

Lecture 15: Transport Layer

Congestion Control

COMP 332, Spring 2018

Victoria Manfredi

WESLEYAN
UNIVERSITY



Acknowledgements: materials adapted from Computer Networking: A Top Down Approach 7th edition: ©1996-2016, J.F Kurose and K.W. Ross, All Rights Reserved as well as from slides by Abraham Matta at Boston University, and some material from Computer Networks by Tannenbaum and Wetherall.

Today

1. Announcements
 - homework 6 out Wed.
 - midterm should be graded Wed.
2. Congestion causes and costs
3. TCP congestion control
4. Network layer overview

Congestion

CAUSES AND COSTS

What if sender overwhelms network?

Receive buffer is not only resource limitation

- every pkt has to travel through **path of routers**
- routers may be **congested**, have **long queues** ...

Causes of network congestion

- many senders **competing** for network resources
- senders **lacking knowledge**
 - amount of resources available (bandwidth)
 - # of other senders competing

Costs of network congestion

As queues in bottleneck link fill up

- large packet delays
- dropped packets

As timeouts expire at sender due to delays/drops

- packets retransmitted



Problem

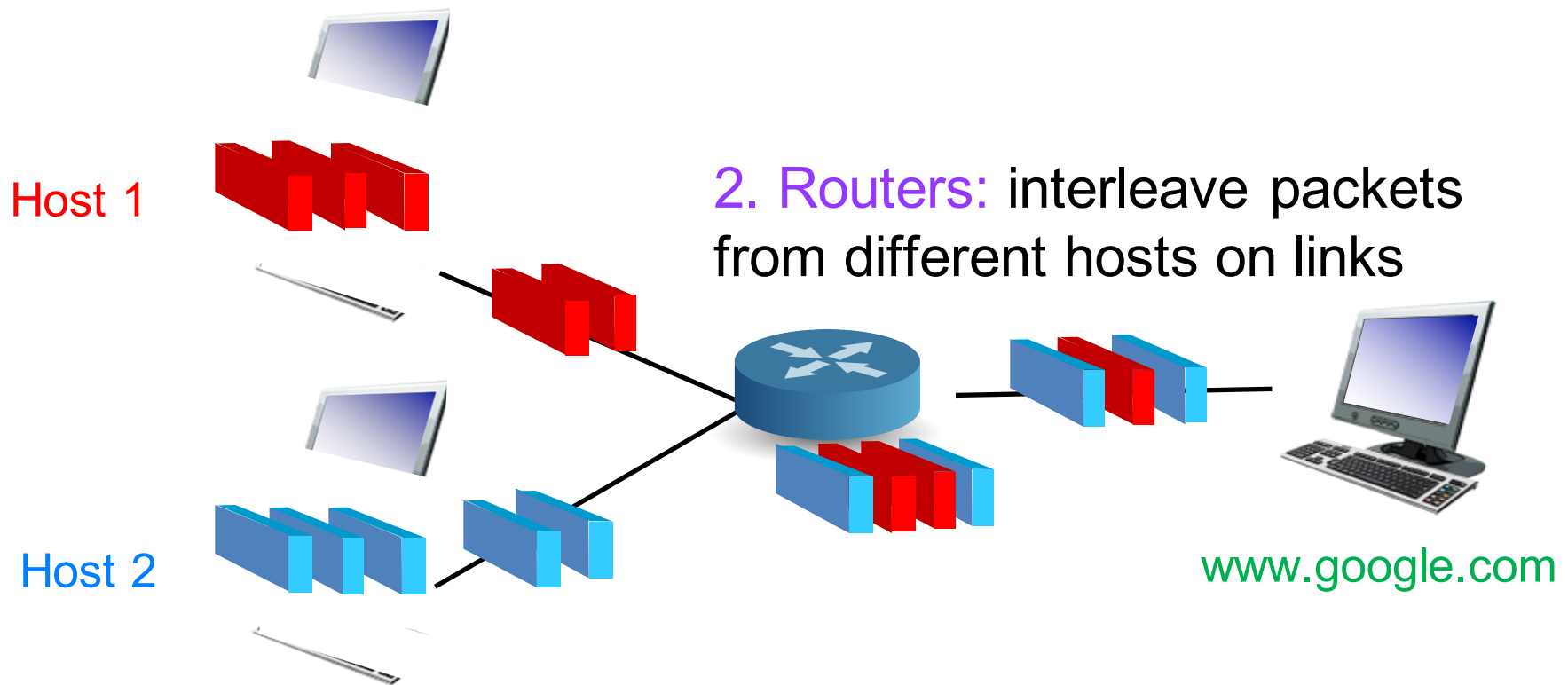
- retransmission treats symptoms but not underlying problem

Q: How to solve underlying problem of congestion?

