# 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]

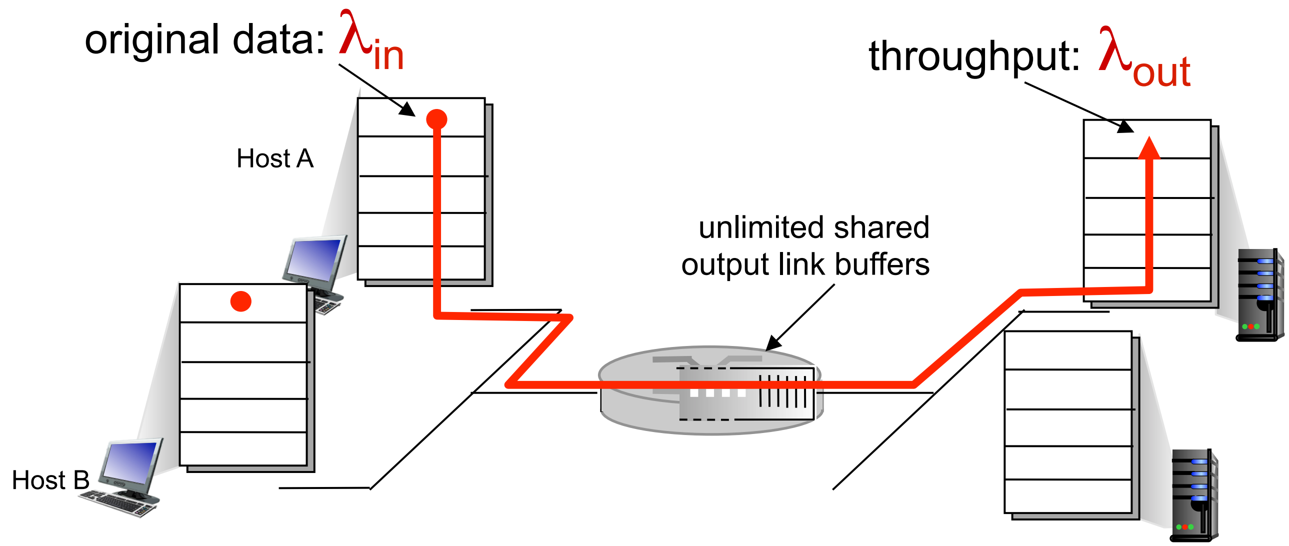
3.7 TCP congestion control

# 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)

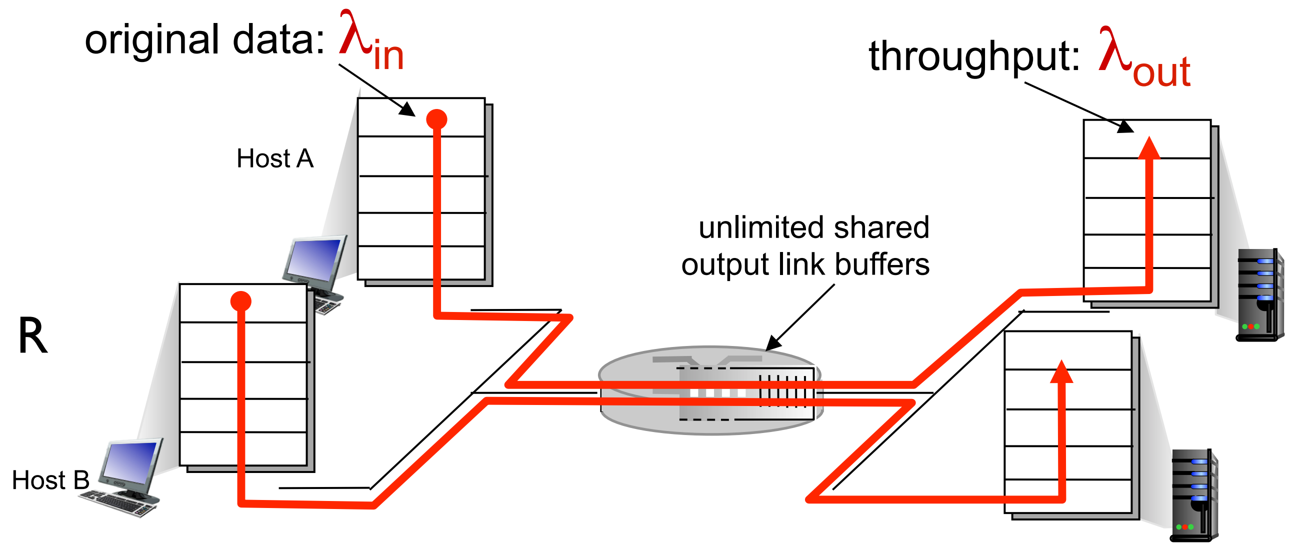
# Causes/costs of congestion: scenario 0

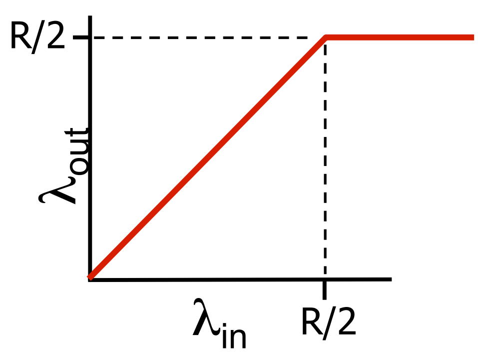


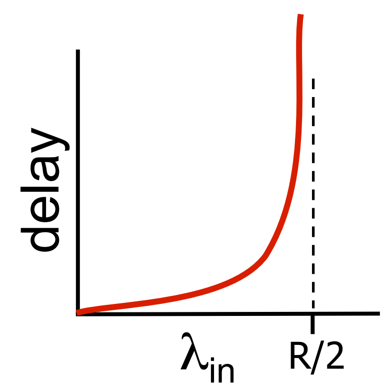

- maximum per-connection throughput: R/2
- large delays as arrival rate, $\lambda_{in}$, approaches capacity

