CS118 Discussion 1C, Week 5

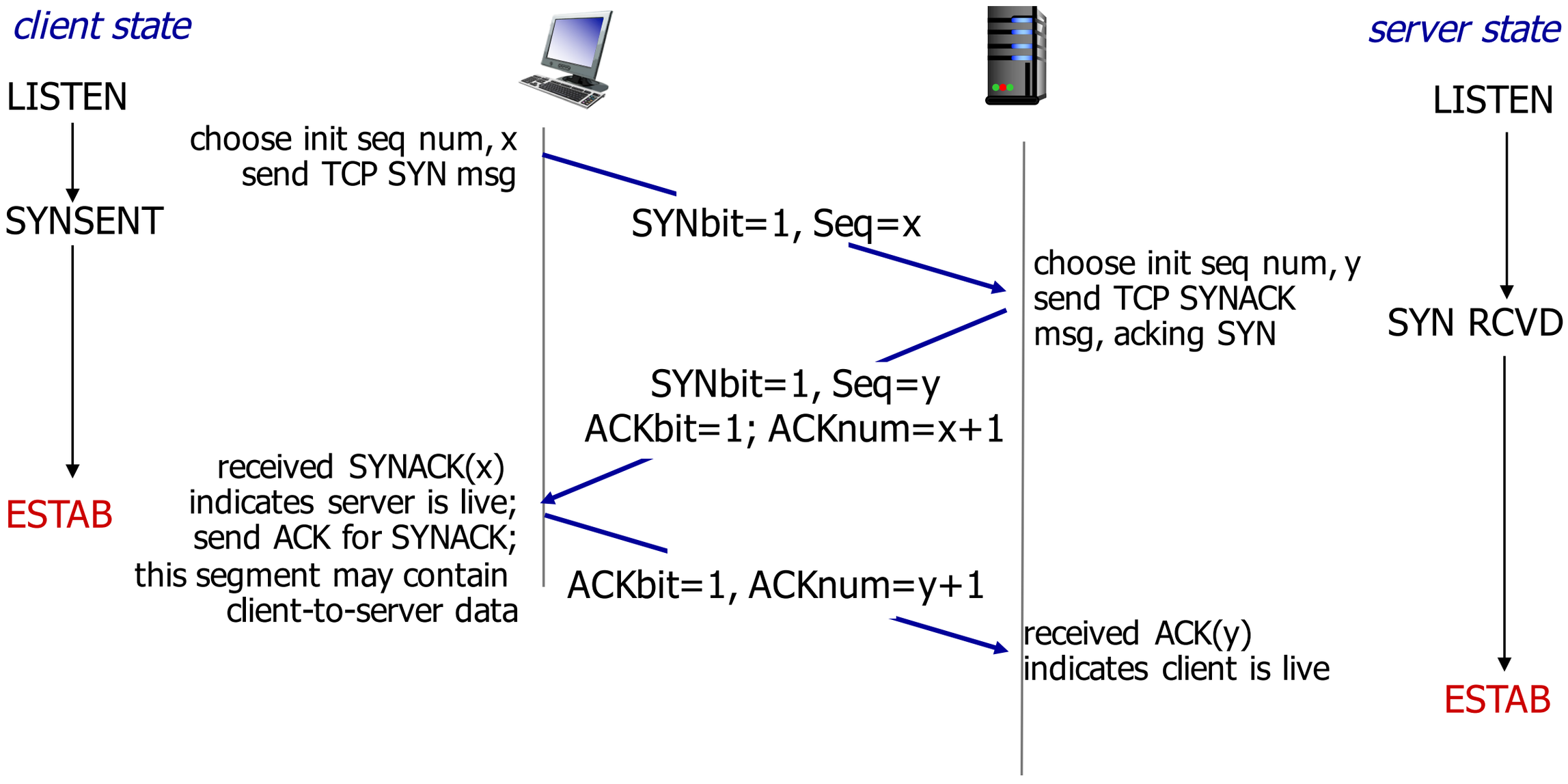
Zengwen Yuan

Bunche Hall 3156, Friday 2:00—3:50 p.m.

