Last time

- UDP socket programming
  - DatagramSocket, DatagramPacket
- TCP
  - Sequence numbers, ACKs
  - RTT, DevRTT, timeout calculations
  - Reliable data transfer algorithm

This time

- TCP
  - Fast retransmit
  - Flow control
  - Connection management
  - Congestion control

Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires

Fast retransmit algorithm:

```
event: ACK received, with ACK field value of y
          if (y > SendBase) {
              SendBase = y
              if (there are currently not-yet-acknowledged segments)
                  start timer
            }
          else {
               increment count of dup ACKs received for y
               if (count of dup ACKs received for y == 3) {
                   resend segment with sequence number y
                }
```

a duplicate ACK for
already ACKed segment

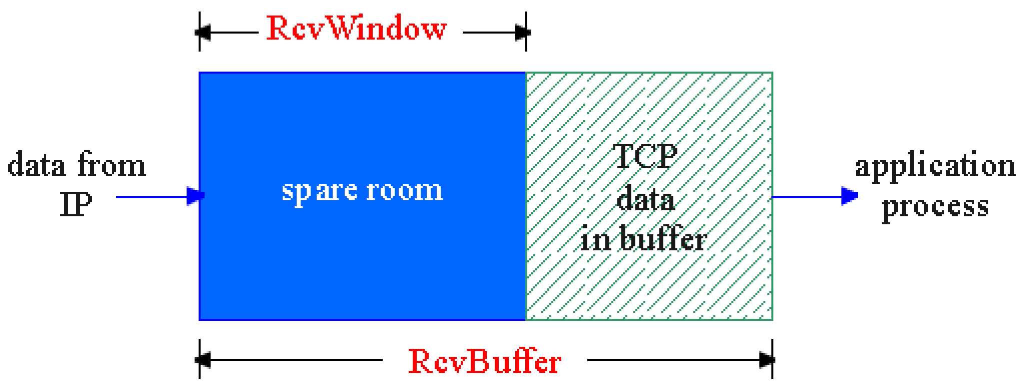
fast retransmit

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 Flow Control

- Receive side of TCP connection has a receive buffer:



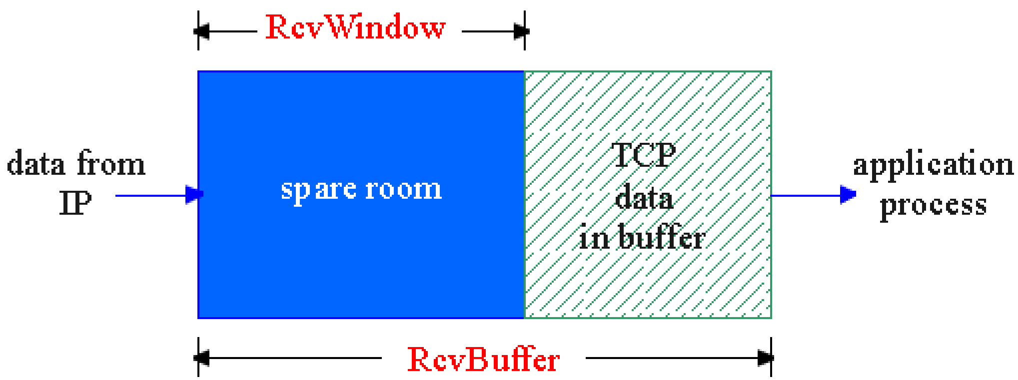
- App process may be slow at reading from buffer

flow control

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

- Speed-matching service: matching the send rate to the receiving app's drain rate

TCP Flow control: how it works



(Suppose TCP receiver discards out-of-order segments)

- spare room in buffer

= RcvWindow

= RcvBuffer-[LastByteRcvd - LastByteRead]

- Rcvr advertises spare room by including value of **RcvWindow** in segments
- Sender limits unACKed data to **RcvWindow**
  - guarantees receive buffer doesn't overflow

See the applet in UW-ACE!

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 Connection Management

Recall: TCP sender, receiver establish "connection" before exchanging data segments

- initialize TCP variables:
  - seq. #s
  - buffers, flow control info (e.g. **`RcvWindow`**)
- *client:* connection initiator

```
Socket clientSocket = new
 Socket("hostname","port

 number");
```

- *server:* contacted by client

```
Socket connectionSocket =
 welcomeSocket.accept();
```

Three way handshake:

Step 1: client host sends TCP SYN segment to server
- specifies initial seq #
- no data

Step 2: server host receives SYN, replies with SYNACK segment
- server allocates buffers
- specifies server initial seq. #

