

CSCI 491-01

Topics: Internet Programming

Fall 2008

Transport Layer

Derek Leonard

Hendrix College

October 20, 2008

Chapter 3: Roadmap

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

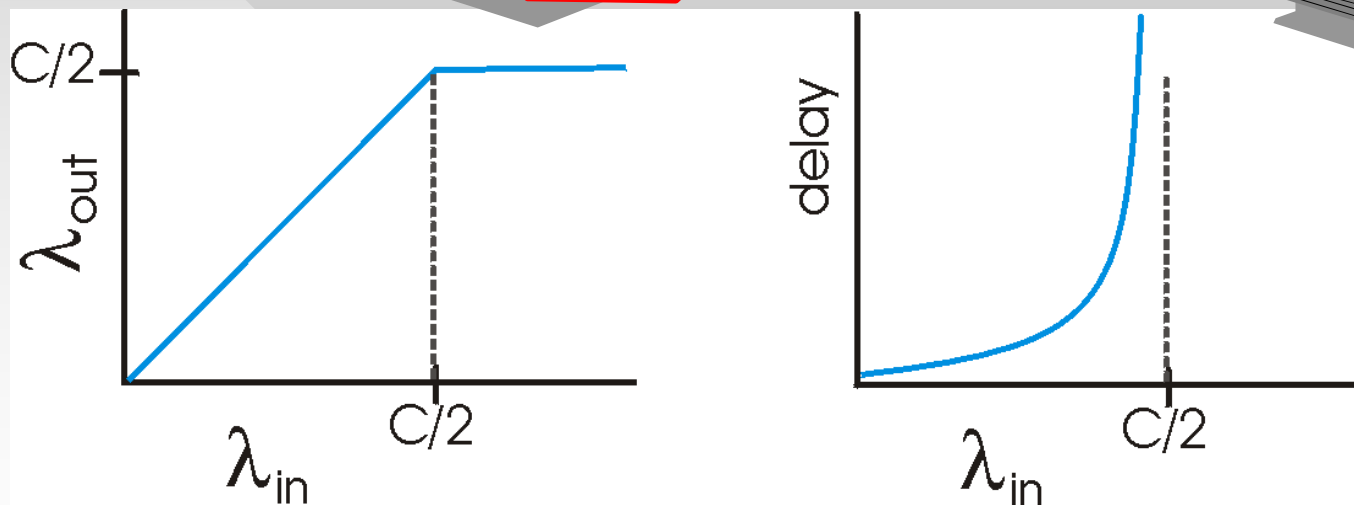
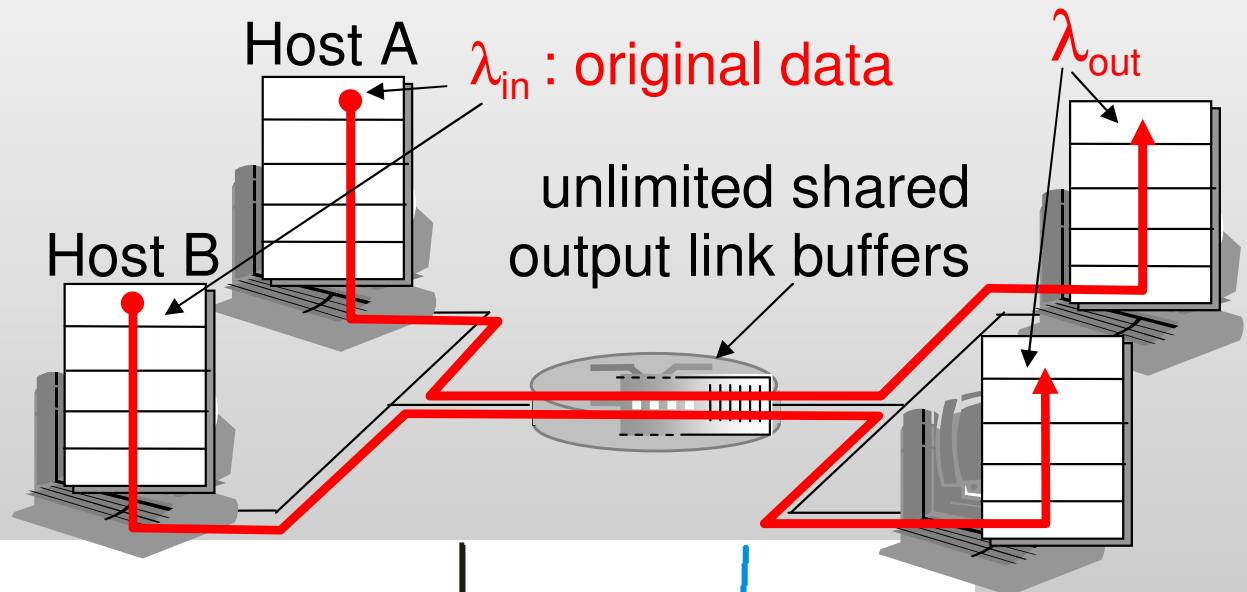
Principles of Congestion Control

