TCP Connection Management

- TCP sender, receiver establish "connection" before exchanging data segments
- Initialize TCP variables:
  - Sequence numbers
  - Buffers, flow control info (e.g. RcvWindow)
- Client: connection initiator
  - Socket clientSocket = new Socket("hostname", "port number");
- Server: contacted by client
  - Socket connectionSocket = welcomSocket.accept();

Three way handshake:

1. Client host sends TCP SYN segment to server
   - Specifies initial sequence number
   - No data
2. Server host receives SYN, replies with SYNACK segment
   - Server allocates buffers
   - Specifies server initial sequence number
3. Client receives SYNACK, replies with ACK segment, which may contain data

Closing a connection:

Client closes socket:
clientSocket.close();

Step 1: Client end system sends TCP FIN control segment to server

Step 2: Server receives FIN, replies with ACK.

Steps client sockets FIN, sends FIN.

TCP Client Lifecycle

Step 3: Client receives FIN, replies with ACK.
- Enters "timed wait" - will respond with ACK to received FINs

Step 4: Server receives ACK. Connection closed.

TCP Server Lifecycle

- Server application creates a listen socket
- Listens for incoming connections
- Receives SYN, sends SYN & ACK
- CLOSE_WAIT
- Establishes connection
- Receives FIN, sends ACK
- Data transfer
- CLOSE_WAIT

Principles of Congestion Control

Congestion:
- Informally: "too many sources sending too much data too fast for network to handle"
- Different from flow control
- Manifestations:
  - Lost packets (buffer overflow at routers)
  - Long delays (queueing in router buffers)
- A top-10 problem
**Causes/costs of congestion: scenario 1**
- Two senders, two receivers, bandwidth of the shared link C
- One router, infinite buffers
- No retransmission
- Large delays when congested
- Maximum achievable throughput

**Causes/costs of congestion: scenario 2**
- One router, finite buffers
- Sender retransmission of lost packet

**Causes/costs of congestion: scenario 2**
- Always: \( \lambda_{in} = \lambda_{out} \) (goodput)
- "Perfect" retransmission only when loss: \( \lambda_{in} = \lambda_{out} \)
- Retransmission of delayed (not lost) packet makes \( \lambda_{in} \) larger than perfect case, for same \( \lambda_{out} \)

**Causes/costs of congestion: scenario 3**
- Four senders
- Multi-hop paths
- Retransmit upon timeout

**Approaches towards congestion control**

Two broad approaches towards congestion control:
- End-end congestion control:
  - No explicit feedback from network
  - Congestion inferred from end-system observed loss, delay
  - Approach taken by TCP

- Network-assisted congestion control:
  - Routers provide feedback to end systems
    - Single bit indicating congestion (SNA, DEQbit, TCP/IP ECN, ATM)
  - Explicit rate sender should send at
TCP congestion control: additive increase, multiplicative decrease

- **Approach:** Increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - Additive increase: increase CongWin by 1 MSS every RTT until loss detected
  - Multiplicative decrease: cut CongWin in half after loss

Sawtooth behavior: probing for bandwidth

TCP Congestion Control: details

- **Sender limits transmission:**
  - LastByteSent - LastByteAcked ≤ CongWin

  **Roughly:**
  \[ \text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec} \]

  CongWin is dynamic, function of perceived network congestion

- **How does sender perceive congestion?**
- Loss event = timeout or 3 duplicates acks
- TCP sender reduces rate (CongWin) after loss event

  **Three mechanisms:**
  - **ATMD:**
  - **Slow start**
  - **Conservative after timeout events**

TCP Slow Start

- **When connection begins, CongWin = 1 MSS**
  - Example: MSS = 500 bytes & RTT = 200 msec
  - Initial rate = 20 kbps
- Available bandwidth may be >> MSS/RTT
- Desirable to quickly ramp up to respectable rate
- **When connection begins, increase rate exponentially until first loss event:**
  - Double CongWin every RTT
  - Done by incrementing CongWin for every ACK received
- **Summary:** Initial rate is slow but ramps up exponentially fast

Refinement

- **Q:** When should the exponential increase switch to linear?
  - **A:** When CongWin gets to 1/2 of its value before timeout.