Step 3: client receives SYNACK, replies with ACK segment, which may contain data

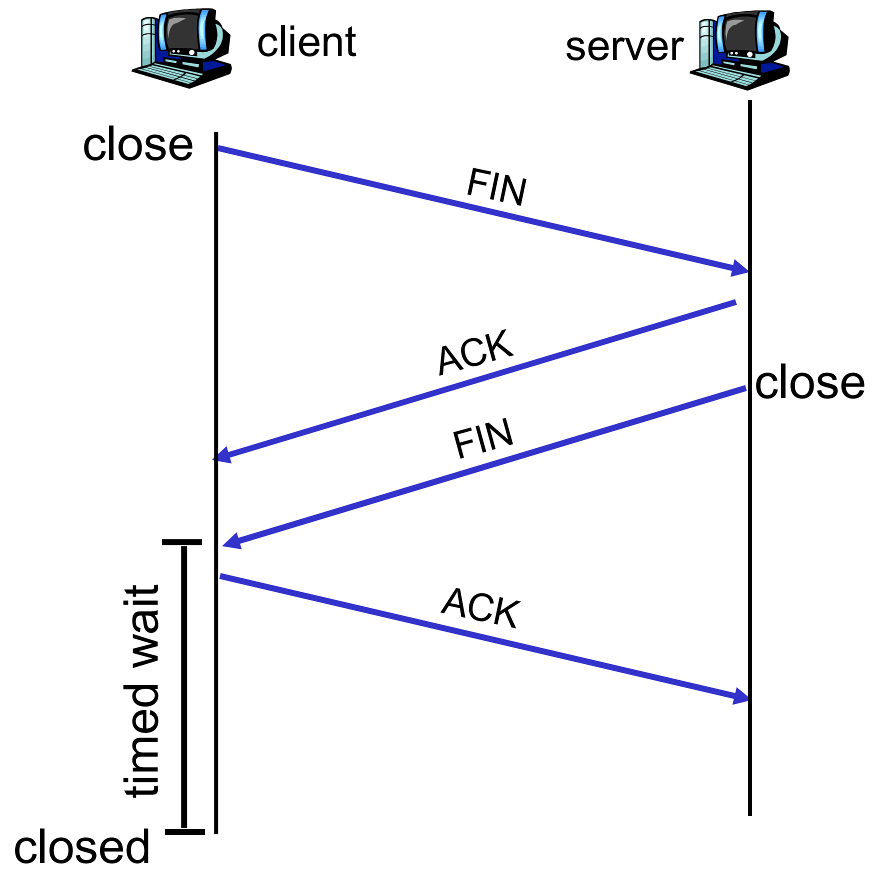
TCP Connection Management (cont.)

Closing a connection:

client closes socket:
```
clientSocket.close();
```

Step 1: client end system sends TCP FIN control segment to server.

Step 2: server receives FIN, replies with ACK. Closes connection, sends FIN.



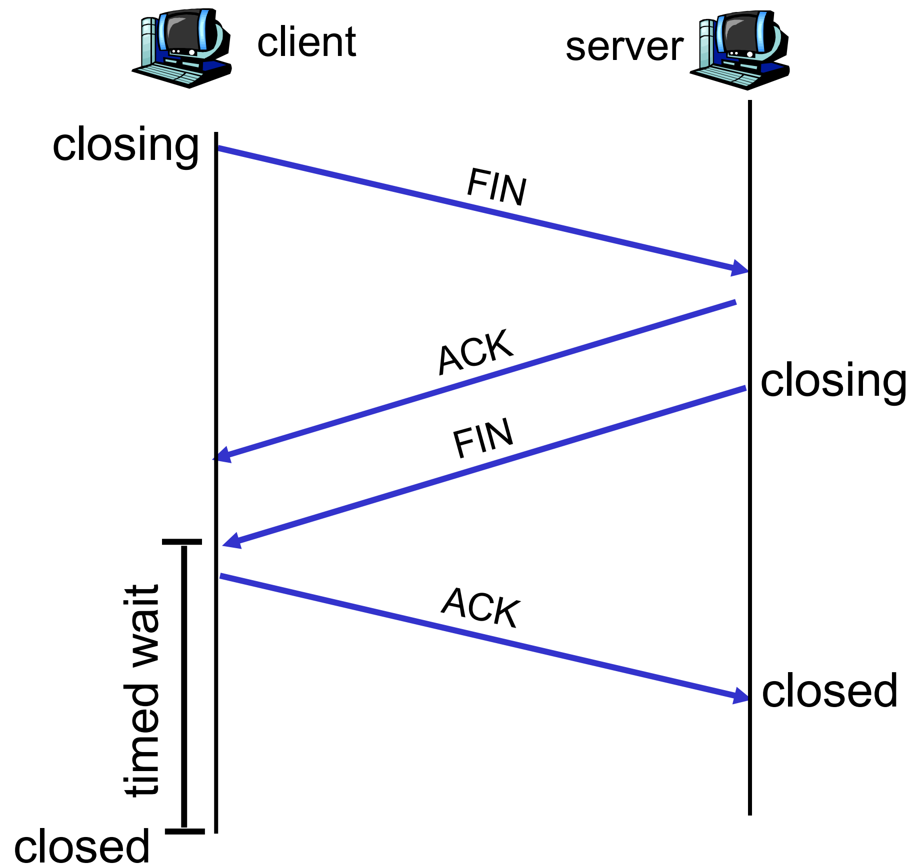
TCP Connection Management (cont.)

