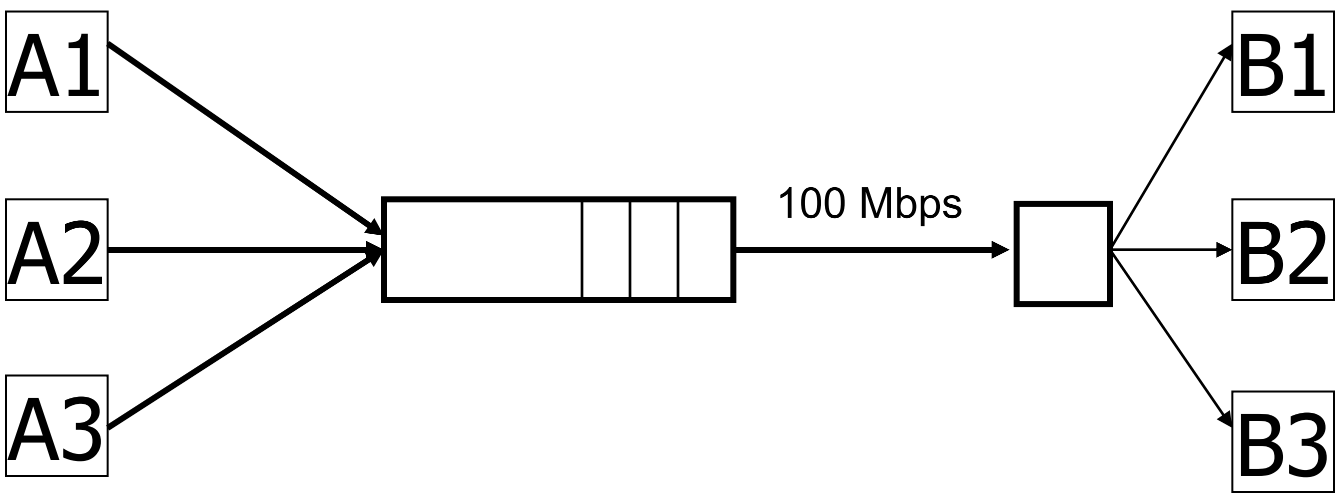
Transport Layer: Congestion Control Overview

ECE 50863 – Computer Network Systems

Congestion Control

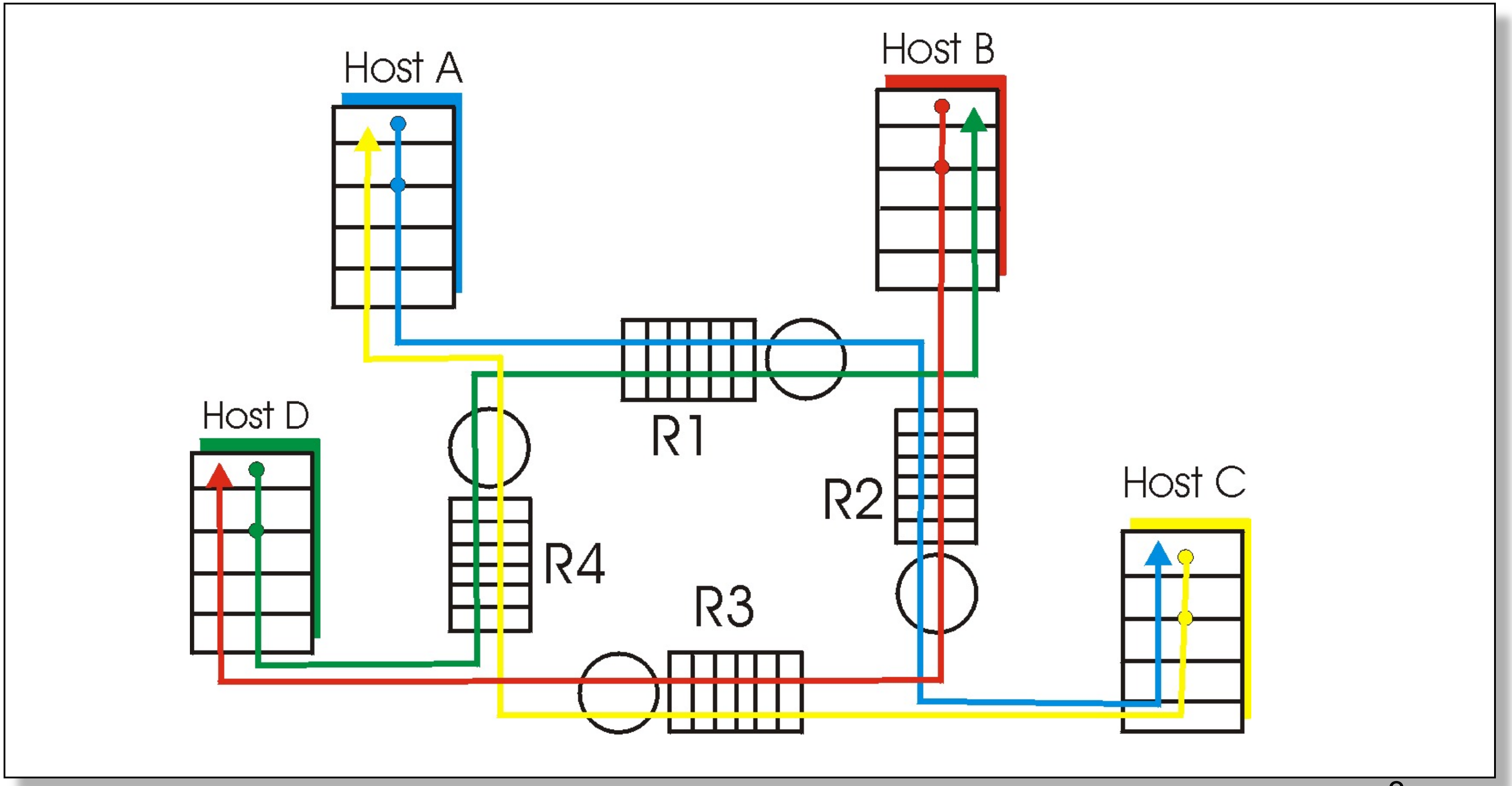
- What is congestion?
- Why does it occur?



