CSC458 Programming Assignment II: NAT

Recall PA I

- Static topology + static routing table
- IP Routing + ICMP messages
- ARP requests and replies

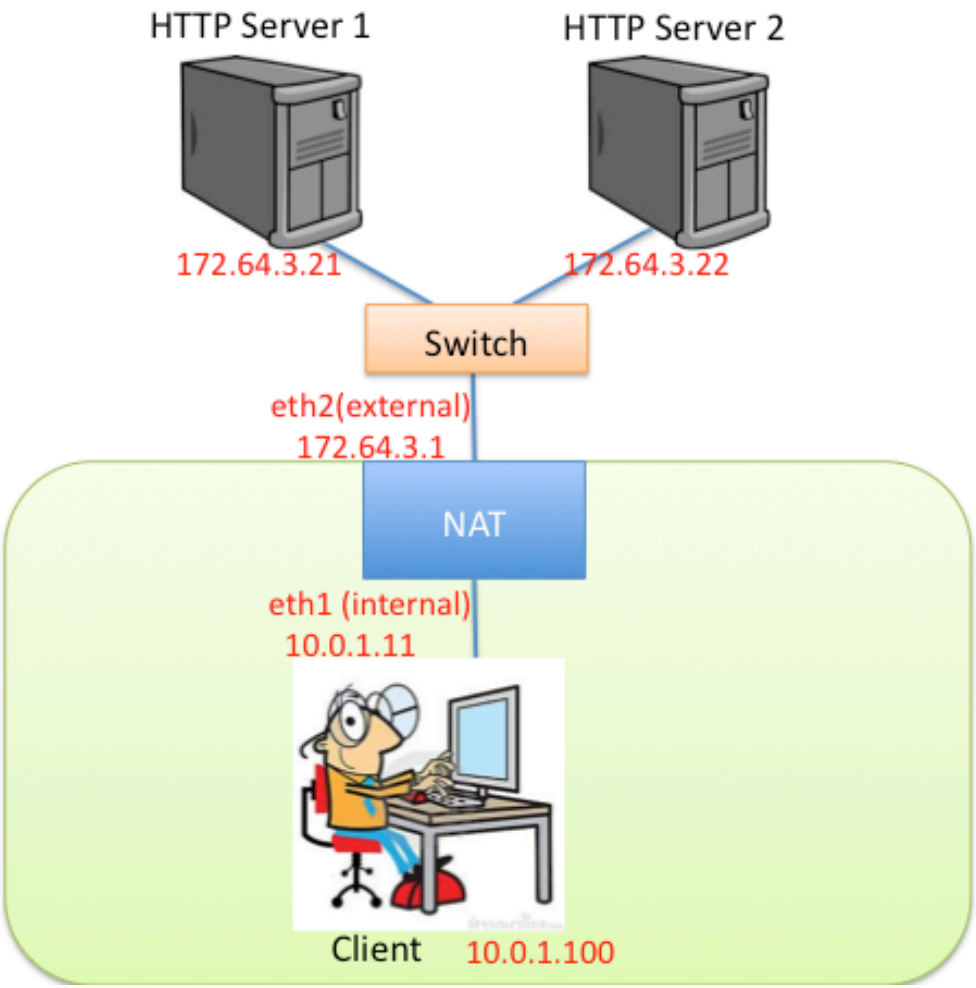
PA II Overview

- You’re going to write a “simplified” NAT (+Router)
- Take your PA I
- NAT handling ICMP and TCP

Recap: What is NAT?

- NAT = **N**etwork **A**ddress **T**ranslation
- Translate between IP address ranges
  - Private to public IP address
  - Private: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
- Enables one IP address to be shared among lots of devices
- Conserve IPv4 addresses

Outline



- Use mininet to create a NAT
- HTTP servers and switch are outside of NAT
- Client is inside NAT

