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

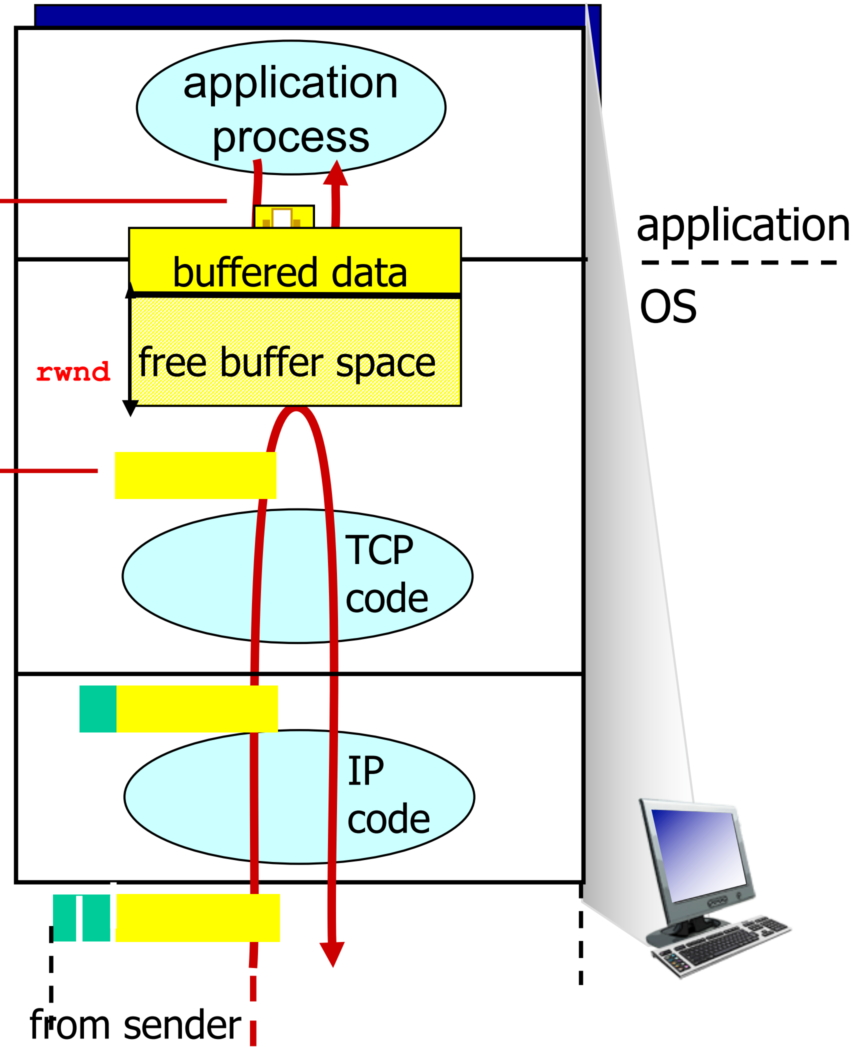
What is Congestion?

- informally: "too many sources sending too much data too fast for *network* to handle"
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)

Different from flow control!

application may remove data from TCP socket buffers ….

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

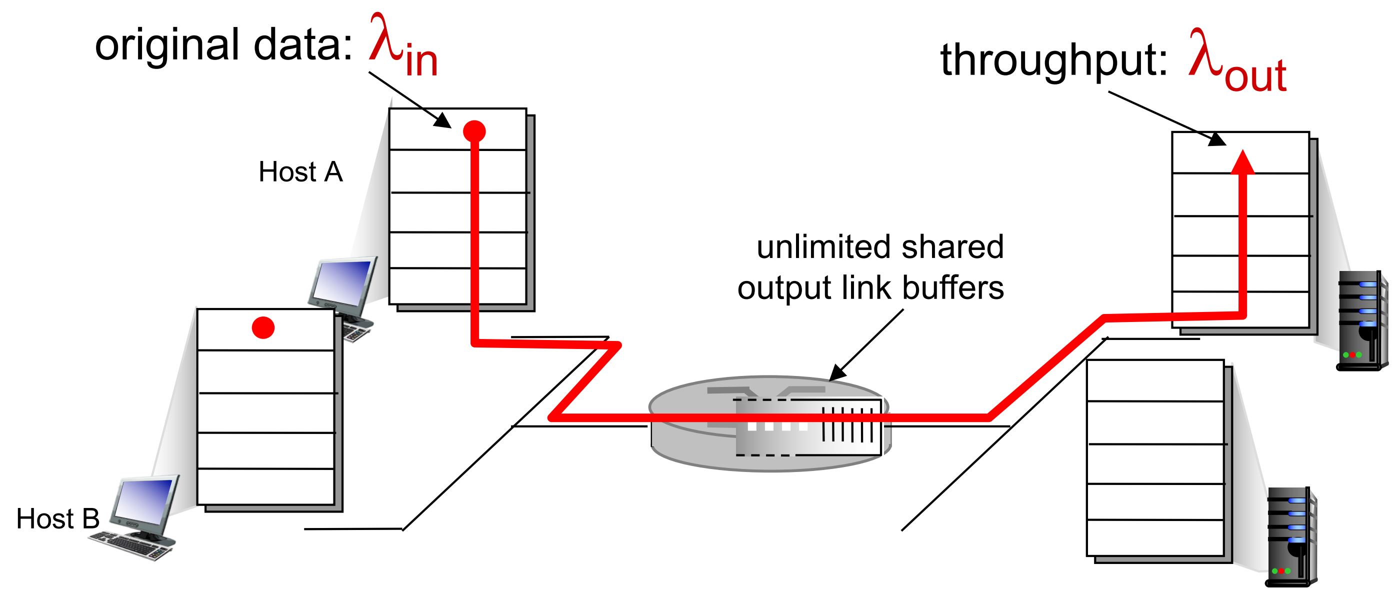


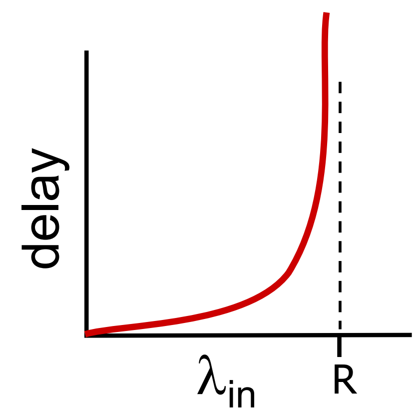
receiver protocol stack

flow control

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

Causes/costs of congestion: scenario 0

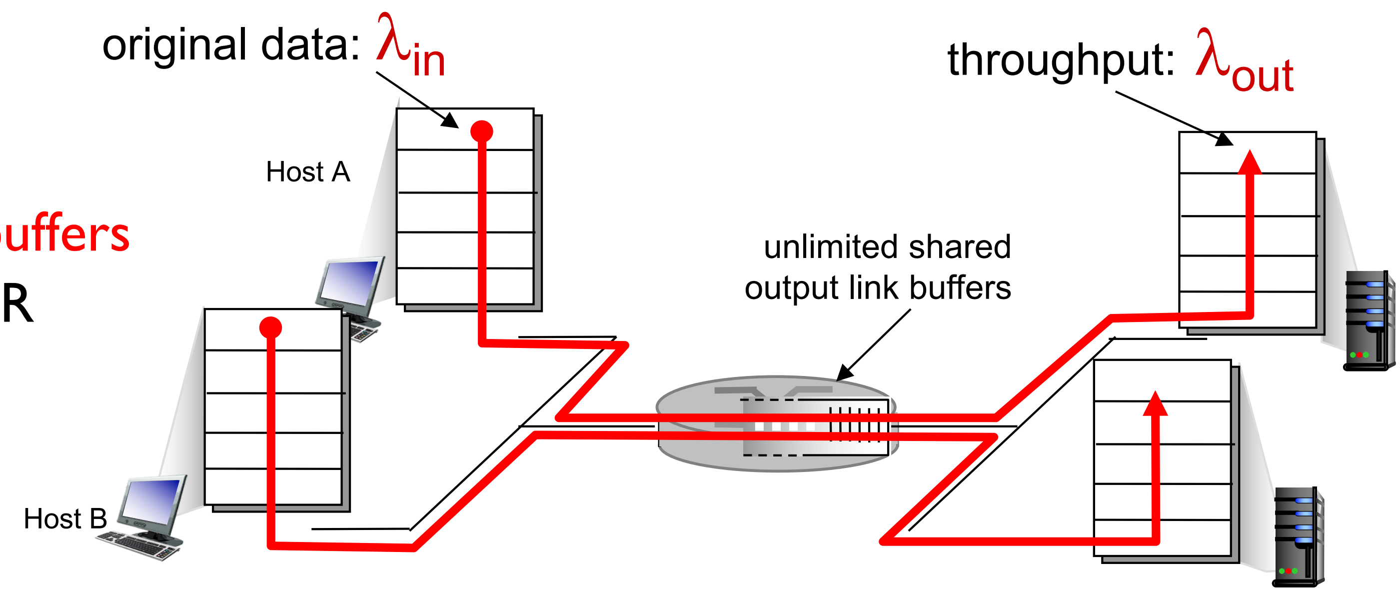


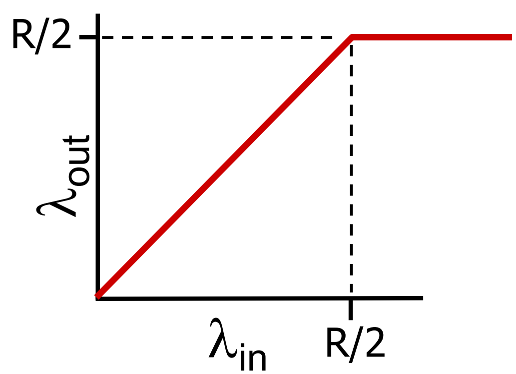


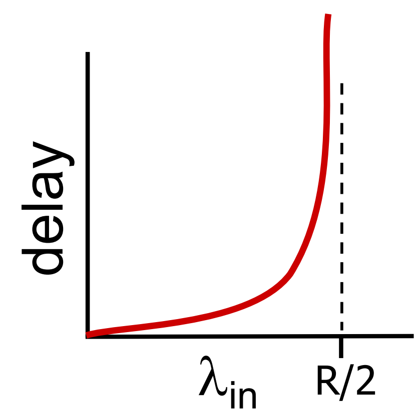
- maximum per-connection throughput: R (i.e.,the bottleneck link speed)
- λ_{in} >R → congestion