Congestion:

- Informally: “too many sources sending too much data too fast for the *network* to handle”
- Different from flow control!
- Manifestations:
 - Lost packets (buffer overflow at routers)
 - Long delays (queueing in router buffers)
- Very important networking problem

Causes/costs of Congestion: Scenario 1

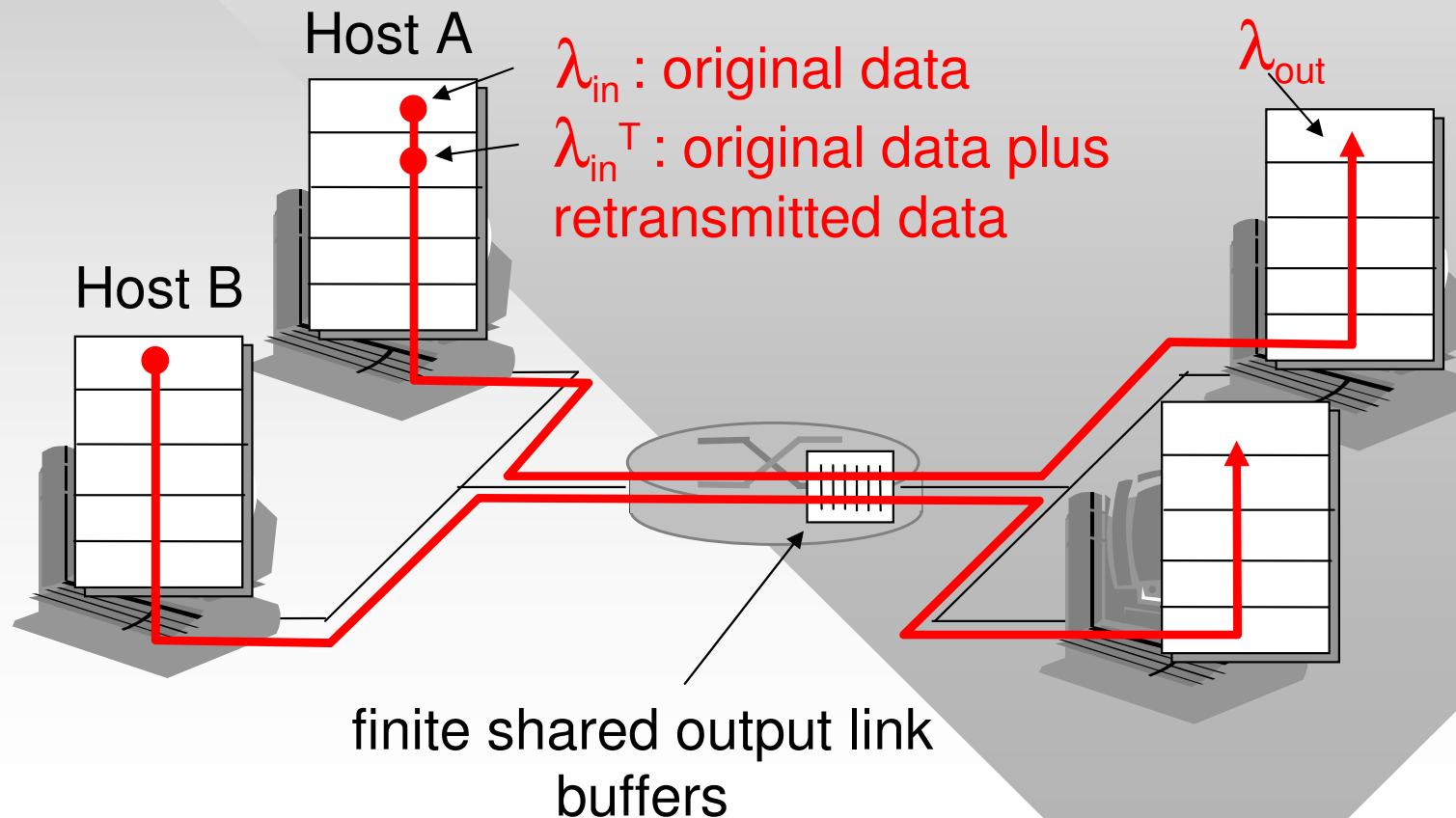
- Two senders, two receivers
- One router, infinite buffers
- No retransmission