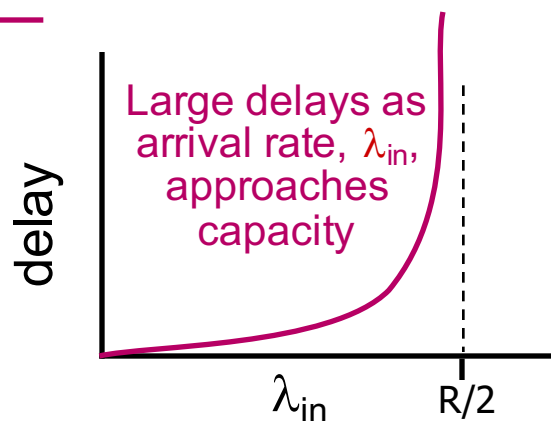
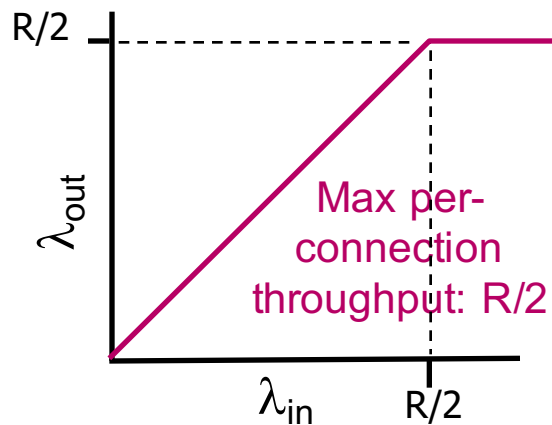
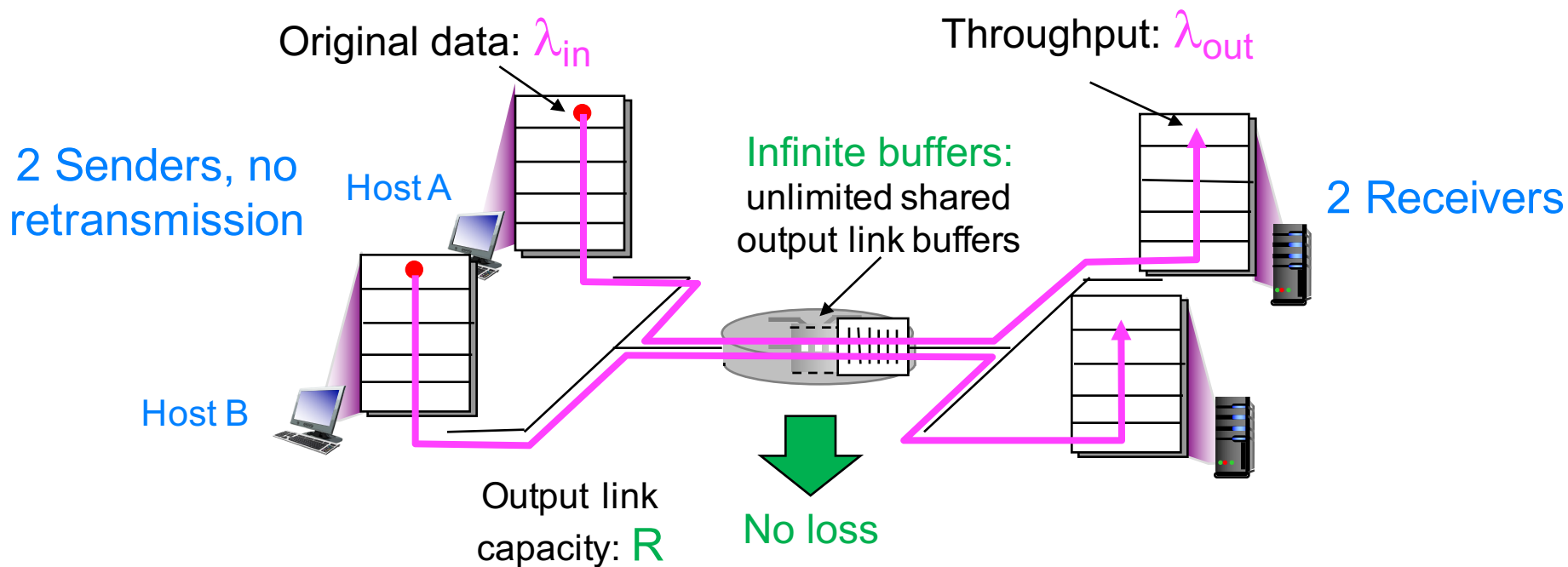
- reduce sending rate ... but what should sending rate be?
 - depends on available bandwidth
 - sender increases/decreases sending rate based on congestion level