Causes/costs of congestion: scenario 1

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



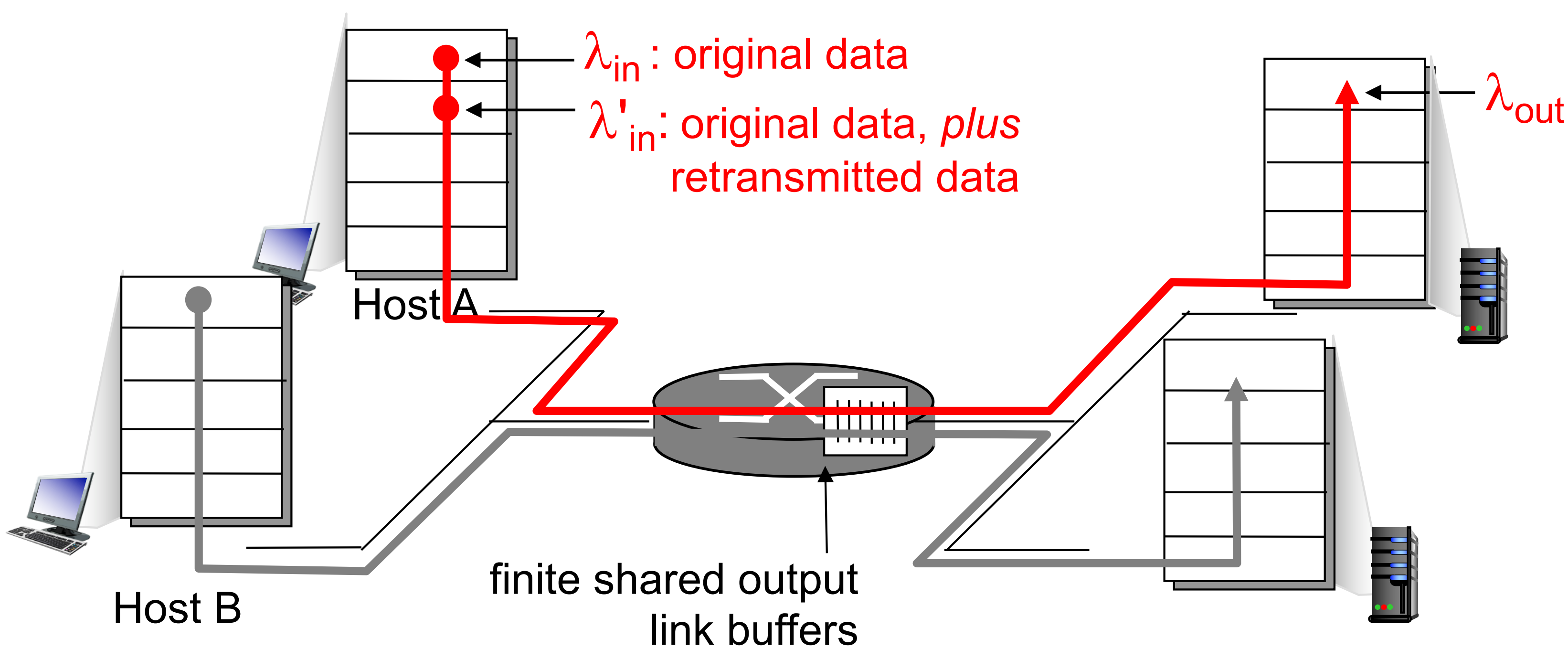




- maximum per-connection throughput: R/2
- large delays as arrival rate, λ_{in}, approaches capacity

Causes/costs of congestion: scenario 2

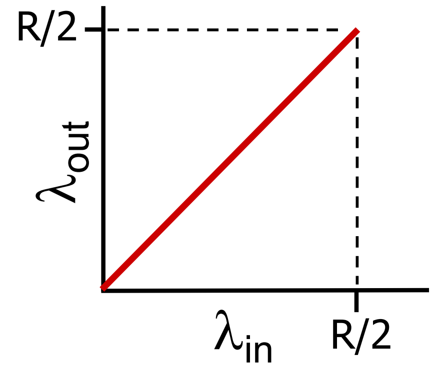
- one router, *finite buffers*
- sender retransmission of timed-out packet
  - application-layer input (λ_{in}), application-layer output(λ_{out})
  - transport-layer input includes *retransmissions* : $\lambda_{in}' \geq \lambda_{in}$