Cost 1: large network delays in congested routers

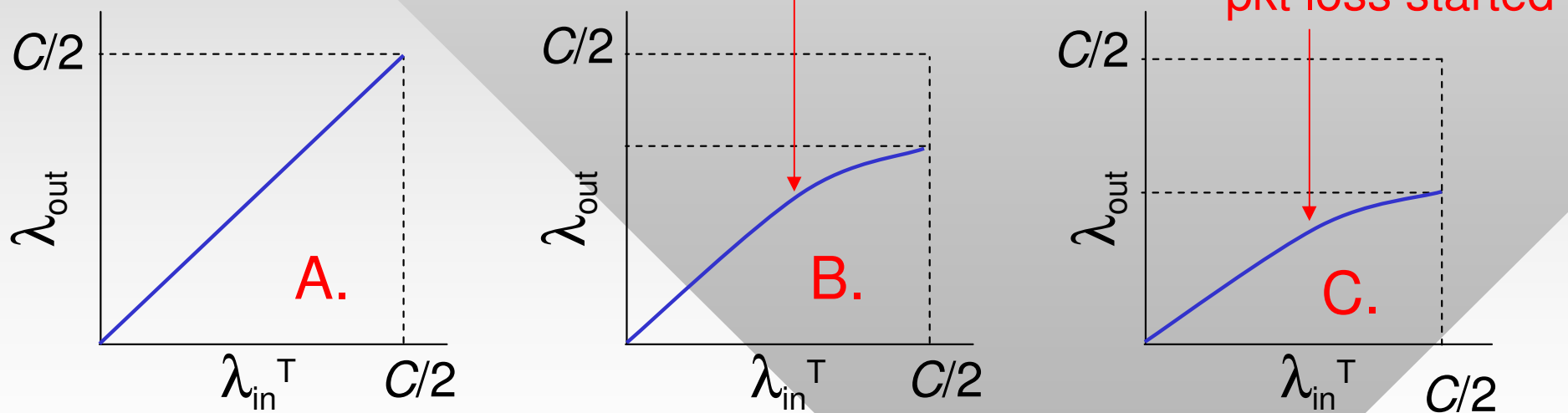
Causes/costs of Congestion: Scenario 2

- One router, *finite* buffers (pkt loss is possible now)
- Sender retransmission of lost packet



Causes/costs of Congestion: Scenario 2

- Always: $\lambda_{out} = \lambda_{in}$, which we call *goodput*
 - Case A: pkts never lost while $\lambda_{in} < C/2$ (not realistic)
 - Case B: pkts are lost when λ_{in} is “sufficiently large,” but timeouts are perfectly accurate (not realistic either)
 - Case C: same as B, but timer is not perfect (duplicate packets are possible)



