Lesson 05-04: TCP Congestion Control

CS 356 Computer Networks

Mikyung Han

mhan@cs.utexas.edu

Internet Reference Model

Example Protocols		Responsible for
FTP, HTTP, SMTP	Application	application specific needs
TCP, UDP	Transport	process to process data transfer
IP	Network	host to host data transfer across different network
Ethernet, WiFi	Link	data transfer between physically adjacent nodes
802.3 PHY	Physical	bit-by-bit or symbol-by-symbol delivery

2

Outline

1. Approaches to Congestion Control

Congestion control has 2 approaches

- **First, solely based on sender's detection**
  - Loss-based: Increase sending rate until a loss (timeout) and then cut back
  - Delay-based: Do the same until RTT reaches $RTT_{congested}$

- **Second, network assisted approach**
  - Sender, network core (routers), and the receiver all participates

Let's first look at the loss-based approach!

- AIMD
- TCP CUBIC

Outline

1. Approaches to Congestion Control
2. TCP CC Basic Principle: AIMD

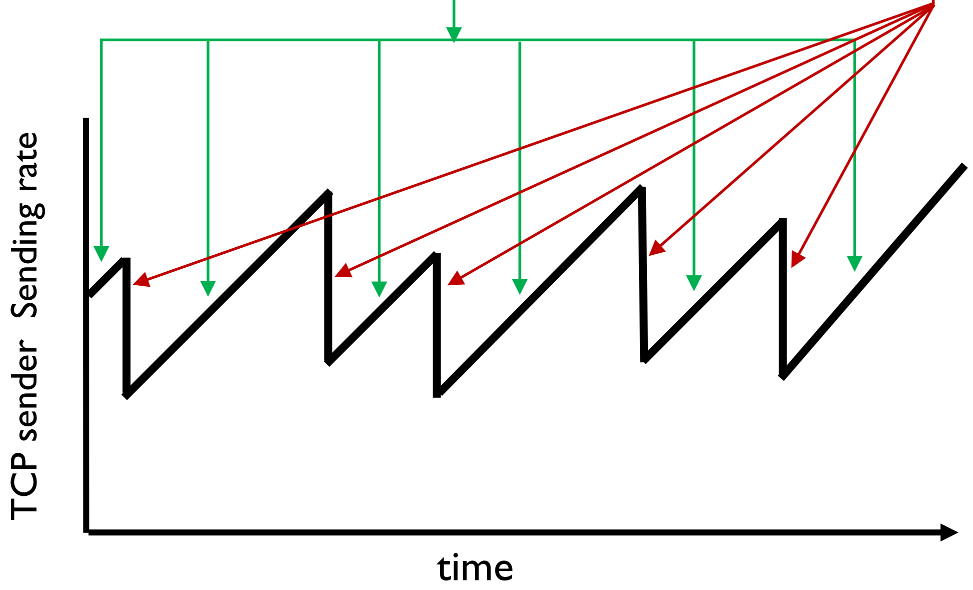
AIMD: sender increases sending rate until packet loss then decrease sending rate on loss

Additive Increase

increase sending rate by 1 MSS every RTT until loss detected

Multiplicative Decrease

cut sending rate in half at each loss event



AIMD sawtooth behavior: probing for bandwidth

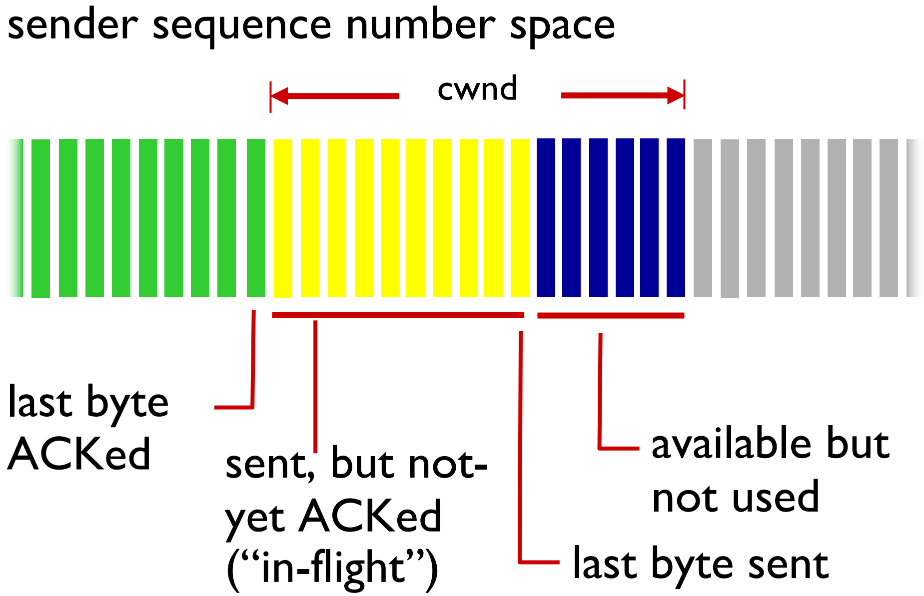
Why AIMD?

