TCP Protocol Details, Part 1

CIS-3152, Spring 2013
Peter C. Chapin
Frame Structure

Frame → Packet → Segment

- Ethernet Header (Data Link): 20 bytes (nominal)
- IP Header (Network): 20 bytes (nominal)
- TCP Header (Transport): 20 bytes (nominal)
- Actual Data
- CRC Checksum
TCP Header Notes

- TCP described in RFC-793 (with updates)
- Note...
  - Source/Destination addresses in IP header.
  - Every byte has a sequence number.
    - Seq # gives number for first byte in segment.
    - Ack # gives number for next byte expected.
  - Header length ("data offset") in units of 32 bits.
  - Window size: We will discuss later.
  - Checksum made over "pseudo header" and data.
  - Options typically only occur on initial segments.
TCP Header Flags

• Several flag bits are defined...
  • **URG**: The value of the Urgent Pointer is valid.
  • **ACK**: The value of the Ack # is valid.
  • **PSH**: The data should be “pushed” to the receiver.
  • **RST**: Reset (end) the TCP connection abruptly.
  • **SYN**: Synchronize (initiate the connection).
  • **FIN**: Close the connection cleanly.
MSS Option

- Most common option is “Maximum Segment Size”
  - Discussed at length in RFC-879
  - Used when connection established. Can only appear in a segment with the SYN flag.
  - Can be different in the different directions.
  - Default 536 bytes (data)
    - Overall packet size 576 bytes (data+TCP+IP).
  - Bigger is better (reduces overhead)
- Ethernet MSS usually 1460 bytes
  - Ethernet frame payload 1500 bytes.
Sequence Numbers

<table>
<thead>
<tr>
<th>H</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>o</th>
<th>\n</th>
</tr>
</thead>
<tbody>
<tr>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
</tr>
</tbody>
</table>

- Every byte has its own number!
- Sequence numbers independent in the two directions.
- Each segment specifies the sequence number of its first data byte and acknowledges the next sequence number expected from the other side.
- Segment boundaries are arbitrary.
Stream Oriented

• TCP is a *stream oriented* protocol
  • Data is **not** broken into records, but instead treated as a continuous stream.
  • TCP breaks data into segments **arbitrarily**.
  • Applications unaware of segment boundaries.
  • A single call to *write* might...
    – Generate multiple segments
    – Generate only part of a segment
  • A single call to *read* might...
    – Obtain data from multiple segments.
    – Obtain data from only part of a segment
TCP Buffers

Application

write()

Send Buffer

OS

TCP Connection (Bidirectional pipe)

read()
TCP Buffers Notes

- **write** normally returns at once.
  - Even before the data has been sent!
- If data arrives it is buffered.
- If receive buffer non-empty, **read** returns at once.
  - Even if data size less than requested amount.
- If receive buffer empty, **read** blocks.
- When connection closed, buffers drain normally.
- Application can terminate before TCP is done!
Establishing TCP Connection

- **“Three way handshake”:** SYN, SYN/ACK, ACK
- **Initial sequence numbers (ISNs) independent and arbitrary**
- **Connection ESTABLISHED once complete.**
Close TCP Connection

**ACTIVE CLOSE**

- close()
- FIN segment (may or may not contain data)
- ACK of FIN + 1

**PASSIVE CLOSE**

- read returns EOF
- Still possible to write.
- close()

- FIN segment
- ACK of FIN + 1

**Middle two segments might get combined**

- For example: If application closes very quickly after read returns EOF.
TCP State Diagram, Part 1

CLOSED

LISTEN

app: calls listen
recv: SYN
send: SYN, ACK

SYN_RCVD

app: calls connect
send: SYN

SYN_SENT

recv: SYN, ACK
send: ACK

ESTABLISHED

recv: ACK
TCP State Diagram, Part 2

app: calls close
send: FIN

recv: FIN
send: ACK

recv: ACK

recv: FIN
send: ACK

recv: FIN, ACK
send: ACK

recv: FIN
send: ACK

recv: FIN, ACK
send: ACK

recv: ACK

Done!

