Lecture 22: TCP & NAT
Lecture 22 Overview

- TCP Connection Management
  
- TCP Slow Start
  - Allow TCP to adjust to links of any speed

- Fast Retransmit & Recovery
  - Avoid wasting capacity due to inevitable packet loss

- Network Address Translation
  - Yet another layer of indirection!
TCP State Transitions

- CLOSED
  - Passive open
  - Close
  - Active open/SYN

- LISTEN
  - SYN/SYN + ACK
  - Send/ SYN
  - Close

- SYN_RCVD
  - SYN/SYN + ACK
  - ACK
  - Close /FIN

- ESTABLISHED
  - SYN + ACK/ACK
  - FIN/ACK
  - Close /FIN

- FIN_WAIT_1
  - ACK
  - FIN/ACK
  - Close /FIN

- FIN_WAIT_2
  - FIN/ACK

- CLOSING
  - ACK
  - Timeout after two segment lifetimes

- CLOSE_WAIT
  - ACK
  - Close /FIN

- LAST_ACK
  - ACK

- TIME_WAIT
  - ACK

- CLOSED
Again, with States

Active participant (client)

SYN_SENT

SYN, SequenceNum = x

SYN + ACK, SequenceNum = y, Acknowledgment = x + 1

ACK, Acknowledgment = y + 1

ESTABLISHED

+data

Passive participant (server)

LISTEN

SYN_RCVD

ESTABLISHED
Connection Teardown

- Orderly release by sender and receiver when done
  - Delivers all pending data and “hangs up”

- Cleans up state in sender and receiver

- TCP provides a “symmetric” close
  - Both sides shutdown independently
TCP Connection Teardown

**Web server**

- FIN_WAIT_1
- FIN_WAIT_2
- TIME_WAIT
- CLOSED

**Web browser**

- CLOSE_WAIT
- LAST_ACK
- CLOSED

Diagram showing the process of TCP connection teardown with state transitions such as **FIN_WAIT_1**, **FIN_WAIT_2**, **TIME_WAIT**, and **CLOSED**.
The TIME_WAIT State

- We wait $2^{\text{MSL}}$ (maximum segment lifetime of 60 seconds) before completing the close
  - Why?

- ACK might have been lost and so FIN will be resent
  - Could interfere with a subsequent connection

- Real life: Abortive close
  - Don’t wait for $2^{\text{MSL}}$, simply send Reset packet (RST)
  - Why?
TCP Bandwidth Probing

- TCP uses AIMD to adjust congestion window
  - Converges to fair share of bottleneck link
  - Increases modestly in good times
  - Cuts drastically in bad times

- But what rate should a TCP flow use initially?
  - Need some initial congestion window
  - We’d like to TCP to work on all manner of links
  - Need to span 6+ orders of magnitude, e.g., 10 K to 10 Gbps.
  - Starting too fast is catastrophic!
Slow Start

- Goal: quickly find the equilibrium sending rate
- Quickly increase sending rate until congestion detected
  - Remember last rate that worked and don’t overshoot it
- TCP Reno Algorithm:
  - On new connection, or after timeout, set $cwnd=1$ MSS
  - For each segment acknowledged, increment $cwnd$ by 1 MSS
  - If timeout then divide $cwnd$ by 2, and set $ssthresh = cwnd$
  - If $cwnd \geq ssthresh$ then exit slow start
- Why called slow? Its exponential after all…
Slow Start Example

Sender

<table>
<thead>
<tr>
<th>cwnd=1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ack 2</td>
</tr>
<tr>
<td>cwnd=2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>cwnd=4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td>cwnd=8</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Ack 3</td>
</tr>
<tr>
<td></td>
<td>Ack 4</td>
</tr>
<tr>
<td></td>
<td>Ack 5</td>
</tr>
<tr>
<td></td>
<td>Ack 6</td>
</tr>
<tr>
<td></td>
<td>Ack 7</td>
</tr>
<tr>
<td></td>
<td>Ack 8</td>
</tr>
</tbody>
</table>

Receiver

0 1 2 3 4 5 6 7 8
0 50 100 150 200 250 300

round-trip times
cwnd
Basic Mechanisms

Slow Start + Congestion Avoidance

- cwnd
- round-trip times

Timeout
Congestion avoidance
ssthresh

Slow start
Fast Retransmit & Recovery

- **Fast retransmit**
  - Timeouts are slow (default often 200 ms or 1 second)
  - When packet is lost, receiver still ACKs last in-order packet
  - Use 3 duplicate ACKs to indicate a loss; detect losses quickly
    » Why 3? When wouldn’t this work?