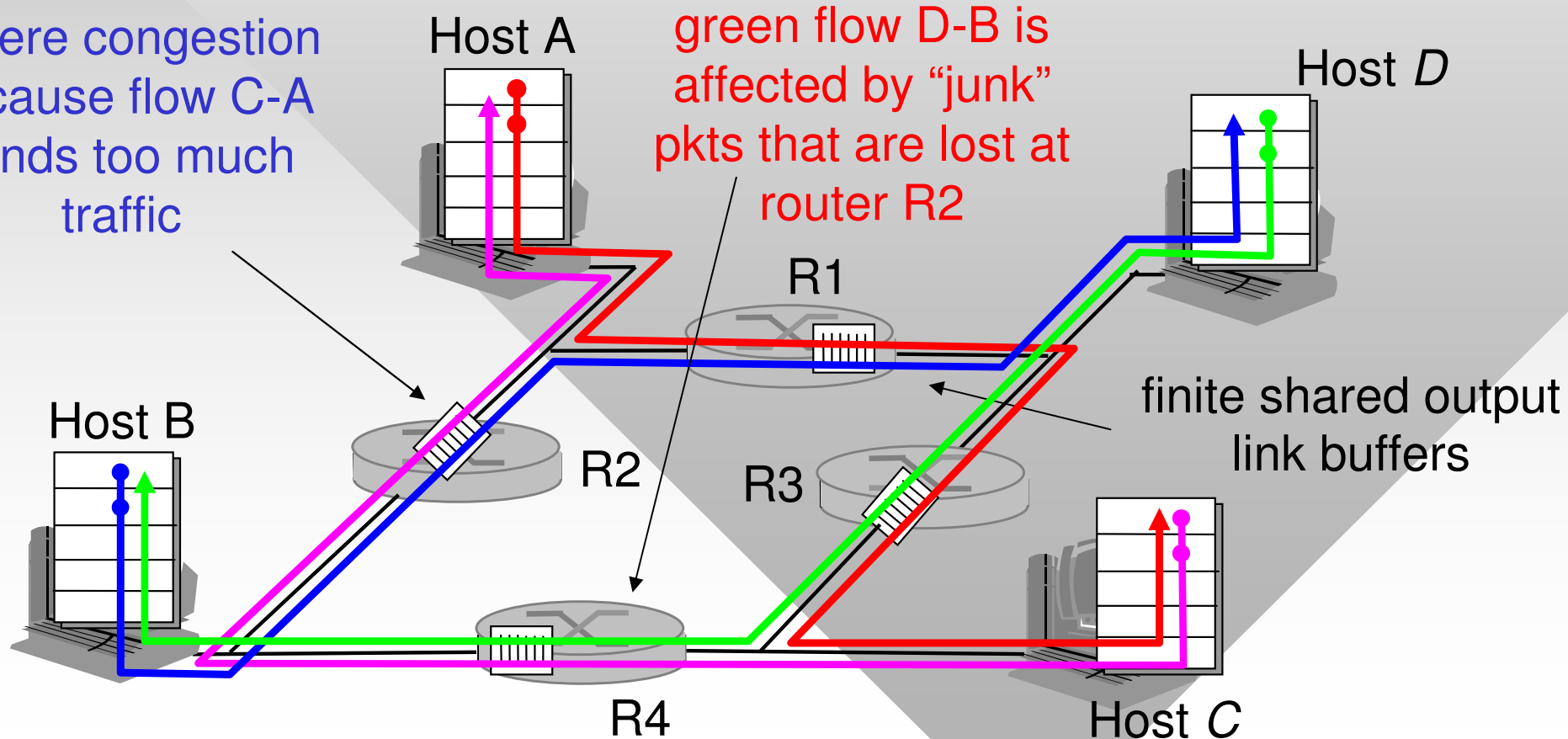
Cost 2: retransmission of lost packets and premature timeouts increase network load, but do nothing for the goodput

Causes/costs of Congestion: Scenario 3

- Four senders
- Multihop paths
- Timeout/retransmit

Cost 3: congestion causes bandwidth waste at other routers

severe congestion because flow C-A sends too much traffic



How to Deal with Congestion?

