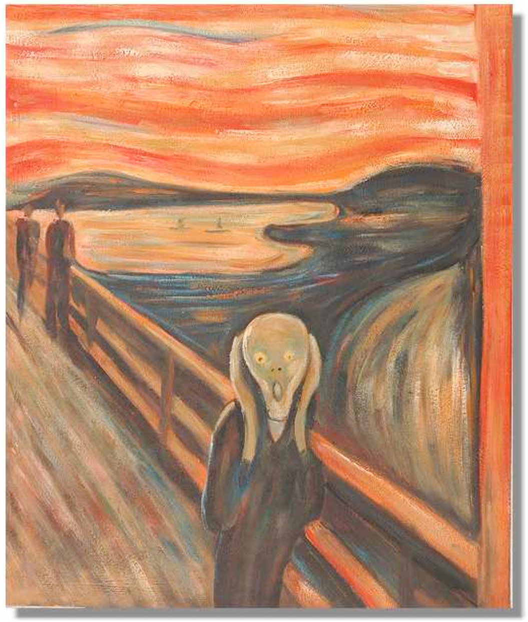
CMSC 332 Computer Networks

TCP: Congestion Control

Professor Szajda

Announcements

- Project 2 has been posted. It will take time
  - Form your groups and get started soon!



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

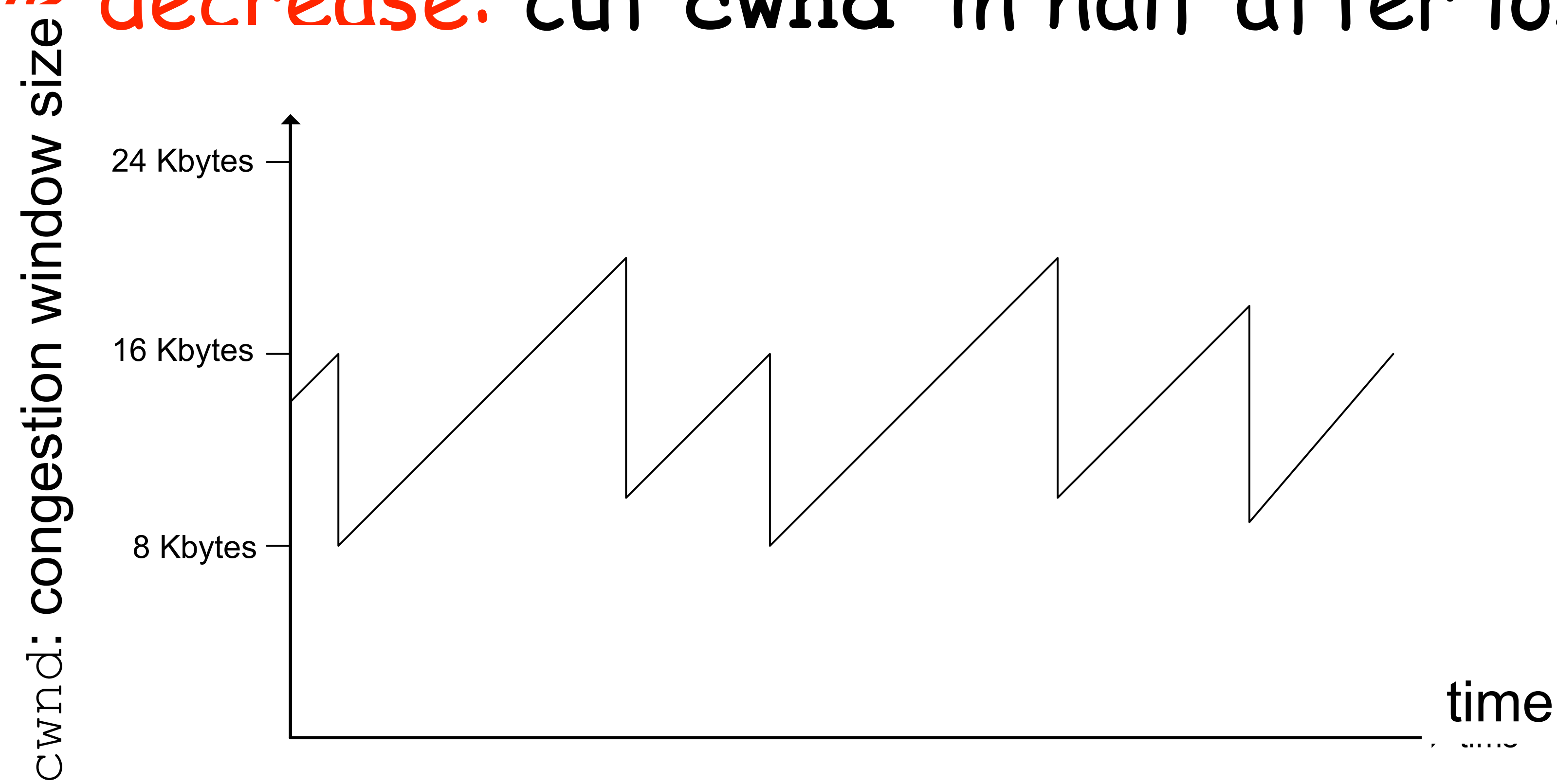
3.6 Principles of congestion control

3.7 TCP congestion control

TCP congestion control: additive increase, multiplicative decrease

