ECE 356/COMPSI 356
Computer Network Architecture

TCP Congestion Control

Wednesday November 6th, 2019

Recap

- Last lecture:
  - TCP reliable communications
  - TCP flow control
  - TCP connection establishment

- Readings for this lecture: **PD 6.3**

Lecture Outline

- Understanding congestion
- Principles of congestion control
- Congestion control algorithm components:
  - Slow start
  - Congestion avoidance
  - Fast recovery
- Congestion control as a feedback control system

Recap: Flow Control vs. Congestion Control

- **Flow control**: receiver controls sender so sender won't overflow receiver's buffer by transmitting too much, too fast

- **Congestion control**: throttling the sender due to congestion on the network

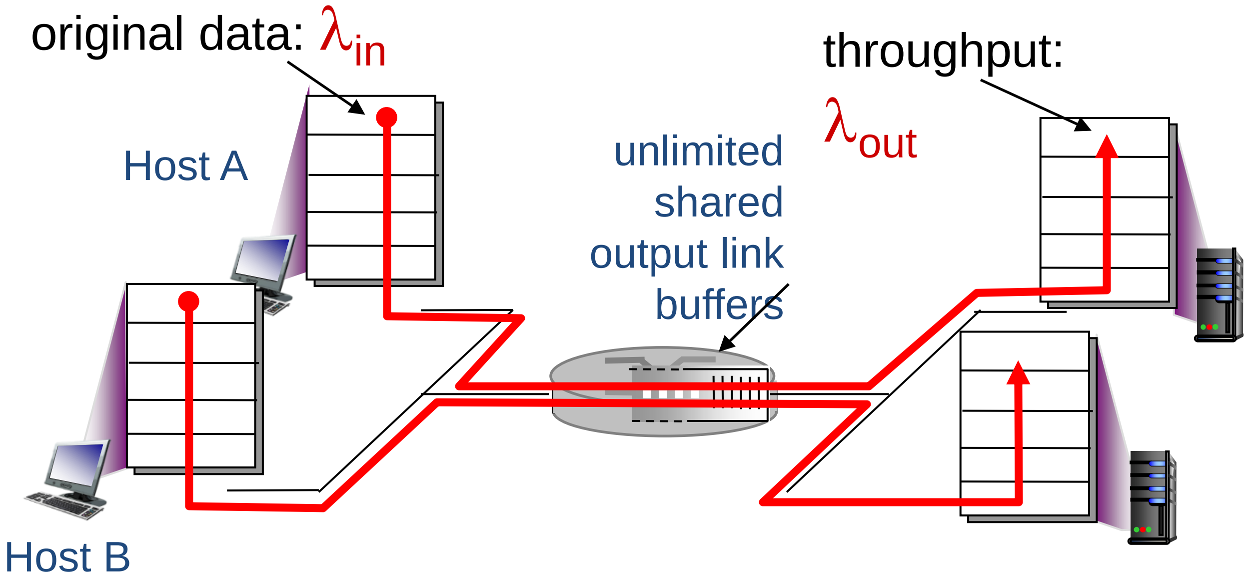
Understanding Congestion

- Like road congestion, if we could also lose cars in transit

Understanding Congestion

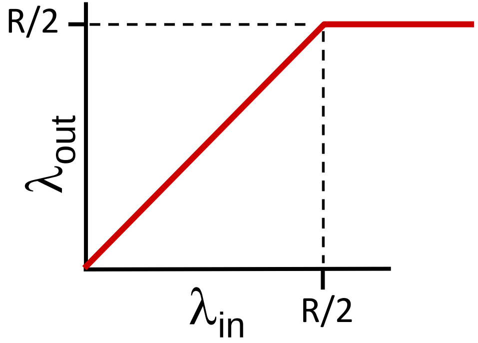
- Informally: "*too many sources sending too much data too fast for network to handle*"
  - Different from flow control
- Manifestations:
  - Long delays (queuing in router buffers)
  - Lost packets (buffer overflow at routers)
- An important and interesting problem
  - Nodes make independent distributed decisions

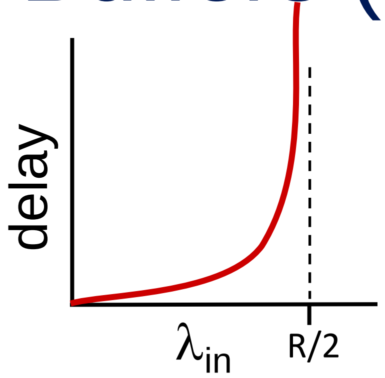
The Causes and Costs of Congestion: One Router, Infinite Buffers (1/2)



- Two senders, two receivers
- One router, infinite buffers
- Output link capacity: R
- *No retransmissions*

The Causes and Costs of Congestion: One Router, Infinite Buffers (2/2)

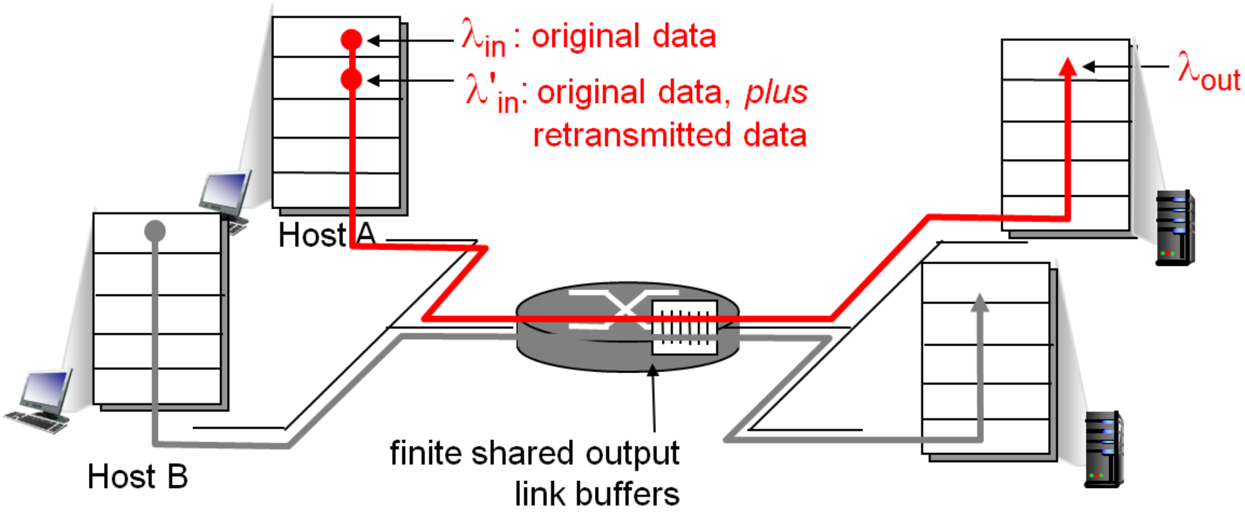




- Maximum per-connection throughput: R/2
- Large delays as arrival rate, λ_{in}, approaches capacity