# Causes/costs of congestion: scenario 1

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



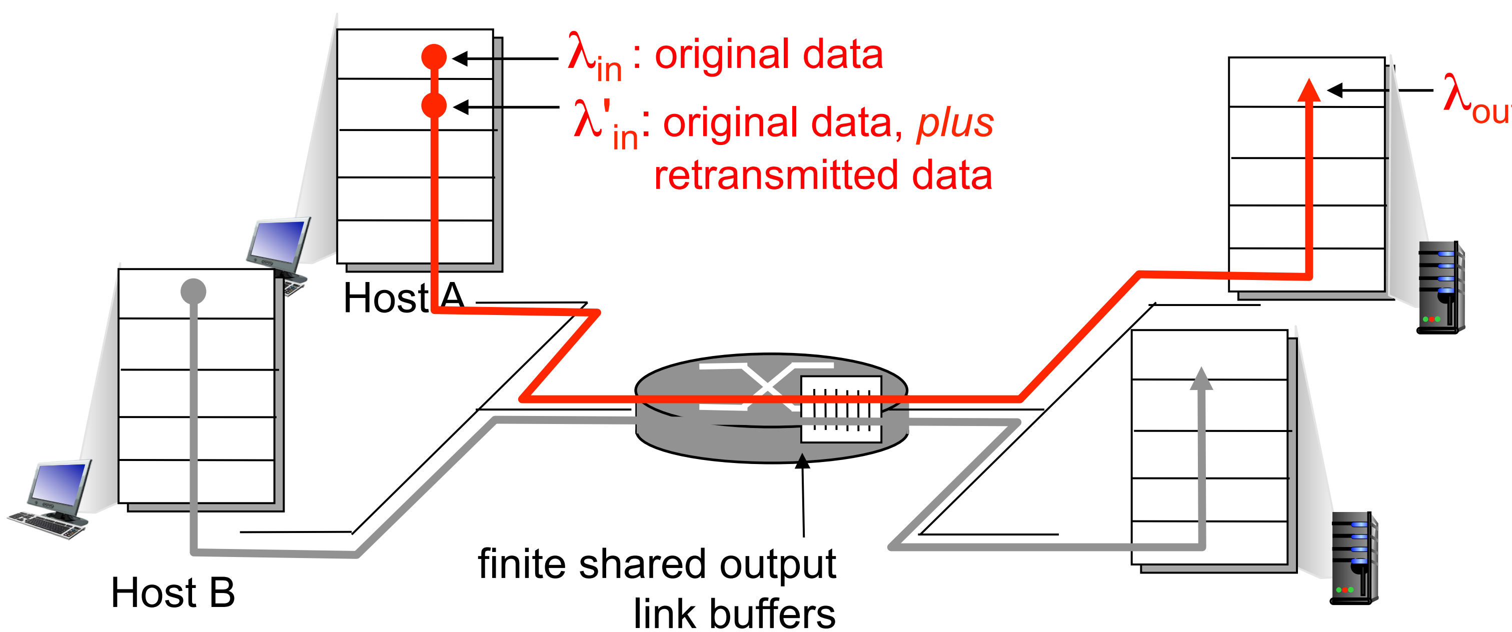




- maximum per-connection throughput: R/2
- large delays as arrival rate, $\lambda_{in}$, approaches capacity

# Causes/costs of congestion: scenario 2

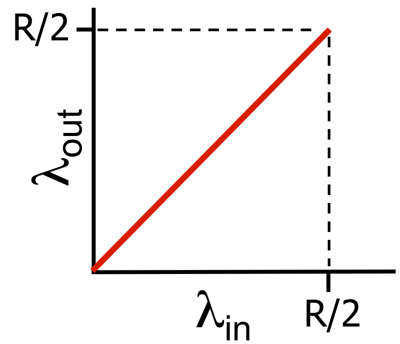
- one router, *finite* buffers
- sender retransmission of timed-out packet
  - application-layer input ($\lambda_{in}$), application-layer output ($\lambda_{out}$) transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$

