TCP congestion control

CE 352, Computer Networks

Salem Al-Agtash

Lecture 11

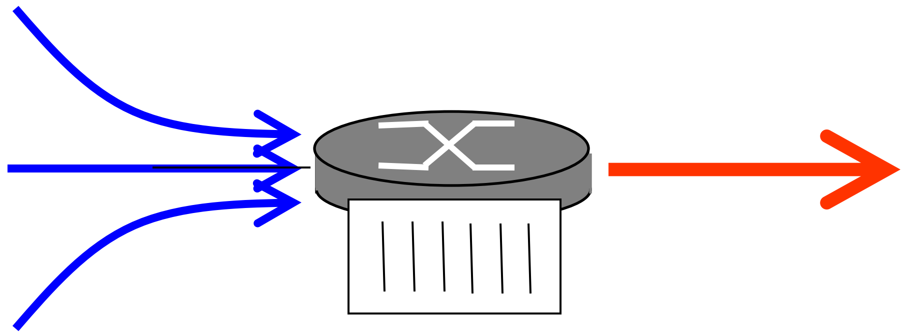
Slides are adapted from Computer Networking: A Top Down Approach, 7th Edition © J.F Kurose and K.W. Ross

Principles of congestion

If two packets arrive at the router, only one can be transmitted and the other will be either buffered in queue or dropped.

If many packets arrive, the router cannot keep up with the arriving traffic, leading to delays and eventually buffer overflow → *congestion, resulting in:*

- lost packets (buffer overflow at routers)
- long delays (queueing in router buffers)



Congestion control

- TCP approach to manage congestion:
  - Implicit/ explicit feedback from network
  - End hosts adjust sending rate

- Historical perspective:
  - Congestion collapse on NSF network in 1986 (32Kbps → 40bps)
  - As packet drops, senders keep retransmitting (flow control limits)
  - TCP congestion control algorithms (Jacobson):
    - Window size adapts to congestion at TCP

- Key elements:
  - Find out available bandwidth
  - Adjust to bandwidth variations
  - Share bandwidth between flows

TCP Approach

TCP congestion window
- Controls number of packets in flight
- Sending rate: ~Window/RTT
- Vary window size to control sending rate

Flow control window: RWND [Receiver]
- How many bytes can be sent without overflowing receiver's buffers

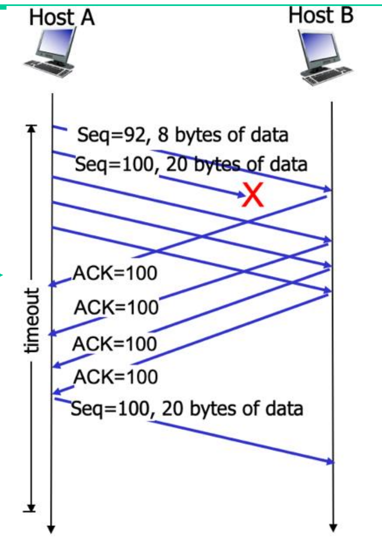
Congestion Window: CWND [Sender]
- How many bytes can be sent without overflowing routers

Sender-side window = minimum{CWND, RWND}
- CWND in units of MSS (Bytes in implementation)
- MSS: Maximum Segment Size, the amount of payload data in a TCP packet

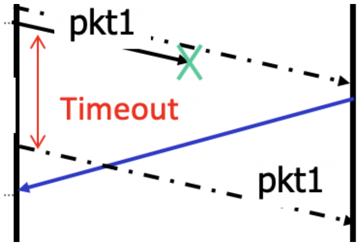
TCP Congestion Control

How to detect congestion

- Packet delays
  - Routers tell endhosts when they're congested (Explicit Congestion Notification)
- Packet loss (Duplicate ACKs, Timeout)
  - TCP to detect



fast retransmit after sender receipt of triple duplicate ACK



Sending rate adjustment

- Upon receipt of ACK (of new data): increase rate
- Upon detection of loss: decrease rate
- How do we increase/decrease the rate depends on the phase of congestion
  - Discovering available bottleneck bandwidth
  - Adjusting to bandwidth variations

TCP Congestion Control Algorithm

[Jacobson 1988] and is standardized in [RFC 5681], algorithm with three major components:

- slow start
- congestion avoidance
- fast recovery.

