Computer Networks
COL 334/672

Congestion Control

Slides adapted from KR

Sem 1, 2025-26

Recap

- Congestion control:
  - Network-assisted vs end-to-end

- Designing an end-to-end congestion control
  - **General approach:** probe the network by increasing the sending rate, decrease when you infer congestion
  - **Additive Increase, Multiplicative Decrease:** TCP fair
  - **Inferring congestion:** use packet loss as an indicator of congestion / other indicator : delays

sending rate controlled using sliding window mechanism

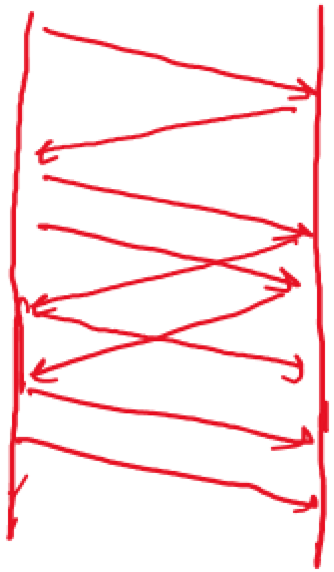
SWS ≤ min (cwnd, rwnd)

cwnd : congestion window (internal to sender)

rwnd : flow control window advertised by receiver

Honing CCA Further

- **One specific instance of AIMD**: Increase sending rate by 1 MSS every RTT, decrease to half on event of congestion
  - □ **When to increase**: on reception of every new ACK, since ACK indicates a packet has be received
  - □ **How much to increase:** MSS*MSS/cwnd



- **What to do in the beginning of the connection?**
  - ○ Linear increase is too slow
  - ○ **Q:** Given a network with RTT of 80ms, initial congestion window of 1 MSS. MSS is 1000 bytes. How long it will take to get to an average rate of 10 Mbps?

$$\text{Rate} = \frac{\text{cwnd}}{\text{RTT}}$$

cwnd = 100 MSS

99 RTTs to increase to 100 MSS if IW = 1 MSS

99 x 0.08 ≃ 8 seconds

What to do in the beginning..

- How to ramp up faster?

OPT-1

Larger initial window (say 100 MSS).