- Suppose user A sends at rate C and a new user B joins to send at initial rate $C/2$
- Can both users continue sending at the same rate?
 - Assume that λ is their combined rate and both senders implement a reliable service (e.g., TCP)
 - Step1: send $\lambda > C$ pkts and lose $x_1 = \lambda - C$ of them
 - Step2: send λ pkts and x_1 retx'ed pkts; observe that $x_2 = \lambda + x_1 - C = 2(\lambda - C) = 2x_1$ are lost
 - Step3: send $\lambda + x_2$ and observe that loss is now $3x_1$ pkts
- Eventually, almost all sent packets are retransmissions
- The only solution: reduce sending rate in response to congestion

Approaches Towards Congestion Control

Two broad approaches towards congestion control:

End-to-end:

- No **explicit** feedback from network
- Congestion *inferred* by end-systems from observed loss/delay
- Approach taken by TCP

Network-assisted:

- Routers provide feedback to end systems
 - Single bit indicating congestion (DECbit, TCP/IP ECN, ATM)
 - Explicit rate senders should send at (ATM)

Chapter 3: Roadmap

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

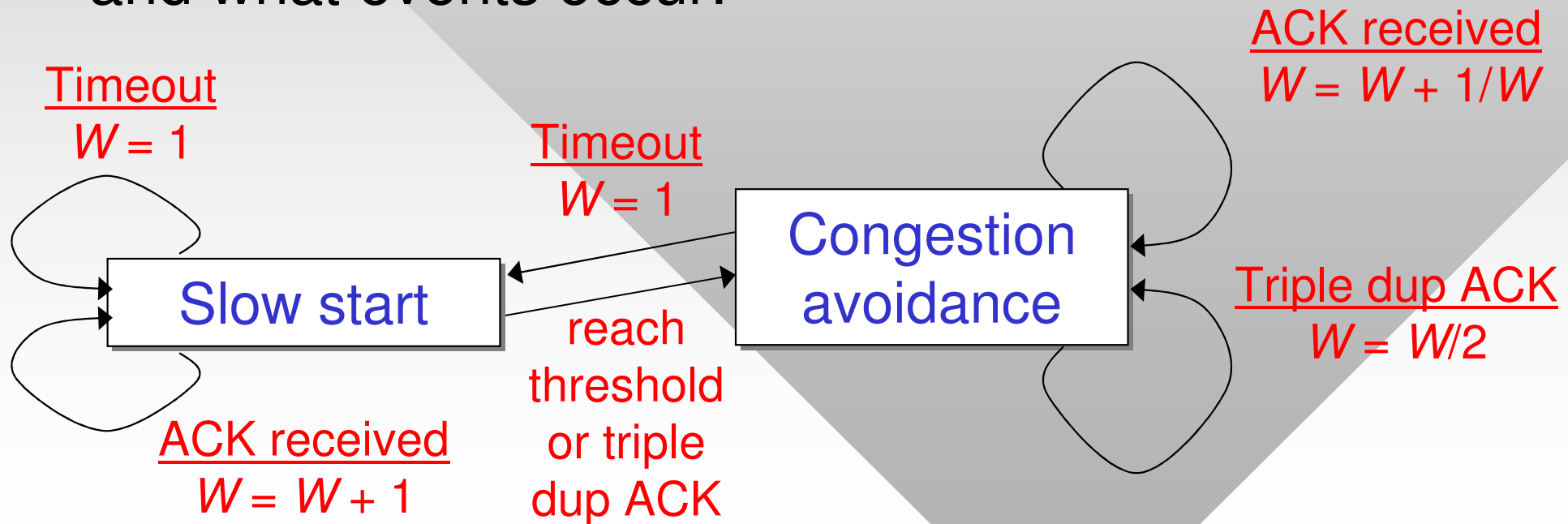
- Segment structure
- Reliable data transfer
- Flow control
- Connection management