- approach: increase 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

saw tooth behavior: probing for bandwidth



TCP Congestion Control: details

- sender limits transmission:

  `LastByteSent-LastByteAcked ≤ cwnd`

- roughly,

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

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

How does sender perceive congestion?

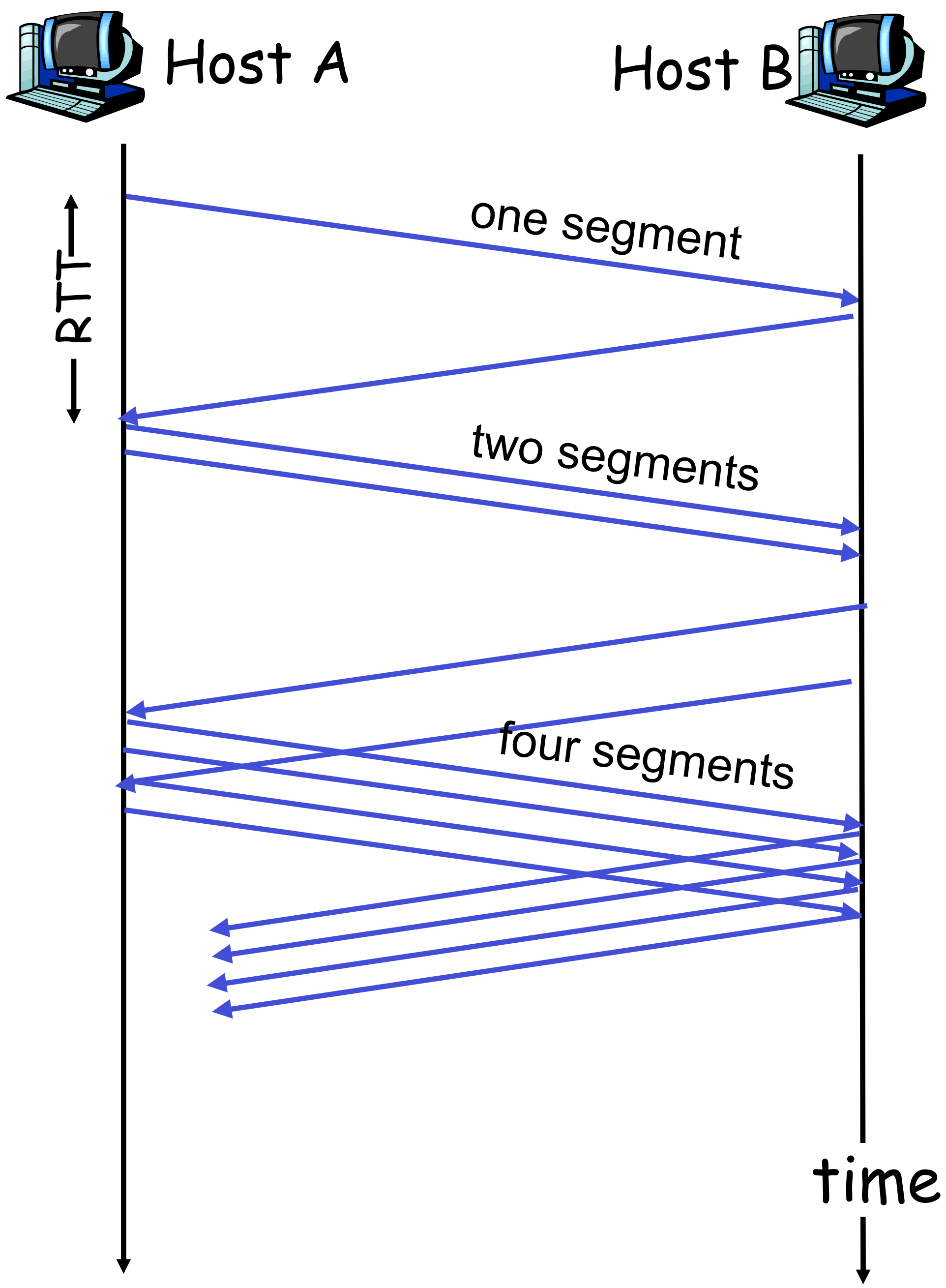
- loss event = timeout or 3 duplicate acks
- TCP sender reduces rate (**cwnd**) after loss event

