Computer Networks
COL 334/672

Congestion Control

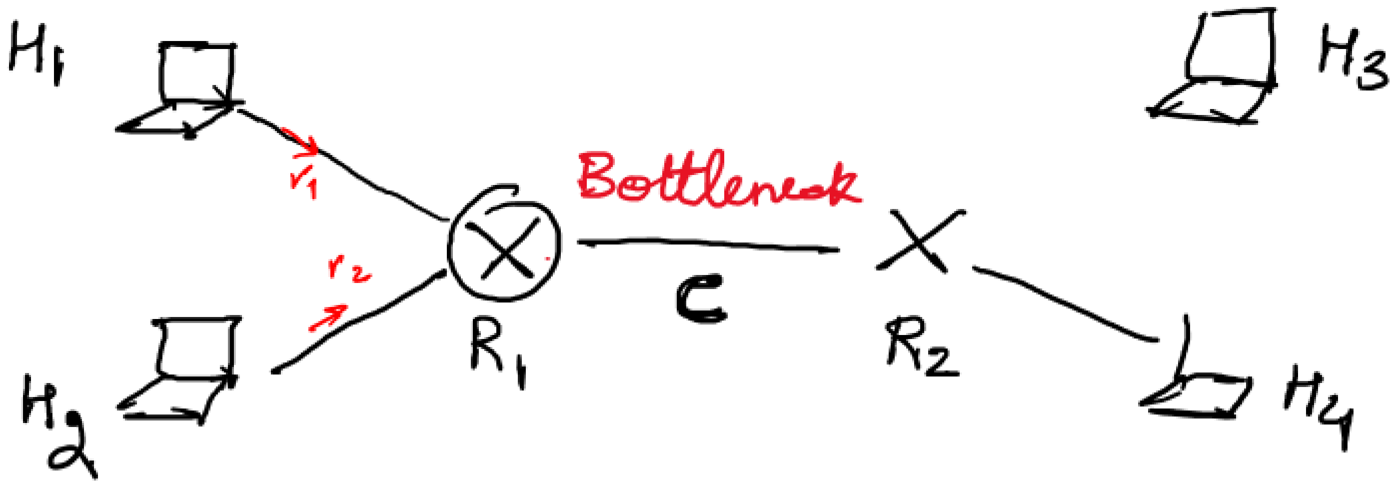
Slides adapted from KR

Sem 1, 2025-26

Quiz on Moodle

Password: reliability

Recap: Congestion Control



$r_1 + r_2 \leq C$

Adapt sending rate such that n/w does not get congested

Approach for designing CCA

① N/w + end hosts

Routers signal end host to slow down when congestion is happening

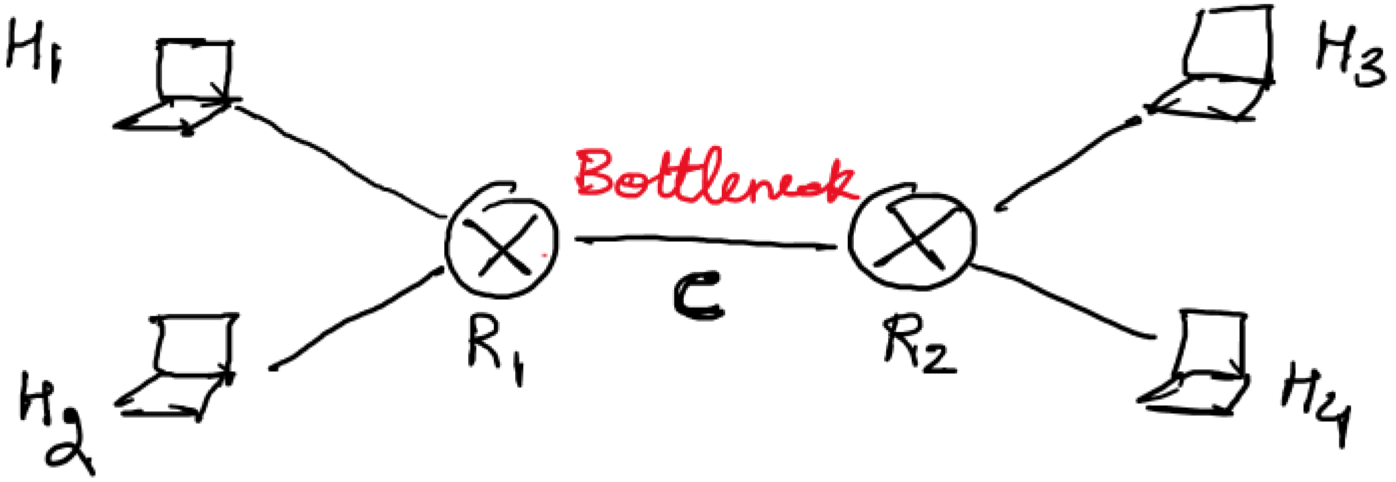
Cons: Increases complexity

END-TO-END PRINCIPLE

End-host only

Each end host independently decide the rate w/o any assistance from the n/w

Metrics for evaluating a Congestion Control Algorithm



(1). High goodput

(2) Low latency

(3) Fairness

Across a diverse set of n/w conditions

Designing an End-to-end Congestion Control Algorithm

Probe the network by increasing the sending rate.

Slow down if you detect congestion

Decrease sending rate

Two Fundamental Questions

1. how to detect congestion

2. how much to increase/decrease

How to increase/decrease the sending rate?

f g

- Many functions are possible
- *Class of linear* control algorithms:
  - $\boxed{w_{i+1} = aw_i + b}$ → a
  - a and b are different in case of increase and decrease