3.6 Principles of congestion control

3.7 TCP congestion control

TCP Congestion Control

- TCP congestion control has a variety of algorithms developed over a number of years
 - TCP Tahoe (1988)
 - TCP Reno (1990)
- Behavior of TCP depends on what phase it is in and what events occur:



TCP Congestion Control

- **End-to-end** control (no network assistance)
- Sender limits transmission:

`LastByteSent` -
`LastByteAked` \leq `CongWin`

- `CongWin` is a function of perceived network congestion
- The *effective* window is the minimum of `CongWin` and flow-control window carried in the ACKs

How does sender perceive congestion?

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

Three mechanisms:

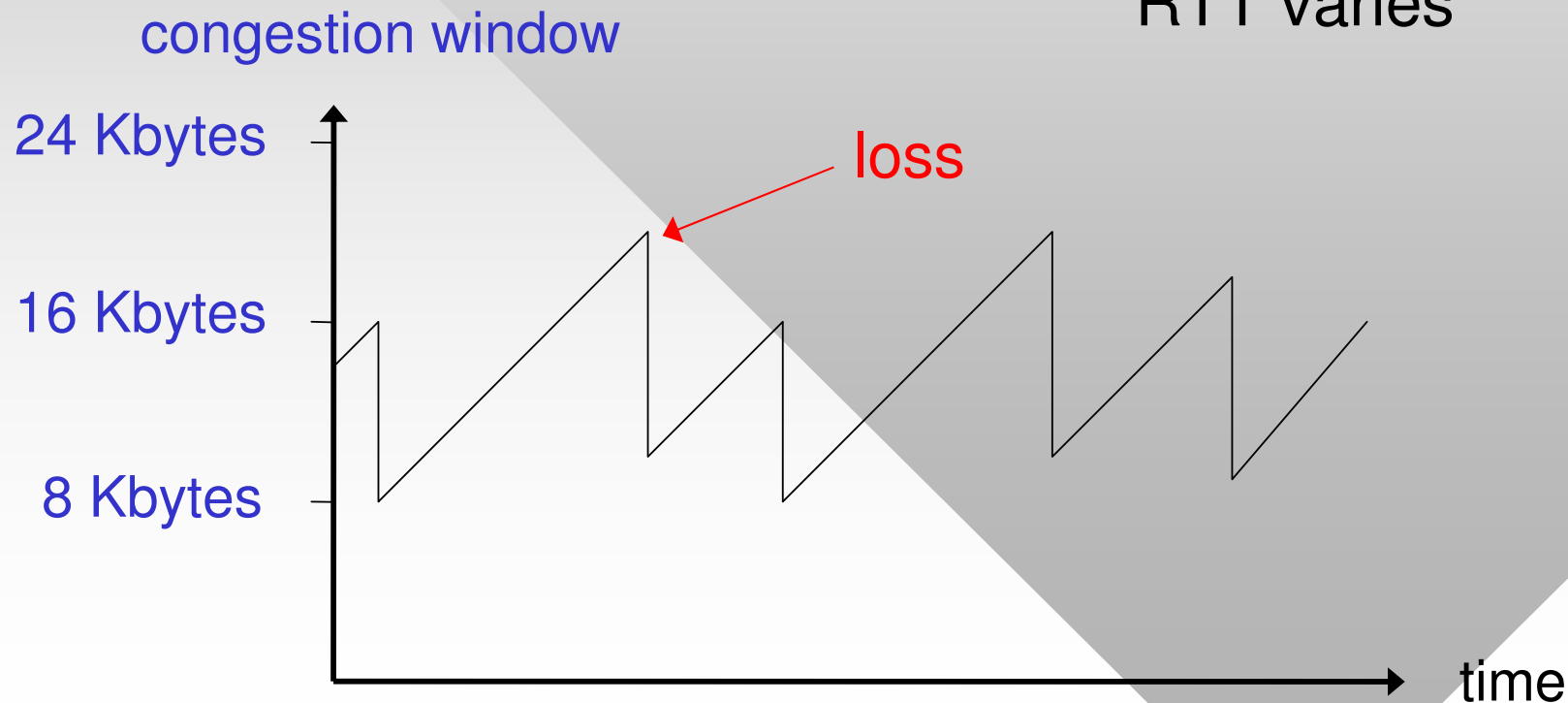
- AIMD
- Slow start
- Conservative after timeout events