**Implementation:**
- Variable Threshold
- At loss event, threshold is set to 1/2 of CongWin just before loss event

Refinement: inferring loss

- **After 3 dup ACKs:**
  - CongWin is cut in half
  - Window then grows linearly
- **But after timeout event:**
  - CongWin set to 1 MSS
  - Window then grows exponentially
  - To a threshold, then grows linearly

**Philosophy:**
- 3 dup ACKs indicates network capable of delivering some segments
- Timeout indicates a "more alarming" congestion scenario
Summary: TCP Congestion Control

- When CongWin is below Threshold, sender in slow-start phase, window grows exponentially.
- When CongWin is above Threshold, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, Threshold set to CongWin/2 and CongWin set to Threshold.
- When timeout occurs, Threshold set to CongWin/2 and CongWin is set to 1 MSS.

TCP sender congestion control

<table>
<thead>
<tr>
<th>State</th>
<th>Event</th>
<th>TCP Sender Action</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow Start</td>
<td>ACK receipt</td>
<td>CongWin = CongWin + MSS, if CongWin = Threshold, set state to &quot;Congestion Avoidance&quot;</td>
<td>Resulting in a doubling of CongWin every RTT</td>
</tr>
<tr>
<td></td>
<td>ACK receipt</td>
<td>CongWin = CongWin - MSS, if CongWin = Threshold, decrease by a half of CongWin</td>
<td>Additive increase, resulting in increase of CongWin by 1 MSS every RTT</td>
</tr>
<tr>
<td></td>
<td>Loss event</td>
<td>Threshold = CongWin/2, CongWin = Threshold, set state to &quot;Congestion Avoidance&quot;</td>
<td>Fast recovery, implementing multiplicative decrease, CongWin will not drop below 1 MSS.</td>
</tr>
<tr>
<td>Slow Start</td>
<td>Timeout</td>
<td>Threshold = CongWin/2, CongWin = Threshold, set state to &quot;Slow Start&quot;</td>
<td>Enter slow start</td>
</tr>
<tr>
<td>Slow Start</td>
<td>Duplicate ACK</td>
<td>Increment duplicate ACK count for segment being acked</td>
<td>CongWin and Threshold not changed</td>
</tr>
</tbody>
</table>

TCP Throughput

- What’s the average throughput of TCP as a function of window size and RTT?
  - Ignore slow start
  - Let W be the window size when loss occurs.
  - When window is W, throughput is W/RTT
  - Just after loss, window drops to W/2, throughput to W/2RTT
  - Average throughput: .75 W/RTT

TCP Futures

- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- Requires window size $W = 83,333$ in-flight segments
- Throughput in terms of loss rate:
  \[
  \frac{1.22 \times \text{MSS}}{\text{RTT} \sqrt{L}}
  \]
  \[
  \text{L} = 2 \times 10^{10}
  \]
  \text{Wow}
- New versions of TCP for high-speed needed!

TCP Fairness

Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of $R/K$

Why is TCP fair?

Two competing sessions:
- Additive increase gives slope of 1, as throughput increases
- Multiplicative decrease decreases throughput proportionally

Why is TCP fair?

Two competing sessions:
- Additive increase gives slope of 1, as throughput increases
- Multiplicative decrease decreases throughput proportionally
Fairness

Fairness and UDP
- Multimedia apps often do not use TCP
  - Do not want rate throttled by congestion control
- Instead use UDP:
  - Pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

Fairness and parallel TCP connections
- Nothing prevents app from opening parallel connections between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 connections:
  - New app asks for 1 TCP, gets rate R/10
  - New app asks for 11 TCPs, gets R/2