**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();

Three way handshake:

**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

---

**TCP segment structure**

- source port
- dest port
- sequence number
- acknowledgment number
- options (variable length)
- Internet checksum (as in UDP)
- Urg data pointer
- RST, SYN, FIN: connection estab (setup, teardown commands)
- timestamp
- window size
- checksum
- urgent pointer
- reserved
- flags

---

**Transport Layer 3-4**

**Three-way handshake**

<table>
<thead>
<tr>
<th>Time</th>
<th>Host 1</th>
<th>Host 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN</td>
<td>SYN =1 (SEQ = x)</td>
<td>SYN = 1 ACK = 1 (SEQ = y, ACK = x+1)</td>
</tr>
<tr>
<td>ACK</td>
<td>ACK = 1 (SEQ = x+1, ACK = y+1)</td>
<td>ACK = 1 (SEQ = y+1)</td>
</tr>
</tbody>
</table>

---

**TCP: Overview**

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

---

**RFCs: 793, 1122, 1323, 2018, 2581**

- RFCs: 793, 1122, 1323, 2018, 2581

---

**Transport Layer 3-3**

**TCP segment structure**

- source port
- dest port
- sequence number
- acknowledgment number
- options (variable length)
- Internet checksum (as in UDP)
- Urg data pointer
- RST, SYN, FIN: connection estab (setup, teardown commands)
- timestamp
- window size
- checksum
- urgent pointer
- reserved
- flags

---

**Transport Layer 3-4**

**Three-way handshake**

<table>
<thead>
<tr>
<th>Time</th>
<th>Host 1</th>
<th>Host 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN</td>
<td>SYN =1 (SEQ = x)</td>
<td>SYN = 1 ACK = 1 (SEQ = y, ACK = x+1)</td>
</tr>
<tr>
<td>ACK</td>
<td>ACK = 1 (SEQ = x+1, ACK = y+1)</td>
<td>ACK = 1 (SEQ = y+1)</td>
</tr>
</tbody>
</table>

---

**TCP Connection Management**

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 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.

TCP's statechart

- On board
  - Statechart appears in RFC 793
- Discussion of:
  - TIME_WAIT state
  - Syn flood attacks