- Cost of congestion: large queuing delays experienced as packet-arrival rates near link capacity

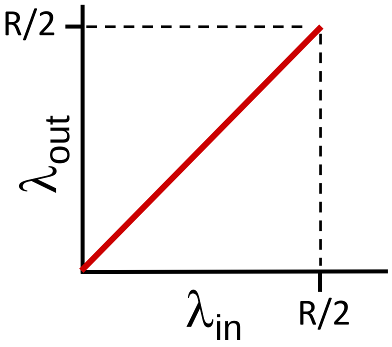
The Causes and Costs of Congestion: One Router, Finite Buffers (1/3)

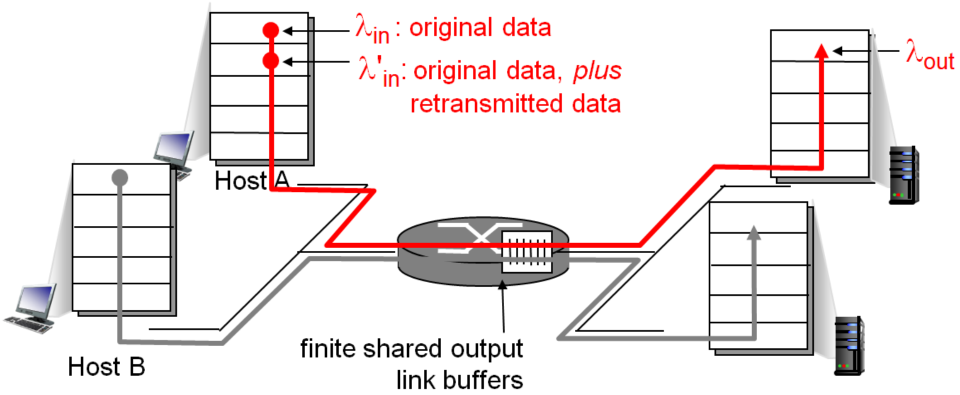


- Sender retransmission of timed-out packets
  - Application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - Transport-layer input includes *retransmissions* : $\lambda_{in}' \geq \lambda_{in}$

The Causes and Costs of Congestion: One Router, Finite Buffers (2/3)

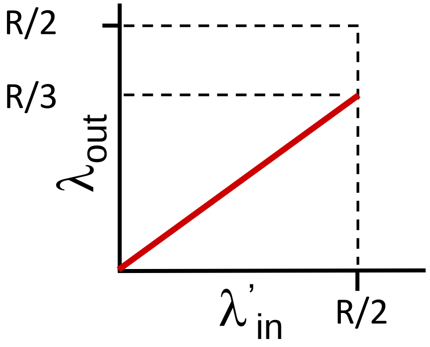
- Idealization: perfect knowledge
  - Sender sends only when router buffers available

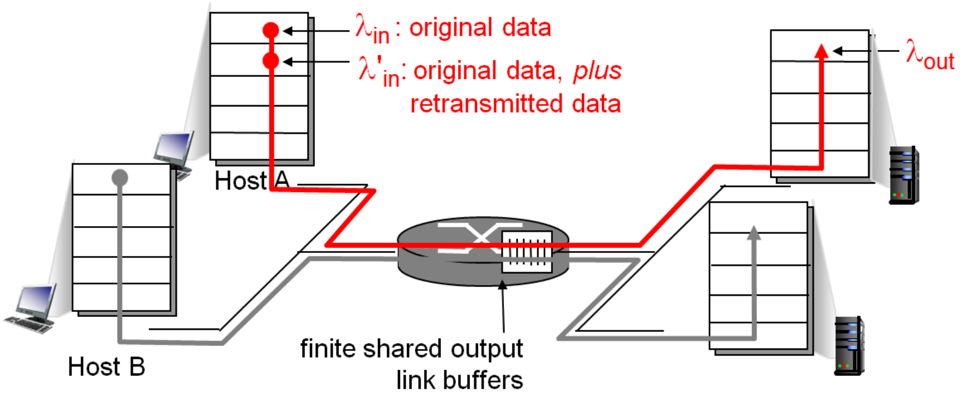




The Causes and Costs of Congestion: One Router, Finite Buffers (3/3)