Recall link and network resources are shared

1. **Hosts:** divide data to send into fixed-length packets

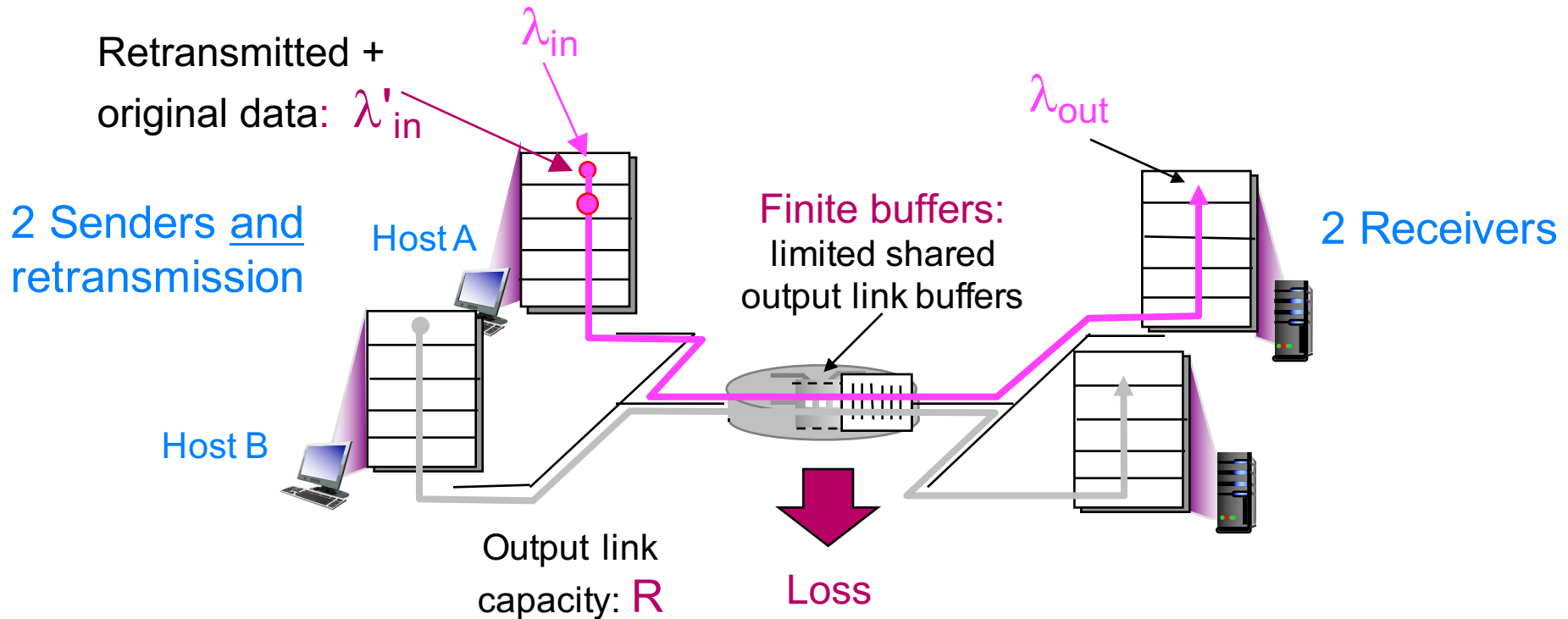


Scenario 1: no retransmission



Q: Why $R/2$?

Scenario 2: retransmission

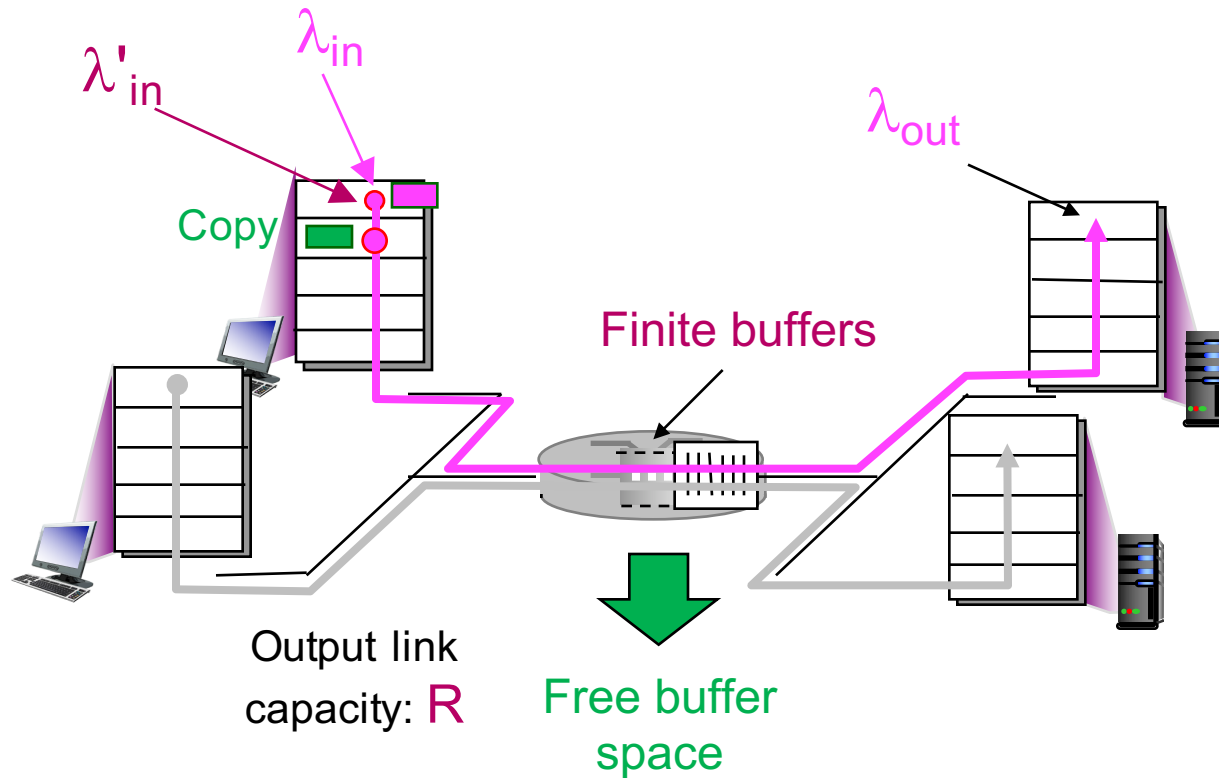


Sender retransmits timed-out packet

- $\lambda_{in} = \lambda_{out}$: app-layer input equals app-layer output
- $\lambda'_{in} \geq \lambda_{in}$: transport-layer input includes **retransmissions**