- AIMD has been shown to:
  - optimize congested flow rates network wide!
  - have desirable stability properties
  - Does not need coordination among other TCP senders

AIMD is implemented by 2 variables

- **Congestion window (cwnd)**
  - Max bytes TCP sender can send out
  - Additive increase when no loss
- **Slow Start Threshold (ss threshhold)**
  - Upon loss ss threshold is set to half of cwnd
  - Helps with multiplicative decrease

Congestion Window: TCP sending rate is limited by `cwnd`



$$\text{TCP sending rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

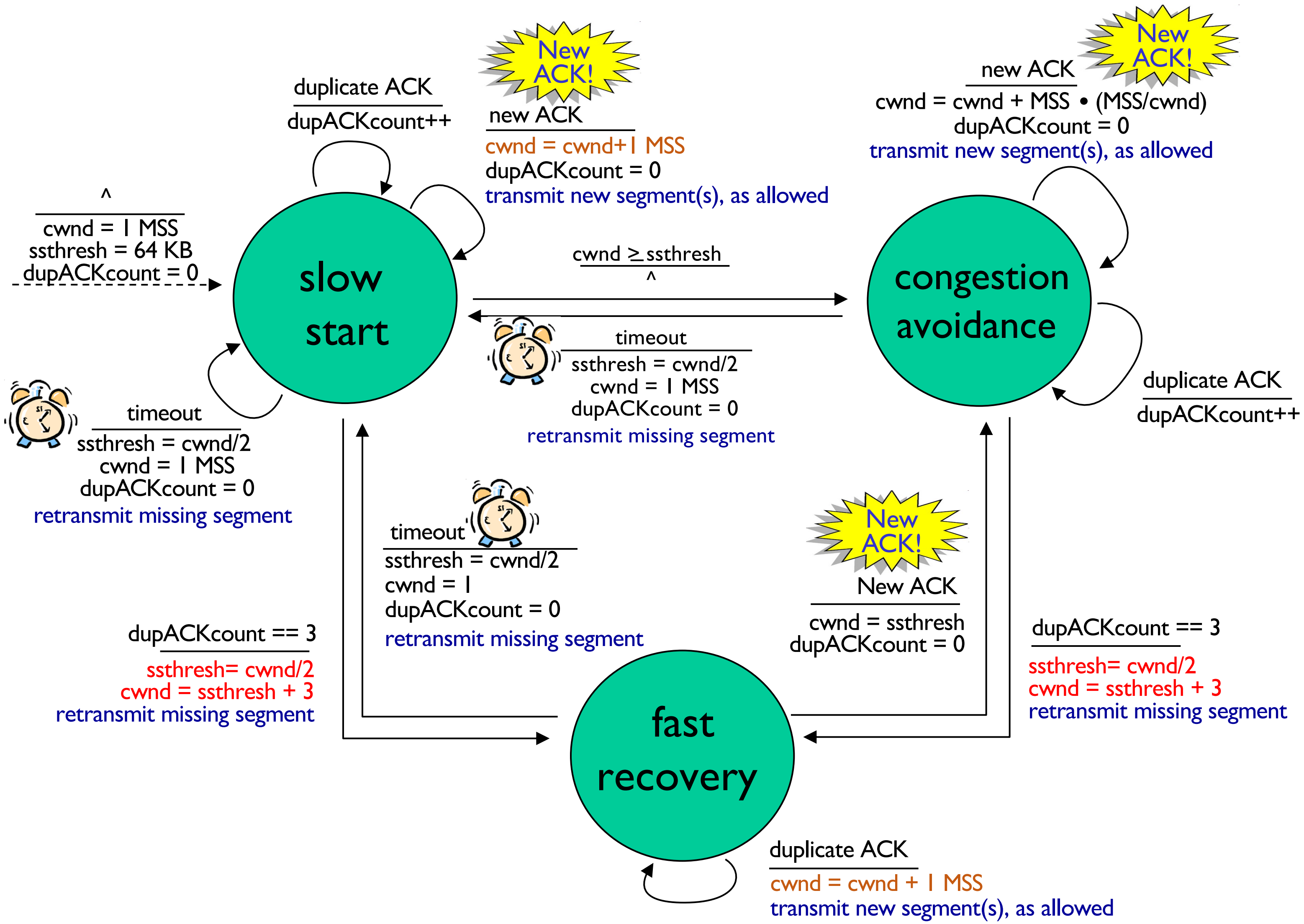
LastByteSent- LastByteAcked $\leq$ cwnd

cwnd is dynamically adjusted in response to observed congestion

Outline

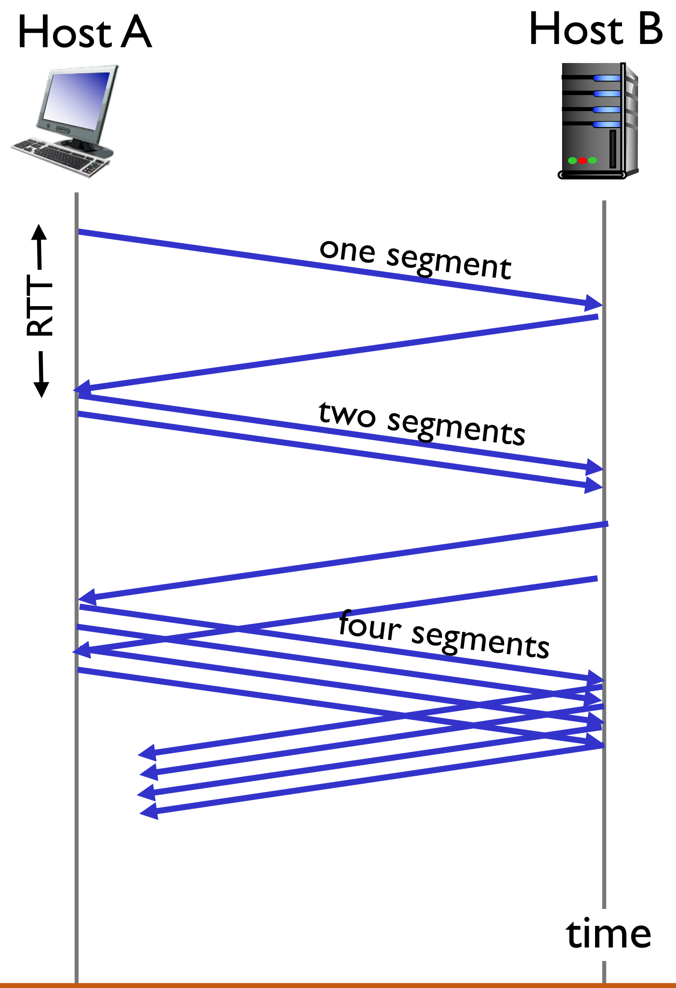
1. Approaches to Congestion Control
2. TCP's AIMD
3. 3 States in TCP Congestion Control

