Transmission Control Protocol (TCP)

- Protocol:
  - Chapter 25
- Sockets programming
  - Practical Guide, section 2.2 (simple iterative server)
  - Practical Guide, section 6.4 (concurrent server)

Services Offered to an Application

- Connection orientation
- Point-to-Point
- Reliability
- Full duplex
- Stream interface
- Reliable startup
- Graceful connection shutdown
Transmission Control Protocol (TCP)

TCP is an end-to-end protocol because it provides a connection directly from one application to another running on a remote computer.

The connections are virtual connections because they are achieved in software.

Based on notes by Jim Martin

IP Datagram Format: UDP example

This is what gets sent on the wire: A frame which contains an IP Packet.

Based on notes by Jim Martin
TCP Datagram Format

Packet Loss and Retransmission

Example of retransmission. Items on the left correspond to events in a computer sending data, items on the right correspond to events in a computer receiving data, and time goes down the figure. The sender retransmits lost data.

Based on notes by Jim Martin
Transmission Control Protocol (TCP)

- Error detection/recovery
- Adaptive retransmission:
  - If the retransmission timeout is too low, you might retransmit unnecessarily.
  - If the timeout is too high, user perceived performance can be very bad.

Flow Control

- A TCP receiver allocates a receive buffer.
- As data arrives (and fills the buffer), the receiver sends ACKs that also specify the remaining buffer size.
- The amount of buffer space available at any time is the window and the notification that specifies the size is the window advertisement.

Based on notes by Jim Martin
Flow Control

**Network Congestion Control**

What if Host A sends many back-to-back UDP Echo Messages to Host B?

TCP's base congestion control algorithm based on a dynamic window (slow start):

- cwnd = 1; //Init the congestion window to 1 segment
- wnd = min(cwnd, advertised window) //The actual window used...
- On a timeout, cwnd = 1
- When a new ACK arrives, cwnd += 1;

Leads to TCP’s “saw-tooth behavior”
TCP Sockets Programming

TCP Client

- socket()
- connect()
- write()
- read()
- close()

TCP Server

- socket()
- bind()
- listen()
- accept()
- read()
- write()
- close()

Based on notes by Jim Martin

Server Side Calls: Listen and Accept

- **Listen**
  - Converts an active socket into a passive socket meaning the kernel should accept incoming connection requests.
  - Sets the maximum number of connections the kernel should queue for this socket.
  
  ```c
  int listen (int sockfd, int backlog)
  ```
  - There are two queues:
    - Incomplete connection queue
    - Completed connection queue
  - The backlog (roughly) indicates the sum of the two queues

- **Accept** returns a connection from the completed queue
  
  ```c
  int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen)
  ```

  Returns a new socket descriptor.
  - Server will have at least a listenfd and a connectfd socket descriptors

Based on notes by Jim Martin
TCP startup/tear down

- **TCP connection setup:** 3 way handshake
  - guarantees that both sides are ready to transfer data (handles simultaneous opens)
  - Allows both sides to agree on initial sequence numbers (ISNs).

- **TCP connection termination:**
  - modified 3 way handshake
  - The TCP close requires 4 flows rather than 3
  - A FIN leads to an EOF to a receiving application
  - Supports a half close: one side terminates its output but still receives data from the other side.
  - Sockets supports this but most applications don’t use it.

Based on notes by Jim Martin
TCP Acknowledgement Strategy

- Acks indicate the next byte the receiver expects (we show the next packet that is expected)
- Acks are cumulative
  - Easy to generate
  - But they are ambiguous.
  - Lost Acks don’t necessarily force a retransmission.
- Ack strategy: an Ack is typically generated for every 2 segments that arrive

Congestion Avoidance Algorithm

- Combined slow start and congestion avoidance algorithm (TCP-Tahoe)
- Create a second state variable, ssthresh, to switch between the two algorithms.
- Assume the wnd = min(cwnd, advertised window)
- On a timeout
  - ssthresh = wnd / 2 (min value of 2 segments)
  - cwnd = 1
- When a new ACK arrives
  - if (cwnd <= ssthresh)
    - /*open the window exponentially*/
    - cwnd += 1;
  - else
    - /*otherwise do Congestion Avoidance-
      increment linearly*/
    - cwnd += 1/cwnd;

Based on notes by Jim Martin
Fast Recovery, Fast Retransmit Algorithms

- **Fast Retransmit:**
  - If three or more duplicate ACKs arrive at a sender, this is a strong indicator that a packet was dropped.
  - The sender retransmits without waiting for the retransmit timer.
- **Fast Recovery:** After a fast retransmit, the sender goes to congestion avoidance rather than slow-start.

