CS 457 – Lecture 21
More TCP

Fall 2011
Establishing a TCP Connection

Three-way handshake to establish connection
- Host A sends a **SYN** (open) to the host B
- Host B returns a SYN acknowledgment (**SYN ACK**)
- Host A sends an **ACK** to acknowledge the SYN ACK

Each host tells its ISN to the other host.
What if the SYN Packet Gets Lost?

• Suppose the SYN packet gets lost
  – Packet is lost inside the network, or
  – Server rejects the packet (e.g., listen queue is full)

• Eventually, no SYN-ACK arrives
  – Sender sets a timer and wait for the SYN-ACK
  – … and retransmits the SYN-ACK if needed

• How should the TCP sender set the timer?
  – Sender has no idea how far away the receiver is
  – Hard to guess a reasonable length of time to wait
  – Some TCPs use a default of 3 or 6 seconds
TCP Retransmissions
Automatic Repeat reQuest (ARQ)

- **Automatic Repeat Request**
  - Receiver sends acknowledgment (ACK) when it receives packet
  - Sender waits for ACK and timeouts if it does not arrive within some time period

- **Simplest ARQ protocol**
  - Stop and wait
  - Send a packet, stop and wait until ACK arrives
Reasons for Retransmission

Packet lost

ACK lost
DUPLICATE PACKET

Early timeout
DUPLICATE PACKETS
How Long Should Sender Wait?

• Sender sets a timeout to wait for an ACK
  – Too short: wasted retransmissions
  – Too long: excessive delays when packet lost

• TCP sets timeout as a function of the RTT
  – Expect ACK to arrive after an RTT
  – … plus a fudge factor to account for queuing

• But, how does the sender know the RTT?
  – Can estimate the RTT by watching the ACKs
  – Smooth estimate: keep a running average of the RTT
    • EstimatedRTT = a * EstimatedRTT + (1 –a ) * SampleRTT
  – Compute timeout: TimeOut = 2 * EstimatedRTT
Example RTT Estimation

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr

![Graph showing RTT (milliseconds) over time (seconds) with two lines: SampleRTT and Estimated RTT.](image-url)
A Flaw in This Approach

• An ACK doesn’t really acknowledge a transmission
  – Rather, it acknowledges receipt of the data

• Consider a retransmission of a lost packet
  – If you assume the ACK goes with the 1st transmission
  – ... the SampleRTT comes out way too large

• Consider a duplicate packet
  – If you assume the ACK goes with the 2nd transmission
  – ... the Sample RTT comes out way too small

• Simple solution in the Karn/Partridge algorithm
  – Only collect samples for segments sent one single time
Yet Another Limitation…

• Doesn’t consider variance in the RTT
  – If variance is small, the EstimatedRTT is pretty accurate
  – … but, if variance is large, the estimate isn’t all that good

• Better to directly consider the variance
  – Consider difference: SampleRTT – EstimatedRTT
  – Boost the estimate based on the variance

• Jacobson/Karels algorithm
  – See Section 5.2 of the Peterson/Davie book for details
TCP Sliding Window
Motivation for Sliding Window

- Stop-and-wait is inefficient
  - Only one TCP segment is “in flight” at a time
  - Especially bad when delay-bandwidth product is high

- Numerical example
  - 1.5 Mbps link with a 45 msec round-trip time (RTT)
    - Delay-bandwidth product is 67.5 Kbits (or 8 KBytes)
    - But, sender can send at most one packet per RTT
    - Assuming a segment size of 1 KB (8 Kbits)
      - … leads to 8 Kbits/segment / 45 msec/segment  182 Kbps
  - That’s just one-eighth of the 1.5 Mbps link capacity
Sliding Window

- Allow a larger amount of data “in flight”
  - Allow sender to get ahead of the receiver
  - … though not *too far* ahead

Sending process

TCP

Last byte written

Last byte sent

Last byte ACKed

Receiving process

TCP

Last byte read

Next byte expected

Last byte received
Flow Control and Receiver Buffering

- **Window size**
  - Amount that can be sent without acknowledgment
  - Receiver needs to be able to store this amount of data

