TCP

- TCP provides the end-to-end reliable connection that IP alone cannot support.

The TCP protocol
- Segment format
- Connection Creation
- Flow control
- Congestion control
- Connection termination

TCP Segment Format

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

And more...
- 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).

And more...
- There are a bunch of control flags:
  - URG: urgent data included.
  - ACK: this segment is (among other things) an acknowledgement.
  - RST: error - abort the session.
  - SYN: Used to establish connection; synchronize Sequence Numbers (setup)
  - FIN: polite connection termination.
TCP Lingo

- 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 Connection Establishment - Three-way handshake

1. **Client**
   - SYN
   - ISN = X
   - “I want to talk, and I’m starting with byte number X+1.”

2. **Server**
   - SYN
   - ISN = Y
   - ACK = X + 1
   - “OK, I’m here and I’ll talk. My first byte will be called number Y+1, and I know your first byte will be number X+1.”

3. **Client**
   - ACK
   - Y + 1
   - “Got it - you start at byte number Y+1.”

TCP Connection Creation

- A server accepts a connection.
  - Must be looking for new connections!

- A client requests a connection.
  - Must know where the server is!

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)

Server 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)

Finally

- When the Server’s SYN is received, the client sends back an ACK with:
  - Request Number is Server’s ISN+1
Why 3-Way?
- Why is the third message necessary?

HINTS:
- TCP is a reliable service.
- IP delivers each TCP segment.
- IP is not reliable.

TCP Data and ACK
- Once the connection is established, data can be sent.
- Each data segment includes a sequence number identifying the first byte in the segment.
- Each ACK segment includes a request number indicating what data has been received. (bytes instead of packets)

Important Information in TCP/IP packet headers

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.

Recv Buffers
- The received data is put in the recv buffer, where it stays until the application reads data
  - The Recv Buffer won’t accept data from network unless (or until) there is buffer space, and the sequence number is within its sliding windows.
  - The recv buffer can store out of order data as long as there is buffer space and the sequence number is within its sliding windows.
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 Flow Control

- The sender and receiver are shown in relation to each other.
- The receiver's buffer is shown as empty and full.
- The sender can send up to 2K segments.
- The receiver can receive 2K segments and ACK 4K segments.
- The sender may send up to 2K segments with a single ACK segment.

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.

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.

TCP Termination

- The diagram shows the sequence of events in the termination process.
- App1 sends a FIN segment.
- App2 sends an ACK segment.
- The other end sends a FIN segment.
- Both ends release the connection.
TCP Termination

App1

---

1. FIN
   SN=X

---

2. FIN
   SN= confronted
   ACK=X+1

---

3. FIN
   SN= Y

---

4. FIN
   ACK=Y+1

---

Closed

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 to handle these situations.

Test Questions

- Why is a 3-way handshake necessary?
- Who sends the first FIN - the server or the client?
- Once the connection is established, what is the difference between the operation of the server's TCP layer and the client's TCP layer?
- What happens if a bad guy can guess ISNs?

Assignment & Next time

- Reading:
  - UNP 2.4, 2.6, 2.7 **
- Next Lecture:
  - TCP Socket Programming