Causes & Costs of Congestion

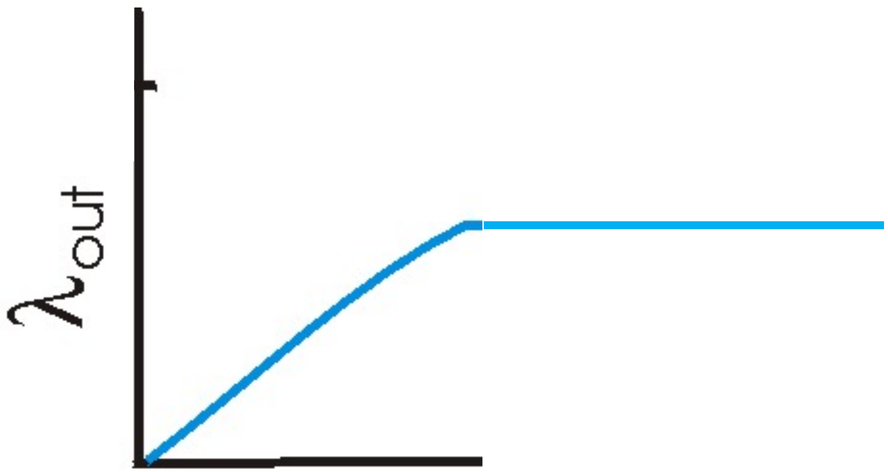
- Four senders – multihop paths

Q: What happens as rate increases?

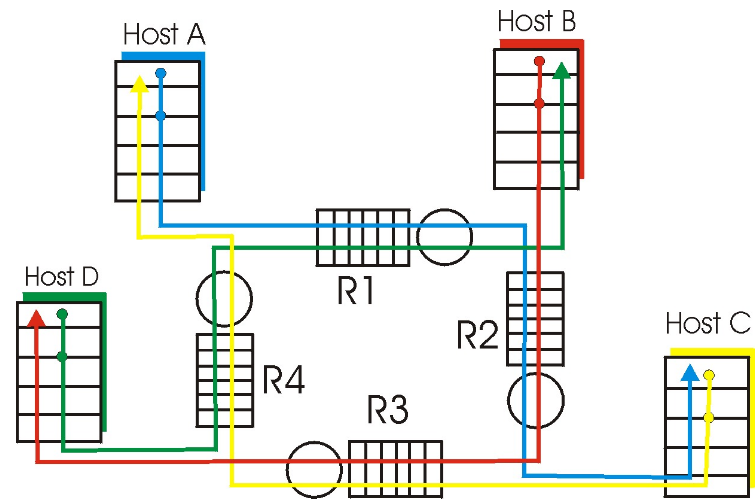


What one might expect

What each receiver gets

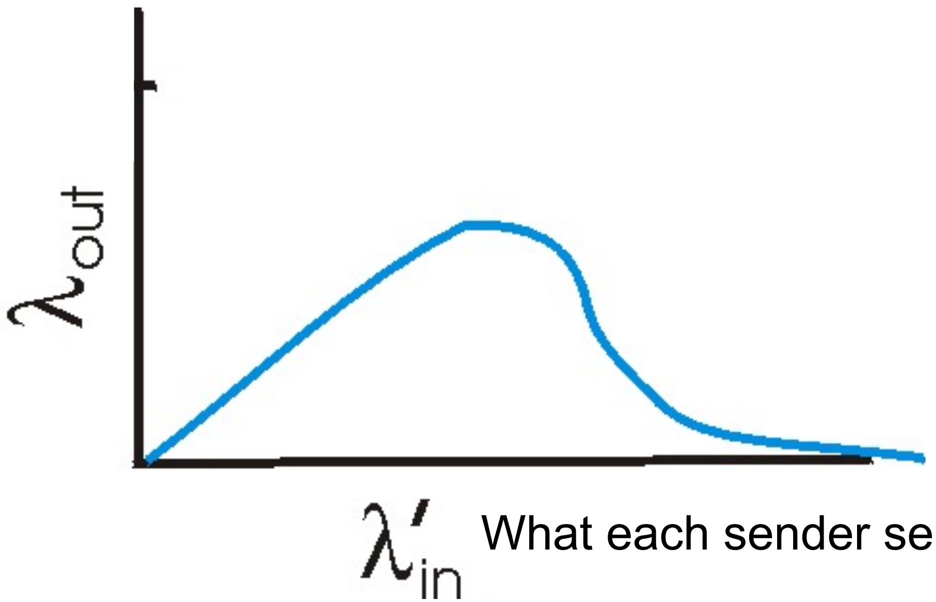


λ'_{in} What each sender sends

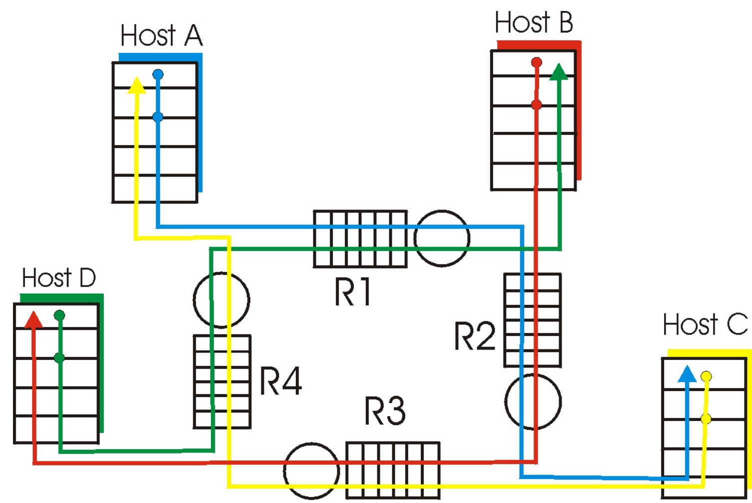


What in fact happens..

What each receiver gets



What each sender sends



- When packet dropped, any “upstream transmission capacity used for that packet was wasted!
- Congestion Collapse: *Increase in network load results in decrease of useful work done*
  - Actually observed in practice.

Approaches Towards Congestion Control

- End-end congestion control:
  - No explicit feedback from network
  - Congestion inferred by end-systems
  - Approach taken by TCP
- Network-assisted congestion control:
  - Routers provide feedback to end systems
    - DECbit, TCP/IP ECN
  - Routers employ clever scheduling algorithms
  - Problem: makes routers complicated

Basic TCP Control Model

- Reduce speed when congestion is perceived
  - How much to reduce?
- Increase speed otherwise
  - Probe for available bandwidth – how?

