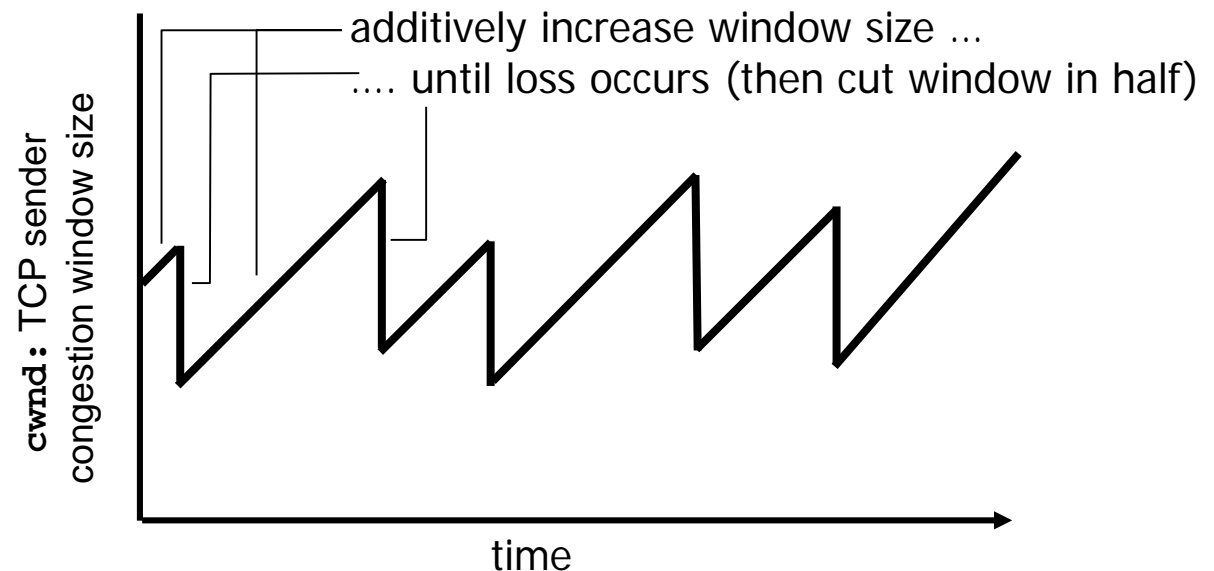
TCP Congestion Control: details

sender sequence number space

last byte ACKed

sent, not-yet ACKed ("in-flight")
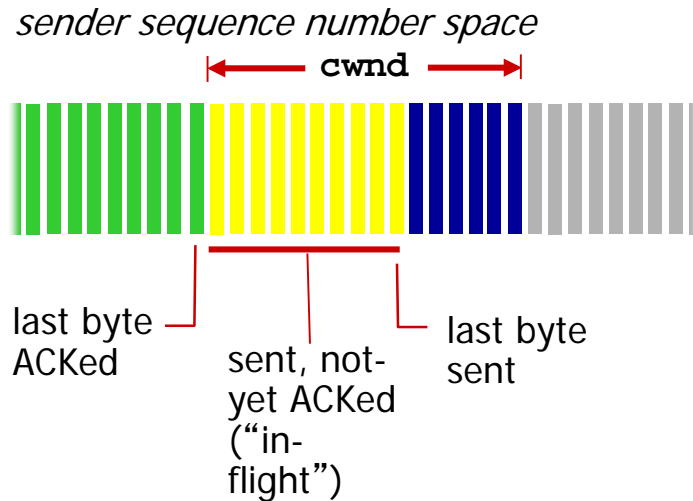
last byte sent