3 states of TCP Congestion Control



TCP slow start is not that slow

- 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

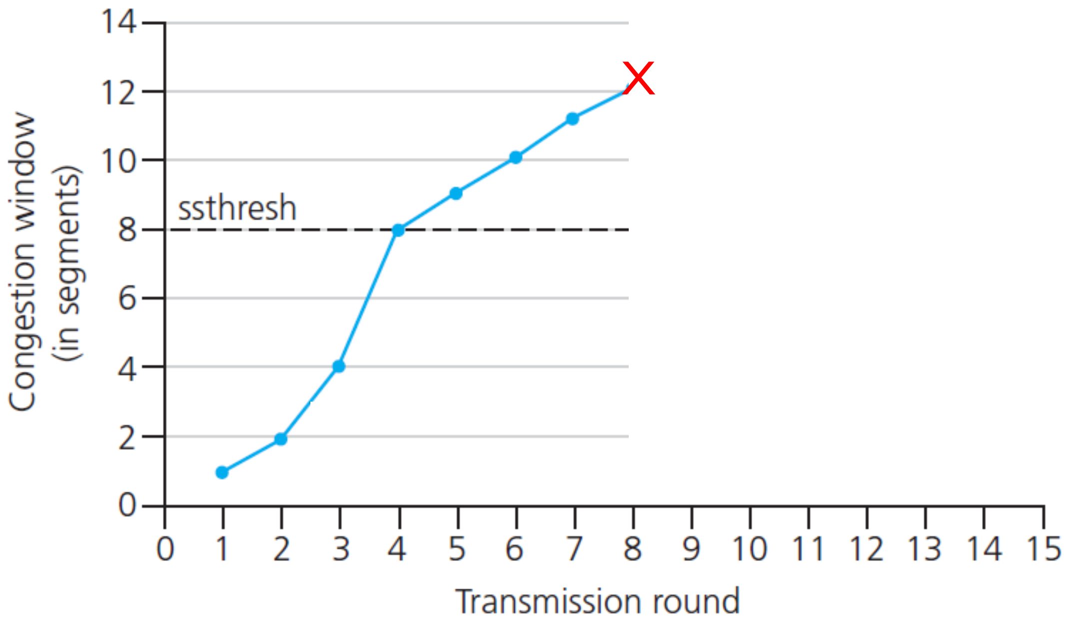


Initial rate is slow but ramps up exponentially fast!

TCP: from slow start to congestion avoidance

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

cwnd < ssthresh implies we are in slow start state

TCP Reno vs TCP Tahoe

- Reno: Cut to roughly half on loss detected by triple duplicate ACK
- Tahoe: Cut to 1 MSS when loss detected (either t-d-ACK or timeout)

Reno implements all 3 states
whereas Tahoe only has 2 states (no fast recovery state)