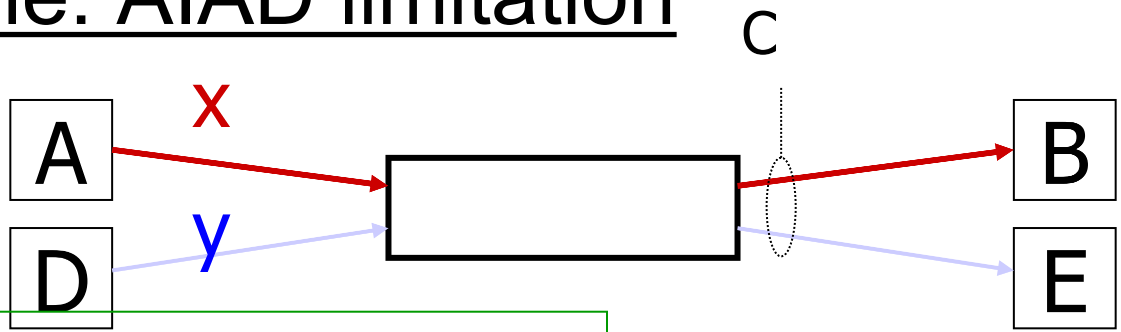
TCP: Objectives

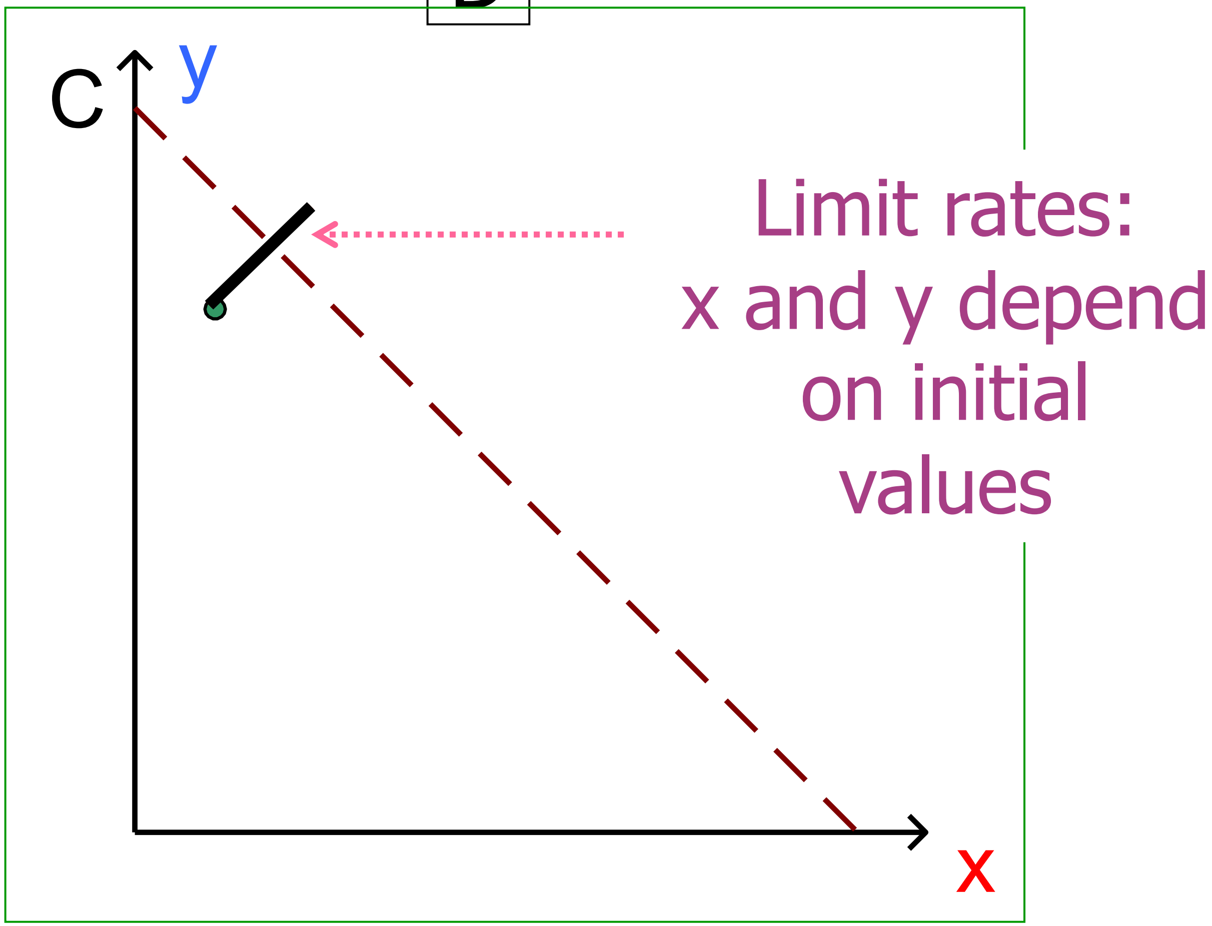
- Simple router behavior
- Distributed
- Efficiency
- Fairness
- Convergence: system must be stable