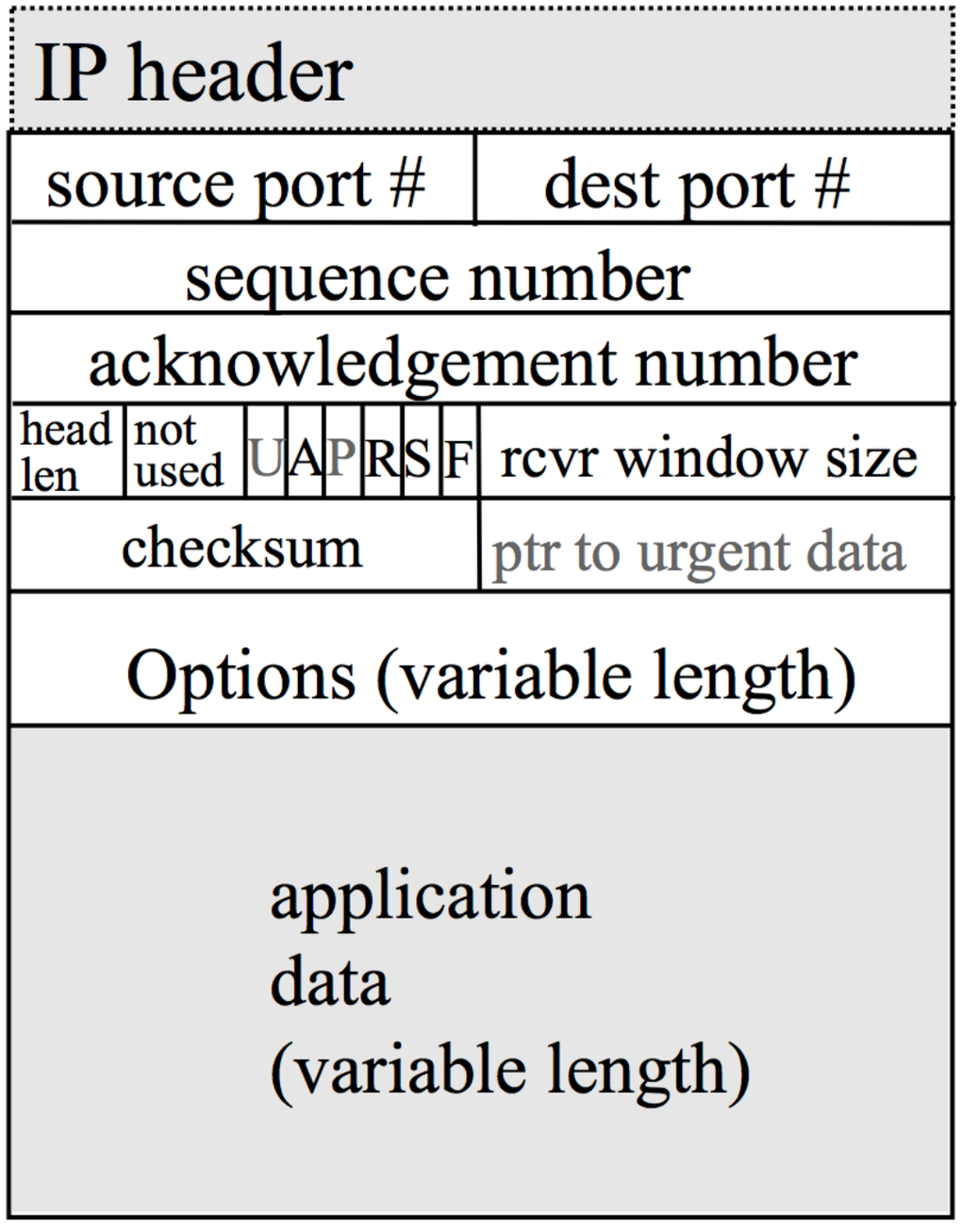
Outline

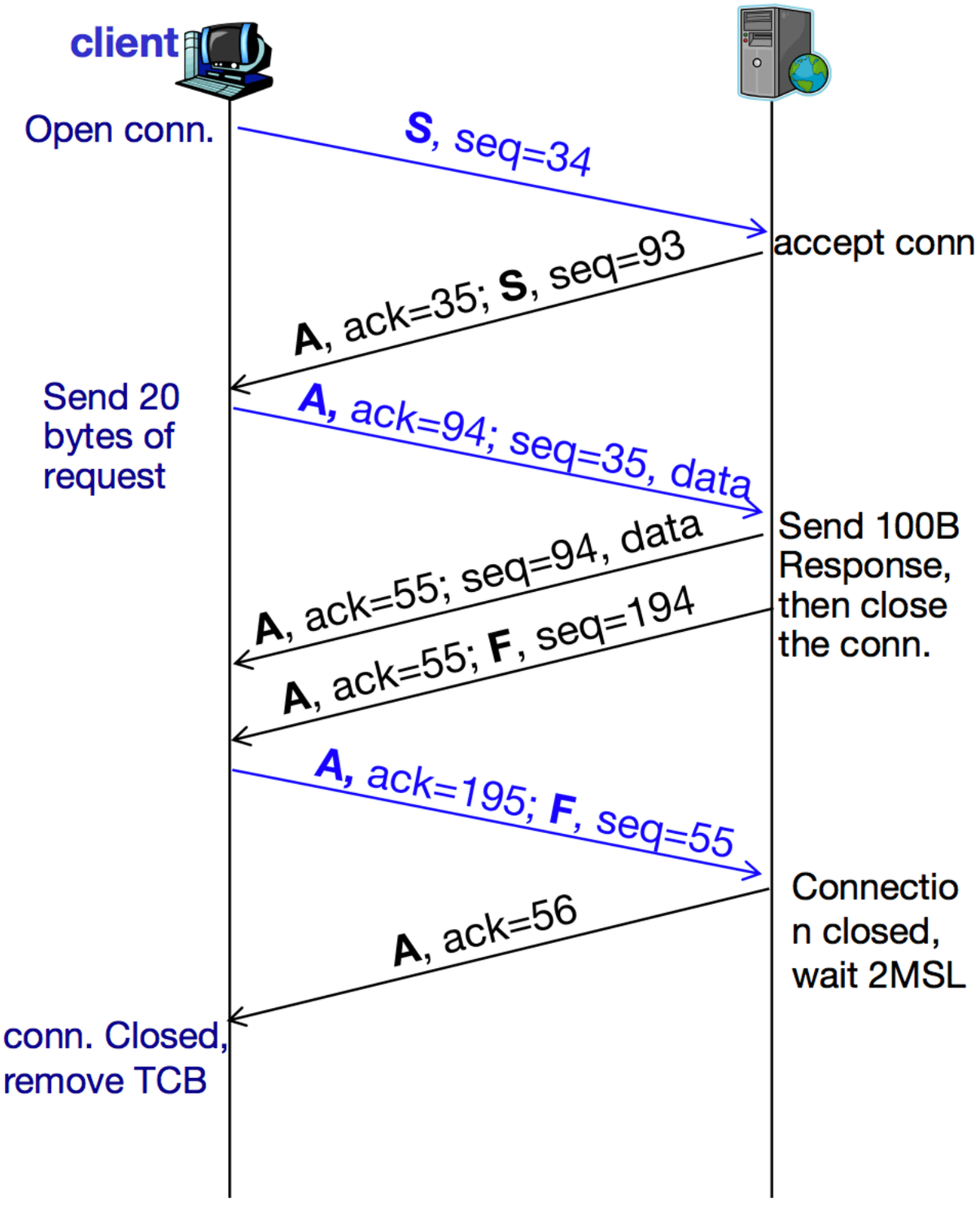
- Lecture review: TCP handshake; congestion control
- Midterm **poll**!
  - Original: Thursday, May 9th, in-class
    - closed book, no electronic devices
    - one two-sided US letter size (8.5x11) cheatsheet is allowed
- An exercise on congestion control