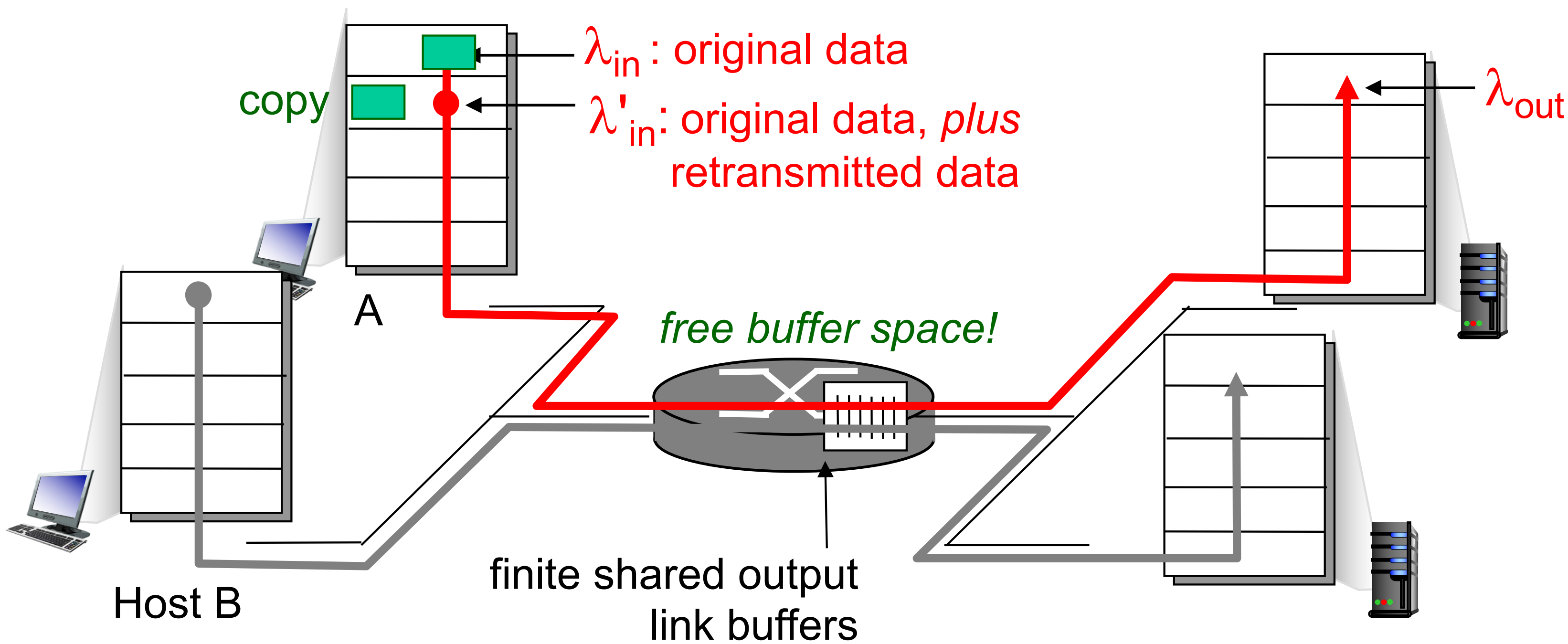


Causes/costs of congestion: scenario 2

idealization: perfect knowledge

- sender sends only when router buffers available

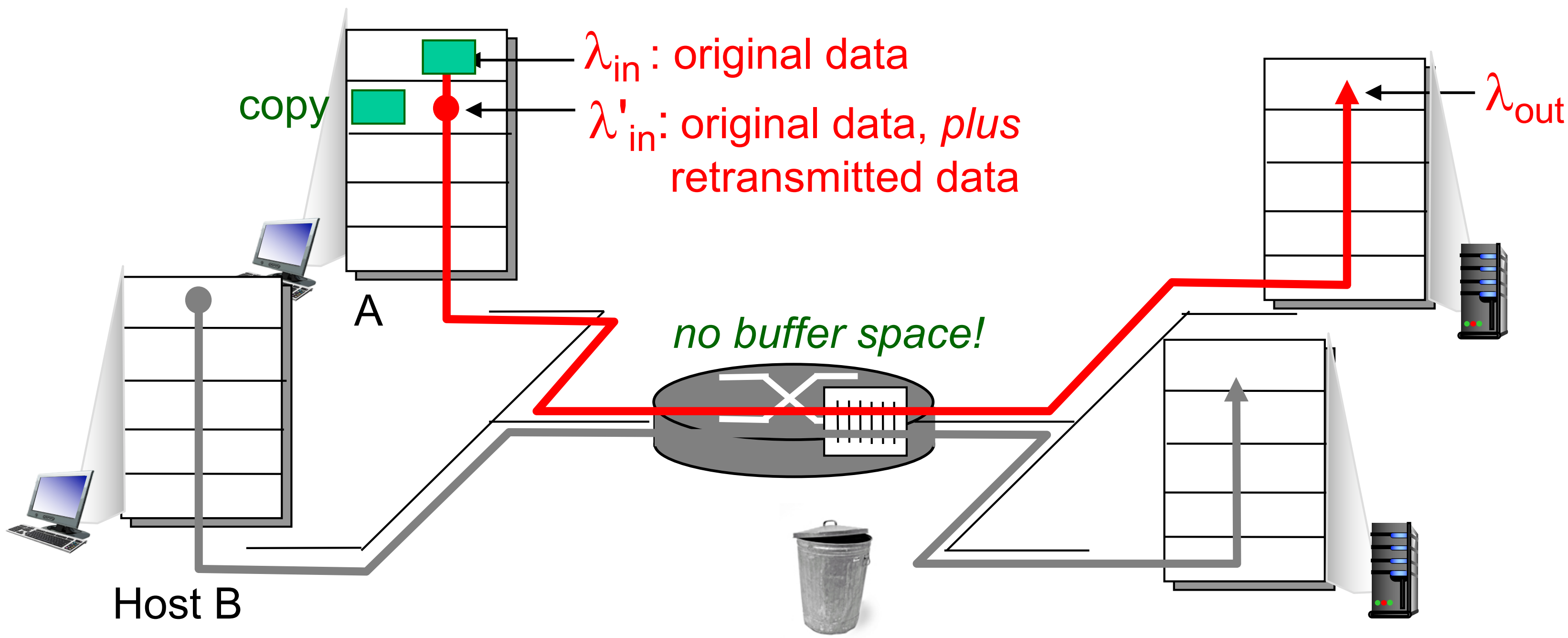




Causes/costs of congestion: scenario 2

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

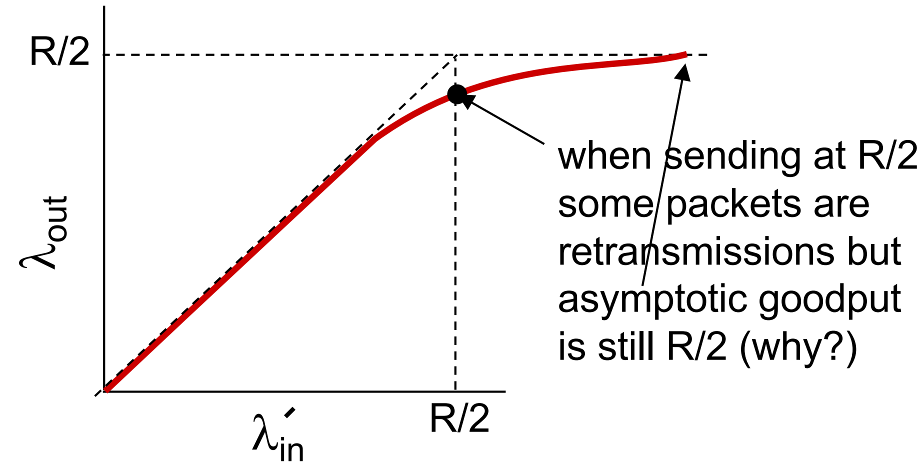
- sender only resends if packet *known* to be lost

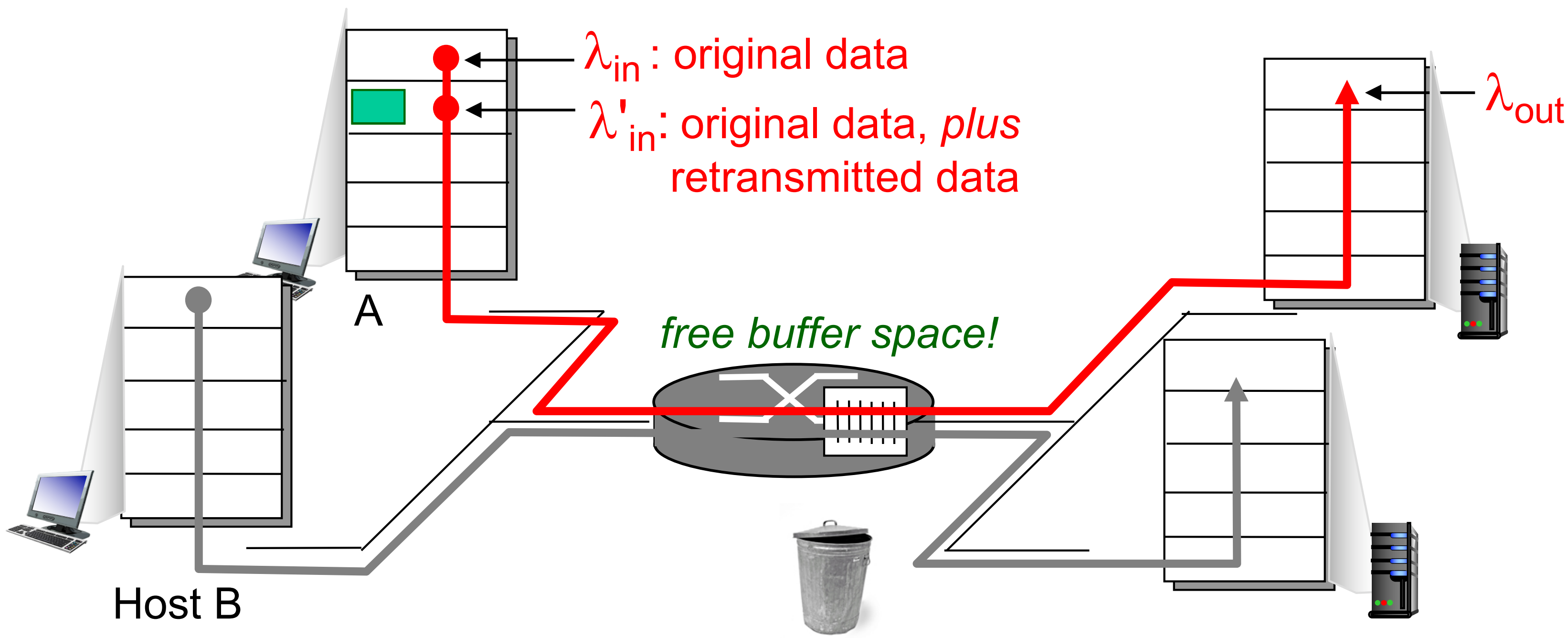


Causes/costs of congestion: scenario 2

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

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

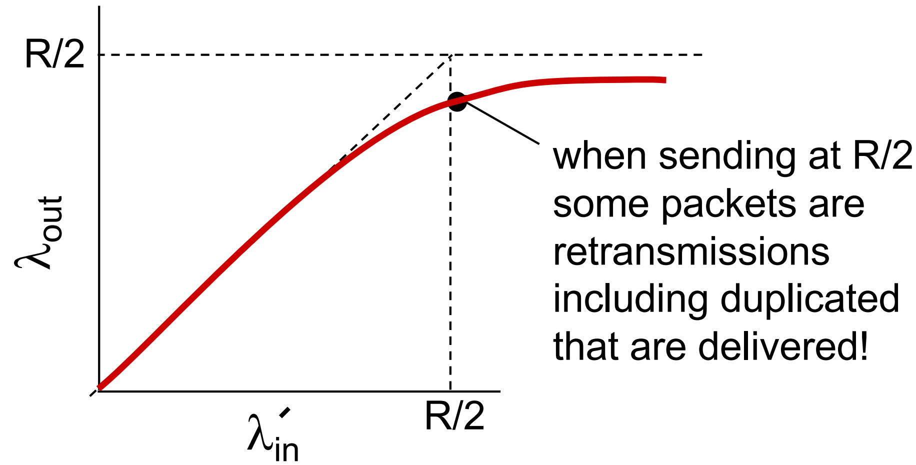


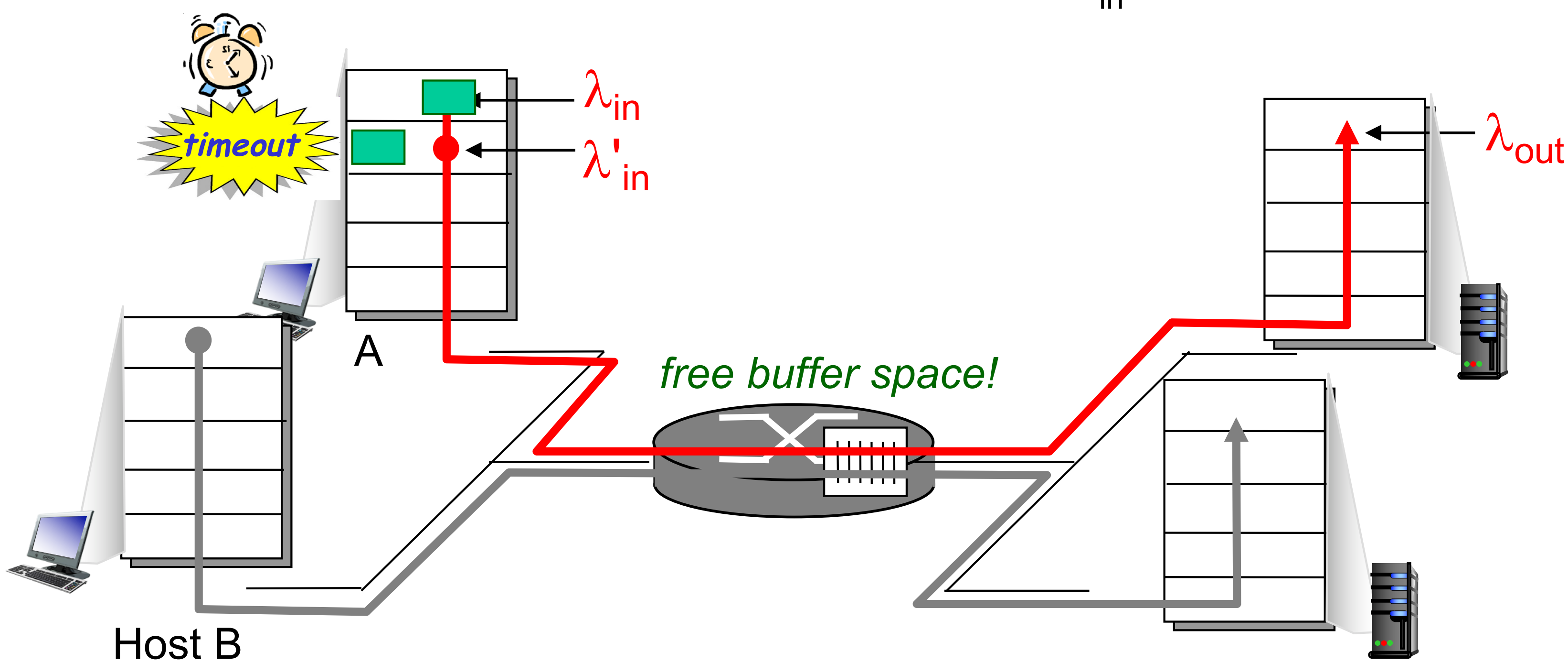


Causes/Costs of congestion: scenario 2

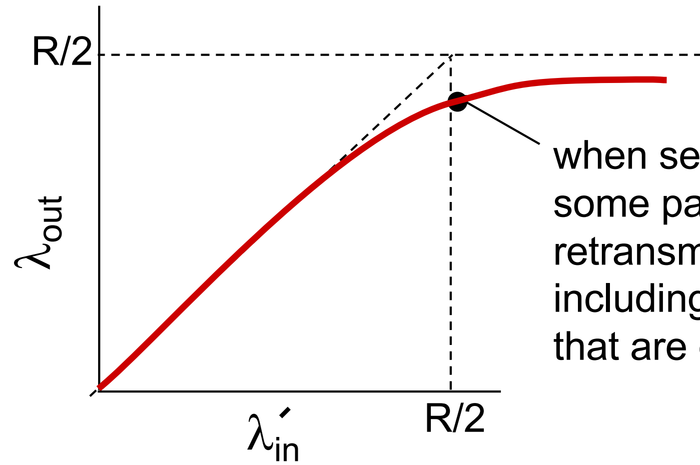
Realistic: duplicates

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





Causes/Costs of congestion: scenario 2



Downsides of congestion:

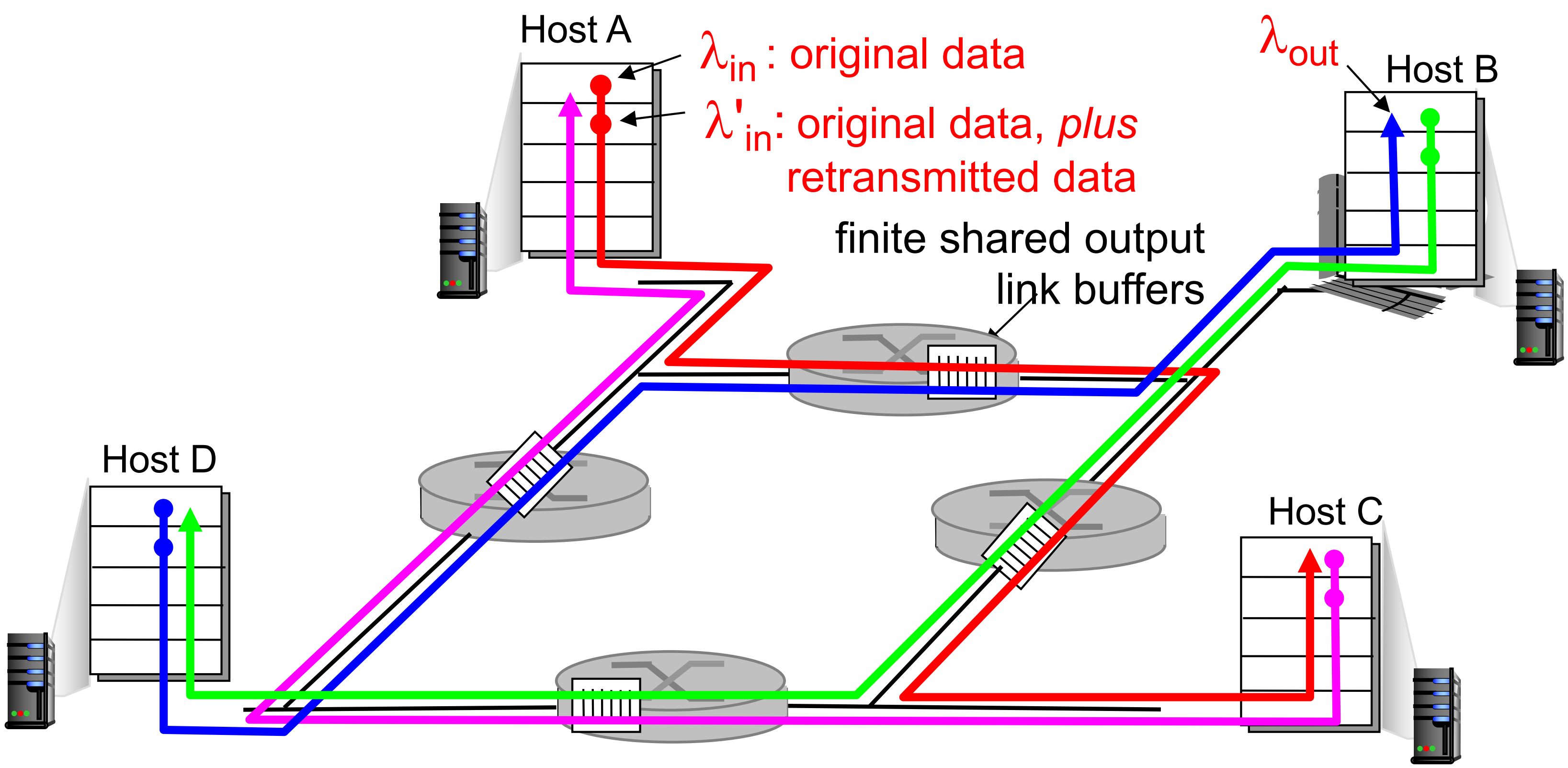
- queueing delay→ loss
- more work (retransmission, duplicates) for given "goodput"
- unneeded retransmissions: link carries multiple copies of pkt
  - decreasing goodput

Causes/costs of congestion: scenario 3

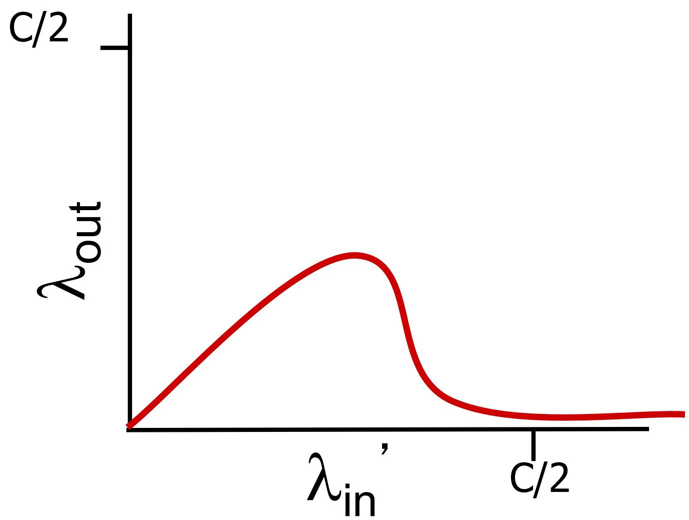
- four senders
- multihop paths
- timeout/retransmit

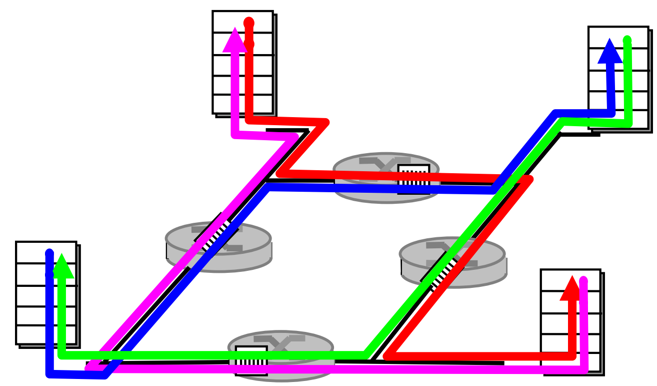
Q: what happens as λ_{in} and λ_{in}' increase ?

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



Causes/costs of congestion: scenario 3





Another downside 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
  - adapt sending window

~~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

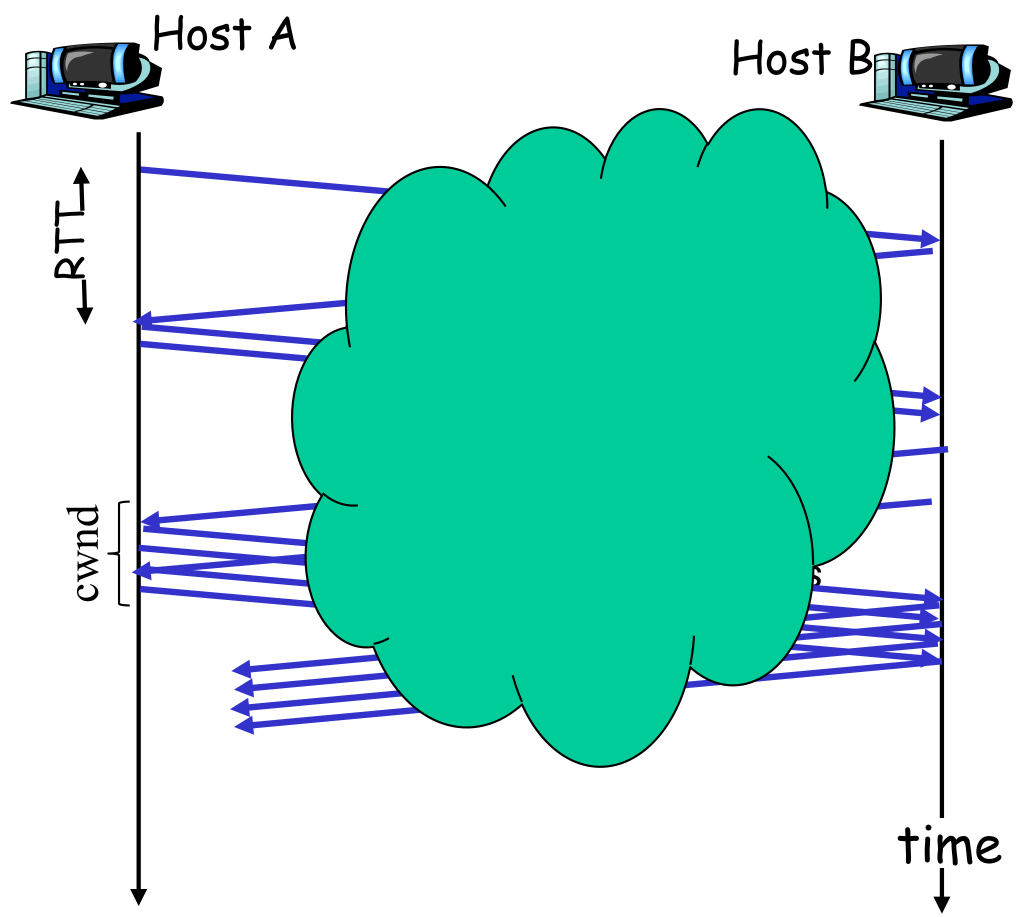
3.6 principles of congestion control

3.7 TCP congestion control

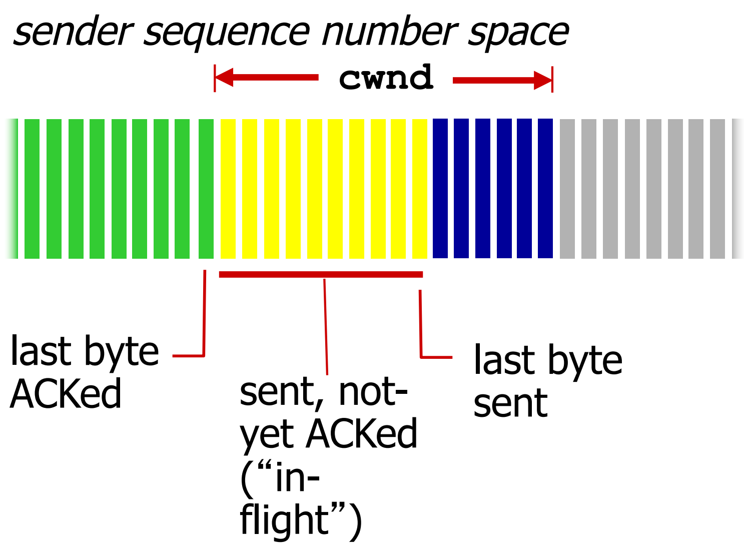
End-to-End congestion control from the (TCP) source's point of view