- Additive Increase, Multiplicative Decrease (AIMD)
  - $w_{i+1} = w_i + b$ → increase function $b > 0$
  - $w_{i+1} = aw_i$ → decrease function

    $a < 1$

→ Assuming rate is controlled using congestion window (cwnd)

Congestion control: AIMD

- *approach:* senders can increase sending rate until congestion occurs, then decrease sending rate on congestion

$w_{i+1} \leftarrow w_i + MSS$ (every time unit)

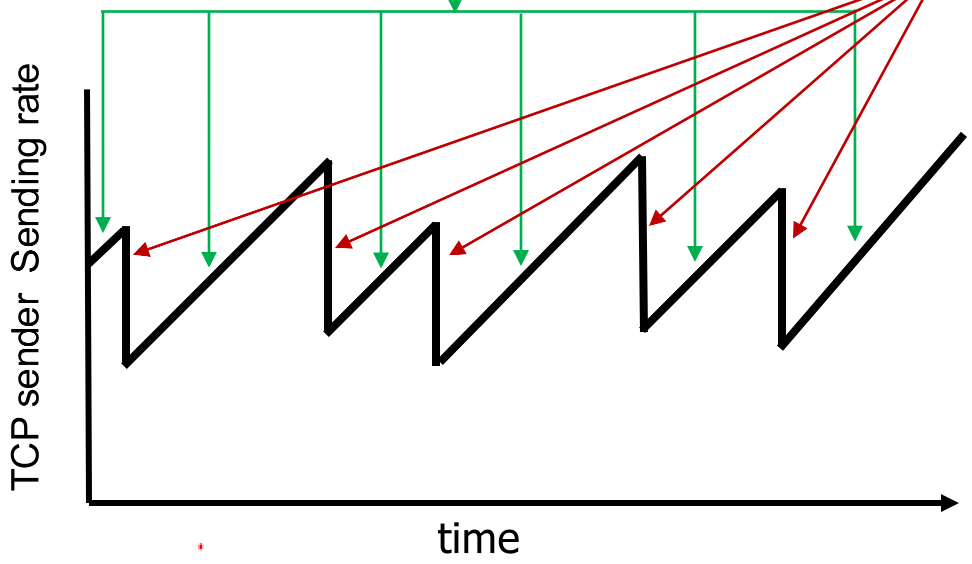
$w_{i+1} \leftarrow w_i/2$

Additive Increase

increase sending rate by 1 maximum segment size (MSS) every time unit until congestion detected

Multiplicative Decrease

cut sending rate in half at each congestion event



AIMD sawtooth behavior: *probing* for bandwidth

→ sawtooth behavior

Congestion control: AIMD

- *approach:* senders can increase sending rate until congestion occurs, then decrease sending rate on congestion

Additive Increase

increase sending rate by 1 maximum segment size (MSS) every time unit until congestion detected

Multiplicative Decrease

cut sending rate in half at each congestion event

Questions?
- How to detect congestion?
- Why AIMD?
- What is the time unit?