- **Enhanced algorithm:**
  - When the third duplicate ACK is received
    - set \( ssthresh = \frac{cwnd}{2} \)
    - retransmit the segment
  - For each additional duplicate ACK (after the third duplicate ACK), increment \( cwnd \) by 1 and transmit a new packet (if allowed by the new \( cwnd \) value).
  - When the next ACK arrives that acknowledges new data, set \( cwnd \) to \( ssthresh \).

Based on notes by Jim Martin

TCP Dump Trace

<table>
<thead>
<tr>
<th>Time</th>
<th>Source IP</th>
<th>Destination IP</th>
<th>Data</th>
<th>Sequence Number</th>
<th>Acknowledgment Number</th>
<th>Window Size</th>
<th>MSS</th>
<th>DF</th>
<th>ToS</th>
</tr>
</thead>
<tbody>
<tr>
<td>982120155.354974</td>
<td>192.168.1.100.1279</td>
<td>192.168.2.208.320.29.45912</td>
<td>S</td>
<td>143851464</td>
<td>143851464</td>
<td>16384</td>
<td>1460</td>
<td>DF</td>
<td>0x10</td>
</tr>
<tr>
<td>982120155.567089</td>
<td>192.168.2.208.320.29.45912</td>
<td>192.168.1.100.1279</td>
<td>A</td>
<td>143851465</td>
<td>143851465</td>
<td>32120</td>
<td>1460</td>
<td>DF</td>
<td>0x10</td>
</tr>
<tr>
<td>982120156.248161</td>
<td>212.208.230.45912</td>
<td>192.168.1.100.1279</td>
<td>A</td>
<td>143851465</td>
<td>143851465</td>
<td>32120</td>
<td>1460</td>
<td>DF</td>
<td>0x10</td>
</tr>
<tr>
<td>982120156.248161</td>
<td>212.208.230.45912</td>
<td>192.168.1.100.1279</td>
<td>A</td>
<td>143851465</td>
<td>143851465</td>
<td>32120</td>
<td>1460</td>
<td>DF</td>
<td>0x10</td>
</tr>
<tr>
<td>982120156.248161</td>
<td>212.208.230.45912</td>
<td>192.168.1.100.1279</td>
<td>A</td>
<td>143851465</td>
<td>143851465</td>
<td>32120</td>
<td>1460</td>
<td>DF</td>
<td>0x10</td>
</tr>
<tr>
<td>982120156.248161</td>
<td>212.208.230.45912</td>
<td>192.168.1.100.1279</td>
<td>A</td>
<td>143851465</td>
<td>143851465</td>
<td>32120</td>
<td>1460</td>
<td>DF</td>
<td>0x10</td>
</tr>
<tr>
<td>982120156.248161</td>
<td>212.208.230.45912</td>
<td>192.168.1.100.1279</td>
<td>A</td>
<td>143851465</td>
<td>143851465</td>
<td>32120</td>
<td>1460</td>
<td>DF</td>
<td>0x10</td>
</tr>
<tr>
<td>982120156.248161</td>
<td>212.208.230.45912</td>
<td>192.168.1.100.1279</td>
<td>A</td>
<td>143851465</td>
<td>143851465</td>
<td>32120</td>
<td>1460</td>
<td>DF</td>
<td>0x10</td>
</tr>
</tbody>
</table>

Based on notes by Jim Martin
Network Performance Characteristics

- Bandwidth-delay product
  - The product of the bw * delay measures the volume of data that can be in the network.
  - The capacity of a point-to-point link that directly connects two hosts is 1.5Mbps and the round trip time is 60ms. How much data can fit in the ‘pipe’?

• Remember this from Chapter 27....
• Comer calls this the Delay-Throughput but does not define delay (i.e., one-way or RTT).
• The best source of information on TCP is Richard Steven’s "TCP/IP Illustrated, Volume 1".

• There’s really two meanings of Bandwidth-delay product:
  - First, a path property
  - Second, a windowed protocol property

Based on notes by Jim Martin

Path level: BW*Delay = 1.5EXP6 * .03/ (8 * 1500) = 3.75 packets
Windowed protocol level: If the max window is set to 4 packets, the link will only be 50% utilized (if we are the only sender)
So BW*Delay is used to define how big the window should be
BW/RTT = just under 8 packets. If the rx advertised window is set for 8 packets, the link will be 100% utilized (i.e., always busy!!!).