Lecture 8: TCP Friendliness, DCCP, NATs, and STUN
Congestion Control

- TCP dynamically adapts its rate in response to congestion
- AIMD causes flows to converge to fair goodput
- But how do losses (e.g., bit errors) affect goodput?
- What about UDP?
Chiu Jain Phase Plots

Flow A rate (bps)
Flow B rate (bps)

Fair
A=B

Efficient
A+B=C

overload

underload

t1
t2
t3
t4
t5
t6
Responding to Loss

- Set threshold to $\frac{cwnd}{2}$

- On timeout
  - Set cwnd to 1
  - Causes TCP to enter slow start

- On triple duplicate ACK (Reno)
  - Set cwnd to $\frac{cwnd}{2}$
  - Retransmit missing segment
  - Causes TCP to stay in congestion avoidance
Analyzing TCP Simply

• Assume all segments are MSS long
• Assume a packet loss rate $p$
• Assume a constant RTT
• Assume $p$ is small (no timeouts)
Analysis

- Window size $W$ cuts to $\frac{W}{2}$ after a loss
- Grows to $W$ after $\frac{W}{2}$ RTTs
- Goodput $= \frac{3}{4} \cdot W \cdot MTU \cdot \frac{1}{RTT}$
Window Size

- \( p = \frac{1}{\left(\frac{W}{2} + \left(\frac{W}{2} + 1\right) + \ldots + W\right)} \)
- \( p \approx \frac{1}{\frac{3}{8} W^2} \)
- \( W \approx \sqrt{\frac{8}{3 \cdot p}} \)
- \( \text{Goodput} = \frac{3}{4} \cdot \sqrt{\frac{8}{3 \cdot p}} \cdot MTU \cdot \frac{1}{RTT} \)
- \( \text{Goodput} = \frac{1.22 \cdot MTU}{RTT \cdot \sqrt{p}} \)
- Constant factor changes based on delayed acks, etc.
TCP Friendliness

• Don’t want other protocols to disrupt TCP
• UDP happily shuts down TCP flows
• “TCP friendliness:” obeying TCP congestion control as per prior goodput equation
  - Does not imply acting like TCP
  - E.g., does not require abrupt window changes
DCCP

- Datagram Congestion Control Protocol (DCCP) provides congestion control for unreliable datagrams (RFC 4340)

- Connection-oriented protocol
  - Request-response-ack establishment
  - Close-reset or CloseReq-Close-reset teardown

- Counts packets, not bytes
DCCP Segment

(a)

0  8  16  24
Source Port  Destination Port
Data Offset  CCVal  CsCov  Checksum
Res  Type  1  Reserved

Sequence Number

Sequence Number (low bits)

(b)

Reserved  Acknowledgement Number

Acknowledgement Number (low bits)
Sequence Numbers

• Every DCCP packet uses a new sequence number
  - Data
  - Acknowledgements
  - Control traffic

• Acknowledgements are for *last packet received*
  - Not cumulative acknowledgements
  - Does not succinctly describe connection history
  - Options can give packet vectors
Synchronization

- DCCP uses sequence number windows to protect from attacks
- Large bursts of losses cause packets to fall past windows
- Need to resynchronize
Synchronization Exchange

- Data(seq 0)
- Data(seq 1) $\times$
- Data(seq 500) $\times$
- Data(seq 501)
- Sync(seq 1, ack 501)
- SyncAck(seq 502, ack 1)

Expect seqnos [0, 100]
Odd seq, send Sync
Update to [502, 602]
Synchronization on Reset Problem

- Expect seqnos [200, 300]
- Old seq/ack, send Sync
- Repeat

- Data(seq 1)
- Data(seq 500)
- Reset(seq 0, ack 1)
- Sync(seq 501, ack 0)
- Reset(seq 1, ack 501)

- Timeout and close
- No socket
- No socket

...
Synchronization on Reset Solution

Expect seqnos [200, 300]

Data(seq 1)

... Data(seq 500)

Reset(seq 0, ack 1)

Sync(seq 501, ack 200)

Reset(seq 201, ack 501)

Timeout and close

No socket

No socket

Old seq/ack, send Sync

OK, reset
DCCP options

• Data offset field $\geq$ generic header size

• Optional header fields
  - Padding (0x00)
  - Mandatory (0x01), reset if not possible
  - Change/Confirm L/R (0x20, 0x21, 0x22, 0x23)
  - Ack Vector
Congestion Control

- Defines Congestion Control IDs (CCIDs)
- Negotiated with change/confirm L/R options
- Each half-connection can have different congestion control
  - CCID 2: TCP congestion control (AIMD) (RFC 4941)
  - CCID 3: TCP-friendly congestion control (RFC 4942)
Ack Vector

- 0: Received, 1: Received ECN, 2: Reserved, 3: Not yet received

+------------------+----------------------------------+
| 0010011? | Length | SSLLLLLL | SSLLLLLL | SSLLLLLL | ... |
+------------------+----------------------------------+
Type=38/39 \___________ Vector __________...

