CS118 Discussion 1D, Week 5

Zengwen Yuan
Boelter Hall 2760, Friday 4:00—5:50 p.m.
Outline

• Change of my office hours: Tue 2 — 4 p.m. Engr VI 392

• Lecture review: TCP handshake; congestion control

• Midterm next week!
  • Thursday, Feb. 15th, in-class
  • closed book, no electronic devices
  • one two-sided US letter size (8.5x11) cheatsheet is allowed
  • Sample midterm
TCP 3-way handshake

**Client state**

- **LISTEN**
  - choose init seq num, \( x \)
  - send TCP SYN msg

- **SYNSENT**
  - received SYNACK(x)
  - ACKbit=1; ACKnum=x+1
  - indicates server is live; send ACK for SYNACK; this segment may contain client-to-server data

- **ESTAB**

**Server state**

- **LISTEN**

- **SYN RCVD**
  - choose init seq num, \( y \)
  - send TCP SYNACK msg, acking SYN

- **ESTAB**
  - received ACK(y)
  - ACKbit=1, ACKnum=y+1
  - indicates client is live
On TCP handshake sequence numbers

1) A --> B  SYN my sequence number is X
2) A <-- B  ACK your sequence number is X
3) A <-- B  SYN my sequence number is Y
4) A --> B  ACK your sequence number is Y

• Because steps 2 and 3 can be combined in a single message this is called the three way (or three message) handshake.

• How X and Y are chosen? Not specified; could be random numbers (using clock), as this is more secure. [RFC 793]
  https://tools.ietf.org/html/rfc793#section-3.3
  https://support.microsoft.com/en-us/help/172983/
An HTTP 1.0 connection example
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) = $W \times \text{Message Size}/\text{RTT}$
  - 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

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 $cwnd \geq ssthresh$

- Linear growth
  - each RTT:
    - $cwnd \leftarrow cwnd + 1$ (MSS)
  - equivalently, each ACK:
    - $cwnd \leftarrow cwnd + 1/cwnd$ (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 $ssthresh \leftarrow \frac{cwnd}{2}$
  - set $cwnd \leftarrow ssthresh + 3 \text{ (MSS)}$
  - upon additional dup ACK: grow $cwnd$ linearly
  - New ACK: $cwnd \leftarrow ssthresh$

- Time Out
  - set $ssthresh \leftarrow \frac{cwnd}{2}$
  - set $cwnd \leftarrow 1 \text{ (MSS)}$
  - 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
  - \[ cwnd \leftarrow \frac{cwnd}{2} + 3MSS \]
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 5. 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 5 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 16? 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/