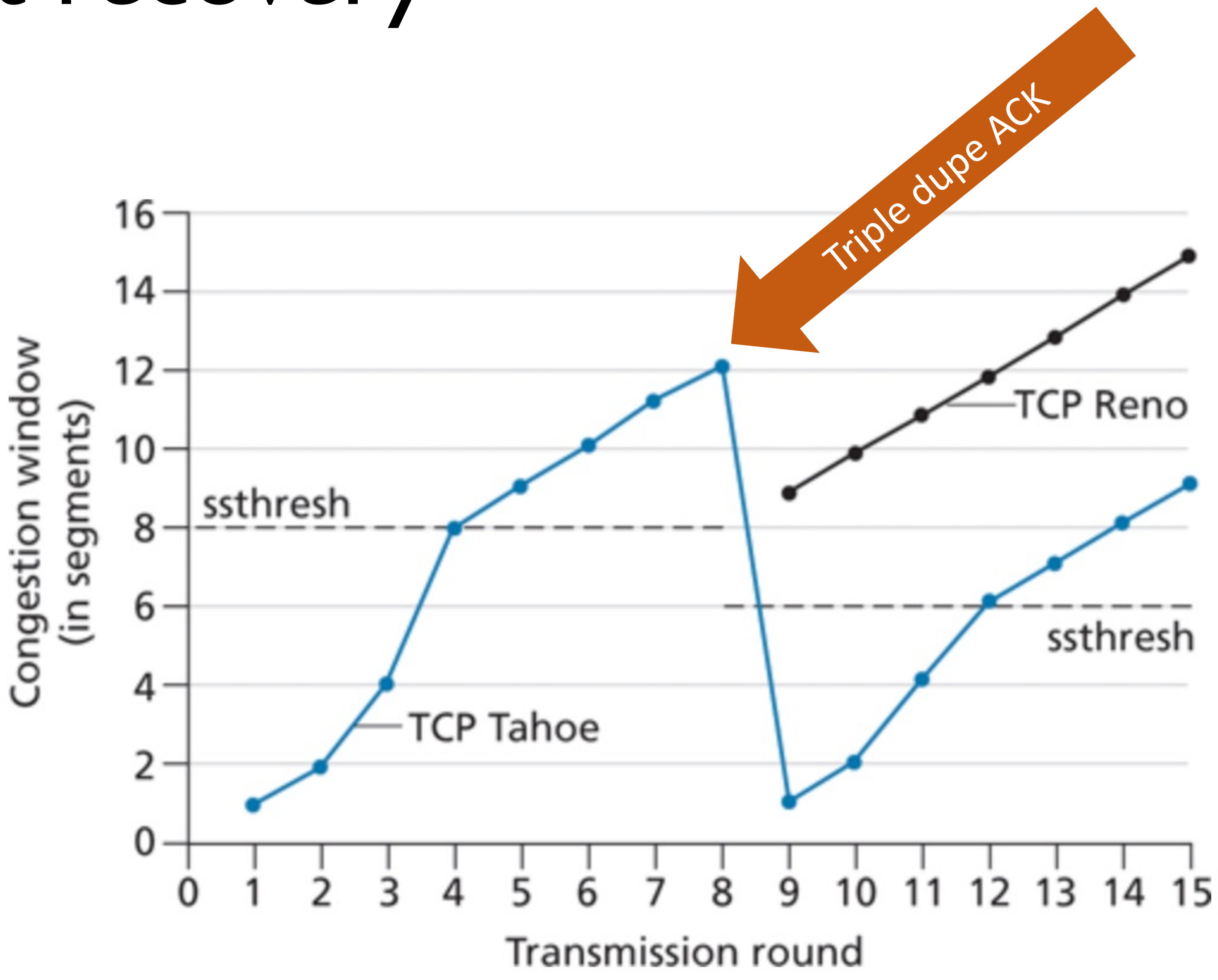
Tahoe vs Reno's fast recovery

Tahoe

- ssthresh = cwnd/2
- cwnd = 1 MSS

Reno

- ssthresh = cwnd/2
- cwnd = ssthresh + 3MSS



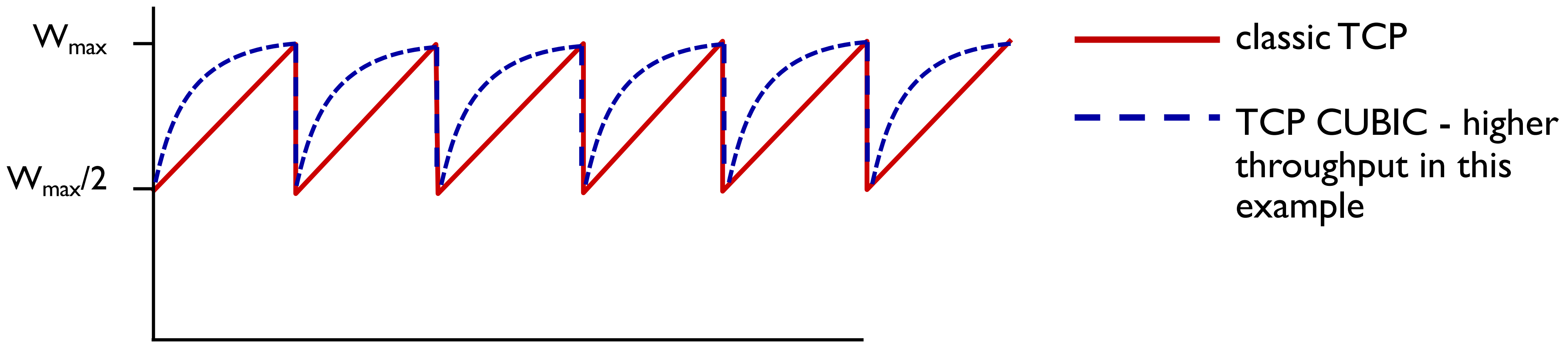
Outline

1. Approaches to Congestion Control
2. 3 States in TCP Congestion Control
3. TCP's AIMD
4. TCP CUBIC

Is there a better way to "probe" available bandwidth?

TCP CUBIC: more aggressive initially but more cautious later with higher probability of loss