three mechanisms:

- AIMD
- slow start
- conservative after timeout events

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



Refinement: inferring loss

- after 3 dup ACKs:
  - **cwnd** is cut in half
  - window then grows linearly
- but after timeout event:
  - **cwnd** instead set to 1 MSS;
  - window then grows

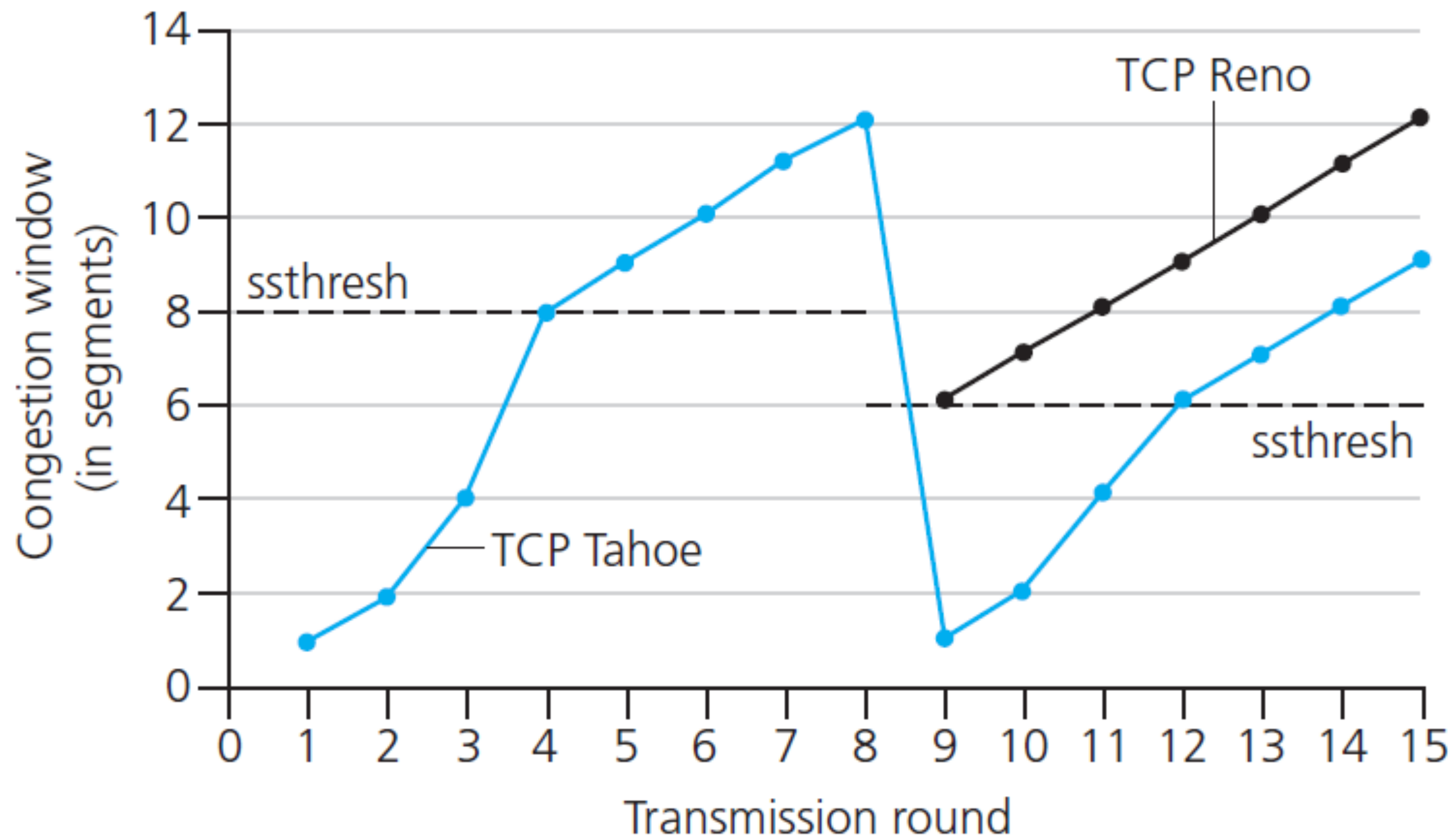
Philosophy:

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

Refinement

Q: when should the exponential increase switch to linear?

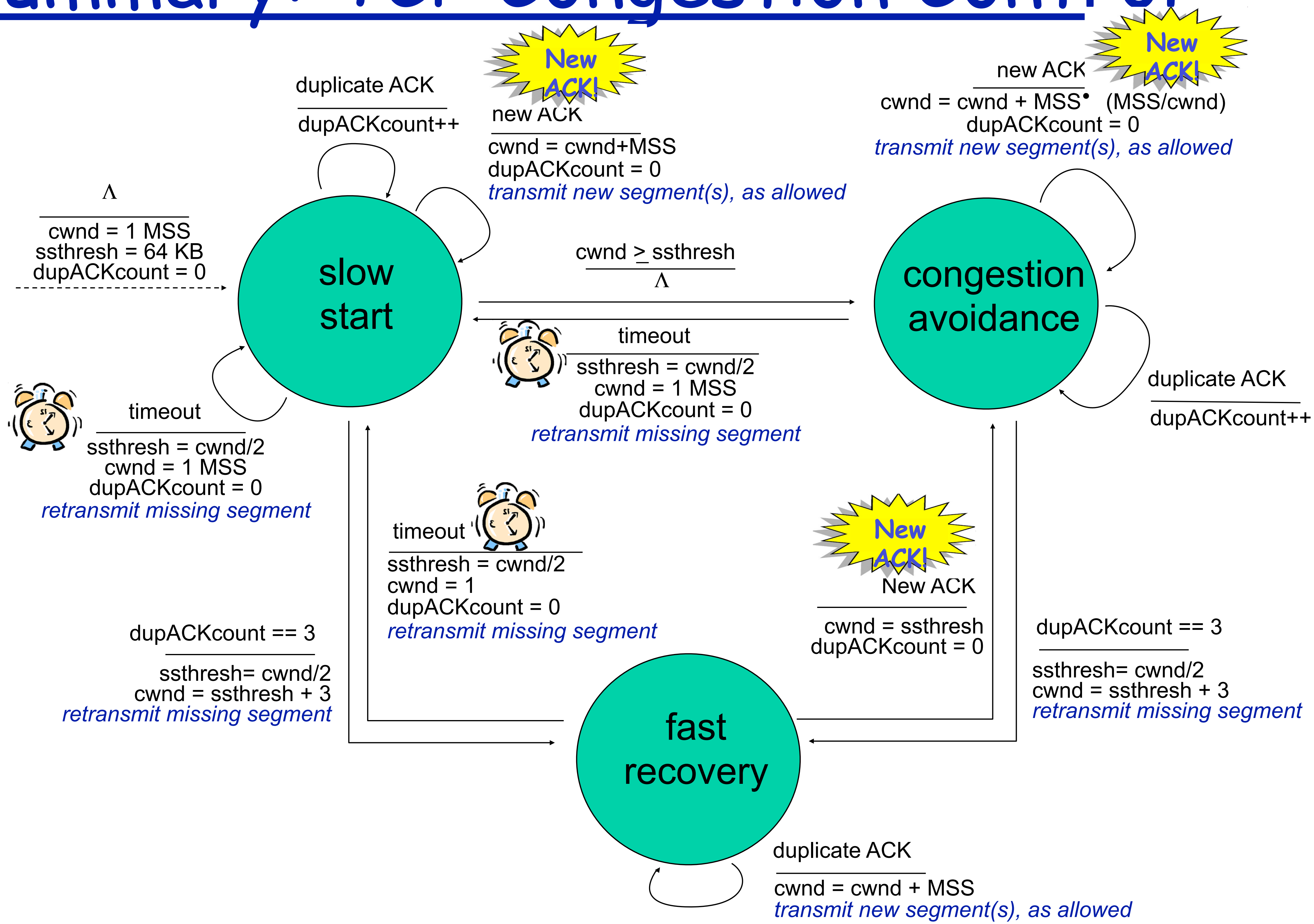
A: when **cwnd** gets to 1/2 of its value before timeout.



Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event

Summary: TCP Congestion Control



TCP throughput

- what's the average throughout of TCP as a function of window size and RTT?
  - ignore slow start
- let W be the window size when loss occurs.
  - when window is W, throughput is W/RTT
  - just after loss, window drops to W/2, throughput to W/2RTT.
  - average throughout: .75 W/RTT

TCP Futures: TCP over "long, fat pipes"

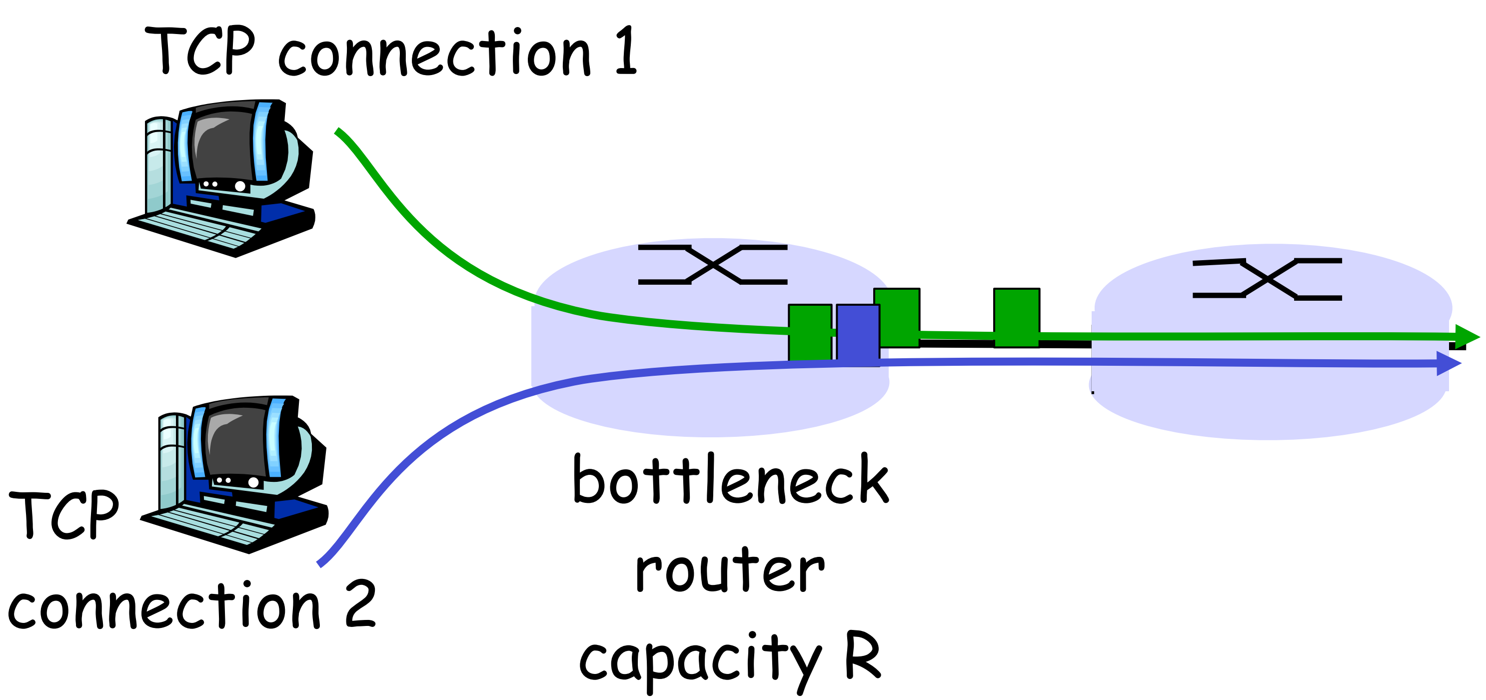
- example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- requires window size W = 83,333 in-flight segments
- throughput in terms of loss rate:

$$\frac{1.22 \times MSS}{RTT\sqrt{L}}$$

- ➜ $L = 2 \cdot 10^{-10}$ Wow – a very small loss rate!
- new versions of TCP for high-speed

