TCP

**TCP: Overview**

- **Point-to-point:**
  - one sender, one receiver
  - Reliable, in-order byte steam:
    - 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: sender, receiver state before data exchange
- **Flow controlled:**
  - Sender will not overwhelm receiver
- **Point-to-point:**
  - One sender, one receiver
  - Reliable, in-order byte steam:
- **Steam:**
  - No "message boundaries"
  - Pipelined:
    - TCP congestion and flow control set window size
    - Send & receive buffers
- **Send & receive buffer:**

**TCP Segment Structure**

- Source port #
- Destination port #
- Sequence number
- Acknowledgement number
- ACK: Acknowledgement number
- Urg: Urgent pointer
- PSH: Push data now
- RST, SYN, FIN: Reset, Synchronize, Finish
- Options: Variable length
- Application data: Variable length
- Internet checksum
- TCP checksum

**TCP Seq. #’s and ACKs**

- Seq #: Byte stream "number" of first byte in segment's data
- ACK: Acknowledgement number
- Urg: Urgent pointer
- PSH: Push data now
- Seq #: Of next byte expected from other side
- RST, SYN, FIN: Reset, Synchronize, Finish
- Seq #: Standard TCP seq number
- Ack #: Cumulative ACK

**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

**Example RTT Estimation:**

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>RTT (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>36</td>
</tr>
<tr>
<td>6</td>
<td>43</td>
</tr>
<tr>
<td>7</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>57</td>
</tr>
<tr>
<td>9</td>
<td>64</td>
</tr>
<tr>
<td>10</td>
<td>71</td>
</tr>
<tr>
<td>11</td>
<td>78</td>
</tr>
<tr>
<td>12</td>
<td>85</td>
</tr>
<tr>
<td>13</td>
<td>92</td>
</tr>
<tr>
<td>14</td>
<td>99</td>
</tr>
<tr>
<td>15</td>
<td>106</td>
</tr>
</tbody>
</table>

**Transport Layer**

- 3-1
- 3-2
- 3-3
- 3-4
- 3-5
- 3-6
TCP reliable data transfer

- TCP creates rdt service on top of IP's unreliable service
- Retransmissions are triggered by:
  - timeout events
  - duplicate acks
- Pipelined segments
- Cumulative acks
- TCP uses single retransmission timer
- TCP creates rdt service on top of IP's unreliable service
- Retransmissions are triggered by:
  - timeout events
  - duplicate acks
- Pipelined segments
- Cumulative acks
- TCP uses single retransmission timer

TCP sender events:

- data rcvd from app:
  - Create segment with seq #
  - seq # is byte-stream number of first data byte in segment
- timeout:
  - retransmit segment that caused timeout
  - restart timer

TCP: retransmission scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host A Seq=100, 20 bytes data</td>
<td><img src="image" alt="Diagram of TCP retransmission scenarios" /></td>
</tr>
<tr>
<td>Host B Seq=92, 8 bytes data</td>
<td><img src="image" alt="Diagram of TCP retransmission scenarios" /></td>
</tr>
<tr>
<td>Loss</td>
<td>Host A Seq=92, 8 bytes data</td>
</tr>
<tr>
<td>Lost ACK scenario</td>
<td><img src="image" alt="Diagram of TCP retransmission scenarios" /></td>
</tr>
</tbody>
</table>

TCP ACK generation (more)

<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 # 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 #</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>

TCP ACK generation [RFC 1122, RFC 2581]

TCP ACK generation [RFC 1122, RFC 2581]
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

TCP Flow Control

- receive side of TCP connection has a receive buffer:
  - Rcvr advertises spare room by including value of RcvWindow in segments
  - Sender limits unACKed data to RcvWindow

TCP Flow control: how it works

(Remember, TCP receiver discards out-of-order segments)
- spare room in buffer
  - RcvrBuffer = [LastByteRcvd - LastByteRead]

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 = welcomeSocket.accept();

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.
Transport Layer

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.