- 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
- Sending rate (roughly):
  - send cwnd bytes, wait RTT for ACKS, then send more bytes

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



TCP Congestion Control: details



- Sender limits transmission:

$$\texttt{LastByteSent-LastByteAcked} \leq \texttt{min\{cwnd, rwnd\}}$$

- Sender adapts **cwnd**, dynamically, as a function of perceived network congestion

TCP sending rate:

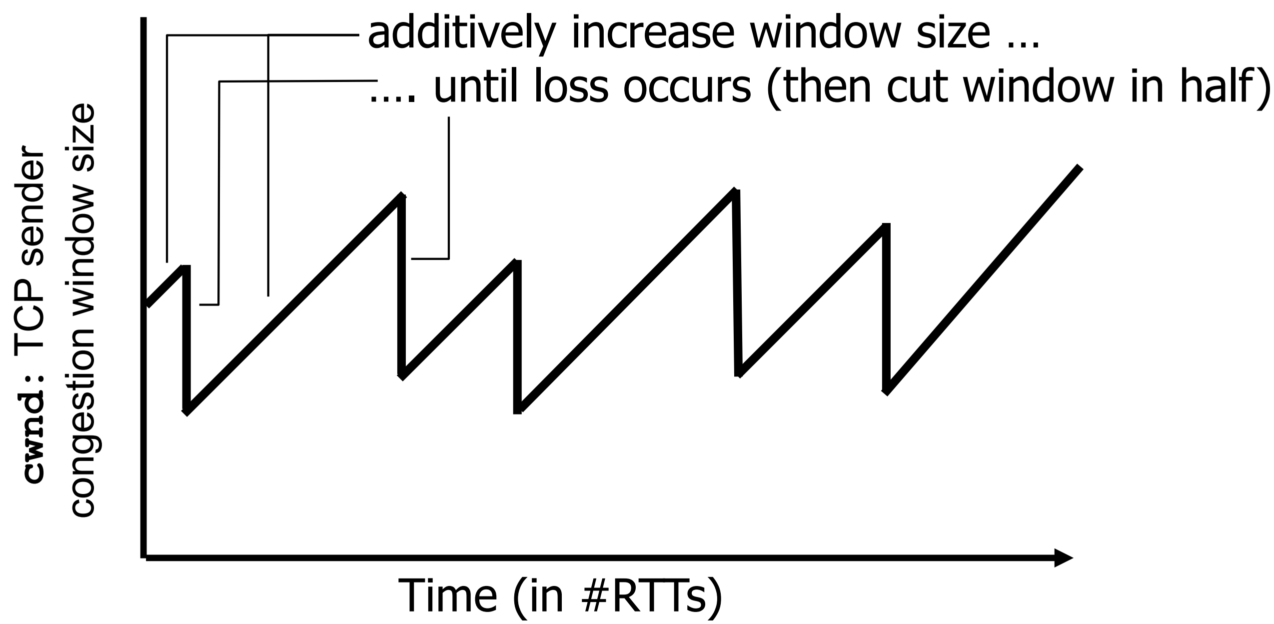
- *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

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

TCP congestion avoidance (CA):
additive increase multiplicative decrease (AIMD)

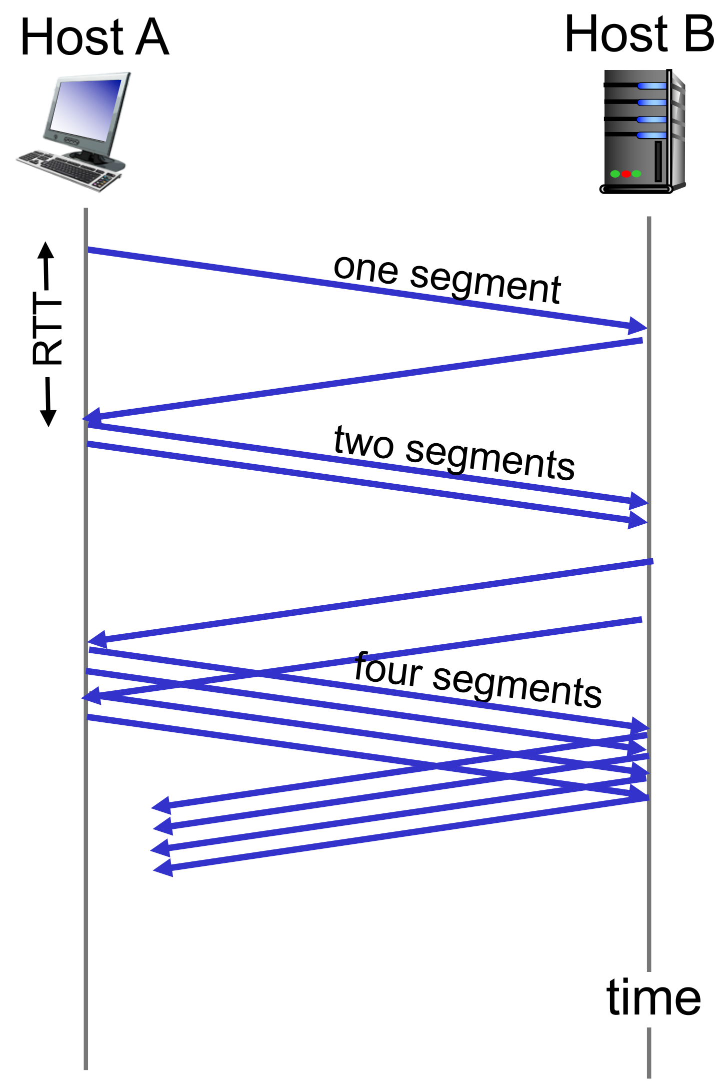
- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **`cwnd`** by 1 MSS every RTT until loss detected
  - *multiplicative decrease:* cut **`cwnd`** in half after loss

AIMD saw tooth behavior: probing for bandwidth



TCP 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 can deliver some segments
  - `cwnd` is cut in half window then grows linearly
- TCP Tahoe always sets `cwnd` to 1
  - detected by timeout or 3 duplicate acks

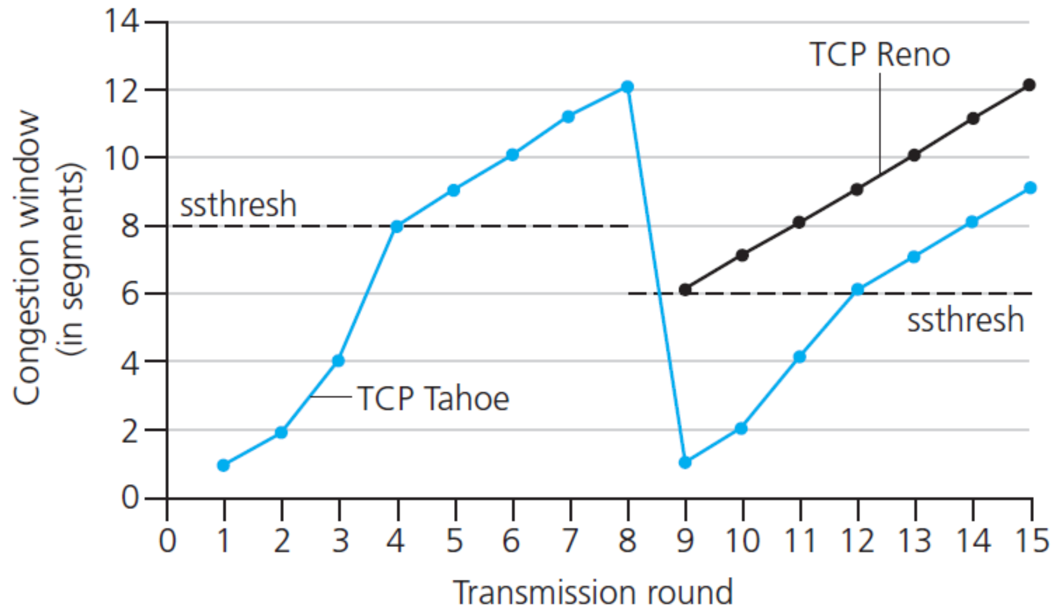
When to switch from SS to CA

Q: when should the exponential increase switch to linear?

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

- congestion is around the corner.
- do not double, be conservative.

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



Implementation:

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

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

When/How to exit Additive Increase?

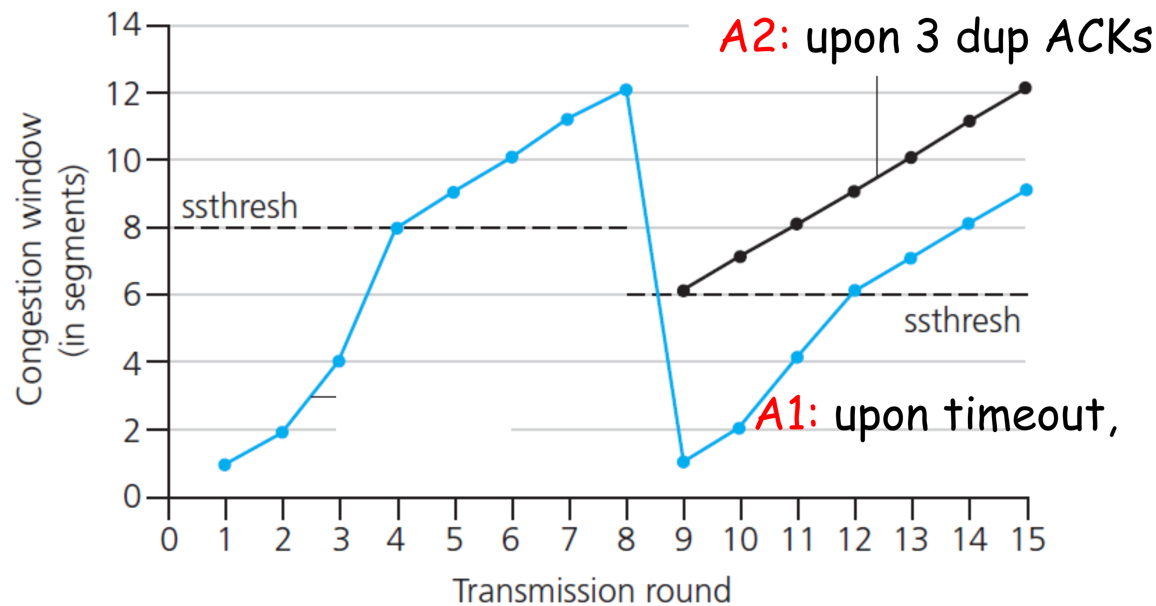
Q: when/how to end additive increase?

