Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control
**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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port #</td>
<td>Source port number</td>
</tr>
<tr>
<td>Dest Port #</td>
<td>Destination port number</td>
</tr>
<tr>
<td>Sequence Number</td>
<td>Sequence number of the segment</td>
</tr>
<tr>
<td>Acknowledgement Number</td>
<td>Acknowledgment of the sequence number</td>
</tr>
<tr>
<td>Head len</td>
<td>Length of the header</td>
</tr>
<tr>
<td>Urg len</td>
<td>Length of the urgent data pointer</td>
</tr>
<tr>
<td>URG</td>
<td>URG (urgent data)</td>
</tr>
<tr>
<td>ACK</td>
<td>ACK (acknowledgement)</td>
</tr>
<tr>
<td>PSH</td>
<td>PSH (push data)</td>
</tr>
<tr>
<td>RST, SYN, FIN</td>
<td>RST, SYN, FIN (reset, synchronize, finish)</td>
</tr>
<tr>
<td>Receive Window</td>
<td>Number of bytes the receiver is willing to accept</td>
</tr>
<tr>
<td>Checksum</td>
<td>Internet checksum</td>
</tr>
<tr>
<td>Urg Data Punter</td>
<td>Urgent data pointer</td>
</tr>
<tr>
<td>Options</td>
<td>Options (variable length)</td>
</tr>
<tr>
<td>Application Data</td>
<td>Application data (variable length)</td>
</tr>
</tbody>
</table>

**URG:** urgent data (generally not used)

**ACK:** ACK # valid

**PSH:** push data now (generally not used)

**RST, SYN, FIN:** connection establish (setup, teardown commands)

**Internet checksum** (as in UDP)
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 receiver handles out-of-order segments
- A: TCP spec doesn’t say, - up to implementation
**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*
TCP Round Trip Time and Timeout

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

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

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr

SampleRTT
Estimated RTT
Setting the timeout

- **EstimatedRTT** plus “safety margin”
  - large variation in EstimatedRTT -> larger safety margin

- first estimate of how much SampleRTT deviates from EstimatedRTT:
  \[ \text{DevRTT} = (1 - \beta) \times \text{DevRTT} + \beta \times |\text{SampleRTT} - \text{EstimatedRTT}| \]

(typically, $\beta = 0.25$)

Then set timeout interval:

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