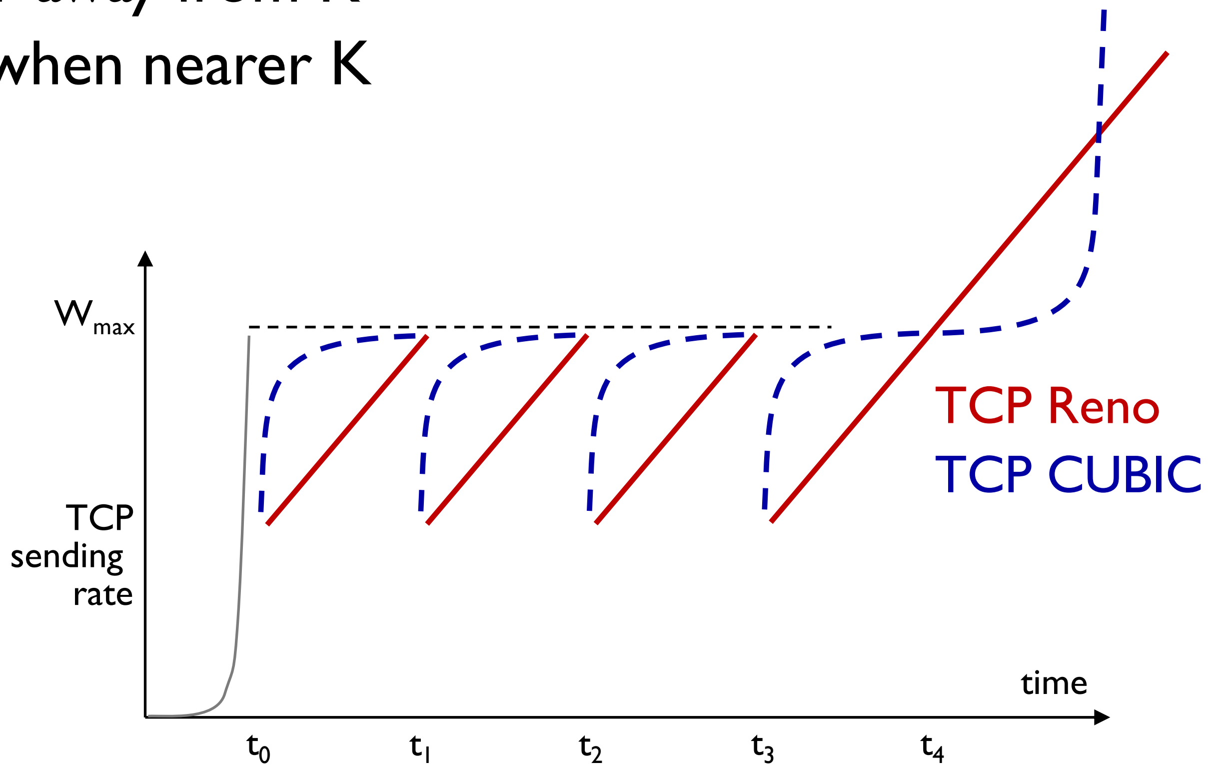
- Insight/intuition:
  - W_{max}: sending rate at which congestion loss was detected
  - congestion state of bottleneck link probably (?) hasn't changed much
  - after cutting rate/window in half on loss, initially ramp to to W_{max} faster, but then approach W_{max} more slowly



TCP CUBIC has higher throughput than Reno

- Tune-able K: point in time when TCP window size will reach W_{max}
- increase W as a function of the cube of |K - current time|
  - larger increases when further away from K
  - smaller increases (cautious) when nearer K

CUBIC is default in Linux, widely used among popular Web servers

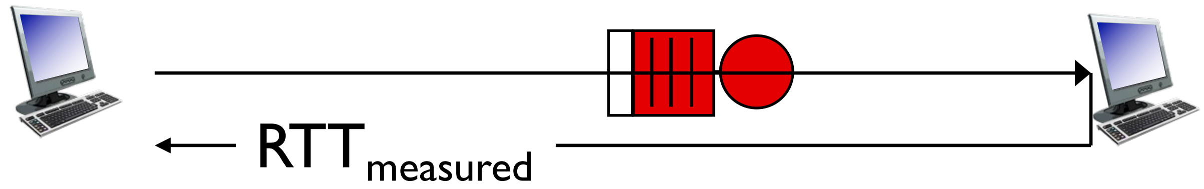


Outline

1. Approaches to Congestion Control
2. 3 States in TCP Congestion Control
3. TCP's AIMD
4. TCP CUBIC
5. Delay-based CC

Delay-based TCP CC monitors throughput

Keeping the pipe "just full enough, but no fuller"



$$\text{Throughput}_{measured} = \frac{\text{\# bytes sent in last RTT interval}}{\text{RTT}_{measured}}$$

- RTT_{min} - minimum observed RTT
- uncongested throughput - $\text{cwnd}/\text{RTT}_{min}$