Slow start and congestion avoidance are mandatory components of TCP, differing in how they increase the size of cwnd in response to received ACKs.

Fast recovery is recommended, but not required, for TCP senders.

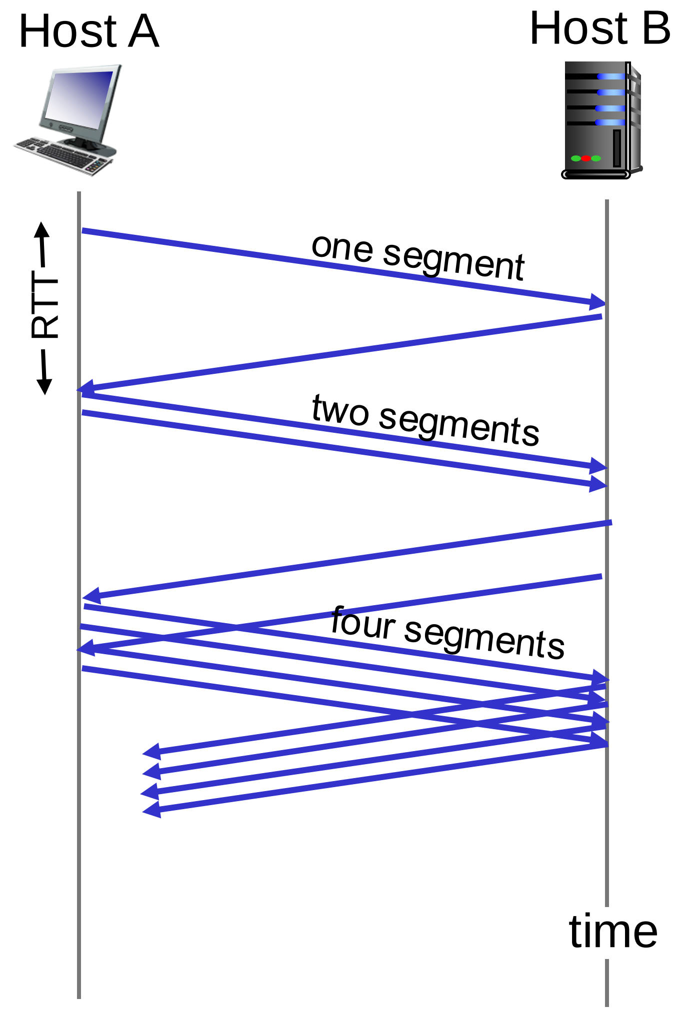
TCP Slow Start: Discover bandwidth

Goal: estimate available bandwidth

- start slow (for safety)
- but ramp up quickly (for efficiency)

Idea: when connection begins, increase rate exponentially until first loss event:

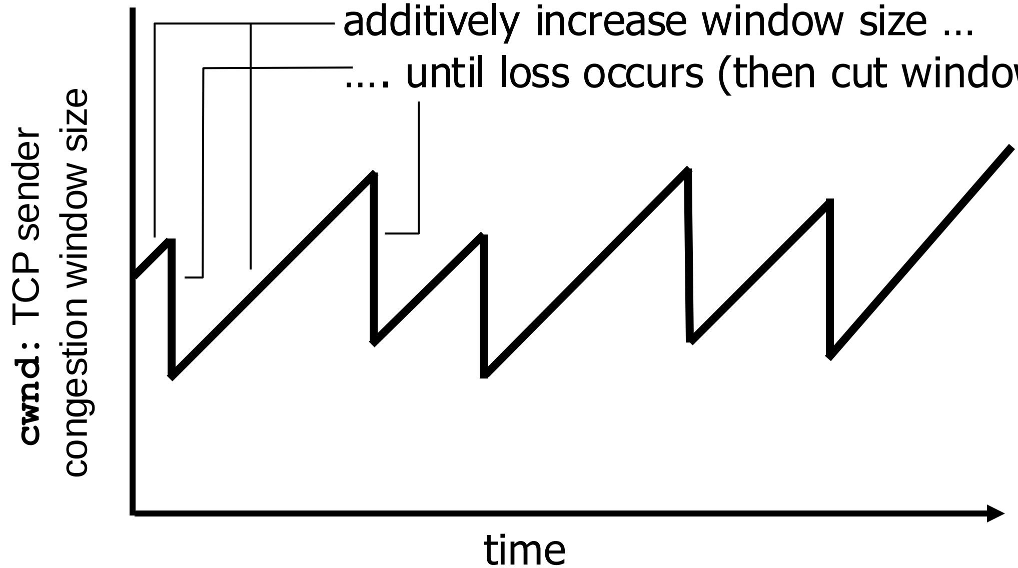
- initially **cwnd** = 1 MSS, sending rate = MSS/RTT
- double **cwnd** every RTT
- →done by incrementing **cwnd** for every ACK received



