AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

15CSE312
COMPUTER NETWORKS
3-0-0 3

TRANSPORT LAYER

Chapter 3: Transport Layer

- TCP Congestion Control

Chapter 3: Transport Layer

Transport-layer protocols are implemented in the end systems but not in network routers.

On the sending side, the transport layer converts the application layer messages it receives from a sending application process into transport-layer packets, known as transport-layer segments in Internet terminology. This is done by (possibly) breaking the application messages into smaller chunks and adding a transport-layer header to each chunk to create the transport-layer segment. The transport layer then passes the segment to the network layer at the sending end system, where the segment is encapsulated within a network-layer packet (a data gram) and sent to the destination

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

Packet Loss : Typically results from the overflowing of router buffers as the network becomes congested. Packet retransmission thus treats a symptom of network congestion (the loss of a specific transport-layer segment) but does not treat the cause of network congestion—too many sources attempting to send data at too high a rate. To treat the cause of network congestion, mechanisms are needed to throttle senders in the face of network
congestion.

Principles of Congestion Control : In this section, we consider the problem of congestion control in a general context, seeking to understand why congestion is a bad thing, how network congestion is manifested in the performance received by upper-layer applications, and various approaches that can be taken to avoid, or react to, network congestion

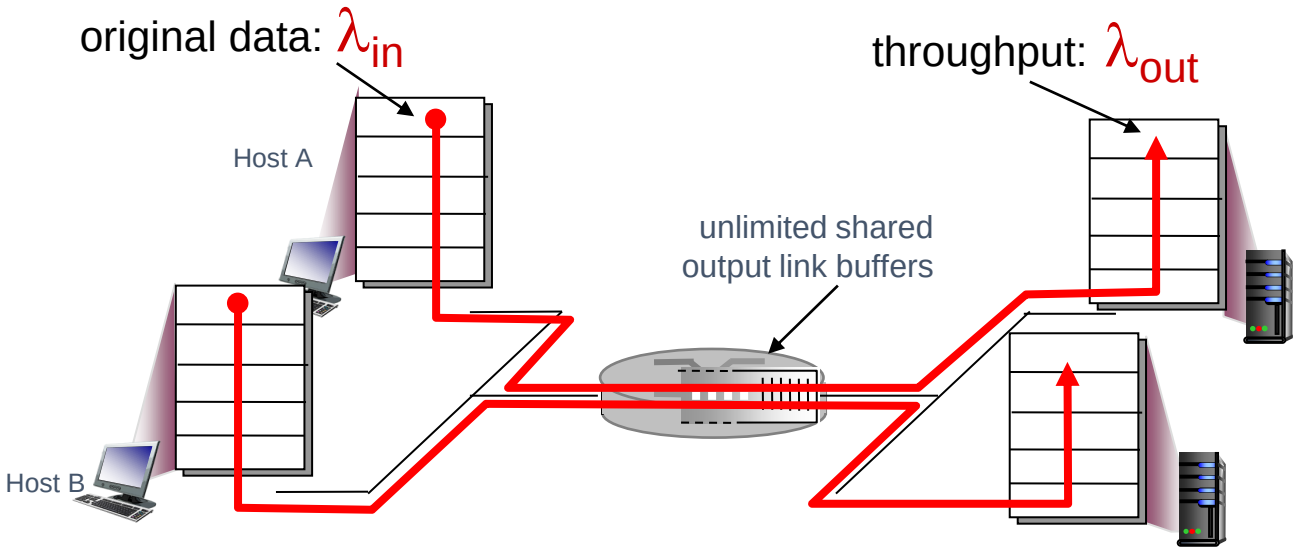
Principles of congestion control

congestion:

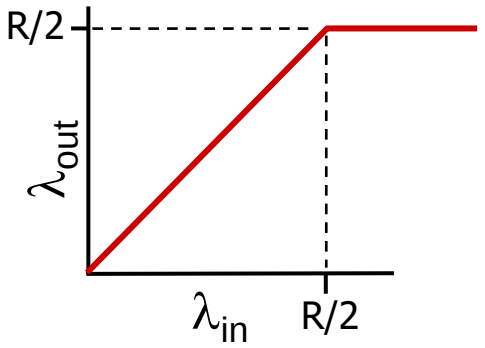
- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

Causes/costs of congestion: scenario 1

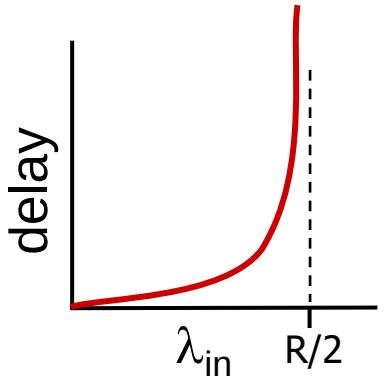
- two senders, two receivers
- one router, infinite buffers
- output link capacity: R
- no retransmission



From a throughput standpoint, performance is ideal—everything that is sent is received. Note that the average host sending rate cannot exceed *R*/2 under this scenario, since packet loss is assumed never to occur.



- maximum per-connection throughput: R/2

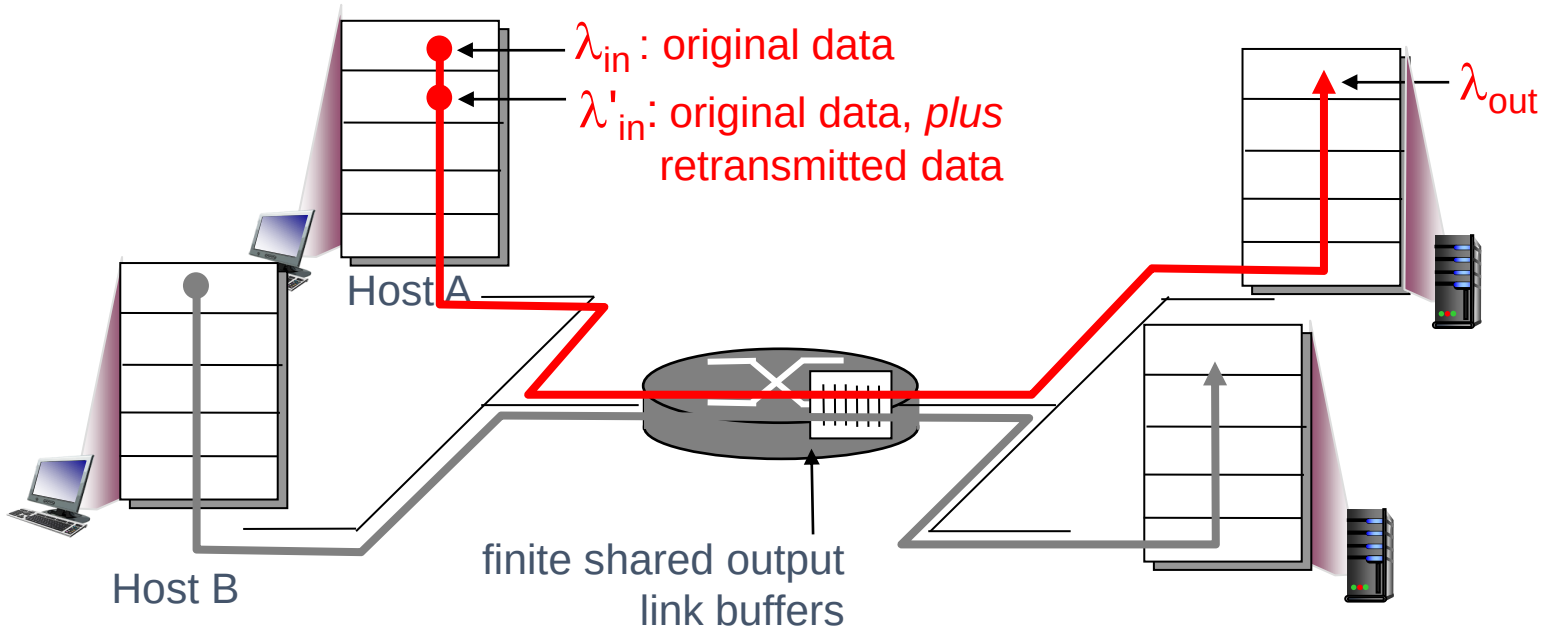


- large delays as arrival rate, λ_{in}, approaches capacity

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* : $\lambda_{in}' \geq \lambda_{in}$