- Packets can be lost, dropped at router due to full buffers

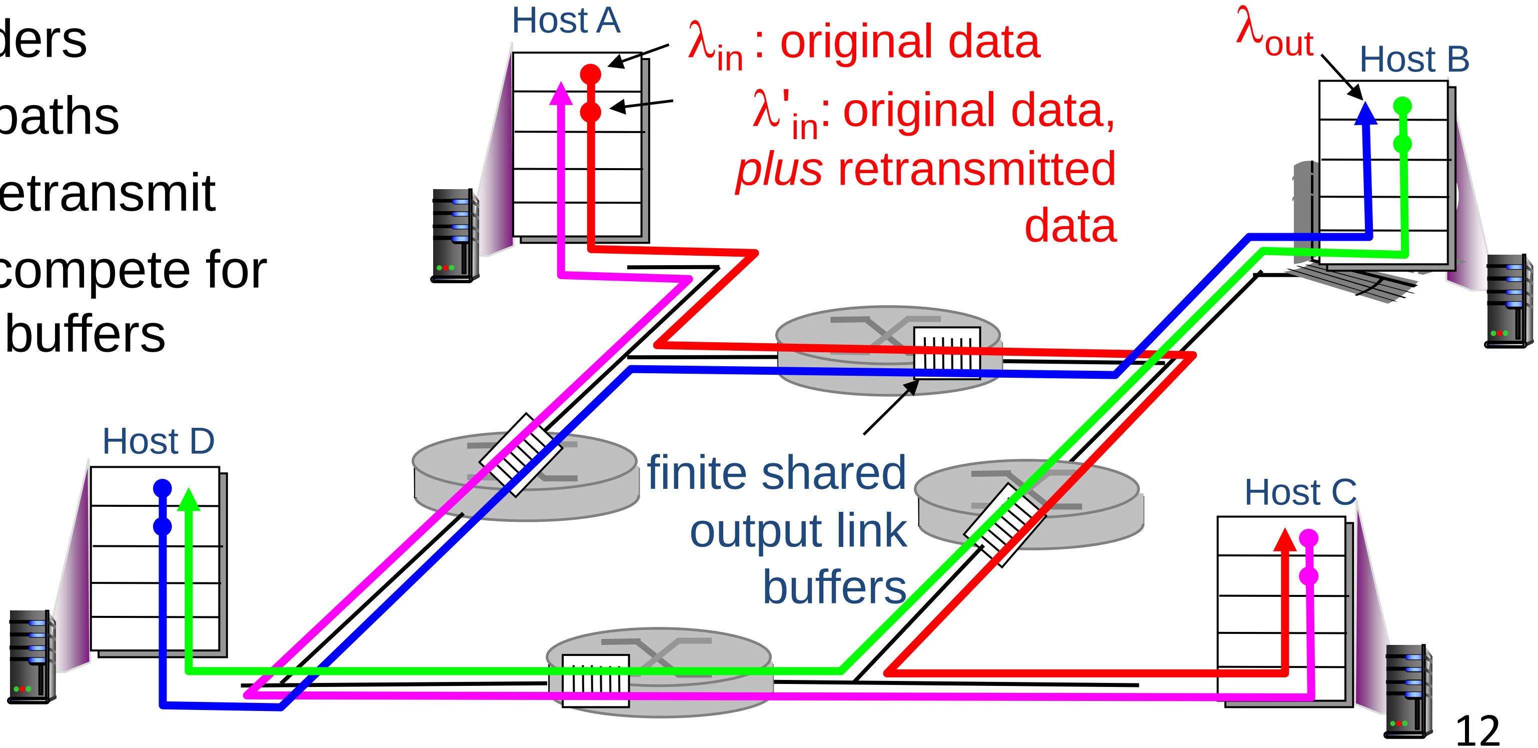




- Cost of congestion: sender must perform retransmissions in order to compensate for dropped packets due to buffer overflow

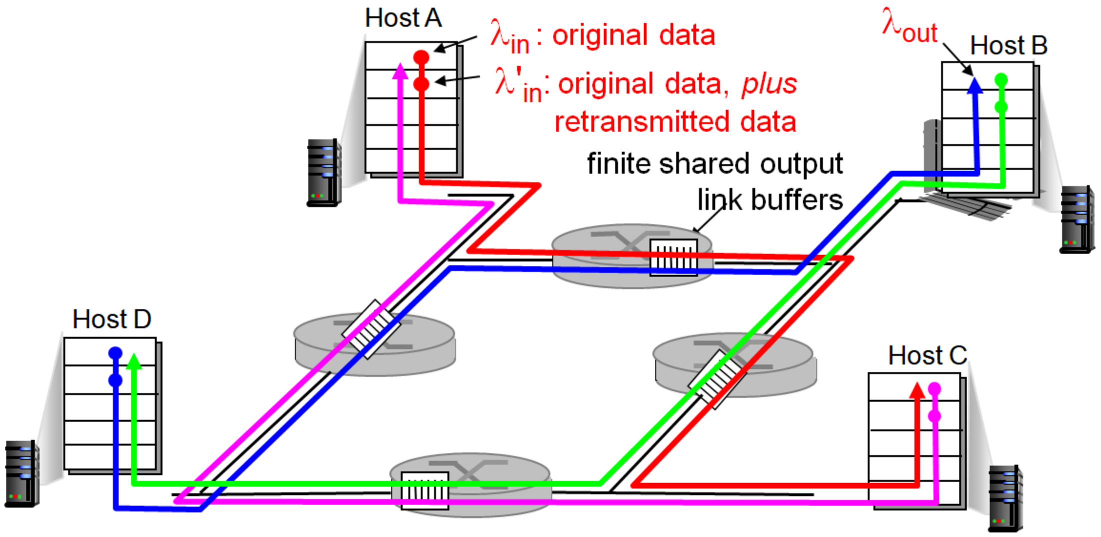
The Causes and Costs of Congestion: Multi-Hop Paths (1/3)

- Four senders
- Multihop paths
- Timeout/retransmit
- Senders compete for space on buffers

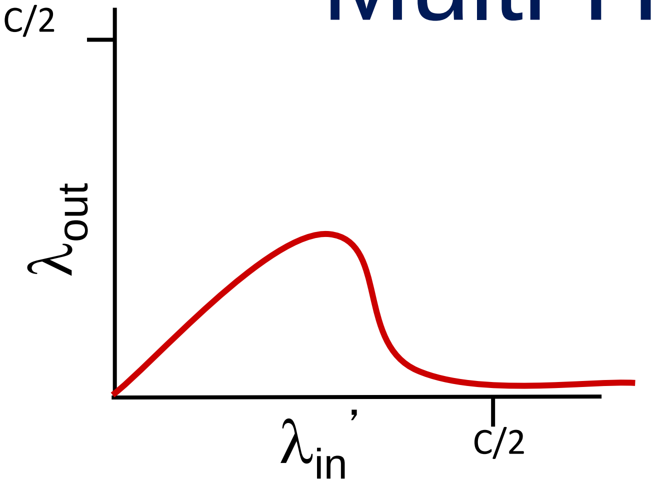


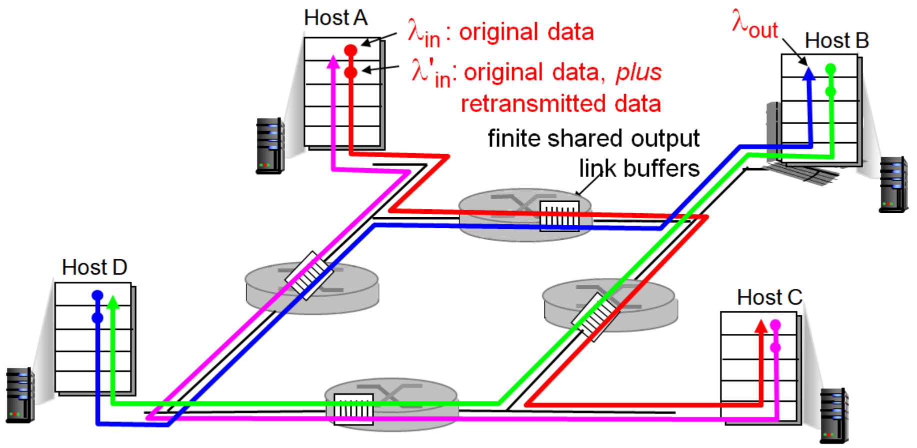
The Causes and Costs of Congestion: Multi-Hop Paths (2/3)

- 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



The Causes and Costs of Congestion: Multi-Hop Paths (3/3)





- Cost of congestion: when a packet is dropped along the path, the capacity used for it is wasted

History (1/2)

- The original TCP/IP design did not include congestion control
  - Receiver uses advertised window to do flow control
  - No exponential backoff after a timeout

History (2/2)

- It led to **congestion collapse** in October 1986
  - The NSFnet phase-I backbone dropped three orders of magnitude from its capacity of 32 kbit/s to **40 bit/s**
    - 800x difference
  - This continued until end nodes started implementing Van Jacobson's congestion control between 1987 and 1988

Understanding Congestion: Key Points to Remember

- Too many sources sending too much data too fast for network to handle
  - ➢ A network-level phenomenon
- Congestion control ≠ flow control
- Costs of congestion include:
  - ➢ Large queuing delays
  - ➢ Retransmissions to compensate for packets dropped at intermediate routers
  - ➢ Wasted work in forwarding packets that will be dropped

