Transmission Control Protocol (TCP)

Srinidhi Varadarajan

TCP: Overview

- **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

TCP: Reliable Data Transfer

Simplified TCP sender

Simplified TCP receiver

TCP: Reliable Data Transfer

RFCs: 793, 1122, 1323, 2018, 2581

TCP seq. #’s:
- byte stream “number” of first byte in segment's data
- seq # of next byte expected from other side
- cumulative ACK
- how receiver handles out-of-order segments
  - TCP spec doesn’t say, up to implementation

Q: how receiver handles ACKs:
- cumulative ACK
  - seq # of next byte
- A: TCP spec
  - send buffer
  - steam:
  - application
  - no “message one sender, one receiver
  - bi-directional data flow in same connection
  - MSS: maximum segment size
  - handshaking (exchange of control msgs) init’s sender, receiver state before data exchange
  - sender will not overwhelm receiver

TCP: Reliable Data Transfer

Simplified TCP sender

Simplified TCP receiver

TCP: Reliable Data Transfer

RFCs: 793, 1122, 1323, 2018, 2581
TCP ACK generation [RFC 1122, RFC 2581]

<table>
<thead>
<tr>
<th>Event</th>
<th>TCP Receiver action</th>
</tr>
</thead>
<tbody>
<tr>
<td>in-order segment arrival, no gaps, everything else already ACKed</td>
<td>delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK</td>
</tr>
<tr>
<td>in-order segment arrival, no gaps, one delayed ACK pending</td>
<td>immediately send single cumulative ACK</td>
</tr>
<tr>
<td>out-of-order segment arrival higher than expected seq # gap detected</td>
<td>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 ACK if segment starts at lower end of gap</td>
</tr>
</tbody>
</table>

TCP: retransmission scenarios

TCP Flow Control

- Flow control
  - sender won’t ever use receiver’s buffers by transmitting too much too fast
  - receiver: explicitly informs sender of (dynamically changing) amount of free buffer space
    - RcvWindow field in TCP segment
  - sender: keeps the amount of transmitted, unACKed data less than most recently received RcvWindow

TCP Round Trip Time and Timeout

- **Q**: how to set TCP timeout value?
  - longer than RTT: note: RTT will vary
  - too short: premature timeout
  - too long: slow reaction to segment loss

- **Q**: how to estimate RTT?
  - SampleRTT: measured time from segment transmission until ACK receipt
    - ignore retransmissions, cumulatively ACKed segments
  - SampleRTT will vary, want estimated RTT “smoother”
    - use several recent measurements, not just current SampleRTT

TCP Round Trip Time and Timeout

EstimatedRTT = (1-x) * EstimatedRTT + x * SampleRTT
- Exponential weighted moving average
- Influence of given sample decreases exponentially fast
- Typical value of x: 0.1

Setting the timeout
- EstimatedRTT plus “safety margin”
- large variation in EstimatedRTT -> larger safety margin

Timeout = EstimatedRTT + #Deviation
Deviation = (1-x) * Deviation + x * |SampleRTT-EstimatedRTT|

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");
- server: contacted by client
  - Socket connectionSocket = serverSocket.accept();

**Three way handshake**:

**Step 1**: client and system sends TCP SYN control segment to server
- specifies initial seq #

**Step 2**: server and system receives SYN, replies with SYNACK control segment
- ACKs received SYN
- allocates buffers
- specifies server-> receiver initial seq #

**Step 3**: client receives SYNACK control segment
- ACKs received SYN
- SYN=0, connection has been established.
- Client data may be piggybacked
TCP Connection Management (cont.)

**Closing a connection:**
- **client** closes socket: `clientSocket.close();`

**Step 1:** client end system sends TCP FIN control segment to server

**Step 2:** server receives FIN, replies with ACK. Closes connection, sends FIN.

**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.