Congestion avoidance: AIMD to adjust to varying bandwidth

- *approach:* sender increases 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

AIMD saw tooth behavior: probing for bandwidth



TCP uses: "Additive Increase Multiplicative Decrease" (AIMD)

Fast recovery

In fast recovery, the value of cwnd is increased by 1 MSS for every duplicate ACK received for the missing segment that caused TCP to enter the fast-recovery state.

TCP detecting and reacting to:

- loss as indicated by timeout or 3 duplicate ACKs: TCP Tahoe
  - **`cwnd`** set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold, then grows linearly
- loss indicated by 3 duplicate ACKs: TCP RENO
  - **`cwnd`** is cut in half window then grows linearly

Slow-Start vs. AIMD (congestion avoidance)

Sender stops Slow-Start and starts Additive Increase

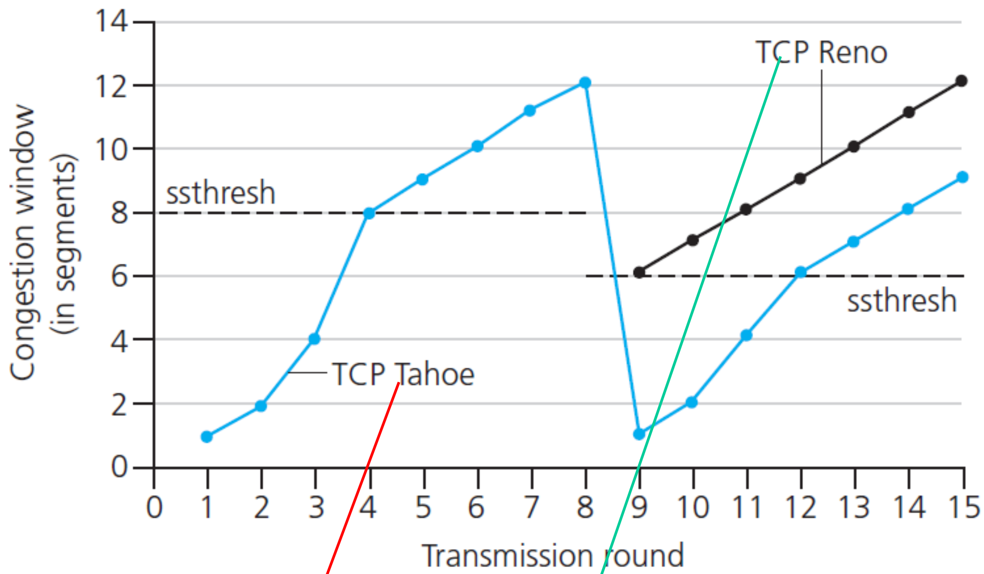
Introduce a “slow start threshold” (ssthresh)

- Initialized to a large value
- On timeout, ssthresh = CWND/2

When CWND = ssthresh, sender switches from slow-start to AIMD-style increase

TCP: switching from slow start to CA

The exponential increase switches to linear 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

`TCP Tahoe:` **`cwnd`** to 1 (timeout or 3 duplicate acks)

`TCP Reno:` **`cwnd`** is cut in half window then grows linearly (3 duplicate Ack

