TCP review

- Connection establishment is “3-way handshake”
- Closing connections: both sides must send FIN, and ack that FIN
  - timeout possible
  - can be messy
- Example: HTTP session

“Messy” connection closes

- Other side never sends FIN
  - sender stuck in FIN_WAIT_2, receiver stuck in CLOSE_WAIT
- One side sends RST
  - meant for: “illegal” connections, refused connections, abnormal termination of connection, ...
  - older browsers, some apps do this by default (bad!)
  - problem: what state are you in now?

Today

- Sending and receiving data over TCP
  - return of the sliding window
- Slow start
- Retransmissions
- Congestion control
TCP's sliding window

- Updated with every transmission in win field
- Indicates \(<\text{size of receive buffer}> - \langle\text{number of bytes not yet read by application}\rangle\)
- Note: in tcpdump files from Friday, window size actually increased as session went on!
  - we'll explain this in a minute
- Sender: max bytes to send \(\leq\) receiver's window – number of bytes not yet acked
  - if no room in receive buffer, sending TCP process blocks

Sliding window: issues

- Zero-sized advertised window
  - problem: how does the sender determine when the buffer is no longer full?
  - solution: send periodic “probe” packet (1 byte of data)
- Small sending window
  - problem: should the sender always send a packet if the receiver has room in its buffer?
  - silly window syndrome

Nagle algorithm

- Designed to counteract silly window syndrome, by indicating when packets can and should be sent
- Algorithm:
  - always send a full packet (MSS) if the window allows
  - send small packet if there are no outstanding acks
  - else, wait for an ack before sending the next packet
- Can be disabled

Q: How does the sender know how quickly to send packets into the network?
Slow start

- Idea: send packets into the network at the rate at which they are acked
  - start slow, to avoid disturbing the network
  - gradually build up send rate, up to some maximum value
- Avoid introducing or exacerbating congestion
  - congestion: situation in which there is more network traffic than the network can handle

Slow start algorithm

- Congestion window (cwnd): size (in bytes) of the current “send” window
- Always starts at 1 segment
- Additive increase, multiplicative decrease (AIMD)
  - increase cwnd by one segment for each segment acked
  - decrease: ?
- Sender can send up to \( \min(\text{cwnd}, \text{win}) \)

Slow start: example

Multiplicative decrease...?

- We need a way to throttle back transmission rates if we detect congestion on the network
  - timeout
- Idea:
  - set cwnd = cwnd/2
  - each time ack received, set cwnd = cwnd + MSS (MSS/cwnd)
  - additive increase
Q: How is TCP's retransmit timer calculated?

Retransmit timer

- Based on current RTT
- Original algorithm:
  - $eRTT = \alpha eRTT + (1-\alpha) sRTT$
  - $TO = 2 \cdot eRTT$
  - $eRTT$: estimated RTT
  - $sRTT$: sample RTT (last measured RTT)
  - $\alpha$: “smoothing parameter” (0.8-0.9)

Retransmit timer (cont.)

- Problem: RTTs for retransmitted packets throw off the estimate
- Solution: two modifications
  - Karn/Partridge: when retransmission occurs, set $eRTT = 2 \cdot eRTT$
  - Jacobson/Karels: calculate both the mean and variance of the RTT, then $TO = eRTT + 4 \cdot \text{variance}$
Fast retransmit/fast recovery

- If a packet is received out of order, the receiver sends an ack for the last packet received
- Idea: preempt the retransmit timer when duplicate acks are observed
- Fast retransmit:
  - occurs when 3 duplicate acks are received in a row
  - retransmit “missing” packet immediately
- Fast recovery: additive increase

Summary

- TCP uses a sliding window mechanism for sending data
- Data flow is controlled by
  - slow start
  - AIMD (congestion avoidance)
  - receiver's advertised window
  - fast recovery/retransmit
- Nagle algorithm helps us avoid sending too little data (silly window syndrome)