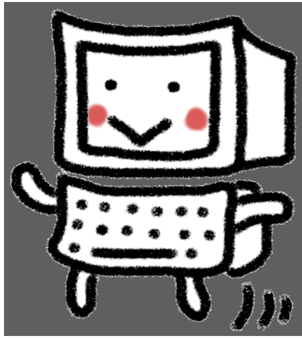


Lab3

Dynamic Routing
and
Network Address Translation

Date :	2021/03/16
Deadline :	2021/03/29 23:59

Outline

- Objective
- Quagga
- Dynamic Routing
- iptables overview
- NAT scenarios
- Lab requirement
- Appendix

Objective

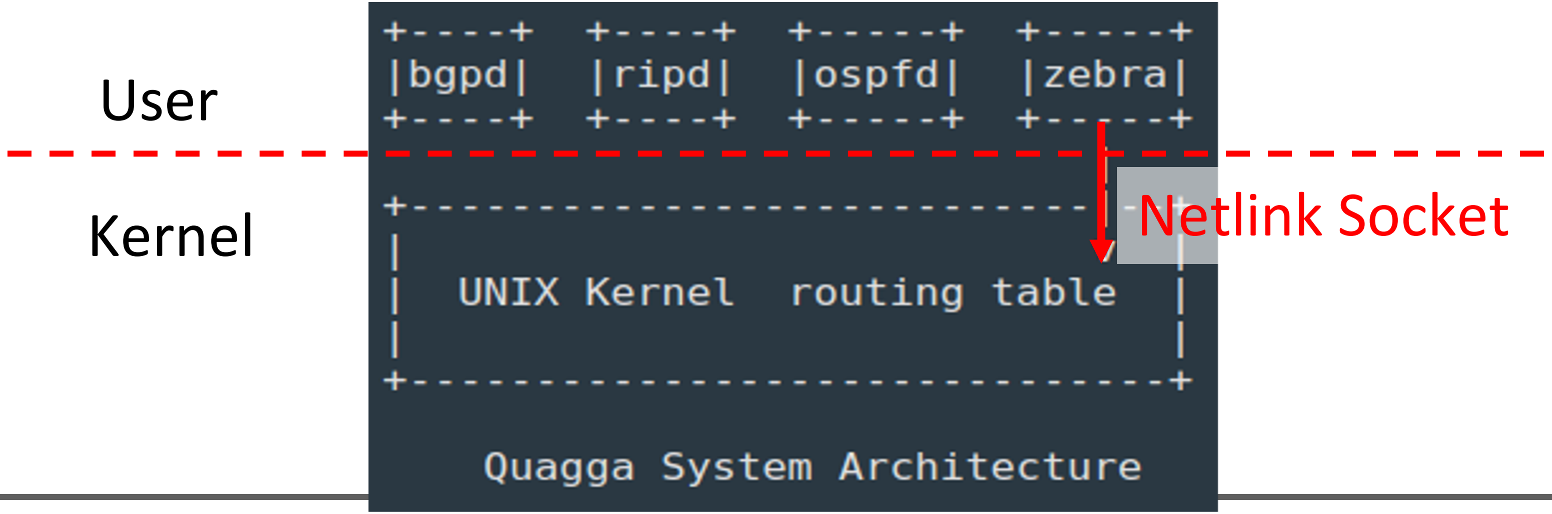
- Dynamic routing configuration
- To learn how Linux kernel handles received packets
- Configure NAT rules on routers with iptables
- Observe packets before/after NAT

Outline

- Objective
- Quagga
  - Introduction
  - Installation
- Dynamic Routing
- iptables overview
- NAT scenarios setup
- Lab requirement
- Appendix

Introduction of Quagga

- Quagga is an open source software that provides routing services
  - Supports common routing protocols: BGP, OSPF, RIP, and IS-IS
  - Consists of a **core daemon Zebra** and separate **routing protocol** daemons
- Routing Protocols (daemons) communicate their best routes to Zebra
- Zebra computes best routes and modifies **kernel routing table** through netlink



Outline

- Objective
- Quagga
  - Introduction
  - Installation
- Dynamic Routing
- iptables overview
- NAT scenarios setup
- Lab requirement
- Appendix