Adjusting TCP transmission rates

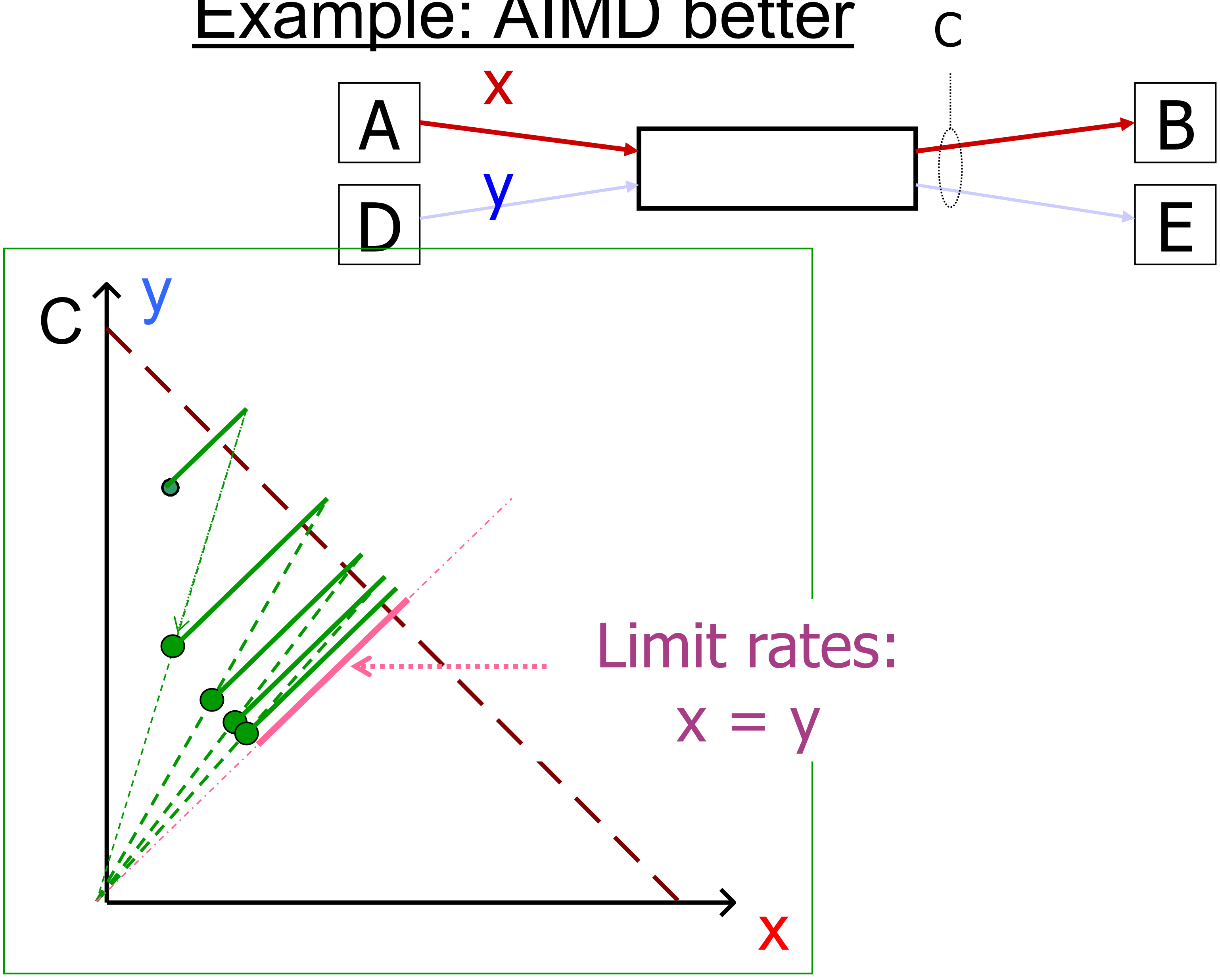
- Many different possibilities for reaction to congestion and probing
  - Additive schemes:
    - Window(t + 1) = a + Window(t)
  - Multiplicative schemes:
    - Window(t + 1) = b Window(t)
- Different combinations possible. E.g.,
  - AIAD: Additive Increase Additive Decrease
  - AIMD: Additive Increase Multiplicative Decrease
- Analysis has shown that AIMD schemes better
  - Better convergence to fair and efficient solutions.

Example: AIAD limitation





Example: AIMD better



TCP Congestion Control

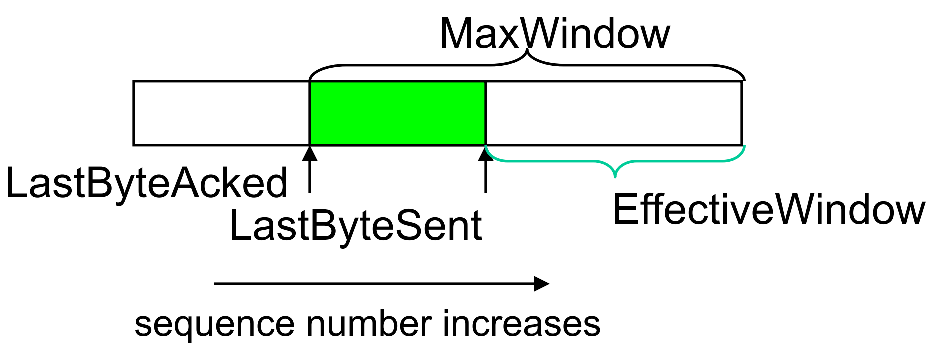
- TCP connection has window
  - controls number of unacknowledged packets
- Sending rate: ~Window/RTT
- Vary window size to control sending rate
- Introduce a new parameter called congestion window (cwnd) at the sender
  - Congestion control is mainly a sender-side operation

Congestion Window (*cwnd*)

- Limits how much data can be in transit

MaxWindow = min(cwnd, AdvertisedWindow)

EffectiveWindow = MaxWindow – (LastByteSent – LastByteAcked)



Detecting Congestion

- Detected based on packet drops
  - Alternative: delay-based methods
- How do you detect packet drops? ACKs
  - TCP uses ACKs to signal receipt of data
  - ACK denotes last contiguous byte received
    - actually, ACKs indicate next segment expected
- Two signs of packet drops
  - No ACK after certain time interval: time-out
  - Several duplicate ACKs (used in later versions of TCP)
- May not work well for wireless networks, why?

TCP's Basic Congestion Control Algorithm

ECE 50863 – Computer Network Systems

TCP Rate Adjustment

- Basic structure:
  - Upon receipt of ACK (of new data): increase rate
    - Data successfully delivered, perhaps can send faster
  - Upon detection of loss: decrease rate
- Adjust rate by controlling the congestion window size (cwnd)

Adapting cwin

- How to know the best cwnd (and best transmission rate)?