- **Fast recovery**
  - Goal: avoid stalling after loss
  - If there are still ACKs coming in, then no need for slow start
  - If a packet has made it through -> we can send another one
  - Divide $cwnd$ by 2 after fast retransmit
  - Increment $cwnd$ by 1 MSS for each additional duplicate ACK
Fast Retransmit Example

Fast Retransmit Example

- Fast retransmit
  - (increase cwnd by 1)

CSE 123 – Lecture 22: TCP & NAT
More Sophistication

Slow Start + Congestion Avoidance + Fast Retransmit + Fast Recovery

round-trip times

cwnd

Fast recovery

CSE 123 – Lecture 22: TCP & NAT
Delayed ACKs

- In request/response programs, want to combine an ACK to a request with a response in same packet
  - Wait 40—200 ms before ACKing
  - Must ACK every other packet (or packet burst)
  - Impact on slow start?

- Must not delay duplicate ACKs
  - Why? What is the interaction with the congestion control algorithms?
Short Connections

- Short connection: only contains a few pkts
- How do short connections and Slow-Start interact?
  - What happens when a packet is lost during Slow-Start?
  - What happens when the SYN is dropped?
- Bottom line: Which packet gets dropped matters a lot
  - SYN
  - Slow-Start
  - Congestion avoidance
- Do you think most flows are short or long?
- Do you think most traffic is in short flows or long flows?
Open Issues

● TCP is designed around the premise of cooperation
  ◆ What happens to TCP if it competes with a UDP flow?
  ◆ What if we divide cwnd by 3 instead of 2 after a loss?

● There are a bunch of magic numbers
  ◆ Decrease by 2x, increase by $1/cwnd$, 3 duplicate acks, $g=1/8$, initial timeout = 3 seconds, etc.

● But overall it works really well!
  ◆ Still being constantly tweaked…
TCP Probes the network for bandwidth, assuming that loss signals congestion.

The congestion window is managed with an additive increase/multiplicative decrease policy:
- It took fast retransmit and fast recovery to get there.
- Fast recovery keeps pipe “full” while recovering from a loss.

Slow start is used to avoid lengthy initial delays:
- Ramp up to near target rate, then switch to AIMD.
Lots of Icky Details

- Window probes
- Silly Window Syndrome
- Nagle’s algorithm
- PAWS
- Etc…

Steven’s books “TCP/IP Illustrated (vol 1,2)” is a great source of information on this
Private Address Space

- Sometimes you can’t get/don’t want IP addresses
  - An organization wants to change service providers without having to renumber its entire network
  - A network may be unable obtain (or cannot afford) enough IP addresses for all of its hosts

- IP provides **private address space** anyone can use
  - 10/8, 192.168/16, 172.16.0/20
  - These addresses are not routable—Internet routers should drop packets destined to these so-called **bogons**

- What good are they if can’t use them on the Internet?
Network Address Translation

- Gateway router can rewrite IP addresses as packets leave or enter a given network
  - I.e., replace private addresses with public ones
  - Router needs to see and update every packet

- Maintains a mapping of private-to-public addresses
  - Simple case is a one-to-one mapping
  - Anytime network changes provider, just update mapping table
  - In more clever scenarios, can map a set of private addresses to a smaller set of public addresses
  - In the extreme map the entire private network to one public IP!
IP Masquerading

- A.K.A. Network Address and port Translation (NAPT), Port Address Translation (PAT), or, colloquially, just NAT.
- Entire local network uses just one IP address as far as outside world is concerned:
  - can change addresses of devices in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - devices inside local net not explicitly addressable, visible by outside world (a security plus).
All packets leaving local network have same single source NAT IP address: 138.76.29.7, different source port numbers

Packets with source or destination in this network have 10.0.0.0/8 address for source, destination (as usual)
NA(p)T Example

1: host 10.0.0.4 sends packet to 132.239.8.45:80

2: NAT router changes packet source addr from 10.0.0.1:3345 to 138.76.29.7:5001, updates table

S: 10.0.0.4
D: 132.239.8.45:80

WAN side addr | LAN side addr
---|---
138.76.29.7:5001 | 10.0.0.4:3345

3: Reply arrives
dest. address:
138.76.29.7:5001

S: 132.239.8.45:80
D: 10.0.0.4:3345

4: NAT router changes packet dest addr from 138.76.29.7:5001 to 10.0.0.4:3345
NAT Challenges

- End hosts may not be aware of external IP address
  - Some applications include IP addresses in application data
  - Packets will contain private IP addresses inside payload
  - Many NATs will inspect/rewrite certain protocols, e.g., FTP

- NAT’d end hosts are not reachable from the Internet
  - All connections must be initiated from within private network
  - Alternative is to configure fixed forwarding in NAT
  - Many protocols for NAT traversal to get around this
For next time…

- Read P&D 6.2, 6.5