(Wait for 2MSL seconds)
TCP State Diagram, Part 3

- **ESTABLISHED**
  - recv: FIN
  - send: ACK

- **CLOSE_WAIT**
  - app: calls close
  - send: FIN

- **LAST_ACK**
  - recv: ACK

Done!
Tools

- On Unix use `netstat` to view connection state
  - `netstat -a` shows “all” connections (including listening sockets).
  - `netstat -A inet6` shows connections in the “inet6” address family (TCP running on IPv6).
  - See man page for more details.

- On Windows TCPView is a GUI netstat tool
  - [http://www.sysinternals.com/](http://www.sysinternals.com/)
Stop-And-Wait

• Simple protocol for transferring data.
  ● Send one segment.
  ● Wait for acknowledgement.

• Easy to implement, but has disadvantages:
  ● Only one segment on the network.
    – Inefficient use of network bandwidth.
  ● Sender must wait for $2 \times T_T$ (where $T_T$ is the transit time across the network).
    – Could be many milliseconds... or even seconds!
    – Lots of waiting; slow data transfer.
Transit Time

- Finite speed of light ($2.998 \times 10^8$ m/s)
  - Time is required to move bits (on the order of 50 ms to go 10,000 miles).
  - Speed on cables is actually less.
    - "Velocity factor" on typical cables might be 0.80.
    - Due to dielectric material used as insulation and cable geometry.
- Also router delays.
  - This is usually the biggest factor.
Router Delay

Packets wait in buffers
Waiting time can be considerable

Slow or congested
Stop-And-Wait Computation

- Assume...
  - Transit time = 50 ms (one way) or 100 ms round.
  - Each packet contains 1000 bytes.
  - Transfer rate = 1000 bytes/0.1s = \textbf{10,000 bytes/s}.
- Stop-And-Wait must wait for the ACK
  - Spends most of its time waiting.
- Transfer rate is independent of bandwidth!
  - Calculation above is valid as long as bandwidth is sufficient.
Latency vs Bandwidth

- Be aware that “latency” is different than “bandwidth.”
  - **Latency:**
    - How long does it take for the first bit transmitted to reach the destination?
    - Delays due to the speed of light and router buffering, etc.
  - **Bandwidth:**
    - How many bits/s can be transmitted?
- High latency, high bandwidth connections...
  - “Long fat pipes.”
• Data is sent to fill a “window”
  • TCP speculatively sends without ACKs
  • Fills the network with data
    – Much more efficient than stop-and-wait
• By the time the window is transmitted, the first ACKs arrive (we hope).
• Each ACK moves the window forward.
TCP Window

Now: segment arrives: ACK# 150; window 100
TCP Window

- ACK: “I've received everything below the ACK number.”
- Window moves as ACK advances.
  - Exposed data can now be sent until window is filled again.
  - Try to keep window-size bytes of data in flight at all times.
TCP Window

Now: segment arrives: ACK# 150, window = 200

Window size can change!
**TCP Window**

- Receiver modulates window size to reflect receive buffer size.
  - If the receiver has only a small buffer: small window.
  - As receiver consumes data, buffer empties. Window opens.
- Sender never sends more than receiver can handle!
TCP Window

Now: segment arrives: ACK# 150, window = 50

Zero size means RX buffer full
TCP Window

- Data ACKed
- Seq #100
- Data not yet sent
- Seq #200

Window doesn't expose new data Can't send.
TCP Window

Now: segment arrives: ACK #150, window = 100

Segment ACKs nothing new. Just used for window size update.
TCP Window

Data ACKed

Seq #100

Data not yet sent

Seq #200

TCP can begin sending
Example

Seq# 0x12345678
Win: 16384

SYN

Seq# 0x12345678
Ack# 0x9ABCDEF1
Win: 16384

ACK

Seq# 0x12345679
Ack# 0x9ABCDEF1
Win: 16384

1024 bytes data

Seq# 0x12345679
Ack# 0x9ABCDEF1
Win: 16384

1024 bytes data

Seq# 0x12345879
Ack# 0x9ABCDEF1
Win: 16384

0 bytes data

ETC...

Seq# 0x9ABCDEF0
Ack# 0x12345A79
Win: 30720

Win: 32768