Interactive examples:
http://gaia.cs.umass.edu/kurose_ross/interactive/index.php

TCP throughput

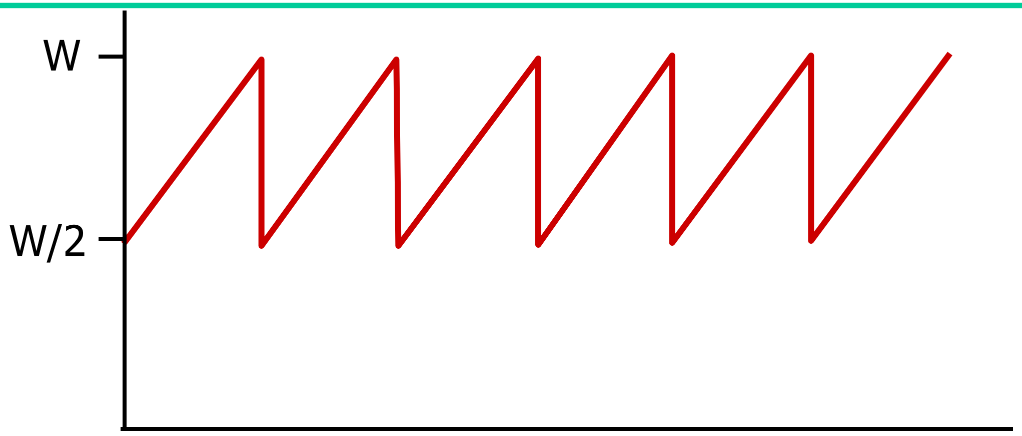
avg. TCP throughput as function of window size, RTT?

- ignore slow start, assume always data to send

W: window size (measured in bytes) where loss occurs

- avg. window size (# in-flight bytes) is ¾ W
- avg. thruput is 3/4W per RTT

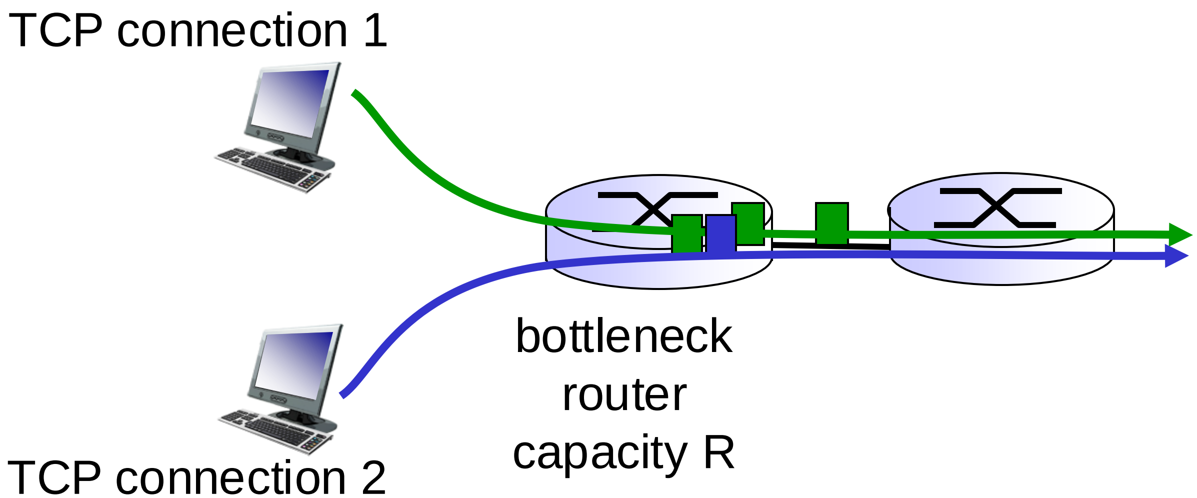
$$\text{avg TCP thruput} = \frac{3}{4}\frac{W}{RTT}\ \text{bytes/sec}$$



TCP Futures: TCP over "long, fat pipes"