- **Receiver advertises the window to the receiver**
  - Tells the receiver the amount of free space left
  - … and the sender agrees not to exceed this amount
TCP Header for Receiver Buffering

<table>
<thead>
<tr>
<th>Flags:</th>
<th>SYN</th>
<th>FIN</th>
<th>RST</th>
<th>PSH</th>
<th>URG</th>
<th>ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source port</td>
<td>Destination port</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HdrLen</td>
<td>0</td>
<td>Flags</td>
<td>Advertised window</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fast Retransmission
Timeout is Inefficient

- Timeout-based retransmission
  - Sender transmits a packet and waits until timer expires
  - ... and then retransmits from the lost packet onward

```
sender
send pkt0
send pkt1
send pkt2
send pkt3 (wait)
rcv ACK0
send pkt4
rcv ACK1
send pkt5
pkt2 timeout
send pkt2
send pkt3
send pkt4
send pkt5
```

```
receiver
rcv pkt0
send ACK0
rcv pkt1
send ACK1
rcv pkt3, discard
send ACK1
rcv pkt4, discard
send ACK1
rcv pkt5, discard
send ACK1
rcv pkt2, deliver
send ACK2
rcv pkt3, deliver
send ACK3
```
Fast Retransmission

• Better solution possible under sliding window
  – Although packet n might have been lost
  – … packets n+1, n+2, and so on might get through

• Idea: have the receiver send ACK packets
  – ACK says that receiver is still awaiting n^{th} packet
    • And repeated ACKs suggest later packets have arrived
  – Sender can view the “duplicate ACKs” as an early hint
    • … that the n^{th} packet must have been lost
    • … and perform the retransmission early

• Fast retransmission
  – Sender retransmits data after the triple duplicate ACK
Effectiveness of Fast Retransmit

• When does Fast Retransmit work best?
  – Long data transfers
    • High likelihood of many packets in flight
  – Large window size
    • High likelihood of many packets in flight
  – Low burstiness in packet losses
    • Higher likelihood that later packets arrive successfully

• Implications for Web traffic
  – Most Web transfers are short (e.g., 10 packets)
    • Short HTML files or small images
  – So, often there aren’t many packets in flight
  – … making fast retransmit less likely to “kick in”
  – Forcing users to like “reload” more often… 😊
Tearing Down the Connection
Tearing Down the Connection

- Closing the connection
  - Finish (FIN) to close and receive remaining bytes
  - And other host sends a FIN ACK to acknowledge
  - Reset (RST) to close and not receive remaining bytes
Sending/Receiving the FIN Packet

• Sending a FIN: close()
  – Process is done sending data via the socket
  – Process invokes “close()” to close the socket
  – Once TCP has sent all of the outstanding bytes…
  – … then TCP sends a FIN

• Receiving a FIN: EOF
  – Process is reading data from the socket
  – Eventually, the attempt to read returns an EOF
TCP Connection Management - Close

Closing a connection:

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 - Close

**Step 3:** client receives FIN, replies with ACK.

- Enters “time-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

TCP client lifecycle:
- CLOSED
- TIME_WA
- FIN_WAIT_2
- FIN_WAIT_1
- ESTABLISHED
- LAST_ACK
- CLOSE_WAIT
- ESTABLISHED

TCP server lifecycle:
- CLOSED
- LISTEN
- SYN_RCVD
- ESTABLISHED

Client application initiates a TCP connection with SYN.
Client application initiates close connection with FIN.
Server application creates a listen socket with LISTEN.
Receive FIN and send ACK.
Send FIN and receive ACK.
Wait 30 seconds.
Receive FIN and send ACK.
Receive SYN & ACK and send ACK.
What’s Next

• Read Chapter 1, 2, 3, 4.1-4.3, and 5.1-5.2
• Next Lecture Topics from Chapter 6.1 and 6.2
  – Congestion Control
• Homework
  – Due Thursday in lecture
• Project 3
  – Posted on the course website