Makefile Dependency Installation

- gawk
  - Patten scanning and processing language

```
bash$ sudo apt install gawk -y
```

- libreadline
  - Readline and history libraries

```
bash$ sudo apt install libreadline7 libreadline-dev -y
```

- pkg-config
  - program used to retrieve information about installed libraries

```
bash$ sudo apt install pkg-config -y
```

Quagga Dependency Installation

- c-ares:
  - Library for asynchronous DNS request
- Download c-ares-1.17.1.tar.gz from e3
- Install c-ares

```
~/Downloads$ tar -xzvf  c-ares-1.17.1.tar.gz            #unzip this package
~/Downloads$ cd c-ares-1.17.1
~/Downloads/c-ares-1.17.1$ sudo ./configure
#check dependency and generate makefile
~/Downloads/c-ares-1.17.1 $ sudo make            #compile c-ares source code
~/Downloads/c-ares-1.17.1 $ sudo make install            #install c-ares
```

Quagga Installation

- Download quagga-1.2.4.tar.gz from e3
- Install Quagga

```
~/Downloads$ tar -xzvf  quagga-1.2.4.tar.gz  #unzip this package
~/Downloads$ cd quagga-1.2.4
~/Downloads/quagga-1.2.4$ sudo ./configure --enable-vtysh --enable-user=root --enable-group=root --enable-vty-group=root
#check dependency and generate makefile with options
~/Downloads/quagga-1.2.4$ sudo make                #compile source code
~/Downloads/quagga-1.2.4$  sudo make install       #install quagga
```

- Copy libzebra to /lib

```
bash $ sudo cp /usr/local/lib/libzebra.so.1   /lib       #copy libzebra to /lib
```

Check Quagga Daemons Version

- Check bgpd version

```
bash$ sudo bgpd -v
```

```
jin@ubuntu:~/Desktop/BGP$ sudo bgpd -v
bgpd version 1.2.4
Copyright 1996-2005 Kunihiro Ishiguro, et al.
configured with:
        --enable-vtysh --enable-user=root --enable-group=root --enable-vty-group
=root
```

- Check zebra version

```
bash$ sudo zebra -v
```

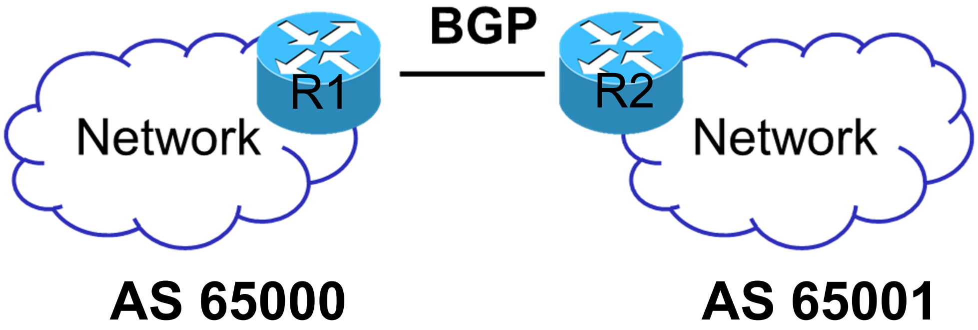
```
jin@ubuntu:~/Desktop/BGP$ sudo zebra -v
zebra version 1.2.4
Copyright 1996-2005 Kunihiro Ishiguro, et al.
configured with:
        --enable-vtysh --enable-user=root --enable-group=root --enable-vty-group
=root
```

Outline

- Objective
- Quagga
- Dynamic Routing
- iptables overview
- NAT scenarios setup
- Lab requirement
- Appendix