Lecture Outline

- Understanding congestion
- **Principles of congestion control**
- Congestion control algorithm components:
  - Slow start
  - Congestion avoidance
  - Fast recovery
- Congestion control as a feedback control system

Congestion Control: Challenge

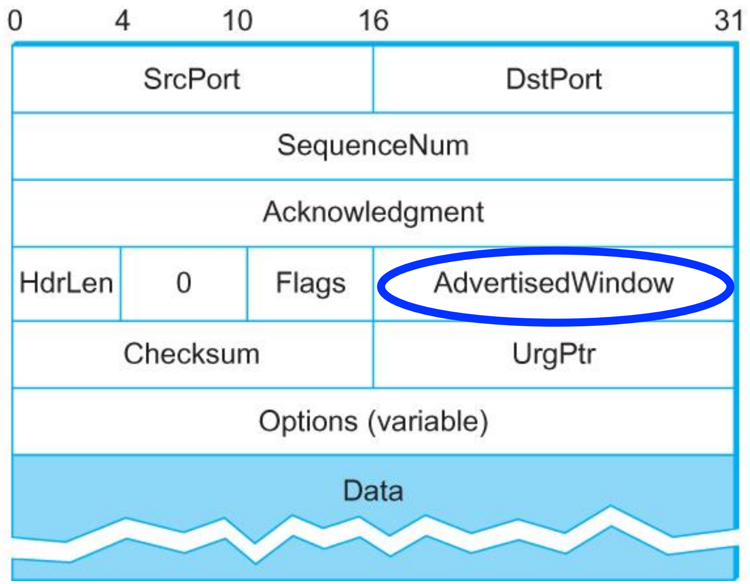
- Send at the “right” speed
  - Fast enough to keep the pipe full
  - But not to overload the network
  - Share nicely with other senders

Congestion Control: Approach

- Each sender limits the rate at which it sends traffic into its connection, as a function of *perceived network congestion*
  - Q: How does the sender limit the rate?
  - Q: How does the sender perceive congestion?
  - Q: Which algorithm does the sender use to change its send rate?

How Does the Sender Limit Transmission Rate?

- CongestionWindow
- Counterpart to flow control's AdvertisedWindow
  - ➢ But, unlike it, is not explicitly signaled
- Maximum number of bytes in transit: *min(CongestionWindow, AdvertisedWindow)*
  - ➢ Window-based congestion control



How Does a Sender Perceive Congestion?

- ***Packet loss*** is a congestion signal
- Loss events: familiar retransmission triggers
  - ➢ Timeout of a retransmission timer
    - *Nothing* is getting through?
  - ➢ Receipt of three duplicate ACKs
    - *Something* is passing through the channel

Congestion Detection: Wireless Network Complications

- Recall that wireless networks are much more error prone than wired networks
- In wireless networks, loss $\neq$ congestion
  - Could be due to weaker signal, interference
  - A large number of packets can get lost
- TCP can slow down to a crawl

TCP for Wireless: Active Area of Research

- One option: splitting the connection into wired and wireless segments
  - Creating a *middlebox*
  - Deviating from end-to-end transport layer architecture
- Another option: distinguish between congestion and bit errors
  - Other congestion clue: explicit congestion notification
  - Another one: increasing RTT values

How Does a Sender Know There is No Congestion?

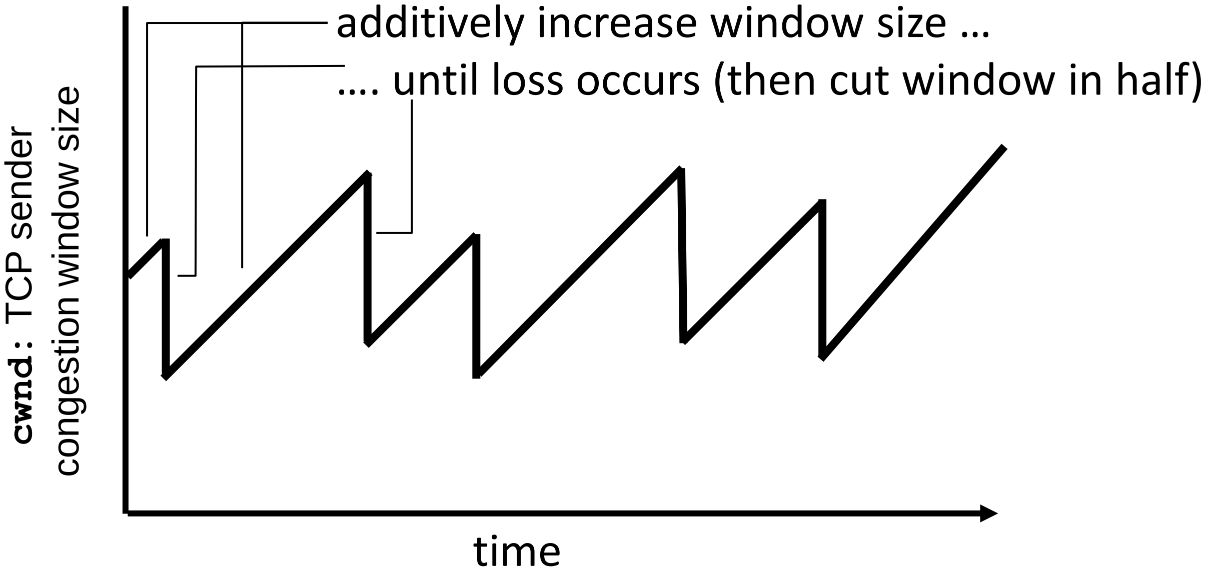
- Receiving acknowledgements
- Increase congestion window size when acknowledgements are received
  - ➢ Acknowledgements arrive slowly → slow increase
  - ➢ Acknowledgements arrive quickly → fast increase
- "*Self-clocking*" mechanism

Algorithm: Additive Increase Multiplicative Decrease (1/2)

- *Bandwidth probing*
- 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

Algorithm: Additive Increase Multiplicative Decrease (2/2)