TCP AIMD (Additive Increase, Multiplicative Decrease)

Additive increase: increase CongWin by 1 MSS every RTT in the absence of loss events: *probing*

Multiplicative decrease: cut CongWin in half after loss event

Peaks are different: # of flows changes or their RTT varies



TCP Equations

- To better understand TCP, we next derive its AIMD equations (**congestion avoidance**)
- Assume that W is the window size in pkts and $B = \text{CongWin}$ is the same in bytes ($B = \text{MSS} * W$)

- General form:

$$W = \begin{cases} W + \frac{1}{W} & \text{per ACK} \\ W/2 & \text{per loss} \end{cases}$$

- Reasoning
 - For each window of size W , we get exactly W acknowledgments in one RTT (assuming no loss!)
 - This increases window size by roughly 1 packet per RTT (0.9 for $W=2$ and very close to 1 for large W)

TCP Equations

$$W = \begin{cases} W + \frac{1}{W} & \text{per ACK} \\ W/2 & \text{per loss} \end{cases}$$

- What is the equation in terms of B ?

$$B = \begin{cases} B + \frac{MSS^2}{B} & \text{per ACK} \\ B/2 & \text{per loss} \end{cases}$$

- Equivalently, TCP increases B by MSS per RTT
- What is the rate of TCP given that its window size is B (or W)?
- Since TCP sends a full window of pkts per RTT, its ideal rate can be written as:

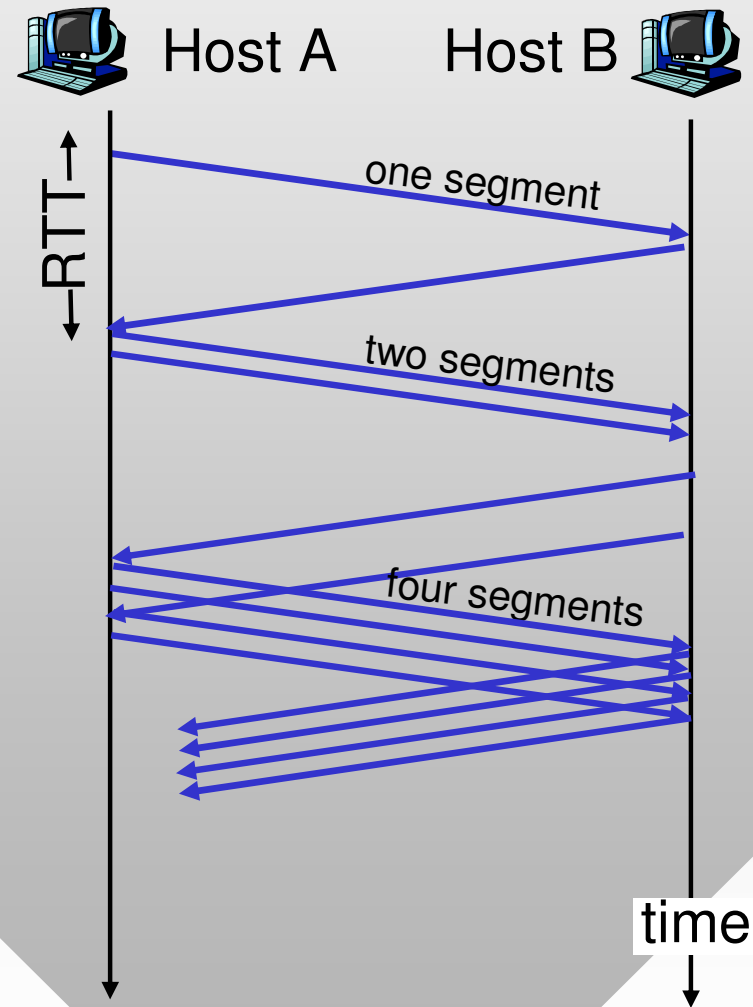
$$r = \frac{B}{RTT + L/R} \approx \frac{B}{RTT} = \frac{MSS * W}{RTT}$$

