TCP: Overview

- **point-to-point:**
  - one sender, one receiver
- **reliable, in-order byte steam:**
  - no "message boundaries"
- **pipelined:**
  - TCP connection and flow control set window size
- **connection-oriented:**
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- **flow controlled:**
  - sender will not overwhelm receiver

TCP segment structure

- **TCP seq. #s and ACKs**
  - Seq #s:
    - byte stream "number" of first byte in segment's data
  - ACKs:
    - seq # of next byte expected from other side
  - cumulative ACK
  - how receiver handles out-of-order segments
    - A: TCP spec doesn't say, - up to implementor

TCP: reliable data transfer

- simplified sender, assuming one way data transfer
- no flow, congestion control
TCP: reliable data transfer

Simplified TCP sender

TCP ACK generation [RFC 1122, RFC 2581]

TCP: retransmission scenarios

TCP Flow Control
**TCP Round Trip Time and Timeout**

**Q:** how to set TCP timeout value?
- longer than RTT
- note: RTT will vary
- too short: premature timeout
- unnecessary retransmissions
- too long: slow reaction to segment loss

**Q:** how to estimate RTT?
- SampleRTT: measured time from segment transmission until ACK receipt
- ignore retransmissions, cumulatively ACKed segments
- SampleRTT will vary, want estimated RTT "smoother"
- use several recent measurements, not just current SampleRTT

**TCP Round Trip Time and Timeout**

EstimatedRTT = \( (1 - x) \cdot \text{EstimatedRTT} + x \cdot \text{SampleRTT} \)
- Exponential weighted moving average
- influence of given sample decreases exponentially fast
- typical value of \( x \): 0.1

**Setting the timeout**
- \( \text{EstimatedRTT} \) plus "safety margin"
- large variation in \( \text{EstimatedRTT} \) -> larger safety margin

\[
\text{Timeout} = \text{EstimatedRTT} + 4 \cdot \text{Deviation} \\
\text{Deviation} = (1 - x) \cdot \text{Deviation} + x \cdot |\text{SampleRTT} - \text{EstimatedRTT}| 
\]

**TCP Connection Management**

Recall: TCP sender, receiver establish "connection" before exchanging data segments
- initialize TCP variables:
  - seq. #s
  - buffers, flow control info (e.g. RcvWindow)
- client: connection initiator
  - Socket clientSocket = new Socket("hostname", "port number");
- server: contacted by client
  - Socket connectionSocket = connectionServer.accept();

**Three way handshake:**

**Step 1:** client end system sends TCP SYN control segment to server
- specifies initial seq #

**Step 2:** server end system receives SYN, replies with SYNACK control segment
- ACKs received SYN
- allocates buffers
- specifies server -> receiver initial seq #

**TCP Connection Management (cont.)**

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
- closes connection, sends FIN
TCP Connection Management (cont.)

**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.

**Note:** with small modification, can handle simultaneous FINs.

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
- 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 3
- Four senders
- Multihop paths
- Timeout/retransmit

Q: What happens as $\lambda_{in}$ and $\lambda_{out}$ increase?

Another "cost" of congestion:
- When packet dropped, any upstream transmission capacity used for that packet was wasted!
**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, DECBT, TCP/IP ECN, ATM)
  - Explicit rate sender should send at

**TCP Congestion Control**

- End-end control (no network assistance)
- Transmission rate limited by congestion window size, Congwin, over segments:
  - \( \text{Throughput} = \frac{w \times \text{MSS}}{\text{RTT}} \) Bytes/sec

**TCP Slowstart**

- Two "phases"
  - Slow start
  - Congestion avoidance
- Important variables:
  - Congwin
  - Threshold: defines threshold between two slow start phase, congestion control phase
- "probing" for usable bandwidth:
  - Ideally: transmit as fast as possible (Congwin as large as possible) without loss
  - Increase Congwin until loss (congestion)
  - Loss: Decrease Congwin, then begin probing (increasing) again
- Exponential increase (per RTT) in window size (not so slow)
- Loss event: timeout (Tahoe TCP) and/or three duplicate ACKs (Reno TCP)
TCP Congestion Avoidance

```c
/* slowstart is over */
/* Congwin > threshold */
Until (loss event) {
    every w segments ACKed:
        Congwin++
}
threshold = Congwin/2
Congwin = 1
perform slowstart
```

1: TCP Reno skips slowstart (fast recovery) after three duplicate ACKs

TCP Congestion Avoidance

**TCP Fairness**

Fairness goal: if N TCP sessions share a bottleneck link, each should get 1/N of link capacity

**AIMD**

TCP congestion avoidance:
- **AIMD:** additive increase, multiplicative decrease
  - increase window by 1 per RTT
  - decrease window by factor of 2 on loss event

**TCP latency modeling**

**Why is TCP fair?**

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

**TCP latency modeling**

Q: How long does it take to receive an object from a Web server after sending a request?
- **TCP connection establishment**
- **data transfer delay**

Notation, assumptions:
- Assume one link between client and server of rate R
- Assume: fixed congestion window, W segments
- S: MSS (bits)
- Q: object size (bits)
- No retransmissions (no loss, no corruption)

**Two cases to consider**:
- $W/S + RTT + S/R$: ACK for first segment in window returns before window's worth of data sent
- $W/S + RTT + S/R$: wait for ACK after sending window's worth of data sent
TCP Latency Modeling

\[ \text{Case 1: latency = } 2\text{RTT} + O/R \]

\[ \text{Case 2: latency = } 2\text{RTT} + O/R + (K-1)[S/R + \text{RTT} - WS/R] \]

\[ K := \frac{O}{WS} \]

TCP Latency Modeling: Slow Start

- Now suppose window grows according to slow start.
- Will show that the latency of one object of size \( O \) is:

\[ \text{Latency} = 2\text{RTT} + \frac{O}{R} + \left(\frac{R}{S} + \frac{S}{R}\right) - \left(2^p - 1\right) \frac{S}{R} \]

where \( p \) is the number of times TCP stalls at server:

\[ p = \min\{Q, K - 1\} \]

- where \( Q \) is the number of times the server would stall if the object were of infinite size.
- and \( K \) is the number of windows that cover the object.

TCP Latency Modeling: Slow Start (cont.)

Example:

- \( O/S = 15 \) segments
- \( K = 4 \) windows
- \( Q = 2 \)
- \( P = \min\{K-1, Q\} = 2 \)
- Server stalls \( P=2 \) times.