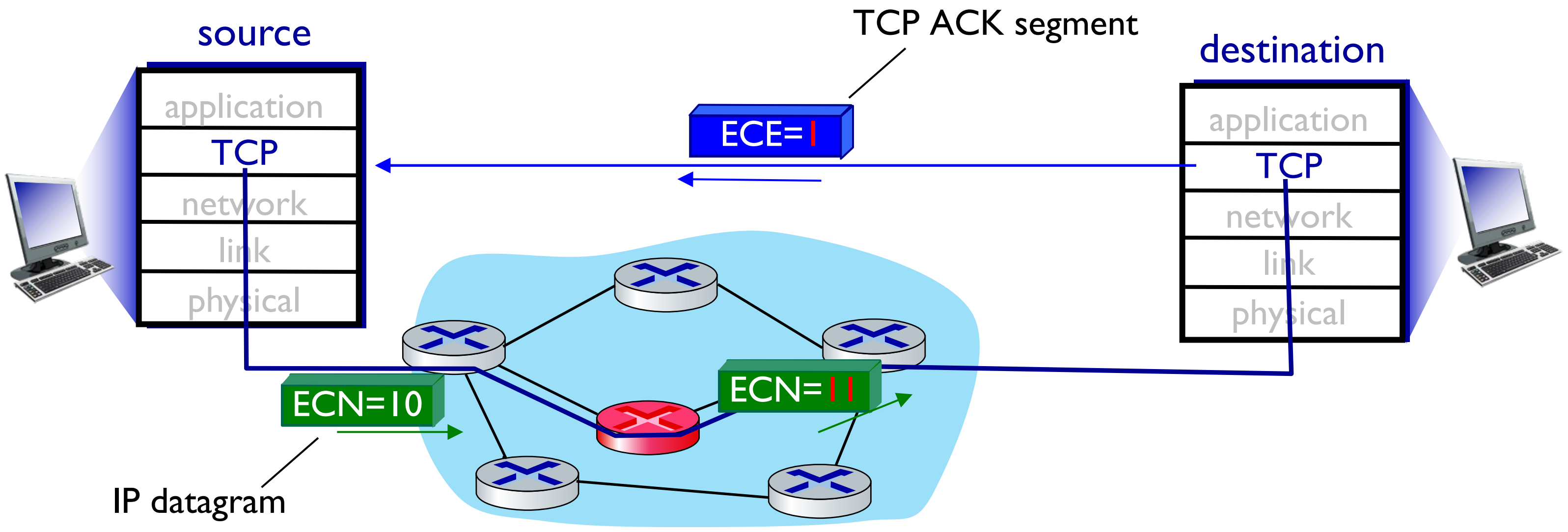
if $\text{Throughput}_{measured}$ "very close" to $\text{cwnd}/\text{RTT}_{min}$ //not congested
    increase cwnd linearly
else if $\text{Throughput}_{measured}$ "far below" $\text{cwnd}/\text{RTT}_{min}$ //congested
    decrease cwnd linearly

Outline

1. Approaches to Congestion Control
2. 3 States in TCP Congestion Control
3. TCP's AIMD
4. TCP CUBIC
5. Delay-based CC
6. Network assisted CC

Network-assisted approach: Explicit congestion notification (ECN)

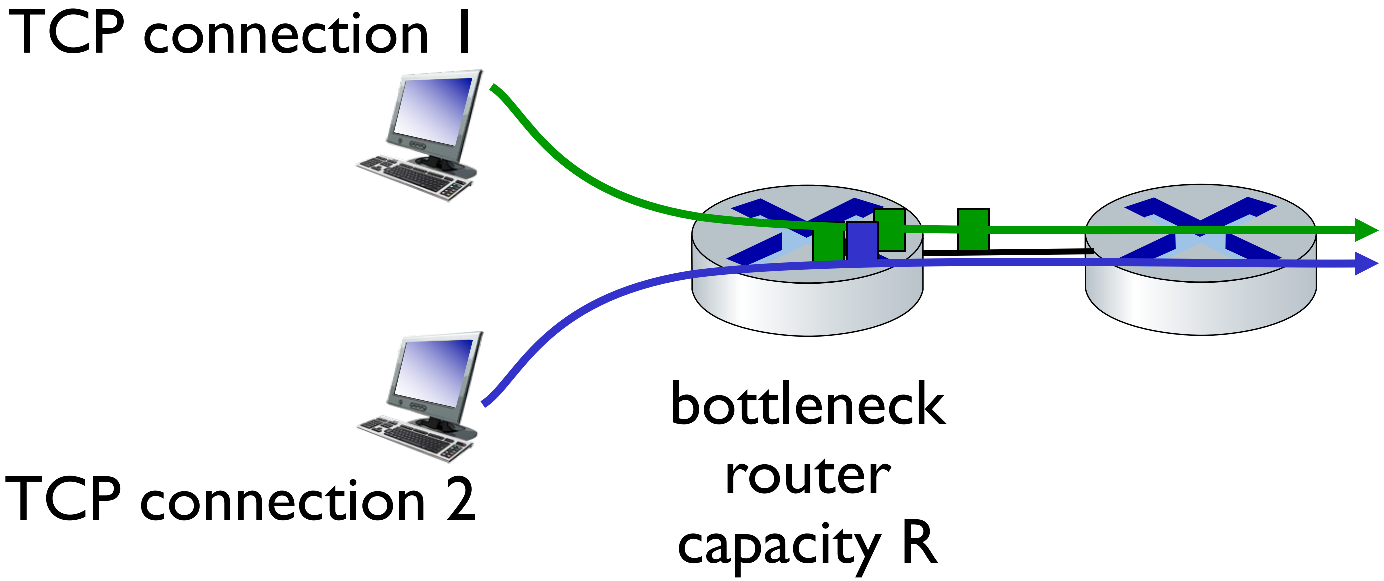
- two bits in IP header (ToS field) marked by network router to indicate congestion
  - policy to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets ECE bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)