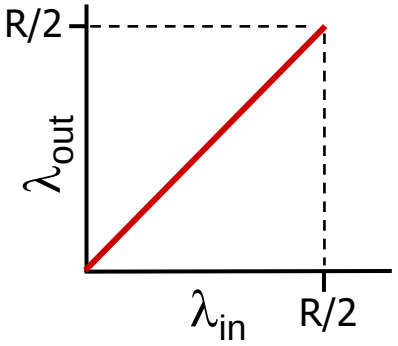
A more realistic case that the sender retransmits only when a packet is known for certain to be lost

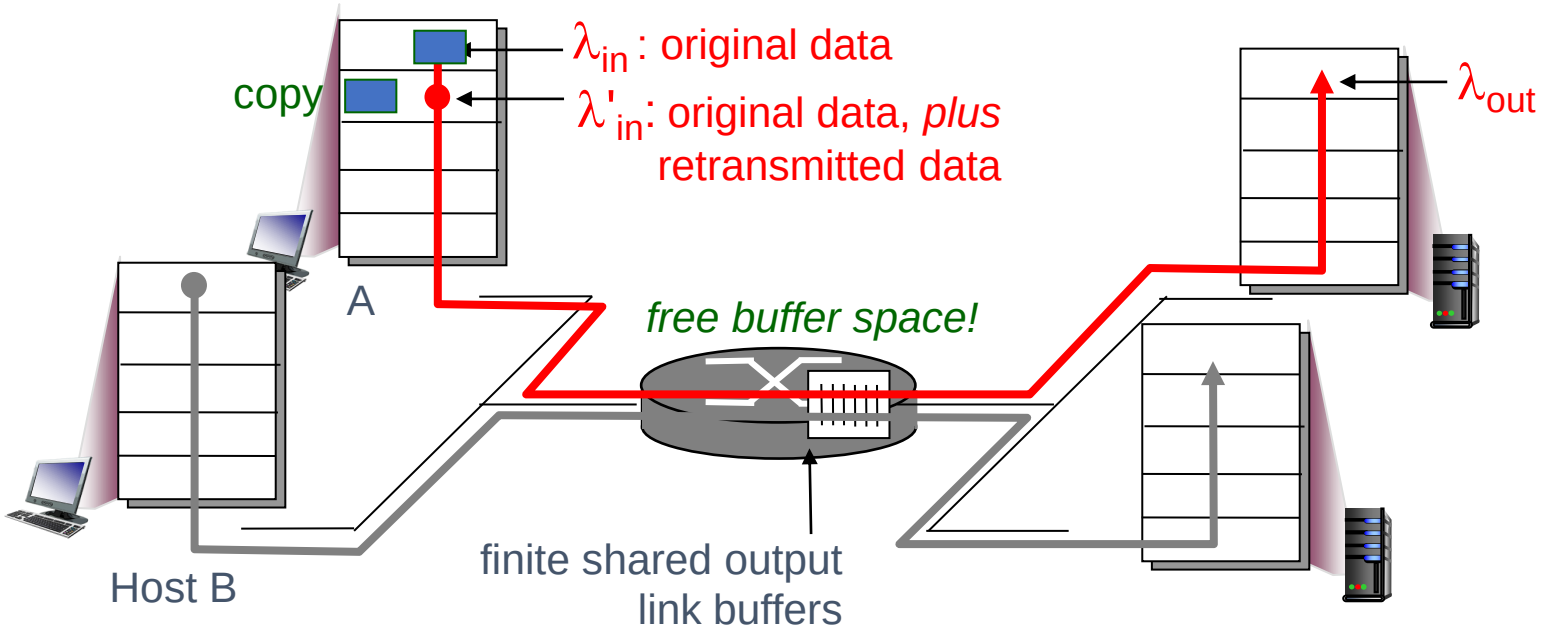


Causes/costs of congestion: scenario 2

idealization: perfect knowledge

- sender sends only when router buffers available

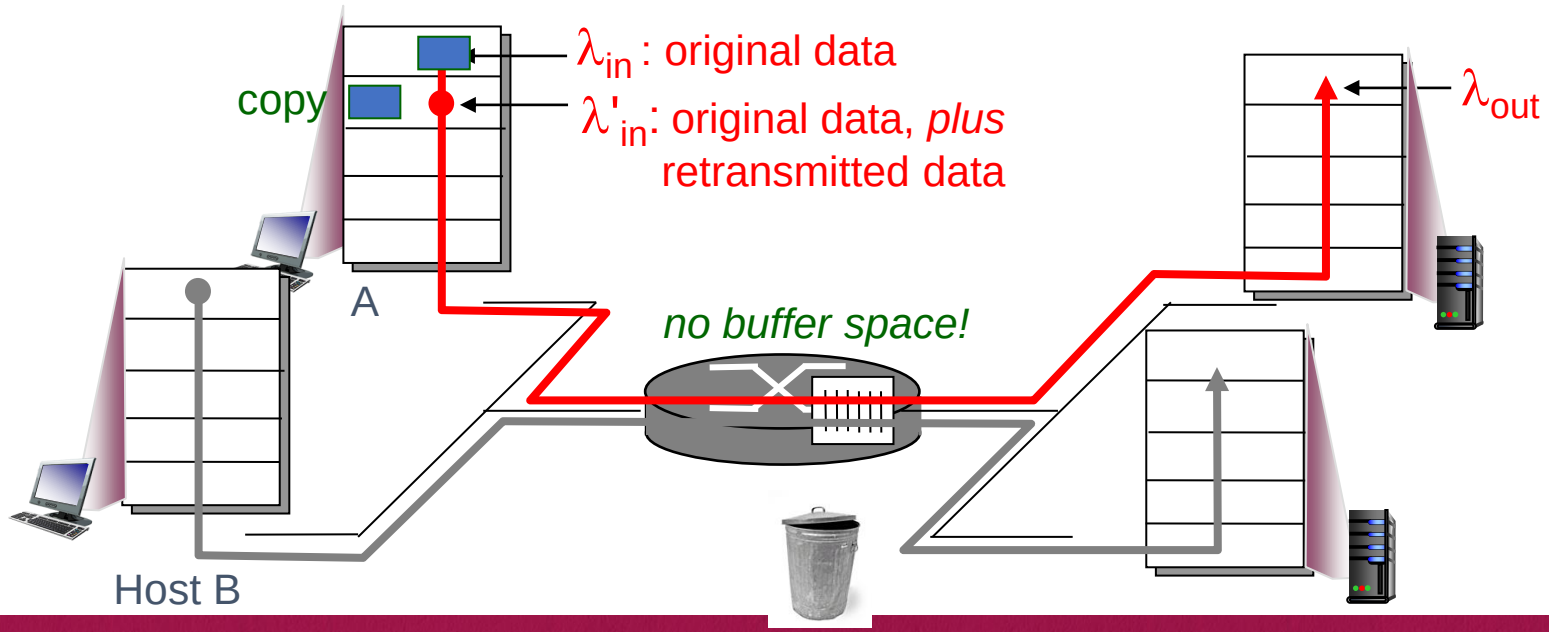




Causes/costs of congestion: scenario 2

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

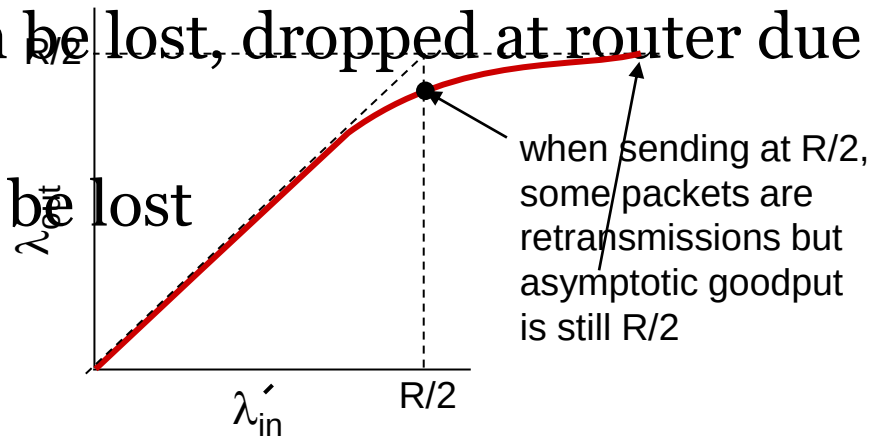
- sender only resends if packet *known* to be lost