Step 3: client receives FIN, replies with ACK.

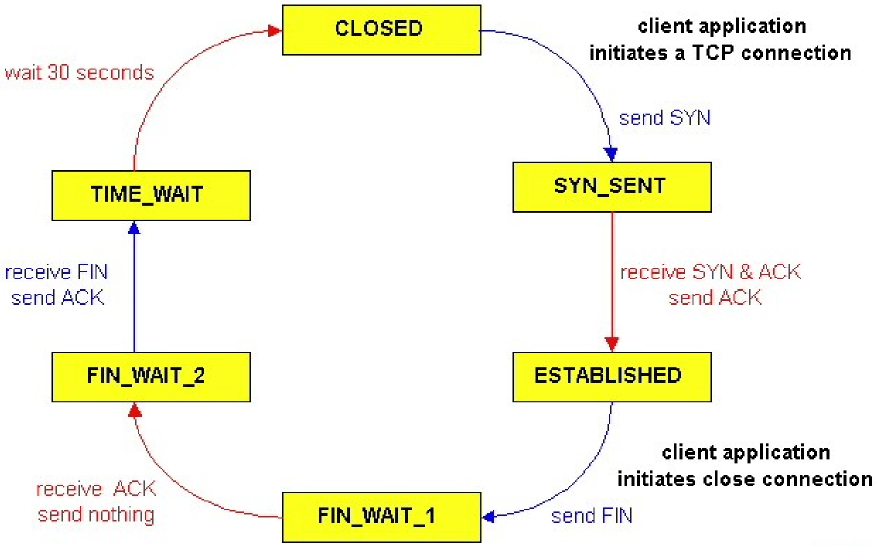
- Enters "timed wait" - will respond with ACK to received FINs

Step 4: server, receives ACK. Connection closed.

Note: with small modification, can handle simultaneous FINs.

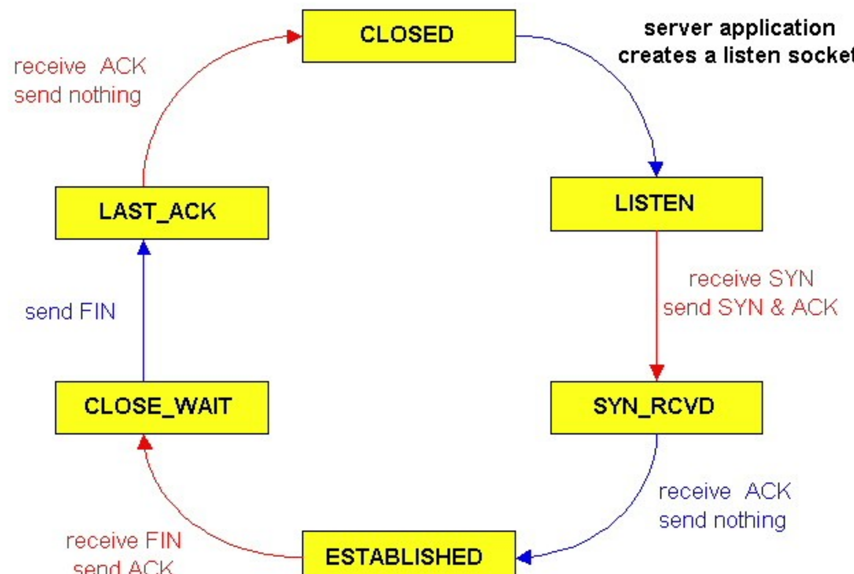


TCP Connection Management (cont)



