TCP Overview

Kurose & Ross, Chapter 3 (5th ed.)

Many slides adapted from:
J. Kurose & K. Ross
Computer Networking: A Top Down Approach (5th ed.)
Addison-Wesley, April 2009.
Copyright 1996-2010, J.F Kurose and K.W. Ross, All Rights Reserved.

TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order byte stream:
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- send & receive buffers:
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP segment structure

TCP seq. #s and ACKs

Seq. #s:
- byte stream "number" of first byte in segment's data

ACKs:
- seq # of next byte expected from other side
- cumulative ACK

Q: how TCP handles out-of-order segments
- A: TCP spec doesn't say, up to implementor

TCP Round Trip Time and Timeout

Q: how to set TCP timeout value?
- longer than RTT
  - but RTT varies
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

Q: how to estimate RTT?
- SampleRTT: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- SampleRTT will vary, want estimated RTT "smoother"
  - average several recent measurements, not just current SampleRTT

Transport Layer 3-3
Transport Layer 3-4
Transport Layer 3-5
Transport Layer 3-6
TCP Round Trip Time and Timeout

EstimatedRTT = \((1-\alpha)\text{EstimatedRTT} + \alpha\text{SampleRTT}\)

- Exponential weighted moving average
- Influence of past sample decreases exponentially fast
- Typical value: \(\alpha = 0.125\)

TCP Round Trip Time and Timeout

Setting the timeout

- EstimatedRTT plus "safety margin"
  - Large variation in EstimatedRTT -> larger safety margin
  - First estimate of how much SampleRTT deviates from EstimatedRTT:
    \[ \text{DevRTT} = (1-\beta)\text{DevRTT} + \beta(|\text{SampleRTT} - \text{EstimatedRTT}|) \]
  (typically, \(\beta = 0.25\))

Then set timeout interval:

\[ \text{TimeoutInterval} = \text{EstimatedRTT} + 4\text{DevRTT} \]

TCP Connection Management

Recall: TCP sender, receiver establish "connection" before exchanging data segments
- Initialize TCP variables:
  - Seq. #s
  - Buffers, flow control info (e.g., RcvWindow)
- Client: connection initiator
  - Socket clientSocket = new Socket("hostname", "port number");
  - Step 1: client host sends TCP SYN segment to server
    - Specifies initial seq #
    - No data
  - Step 2: server host receives SYN, replies with SYNACK segment
    - Server allocates buffers
    - Specifies server initial seq #
  - Step 3: client receives SYNACK, replies with ACK segment, which may contain data

Closing a connection:

- Client closes socket:
  - clientSocket.close();
- Step 1: client closes socket, sends TCP FIN control segment to server
- Server receives FIN, replies with ACK, closes connection, sends FIN.
TCP Connection Management (cont.)

**Step 3:** client receives FIN, replies with ACK,
- Enters "timed wait" - will respond with ACK to received FINs

**Step 4:** server, receives ACK. Connection closed.

**Note:** with small modification, can handle simultaneous FINs.

TCP's statechart

- **On board**
  - Statechart appears in RFC 793
- **Discussion of:**
  - TIME_WAIT state
    - Connection in TIME_WAIT state cannot move to the CLOSED state until it has waited for two times the maximum segment lifetime (MSL).
    - Why? We do not know whether the ack sent in response to the other side’s FIN was delivered. The other side might retransmit its FIN segment.
    - The delayed FIN from the previous incarnation terminates the later incarnation of the same connection.
    - Because only a connection between the same endpoints can cause the confusion, only one endpoint needs to hold the state.
  - Syn flood attacks

TCP ACK generation [RFC 1122, RFC 2581]

<table>
<thead>
<tr>
<th>Event at Receiver</th>
<th>TCP Receiver action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed</td>
<td>Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK</td>
</tr>
<tr>
<td>Arrival of in-order segment with expected seq #. One other segment has ACK pending</td>
<td>Immediately send single cumulative ACK, ACKing both in-order segments</td>
</tr>
<tr>
<td>Arrival of out-of-order segment higher-than-expect seq #. Gap detected</td>
<td>Immediately send duplicate ACK, indicating seq. # of next expected byte</td>
</tr>
<tr>
<td>Arrival of segment that partially or completely fills gap</td>
<td>Immediate send ACK, provided that segment starts at lower end of gap</td>
</tr>
</tbody>
</table>

Fast Retransmit

- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.
- if sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires
Fast retransmit algorithm:

- Event: ACK received, with ACK field value of y
  - if (y > SendBase) {
    - SendBase = y
    - start timer
  }
  - else {
    - increment count of dup ACKs received for y
    - if (count of dup ACKs received for y = 3) {
      - resend segment with sequence number y
    }

TCP Flow Control

- receive side of TCP connection has a receive buffer:
  - receiver won't overflow receiver's buffer by transmitting too much too fast

- app process may be slow at reading from buffer

TCP Flow control: how it works

- receiver advertises spare room by including value of RcvWindow in segments
- sender limits unACKed data to RcvWindow
  - guarantees receive buffer doesn't overflow

(suppose TCP receiver discards out-of-order segments)
- spare room in buffer = RcvWindow
  - RcvBuffer = RcvBuffer - [LastByteRcvd - LastByteRead]