Example

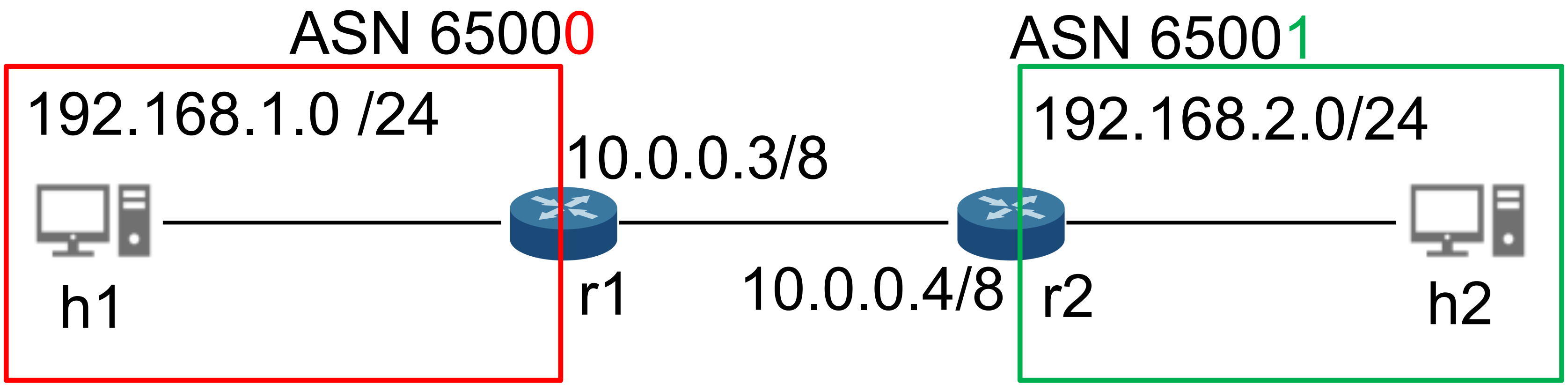
- Create two ASs in mininet
  - AS: Autonomous System



- Routers use BGP to exchange routes
  - BGP: Border Gateway Protocol

Network Configuration

- Both r1 and r2 run BGP and Zebra daemons
  - Each router needs a configuration file for each daemon
- Create configuration files for daemons



Python script for example scenario

- Download example.py from e3
- Create and edit configuration files for daemons on r1 and r2
- Put configuration files in the directory **specified by example.py**
  - E.g., configuration files and example.py in the same directory

```
jin@ubuntu:~/Desktop/exampleScen$ tree
.
├── bgp_r1.conf
├── bgp_r2.conf
├── example.py
└── zebra.conf
```

Run BGP and Zebra daemons on Routers

- Create a directory for pid-files of daemons

```
bash$ sudo mkdir /var/run/quagga/
```

- Python script that runs Zebra and bgpd daemons on r1 and r2

```
r1.cmd('zebra -f ./zebra.conf -d -i /var/run/quagga/zebraR1.pid')
r1.cmd('bgpd -f ./bgp_r1.conf -d -i /var/run/quagga/bgpdR1.pid')
r2.cmd('zebra -f ./zebra.conf -d -i /var/run/quagga/zebraR2.pid')
r2.cmd('bgpd -f ./bgp_r2.conf -d -i /var/run/quagga/bgpdR2.pid')
```

  - -f: specify a config file
  - -d: runs in daemon mode
  - -i: create a pid-file for this daemon