- **sender limits transmission:**

$$\text{LastByteSent-} \atop \text{LastByteAcked} \leq \text{cwnd}$$

- **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

Host A   Host B

RTT

one segment

two segments

four segments
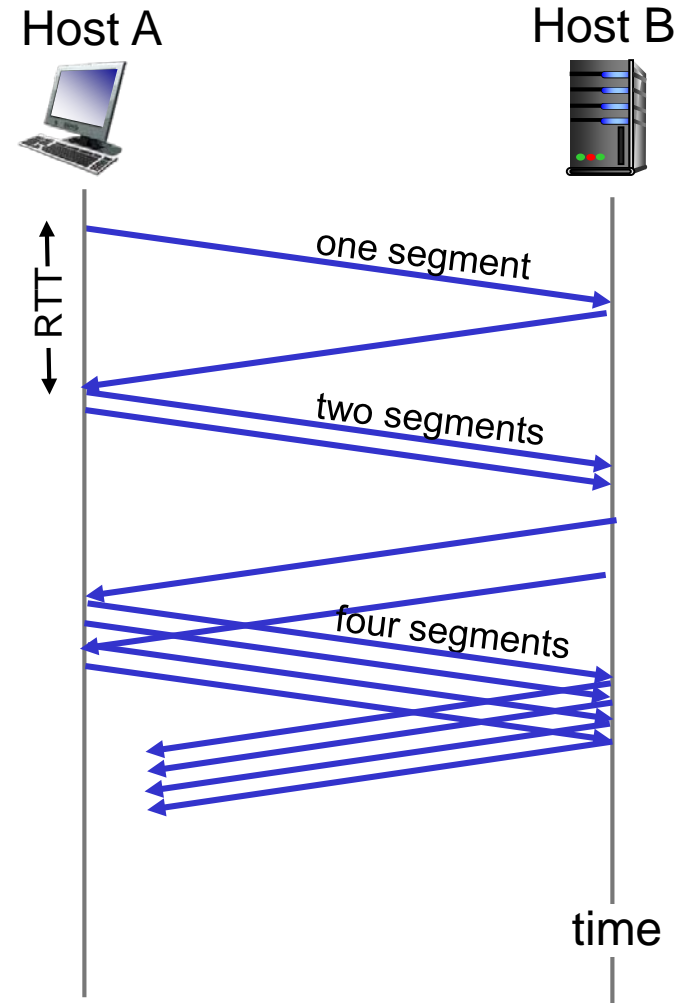
time

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)