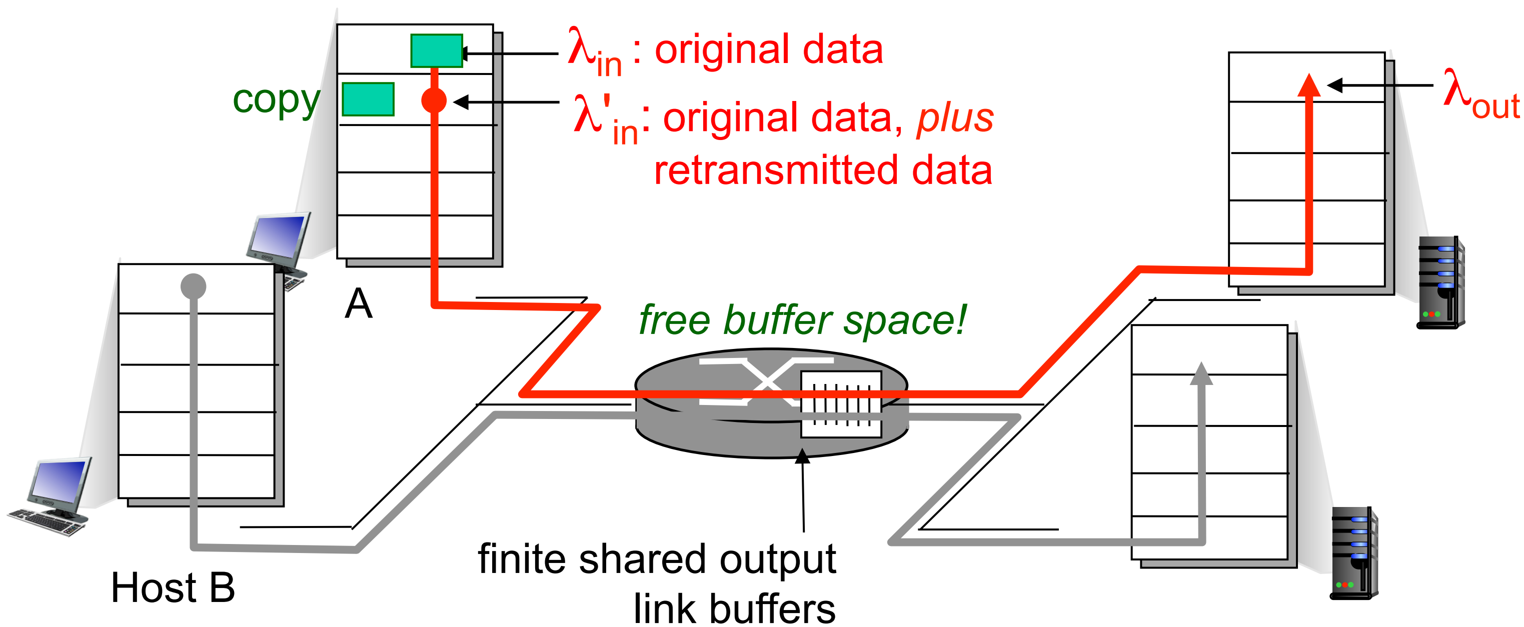


# Scenario 2a: ideal case

idealization: perfect knowledge

- sender sends only when router buffers available

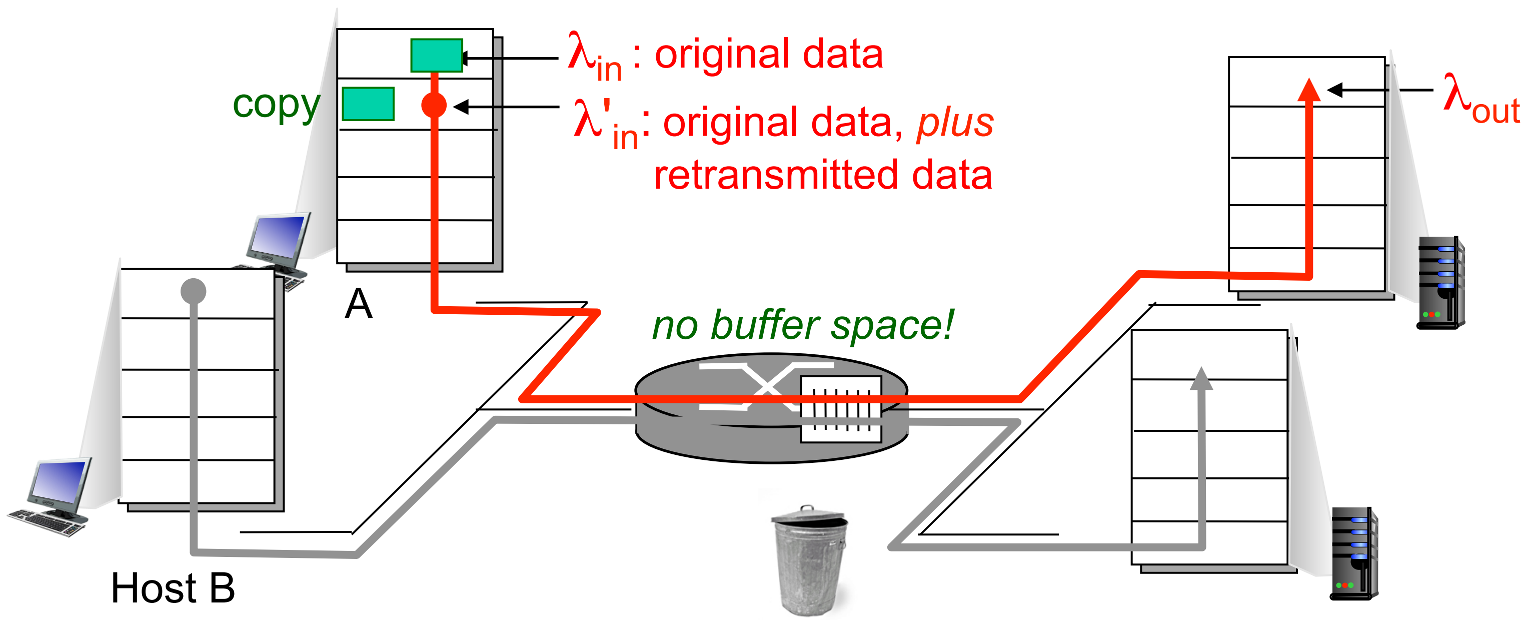


# Scenario 2b: known loss

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

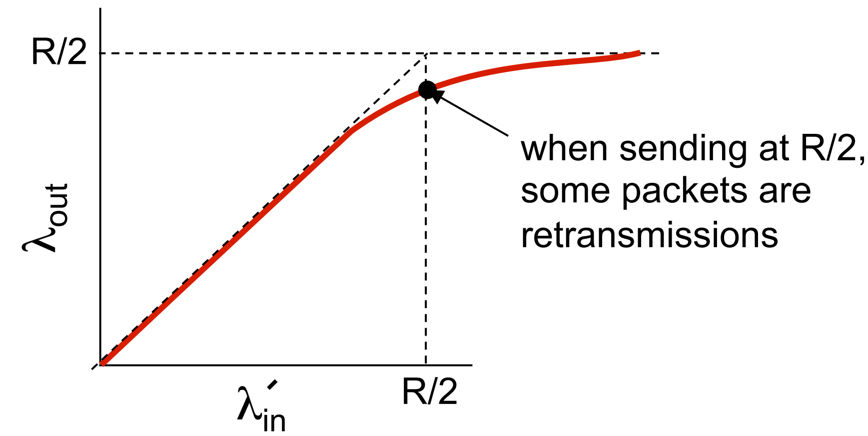
- sender only resends if packet *known* to be lost