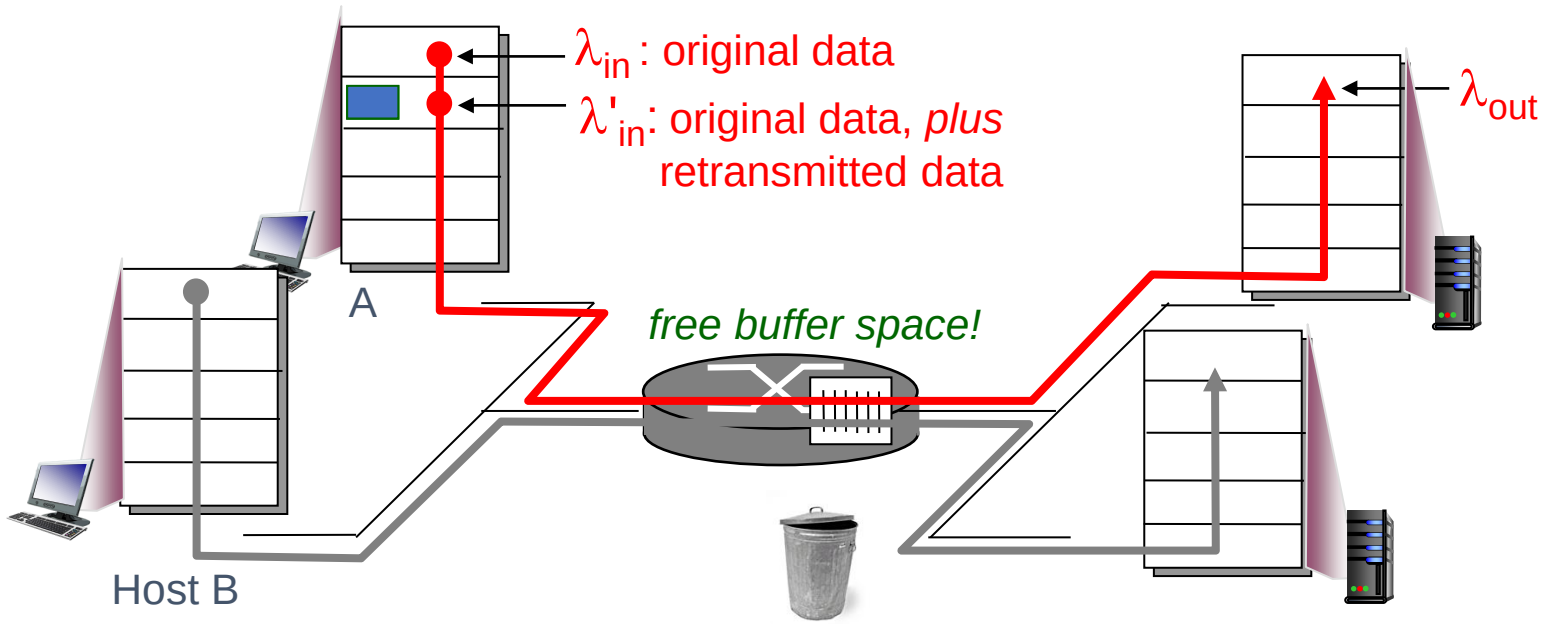


Causes/costs of congestion: scenario 2

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

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

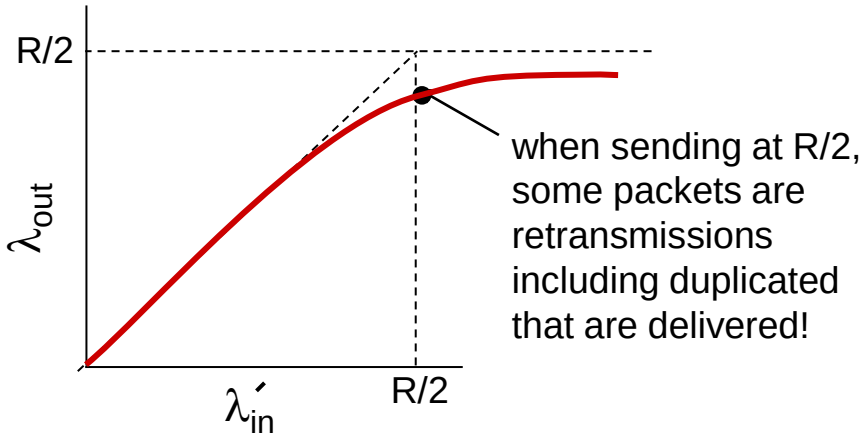


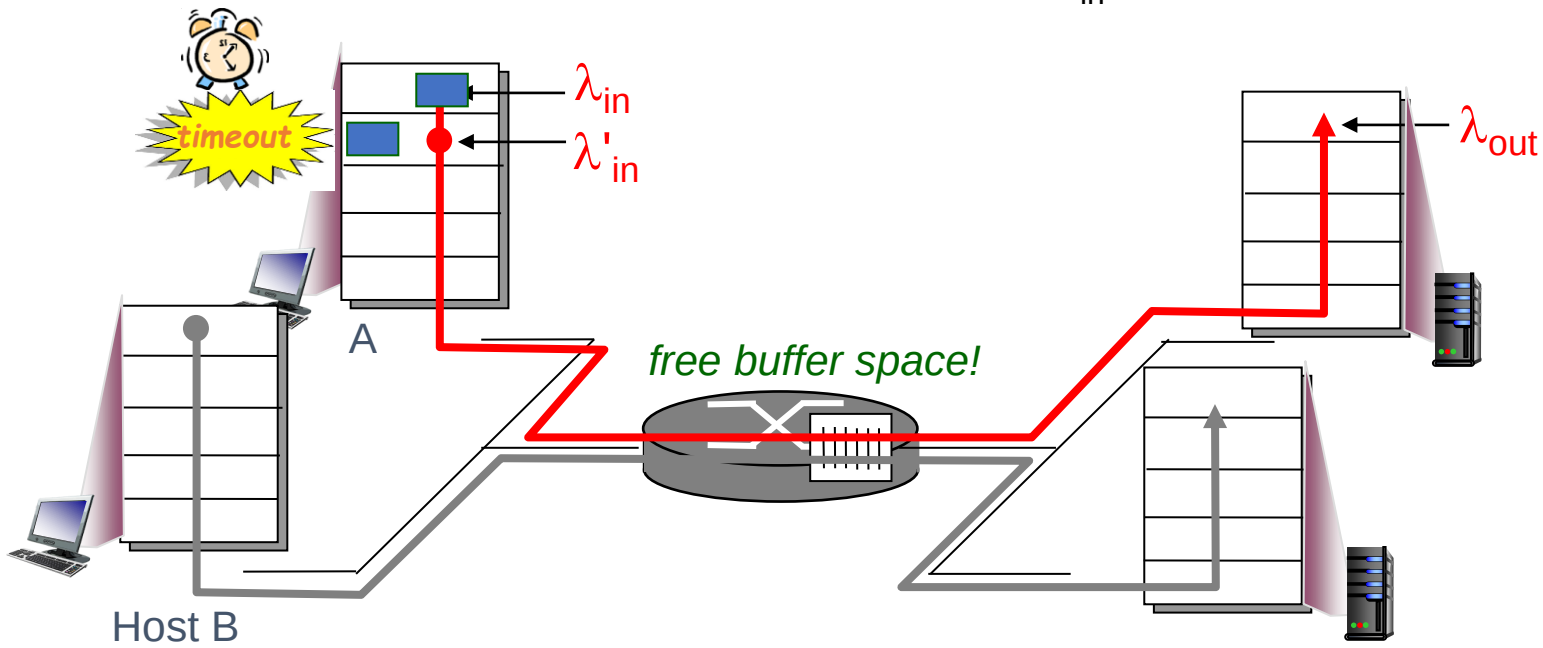


Causes/costs of congestion: scenario 2

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

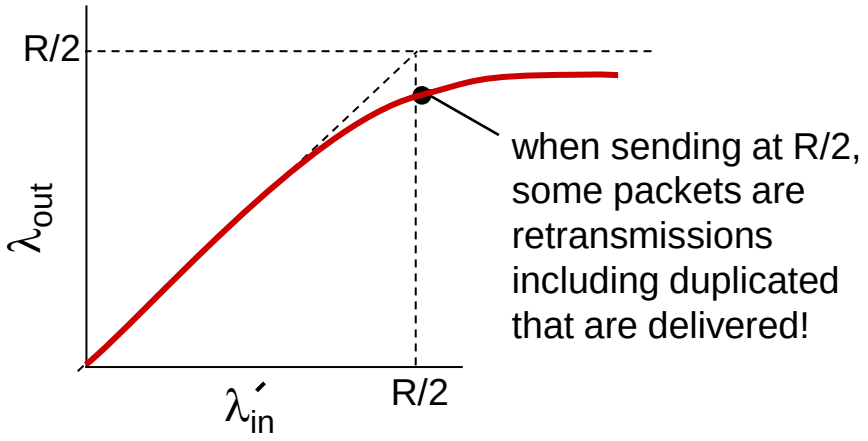




Causes/costs of congestion: scenario 2