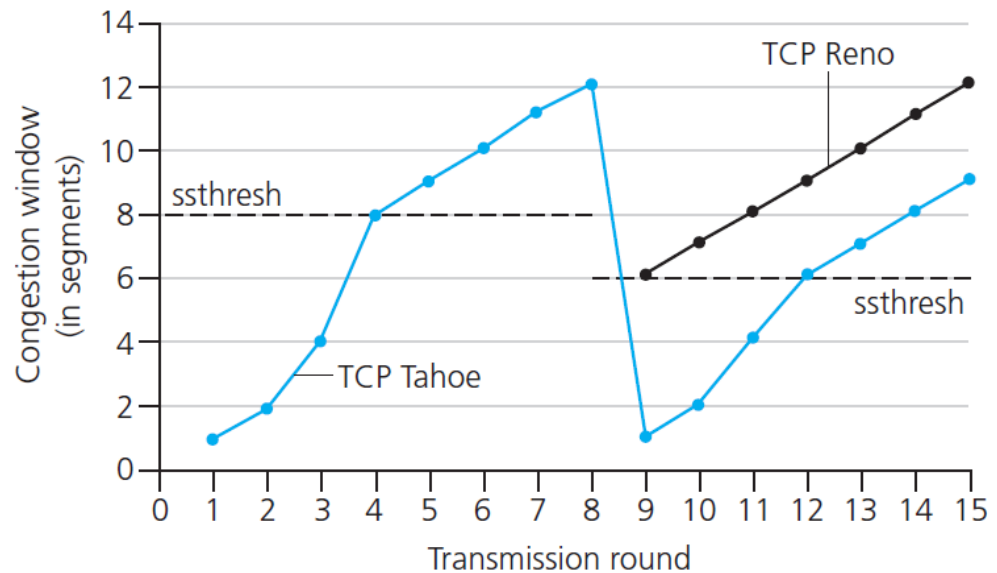# TCP: switching from slow start to CA

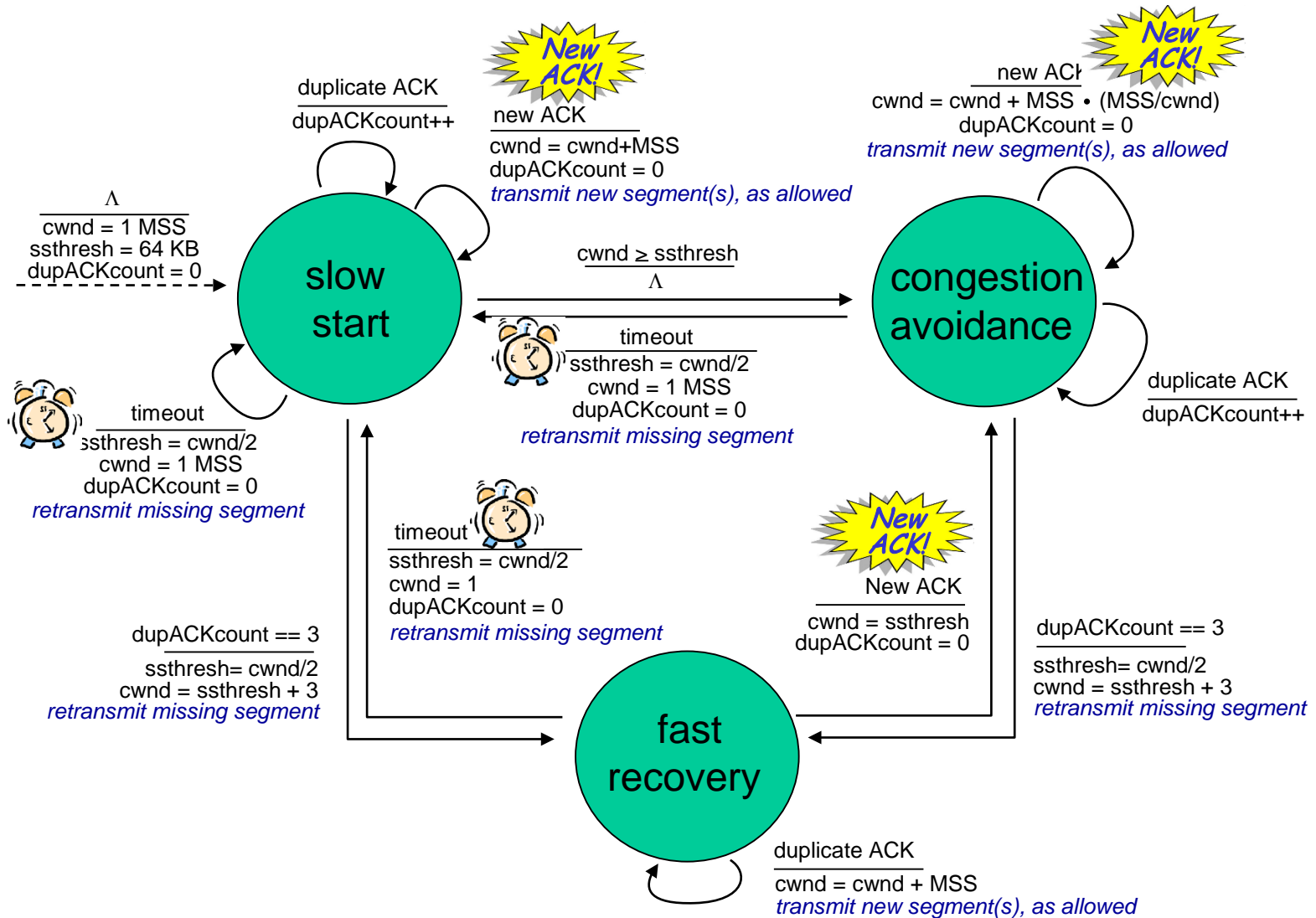Q: when should the exponential increase switch to linear?

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

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Summary: TCP Congestion Control

duplicate ACK
_____
dupACKcount++

New ACK!

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

New ACK!

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

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

slow start

cwnd ≥ ssthresh
_____
Λ

congestion avoidance

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!

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

fast recovery

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

$$\text{avg TCP thruput} = \frac{3}{4}\frac{W}{RTT} \text{ bytes/sec}$$
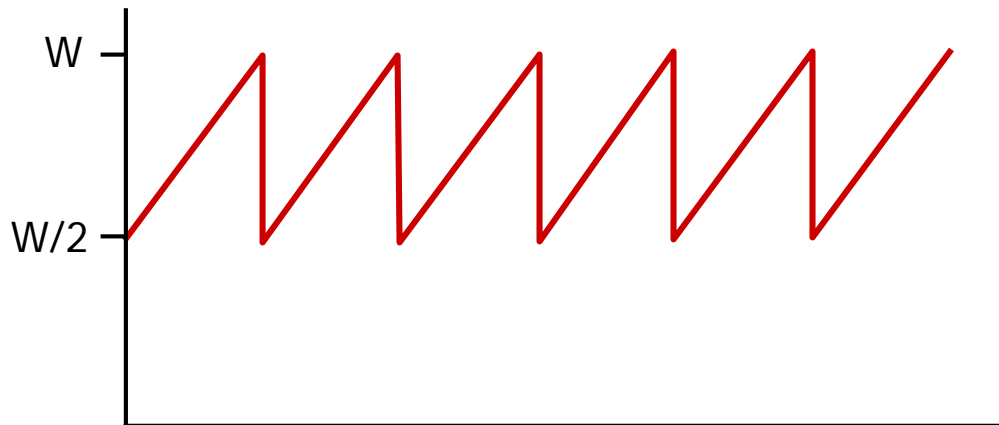
TCP Futures: TCP over "long, fat pipes"