TCP Slow Start

- When connection begins, $\text{CongWin} = 1 \text{ MSS}$
 - Example: $\text{MSS} = 500 \text{ bytes}$ and $\text{RTT} = 200 \text{ msec}$
 - Q: initial rate?
- A: 20 kb/s
- Available bandwidth may be much larger than MSS/RTT
 - Desirable to quickly ramp up to a “respectable” rate
- Solution: **Slow Start (SS)**
 - When a connection begins, it increases rate exponentially fast until first loss event
 - Term “slow” is used to distinguish this algorithm from earlier TCPs which directly jumped to some huge rate

TCP Slow Start (More)

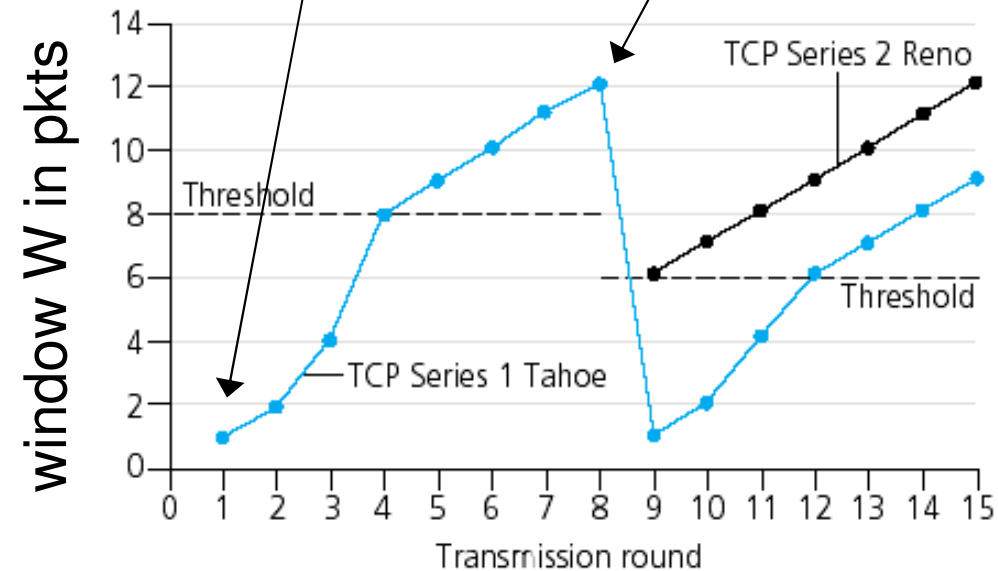
- Slow start
 - Double CongWin every RTT
- Done by incrementing CongWin for every ACK received:
 - $W = W + 1$ per ACK
 - $(B = B + \text{MSS per ACK})$
- Summary: initial rate is slow but ramps up exponentially fast



Refinement

- Upon timeout:
 - `CongWin` is set to 1 MSS
 - Window then grows **exponentially** (slow start) to a threshold, then grows **linearly** (congestion avoidance)
 - *Introduced in TCP Tahoe*
- After 3 dup ACKs:
 - `CongWin` is cut in half
 - Window then grows linearly
 - Called “**Fast Recovery**”
 - *Introduced in TCP Reno*

Reno: triple ACK, Tahoe: timeout
previous timeout



Philosophy:

Three dup ACKs indicate that network is capable of delivering some segments

Timeout before 3 dup ACKs is “more alarming”