UDP header revisited

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

- **Version**
- **IHL**
- **DSCP**
- **ECN**
- **Identification**
- **Flags**
- **Fragment Offset**
- **Time To Live**
- **Protocol**
- **Header Checksum**

**Source IP Address**

**Destination IP Address**

- **Options (if IHL > 5)**

<table>
<thead>
<tr>
<th></th>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Length**
- **Checksum**

**Application data**
TCP
TCP

Consists of 3 primary phases:

• Connection Establishment (Setup)
• Sliding Windows/Flow Control
• Connection Release (Teardown)
Connection Establishment

• Both sender and receiver must be ready before we start the transfer of data
  • Need to agree on a set of parameters
  • e.g., the Maximum Segment Size (MSS)

• This is signaling
  • It sets up state at the endpoints
  • Like “dialing” for a telephone call
Three-Way Handshake

• Used in TCP; opens connection for data in both directions

• Each side probes the other with a fresh Initial Sequence Number (ISN)
  • Sends on a SYNchronize segment
  • Echo on an ACKnowledge segment

• Chosen to be robust even against delayed duplicates
Three-Way Handshake (2)

• Three steps:
  • Client sends SYN(x)
  • Server replies with SYN(y)ACK(x+1)
  • Client replies with ACK(y+1)
  • SYNs are retransmitted if lost

• Sequence and ack numbers carried on further segments
Three-Way Handshake (3)

• Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
  • Improbable, but anyhow ...

Active party (client)

SYN (SEQ=x)

Passive party (server)

(SEQ=x+1, ACK=z+1)
Three-Way Handshake (4)

• Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
  • Improbable, but anyhow …

• Connection will be cleanly rejected on both sides 😊
TCP Connection State Machine

- Captures the states ([]) and transitions (->)
  - A/B means event A triggers the transition, with action B

Both parties run instances of this state machine
TCP Connections (2)

- Follow the path of the client:
TCP Connections (3)

• And the path of the server:
TCP Connections (4)

• Again, with states ...

Active party (client)  Passive party (server)

CLOSED
SYN_SENT
ESTABLISHED

SYN (SEQ=x) 1
SYN (SEQ=y, ACK=x+1) 2
(SEQ=x+1, ACK=y+1) 3
Time

CLOSED
LISTEN
SYN_RCVD
ESTABLISHED

CSE 461 University of Washington
TCP Connections (5)

• Finite state machines are a useful tool to specify and check the handling of all cases that may occur

• TCP allows for simultaneous open
  • i.e., both sides open instead of the client-server pattern
  • Try at home to confirm it works 😊
Connection Release

• Orderly release by both parties when done
  • Delivers all pending data and “hangs up”
  • Cleans up state in sender and receiver

• Key problem is to provide reliability while releasing
  • TCP uses a “symmetric” close in which both sides shutdown independently
TCP Connection Release

- Two steps:
  - Active sends FIN(x), passive ACKs
  - Passive sends FIN(y), active ACKs
  - FINs are retransmitted if lost

- Each FIN/ACK closes one direction of data transfer
TCP Connection Release (2)

• Two steps:
  • Active sends FIN(x), passive ACKs
  • Passive sends FIN(y), active ACKs
  • FINs are retransmitted if lost

• Each FIN/ACK closes one direction of data transfer
TCP Connection State Machine

Both parties run instances of this state machine
TCP Release

• Follow the active party
TCP Release (2)

• Follow the passive party
TCP Release (3)

• Again, with states ...

<table>
<thead>
<tr>
<th>Active party</th>
<th>Passive party</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESTABLISHED</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>FIN_WAIT_1</td>
<td>CLOSE_WAIT</td>
</tr>
<tr>
<td>FIN_WAIT_2</td>
<td>LAST_ACK</td>
</tr>
<tr>
<td>TIME_WAIT</td>
<td>CLOSED</td>
</tr>
<tr>
<td>(timeout)</td>
<td>(SEQ=x, ACK=x+1)</td>
</tr>
<tr>
<td>CLOSED</td>
<td>(SEQ=y, ACK=x+1)</td>
</tr>
<tr>
<td></td>
<td>(SEQ=x+1, ACK=y+1)</td>
</tr>
</tbody>
</table>
TIME_WAIT State

• Wait a long time after sending all segments and before completing the close
  • Two times the maximum segment lifetime of 60 seconds
• Why?
TIME_WAIT State

• Wait a long time after sending all segments and before completing the close
  • Two times the maximum segment lifetime of 60 seconds
• Why?
  • ACK might have been lost, in which case FIN will be resent for an orderly close
  • Could otherwise interfere with a subsequent connection
Flow Control
Flow control goal