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:

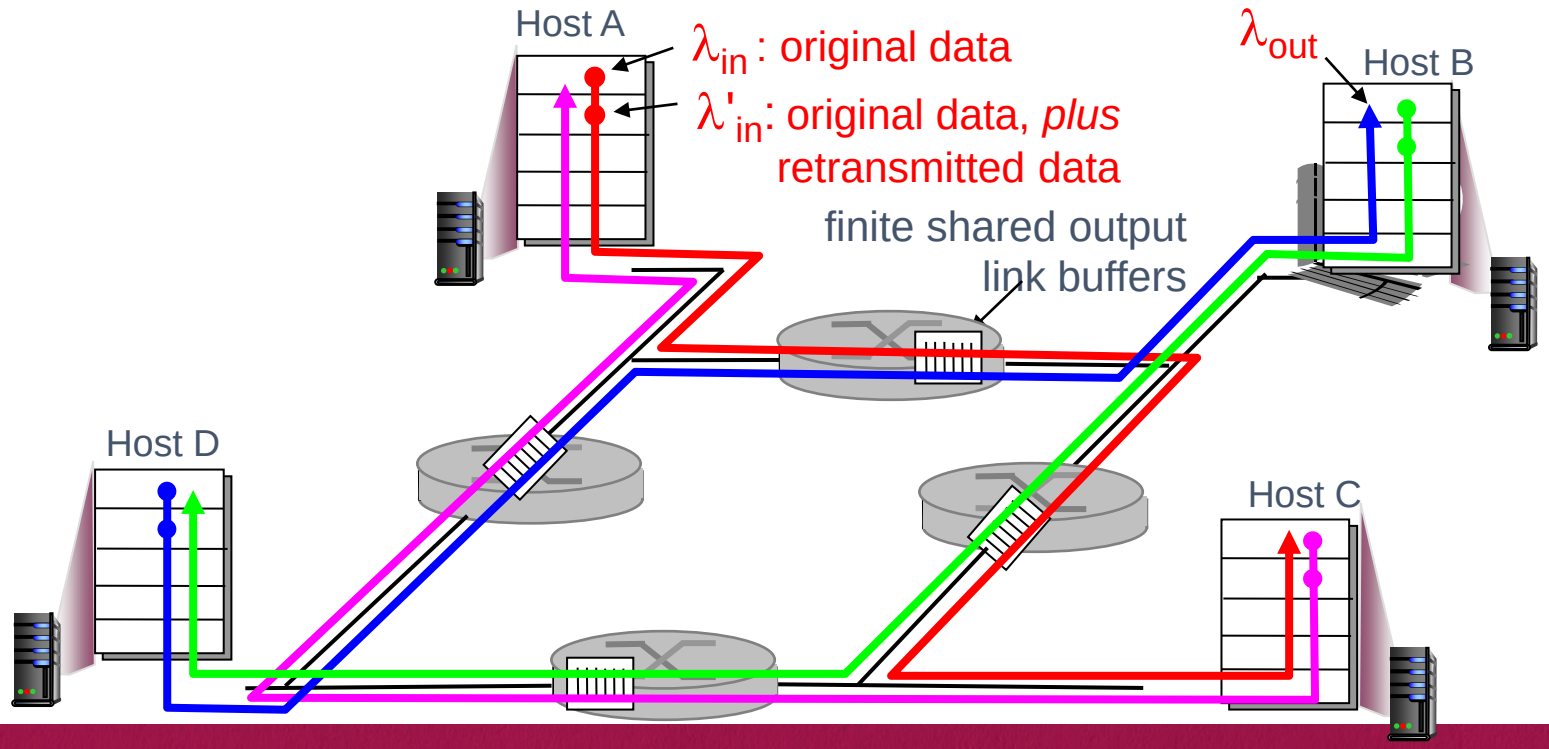
- more work (retrans) for given "goodput"
- unneeded retransmissions: link carries multiple copies of pkt
  - decreasing goodput

Causes/costs of congestion: scenario 3

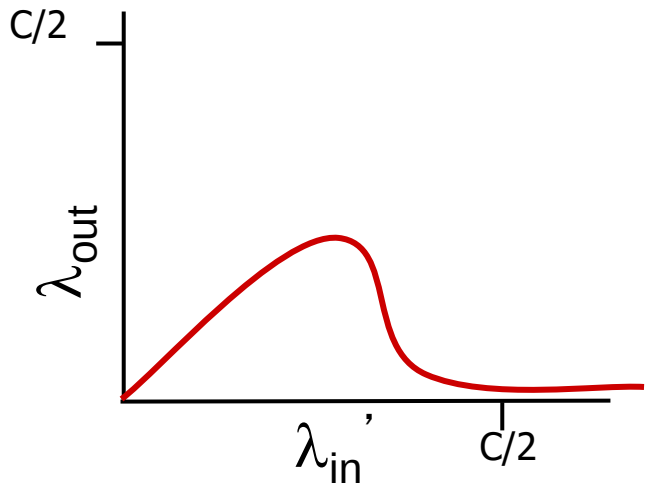
- four senders
- multihop paths
- timeout/retransmit

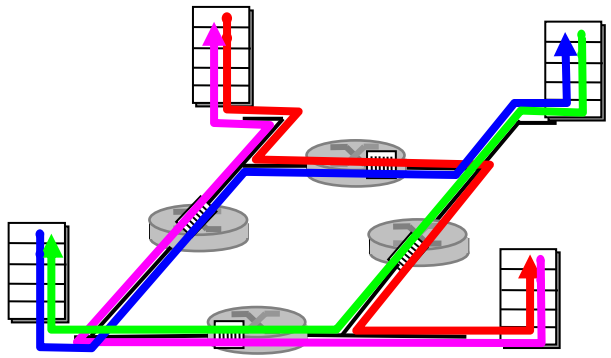
Q: what happens as λ_{in} and λ_{in}' increase ?