AIMD "*sawtooth behavior*": probing for bandwidth



Multiple Flavors of TCP

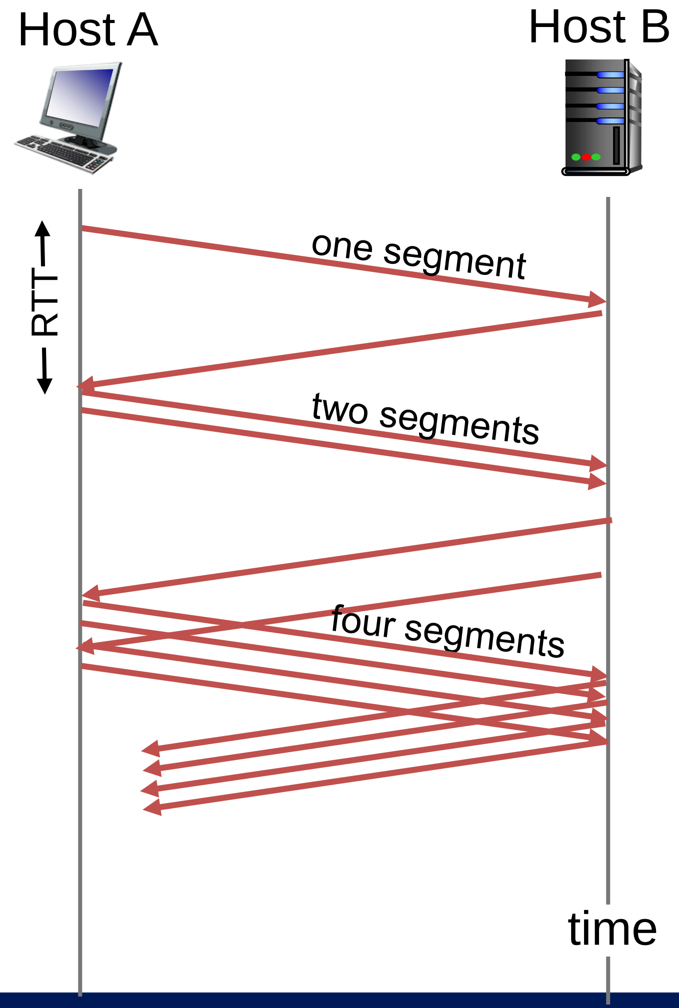
- TCP Tahoe, Reno, Vegas, BBR, CUBIC, …
- Different feedback signals
- Different specifics of sawtooth patterns

Lecture Outline

- Understanding congestion
- Principles of congestion control
- **Congestion control algorithm components:**
  - **Slow start**
  - **Congestion avoidance**
  - **Fast recovery**
- Congestion control as a feedback control system

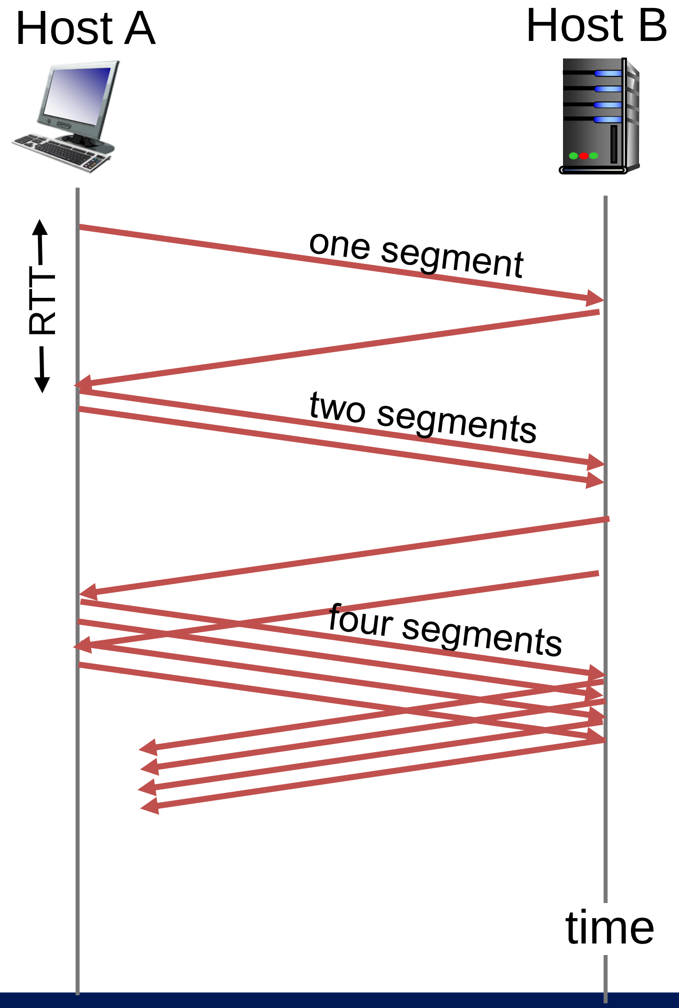
TCP Slow Start (1/2)

- When connection begins, increase rate exponentially until first loss event
  - ➢ Initially cwnd = 1 MSS
    - More in modern TCP variants
  - ➢ Double cwnd every RTT



TCP Slow Start (1/2)

- Done by incrementing cwnd for every ACK received
  - Incrementing per ACK, not per segment count
  - Same if acknowledging less than 1 MSS, or many consecutive transmissions
- Summary: initial rate is slow but ramps up exponentially fast



Switching from Slow Start to Congestion Avoidance

Q: when should the exponential increase stop?

- Switch to linear increase: **congestion avoidance**

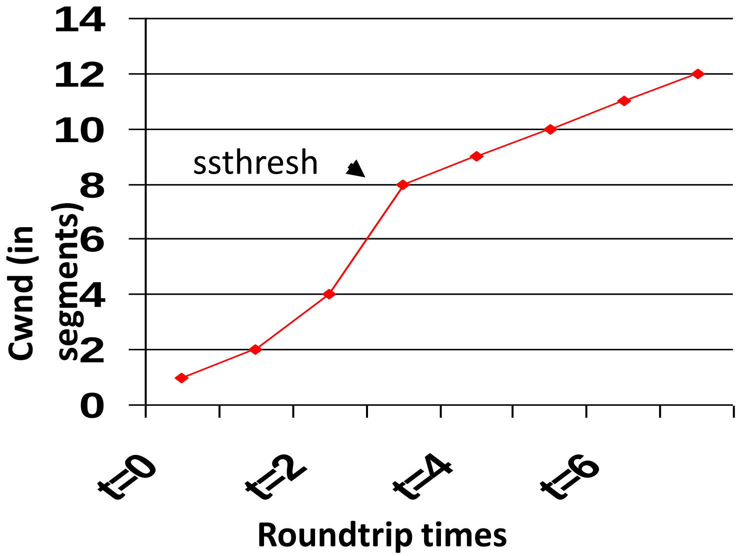
A: when **cwnd** gets to 1/2 of its value before timeout

Implementation:

- Variable **`ssthresh`**
- On loss event, **`ssthresh`** is set to 1/2 of **`cwnd`** just before loss event