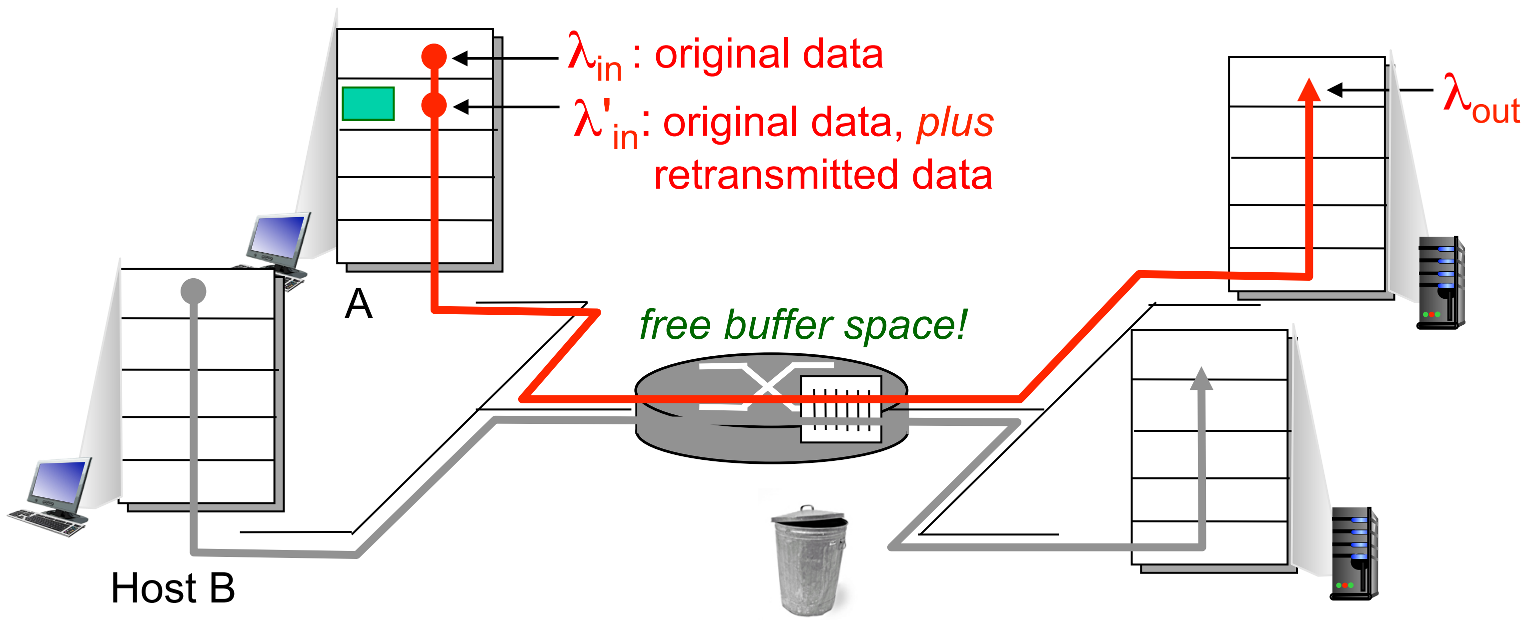


# Scenario 2b: known loss

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

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

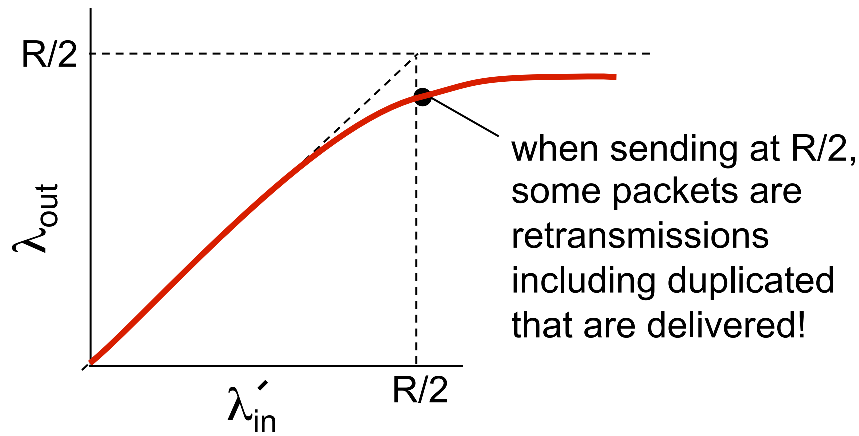


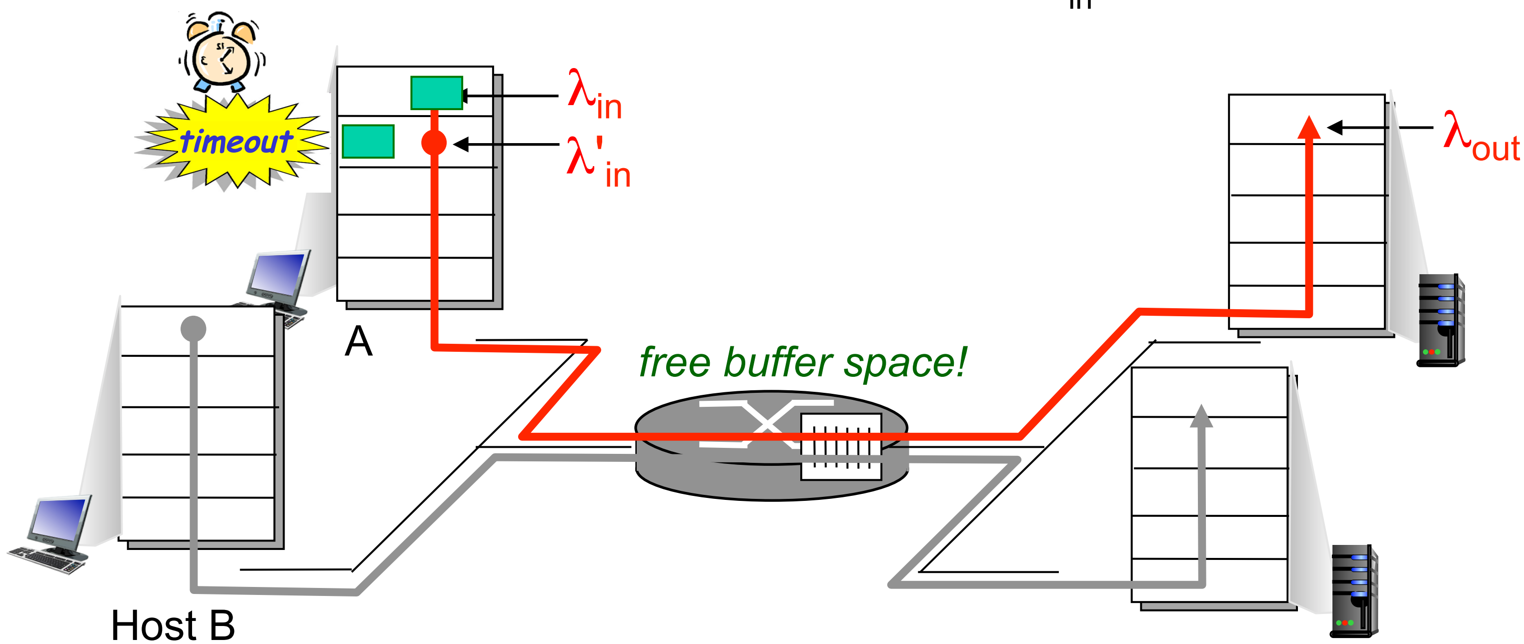



# Scenario 2c: duplicates

*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

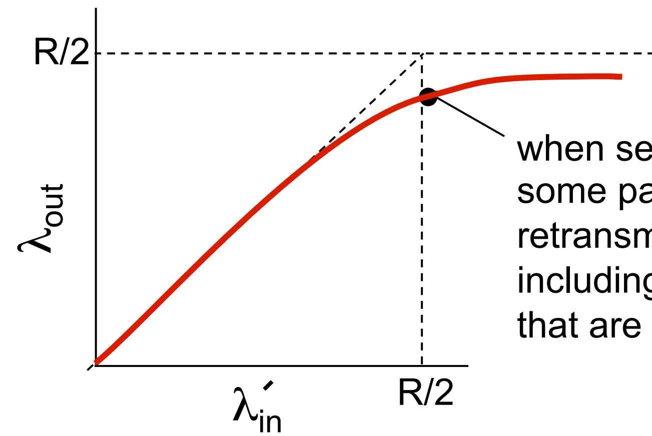


# Scenario 2c: duplicates

*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




"costs" of congestion:

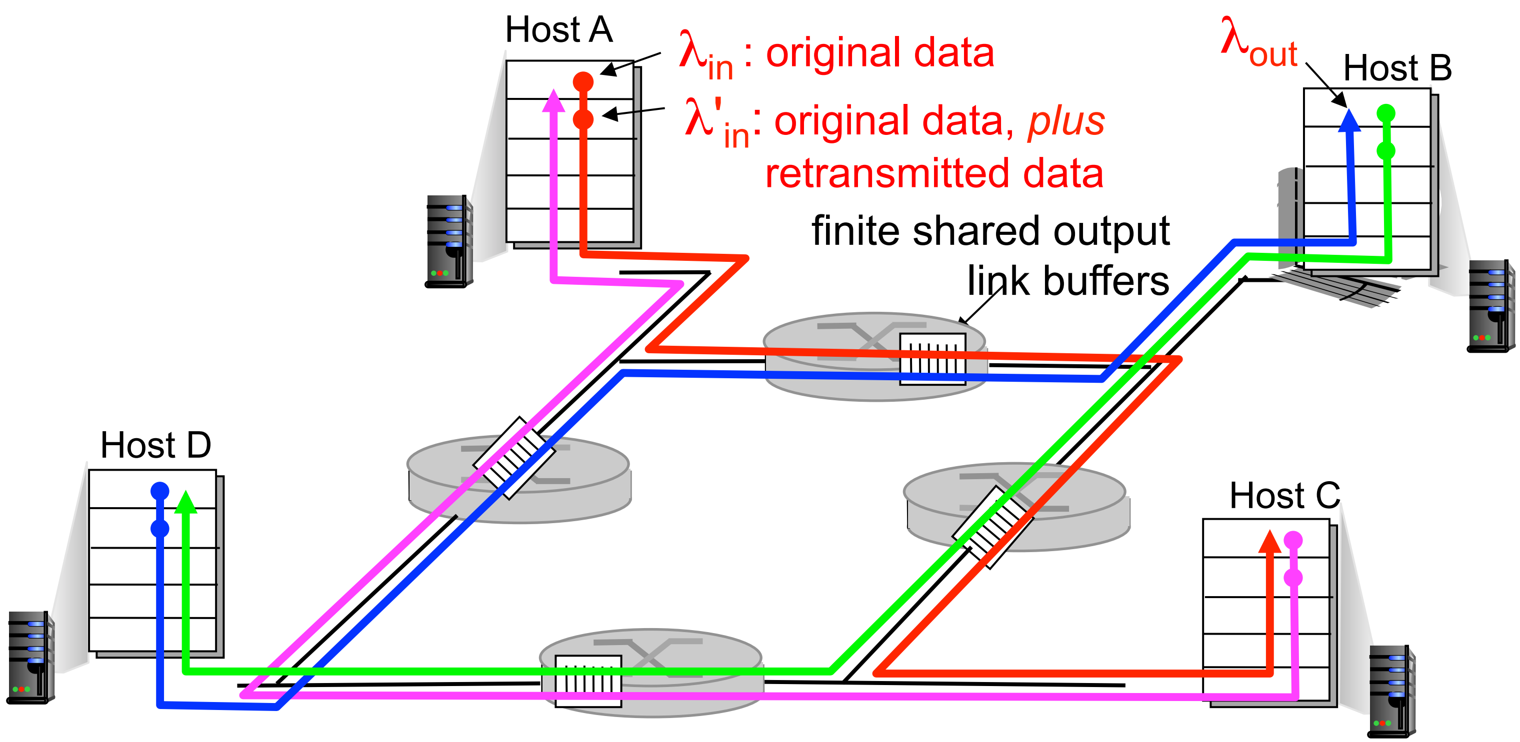
- more work (retrans) for given "goodput"
- unneeded retransmissions: link carries multiple copies of pkt
  - decreasing goodput

# Scenario 3: Multi-hop

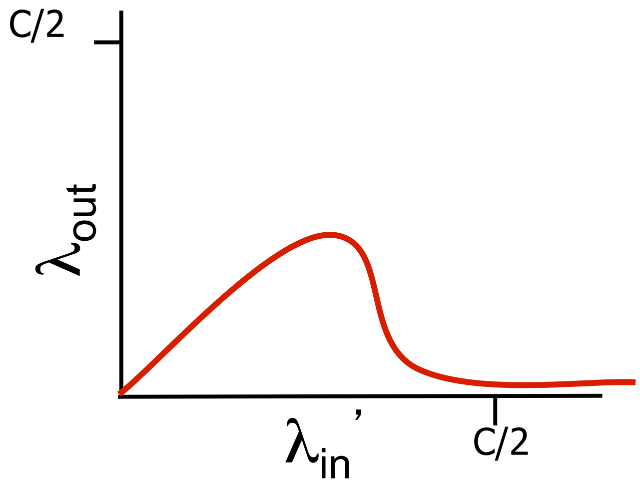
- four senders
- multihop paths
- timeout/retransmit

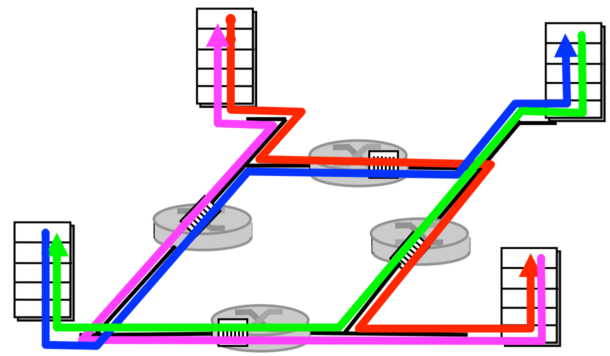
Q: what happens as $\lambda_{in}$ and $\lambda_{in}'$ increase ?