Match transmission speed to reception capacity
  • Otherwise data will be lost
ARQ: Automatic repeat query

- ARQ with one message at a time is Stop-and-Wait
Limitation of Stop-and-Wait

• It allows only a single message to be outstanding from the sender:
  • Fine for LAN (only one frame fits in network anyhow)
  • Not efficient for network paths with longer delays
Limitation of Stop-and-Wait (2)

• Example: B=1 Mbps, D = 50 ms
  • RTT (Round Trip Time) = 2D = 100 ms
  • How many packets/sec?
    • 10
  • Usage efficiency if packets are 10kb?
    • \((10,000 \times 10) / (1 \times 10^6) = 10\%\)

• What is the efficiency if B=10 Mbps?
  • 1\%
Sliding Window

• Generalization of stop-and-wait
  • Allows W packets to be outstanding
  • Can send W packets per RTT (=2D)

• Pipelining improves performance
• Need W=2BD to fill network path
Sliding Window (2)

What $W$ will use the network capacity with 10kb packets?

- Ex: $B=1$ Mbps, $D = 50$ ms
  - $2BD = 2 \times 10^6 \times 50/1000 = 100$ Kb
  - $W = 100$ kb/10 = 10 packets

- Ex: What if $B=10$ Mbps?
  - $W = 100$ packets
Sliding Window Protocol

• Many variations, depending on how buffers, acknowledgements, and retransmissions are handled

• Go-Back-N
  • Simplest version, can be inefficient

• Selective Repeat
  • More complex, better performance
Sender Sliding Window

• Sender buffers up to $W$ segments until they are acknowledged
  • $LFS =$ LAST FRAME SENT, $LAR =$ LAST ACK REC’D
  • Sends while $LFS - LAR \leq W$

```
<table>
<thead>
<tr>
<th>Seq. Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acked</td>
</tr>
<tr>
<td>Unacked</td>
</tr>
<tr>
<td>Unavailable</td>
</tr>
</tbody>
</table>

LAR  \quad LFS

Sliding Window

W = 5

Available

seq. number
```
Sender Sliding Window (2)

• Transport accepts another segment of data from the Application ...
  • Transport sends it (LFS–LAR $\rightarrow$ 5)
Sender Sliding Window (3)

• Next higher ACK arrives from peer...
  • Window advances, buffer is freed
  • LFS–LAR → 4 (can send one more)
Receiver Sliding Window – Go-Back-N

• Receiver keeps only a single packet buffer for the next segment
  • State variable, LAS = LAST ACK SENT

• On receive:
  • If seq. number is LAS+1, accept and pass it to app, update LAS, send ACK
  • Otherwise discard (as out of order)
Receiver Sliding Window – Selective Repeat

• Receiver passes data to app in order, and buffers out-of-order segments to reduce retransmissions

• ACK conveys highest in-order segment, plus hints about out-of-order segments
  • Ex: I got everything up to 42 (LAS), and got 44, 45

• TCP uses a selective repeat design; we’ll see the details later
Receiver Sliding Window – Selective Repeat (2)

• Buffers W segments, keeps state variable $\text{LAS} = \text{LAST ACK SENT}$

• On receive:
  • Buffer segments $[\text{LAS}+1, \text{LAS}+W]$
  • Send app in-order segments from $\text{LAS}+1$, and update $\text{LAS}$
  • Send ACK for $\text{LAS}$ regardless
Sender Sliding Window – Selective Repeat

- Keep normal sliding window
- If out-of-order ACK arrives
  - Send LAR+1 again!
Sender Sliding Window – Selective Repeat (2)

- Keep normal sliding window
- If in-order ACK arrives
  - Move window and LAR, send more messages

![Diagram showing sliding window, acknowledged, unacknowledged, LAR, LFS, and sequence number]
Sliding Window – Retransmissions

• Go-Back-N uses a single timer to detect losses
  • On timeout, resends buffered packets starting at LAR+1

• Selective Repeat uses a timer per unacked segment to detect losses
  • On timeout for segment, resend it
  • Hope to resend fewer segments
Sequence Numbers

Need more than 0/1 for Stop-and-Wait ... but how many?
  • For Selective Repeat: 2W seq numbers
    • W for packets, plus W for earlier acks
  • For Go-Back-N: W+1 sequence numbers

Typically implement seq. number with an N-bit counter that wraps around at $2^N-1$
  • E.g., N=8: ..., 253, 254, 255, 0, 1, 2, 3, ...
Sequence Time Plot

Transmissions (at Sender)

Acks (at Receiver)

Delay (=RTT/2)
Sequence Time Plot (2)

Go-Back-N scenario