⇒ But it can overwhelm the slower networks

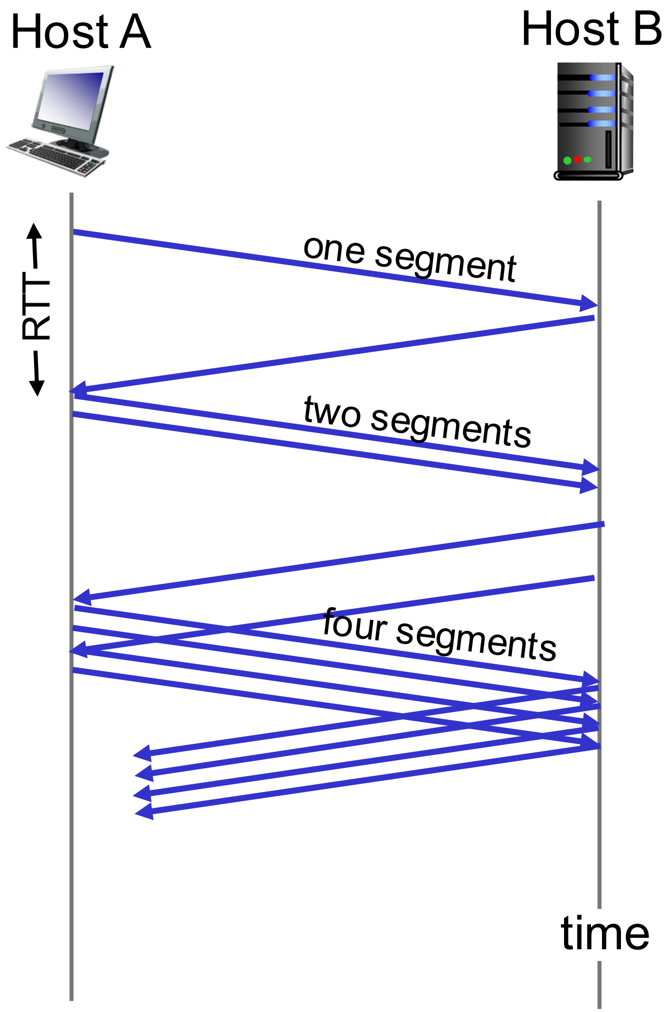
OPT-2

Start with a small window but ramp up faster than linear

- One idea : increase exponentially

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

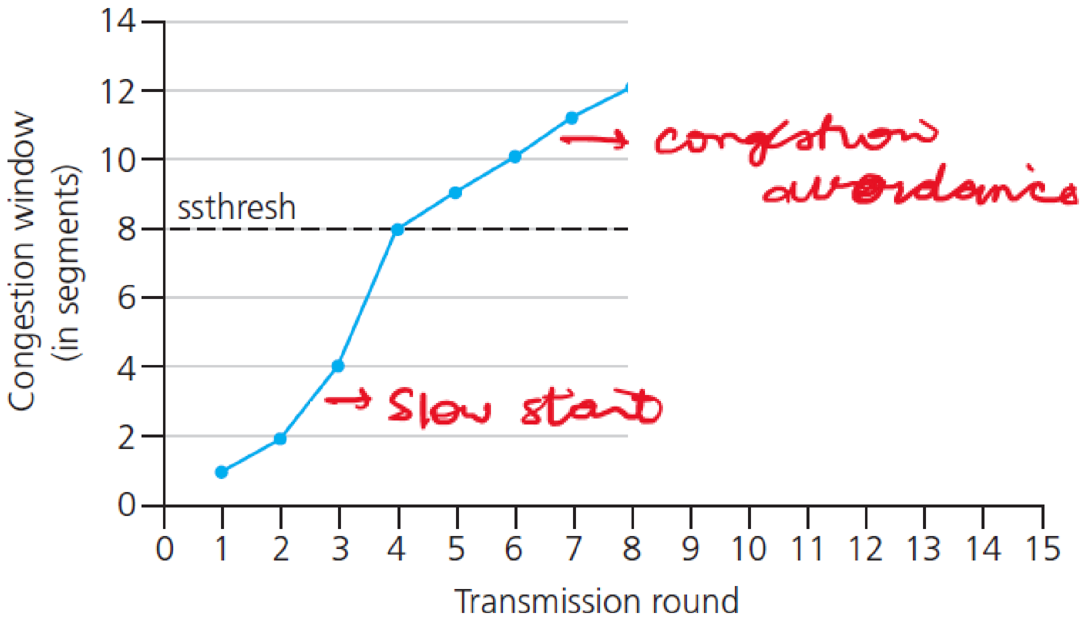


Until when?

TCP Slow Start and Congestion Avoidance

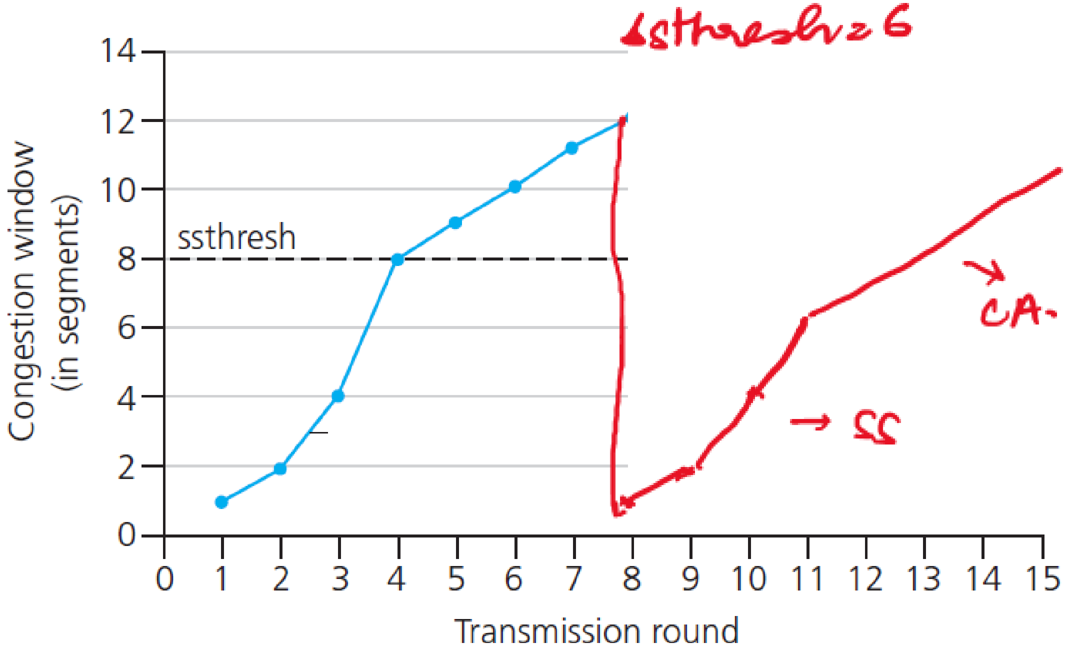
- Uses a threshold, *ssthresh*,after which TCP enters congestion avoidance

- *What happens when a loss occurs?*



What happens when a loss occurs?