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



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

At the broadest level, we can distinguish among congestion-control approaches by whether the network layer provides any explicit assistance to the transport layer for congestion-control purposes

- ***End-to-end congestion control***. In an end-to-end approach to congestion control,
- the network layer provides *no explicit support* to the transport layer for congestion control purposes.
- presence of congestion in the network must be inferred
- Based only on observed network behavior-TCP segment loss (as indicated by a timeout or a triple duplicate acknowledgment) is taken as an indication of network
- congestion and TCP decreases its window size accordingly.
- more recent proposal for TCP congestion control that uses increasing round-tripdelay values as indicators of increased network congestion

- ***Network-assisted congestion control***. With network-assisted congestion control,network-layer components (that is, routers) provide explicit feedback to the sender regarding the congestion state in the network
- used in ATM available bit-rate (ABR) congestion control
- Direct feedback may be sent from a network router to the sender. This form of notification typically takes the form of a **choke packet**
- Upon receipt of a marked packet, the receiver then notifies the sender of the congestion indication.
- Note that this latter form of notification takes at least a full round-trip time

Network-Assisted Congestion-Control Example: ATM ABR Congestion Control

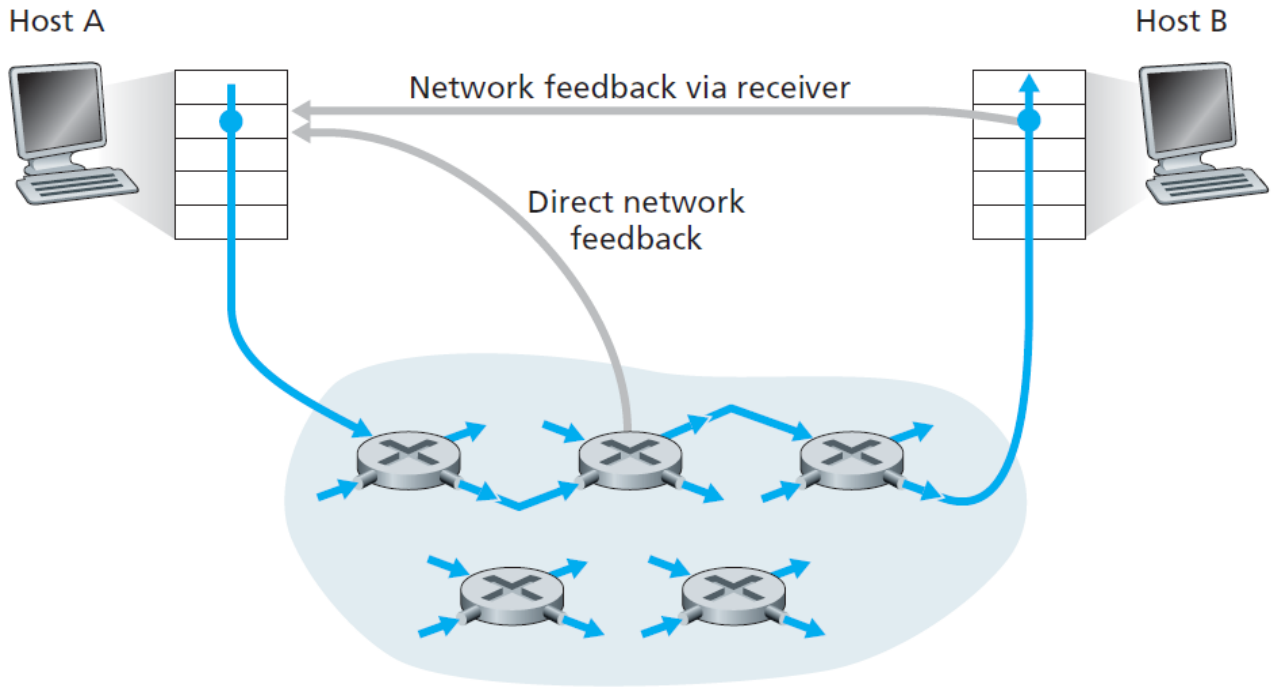
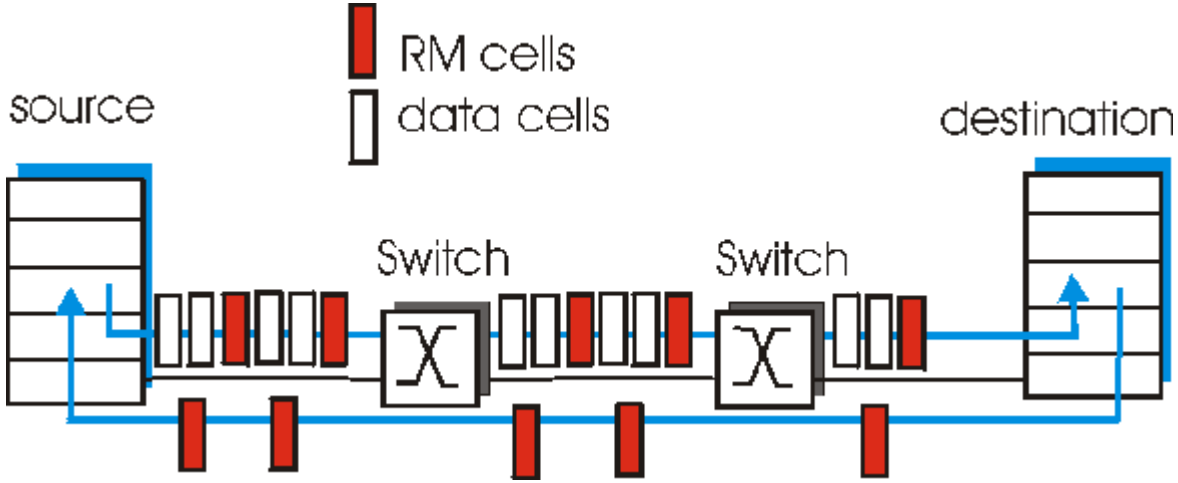


Figure 3.49 ♦ Two feedback pathways for network-indicated congestion information

ATM ABR Congestion Control

- With ATM ABR service, data cells are transmitted from a source to a destination through a series of intermediate switches. Interspersed with the data cells are **resource-management cells (RM cells)**; these RM cells can be used to convey congestion-related information among the hosts and switches. When an RM cell arrives at a destination, it will be turned around and sent back to the sender (possibly after the destination has modified the contents of the RM cell). It is also possible for a switch to generate an RM cell itself and send this RM cell directly to a source. RM cells can thus be used to provide both direct network feedback and network feedback via the receiver,
- ATM ABR congestion control is a rate-based approach. That is, the sender explicitly computes a maximum rate at which it can send and regulates itself accordingly. ABR provides three mechanisms for signaling congestion-related information from the switches to the receiver



ATM ABR Congestion Control

- **EFCI bit**. Each data cell contains an EFCI (Explicit Forward Congestion Indication) bit. A congested network switch can set the EFCI bit in a data cell to 1 to signal congestion to the destination host. The destination must check the EFCI bit in all received data cells. When an RM cell arrives at the destination, if the most recently-received data cell had the EFCI bit set to 1, then the destination sets the CI (Congestion Indication) bit of the RM cell to 1 and sends the RM cell back to the sender. Using the EFCI in data cells and the CI bit in RM cells, a sender can thus be notified about congestion at a network switch.