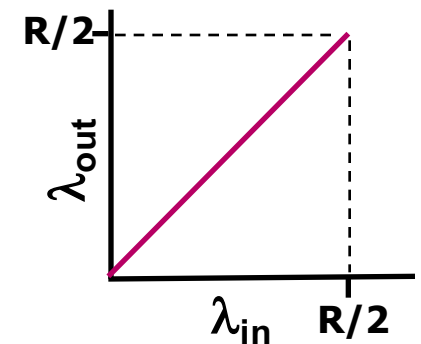
Performance now depends on how retransmission performed

Scenario 2: retransmission + perfect knowledge

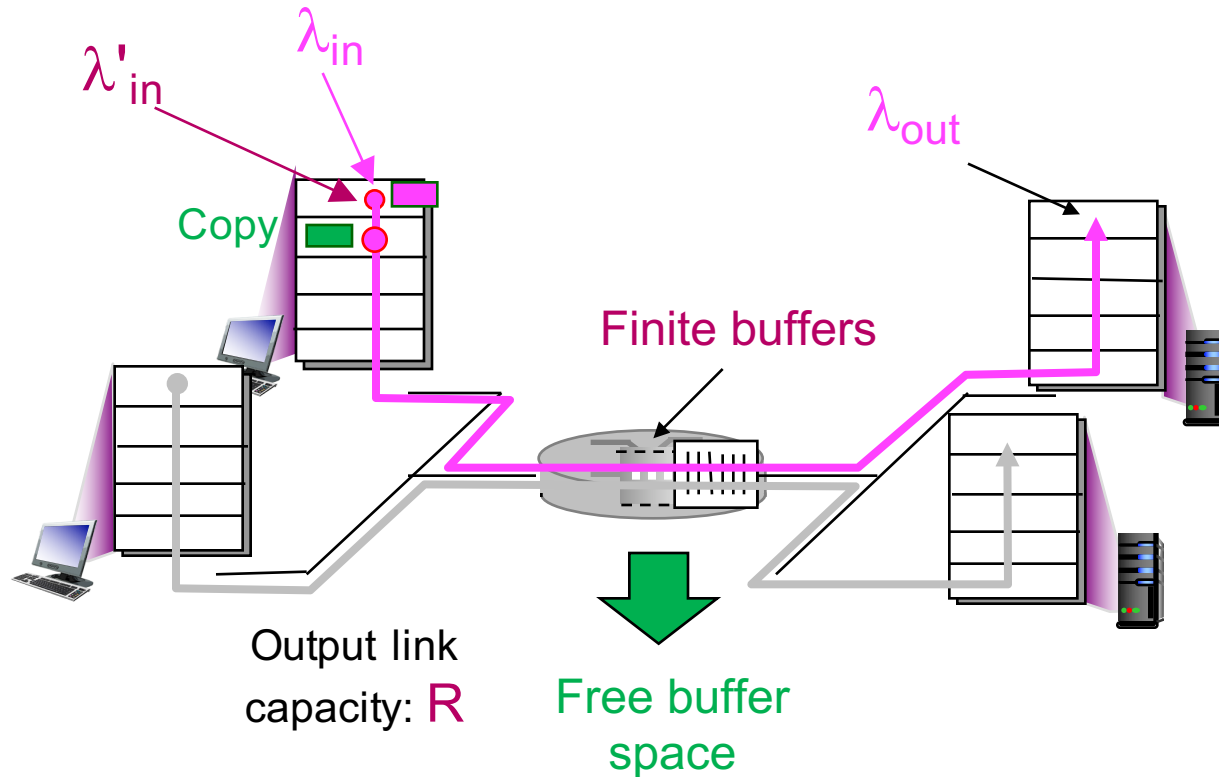


Idealization: perfect knowledge

- sender sends only when router buffers available
- no loss occurs, so $\lambda'_{in} = \lambda_{in}$