TCP 3-way handshake

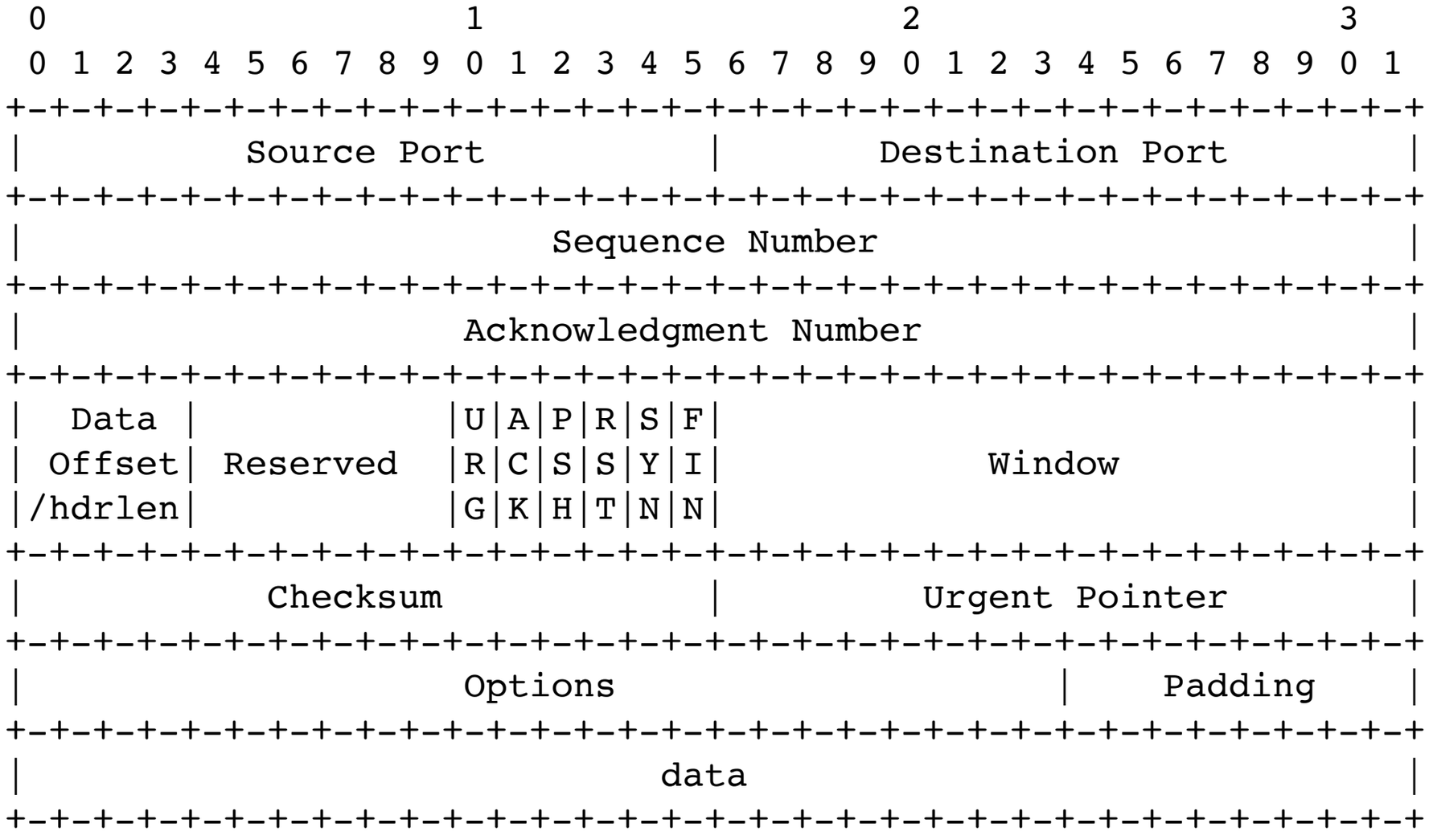


An HTTP 1.0 connection example





TCP: header format



TCP Header Format

Note that one tick mark represents one bit position.