- **CI and NI bits**. As noted above, sender-to-receiver RM cells are interpersed with data cells. The rate of RM cell interspersion is a tunable parameter, with one RM cell every 32 data cells being the default value. These RM cells have a CI bit and a NI (No Increase) bit that can be set by a congested network switch. Specifically, a switch can set the NI bit in a passing RM cell to1 under mild congestion and can set the CI bit to 1 under severe congestion conditions. When a destination host receives an RM cell, it will send the RM cell back to the sender with its CI and NI bits intact (except that CI may be set to 1 by the destination as a result of the EFCI mechanism decribed above).

- **Explicit Rate (ER) setting**. Each RM cell also contains a 2-byte ER (Explicit Rate) field. A congested switch may lower the value contained in the ER field in a passing RM cell. In this manner, the ER field will be set to the minimum supportable rate of all switches on the source-to-destination path.

TCP Congestion Control

- TCP provides a reliable transport service between two processes running on different hosts. Another key component of TCP is its congestion-control mechanism. TCP must use end-to-end congestion control rather than network- assisted congestion control, since the IP layer provides no explicit feedback to the end systems regarding network congestion

- The approach taken by TCP is to have each sender limit the rate at which it sends traffic into its connection as a function of perceived network congestion. If a TCP sender perceives that there is little congestion on the path between itself and the destination, then the TCP sender increases its send rate; if the sender perceives that there is congestion along the path, then the sender reduces its send rate

How does a TCP sender limit the rate at which it sends traffic into its connection?
How does a TCP sender perceive that there is congestion on the path between itself and the destination?
what algorithm should the sender use to change its send rate as a function of perceived end-to-end congestion?

- **Flow control window**: Advertised Window (RWND)
  - How many bytes can be sent without overflowing receivers buffers
  - Determined by the receiver and reported to the sender
- **Congestion Window** (CWND)
  - How many bytes can be sent without overflowing routers
  - Computed by the sender using congestion control algorithm
- **Sender-side window** = minimum{CWND,RWND}
  - Assume for this lecture that RWND >> CWND
  - **no explicit signaling of congestion state by the network—**
  - **ACKs and loss events serve as implicit signals**

TCP- **self-clocking**.

- If acknowledgments arrive at a relatively slow rate (e.g., if the end-end path has high delay or contains a low-bandwidth link), then the congestion window will be increased at a relatively slow rate. On the other hand, if acknowledgments arrive at a high rate, then the congestion window will be increased more quickly.As TCP use acknowledgments to trigger (or clock) its increase in congestion window size, TCP is said to be **self-clocking**.

- ***How should a TCP sender determine the rate at which it should send?***
- ***How then do the TCP senders determine their sending rates such that they don't congest the network but at the same time make use of all the available bandwidth?***
- ***Are TCP senders explicitly coordinated, or is there a distributed approach in which the TCP senders can set their sending rates based only on local information?***

TCP answers these questions using the following guiding principles:

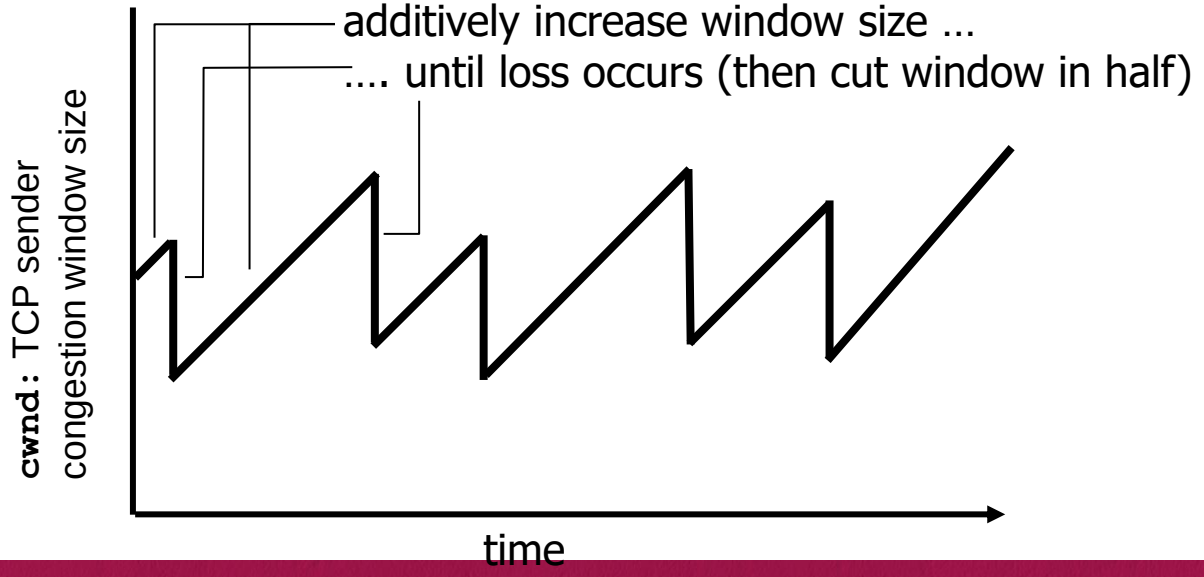
TCP- **self-clocking**

- ***A lost segment implies congestion, and hence, the TCP sender's rate should be decreased when a segment is lost.***
  - *A* timeout event or the receipt of four acknowledgments for a given segment (one original ACK and then three duplicate ACKs) is interpreted as an implicit "loss event" indication of the segment following the quadruply ACKed segment, triggering a retransmission of the lost segment. From a congestion control standpoint, the question is how the TCP sender should **decrease its congestion window size**, and hence its sending rate, in response to this inferred loss event.
- ***An acknowledged segment indicates that the network is delivering the sender's segments to the receiver, and hence, the sender's rate can be increased when an ACK arrives for a previously unacknowledged segment.***
  - The arrival of acknowledgments is taken as an implicit indication that all is well—segments are being successfully delivered from sender to receiver, and the network is thus not congested. The **congestion window size can thus be increased**
- ***Bandwidth probing.***
  - Given ACKs indicating a congestion-free source-to-destination. path and loss events indicating a congested path, TCP's strategy for adjusting its transmission rate is to increase its rate in response to arriving ACKs until a loss event occurs, at which point, the transmission rate is decreased. The TCP sender thus increases its transmission rate to ponset begins, backs off from that rate, and then to begins probing again to see if the congestion onset rate has changed.

TCP congestion control: (AIMD) 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

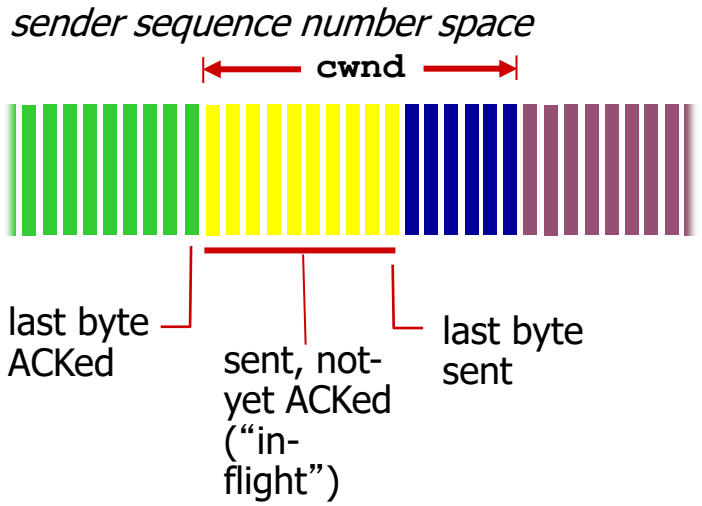
AIMD saw tooth behavior: probing for bandwidth



TCP Congestion Control: details

- sender limits transmission:

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



LastByteSent-LastByteAcked ≤ **cwnd**

TCP sending rate:

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

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

Sender's send rate is roughly cwnd/RTT bytes/sec. By adjusting the value of `cwnd`, the sender can therefore adjust the rate at which it sends data into its connection.

