Announcements

- **Reading**
  - Today: 6.3-6.4
  - Tuesday (after spring break): 6.5-6.6
Multiplexing in the Transport Layer

- **Upward multiplexing**
  - putting multiple transport connections onto one network connection
  - used to accommodate pricing strategies that charge for connections

- **Downward multiplexing**
  - using several network connections per transport connection
  - permits use of multiple copies of network resources
    - if the network layer uses sliding windows
      - a high latency network may under utilize the link
      - multiple connections each get a window
    - per connection buffer allocation
      - get more buffers
    - round-robin scheduling
      - get a larger share of link bandwidth
Crash Recovery

- **Router or Link Crashes**
  - Data in transit can be lost.
  - End nodes have sufficient state to recover lost data.
  - Transport protocol can hide network failures from the application.

- **Host Crashes**
  - Transport level state will be lost at one end.
  - Does the transport layer have sufficient info to recover?, **No!**
    - Information must flow down to network and up to transport user
      - ACKs go down, and data goes up.
      - It is not possible to make these two operations atomic.
    - Lack of stable storage causes this problem
  - Result, higher up layer must deal with host crashes
Protocol State Machines

Established
- <Connect, ~P1>, A3
- <Clear_req, *>, A4
- <DISCON, P4>, A5

Waiting
- <DISCON, P4>, A5
- <Call_acc, *>, A7

Sending
- <SEND, ~P5>, A8
- <Clear_req, *>, A10
- <Credit, *>, A11

Disconnecting
- <clear_Req, *>
- <clear_Conf, *>

Receiving
- <SEND, P5>, A7
- <Clear_req, *>, A10
- <Credit, *>, A11

<LISTEN, P1>, A2

<CONNECT, P1>

<LISTEN, ~P2>, A1
# Predicates And State Transitions

<table>
<thead>
<tr>
<th>Pred</th>
<th>Meaning</th>
<th>Act</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Connection table full</td>
<td>A1</td>
<td>Send Call_acc</td>
</tr>
<tr>
<td>P2</td>
<td>Call_req pending</td>
<td>A2</td>
<td>Wait for Call_req</td>
</tr>
<tr>
<td>P3</td>
<td>LISTEN Pending</td>
<td>A3</td>
<td>Send Call_req</td>
</tr>
<tr>
<td>P4</td>
<td>Clear_req Pending</td>
<td>A4</td>
<td>Start Timer</td>
</tr>
<tr>
<td>P5</td>
<td>Credit Available</td>
<td>A5</td>
<td>Send Clear_conf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A6</td>
<td>Send Clear_req</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A7</td>
<td>Send message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A8</td>
<td>Wait for credit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A9</td>
<td>Send Credit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A10</td>
<td>Set Clr_req_recv flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A11</td>
<td>Record credit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A12</td>
<td>Accept message</td>
</tr>
</tbody>
</table>
TCP Protocol

- **TSAPs**
  - Use <host, port> combination
  - Well known ports provide services
    - first 256 ports
    - SMTP 25, Telnet 23, Ftp 21, HTTP 80
- **Provides a byte stream**
  - this is *not* a message stream
  - a message (single call to send) may be split, merged, etc.
- **Urgent Data field**
  - provides cut through delivery *within* a transport connection
  - used to send breaks or other high priority info
TCP Packet Format

- **Permits ACKs to be piggy packed**
  - ACK is next byte expected
  - ACK is only valid if ACK bit is set
- **Sequence number**
  - first byte in packet
- **Also used for connection establishment**

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sequence Number</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Acknowledgment Number</strong></td>
<td>Window Size</td>
</tr>
<tr>
<td><strong>Checksum</strong></td>
<td><strong>Urgent Pointer</strong></td>
</tr>
</tbody>
</table>

0 Or More Options

32 bits
TCP Connection Management

- Three-way Handshake
- Initial Sequence Numbers
  - Use a 4 micro-second clock
  - hosts must wait T (120 seconds) before a reboot
- Connection Closure
  - Each side uses a FIN and FIN_ACK message
  - A FIN times out after 2 T (240 seconds)
  - Keep alives used to timeout half dead connections
TCP Flow Control

- **Use Variable Sized Sliding Window**
  - ACK indicates start of window
  - Window size indicates current size of window

- **Receiver can send a window of 0**
  - indicates that it wants to pause the connection
  - urgent data need not follow this request

- **Window size of 16 bits is too small**
  - 64K Bytes
  - only a small fraction of the in-flight bytes when
    - bandwidth is high
    - delay is high
  - solution: window shift option:
    - bit shift window up to 16 bits
    - permits up to $2^{32}$ byte windows
    - reduces window granularity