A: upon loss

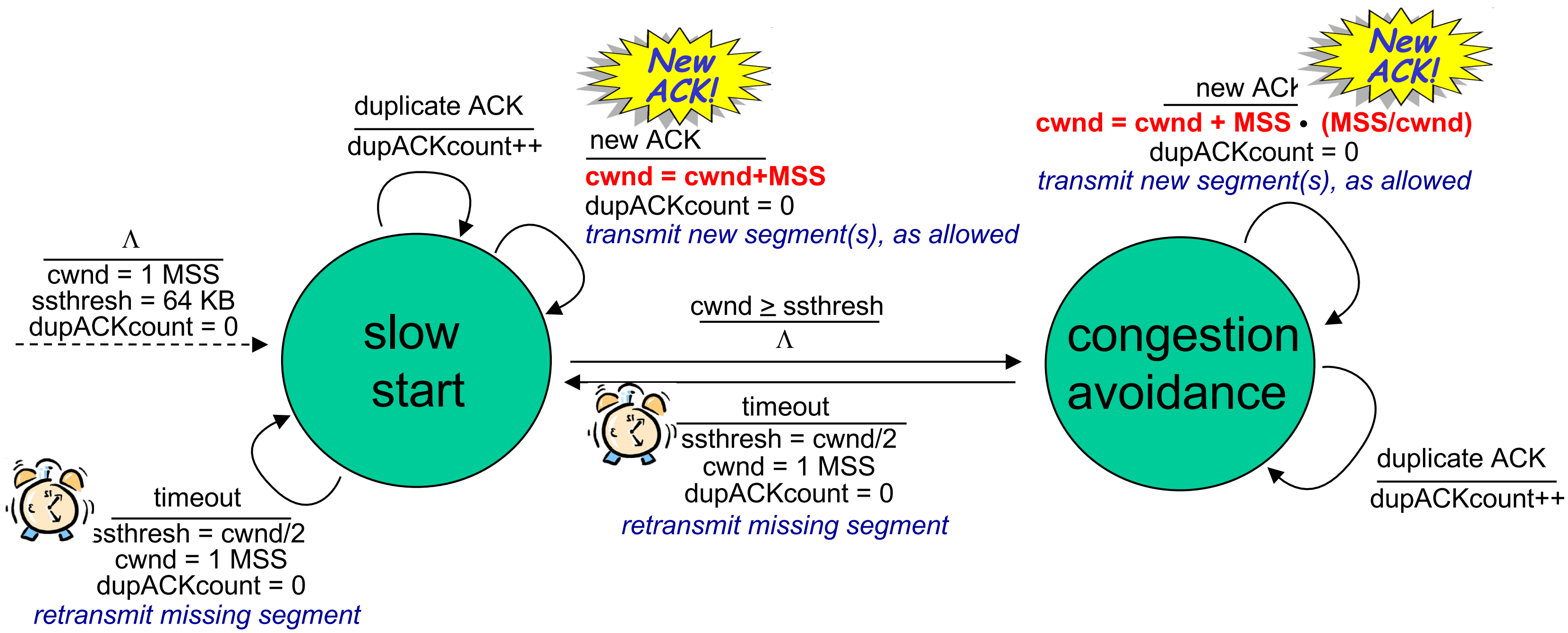
A1: upon timeout, set **cwnd:= 1**, enter slow start

A2: upon 3 dup ACKs

- **cwnd:=cwnd/2+3MS**
- stay in Congestion Avoidance



Summary: TCP Congestion Control



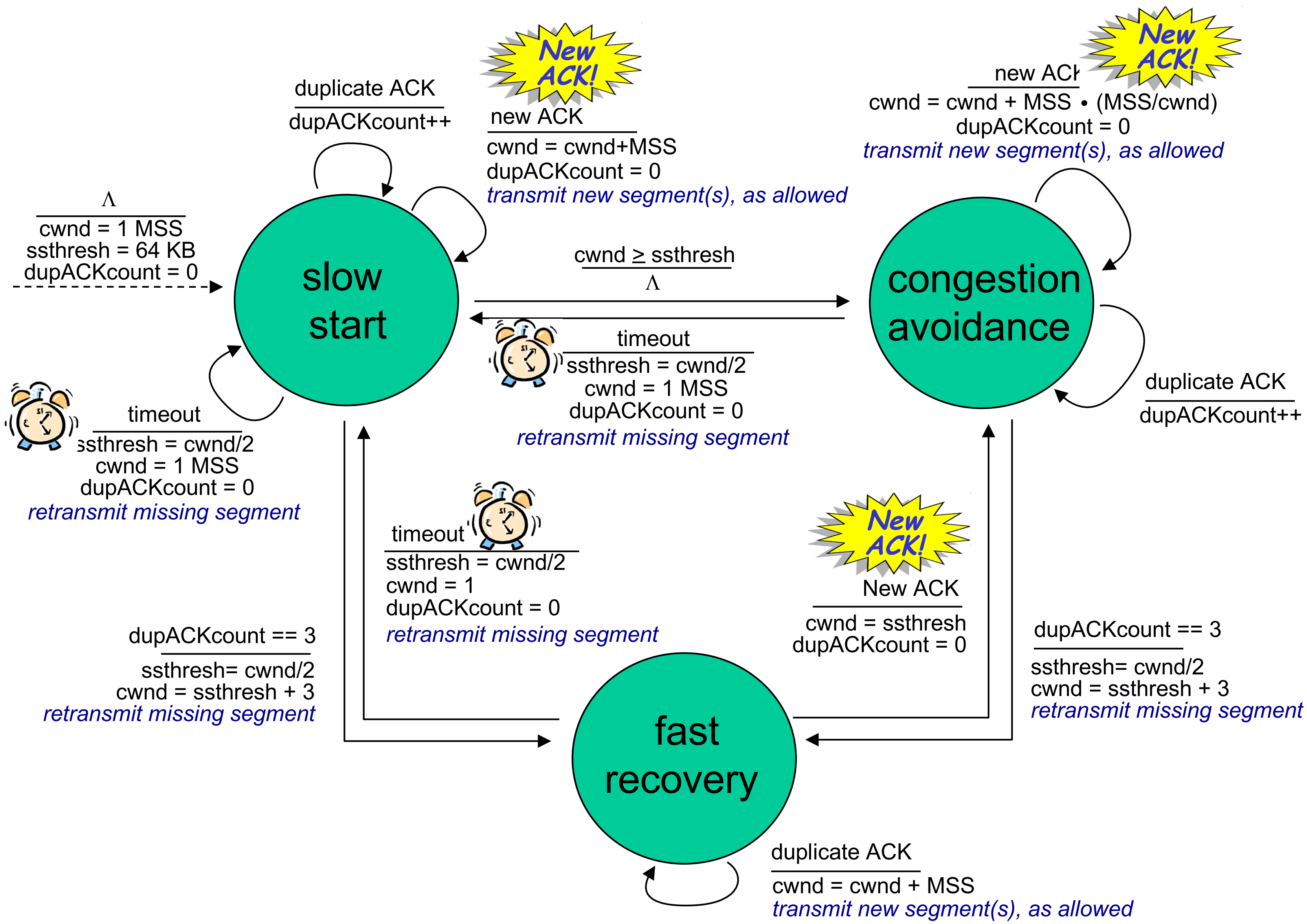
Notes:

- Every segment (MSS B) results in one ACK.
- In **one RTT**: cwnd Bytes outstanding → cwnd/MSS #segments sent and #ACKs received
- Additive increase: +1 MSS B per RTT → +1 MSS/(cwnd/MSS) B **per ACK**
- Exponential increase: +1 MSS B **per ACK** → +cwnd B per RTT.
- [Note: cwnd: +1 per ACK vs +1 per RTT: discuss on the board]

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 (for reliability)
  - Inflate window: Increase by 1 MSS per every duplicate ACK received (3 per RTT)
  - 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



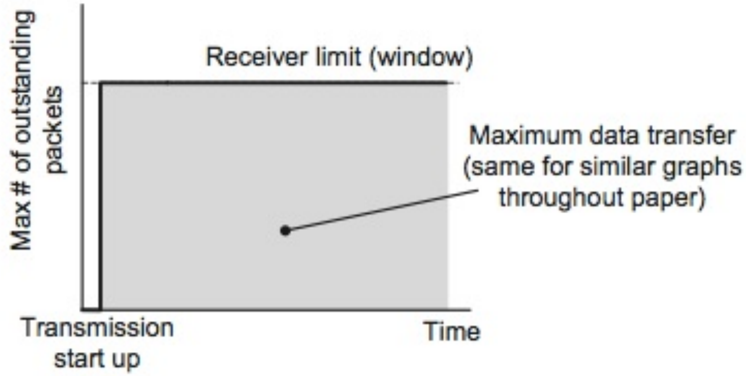


Fig. 6. Outstanding data packets allowance dynamics as defined in RFC793 (network limits are not considered)

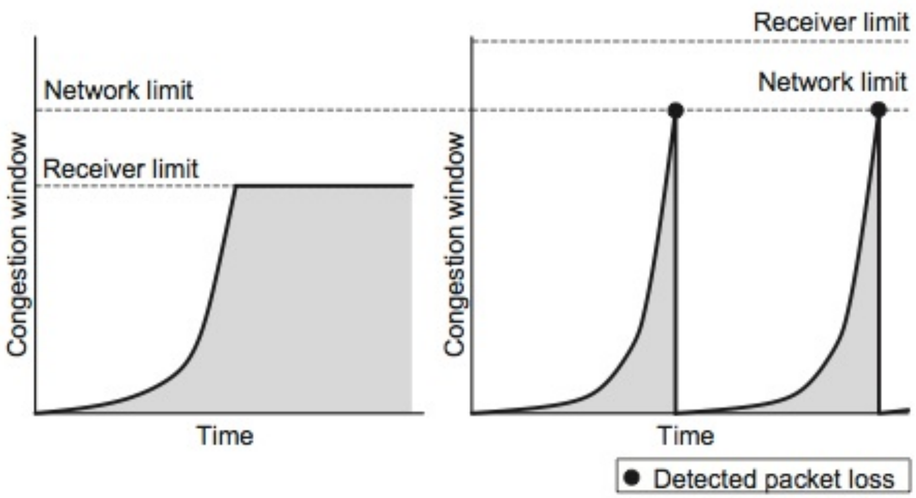


Fig. 7. Congestion window dynamics and effectiveness of *Slow-Start* if limit is imposed by legacy flow control (left) and network (right)