- 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!*
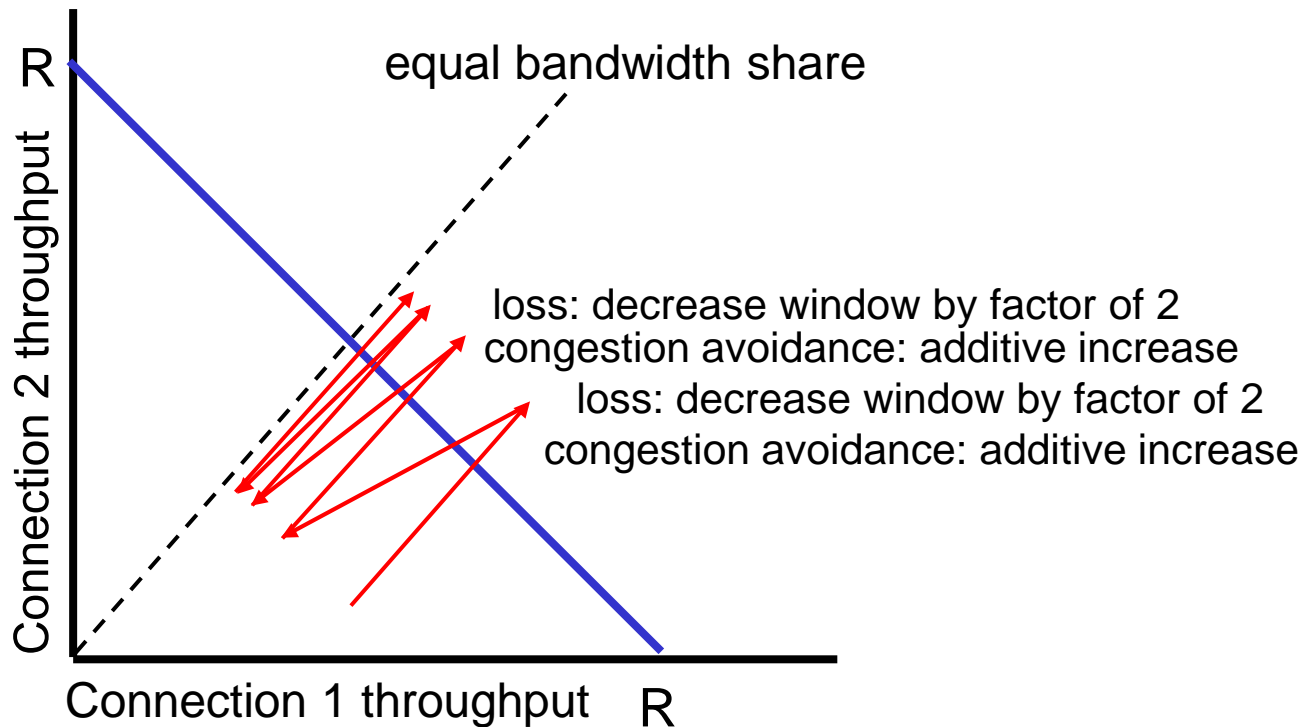- new versions of TCP for high-speed