A: as red $\lambda_{in}'$ increases, all arriving blue pkts at upper queue are dropped, blue throughput $\rightarrow$ 0



# Scenario 3 cont'd



another "cost" of congestion:

- when packet dropped, any "upstream transmission capacity used for that packet was wasted!
- at high loss regime, all bandwidth is wasted on retransmissions

# 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

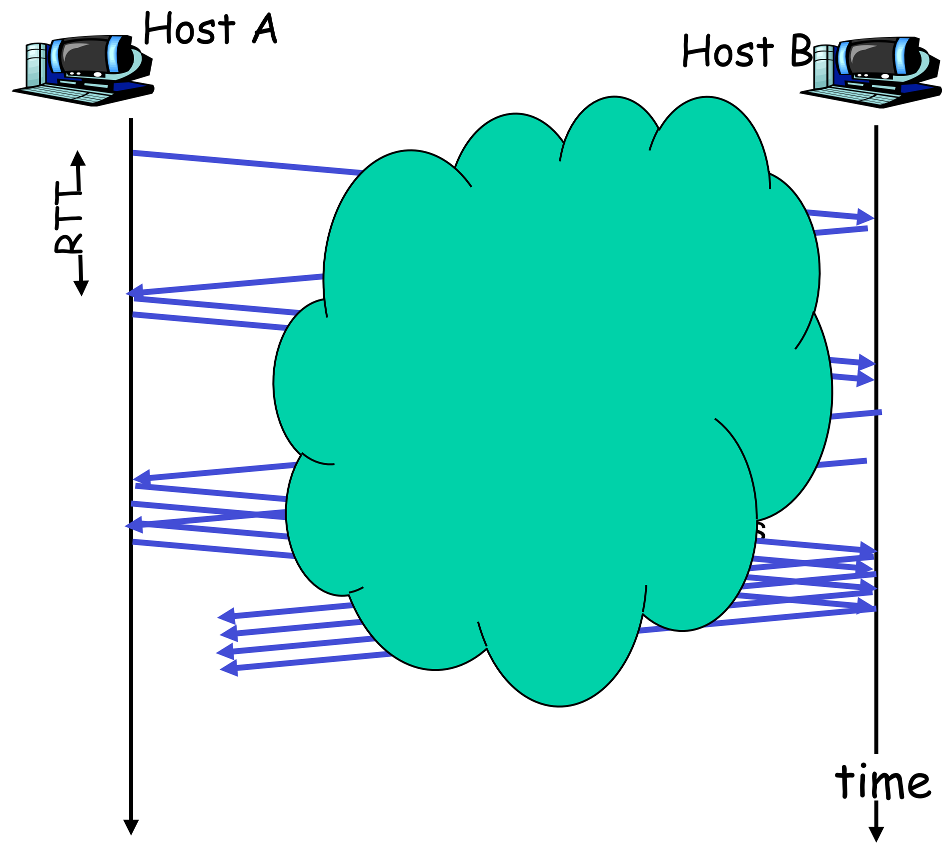
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

# e2e congestion control from the source's point of view

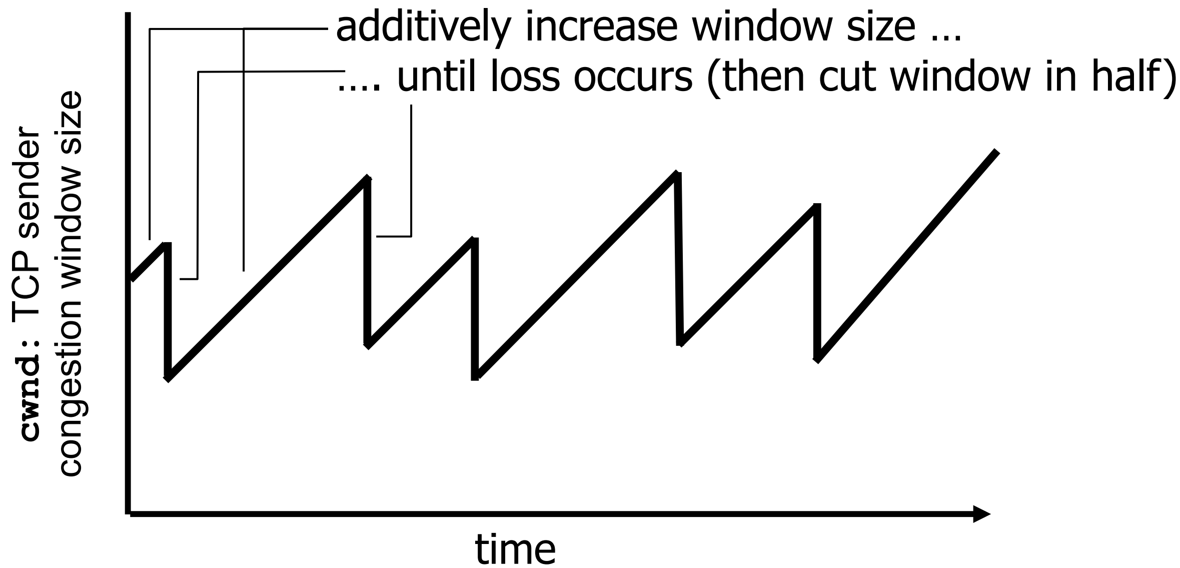
- Sender would like to know know at what rate to send.
- Probe the network and discover the available bandwidth
- Use implicit information(ACKs or timeouts)
  - to infer congestion level and adjust the window



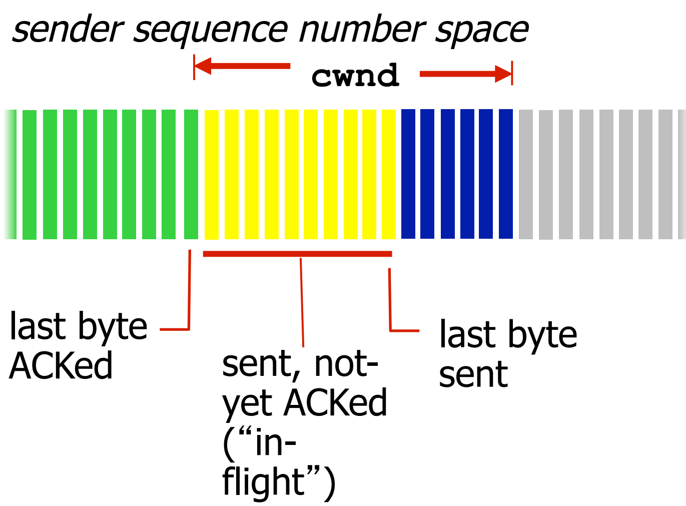
# TCP congestion control: additive increase multiplicative decrease (AIMD)

- *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



# TCP Congestion Control: details




- sender limits transmission:

```
LastByteSent-
  LastByteAcked   ≤  min{cwnd, rwnd}
```

- The sender adjusts **`cwnd`** dynamically, as a function of perceived network congestion