TCP: flow control

- Limits the rate a sender transfers data
- Avoid having the sender send data too fast
- Avoid exceeding the capacity of the receiver to process data
- Receiver specify the receive window
- The window size announce the number of bytes still free in the receiver buffer

TCP: congestion control

- Why Congestion Control
  - Oct. 1986, Internet had its first congestion collapse (LBL to UC Berkeley)
  - 400 yards, 3 hops, 32 kbps
    - throughput dropped by a factor of 1000 to 40 bps
- 1988, Van Jacobson proposed TCP congestion control
  - Window based with ACK mechanism
  - End-to-end

TCP: congestion control — window-based

- Limit number of packets in network to window size W
  - Source rate allowed (bps) = W×Message Size/RTT
  - Too small W?
  - Too large W?

TCP: congestion control — effects

- Packet loss
- Retransmission and reduced throughput
- Congestion may continue after the overload

TCP: congestion control — basics

- Goals: achieve high utilization without congestion or unfair sharing

- Receiver control (**`awnd`**): set by receiver to avoid overloading receiver buffer

- Network control (**`cwnd`**): set by sender to avoid overloading network

  - **`W = min(cwnd, awnd)`**

- Congestion window **`cwnd`** usually is the bottleneck

TCP: congestion control — main parts

- Slow start
- Congestion Avoidance
- Fast retransmit
- Fast recovery

https://tools.ietf.org/html/rfc5681

TCP: congestion control — slow start

- Start with cwnd = 1 (MSS: max. segment size; abstract as pkt)
- Exponential growth
  - each RTT:
    - `cwnd ← 2 × cwnd`
  - equivalently, each ACK:
    - `cwnd ← cwnd + 1 (MSS)`
- Enter Congestion Avoidance when `cwnd > ssthresh`

TCP: congestion control — congestion avoidance

- Start with **`cwnd >= ssthresh`**
- Linear growth
  - each RTT:
    - **`cwnd ← cwnd + 1 (MSS)`**
- equivalently, each ACK:
  - **`cwnd ← cwnd + 1/cwnd (MSS^2/cwnd)`**

TCP: congestion control — packet loss

- Assumption: loss indicates congestion
- Packet loss detected by
  - Retransmission Timer Outs (RTO timer)
  - Duplicate ACKs (three)
    - ignore the 1st or 2nd duplicate ACK

TCP: congestion control — fast retx/recovery

- Upon 3rd duplicate ACK:
  - set `ssthresh ← cwnd/2`
  - set `cwnd ← ssthresh + 3 (MSS)`
  - upon additional dup ACK: grow `cwnd` linearly
    - New ACK: `cwnd ← ssthresh`, enter Congestion Avoidance
- Time Out
  - set `ssthresh ← cwnd/2`
  - set `cwnd ← 1 (MSS)`
  - enter slow start

TCP: congestion control — summary

- Congestion is indicated by packet loss
  - Timeout or duplicated loss
  - Congestion control is coupled with reliable transfer
- AIMD-based congestion window adaptation
  - Fairness v.s. efficiency
- Slow start for fast convergence
- Fast retransmission/recovery based on duplicated ACK
  - `cwnd ← cwnd/2 + 3MSS`

> The lost segment starting at SND.UNA MUST be retransmitted and cwnd set to ssthresh plus 3*SMSS. This artificially "inflates" the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.
>
> —RFC 5681

Let’s try it!

Consider the evolution of a TCP connection with the following characteristics. Assume that all the following algorithms are implemented in TCP congestion control: slow start, congestions avoidance, fast retransmit and fast recovery, and retransmission upon timeout. **If ssthresh equals to cwnd, use the slow start algorithm in your calculation.**

- The receiver acknowledges every segment, and the sender always has data available for transmission.
- Initially ssthresh at the sender is set to 6. Assume cwnd and ssthresh are measured in segments, and the transmission time for each segment is negligible. Retransmission timeout (RTO) is initially set to 500ms at the sender and is **unchanged** during the connection lifetime. The RTT is 100ms for all transmissions.
- The connection starts to transmit data at time t = 0, and the initial sequence number starts from 1. **Segment with sequence number 4 is lost once**. No other segments are lost.

How long does it take, in milliseconds, for the sender to receive the ACK for the segment with the sequence number 12? show your intermediate steps or your diagram.

References

- MIT 18.996: Topic in TCS: Internet Research Problems
- Princeton ELE539A: Optimization of Communication Systems
- http://www.freesoft.org/CIE/Course/Section4/