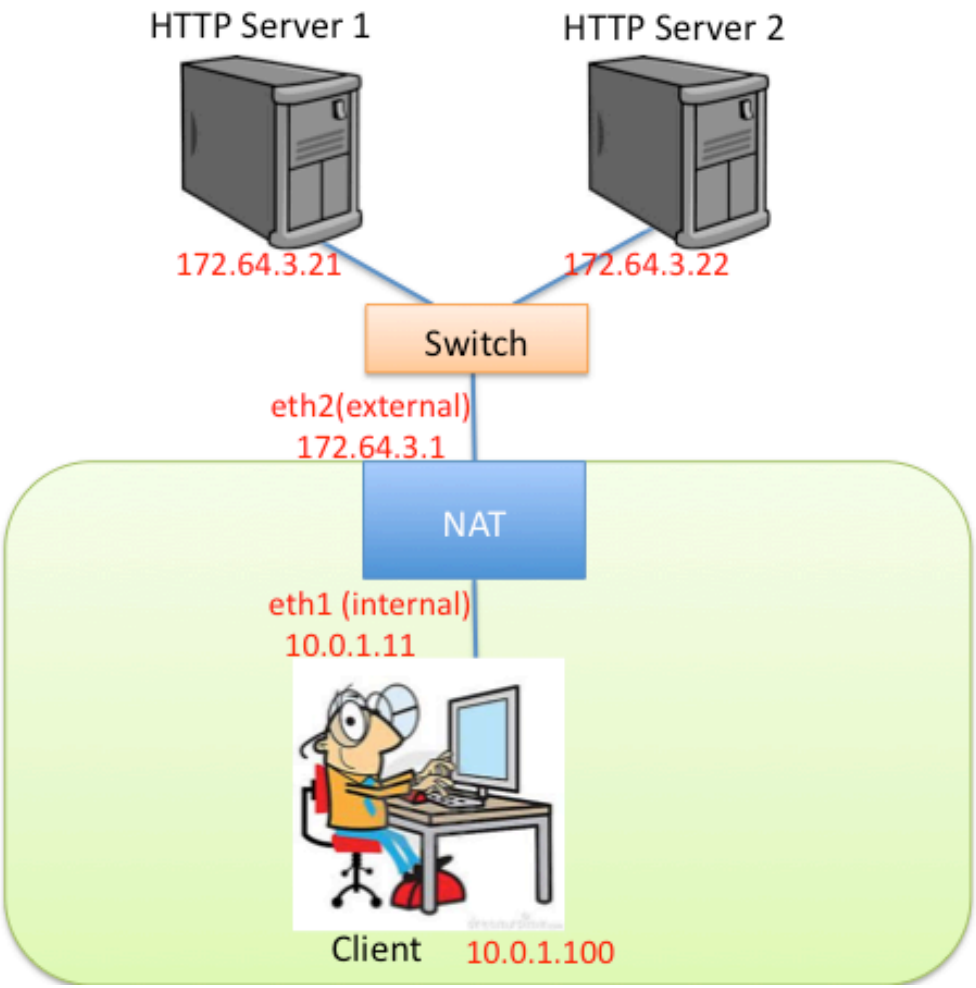
Required Functionality

- Pinging the NAT's internal interface from client host machines
- Pinging any of the HTTP servers
- Downloading files using HTTP from the app servers

Required Functionality

- Keep your PA 1 functionality
  - Enable NAT with ./sr_nat –n
- ICMP messages
  - Ping echo request + reply
  - External host independence
- TCP packets
  - Endpoint-Independent mapping behavior
  - Endpoint-Independent filtering
  - Simultaneous open
- Mapping timeouts MUST be configurable
- Refer to course webpage for detailed instructions

