Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

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
  ```java
  Socket clientSocket = new Socket("hostname","port number");
  ```
- **server:** contacted by client
  ```java
  Socket connectionSocket = welcomeSocket.accept();
  ```

**Three way handshake:**

**Step 1:** client host sends TCP SYN segment to server
  - specifies initial seq #
  - no data

**Step 2:** server host receives SYN, replies with SYNACK segment
  - server allocates buffers
  - specifies server initial seq. #

**Step 3:** client receives SYNACK, replies with ACK segment, which may contain data
Three-Way Handshake

TCP Connection Management (cont.)

**Closing a connection:**

client closes socket:
```java
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.

TCP Connection Management (cont)
Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

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

![Diagram](image1)

- large delays when congested
- maximum achievable throughput

Causes/costs of congestion: scenario 2

- one router, finite buffers
- sender retransmission of lost packet
- unneeded retransmissions: link carries multiple copies of pkt

![Diagram](image2)
Causes/costs of congestion

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, DECbit, TCP/IP ECN, ATM)
  - explicit rate sender should send at
Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP Congestion Control

- end-end control (no network assistance)
- sender limits transmission: \[\text{LastByteSent} - \text{LastByteAcked} \leq \text{CongWin}\]
- 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 duplicate acks
- TCP sender reduces rate (CongWin) after loss event

**three mechanisms:**

- AIMD
- slow start
- conservative after timeout events
**TCP AIMD**

- **multiplicative decrease**: cut CongWin in half after loss event
- **additive increase**: increase CongWin by 1 MSS every RTT in the absence of loss events: probing

![Graph showing TCP AIMD behavior](image)

**Long-lived TCP connection**

---

**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 fast until first loss event
TCP Slow Start (more)

- When connection begins, increase rate exponentially until first loss event:
  - double $\text{CongWin}$ every RTT
  - done by incrementing $\text{CongWin}$ for every ACK received

- **Summary:** initial rate is slow but ramps up exponentially fast

Refinement

- After 3 dup ACKs:
  - $\text{CongWin}$ is cut in half
  - window then grows linearly

- But after timeout event:
  - $\text{CongWin}$ instead 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 before 3 dup ACKs is "more alarming"
Refinement (more)

Q: When should the exponential increase switch to linear?
A: When $\text{CongWin}$ 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

Summary: TCP Congestion Control
- When $\text{CongWin}$ is below Threshold, sender in slow-start phase, window grows exponentially.
- When $\text{CongWin}$ is above Threshold, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, Threshold set to $\text{CongWin}/2$ and $\text{CongWin}$ set to Threshold.
- When timeout occurs, Threshold set to $\text{CongWin}/2$ and $\text{CongWin}$ is set to 1 MSS.
**TCP Fairness**

- **Fairness goal**: if $K$ TCP sessions share the same bottleneck link of bandwidth $R$, each should have average rate of $R/K$.
- Practically, this does not happen in TCP as connections with lower RTT are able to grab the available link bandwidth more quickly.

**Diagram:**

- TCP connection 1
- TCP connection 2
- Bottleneck router
- Capacity $R$

---

**Fairness (more)**

**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 an 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 $R/2$!
### TCP Options: Protection Against Wrap Around Sequence

- **32-bit SequenceNum**

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Time Until Wrap Around</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5 Mbps)</td>
<td>6.4 hours</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>57 minutes</td>
</tr>
<tr>
<td>T3 (45 Mbps)</td>
<td>13 minutes</td>
</tr>
<tr>
<td>FDDI (100 Mbps)</td>
<td>6 minutes</td>
</tr>
<tr>
<td>STS-3 (155 Mbps)</td>
<td>4 minutes</td>
</tr>
<tr>
<td>STS-12 (622 Mbps)</td>
<td>55 seconds</td>
</tr>
<tr>
<td>STS-24 (1.2 Gbps)</td>
<td>28 seconds</td>
</tr>
</tbody>
</table>

### TCP Options: Keeping the Pipe Full

- **16-bit AdvertisedWindow**

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Delay x Bandwidth Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (1.5 Mbps)</td>
<td>18KB</td>
</tr>
<tr>
<td>Ethernet (10 Mbps)</td>
<td>122KB</td>
</tr>
<tr>
<td>T3 (45 Mbps)</td>
<td>549KB</td>
</tr>
<tr>
<td>FDDI (100 Mbps)</td>
<td>1.2MB</td>
</tr>
<tr>
<td>STS-3 (155 Mbps)</td>
<td>1.8MB</td>
</tr>
<tr>
<td>STS-12 (622 Mbps)</td>
<td>7.4MB</td>
</tr>
<tr>
<td>STS-24 (1.2 Gbps)</td>
<td>14.8MB</td>
</tr>
</tbody>
</table>

assuming 100ms RTT
TCP Extensions

- Implemented as header options
- Store timestamp in outgoing segments
- Extend sequence space with 32-bit timestamp (PAWS)
- Shift (scale) advertised window