TCP congestion-control algorithm

- The algorithm has three major components:
  - (1) slow start,
  - (2) congestion avoidance,
  - (3) fast recovery

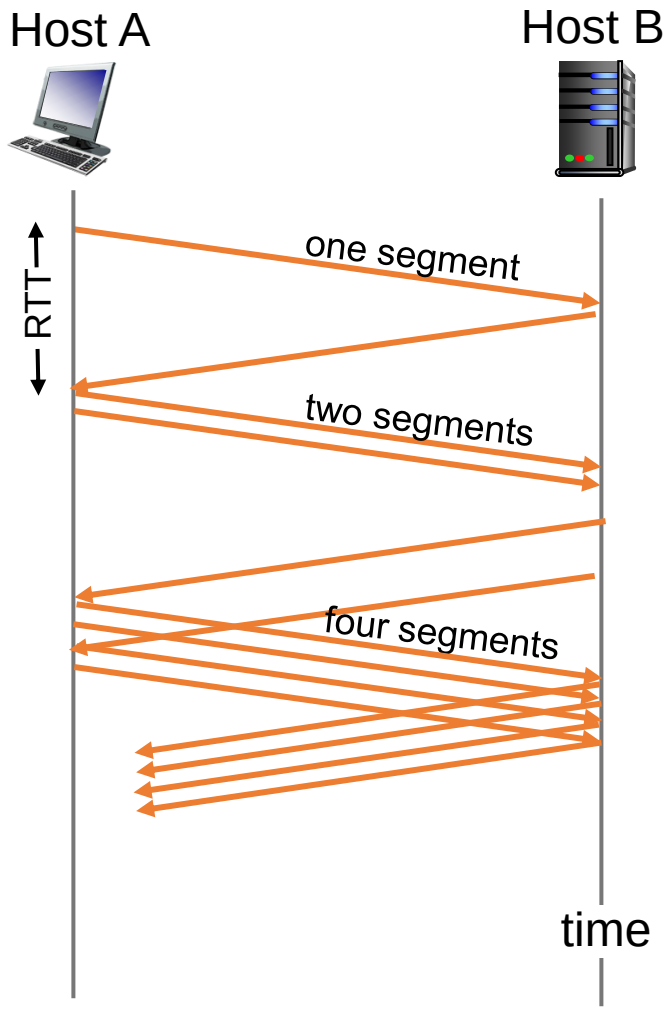
TCP Slow Start

- 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

But when should this exponential growth end?



But when should this exponential growth end?

- First, if there is a loss event (i.e., congestion) indicated by a timeout, the TCP sender sets the value of cwnd to 1 and begins the slow start process anew.

- It also sets the value of a second state variable, ssthresh (shorthand for “slow start threshold”) to cwnd/2—half of the value of the congestion window value when congestion was detected. The second way in which slow start may end is directly tied to the value of ssthresh. Since ssthresh is half the value of cwnd when congestion was last detected, it might be a bit reckless to keep doubling cwnd when it reaches or surpasses the value of ssthresh. Thus, when the value of cwnd equals ssthresh, slow start ends and TCP transitions into congestion avoidance mode

- The final way in which slow start can end is if three duplicate ACKs are detected, in which case TCP performs a fast retransmit

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)

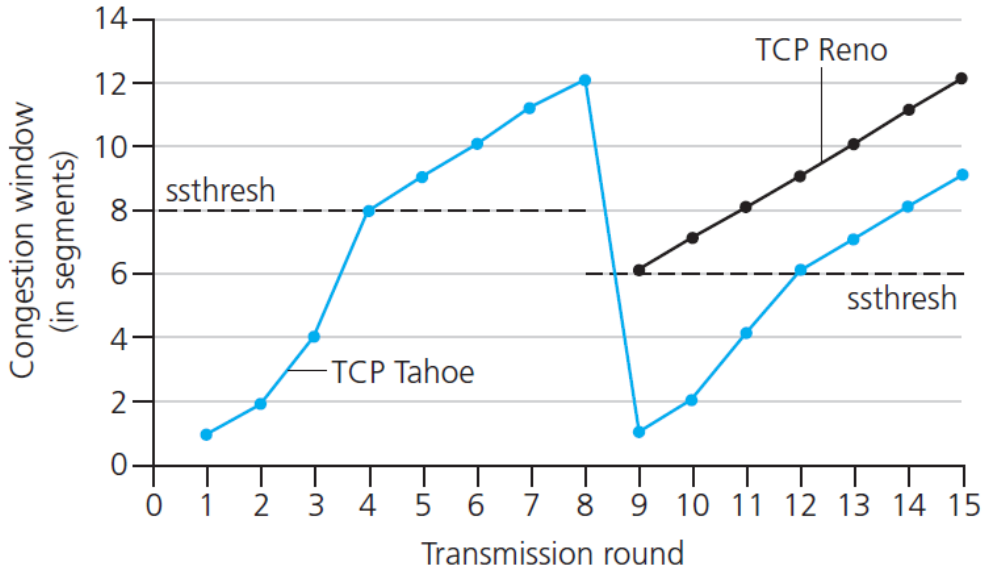
TCP: switching from slow start to CA

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



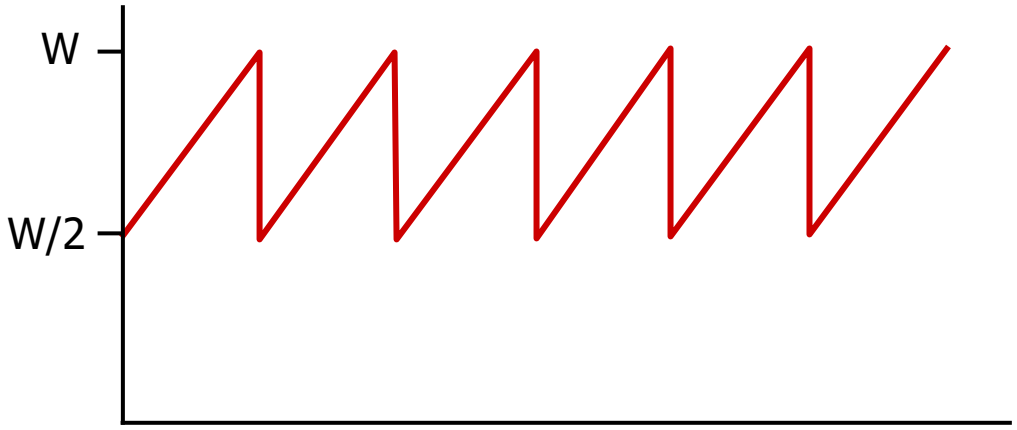
In fast recovery, the value of cwnd is increased by 1 MSS for every duplicate ACK received for the missing segment that caused TCP to enter the fast-recovery state.

