TCP
Transmission Control Protocol
Simple Demultiplexor (UDP)

- Unreliable and unordered datagram service
- Adds multiplexing
- No flow control
- Endpoints identified by ports
  - servers have *well-known* ports
  - see `/etc/services` on Unix
- Header format

- Optional checksum
  - psuedo header + UDP header + data
TCP Overview

- **Connection-oriented**
- **Byte-stream**
  - app writes bytes
  - TCP sends segments
  - app reads bytes

- **Full duplex**
- **Flow control**: keep sender from overrunning receiver
- **Congestion control**: keep sender from overrunning network
TCP Header

- **Flags**: SYN, FIN, RESET, PUSH, URG, ACK
- **Checksum**: IP pseudo header + TCP header + data
TCP Overview

- When a client requests a connection, it sends a “SYN” segment (a special TCP segment) to the server port.
- SYN stands for synchronize. The SYN message includes the client’s ISN.
- ISN is Initial Sequence Number.
TCP Overview (Contd.)

- Every TCP segment includes a **Sequence Number** that refers to the first byte of data included in the segment.

- Every TCP segment includes **Acknowledgement Number** that indicates the byte number of the next data that is expected to be received.
  - All bytes up through this number have already been received.
TCP Overview (Contd.)

- **MSS**: Maximum segment size (A TCP option)

- **Window**: Every ACK includes a Window field that tells the sender how many bytes it can send before the receiver will have to toss it away (due to fixed buffer size).
Three-Way Handshake

**HOST A**

Send SYN (seq = x)

Receive SYN (seq = y, ACK = x + 1)

Send ACK (ack = y + 1)

**HOST B**

Receive SYN (seq = x)

Send SYN (seq = y, ACK = x + 1)

Receive ACK (ack = y + 1)
TCP Connection Establishment

Step 1: Client Starts

- A client starts by sending a SYN segment with the following information:
  - Client’s ISN (generated pseudo-randomly)
  - Maximum Receive Window for client.
  - Optionally (but usually) MSS (largest datagram accepted).
  - No payload! (Only TCP headers)
TCP Connection Establishment
Step 2: Sever Response

- When a waiting server sees a new connection request, the server sends back a SYN segment with:
  - Server’s ISN (generated pseudo-randomly)
  - Request Number is Client ISN+1
  - Maximum Receive Window for server.
  - Optionally (but usually) MSS
  - No payload! (Only TCP headers)
TCP Connection Establishment

Step 3:

- When the Server’s SYN is received, the client sends back an ACK with:
  - Request Number is Server’s ISN+1
TCP Data Transfer

- Once the connection is established, data can be sent.
- Each data segment includes a sequence number identifying the first byte in the segment.
- Each segment (data or empty) includes an acknowledgement Number indicating what data has been received.
Buffering

- Keep in mind that TCP is part of the Operating System. It takes care of all these details.
- The TCP layer doesn’t know when the application will ask for any received data.
- TCP buffers incoming data so it’s ready when we ask for it.
TCP Buffers

- Both the client and server allocate buffers to hold incoming and outgoing data.
  - The TCP layer does this.
- Both the client and server announce how much buffer space remains (the Window field in a TCP segment).
Send Buffers

- The application gives the TCP layer some data to send.
- The data is put in a send buffer, where it stays until the data is ACK’d.
  - it has to stay, as it might need to be sent again!
- The TCP layer won’t accept data from the application unless (or until) there is buffer space.
ACKs

- A receiver doesn’t have to ACK every segment (it can ACK many segments with a single ACK segment).
- Each ACK can also contain outgoing data (piggybacking).
- If a sender doesn’t get an ACK after some time limit (MSL) it resends the data.
TCP Segment Order

- Most TCP implementations will accept out-of-order segments (if there is room in the buffer).
- Once the missing segments arrive, a single ACK can be sent for the whole thing.
- Remember: IP delivers TCP segments, and IP in not reliable - IP datagrams can be lost or arrive out of order.
TCP Connection Termination

- The TCP layer can send a RST segment that terminates a connection if something is wrong.
- Usually the application tells TCP to terminate the connection politely with a FIN segment.
FIN

- Either end of the connection can initiate termination.
- A FIN is sent, which means the application is done sending data.
- The FIN is ACK’d.
- The other end must now send a FIN.
- That FIN must be ACK’d.
App1 → App2

1. FIN, SN=X
2. ACK=X+1
3. FIN, SN=Y
4. ACK=Y+1
TCP TIME_WAIT

- Once a TCP connection has been terminated (the last ACK sent) there is some unfinished business:
  - What if the ACK is lost? The last FIN will be resent and it must be ACK’d.
  - What if there are lost or duplicated segments that finally reach the destination after a long delay?

- TCP hangs out for a while \((2 \times \text{Max. Segment Life})\) to handle these situations.
Checking TCP states with netstat

$ netstat -a -n

Active Connections

<table>
<thead>
<tr>
<th>Proto</th>
<th>Local Address</th>
<th>Foreign Address</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>0.0.0.0:7</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:9</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:13</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:17</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:19</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:21</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:23</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:25</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:80</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:135</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:443</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:445</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>127.0.0.1:1030</td>
<td>127.0.0.1:7161</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>127.0.0.1:1051</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>127.0.0.1:7161</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>127.0.0.1:7161</td>
<td>127.0.0.1:1030</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>141.218.143.76:139</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
</tr>
<tr>
<td>TCP</td>
<td>141.218.143.76:1836</td>
<td>141.218.143.43:445</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>141.218.143.76:2003</td>
<td>66.250.84.31:80</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>141.218.143.76:2136</td>
<td>141.218.143.215:22</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>141.218.143.76:3355</td>
<td>216.155.193.166:5050</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>141.218.143.76:3844</td>
<td>141.218.143.10:143</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>141.218.143.76:4635</td>
<td>141.218.143.46:80</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>141.218.143.76:4683</td>
<td>141.218.143.10:143</td>
<td>ESTABLISHED</td>
</tr>
</tbody>
</table>
References

- Cisco Networking Academy Program (CCNA), Cisco Press.
- CSCI-5273 : Computer Networks, Dirk Grunwald, University of Colorado-Boulder
- TCP/IP Illustrated, Volume 1, Stevens.
- Java Network Programming and Distributed Computing, Reilly & Reilly.
- http://www.firewall.cx
- http://www.javasoft.com