TCP client lifecycle

TCP server lifecycle



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

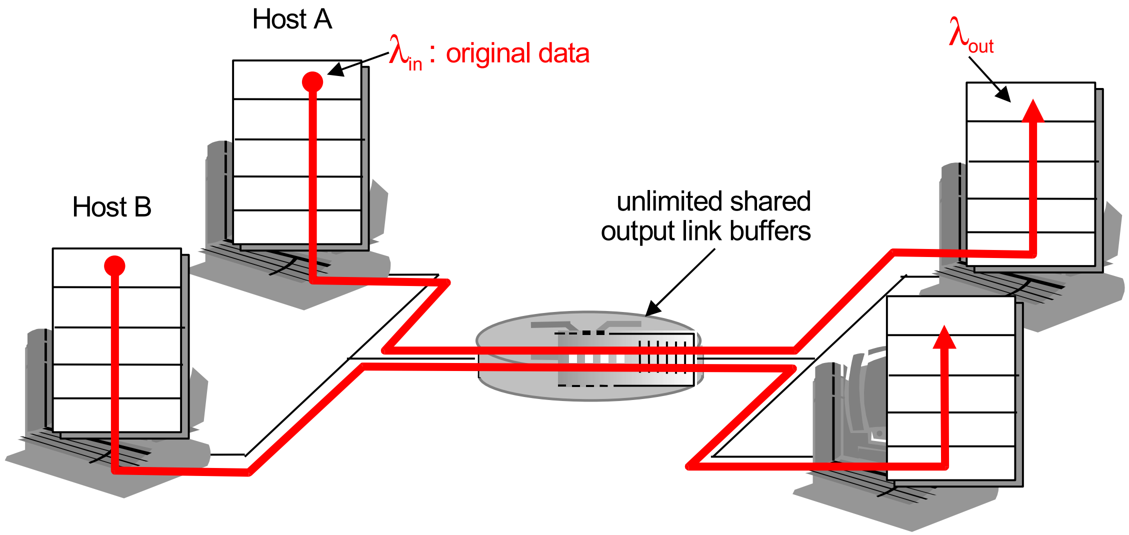
Principles of Congestion Control

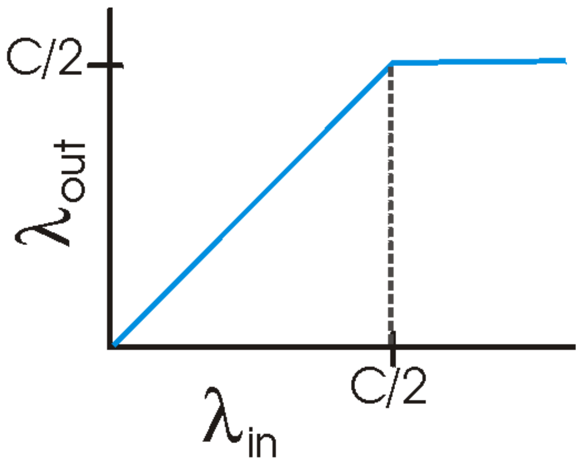
Congestion:

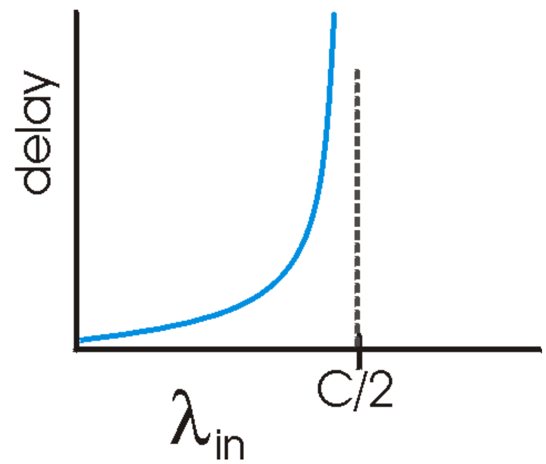
- informally: “too many sources sending too much data too fast for *network* to handle”
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