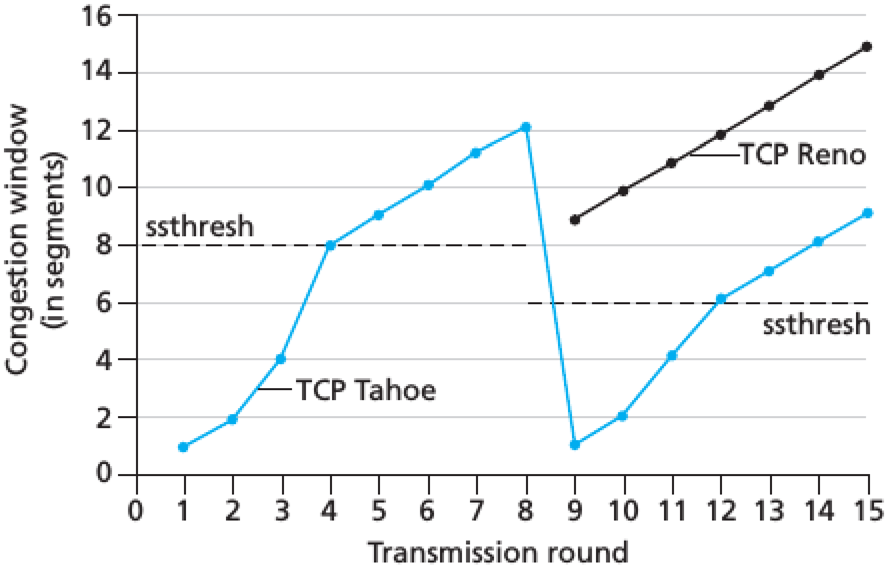
An Example of
Slow Start/Congestion Avoidance

For *ssthresh = 8 MSS*



Slow Start: Reacting to Losses

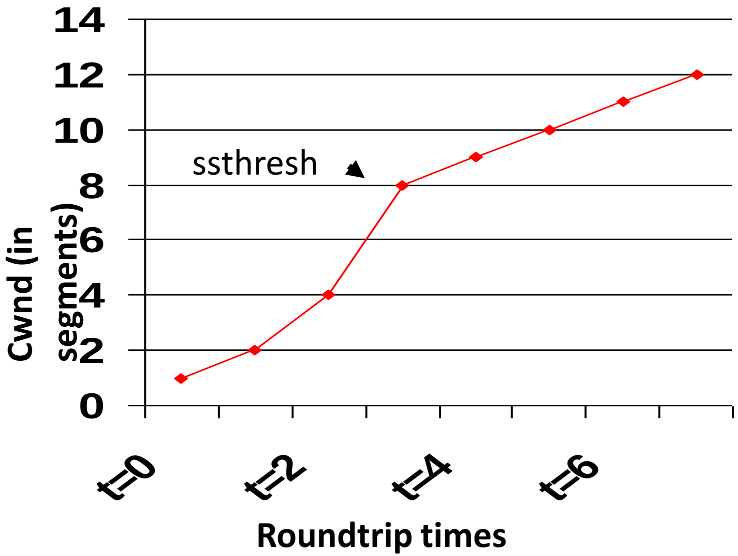
- Timeout
  - ssthresh ← cwnd/2
  - cwnd ← 1 MSS
  - Slow start begins anew
- 3 duplicate ACKs
  - Fast retransmit
  - Enters **fast recovery stage**



TCP Tahoe example

TCP Congestion Avoidance

- On entry to congestion avoidance stage, cnwd is 1/2 the value of what it was when congestion was last encountered
  - ➢ Congestion could be just around the corner
- Conservative growth approach: increase the value of cwnd by 1 MSS every RTT



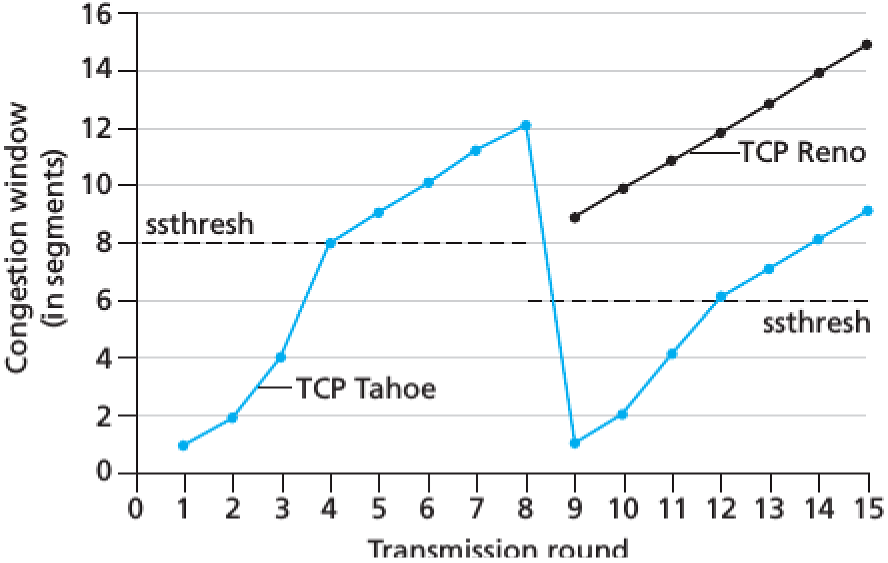
TCP Congestion Avoidance: Exiting

- On a timeout:
  - cwnd set to 1 MSS
  - ssthresh set to 1/2 cwnd when timeout occurred
  - To slow start state
- On a triple duplicate ACK:
  - Fast retransmit
  - cwnd $\leftarrow$ cwnd/2 + 3 MSS
  - ssthresh $\leftarrow$ cwnd/2
  - To fast recovery state

TCP Fast Recovery

- Recommended, but not required
- Avoiding slow start
  - The value of cwnd is increased by 1 MSS for every duplicate ACK received for the missing segment that caused TCP to enter fast recovery state
- When ACK arrives for the missing segment:
  - cwnd $\leftarrow$ ssthresh
  - Enter **congestion avoidance**

Evolution of TCP Congestion Window: Triple Duplicate ACK

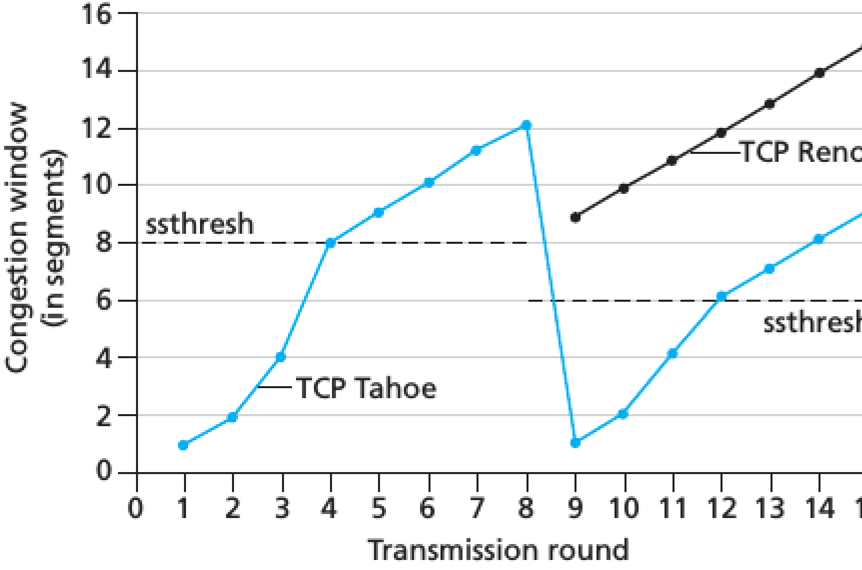


- After a triple duplicate ACK:
  - ssthresh ← cwnd/2
- TCP Tahoe: cwnd ← 1
- TCP Reno: cwnd ← cwnd/2 + 3 MSS