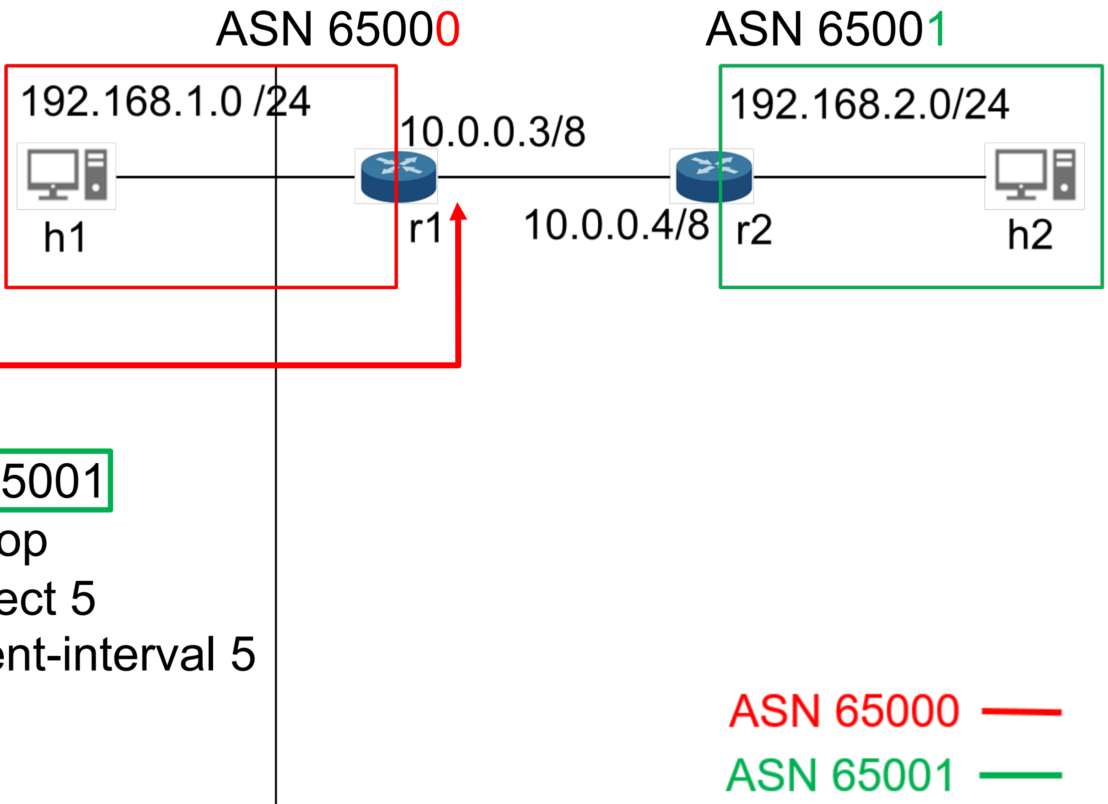
Configuration for Zebra Daemons

- Define hostname and password for Zebra daemon
  - For telnet to Zebra daemon

```
! Configuration for zebra (Note: it is the same for all routers)
!
hostname zebra
password nscap
log stdout
!
```

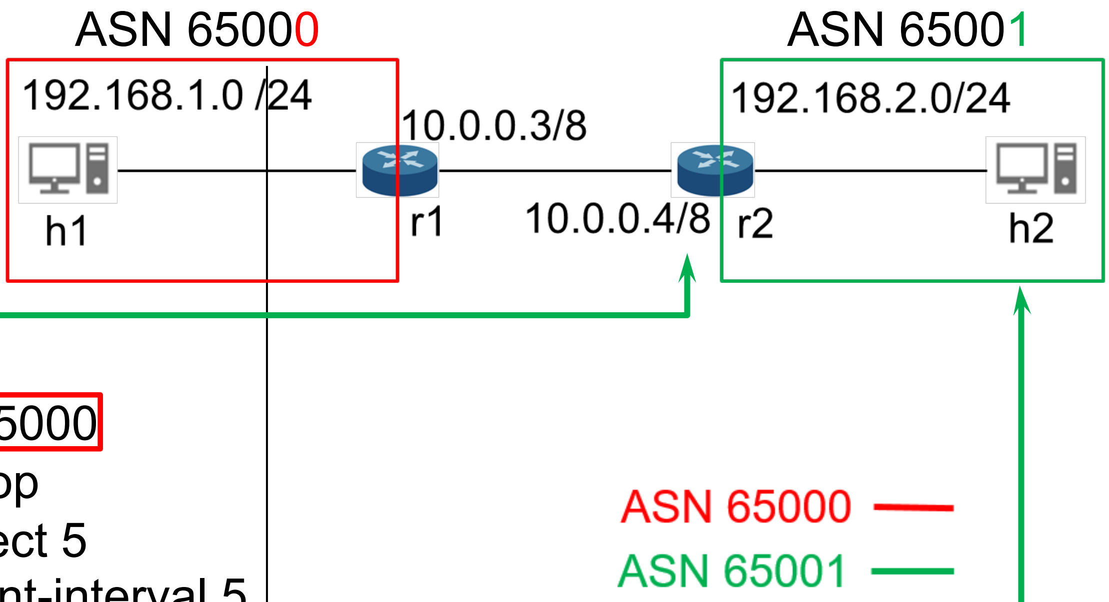
Configuration for BGP Daemon on r1

```
! BGP configuration for r1
!
hostname r1
password nscap
!
router bgp 65000
  bgp router-id 10.0.0.3
  timers bgp 3 9
  neighbor 10.0.0.4 remote-as 65001
  neighbor 10.0.0.4 ebgp-multihop
  neighbor 10.0.0.4 timers connect 5
  neighbor 10.0.0.4 advertisement-interval 5
  network 192.168.1.0/24
!
log stdout
```



