Transport Protocols: TCP and UDP

→ end-to-end protocol

→ runs on top of network layer protocols

→ treat network layer & below as black box

Three-level encapsulation:

→ meaning of protocol “stack”: push/pop headers
Network layer (IP) assumptions:

- unreliable
- out-of-order delivery
- absence of QoS guarantees (delay, throughput, etc.)
- insecure (IPv4)
  \[\rightarrow\] IPsec (native in IPv6)

Additional performance properties:

- works “ok”
- can break down under high load conditions
  \[\rightarrow\] flash crowds
  \[\rightarrow\] DoS and worm attack
- wide behavioral range
  \[\rightarrow\] sometimes good, so so, or bad
Goal of UDP (User Datagram Protocol):

\[ \rightarrow \text{ process identification} \]

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

\[ \rightarrow \text{ minimal support beyond IP} \]
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>
</tbody>
</table>

Payload

Checksum calculation: pseudo header

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

| 00 · · · 0 | Protocol | UDP Length |

→ pseudo header, UDP header and payload
UDP usage:

- multimedia streaming
  → lean and nimble
  → at minimum requires process identification
  → reliability addressed above UDP: FEC or ARQ
  → congestion control addressed above UDP
- stateless client/server applications
  → persistent state can be a hinderance
  → lightweight
- implemented in OS
Goals of TCP (Transmission Control Protocol):

• process identification
• reliable communication: ARQ
• speedy communication: congestion control
• segmentation

→ connection-oriented, i.e., stateful
→ complex mixture of functionalities
Segmentation task: provide stream interface to higher level protocols

→ exported semantics: contiguous byte stream
→ e.g., accessed using read(), write() system calls

Segmentation:

• segment stream of bytes into blocks of fixed size
• segment size determined by TCP MTU (Maximum Transmission Unit)
• actual unit of transmission in ARQ
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>Header Length</th>
<th>U</th>
<th>A</th>
<th>P</th>
<th>S</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</td>
<td>C</td>
<td>S</td>
<td>Y</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>H</td>
<td>T</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Checksum</th>
<th>Urgent Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<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 and receiver
Checksum calculation: pseudo header

4

<table>
<thead>
<tr>
<th>Source Address</th>
<th>Destination Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>00  \cdots 0</td>
<td>Protocol</td>
</tr>
<tr>
<td>TCP Segment Length</td>
<td></td>
</tr>
</tbody>
</table>

→ pseudo header, TCP header and payload
TCP connection establishment (3-way handshake):

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

$\rightarrow$ in general: impossible

$\rightarrow$ can be proven

$\rightarrow$ “acknowledging the ACK problem”

$\rightarrow$ also TCP session ending

$\rightarrow$ lunch date problem
TCP connection termination:

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

\[\text{state transition diagram}\]
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?
  – **ESTABLISHED** is macrostate (partial diagram)

• Connection tear-down:
  – three normal cases
  – special issue with **TIME WAIT** state
  – employs “hack”
TCP’s sliding window protocol:

- sender, receiver maintain buffers `MaxSendBuffer`, `MaxRcvBuffer`
Same as generic sliding window

$\rightarrow$ data unit: byte, not packet

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

- $\text{LastByteSent} - \text{LastByteAcked} \leq \text{AdvertisedWindow}$

$\rightarrow$ $\text{AdvertisedWindow}$: receiver side free space

$\rightarrow$ throttling effect
How much sender can still send:

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

\[\rightarrow \text{ upper bound}\]
\[\rightarrow \text{ sender may choose to send less}\]
\[\rightarrow \text{ self-throttling}\]

Affected through sender side variable

\[\rightarrow \text{ CongestionWindow}\]
EffectiveWindow update procedure:

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

where

\[
\text{MaxWindow} = \min\{\text{AdvertisedWindow}, \text{CongestionWindow}\}
\]

How to set \text{CongestionWindow}.

\[\rightarrow\] TCP congestion control
Receiver side: maintain invariants

- \( \text{LastByteRead} < \text{NextByteExpected} \leq \text{LastByteRcvd} + 1 \)

- \( \text{LastByteRcvd} - \text{NextByteRead} < \text{MaxRcvBuffer} \)
  
  \[ \rightarrow \text{buffer flushing (advance window)} \]
  
  \[ \rightarrow \text{application blocking} \]

Thus,

\[
\text{AdvertisedWindow} = \text{MaxRcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead})
\]
Issues:

How to let sender know of change in receiver window size after $\text{AdvertisisedWindow}$ becomes 0?

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

→ design choice: smart sender/dumb receiver
→ same situation for congestion control
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 $\text{AdvertisedWindow} < \text{MSS}$ then set
  \[
  \text{AdvertisedWindow} \leftarrow 0
  \]
Do not want to send too many 1 B payload packets.

Nagle’s method:

- 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 ssh-like interactive applications
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 \times \text{EstimateRTT} \quad \text{or} \quad \\
  \text{TimeOut} \leftarrow 2 \times \text{TimeOut} \quad \text{(if retransmit)}

\[\rightarrow\] need to be careful when taking \text{SampleRTT}
Issue of variance:

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

\[\rightarrow\] need to account for variance

\[\rightarrow\] not nearly as nice
Jacobson/Karels:

- **Difference** = SampleRTT − EstimatedRTT
- **EstimatedRTT** = EstimatedRTT + δ × Difference
- **Deviation** = Deviation + δ × (|Difference| − Deviation)

Here 0 < δ < 1.

Finally,

- **TimeOut** = μ × EstimatedRTT + φ × Deviation

where μ = 1, φ = 4.