- TCP Tahoe: no fast recovery; TCP Reno: fast recovery

Congestion Control Mechanisms: Key Points to Remember

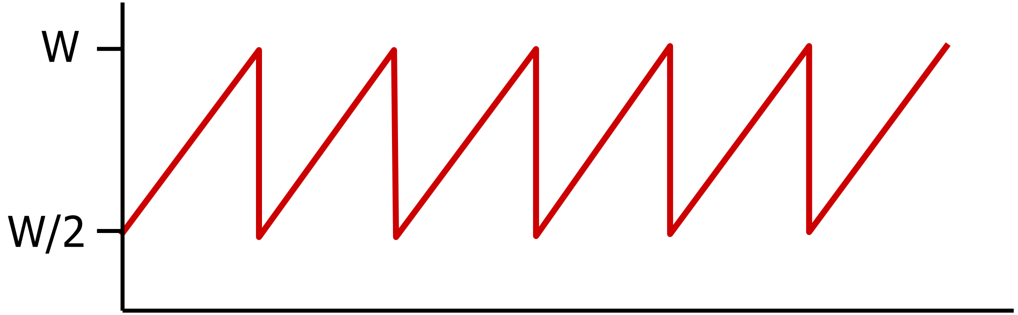
- Congestion window follows a sawtooth pattern
  - ➢ Grows first exponentially, then linearly, until a loss event occurs



Macroscopic Behavior of TCP

- Avg. TCP throughput 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. throughput is 3/4W per RTT

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



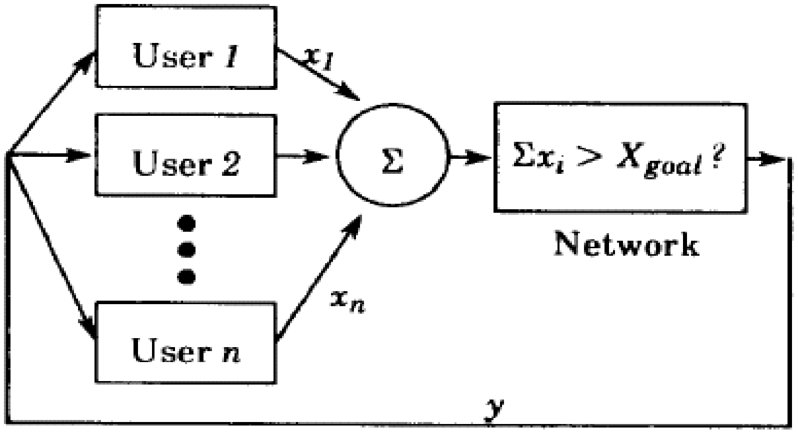
Lecture Outline

- Understanding congestion
- Principles of congestion control
- Congestion control algorithm components:
  - Slow start
  - Congestion avoidance
  - Fast recovery
- **Congestion control as a feedback control system**

Proof of Optimality

- AIMD was developed based on engineering insight and experimentation
- Ten years after, theoretical analysis showed that the congestion control algorithm is optimal
  - Stable
  - Fair

Why Does it Work?



- A feedback control system
- The network uses feedback y to adjust users' load ∑x_i

Goals of Congestion Avoidance

- Efficiency
- Fairness
- Distributedness
  - ➢ A centralized scheme requires complete knowledge of the state of the system
- Convergence
  - ➢ The system approach the goal state from any starting state

Metrics to Measure Convergence

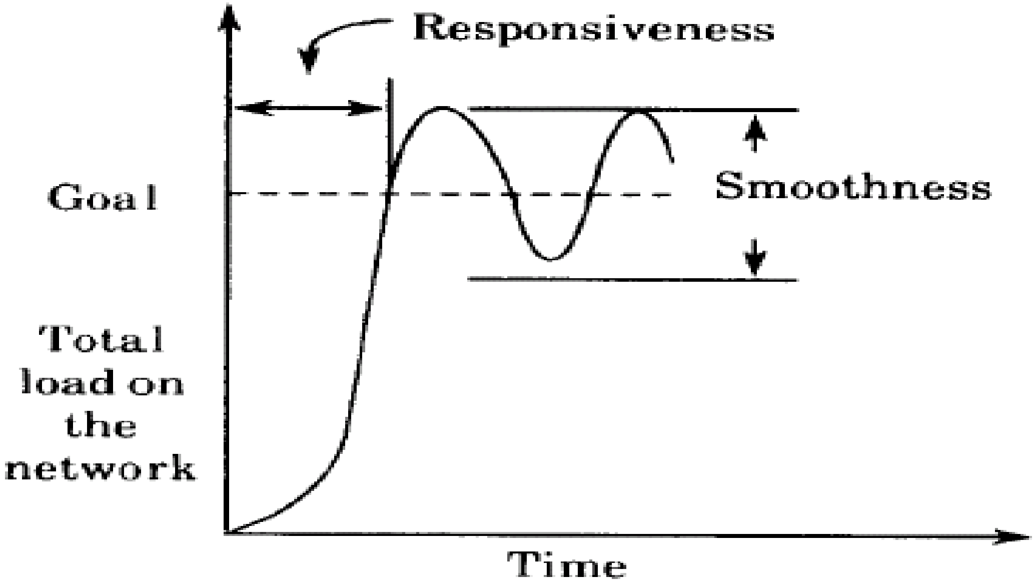
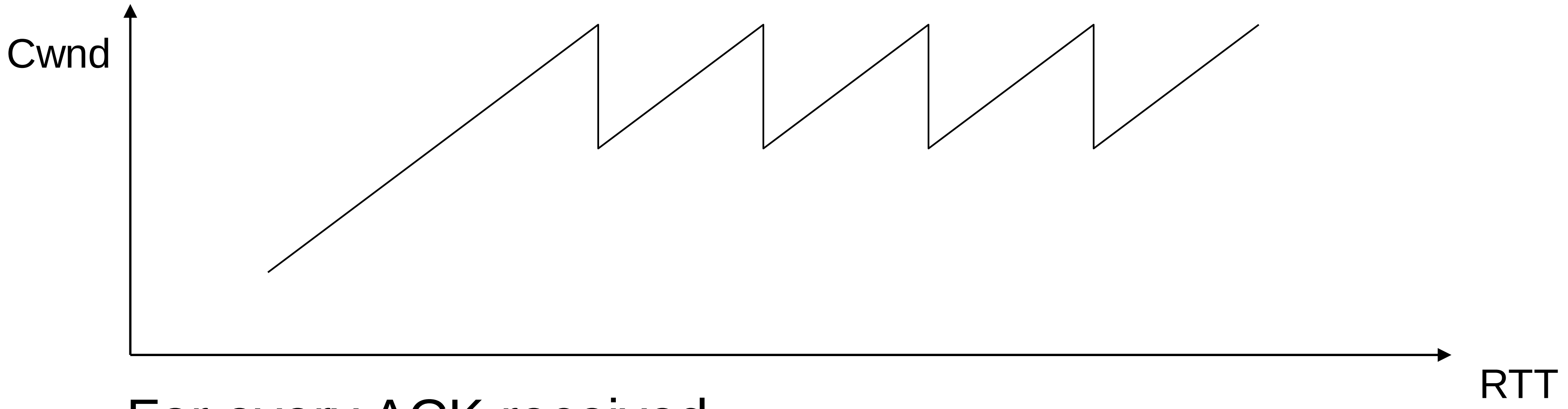