- Phases of TCP congestion control

1. Slow start (getting to equilibrium)
   1. Want to find this very very fast and not waste time
2. Congestion Avoidance
   - Additive increase - gradually probing for additional bandwidth
   - Multiplicative decrease - decreasing cwnd upon loss/timeout

Phases of Congestion Control

- **Congestion Window** (**cwnd**)
  Initial value is 1 MSS (=maximum segment size) counted as bytes
- **Slow-start threshold Value** (**CongestionThreshold = congthresh)**
  Initial value is the advertised window size

- **slow start** (cwnd < congthresh)
- **congestion avoidance** (cwnd >= congthresh)

TCP: Slow Start

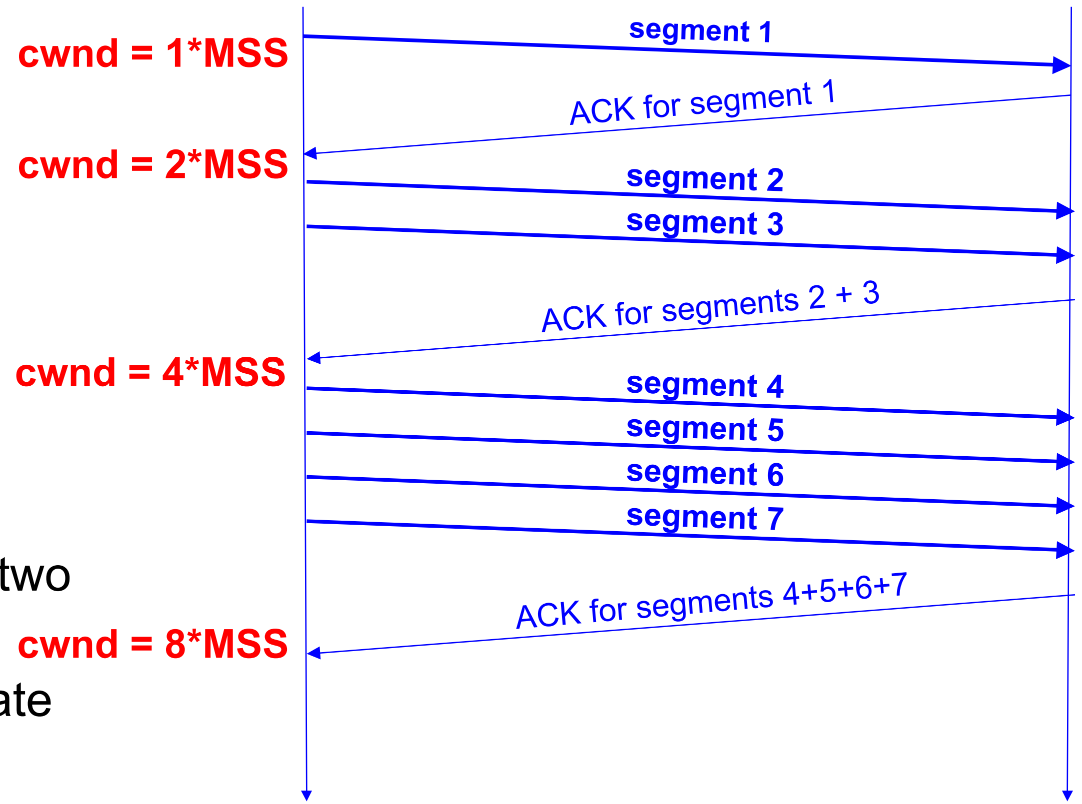
- Goal: discover roughly the proper sending rate quickly

- Whenever starting traffic on a new connection, or whenever increasing traffic after congestion was experienced:
    - Intialize *cwnd* =1 MSS
    - Each time a segment is acknowledged, increment *cwnd* by one MSS (*cwnd* += 1 * MSS).

- Continue until
  – Reach congthresh
  – Packet loss

Slow Start Illustration

- The congestion window size grows very rapidly
- TCP slows down the increase of *cwnd* when ***cwnd >= congthresh***
- Observe:
  - Each ACK generates two packets
  - slow start increases rate exponentially fast (doubled every RTT)!



Congestion Avoidance (After Slow Start)

- Slow Start figures out roughly the rate at which the network starts getting congested
- Congestion Avoidance continues to react to network condition
  - Probes for more bandwidth, increase cwnd if more bandwidth available
  - If congestion detected, aggressive cut back cwnd

Congestion Avoidance: Additive Increase

- After exiting slow start, slowly increase cwnd to probe for additional available bandwidth
  - Competing flows may end transmission
  - May have been “unlucky” with an early drop

- **If** *cwnd > congthresh* **then**
  - Each time a segment is acknowledged, increment *cwnd* by
    - *MSS * (MSS/cwnd)*

- *cwnd* is increased by one MSS only if all segments have been acknowledged
  - Increases by 1 MSS per RTT, vs. doubling per RTT