TCP Fairness

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



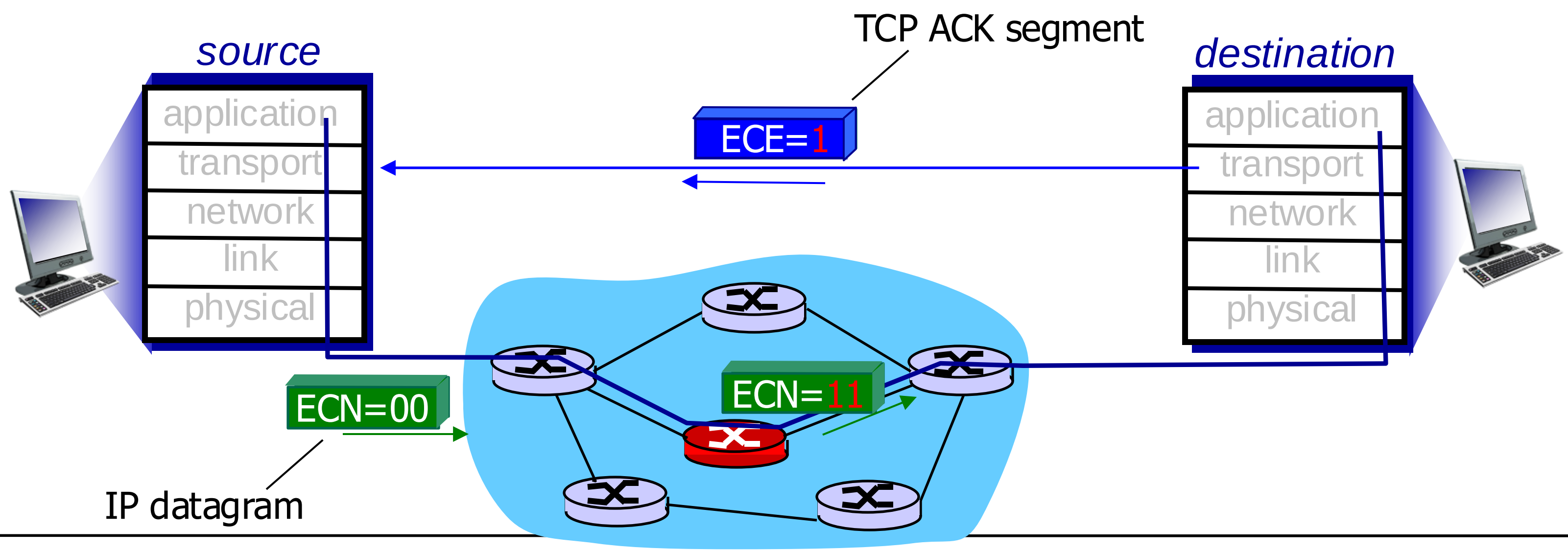
Explicit Congestion Notification (ECN)

network-assisted congestion control:

two bits in IP header (ToS field) marked *by network router* to indicate congestion

congestion indication carried to receiving host

receiver (seeing congestion indication in IP datagram)) sets ECE bit on receiver-to-sender ACK segment to notify sender of congestion



Bonus 4

- TFv2 - Stop and Wait for an Unreliable Channel using UDP (client – server)
- Messages are sent one at a time, and each message needs to be acknowledged when received, before a new message can be sent.
  - TFv2 implements protocol rdt2.2.
  - HEADER
    - seq_ack int (32 bits) // SEQ for data and ACK for Acknowledgement
    - len int (32 bits) // Length of the data in byes (zero for ACKS)
    - cksum int (32 bits) // Checksum calculated (by byte)
  - PACKET
    - header
    - data char (10 bytes)
  - SENDER
    - Member seq_ack is used as SEQ, and the data is in member data.
    - Each packet = 10 or less bytes of data, and the sender only sends the necessary bytes.
    - After transmitting the file, a packet with no data (len = 0) is sent to notify the receiver
  - RECEIVER
    - Member seq_ack is used as ACK, and data is empty (len = 0)

Summary

Today:

- TCP Congestion
- Control algorithm
  - Slow start
  - AIMD
  - Fast recovery
- TCP fairness

Canvas discussion:

- Reflection
- Exit ticket

Next time:

- read 4.1 and 4.2 of K&R (Network layer: data plane)
- follow on Canvas! material and announcements

Any questions?