TCP – Data Exchange

Reading: Section 24.4

Flow Control

- Preventing overflow at receiver
  - Slower receivers
  - Receiver fetches data at some fixed interval, such as video player
- TCP
  - Sliding window
    - In-order delivery
    - Catching missing/duplicate packets
    - Controlling sending credits

Protocol support for flow control

- Header carrying sequence number
  - To syn sender/receiver
- Header carrying window size
  - How much traffic (bytes) another party can send
- Header carrying ack
  - What has been received

Sequence number space

Sender:

<table>
<thead>
<tr>
<th>Acked</th>
<th>Not Acked</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Receiver:

<table>
<thead>
<tr>
<th>Acked</th>
<th>Not Acked</th>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Specifying transmit credit

- tcp_receive_window()
  - Computing remaining size of receiving window
  - \( tp->rcv\textunderscore wnd - (tp->rcv\textunderscore nxt - tp->rcv\textunderscore wup) \)
- tcp_select_window()
  - Checking available memory for receive buffer
  - How much free space do we have?
  - If free space is smaller than MSS
    - Return 0
  - Else,
    - Computing window size as multiples of MSS.

Zero window probing

- If receiver does not have buffer, it will send a zero window size
- Deadlock can occur
  - Receiver has no data to send (window size can be announced)
  - Both sizes have announced zero window size
- Zero-window probing
  - Timer is maintained after receiving zero window size
  - Send a probing packet after expiration
  - Receiver acks this packet, which may have a non-window size, if free buffer available

Specifying transmit credit

- tcp_select_window()
  - Compute current window size
    - (tcp_receive_window())
  - Compute new window size
    - (_tcp_select_window())
  - If new_win < cur_win
    - new_win = cur_win
  - Storing window size locally
  - Some other processing

tcp_probe_timer()

- Called when probe timer goes off
- Checking if maximum number of probes sent
  - If so, report error
- Otherwise,
  - Send probe packet
  - \( (tcp\textunderscore send\textunderscore probe0()) \)
tcp_send_probe0()

- Send a probe packet (tcp_write_wakeup())
- Checking if we need to send probe again
  - We do not need to if
    - No data to be sent
    - Some non ACKed data packet
  - Otherwise
    - We need to reset the probe timer

tcp_write_wakeup()

- If we can send data (we have data, and window size is not zero)
  - Send the data packet (tcp_transmit_skb())
- Otherwise
  - Send probing packet (tcp_xmit_probe_skb())

tcp_xmit_probe_skb()

- Send a packet without payload
- Allocate a skb with TCP header size
- Set ACK flag
- Filling in an OLD sequence number
- Another party sees old sequence number
  - It will ACK, possibly with a non zero window size

tcp_ack_probe()

- Handling (possible) ack to probe packet
- If receive window is opened,
  - Stop probe timer
- Otherwise
  - Restart probe timer
Handling congestion

- Detection
  - Retransmission timer goes off
  - Slow start phase
- Duplicate acks
  - Slow start in old version of TCP
  - Fast retransmission and fast recovery in newer version

Slow start and congestion avoidance

- Learning/adapting to available bandwidth
- Slow start
  - When a connection first starts
  - When retransmission timer goes off
- Parameters
  - Congestion window (snd_cwnd)
    - Number of bytes sent without ACKs
  - Slow-start threshold (ssthresh)
    - From exponential increase to linear increase
    - Entering congestion avoidance phase
    - When a packet gets lost (timer goes off)
      - Ssthresh is reset to half of the current congestion window size

Implementation

- tcp_v4_init_sock()
  - Set parameters
    - snd_cwnd is set to 2 instead of 1
    - ssthresh is set to infinite (0xffffffff)
- tcp_cong_avoid()
  - Slow start and congestion avoidance algorithm
  - Checking which phase we are in
    - If slow-start (snd_cwnd <= sssthresh)
      - snd_cwnd++
    - Otherwise
      - Counting how many ACKs we received
      - Increasing snd_cwnd by 1 only if count is larger than cwnd

- tcp_enter_loss()
  - Called when a retransmission timer goes off
  - Reset ssthresh to a new value
    - Half of the current congestion window size
  - Reset the current congestion window size to 1 (theoretically)
  - Starts in slow start phase again
Fast retransmission

- Fast retransmission
  - In new version of TCP to early detect packet loss
  - Receiving duplicate ACK, which means
  - Some out-of-order packets arrived at receiver
  - Some packet gets lost
  - Sender can retransmit packets instead of waiting for timer to expire

Fast recovery

- Upon receiving duplicate ACKs
  - Some packet possibly gets lost
  - Outrageous congestion is unlikely.
  - Don’t go to slow start
  - Instead directly go to congestion avoidance
    - Cwnd = ssthresh

How they work

- When receiving three duplicate ACKs
  - Set ssthresh to half of current congestion window size
  - Retransmit missing packet
  - snd_cwnd = ssthresh + 3 MSS
  - For each duplicate ACK
    - Increase cwnd by one, send additional data
  - When new ACK arrives
    - Reset cwnd to ssthresh

Misc

- Delayed ACK
  - We do not need to ACK immediately for each packet
  - TCP uses cumulative ACK
  - ACK timer
- Nagle algorithm
  - Small-packet-avoidance algorithm
  - For application like telnet, every time we may only a couple of characters of data
  - Send to the first
  - While waiting for ACK, do not send data
  - Instead, accumulate data and send them together.