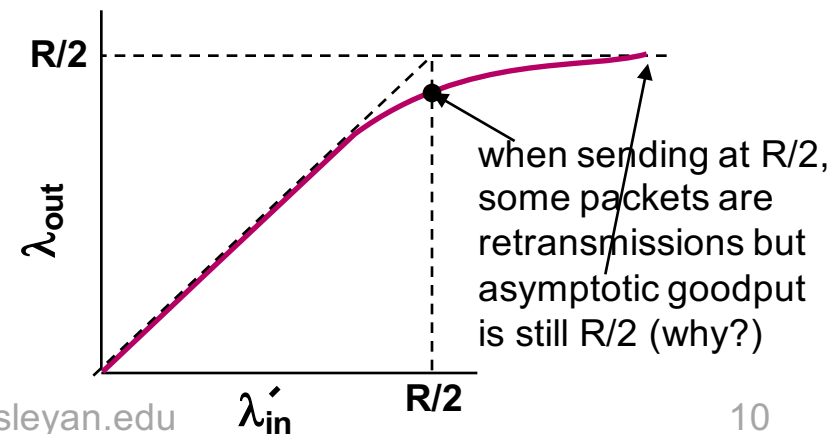


Scenario 2: retransmission only when lost

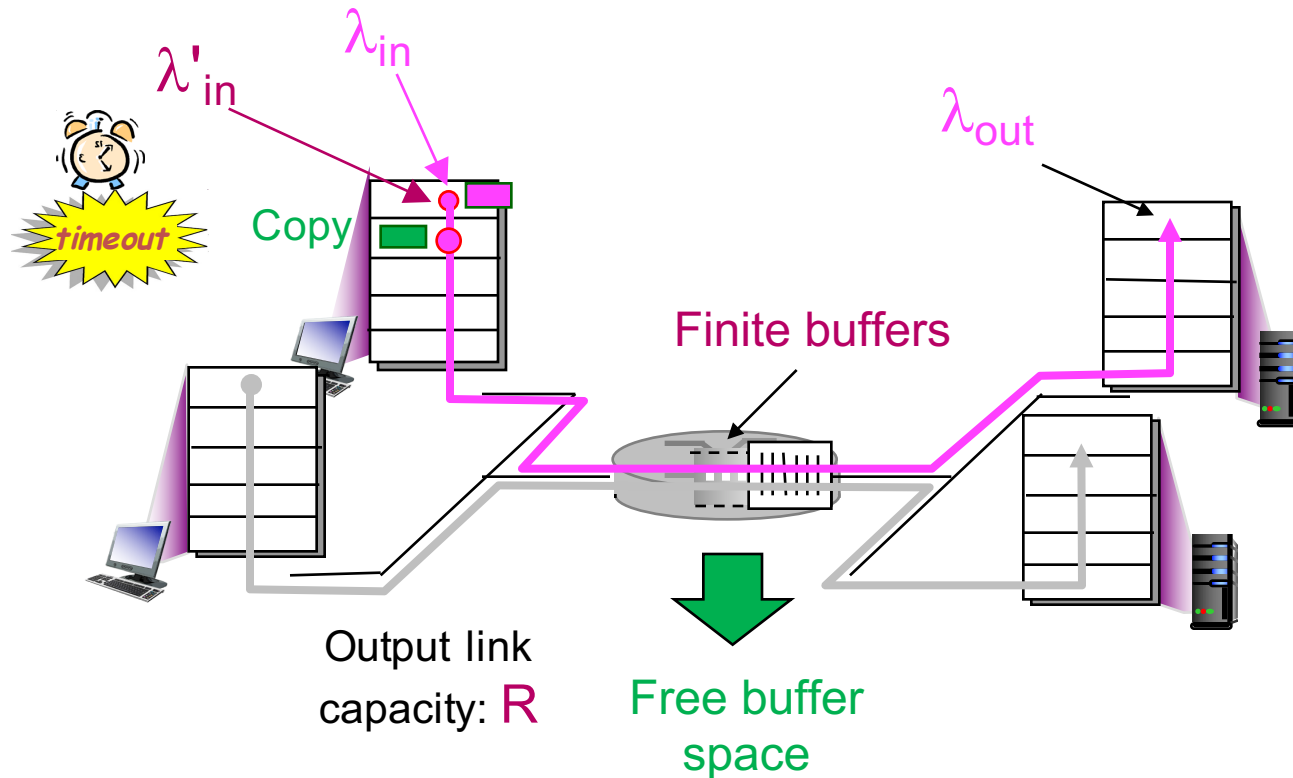


Idealization: known loss

- packets can be lost, dropped at router due to full buffers
- only resend packet known to be lost

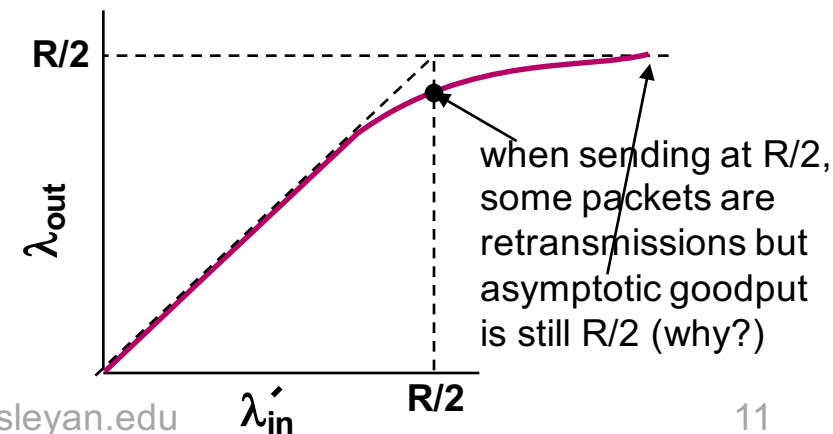


Scenario 2: retransmission causing duplicates



Realistic: duplicates

- packets can be lost, dropped at router due to full buffers
- sender times out prematurely, sending two copies, both of which are delivered



TCP

CONGESTION CONTROL

Goals of TCP congestion control

1. Discover available bandwidth

- how much bandwidth can be used without causing congestion
 - will vary over time
- estimate starting from no info

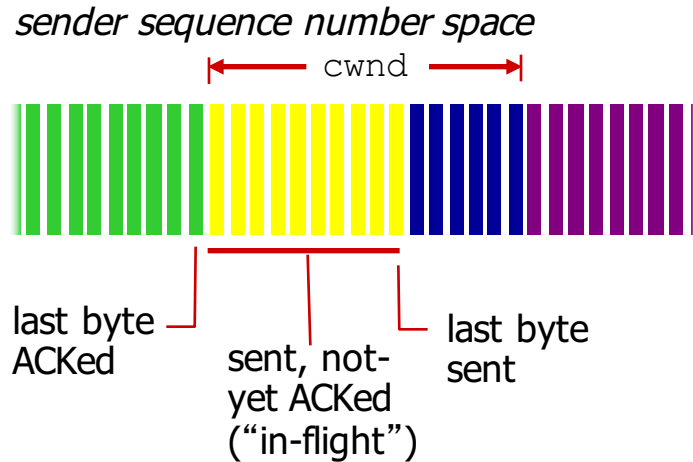
2. Correctly set sending rate

- should not exceed available bandwidth

3. Fairness

- no user gets all of the bandwidth

TCP Congestion Control



TCP sending rate

- roughly
 - send cwnd bytes
 - wait RTT for ACKs
 - then send more bytes

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

Sender limits transmission

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$$

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

Q: How does sender estimate cwnd?