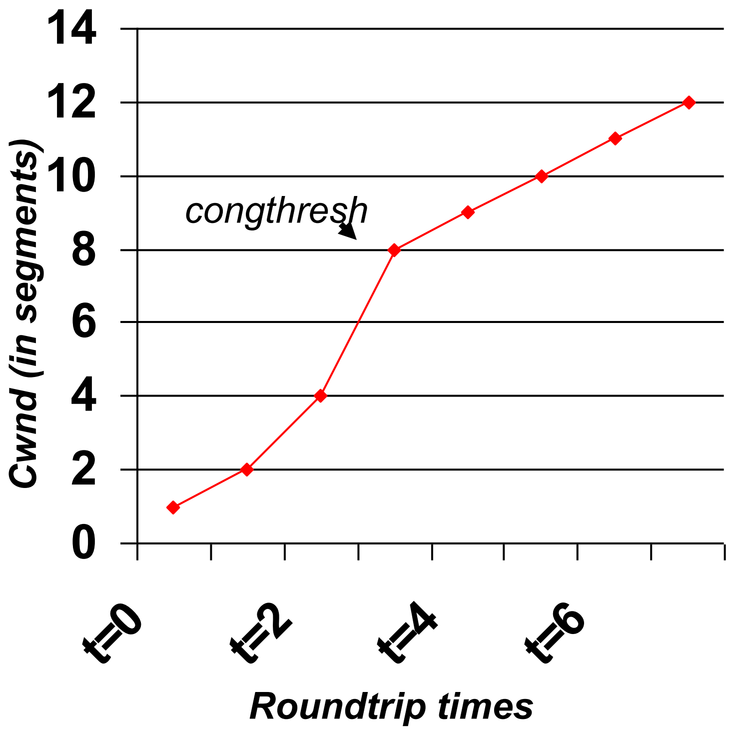
Intuition

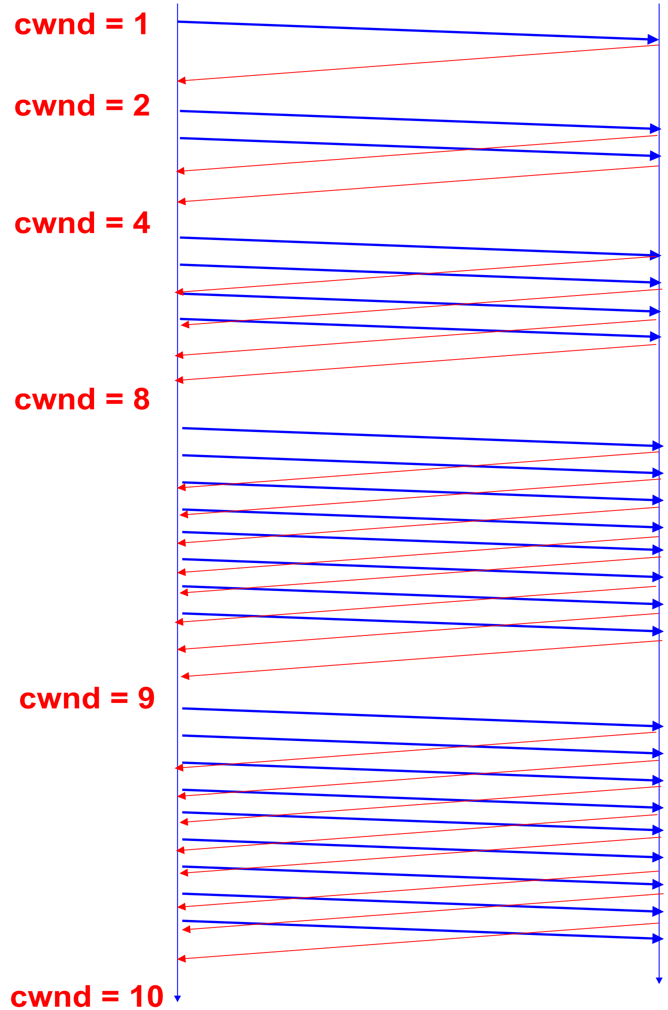
- cwnd is implemented in terms of bytes
  - Lets refer to it as cwnd_b
- More intuitive to think of it in terms of segments
  - Define cwnd_s = cwnd_b / MSS
- Intepreting adjustments in terms of segments
  1) cwnd_b = cwnd_b + (MSS) * (MSS)/ cwnd_b
  2) (cwnd_b)/MSS = (cwnd_b/MSS) + MSS/cwnd_b
  3) cwnd_s = cwnd_s + 1/cwnd_s
- Effectively, in congestion avoidance:
  - cwnd_s increases by 1/cwnd_s for each ACK, or by 1 in each RTT (assuming all ACKs received)
  - cwnd_b increases by 1 MSS over each RTT

Example of Slow Start + Congestion Avoidance

Assume that *congthresh = 8*

(cwnd in units of MSS)





Detecting Congestion via Timeout

- If there is a packet loss, the ACK for that packet will not be received
- The packet will eventually timeout
  - No ack is seen as a sign of congestion

Congestion Avoidance: Multiplicative Decrease

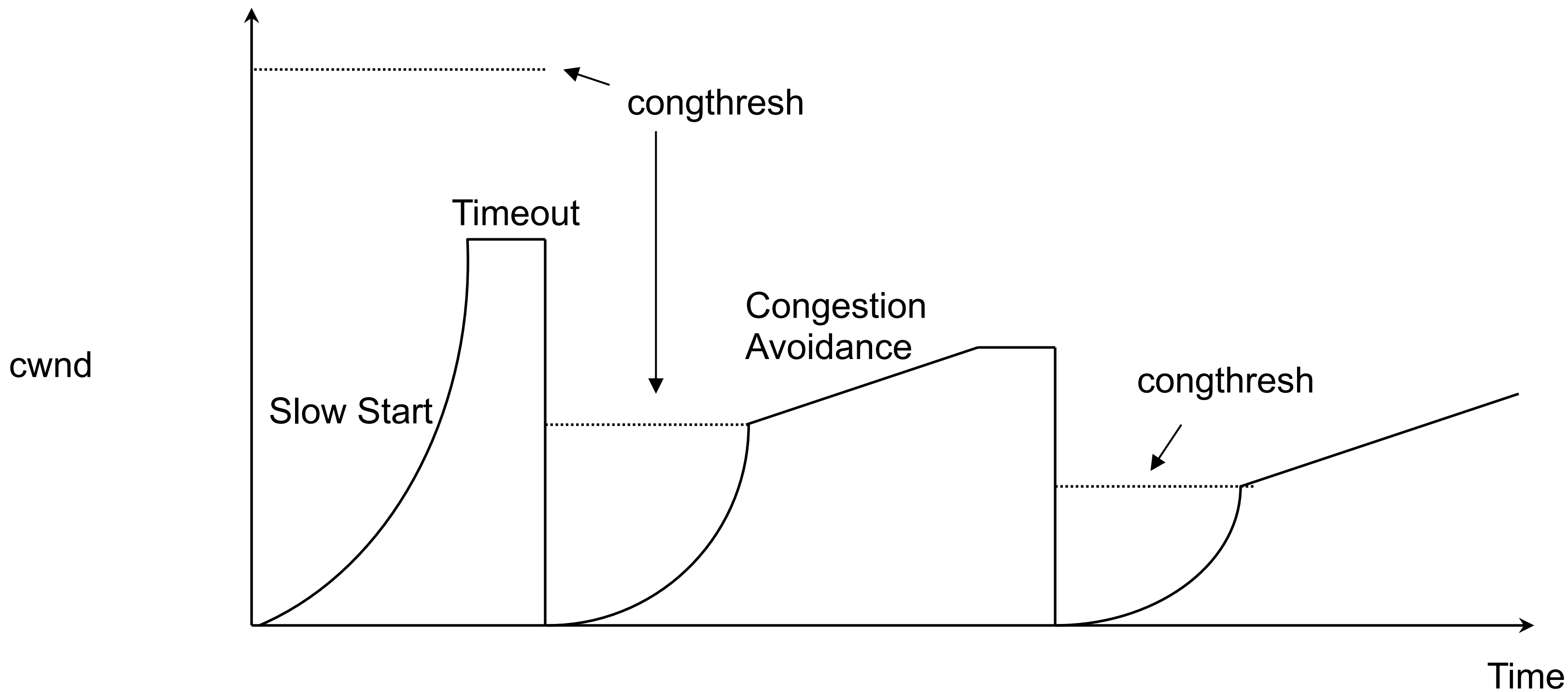
- Timeout = congestion

- Each time when congestion occurs,
  - congthresh is set to half the current size of the congestion window:

    congthresh = cwnd / 2
  - cwnd is reset to one MSS:

    cwnd = 1 * MSS
  - and slow-start is entered

TCP Illustration



TCP Congestion Control Optimizations: Tahoe Vs. Reno

ECE 50863 – Computer Network Systems

Responses to Congestion (Loss)