TCP Fairness

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

TCP connection 1

TCP connection 2

bottleneck router capacity R
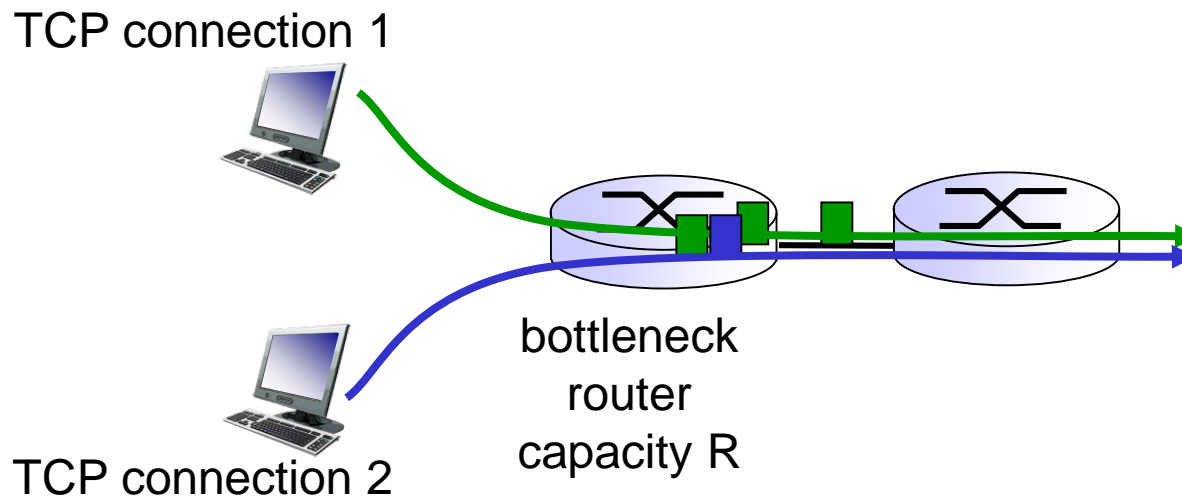
Why is TCP fair?

two competing sessions:

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

R — equal bandwidth share

Connection 2 throughput

loss: decrease window by factor of 2
congestion avoidance: additive increase
loss: decrease window by factor of 2
congestion avoidance: additive increase

Connection 1 throughput   R

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

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
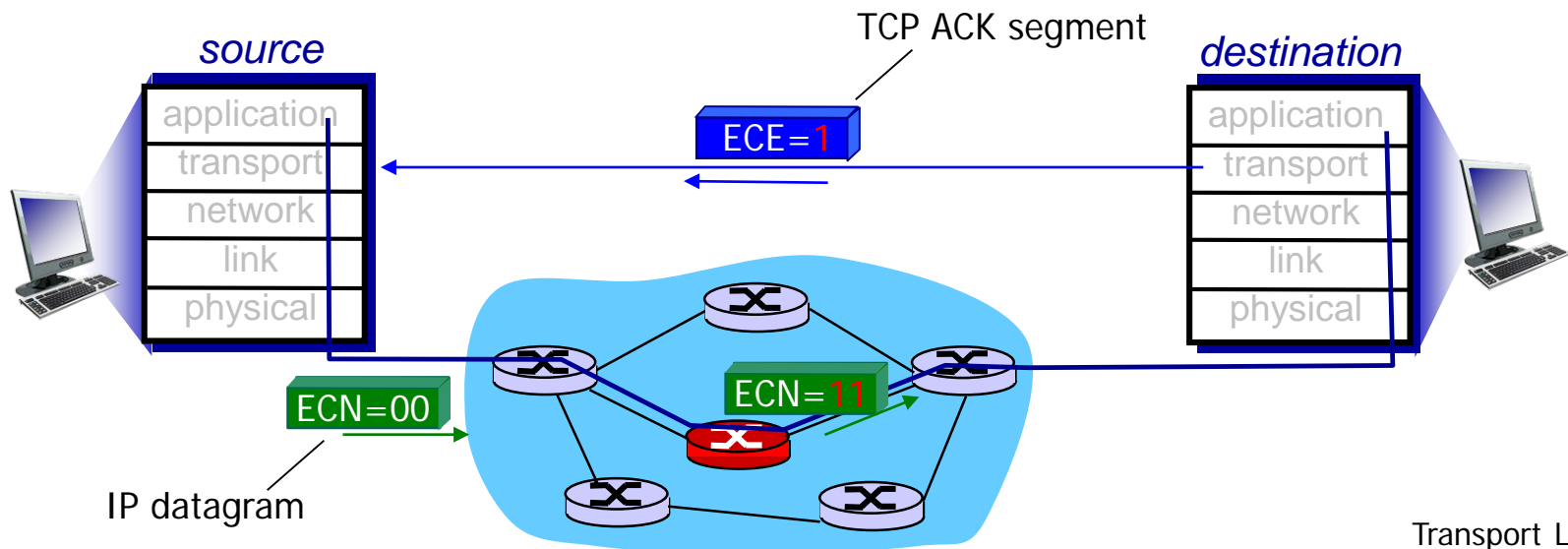
Chapter 3: summary

- 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"
- two network layer chapters:
  - data plane
  - control plane