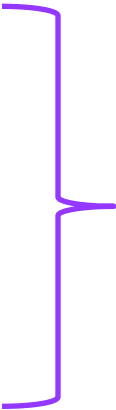
To estimate cwnd

Detect congestion

- delays
 - large RTTs: too variable to be used in practice

- duplicate ACKs
 - isolated loss

- timer expired
 - multiple losses



Use to adjust cwnd,
affecting sending rate

How to intuitively adjust cwnd

- ACK received: increase cwnd
- loss detected: decrease cwnd

3 states in TCP finite state machine

Goal: send segments, adjust cwnd as needed

Slow start

- determine **available bandwidth** starting from no info

Congestion avoidance

- deal with **fluctuations** in bandwidth

Fast recovery

- quickly recover from **isolated lost packets**

We'll first look at different states, then full FSM

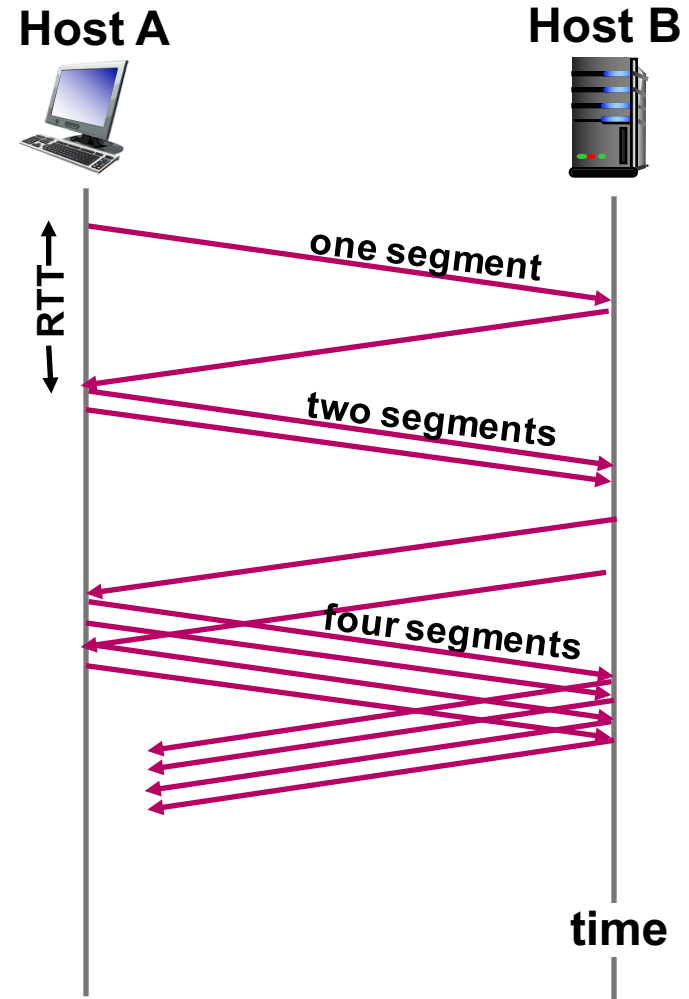
Slow start: initialization

Initial rate is “slow”

- relative to original TCP which had no congestion control
- initially $cwnd = 1 \text{ MSS}$

Ramp up exponentially fast

- every time ACK received
 - $cwnd = cwnd + \text{MSS}$
- essentially doubles $cwnd$ every RTT