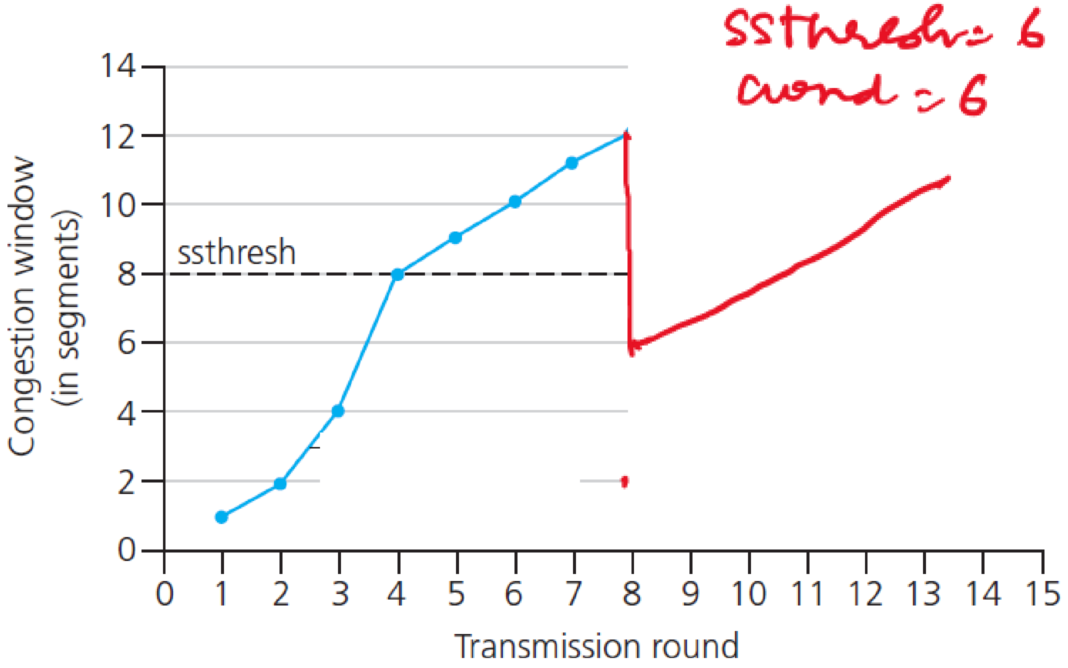
- When loss is due to timeout: severe congestion!!
  - Set *ssthresh to cwnd/2*
  - Reset cwnd to 1
  - Enter slow start phase



What happens when a loss occurs?

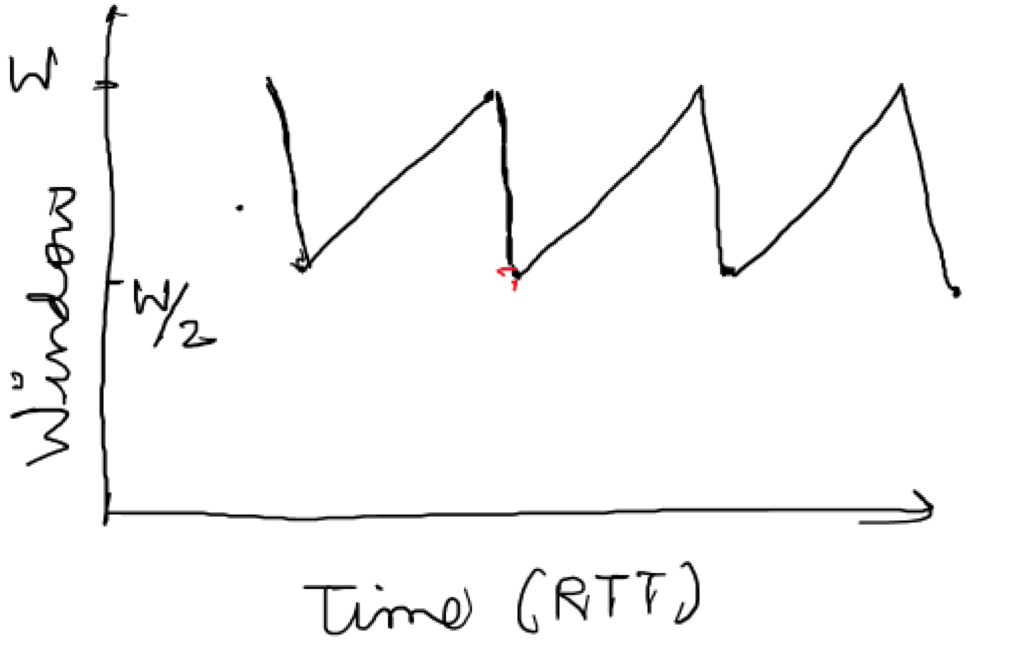
- When loss is due to triple duplicate ACK: congestion is not severe!
  - Set *ssthresh to cwnd/2*
  - On receipt of another duplicate ACK, send 1 new segment
  - Once a new ACK arrives, set *cwnd = ssthresh (or cwnd/2)*
  - Enter congestion avoidance phase