Two signals of congestion

(1) Increase in RTTs ⇒ queuing delay is increasing ⇒ congestion

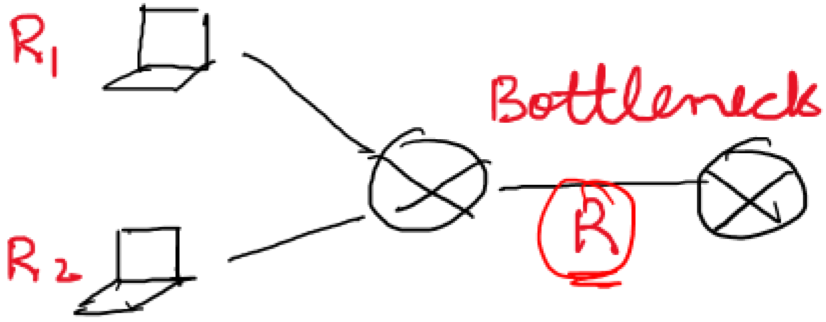
(2) Packet loss

(1) ACK Timeout (2). Triple duplicate ACK

Why AIMD? → TCP fair [MIMD, MIAD, AIAD]

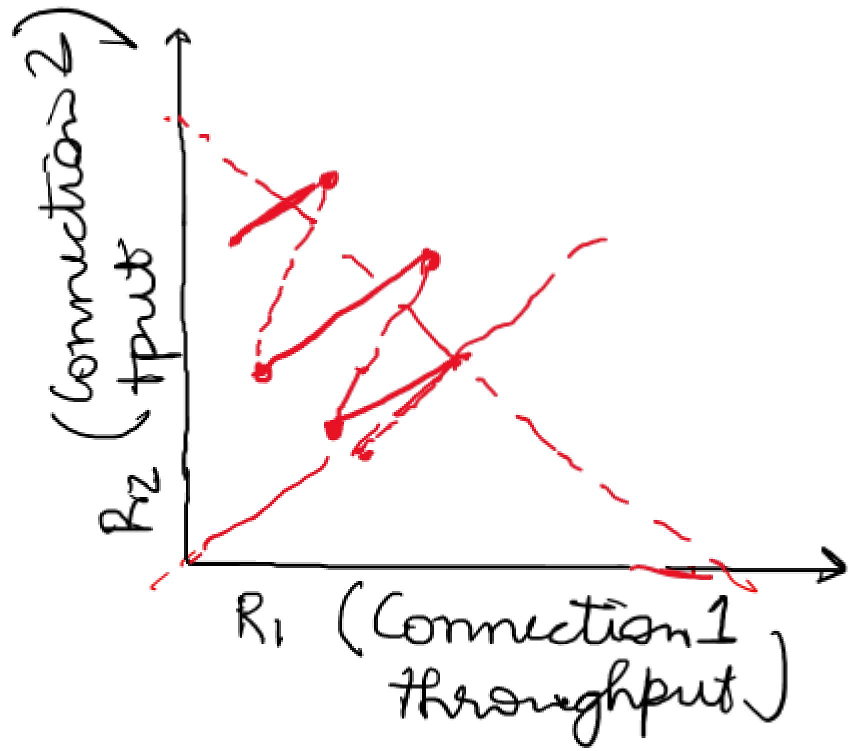
TCP fairness: if *K* TCP sessions share same bottleneck link of bandwidth *R*, each should have a long-term average rate of *R/K*