Backup slides

TCP fairness

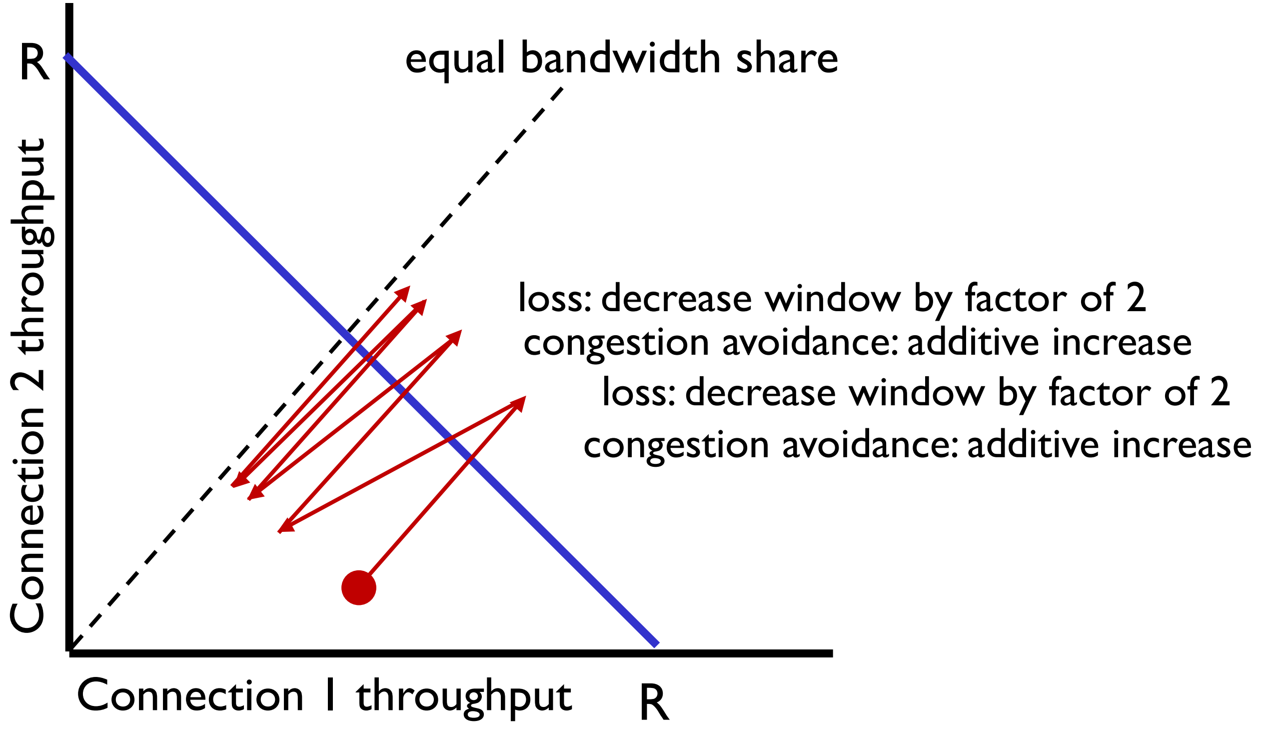
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Q: is TCP Fair?

Example: two competing TCP sessions:

- additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally



Is TCP fair?

A: Yes, under idealized assumptions:

- same RTT
- fixed number of sessions only in congestion avoidance

Fairness: must all network apps be "fair"?

Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss
- there is no "Internet police" policing use of congestion control

Fairness, parallel TCP connections

- application can open multiple parallel connections between two hosts
- web browsers do this , e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2

Acknowledgements

Slides are adopted from Kurose' Computer Networking Slides