Causes/costs of congestion: scenario 1

- Two senders, two receivers
- One router, infinite buffers
- No retransmission



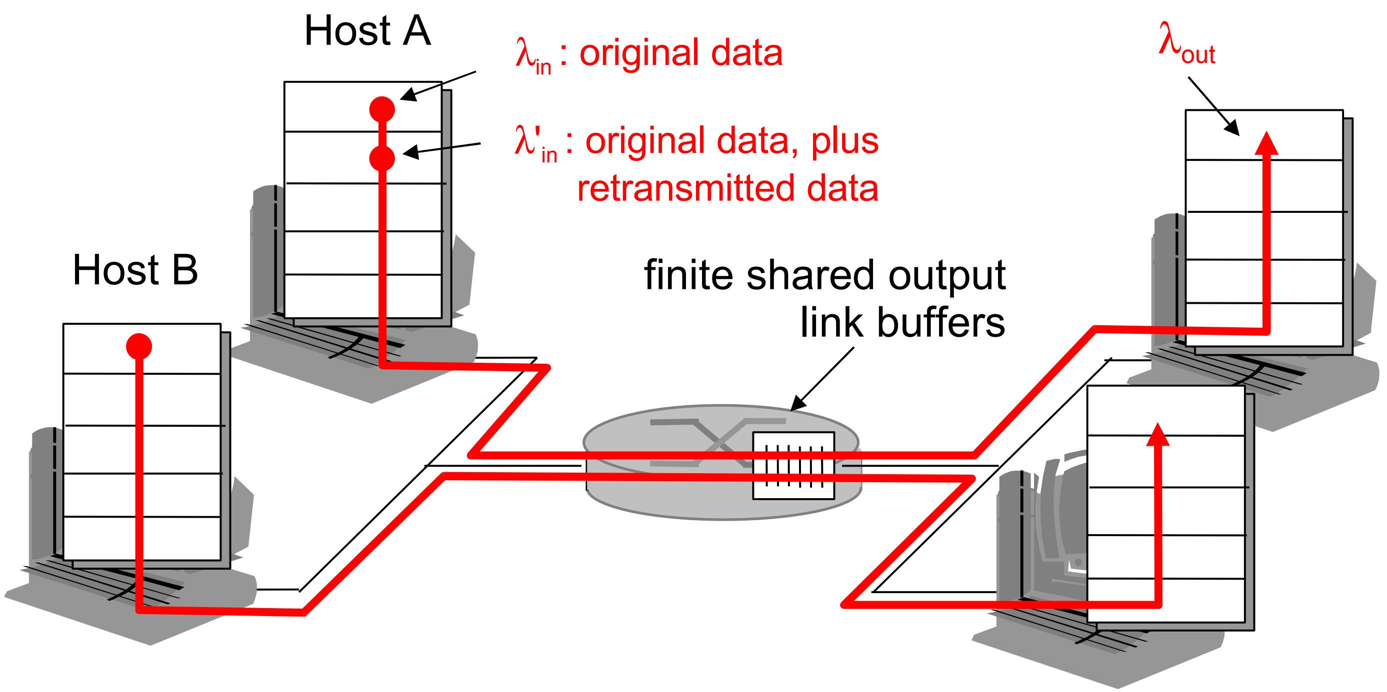




- Large delays when congested
- Maximum achievable throughput

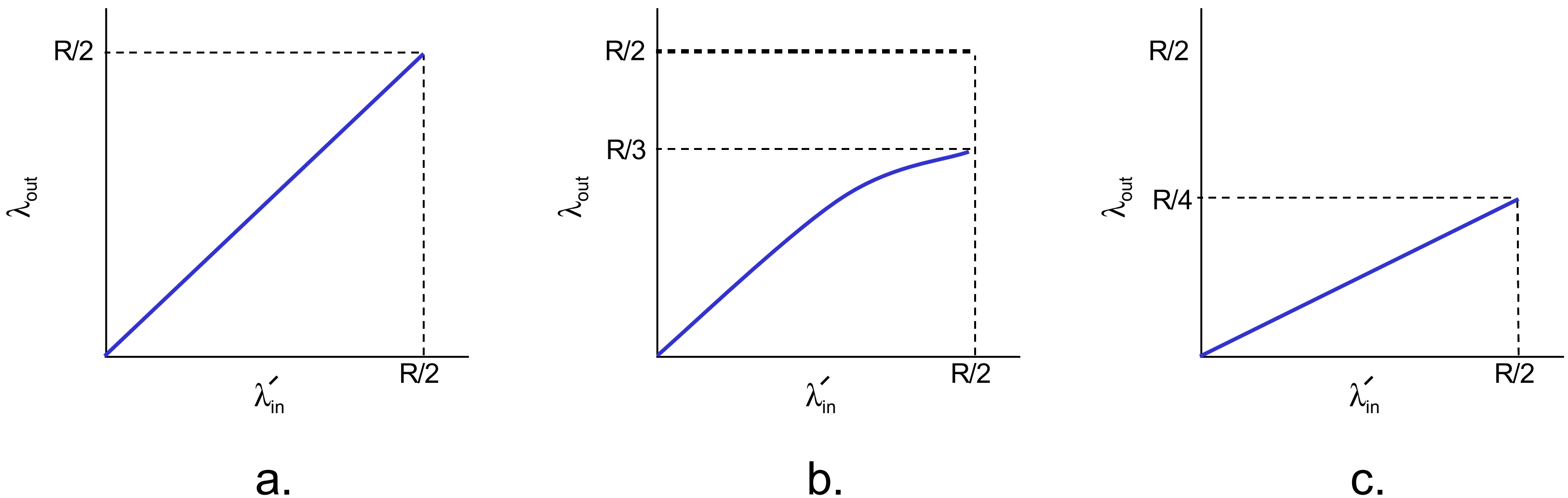
Causes/costs of congestion: scenario 2

- One router, *finite* buffers
- Sender retransmission of lost packet



Causes/costs of congestion: scenario 2

- Always: $\lambda_{in} = \lambda_{out}$ (goodput)
- "Perfect" retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- Retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out}