Fig. 3. Responsiveness and smoothness.

- Responsiveness
- Smoothness

The Sawtooth Behavior of TCP



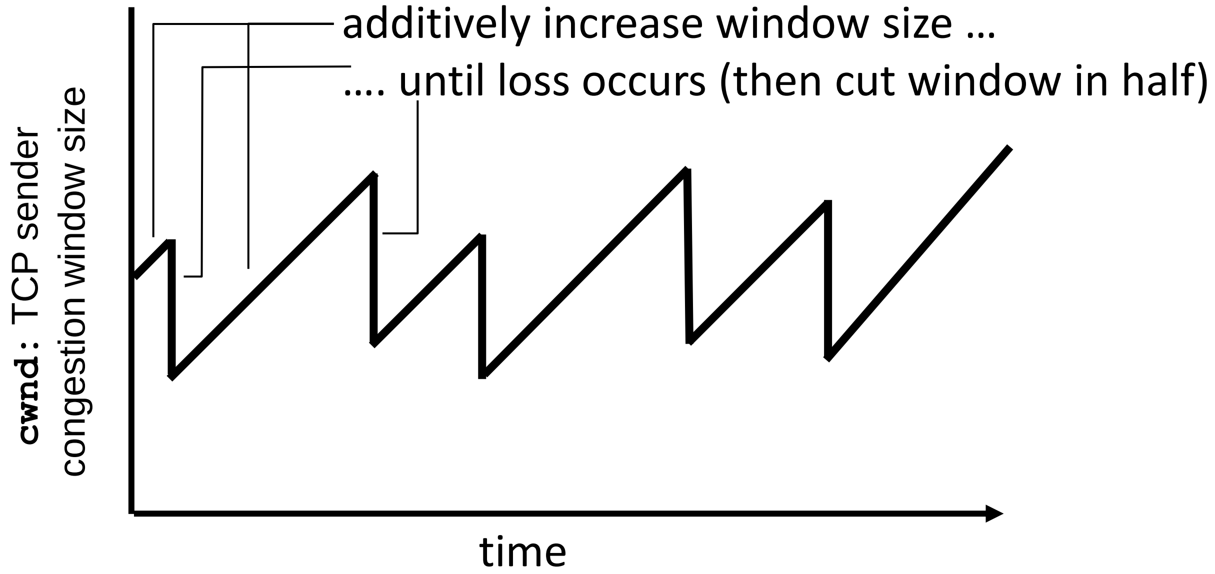
- For every ACK received
  - Cwnd += 1/cwnd *MSS
- For every packet lost
  - Cwnd /= 2

TCP Congestion Control: Key Points to Remember (1/3)

- Network congestion is problematic. It leads to:
  - Delays
  - Segment losses
  - Wasted work of the network
- TCP employs window-based congestion control
  - Maximum number of bytes in transit: *min(CongestionWindow, AdvertisedWindow)*
  - Sender *probes the network* by injecting more and more data in it
  - Backs off when encountering losses

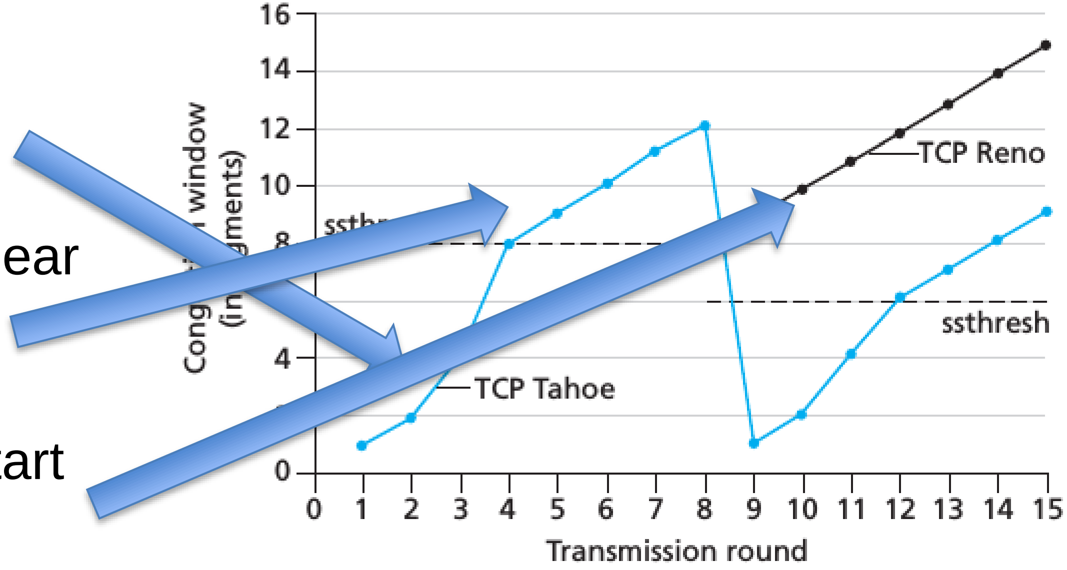
TCP Congestion Control: Key Points to Remember (2/3)

AIMD "*sawtooth behavior*": probing for bandwidth



TCP Congestion Control: Key Points to Remember (3/3)

- Algorithm component:
  - ➢ Slow start: exponential growth of cwnd
  - ➢ Congestion avoidance: linear growth of cwdn
  - ➢ (Recommended) fast recovery: avoiding slow start in case of duplicate ACKs



Next Lecture

- Network resource allocation
  - Queue management
  - Congestion avoidance