$R/2$



$R_1 + R_2 \le R$

AIMD is TCP fair

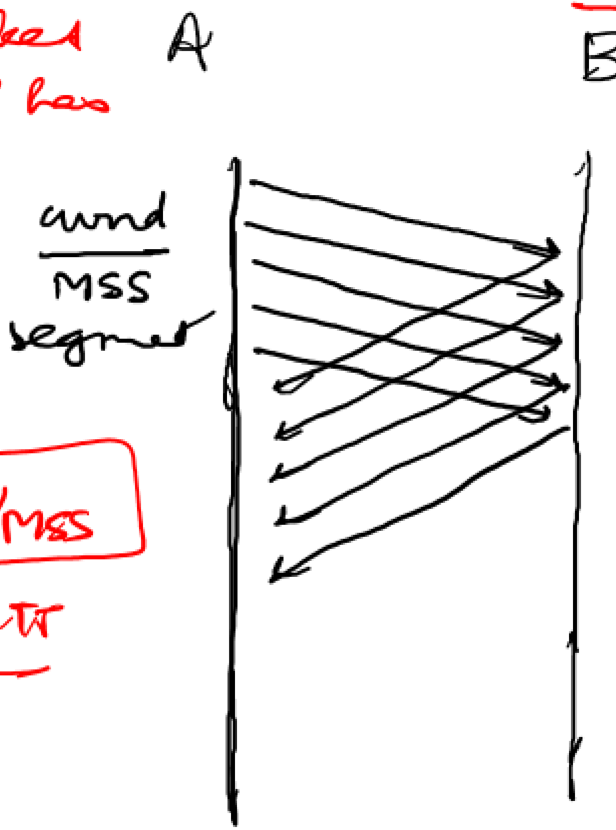


Window Update in AIMD

$\boxed{cwnd} \mathrel{+}= 1$ MSS every RTT or every t seconds

# of unacked bytes that TCP has

- **Goal**: For additive increase, update window size by 1 MSS every RTT. But when?
  - Window updated as ACKs arrive



- How much should be the update?

cwnd (in bytes)

How many segments are in flight: $\boxed{cwnd/MSS}$

want to increase 1 MSS every RTT

$$\left[cwnd \leftarrow cwnd + \frac{1\,MSS \times MSS}{cwnd} \right]$$

$$\frac{cwnd}{RTT} \text{ bytes per second} \rightarrow \frac{cwnd + MSS}{RTT}$$

[Why not increase cwnd by 1 every ACK]

↓

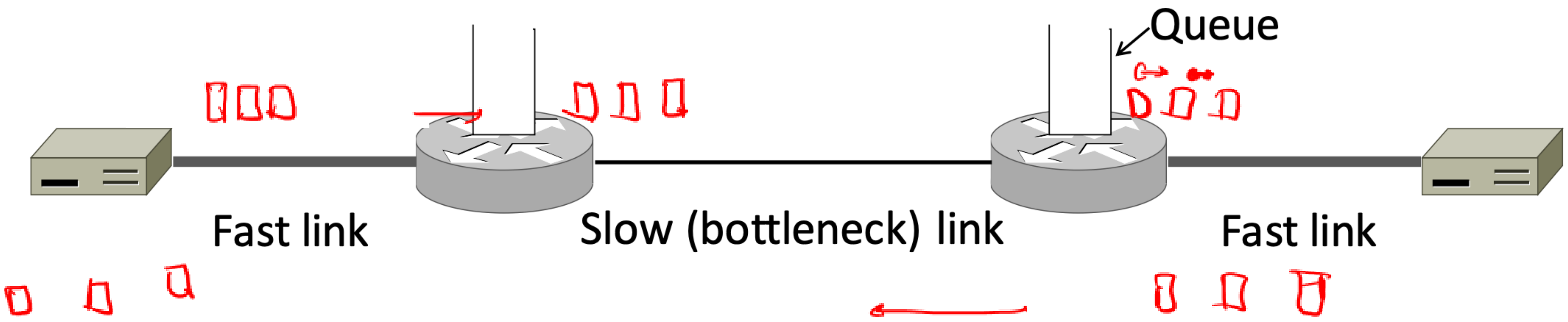
Exponential increase

- Because TCP updates based on ACKs, TCP is self-clocking

Benefit: Self-clocking smooths packet sending rate

TCP Self-Clocking

- ACK clocking smooths packet sending rate



In the beginning of the connection…

- What *cwnd* size should we start with?
  - **Goal**: We want to quickly near the *right* rate.
  - A linear increase with a small value of cwnd is painfully slow!
- What are the options?
  - Start with a large value of cwnd IW = 2 → $\boxed{IW = 10}$
  - Start with a small value of cwnd but increase it faster

↓

exponential increase

instead of chang cwnd by 1 MSS every RTT

double the cwnd every RTT