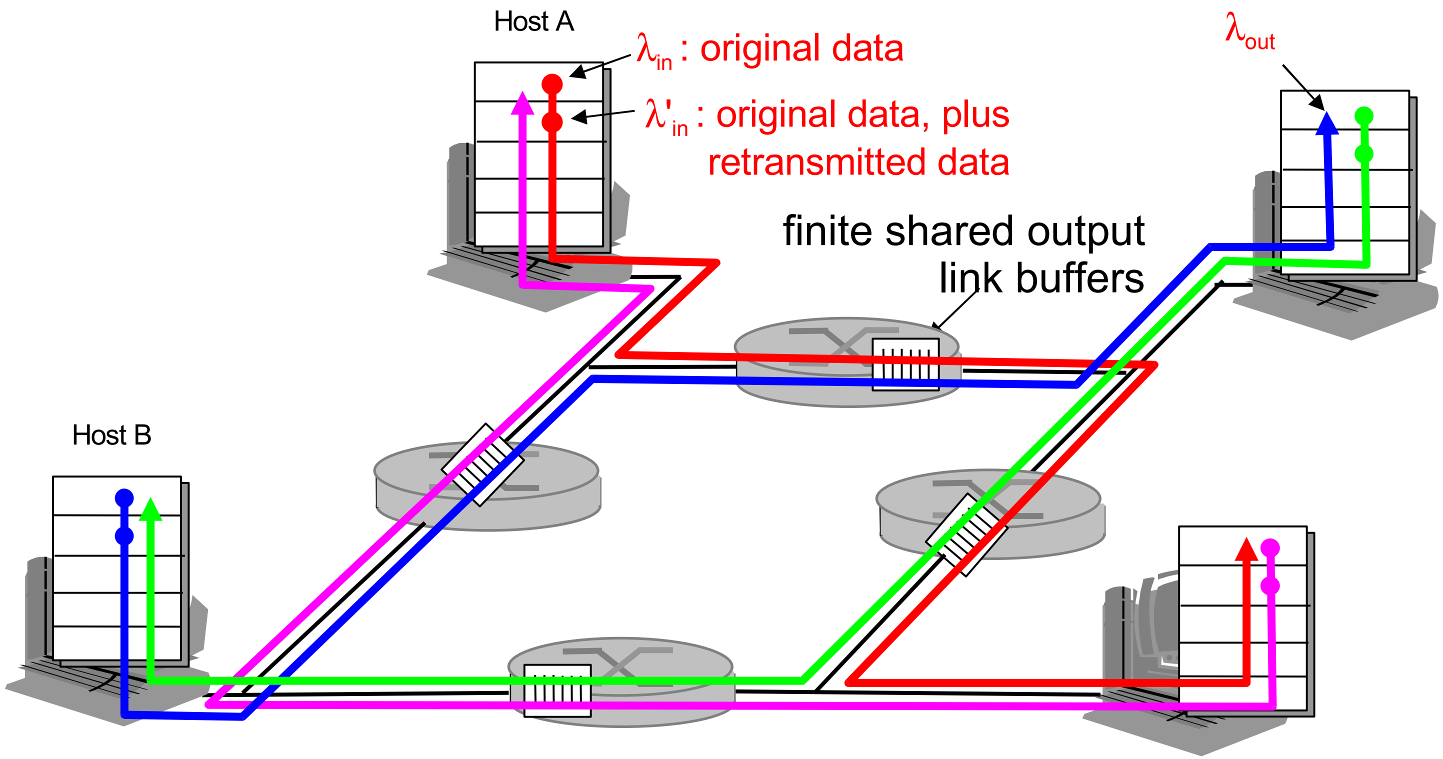
"Costs" of congestion:

- More work (retransmissions) for given "goodput"
- Unneeded retransmissions: link carries multiple copies of packet

