TCP - Part I

Relates to Lab 5. First module on TCP which covers packet format, data transfer, and connection management.

Overview

TCP = Transmission Control Protocol
- Connection-oriented protocol
- Provides a reliable unicast end-to-end byte stream over an unreliable internetwork.
Connection-Oriented

- Before any data transfer, TCP establishes a **connection**:
  - One TCP entity is waiting for a connection ("server")
  - The other TCP entity ("client") contacts the server
- The actual procedure for setting up connections is more complex.
- Each connection is full duplex

![Connection Diagram]

Reliable

- Byte stream is broken up into chunks which are called **segments**
  - Receiver sends acknowledgements (ACKs) for segments
  - TCP maintains a timer. If an ACK is not received in time, the segment is retransmitted

**Detecting errors:**
- TCP has checksums for header and data. Segments with invalid checksums are discarded
- Each byte that is transmitted has a sequence number
Byte Stream Service

- To the lower layers, TCP handles data in blocks, the segments.
- To the higher layers TCP handles data as a sequence of bytes and does not identify boundaries between bytes.
- So: Higher layers do not know about the beginning and end of segments!

TCP Format

- TCP segments have a 20 byte header with \( \geq 0 \) bytes of data.
TCP header fields

- **Port Number:**
  - A port number identifies the endpoint of a connection.
  - A pair `<IP address, port number>` identifies one endpoint of a connection.
  - Two pairs `<client IP address, server port number>` and `<server IP address, server port number>` identify a TCP connection.

```
TCP
IP
```

```
TCP
IP
```

**Q:** What are possible requirements for ISN?
TCP header fields

- **Acknowledgement Number (AckNo):**
  - Acknowledgements are piggybacked, i.e.
    a segment from A -> B can contain an
    acknowledgement for a data sent in the B -> A direction

  Q: *Why is piggybacking good?*

  - A host uses the AckNo field to send acknowledgments.
    (If a host sends an AckNo in a segment it sets the "ACK flag")
  - The AckNo contains the next SeqNo that a host wants to receive
    Example: The acknowledgement for a segment with sequence numbers 0-1500 is AckNo=1501

TCP header fields

- **Acknowledgement Number (cont’d)**
  - TCP uses the sliding window flow protocol (see CS 457) to regulate the flow of traffic from sender to receiver
  - TCP uses the following variation of sliding window:
    - no NACKs (Negative ACKnowledgement)
    - only cumulative ACKs

- Example:
  **Assume:** Sender sends two segments with “1..1500” and “1501..3000”, but receiver only gets the second segment.
  **In this case,** the receiver cannot acknowledge the second packet. It can only send AckNo=1
TCP header fields

• Header Length (4bits):
  – Length of header in 32-bit words
  – Note that TCP header has variable length (with minimum 20 bytes)

TCP header fields

• Flag bits:
  – URG: Urgent pointer is valid
    – If the bit is set, the following bytes contain an urgent message in the range:
      \[ \text{SeqNo} \leq \text{urgent message} \leq \text{SeqNo} + \text{urgent pointer} \]
  – ACK: Acknowledgement Number is valid
  – PSH: PUSH Flag
    – Notification from sender to the receiver that the receiver should pass all data that it has to the application.
    – Normally set by sender when the sender’s buffer is empty
## TCP header fields

<table>
<thead>
<tr>
<th>Flag bits:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RST</strong>: Reset the connection</td>
</tr>
<tr>
<td>The flag causes the receiver to reset the connection</td>
</tr>
<tr>
<td>Receiver of a RST terminates the connection and indicates higher layer application about the reset</td>
</tr>
<tr>
<td><strong>SYN</strong>: Synchronize sequence numbers</td>
</tr>
<tr>
<td>Sent in the first packet when initiating a connection</td>
</tr>
<tr>
<td><strong>FIN</strong>: Sender is finished with sending</td>
</tr>
<tr>
<td>Used for closing a connection</td>
</tr>
<tr>
<td>Both sides of a connection must send a FIN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Window Size:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each side of the connection advertises the window size</td>
</tr>
<tr>
<td>Window size is the maximum number of bytes that a receiver can accept.</td>
</tr>
<tr>
<td>Maximum window size is $2^{16} - 1 = 65535$ bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCP Checksum:</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP checksum covers over both TCP header and TCP data (also covers some parts of the IP header)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Urgent Pointer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only valid if URG flag is set</td>
</tr>
</tbody>
</table>
TCP header fields

- **Options:**
  - **End of Options**: kind=0, 1 byte
  - **NOP (no operation)**: kind=1, 1 byte
  - **Maximum Segment Size**: kind=2, len=4, maximum segment size
  - **Window Scale Factor**: kind=3, len=3, shift count
  - **Timestamp**: kind=8, len=10, timestamp value, timestamp echo reply