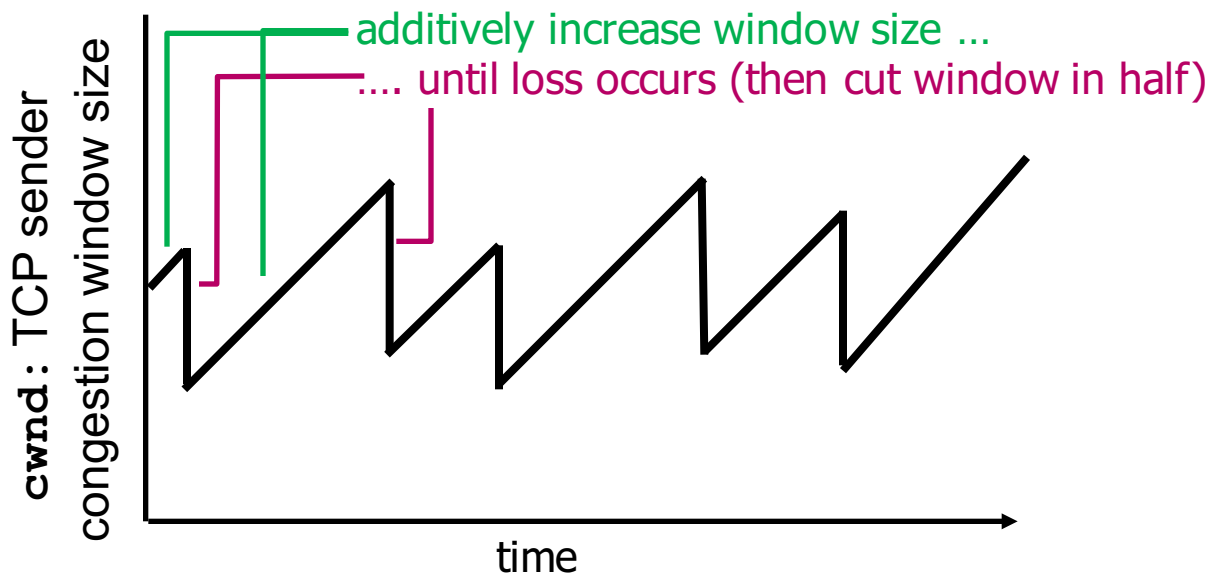
Congestion avoidance

Additive Increase Multiplicative Decrease (AIMD)

- additive increase (cautious)
 - increase cwnd by 1 MSS every RTT until loss detected
- multiplicative decrease (aggressive)
 - cut cwnd in half after loss

Probe cautiously for usable bandwidth

AIMD saw tooth behavior: probing for bandwidth



Slow start: when to stop exponential increase?

Slow start

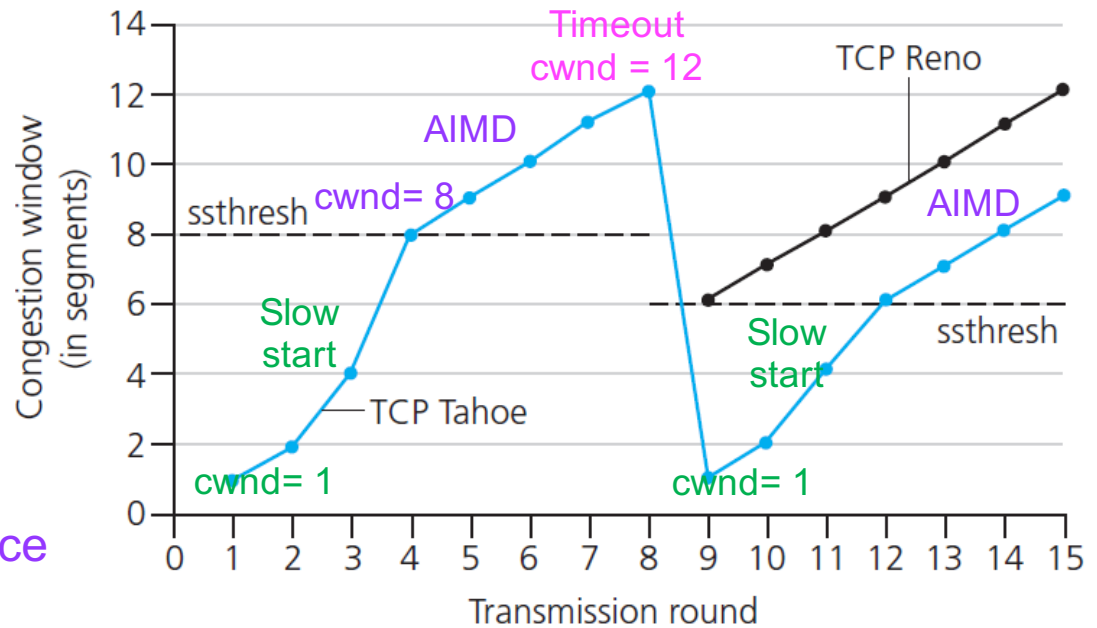
- initially $cwnd = 1$ MSS
- every time ACK received
 - double $cwnd$