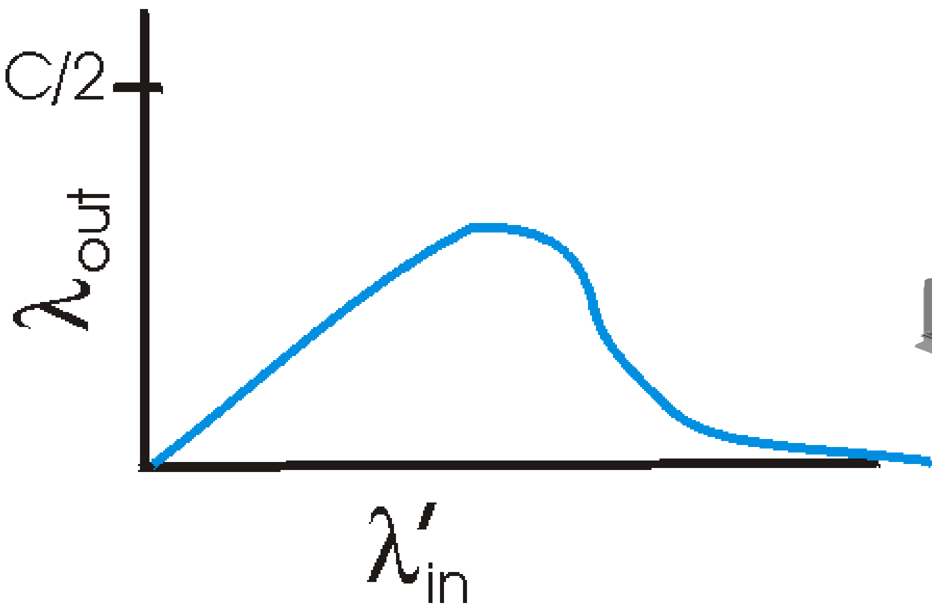
Causes/costs of congestion: scenario 3

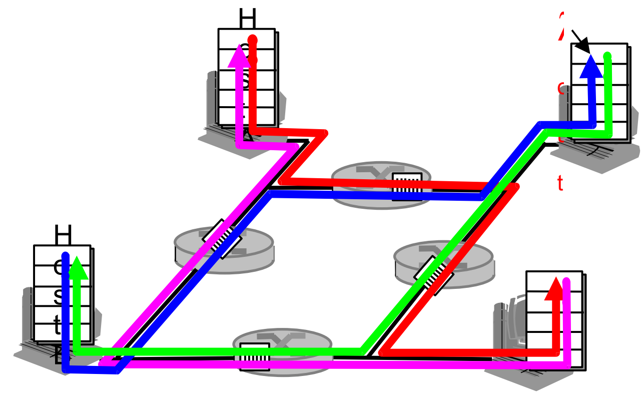
- Four senders
- Multihop paths
- Timeout/retransmit

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



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 sender should 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 limits transmission:

  `LastByteSent-LastByteAcked`
  $\leq$ `CongWin`

- Roughly,

  $$\boxed{\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}}$$

- **`CongWin`** is dynamic, function of perceived network congestion

How does sender perceive congestion?

- Loss event = timeout *or* 3 duplicate acks
- TCP sender reduces rate (**`CongWin`**) after loss event

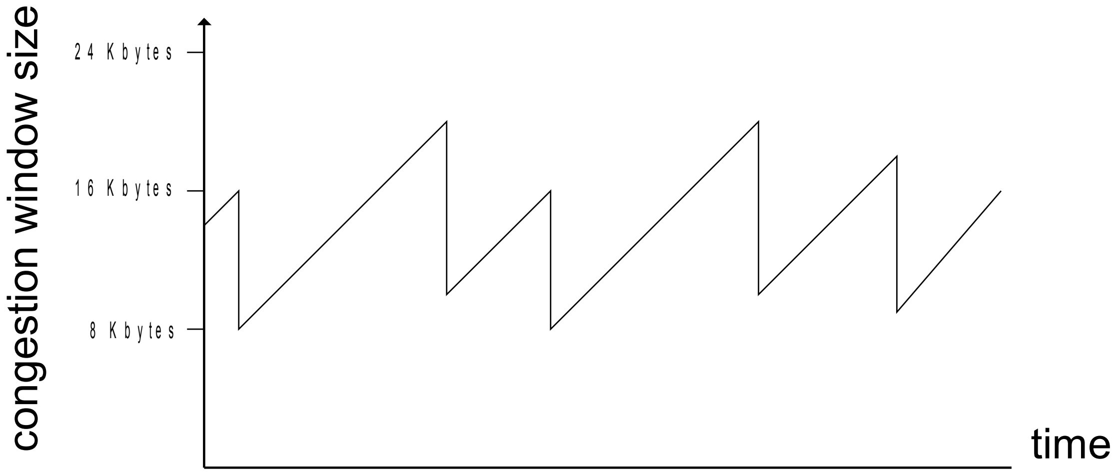
three mechanisms:

- AIMD
- slow start
- conservative after timeout events

TCP congestion control: additive increase, multiplicative decrease

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

Saw tooth behavior: probing for bandwidth

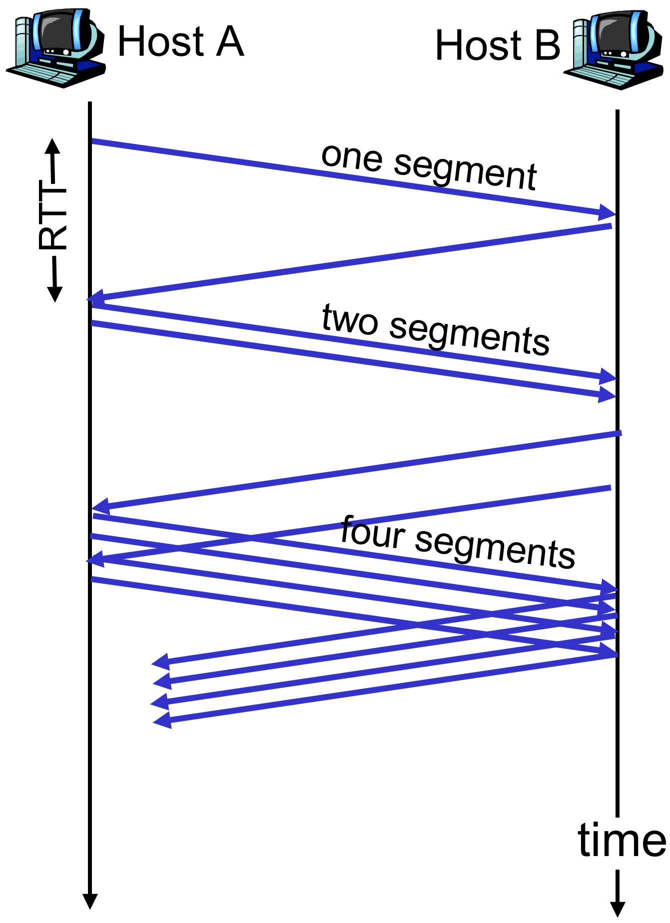


TCP Slow Start

- When connection begins, `CongWin` = 1 MSS
  - Example: MSS = 1 kBytes & RTT = 200 msec
  - initial rate = 40 kbps
- available bandwidth may be >> MSS/RTT
  - desirable to quickly ramp up to respectable rate
- When connection begins, increase rate exponentially fast until first loss event

TCP Slow Start (more)

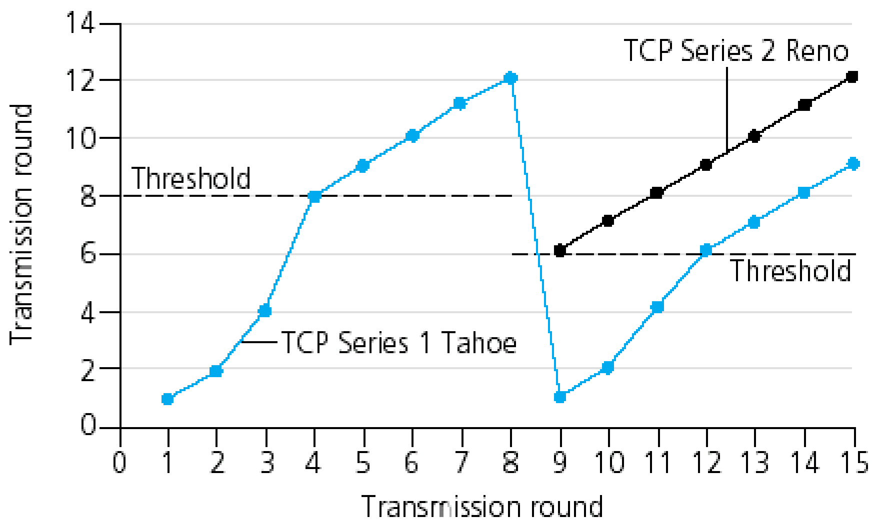
- When connection begins, increase rate exponentially until first loss event:
  - double **`CongWin`** every RTT
  - done by incrementing **`CongWin`** for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast



Refinement

Q: When should the exponential increase switch to linear?

A: When **`CongWin`** gets to 1/2 of its value before timeout.



Implementation:

- Variable Threshold
- At loss event, Threshold is set to 1/2 of CongWin just before loss event

Refinement: inferring loss

- After 3 dup ACKs:
  - **`CongWin`** is cut in half
  - window then grows linearly
- But after timeout event:
  - **`CongWin`** instead set to 1 MSS;
  - window then grows exponentially
  - to a threshold, then grows linearly

Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- Timeout indicates a "more alarming" congestion scenario

Summary: TCP Congestion Control

- When **`CongWin`** is below **`Threshold`**, sender in slow-start phase, window grows exponentially.
- When **`CongWin`** is above **`Threshold`**, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, **`Threshold`** set to **`CongWin/2`** and **`CongWin`** set to **`Threshold`**.
- When timeout occurs, **`Threshold`** set to **`CongWin/2`** and **`CongWin`** is set to 1 MSS.

TCP sender congestion control

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to “Congestion Avoidance”	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to “Congestion Avoidance”	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to “Slow Start”	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

Recap

- Fast retransmit
  - 3 duplicate ACKs
- Flow control
  - Receiver windows
- Connection management
  - SYN/SYNACK/ACK, FIN/ACK, TCP states
- Congestion control
  - General concepts
- TCP congestion control
  - AIMD, slow start, congestion avoidance

Next time

- TCP
  - Throughput
  - Fairness
  - Delay modeling
- TCP socket programming
- NAT