*TCP sending rate*

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

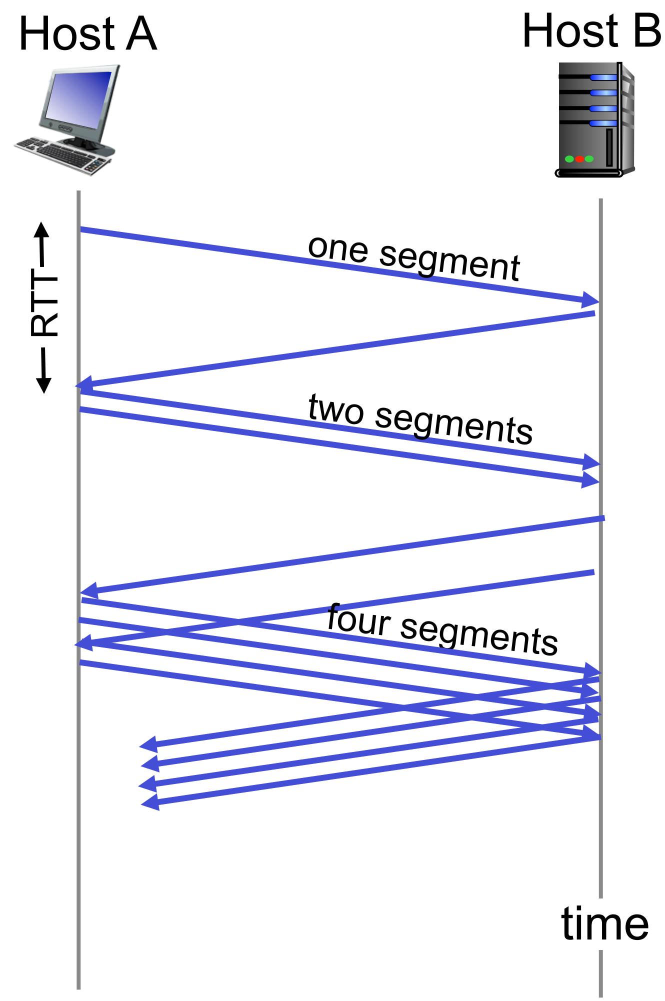
$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

# TCP Slow Start (SS)

- Rationale: initial rate is slow but ramps up exponentially fast

- SS: 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

- Note:
  - **cwnd**:=**cwnd**+MSS per ACK
  - **cwnd** doubles per RTT



# 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 (upon timeout or 3 dup acks)
- TCP Vegas: no loss yet but increased RTT→ queues building up: lower rate linearly

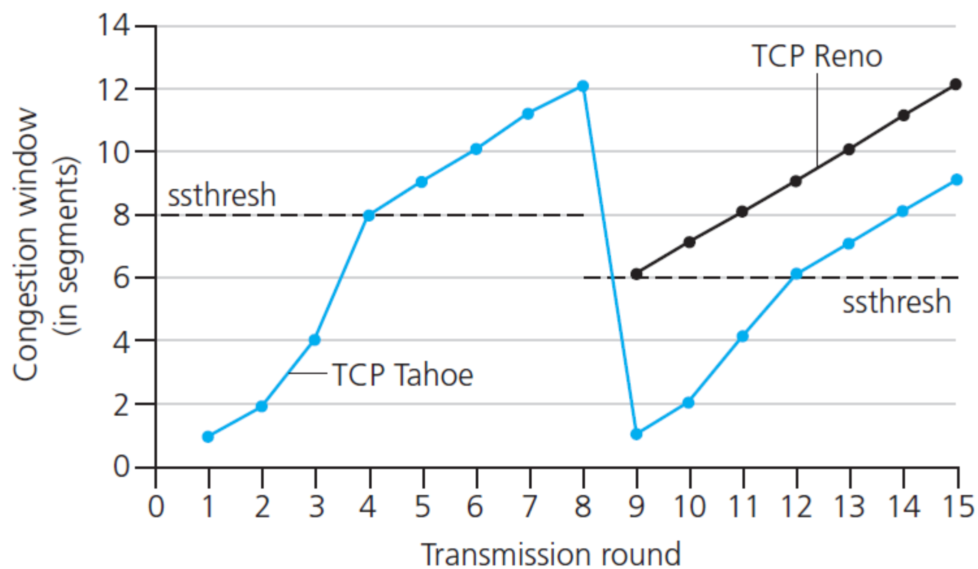
# TCP: switching from SS to CA

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

**A:** when **cwnd** gets to 1/2 of its value before timeout [why?].

- **cwnd** is 1/2 of its value before loss → congestion is around the corner.
- do not double - act more conservatively.