- Algorithms developed for TCP to respond to congestion
  - **TCP Tahoe** - the basic algorithm (discussed previously)
  - **TCP Reno** - Tahoe + fast retransmit & fast recovery
- And many more:
  - TCP Vegas (use timing of ACKs to avoid loss)
  - TCP SACK (selective ACK)
  - TCP Cubic

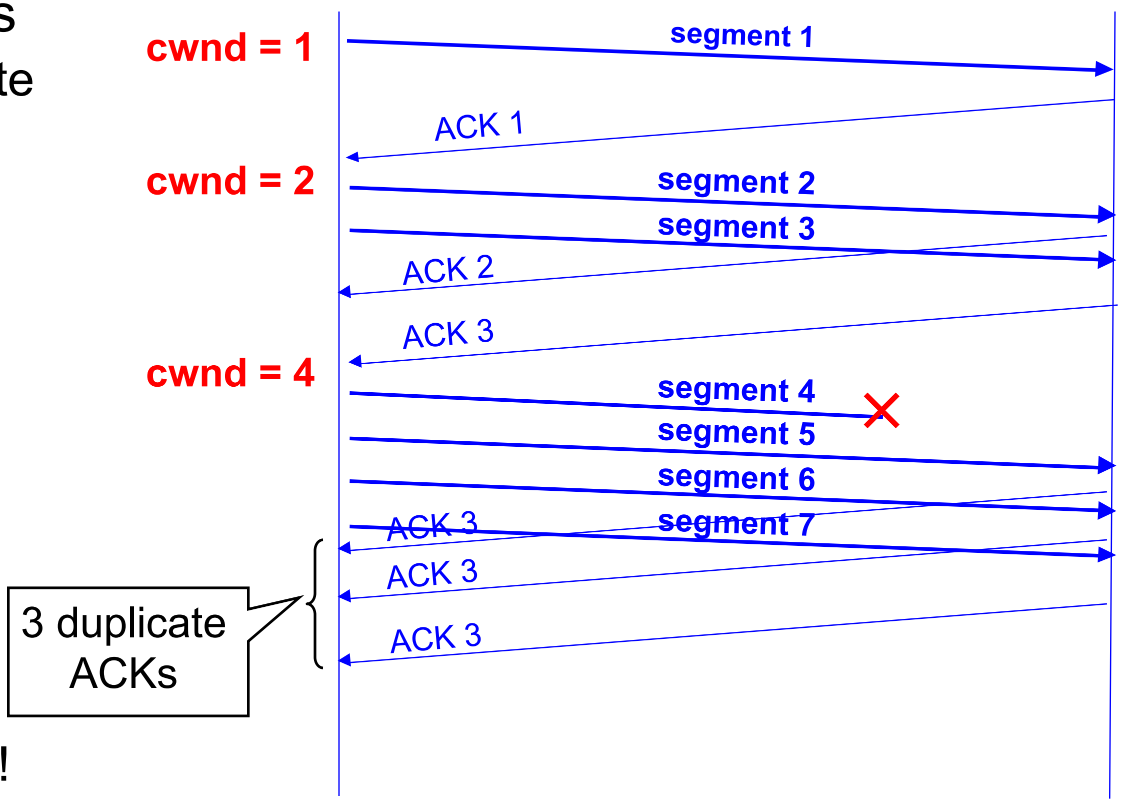
TCP Reno

- Problem with Tahoe: If a segment is lost, there is a long wait until timeout
- Reno adds a **fast retransmit** and **fast recovery mechanism**

- Upon receiving 3 duplicate ACKs, retransmit the presumed lost segment (“fast retransmit”)
- But do not enter slow-start. Instead enter congestion avoidance (“fast recovery”)

Fast Retransmit

- Resend a segment after 3 duplicate ACKs
  - remember a duplicate ACK means that an out-of sequence segment was received
  - ACK-n means packets 1, …, n all received



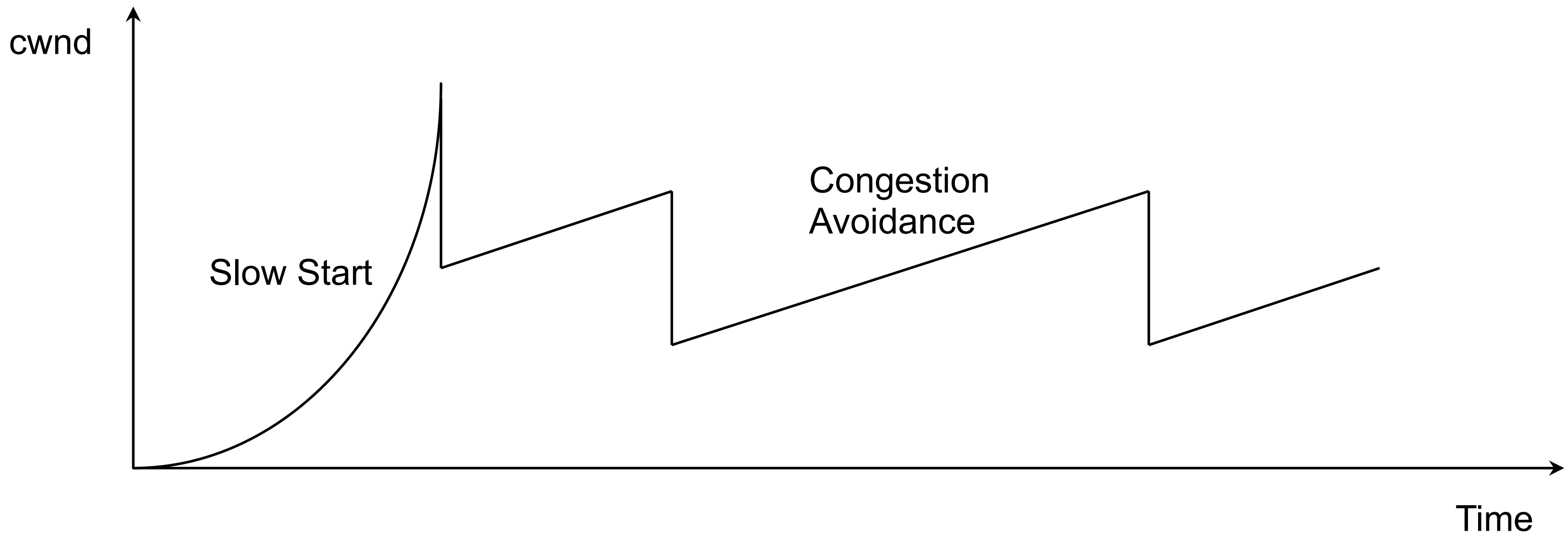
- Notes:
  - duplicate ACKs due to packet reordering!
  - if window is small don't get duplicate ACKs!