TCP Fairness

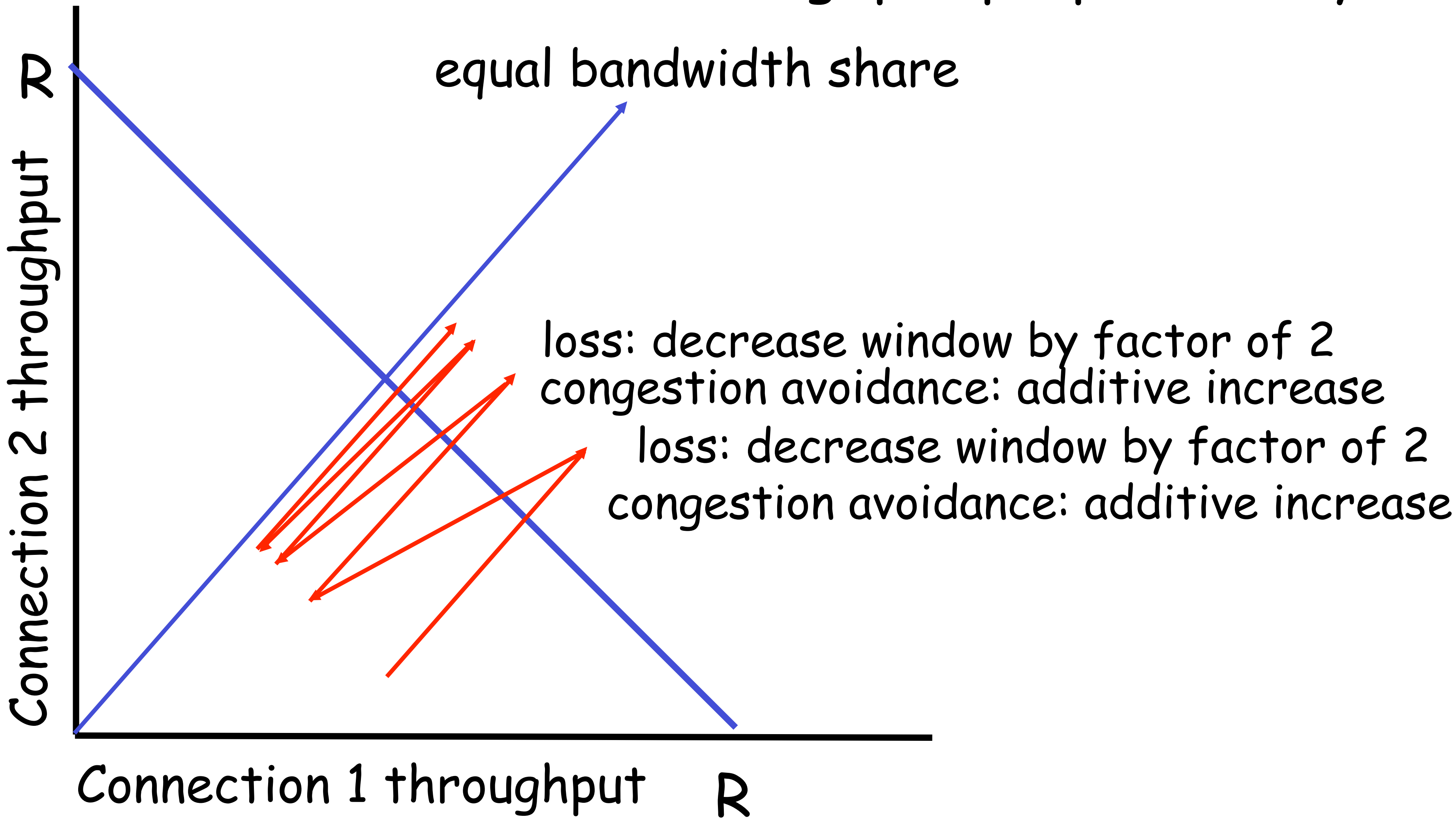
fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



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:
  - pump audio/video at constant rate, tolerate packet loss

Fairness and parallel TCP connections

- nothing prevents app from opening parallel connections between 2 hosts.
- web browsers do this
- example: link of rate R supporting 9 connections;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2 !

Chapter 3: Summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation and

Next:

- leaving the network "edge" (application, transport layers)
- into the network "core"