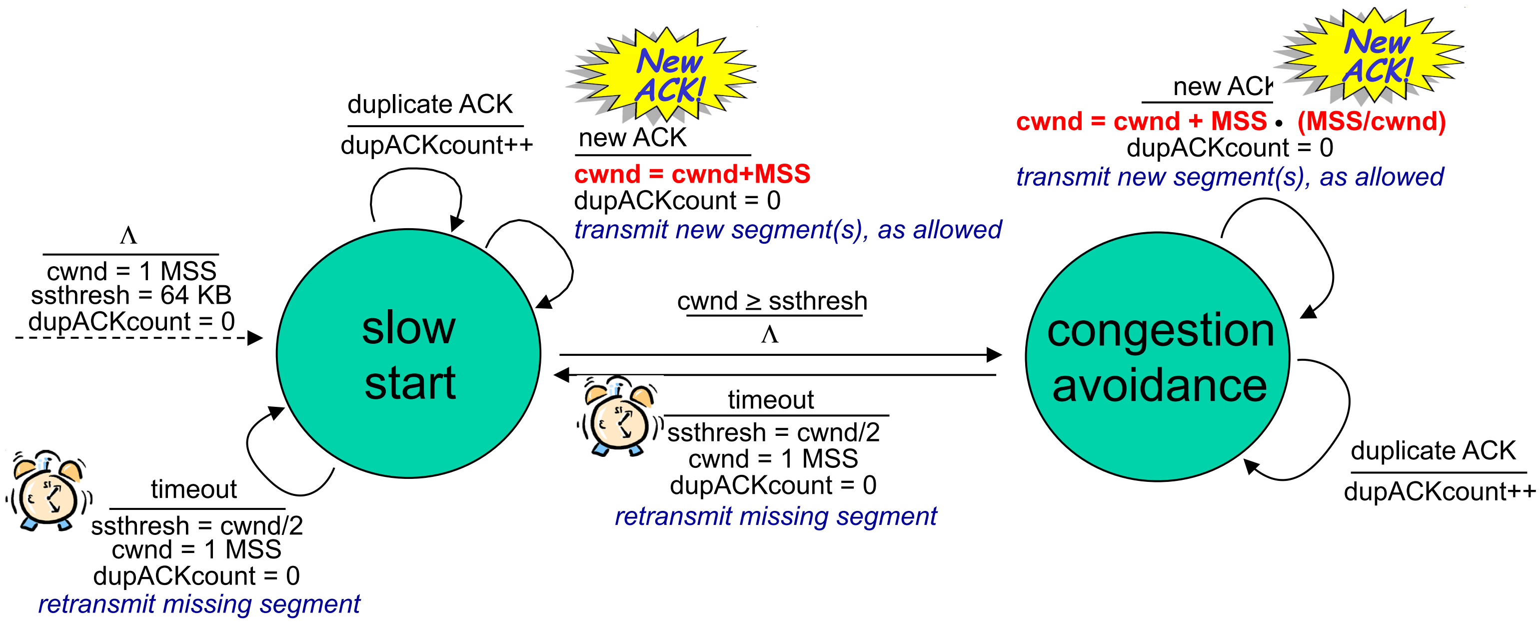
BTW: cwnd:=1, if loss during slow start




## Implementation:

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

# Summary: TCP Congestion Control



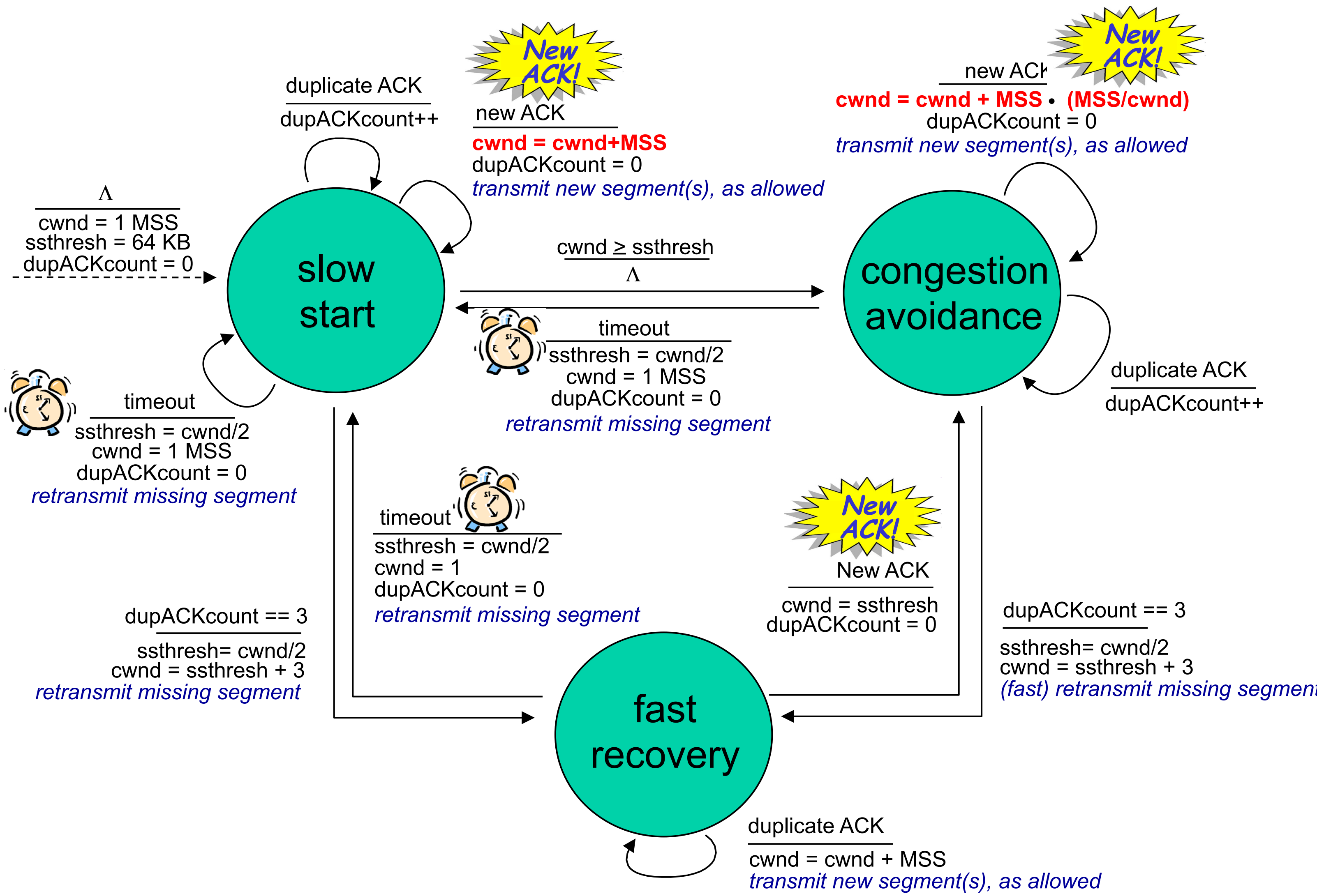

Notes:

- In one RTT: there are cwnd B i.e. cwnd/MSS segments sent and ACKs received
- Additive increase: 1MSS B per RTT, i.e. MSS/(cwnd/MSS) B per ACK
- Exponential increase: cwnd B per RTT, i.e. MSS B per ACK

# Fast Recovery

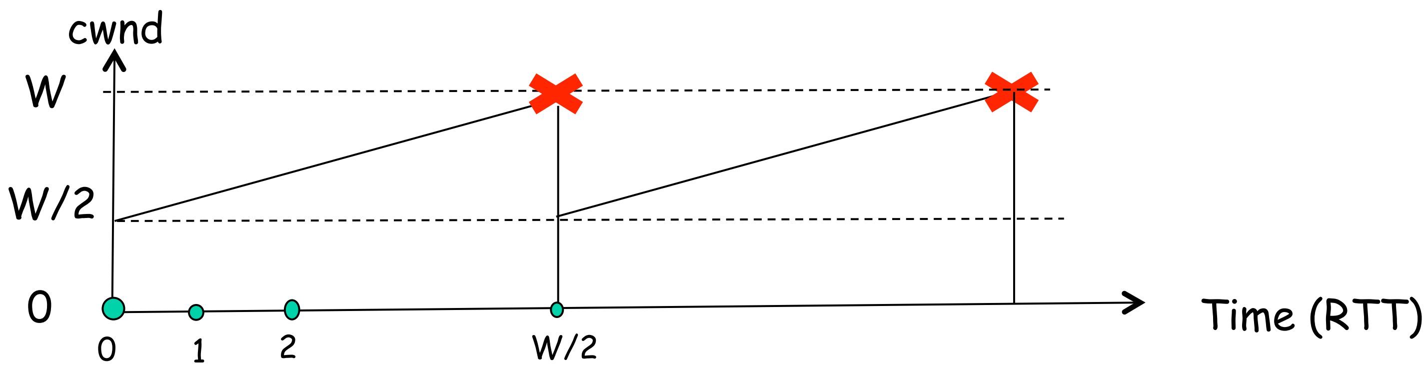
- Main idea:
  - Infer successful  transmission even from dup (not only from new) ACK
- Rationale:
  - Every ACK (even dup) is triggered by some new packet that made it (even out of order)
  - Keep the number of packets in the pipeline  constant
- Action
  - Fast Retransmit upon 3 dup ACK
  - Inflate window: Increase by 1 MSS per every duplicate ACK received
  - Deflate window: when new (non-duplicate) ACK received
- Fast recovery recommended, not required.
  - Not implemented in TCP Tahoe
  - Implemented in TCP Reno, …
  - http://en.wikipedia.org/wiki/TCP_congestion_avoidance_algorithm

# Summary: TCP Congestion Control



# Average TCP throughput

- Simplified analysis (on the board, Problem 45):
  - ignore slow start, assume always data to send
  - consider the period from W/2 to W (when loss occurs)




  - avg. window size (# in-flight segments) is ¾ W
  - avg. throughput is 3/4W per RTT: avg TCP thruput = $\frac{3}{4} \frac{W}{RTT}$ bytes/sec
  - 1 segment lost per W
  - avg. throughput in terms of loss rate L (in bytes/sec):

$$\boxed{\frac{1.22 \cdot MSS}{RTT\sqrt{L}}}$$

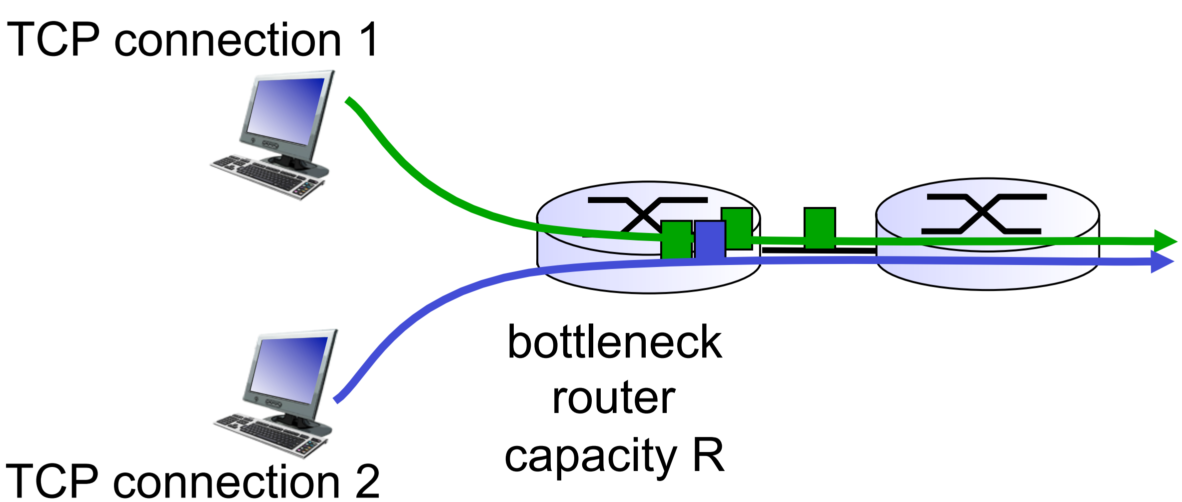
# TCP Future: TCP over "long, fat pipes"