When $cwnd = ssthresh$

- go to congestion avoidance
- use AIMD

Timeout

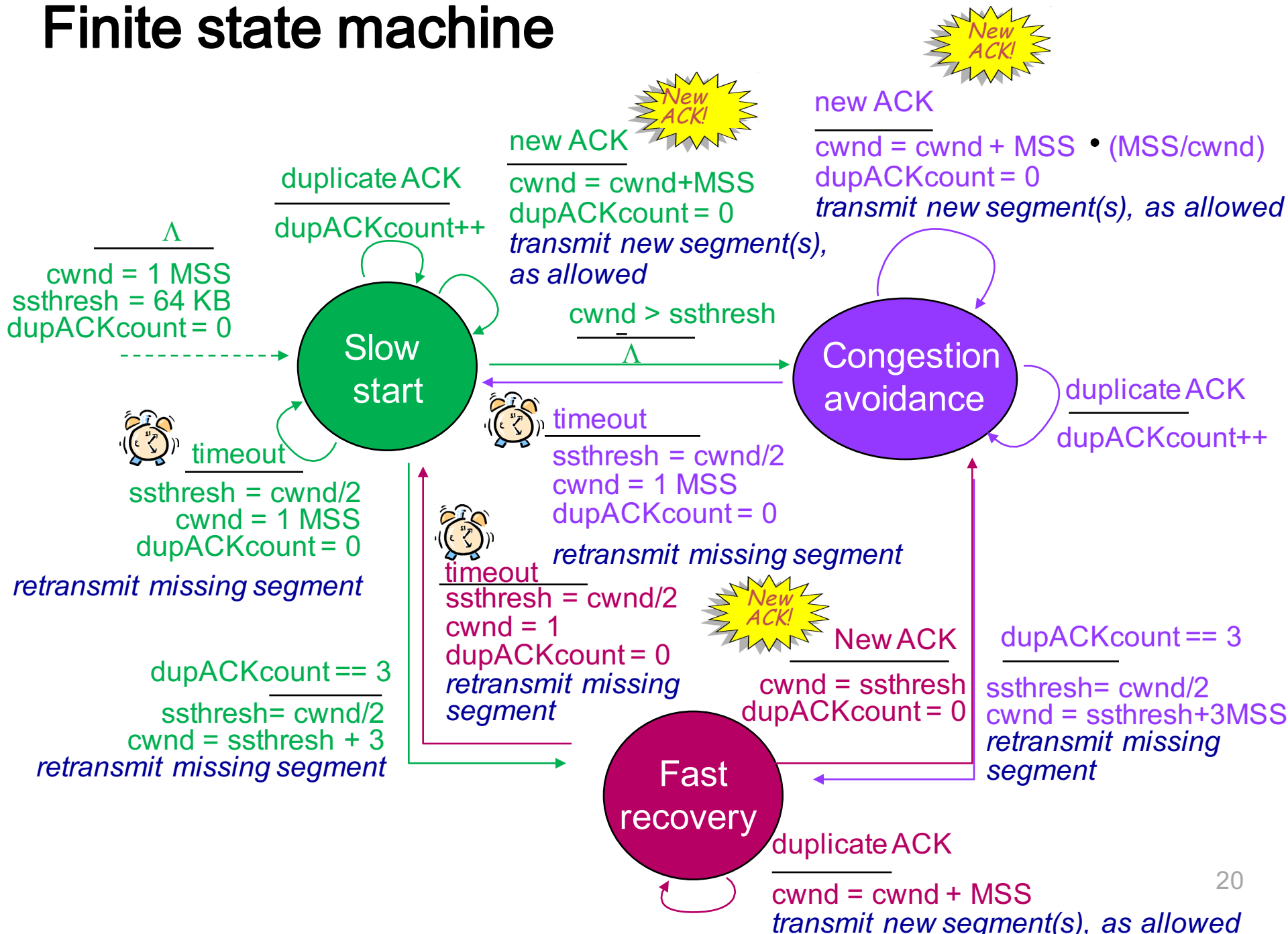
- restart slow start
- $ssthresh = cwnd/2$
- $cwnd = 1$ MSS



If 3 duplicate ACKs

- go to fast recovery
- $ssthresh = cwnd/2$
- $cwnd = ssthresh + 3$ MSS

Finite state machine



Average TCP throughput

Focus just on AIMD

- ignore slow start, assume always data to send

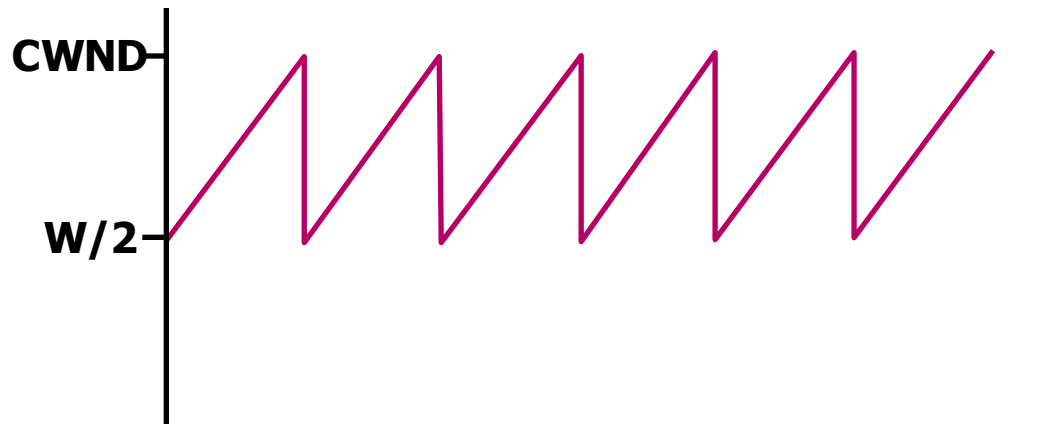
Max rate

- cwnd / RTT

3 dup loss rate

- $0.5 \text{ cwnd} / \text{RTT}$

$$\text{Avg TCP thruput} = \frac{3}{4} \frac{\text{CWND}}{\text{RTT}} \text{ bytes/sec}$$



Setting window size

Window is min (rwnd, cwnd)

```
▼ Transmission Control Protocol, Src Port: 443 (443), Dst Port: 52232 (52232), Seq: 0, Ack: 1,
  Source Port: 443
  Destination Port: 52232
  [Stream index: 0]
  [TCP Segment Len: 0]
  Sequence number: 0 (relative sequence number)
  Acknowledgment number: 1 (relative ack number)
  Header Length: 32 bytes
  ▼ Flags: 0x012 (SYN, ACK)
    000. .... = Reserved: Not set
    ...0 .... = Nonce: Not set
    .... 0... = Congestion Window Reduced (CWR): Not set
    .... .0.. = ECN-Echo: Not set
    .... ..0. = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    ► .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
    [TCP Flags: *****A**S*]
    window size value: 8190 rwnd
    [Calculated window size: 8190]
    Checksum: 0x500 [validation disabled]
```