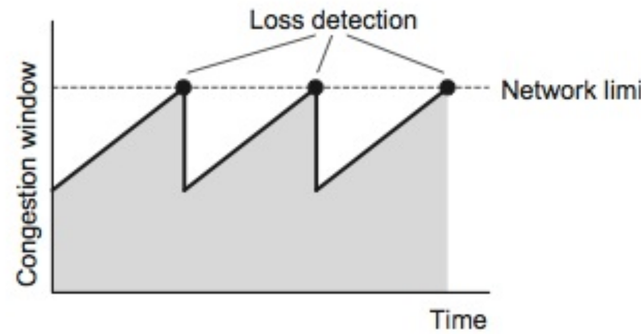


Fig. 8. Congestion window dynamics and effectiveness of *Congestion Avoidance*

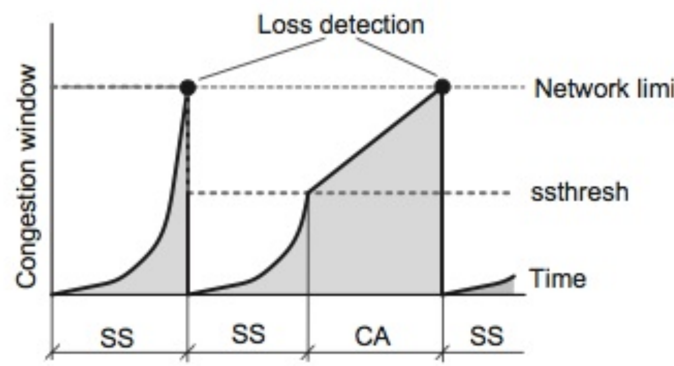


Fig. 9. Congestion window dynamics of combined *Slow-Start* (SS) *Congestion Avoidance* (CA)

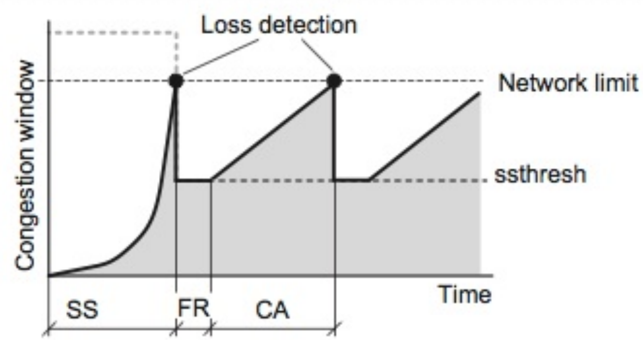
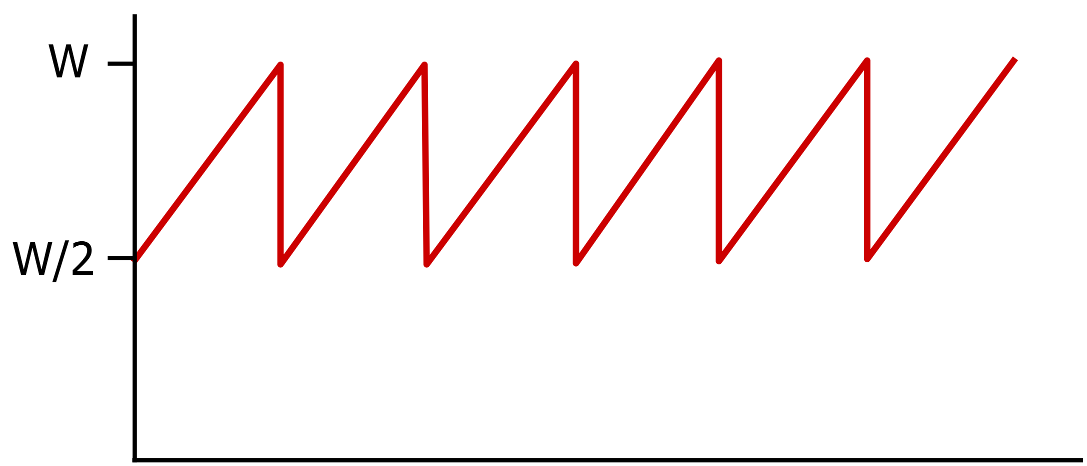


Fig. 15. Congestion window dynamics of TCP Reno
(SS: the *Slow Start* phase, CA: the *Congestion Avoidance* phase, FR: the *Fast Recovery* phase)

TCP throughput

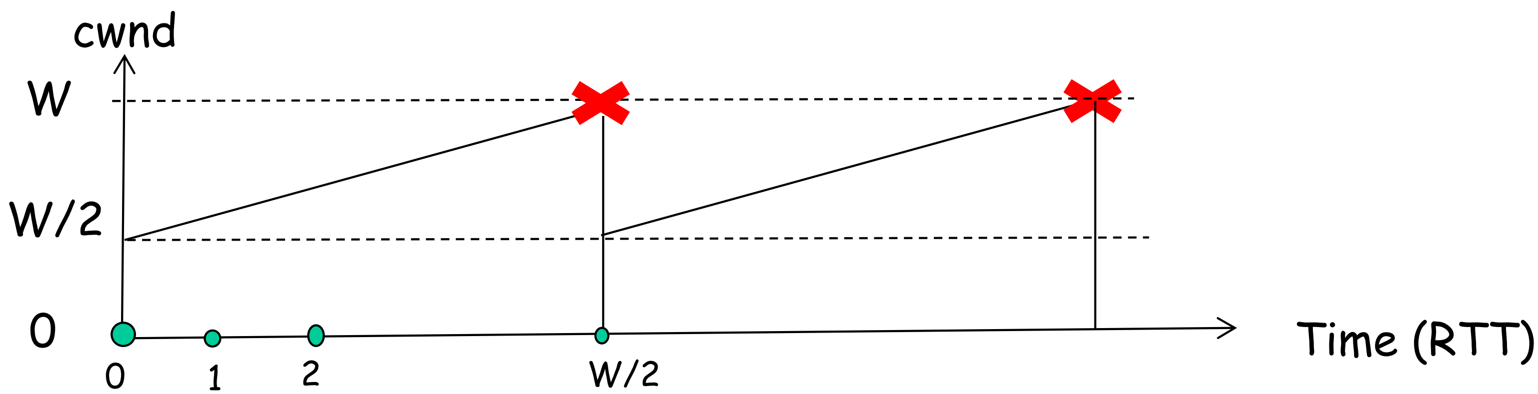
- avg. TCP throuput as function of window size, RTT?
  - ignore slow start, assume always data to send
- W: window size (measured in bytes) where loss occurs
  - avg. window size (# in-flight bytes) is ¾ W
  - avg. throuput is 3/4W per RTT

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



Average TCP throughput][

- Simplified analysis (Problem 45, HW4):
  - ignore slow start, assume always data to send
  - consider the period from W/2 to W (when loss occurs)

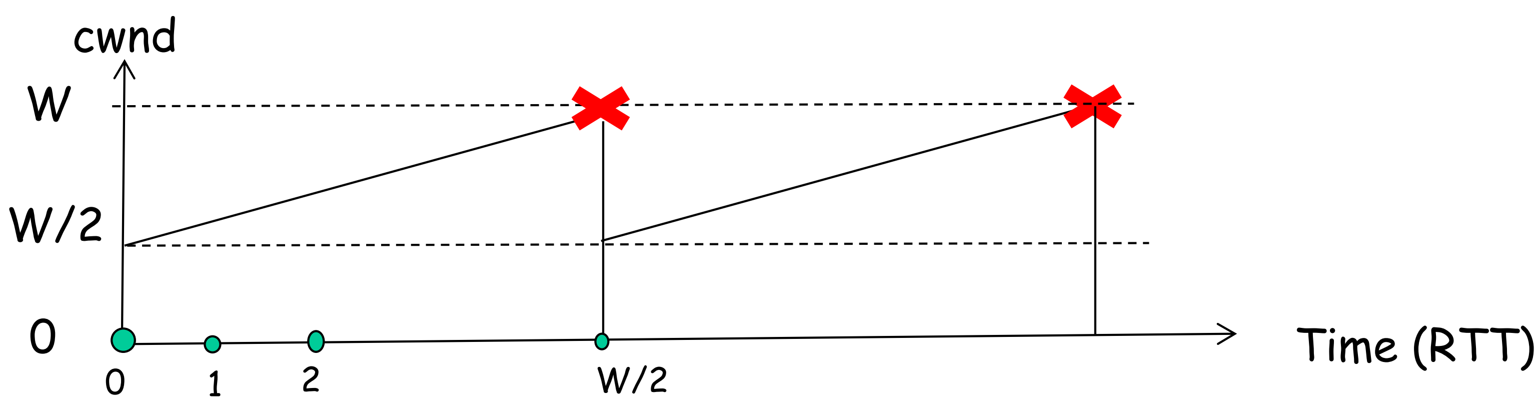


  - avg. window size (# in-flight segments) is ¾ W
  - avg. throughput is 3/4W per RTT or: $\frac{3}{4}\frac{W}{RTT}$ bytes/sec
  - 1 segment lost per W
  - avg. throughput in terms of loss rate L (in bytes/sec):

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

Average TCP throughput

- Simplified analysis (Problem 45, HW3):
  - ignore slow start, assume ther eis always data to send
  - consider the period from W/2 to W (when loss occurs)



- Consider time period [0, (W/2)*RTT]
- #Packets sent: W/2+(W/2+1)+(W/2+2)…+ W=……f(W)
- Avg throughput: $\dfrac{\text{\#packets sent * MSS (Bytes)}}{\text{W/2 * RTT (sec)}}$
- #Packets lost: 1→ Loss rate: L= 1/(#packets sent)=1/f(W)
- Avg. throughput in terms of loss rate L (in B/sec): $\boxed{\dfrac{1.22 \cdot MSS}{RTT\sqrt{L}}}$

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!*
  - *Average throughput*

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

- new versions of TCP needed & developed for "high bandwidth-delay product" networks
  - E.g. XCP