TCP Reno: Fast retransmit, fast recovery!



TCP Reno Throughput: Macroscopic Description

- Throughput: area under the curve



Average window $= \frac{3W}{4}$

Average Tput $= \frac{3W * MSS}{4 RTT}$

HW: derive tput if packet loss probability is p

Still slow if B/w is high or RTT is high

i.e. B/w × RTT is high

↑

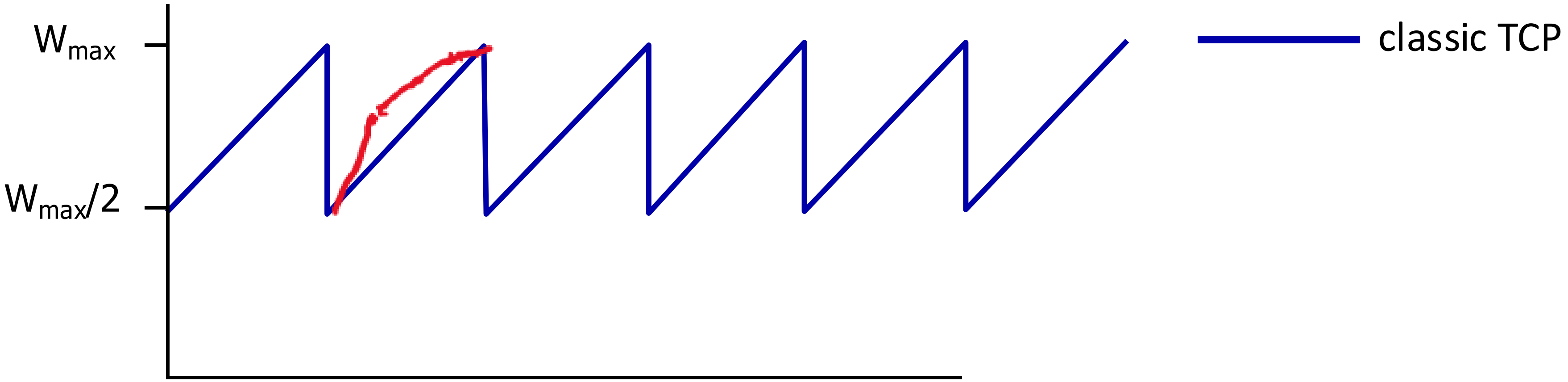
Inefficient for networks with high bandwidth delay product!

Can we do faster?

Is there a better way than AIMD to “probe” for usable bandwidth?

- 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* → Binary search

TCP BIC : Binary Increase Congestion Control



Improving further

- TCP Reno: Window increases by 1 MSS per RTT
- What happens when there are flow with different RTTs?

RTT unfairness

Flow with smaller RTT will ramp up faster ;

TCP CUBIC

- K: point in time when TCP window size will reach W_{max}
  - K itself is tunable
- increase W as a function of the *cube* of the distance between current time and K

$$W(t) = C(t-K)^3 + W_{max}$$

K is the time it takes to reach W_{max}

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}}$$

$W(K) = W_{max}$

$w(0) = (1-\beta) W_{max}$

- TCP CUBIC default in Linux, most popular TCP for popular Web servers

[TCP BBR is replacing CUBIC]