0 1 2 3 4 5 6 7
+------------------+
| Sta | Run Length |
+------------------+
CCID 2

- **Uses TCP congestion control**
  - Maintains a cwnd, slow-start, etc.

- **Adds congestion control to acks**
  - Sender specifies an AckRatio, \( R \)
  - Ratio of data to ack packets (TCP with delayed ACKs is 2)
  - On detecting ack losses, double \( R \)
  - After \( \frac{cwnd}{R^2 - R} \) lossless congestion windows, decrement \( R \)
CCID 3

- Uses TCP-friendly congestion control
- Uses a sending rate, rather than a congestion window
- Receiver sends feedback once per RTT, reporting loss rate
- If sender hears no feedback, halves sending rate
- Security issue with loss rate reporting: report loss intervals, rather than just a loss rate, verifiable with ECN nonces
DCCP Today

• Numerous implementations
• IETF Standards Track
• Well suited to VoIP, Internet Gaming, etc.
• Sees very little use
NAT

- Network Address Translator

Session A-S
128.34.22.8:6101
18.181.0.31:22

Session B-S
76.18.117.20:10001
18.181.0.31:22

Server
(18.181.0.31)

NAT (128.34.22.8)

Client A
(10.0.0.101)

NAT (76.18.117.20)

Client B
(10.1.1.9)
Motivations and Complications

- There are only $2^{32}$ IP addresses
- Firewalls for security
- Breaks end-to-end (node does not know its external IP)
- Node might not even know if it’s behind a NAT
- NAT needs to be able to dynamically assign mappings
Types of NAT (RFC 3849)

• Full Cone: no ingress filter (single local-external mapping)
• Restricted Cone: ingress filter on address
• Port Restricted: ingress filter on address/port
• Symmetric: different mappings for different external destinations
• Terminology is imperfect (static port mappings, etc.)
How a NAT Works

- Maps between global and local (IP, port) pairs
- Requires knowledge of transport packet format
- UDP datagram, TCP SYN
  - Can shut down TCP mapping on FIN+ACK
  - UDP requires timeouts (> 2 minutes, unless IANA says otherwise)
- RFC 4787/BCP 127 defines recommended behaviors
NAT Problems

- Incoming TCP connections
- E.g., Skype
TCP Through NATs

- Server socket doesn’t initiate traffic: NAT can’t set up mapping
- Rendezvous servers (as in Skype)
- Connection reversal through rendezvous if only one is behind a NAT (rendezvous server asks un-NAT node to open a port so NAT node can connect)
TCP Reversal
More NAT Problems

- Port mapping: 0-1023 should map to 0-1023
- Port parity: even port → even port, odd port → odd port (RFC 3550: RTP uses even, RTCP uses odd)
- Hairpinning: two nodes behind a NAT communicate with external IPs
STUN (RFC 3849)

• “Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)”

• Enables a node to
  - Determine if it is behind a NAT, and if so, what kind
  - Obtain a public IP address/port pair

• Client-server protocol, requires no changes to NATs

• STUN server coordinates
STUN Binding Requests

- Node sends BR to STUN server
- STUN sends a response that has the address and port it sees
  - If different than node’s local address and port, it’s behind a NAT
- Node can probe to see what kind of NAT
  - Ask STUN server to respond from different address: no response, address restricted; different response, symmetric
  - Ask STUN server to respond from different port: no response, port restricted
- When does STUN not work?
NAT Hole-Punching

- Problem with STUN: doesn’t work when two nodes are behind same NAT
- Two nodes A, B communicate with server S
- A and B report their local IP address to server
- Server tells the other the address/port pair \((L, G)\)
- A tries to send UDP packets to \((L_B, G_B)\) using \(L_A\)
- B tries to send UDP packets to \((L_A, G_A)\) using \(L_B\)
NAT Hole-Punching Example

Server
(18.181.0.31)

NAT
(128.34.22.8)

Client A
(10.0.0.101)

NAT
(76.18.117.20)

Client B
(10.1.1.9)
NAT Hole-Punching Example

1. Request connection to B

Server (18.181.0.31)

NAT (128.34.22.8)  NAT (76.18.117.20)

Client A (10.0.0.101)  Client B (10.1.1.9)
NAT Hole-Punching Example

2a. S sends $L_B$ and $G_B$ to A

2b. S sends $L_A$ and $G_A$ to B
NAT Hole-Punching Example

3a. A sends to $L_B$ and $G_B$

Server
(18.181.0.31)

3b. B sends to $L_A$ and $G_A$

NAT
(128.34.22.8)

NAT
(76.18.117.20)

Client A
(10.0.0.101)

Client B
(10.1.1.9)
NAT Hole-Punching Example

4. Session A-B established

Server
(18.181.0.31)

NAT
(128.34.22.8)

Client A
(10.0.0.101)

NAT
(76.18.117.20)

Client B
(10.1.1.9)
Multiple NAT Layers

- Common for consumer Internet: ISP has internal NAT, end user places another NAT
- Requires hairpinning at ISP NAT
The New Hourglass

Layers 5-7

TCP  UDP  ICMP

IP

Layers 1-2