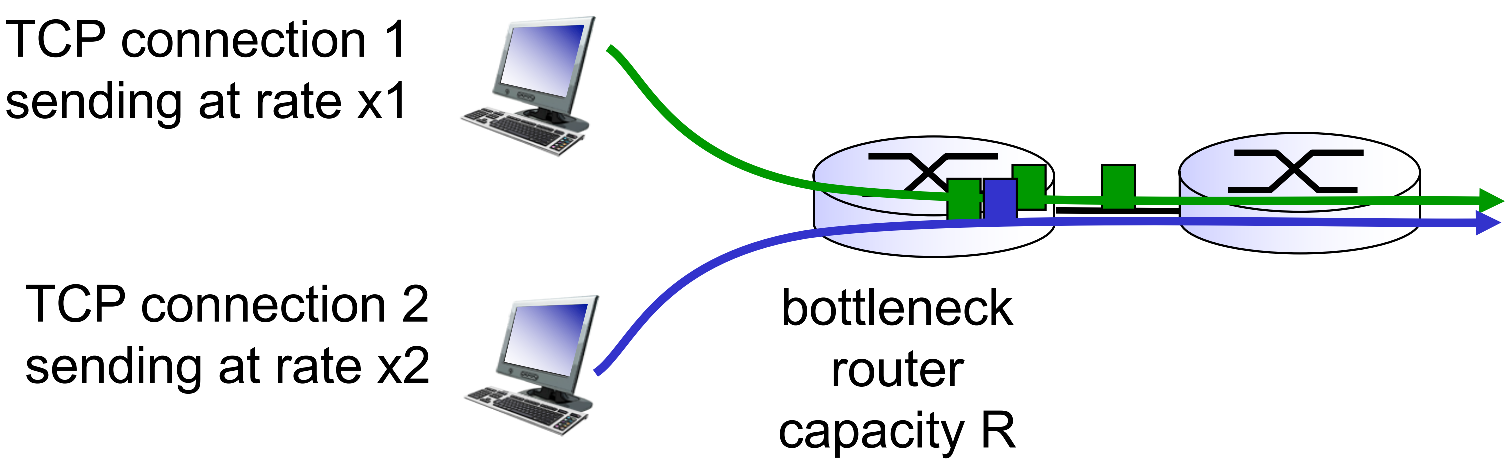
TCP Fairness

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

Example: 2 connections

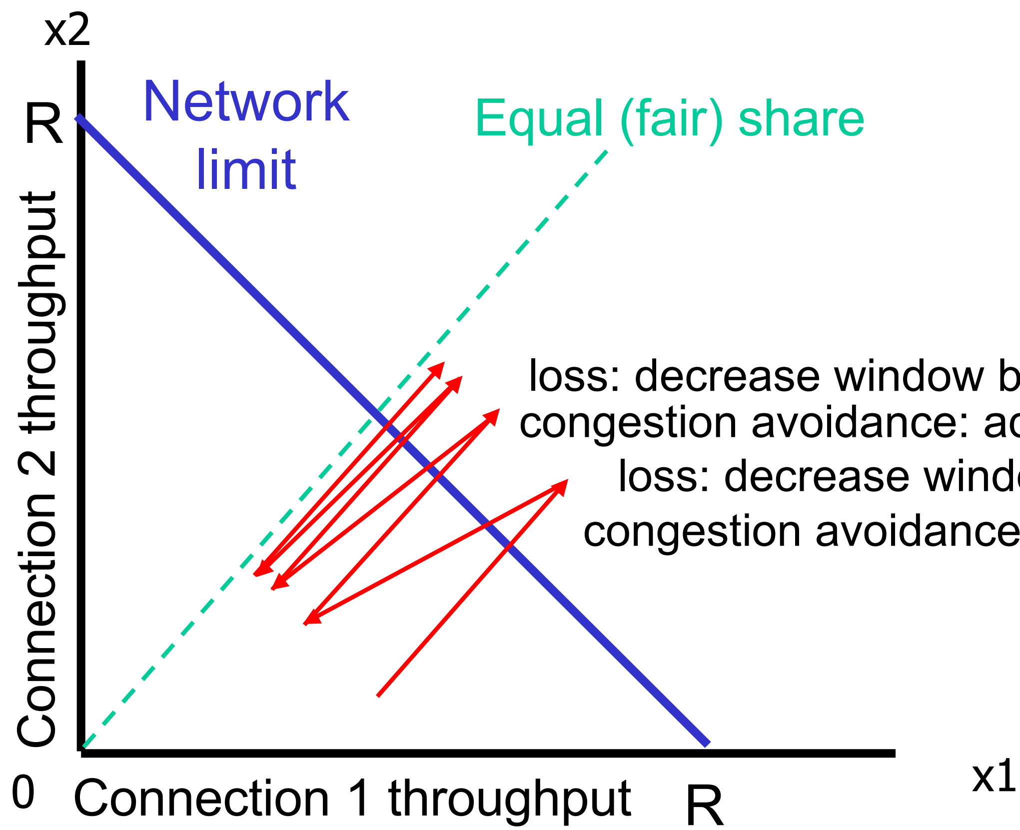
- Capacity constraints: x1+x2 <=R
- Fairness constraint: x1=x2
- Objective ? E.g. max (x1+x2)



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)!

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

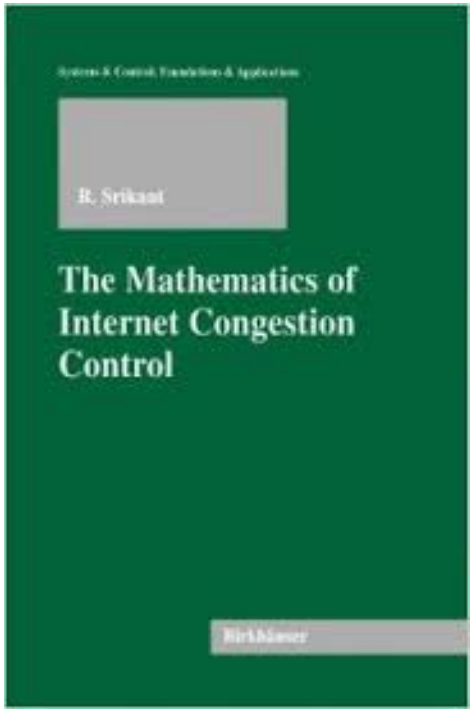
- Flows that go over multiple hops compete with more flows

Rate Control - Discussion

- 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
- Notion of fairness: proportional, max min, etc…
- 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.

History and Food for thought

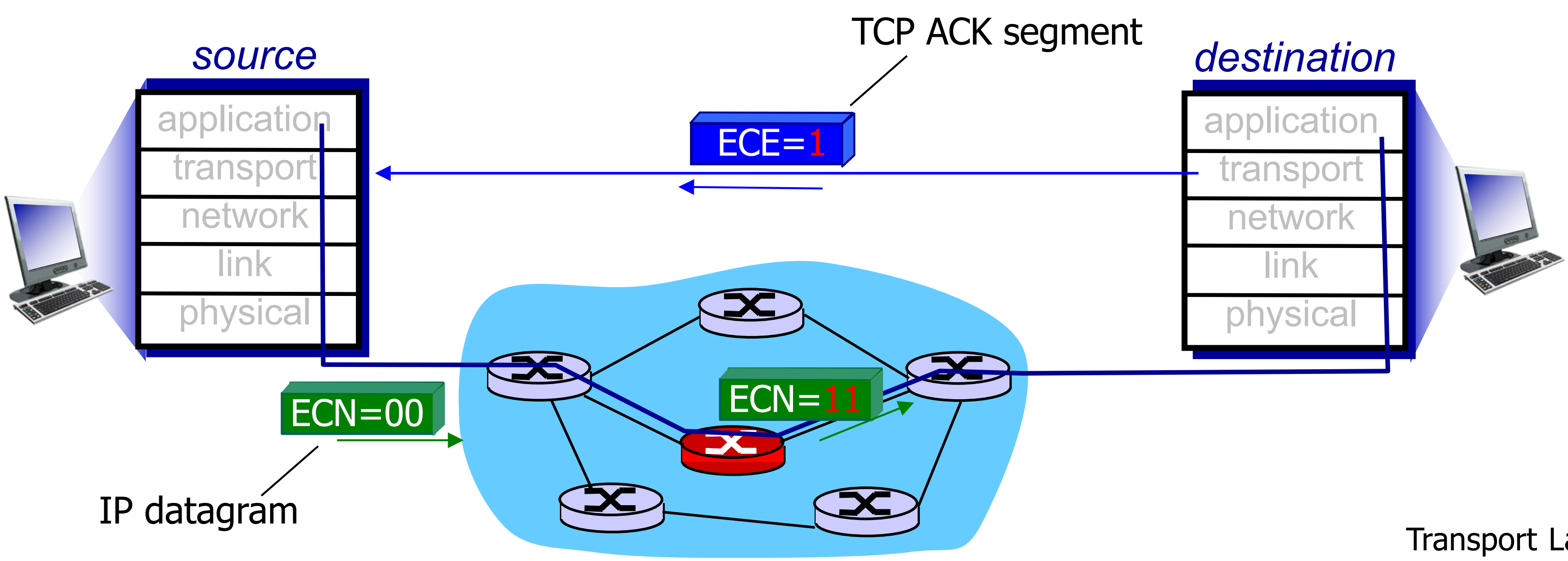
- General problem:
  - From 2 connections, one bottleneck
  - To many flows, many hops, complicated topology
- V.Jacobson's TCP congestion avoidance, 1988
  - Paper: http://research.microsoft.com/en-us/um/people/pcosta/cn_slides/jacobson88congestion.pdf
  - TCP RFCs: https://en.wikipedia.org/wiki/TCP_congestion_control
- F. Kelly's paper started the analysis, 1998
  - http://www.jstor.org/stable/3010473?seq=1#page_scan_tab_contents
- Textbook: The Mathematics of Congestion control by R.Srikant, 2004
  - http://www.amazon.com/Mathematics-Internet-Congestion-Control-Systems/dp/0817632271



Explicit Congestion Notification (ECN)

network-assisted congestion control:

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
  - 00 – Non ECN-Capable Transport, Non-ECT
  - 10 – ECN Capable Transport, ECT(0)
  - 01 – ECN Capable Transport, ECT(1)
  - 11 – Congestion Encountered, CE.
- 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



TCP design - Discussion

- Two 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/RTT
    - Fairness vs. maximizing throughput
- Some Problems
  - TCP not ideal for wireless: Loss over wireless does not always indicate congestion
  - TCP not ideal for "high bandwidth-delay product" networks
  - How to enforce fairness?
  - Recent developments
    - XCP (high BW delay-product networks), TFRC (rate-based vs window-based for multimedia)
    - Google QUIC in Chrome Browser: application later on top of UDP, improves CC+Reliability
    - DCTCP: data center TCPs, uses ECN to support mix of short- and long-lived flows