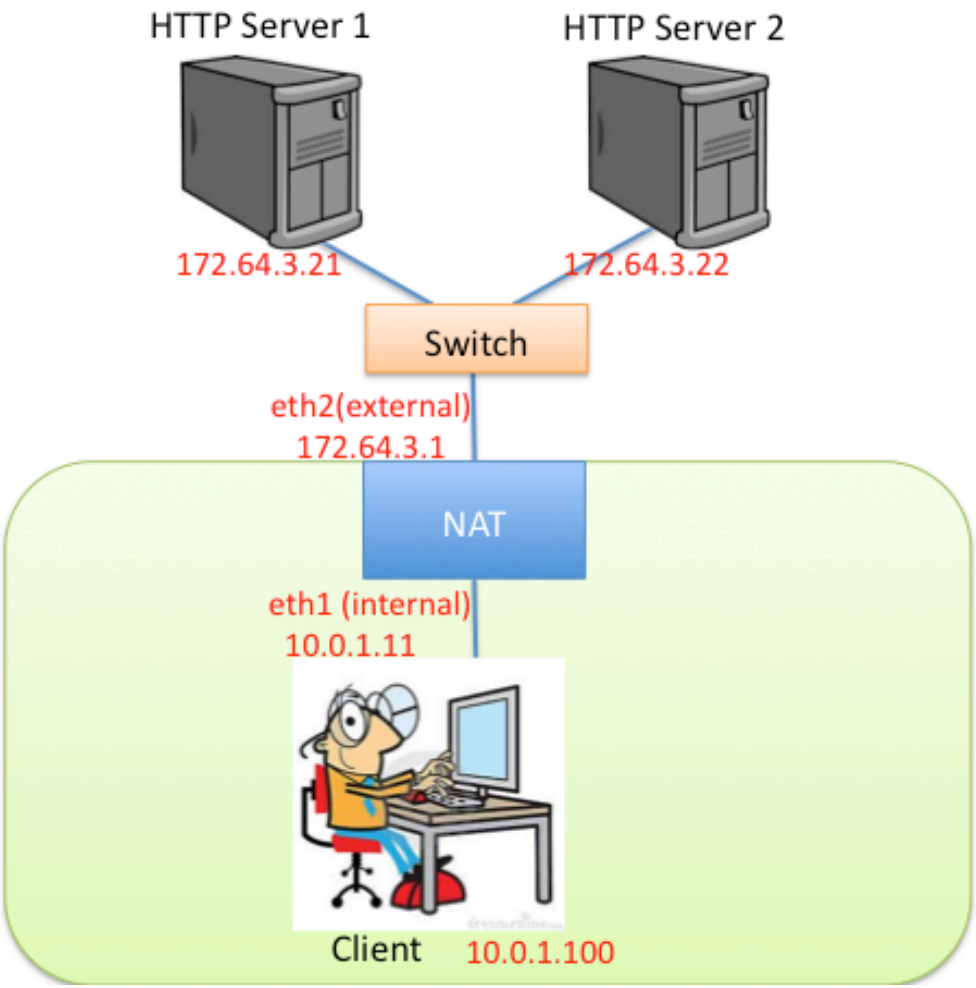
Example: ICMP Echo Request



Ping (ICMP ECHO)
Src: 172.64.3.1
Dst: 172.64.3.21
Id: 100

Ping (ICMP ECHO)
Src: client
Dst: 172.64.3.21
Id: 10

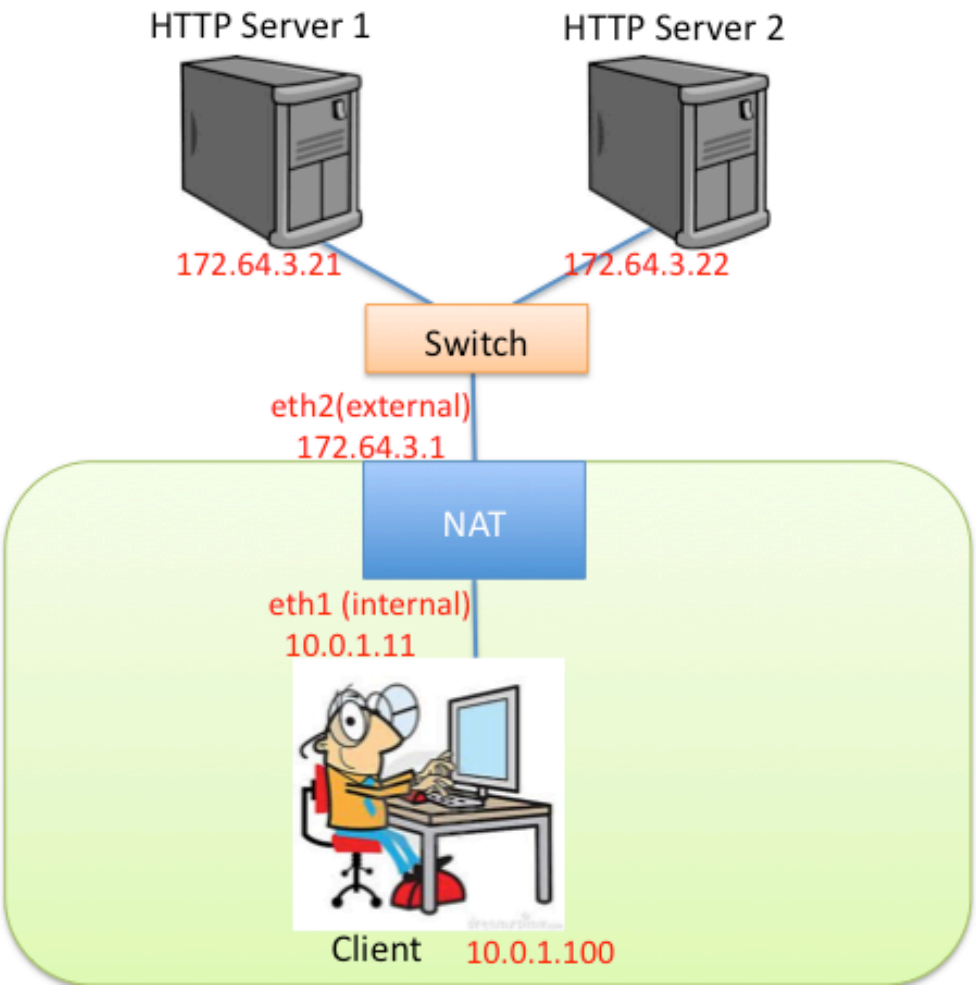
Example: ICMP Echo Reply



Ping (ICMP ECHO Reply)
Src: 172.64.3.21
Dst: 172.64.3.1
Id: 100

Ping (ICMP ECHO Reply)
Src: 172.64.3.21
Dst: client
Id: 10

ICMP: External Host Independence



Rewriting is the same,
Independent of packet destination

Timeout after 60 seconds

Ping (ICMP ECHO)
Src: client
Dst: 172.64.3.21
Id: 10

Ping (ICMP ECHO)
Src: client
Dst: 172.64.3.22
Id: 10

Ping (ICMP ECHO)
Src: 172.64.3.1
Dst: 172.64.3.21
Id: 100

Ping (ICMP ECHO)
Src: 172.64.3.1
Dst: 172.64.3.22
Id: 100

TCP: Requirements

- Endpoint-Independent Mapping behavior
  - If a mapping (x,px)→(x',px') is created for a packet destined to (y1,py1)
  - Then the same mapping is used for for packets from (x,px) to (y2,py2) (i.e. doesn't depend on endpoints)
- Endpoint-Independent Filtering
  - If a mapping (x,px)→(x',px') is created for (y1,py1), then allow (y2,py2) to communicate to (x,px) via (x',px')
- Simultaneous open
  - Dealing with Inbound SYNs
- Timeouts and a few others (webpage lists all)
- Use ports > 1023 for mapping.

Threads!

- Spawn a thread to timeout NAT entries
  - Similar to ARP cache timeouts
  - Search code for pthread_create, pthread_mutex_lock
- Synchronize access to shared data using locks
  - NAT mapping lookup, insertion, deletion, etc.
- Be conservative with locks☺
