Transport Protocols: TCP/UDP Structure

→ end-to-end mechanism

→ runs on top of link-based mechanism

→ treat network layer as black box

Three-level encapsulation:
Network layer assumptions:

- unreliable
- out-of-order delivery (in general)
- absence of QoS guarantees (delay, throughput etc.)
- insecure (IPv4)

Additional (informal) performance properties:

- works “fine” under low load conditions
- can break down under high load conditions
- behavior range predictable (to certain extent)
Goal of UDP: Process identification ("multiplexing").

\[\rightarrow \text{ port number as process demux key}\]

- form of end host processing (O.S.)
- generally: end system support (e.g., scheduling)
UDP packet format:

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Checksum</td>
</tr>
<tr>
<td></td>
<td>Payload</td>
</tr>
</tbody>
</table>

Checksum calculation (pseudo header):

<table>
<thead>
<tr>
<th>Source Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination Address</td>
</tr>
<tr>
<td>00 · · · 0</td>
</tr>
</tbody>
</table>
Goals of TCP:

- process identification
- reliable communication (ARQ)
- speedy communication (congestion/flow control)
- segmentation

→ connection-oriented (i.e., stateful)

→ complex mixture of functionalities
Segmentation task: Provide “stream” interface to higher level protocols

→ view: contiguous stream of bytes

• segment stream of bytes into blocks or segments of fixed size

• segment size determined by TCP MTU (Maximum Transmission Unit)

• use also for reliability mechanism
TCP packet format:

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acknowledgement Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Window Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Header Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options (if any)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATA (if any)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
• Sequence Number: position of first byte of payload
• Acknowledgement: next byte of data expected (receiver)
• Header Length (4 bits): 4 B units
• URG: urgent pointer flag
• ACK: ACK packet flag
• PSH: override TCP buffering
• RST: reset connection
• SYN: establish connection
• FIN: close connection
• Window Size: receiver’s advertised window size
• Checksum: prepend pseudo-header
• Urgent Pointer: byte offset in current payload where urgent data begins
• Options: MTU; take min of sender & receiver (default 556 B)
Checksum calculation (pseudo header):

4

<table>
<thead>
<tr>
<th>Source Address</th>
<th>Destination Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 · · · 0</td>
<td>Protocol</td>
</tr>
<tr>
<td></td>
<td>UDP Length</td>
</tr>
</tbody>
</table>
Nagle’s algorithm:

- do not want to send too many 1 B payload packets
- rule: connection can have only one such unacknowledged packet outstanding
- while waiting for ACK, incoming bytes are accumulated (i.e., buffered)

... compromise between real-time constraints and efficiency.

→ useful for telnet-type applications
TCP connection establishment (3-way handshake):

- $X, Y$ are chosen randomly
- piggybacking
- sequence number prediction
- lingering packet problem
2-person consensus problem: Are $A$ and $B$ in agreement about the state of affairs after 3-way handshake?

$\rightarrow$ impossibility, in general

$\rightarrow$ lunch date problem
Call Collision:

→ only single TCB gets allocated

→ unique full association
TCP connection termination:

- full duplex
- half duplex
More generally, finite state machine representation of TCP’s control mechanism:

TCP’s State-transition Diagram comes here
Features to notice:

- **Connection set-up:**
  - client’s transition to **ESTABLISHED** state without ACK
  - how is server to reach **ESTABLISHED** if client ACK is lost?
  - TCP: default ACKing executed by all data packets; no extra overhead incurred
  - note: **ESTABLISHED** is macrostate
  - not a complete transition diagram

- **Connection tear-down:**
  - three normal cases
  - special issue with **TIME WAIT** state
Basic TCP data transfer:
TCP’s sliding window protocol

Sender:

Receiver:

- sender, receiver maintain buffers \texttt{MaxSendBuffer}, \texttt{MaxRcvBuffer}
Note asynchrony between TCP module and application.

Sender side: maintain invariants

- \( \text{LastByteAcked} \leq \text{LastByteSent} \leq \text{LastByteWritten} \)
- \( \text{LastByteWritten} - \text{LastByteAcked} < \text{MaxSendBuffer} \)
  - \( \rightarrow \) buffer flushing (advance window)
  - \( \rightarrow \) application blocking

- \( \text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow} \)

Thus,

\[
\text{EffectiveWindow} = \text{AdvertisedWindow} - (\text{LastByteSent} - \text{LastByteAcked})
\]

- \( \rightarrow \) upper bound on new send volume
Receiver side: maintain invariants

- $\text{LastByteRead} < \text{NextByteExpected} \leq \text{LastByteRcvd} + 1$
- $\text{LastByteRcvd} - \text{NextByteRead} < \text{MaxRcvBuffer}$

$\rightarrow$ buffer flushing (advance window)
$\rightarrow$ application blocking

Thus,

$$\text{AdvertisedWindow} = \text{MaxRcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead})$$
Three problems:

How to let sender know of changed in receiver window size after \texttt{AdvertisedWindow} becomes 0?