Fast Recovery

- After a fast-retransmit
  - congthresh = cwnd/2
  - cwnd = cwnd/*2* (vs. 1 in Tahoe)
  - i.e. starts congestion avoidance at new cwnd
    - Not slow start from cwnd = 1 * MSS

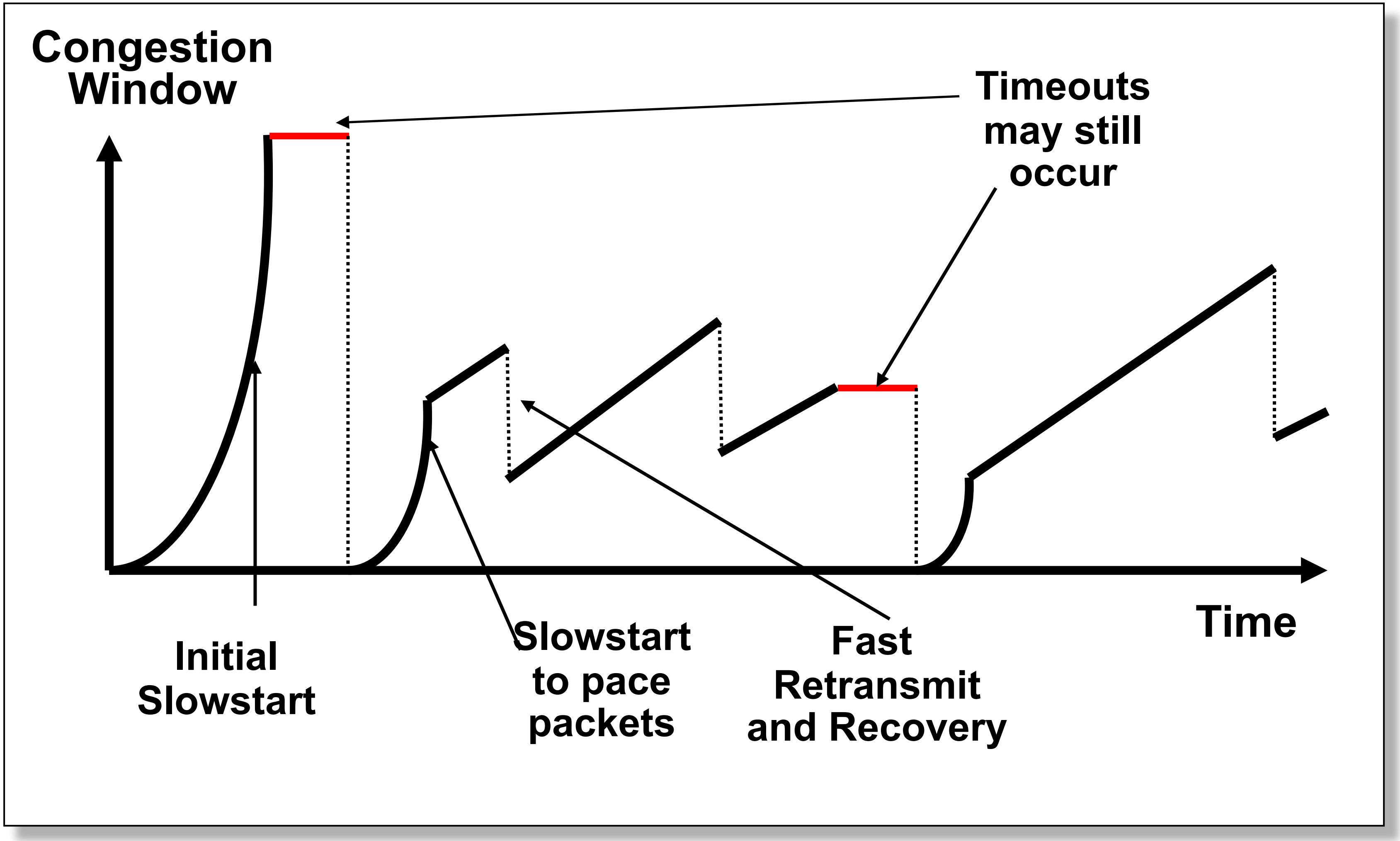
- After a timeout
  - congthresh = *cwnd/2*
  - cwnd = 1 * MSS
  - Do slow start
  - Same as Tahoe

Fast Retransmit and Fast Recovery



- Retransmit after 3 duplicate ACKs
  - prevent expensive timeouts
- Slow start only once per session (if no timeouts)
- In steady state, *cwnd* oscillates around the ideal window size.

TCP Reno Saw Tooth Behavior



TCP Reno Quick Review

- Slow-Start if cwnd < congthresh
  - cwnd+= 1 MSS upon every new ACK (exponential growth)
  - Timeout: congthresh = cwnd/2 and cwnd = 1 MSS
- Congestion avoidance if cwnd >= congthresh
  - Additive Increase Multiplicative Decrease (AIMD)
  - ACK: cwnd = cwnd + MSS * MSS/cwnd
  - Timeout: congthresh = cwnd/2 and cwnd = 1 * MSS
- Fast Retransmit & Recovery
  - 3 duplicate ACKS (interpret as packet loss)
  - Retransmit lost packet
  - congthresh=cwnd/2, cwnd=cwnd/2.