  - Race conditions harder to debug than deadlocks

Data Structures

- Mapping table
  - Linked list is fine, O(n) lookup
  - ICMP or TCP
  - Keep a time field to remember when a mapping was used
- Remember: locks!
  - Protect all accesses: lookup, insertion, deletion

Rough pseudocode

```
Receive packet on an interface
Check if ICMP or TCP
If packet is outbound (internal -> external)
    insert or lookup unique mapping
else:
    if no mapping and not a SYN (for simultaneous open)
        drop packet
Rewrite IP src (dst) for outgoing (incoming) packets
Rewrite ICMP ID / TCP port
Update relevant checksums
Route packet
```

Rough pseudocode

- This is just to give you a rough idea
  - Details missing (rules for filtering, reusing mapping, etc.)
  - ICMP port unreachable, etc. Locks ☺
  - Read instructions for detailed information
- Start early!
- Due Friday Nov. 28th

Summary

- Build on top of PA 1
- NAT must work for ICMP, TCP
  - UDP: not required, but up to you...
- Clear defunct mappings using timeouts
- Timeouts must be configurable
  - ICMP query id timeout
  - TCP established timeout
  - TCP transitory timeout (e.g. SYN sent but no response)
- Which means you have to track TCP state transitions ☺

Things Might Be Helpful

- Modularize your code!
  - To handle NAT state insertions, lookups, etc.
  - Dealing with ICMP, TCP
- Start with ICMP first, then move to TCP
- Debugging workflow
  - Mininet console, tcpdump, ping, wireshark
  - GDB/Valgrind
- Debug Functions
  - print_hdrs, print_addr_ip_int