- trigger ACK event on receiver side when \texttt{AdvertisedWindow} becomes positive
- sender periodically sends 1-byte probing packet

\[\rightarrow \text{ design choice: smart sender/dumb receiver}\]

Silly window syndrome: Assuming receiver buffer is full, what if application reads one byte at a time with long pauses?

- can cause excessive 1-byte traffic
- if \texttt{AdvertisedWindow} \(<\) MSS then set \texttt{AdvertisedWindow} \(\leftarrow 0\)
Sequence number wrap-around problem: recall sufficient condition

$$\text{SenderWindowSize} < (\text{MaxSeqNum} + 1)/2$$

→ 32-bit sequence space/16-bit window space

However, more importantly, time until wrap-around important due to possibility of roaming packets.

<table>
<thead>
<tr>
<th>bandwidth</th>
<th>time until wrap-around †</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5 Mbps)</td>
<td>6.4 hrs</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>57 min</td>
</tr>
<tr>
<td>T3 (45 Mbps)</td>
<td>13 min</td>
</tr>
<tr>
<td>FDDI (100 Mbps)</td>
<td>6 min</td>
</tr>
<tr>
<td>OC-3 (155 Mbps)</td>
<td>4 min</td>
</tr>
<tr>
<td>OC-12 (622 Mbps)</td>
<td>55 sec</td>
</tr>
<tr>
<td>OC-24 (1.2 Gbps)</td>
<td>28 sec</td>
</tr>
</tbody>
</table>

† From P & D for 32-bit sequence space
Even more importantly, “keeping-the-pipe-full” consideration.

<table>
<thead>
<tr>
<th>bandwidth</th>
<th>delay-bandwidth product †</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5 Mbps)</td>
<td>18 kB</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>122 kB</td>
</tr>
<tr>
<td>T3 (45 Mbps)</td>
<td>549 kB</td>
</tr>
<tr>
<td>FDDI (100 Mbps)</td>
<td>1.2 MB</td>
</tr>
<tr>
<td>OC-3 (155 Mbps)</td>
<td>1.8 MB</td>
</tr>
<tr>
<td>OC-12 (622 Mbps)</td>
<td>7.4 MB</td>
</tr>
<tr>
<td>OC-24 (1.2 Gbps)</td>
<td>14.8 MB</td>
</tr>
</tbody>
</table>

† From P & D for 100 ms latency
RTT estimation

... important to not underestimate nor overestimate.

Karn/Partridge: Maintain running average with precautions

\[ \text{EstimateRTT} \leftarrow \alpha \cdot \text{EstimateRTT} + \beta \cdot \text{SampleRTT} \]

- \text{SampleRTT} computed by sender using timer
- \( \alpha + \beta = 1; \ 0.8 \leq \alpha \leq 0.9, \ 0.1 \leq \beta \leq 0.2 \)
- \text{TimeOut} \leftarrow 2 \cdot \text{EstimateRTT} \text{ or } \text{TimeOut} \leftarrow 2 \cdot \text{TimeOut} \ (\text{if retransmit})

\[ \rightarrow \text{ need to be careful when taking SampleRTT} \]
\[ \rightarrow \text{ infusion of complexity} \]
\[ \rightarrow \text{ still remaining problems} \]
Hypothetical RTT distribution:

\[ \text{# Samples} \quad \text{RTT} \]

\[ \text{# Samples} \quad \text{RTT} \]

\[ \rightarrow \text{ need to account for variance} \]
Jacobson/Karels:

- \( \text{Difference} = \text{SampleRTT} - \text{EstimatedRTT} \)
- \( \text{EstimatedRTT} = \text{EstimatedRTT} + \delta \cdot \text{Difference} \)
- \( \text{Deviation} = \text{Deviation} + \delta(|\text{Difference}| - \text{Deviation}) \)

Here \( 0 < \delta < 1 \).

Finally,

- \( \text{TimeOut} = \mu \cdot \text{EstimatedRTT} + \phi \cdot \text{Deviation} \)

where \( \mu = 1, \phi = 4 \).

\[\longrightarrow \text{ persistence timer} \]

\[\longrightarrow \text{ how to keep multiple timers in UNIX} \]