- example: MSS=1500B, RTT=100ms
  - want 10 Gbps throughput
  - requires W = 83,333 in-flight segments and a loss rate of $L = 2 \cdot 10^{-10}$ *– a very small loss rate!*
- new versions of TCP needed & developed for "high bandwidth-delay product" networks

# TCP Fairness

*TCP is "nice" (by backing off) and "fair".*

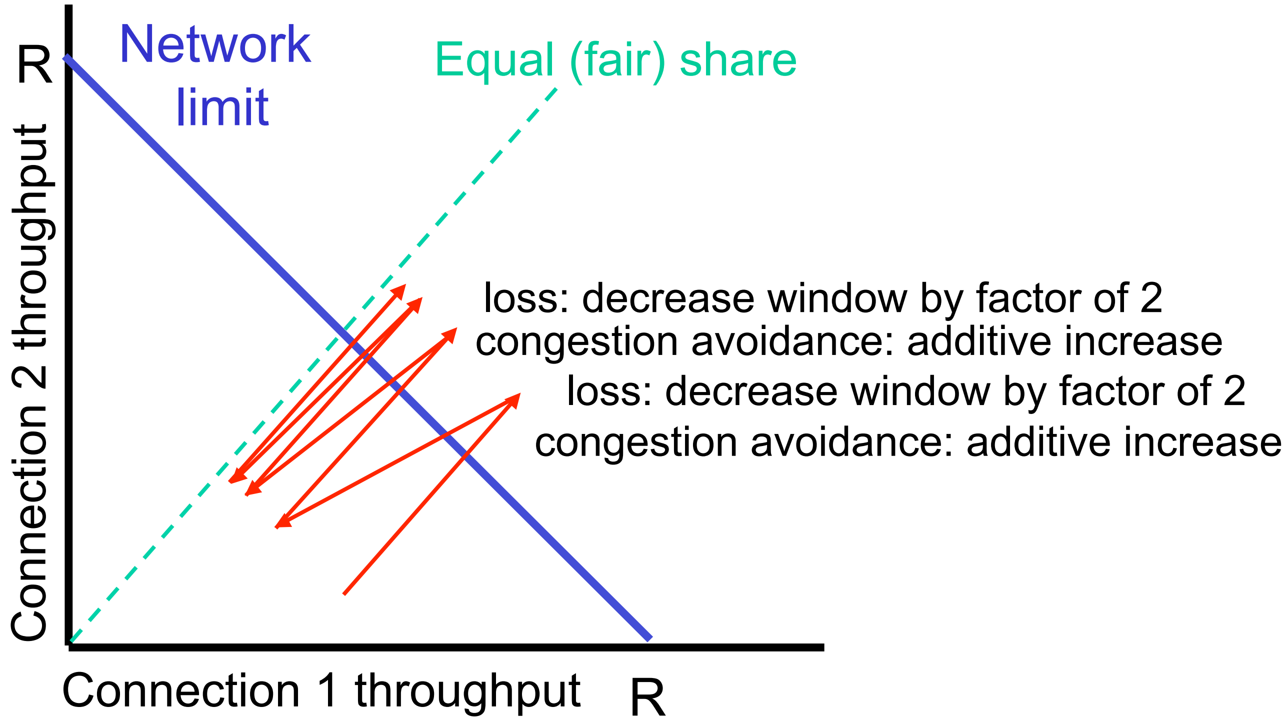
*Fairness definition:* if K TCP sessions share the same bottleneck link of bandwidth R, each should have average rate of R/K



# Why is TCP (AIMD) fair?

two competing TCP sessions:

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




- It can be proved that AIMD converges to (R/2, R/2)!

# [More on convergence]

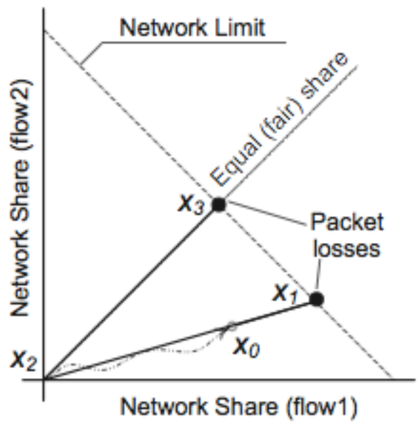



Fig. 10. Slow-Start Algorithm
$x_0 - x_1, \ldots, x_n - x_{n+1}$ multiplicative increase (both flows have the same increase rate of their congestion windows)
$x_1 - x_2$ equalization of the congestion window sizes

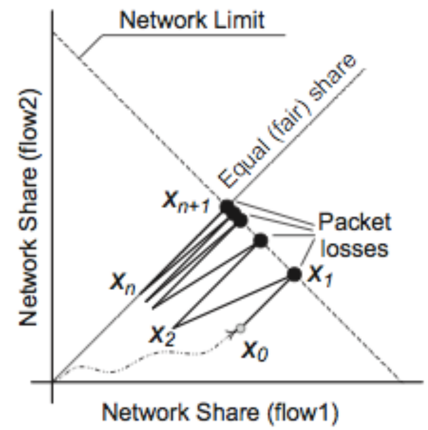



Fig. 11. Congestion avoidance (AIMD)
$x_0 - x_1, \ldots, x_n - x_{n+1}$ additive increase (both flows have the same increase rate of their congestion windows)
$x_1 - x_2, \ldots, x_{n-1} - x_n$ multiplicative decrease (a flow with the larger congestion window decreases more than a flow with the smaller)

- Goal: Fairness vs. efficiency
- Convergence: independently of the starting point and connection parameters (RTT, MSS).
- Efficiency: how quickly we converge, and how fully we utilize the network capacity
- Difficulty: Solving a global problem using distributed algorithms: coordination between network (L, R) and end connections (x1, x2).
- Intuition confirmed by the mathematics of Congestion Control.

# Fairness violated in practice

## Fairness and UDP

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

## Fairness and RTT

- Flows with longer RTTs get lower rate!

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

## Fairness and parallel TCP connections

- nothing prevents an app from opening parallel connections between 2 hosts.
- web browsers do this
  - Remember non-persistent HTTP?
- example: link of rate R supporting 9 connections;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2 !

## Fairness and multi-hop paths

- Flows that go over multiple hops compete with more flows

# TCP design - Discussion

- Several orthogonal goals coupled together
  - Reliability (treats the loss as symptom)
    - Detecting loss (acks/timeouts)
    - Reacting to loss (retransmissions)
  - Congestion control (treats the cause of loss)
    - Rate is controlled by the size of cwnd

      rate=cwnd*MSS/RTT
    - Fairness vs. maximizing throughput
  - Flow Control

- Some Problems
  - Loss over wireless does not always indicate congestion
  - Enforce fairness
  - TCP not ideal for "high bandwidth-delay product" networks
  - ….

# 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"