TCP contd ....

Last class
• Connection setup

This class
• Connection release
Recap: Connection setup

• Three-way handshake:
  • 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

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

Active party (client)

Passive party (server)

SYN (SEQ=x)

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

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

• Connection will be cleanly rejected on both sides 😊
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>
</tbody>
</table>

(SEQ=x, ACK=x+1)

(SEQ=y, ACK=x+1)

(SEQ=x+1, ACK=y+1)
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
  • $W = 100 \text{ kb}/10 = 10$ 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
  
  • $LFS = \text{LAST FRAME SENT}$, $LAR = \text{LAST ACK REC'D}$
  
  • Sends while $LFS - LAR \leq W$

![Diagram of sliding window with Acked, Unacked, and Unavailable segments.

LFS (Last Frame Sent) and LAR (Last Ack Received) are indicated with arrows pointing to their respective positions within the window. The window size $W=5$ is shown.](https://example.com/diagram.png)
Sender Sliding Window (2)

• Transport accepts another segment of data from the Application ...
  • Transport sends it (LFS–LAR → 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 LAS
  • Send ACK for LAS regardless
Sender Sliding Window – Selective Repeat

• Keep normal sliding window
• If out-of-order ACK arrives
  • Send LAR+1 again!

[Diagram showing sliding window with Acked, Unacknowledged, and Unavailable states, with an arrow indicating Ack Arrives Out of Order! and a sequence number indicating LAR and LFS.]

W=5

CSE 461 University of Washington
Sender Sliding Window – Selective Repeat (2)

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

![Diagram of sliding window and sequence numbers](image-url)
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
Sequence Time Plot (3)