**TCP header fields**

- **Options:**
  - **NOP** is used to pad TCP header to multiples of 4 bytes
  - **Maximum Segment Size**
  - **Window Scale Options**
    - Increases the TCP window from 16 to 32 bits, i.e., the window size is interpreted differently
    - Q: What is the different interpretation?
    - This option can only be used in the SYN segment (first segment) during connection establishment time
  - **Timestamp Option**
    - Can be used for roundtrip measurements
Connection Management in TCP

- Opening a TCP Connection
- Closing a TCP Connection
- Special Scenarios
- State Diagram

TCP Connection Establishment

- TCP uses a **three-way handshake** to open a connection:
  1. **ACTIVE OPEN**: Client sends a segment with
     - SYN bit set *
     - port number of client
     - initial sequence number (ISN) of client
  2. **PASSIVE OPEN**: Server responds with a segment with
     - SYN bit set *
     - initial sequence number of server
     - ACK for ISN of client
  3. **Client acknowledges by sending a segment with**:
     - ACK ISN of server (* counts as one byte)
Three-Way Handshake

A Closer Look with tcpdump

aida issues an "telnet mng"

1 aida.poly.edu.1121 > mng.poly.edu.telnet: S 1031880193:1031880193(0) win 16384 <mss 1460,nop,wscale 0,nop,nop,timestamp>
2 mng.poly.edu.telnet > aida.poly.edu.1121: S 172488586:172488586(0) ack 1031880194 win 8760 <mss 1460>
3 aida.poly.edu.1121 > mng.poly.edu.telnet: . ack 172488587 win 17520
4 aida.poly.edu.1121 > mng.poly.edu.telnet: P 1031880194:1031880218(24) ack 172488587 win 17520
5 mng.poly.edu.telnet > aida.poly.edu.1121: P 172488587:172488590(3) ack 1031880218 win 8736
6 aida.poly.edu.1121 > mng.poly.edu.telnet: P 1031880218:1031880221(3) ack 172488590 win 17520
Three-Way Handshake

Why is a Two-Way Handshake not enough?

When aida initiates the data transfer (starting with SeqNo=15322112355), mng will reject all data.
TCP Connection Termination

- Each end of the data flow must be shut down independently ("half-close")
- If one end is done it sends a FIN segment. This means that no more data will be sent.

- Four steps involved:
  (1) X sends a FIN to Y (active close)
  (2) Y ACKs the FIN,
      (at this time: Y can still send data to X)
  (3) and Y sends a FIN to X (passive close)
  (4) X ACKs the FIN.

Connection termination with tcpdump

aida issues an "telnet mng"

aida.poly.edu  mng.poly.edu

1  mng.poly.edu.telnet > aida.poly.edu.1121: F 172488734:172488734(0) ack 1031880221 win 8733
2  aida.poly.edu.1121 > mng.poly.edu.telnet: . ack 172488735 win 17484
3  aida.poly.edu.1121 > mng.poly.edu.telnet: F 1031880221:1031880221(0) ack 172488735 win 17520
4  mng.poly.edu.telnet > aida.poly.edu.1121: . ack 1031880222 win 8733
TCP Connection Termination

TCP States

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSED</td>
<td>No connection is active or pending</td>
</tr>
<tr>
<td>LISTEN</td>
<td>The server is waiting for an incoming call</td>
</tr>
<tr>
<td>SYN RCVD</td>
<td>A connection request has arrived; wait for Ack</td>
</tr>
<tr>
<td>SYN SENT</td>
<td>The client has started to open a connection</td>
</tr>
<tr>
<td>ESTABLISHED</td>
<td>Normal data transfer state</td>
</tr>
<tr>
<td>FIN WAIT 1</td>
<td>Client has said it is finished</td>
</tr>
<tr>
<td>FIN WAIT 2</td>
<td>Server has agreed to release</td>
</tr>
<tr>
<td>TIMED WAIT</td>
<td>Wait for pending packets (“2MSL wait state”)</td>
</tr>
<tr>
<td>CLOSING</td>
<td>Both Sides have tried to close simultaneously</td>
</tr>
<tr>
<td>CLOSE WAIT</td>
<td>Server has initiated a release</td>
</tr>
<tr>
<td>LAST ACK</td>
<td>Wait for pending packets</td>
</tr>
</tbody>
</table>
TCP States in “Normal” Connection Lifetime

TCP State Transition Diagram
Opening A Connection
TCP State Transition Diagram
Closing A Connection

2MSL Wait State

2MSL Wait State = TIME_WAIT
- When TCP does an active close, and sends the final ACK, the connection must stay in the TIME_WAIT state for twice the maximum segment lifetime.

2MSL= 2 * Maximum Segment Lifetime

- Why?
  TCP is given a chance to resent the final ACK. (Server will timeout after sending the FIN segment and resend the FIN)
  - The MSL is set to 2 minutes or 1 minute or 30 seconds.
Resetting Connections

- Resetting connections is done by setting the RST flag
- **When is the RST flag set?**
  - Connection request arrives and no server process is waiting on the destination port
  - Abort (Terminate) a connection
    Causes the receiver to throw away buffered data. Receiver does not acknowledge the RST segment