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

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]:

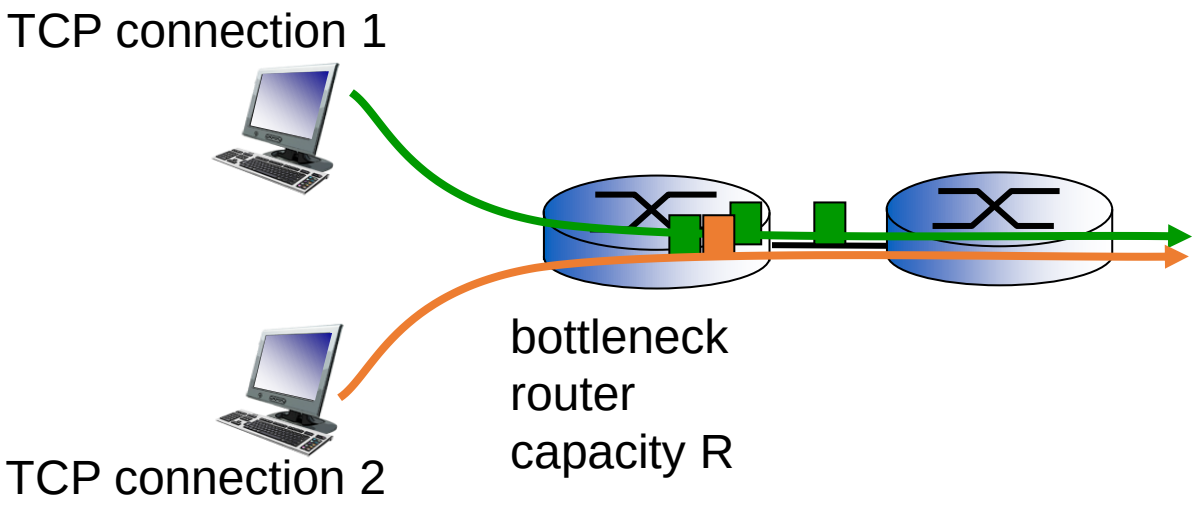
$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT}\sqrt{L}}$$

➜ 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

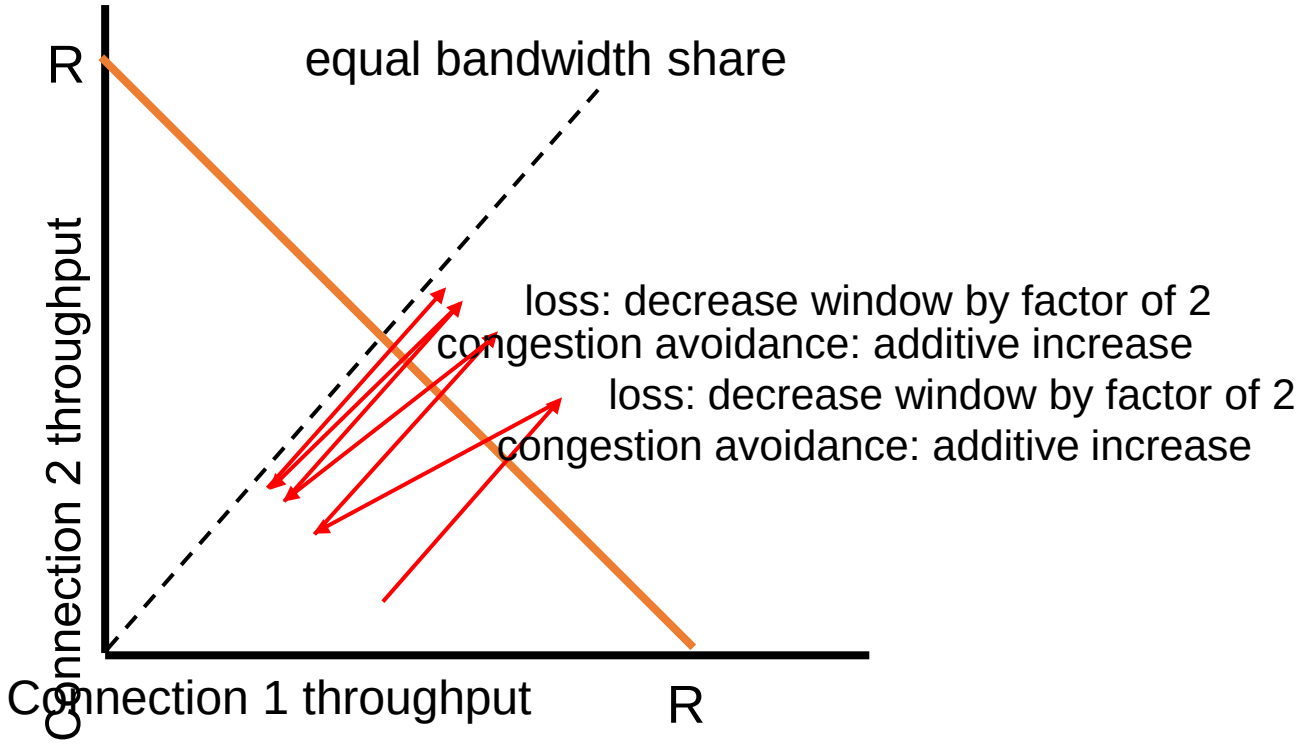
fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

two competing sessions:

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



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

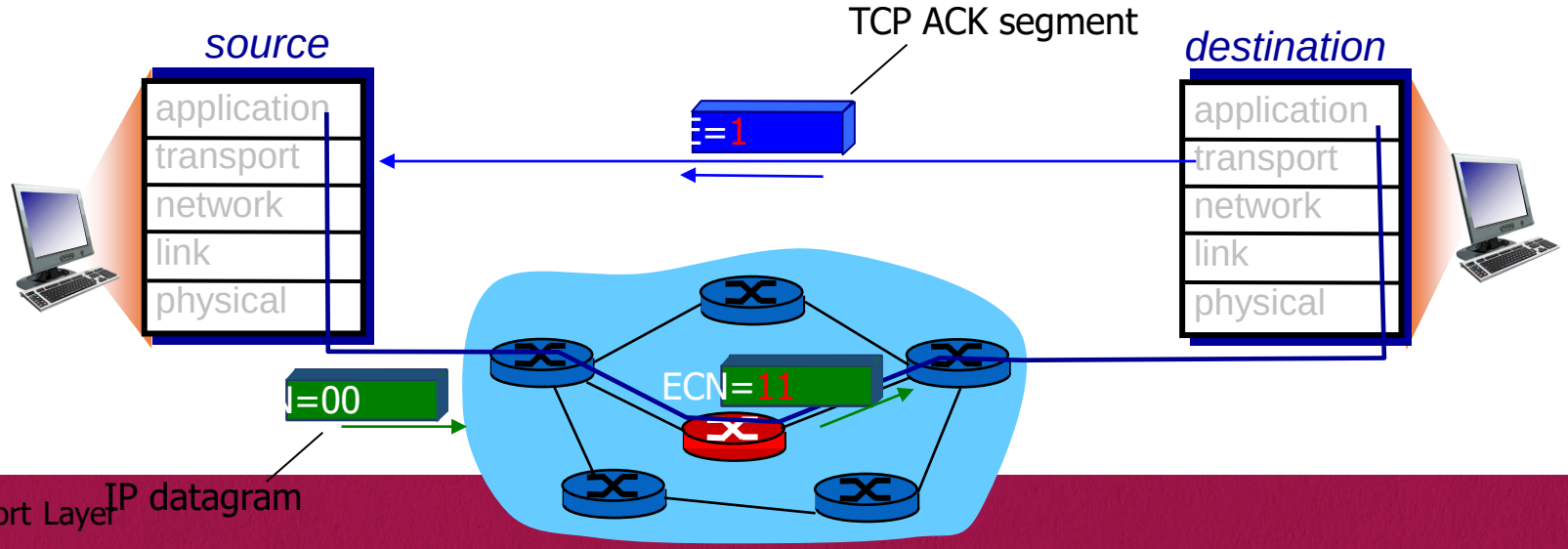
Fairness, parallel TCP connections

- 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

Namah Shivaya