Configuration for BGP Daemon on r2

```
! BGP configuration for r2
!
hostname r2
password nscap
!
router bgp 65001
  bgp router-id 10.0.0.4
  timers bgp 3 9
  neighbor 10.0.0.3 remote-as 65000
  neighbor 10.0.0.3 ebgp-multihop
  neighbor 10.0.0.3 timers connect 5
  neighbor 10.0.0.3 advertisement-interval 5
  network 192.168.2.0/24
!
log stdout
```



Check Route in routing table

- Check Route

mininet> r1 route

```
mininet> r1 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 r1-eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 r1-eth1
192.168.2.0     10.0.0.4        255.255.255.0   UG    20     0        0 r1-eth0
```

mininet> r2 route

```
mininet> r2 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 r2-eth0
192.168.1.0     10.0.0.3        255.255.255.0   UG    20     0        0 r2-eth0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 r2-eth1
```

Check Route in Zebra

- Telnet r1 zebra daemons (on port 2601)

```
mininet> r1 xterm & #invoke a terminal for r1
r1> telnet 127.0.0.1 2601
```

```
User Access Verification

Password:
zebra>
```

Type in password defined in zebra.conf

- Show bgp route in r1

```
zebra> show ip route bgp
```

```
zebra> show ip route bgp
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, P - PIM, A - Babel, N - NHRP,
       > - selected route, * - FIB route

B>* 192.168.2.0/24 [20/0] via 10.0.0.4, r1-eth0, 00:18:25
zebra>
```

Check route in bgpd

- Telnet r1 bgpd daemons (om port 2605)

```
r1> telnet 127.0.0.1 2605
```

```
User Access Verification

Password:
r1>
```

Type in Password defined in bgp_r1.conf

- Show r1 bgp neighbor summary

```
zebra> show ip bgp summary
```

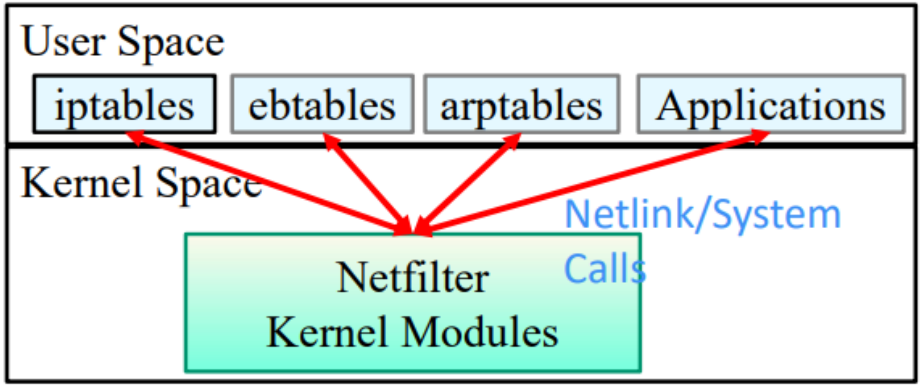
```
r1> show ip bgp summary
BGP router identifier 10.0.0.3, local AS number 65000
RIB entries 3, using 336 bytes of memory
Peers 1, using 9088 bytes of memory

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.0.0.4        4 65001     426     429        0    0    0 00:21:12        1

Total number of neighbors 1

Total num. Established sessions 1
Total num. of routes received     1
```

Outline

- Objective
- Quagga
- Dynamic Routing
- iptables
  - Overview
  - Basic usage
- NAT scenarios
- Lab requirement
- Appendix

iptables overview

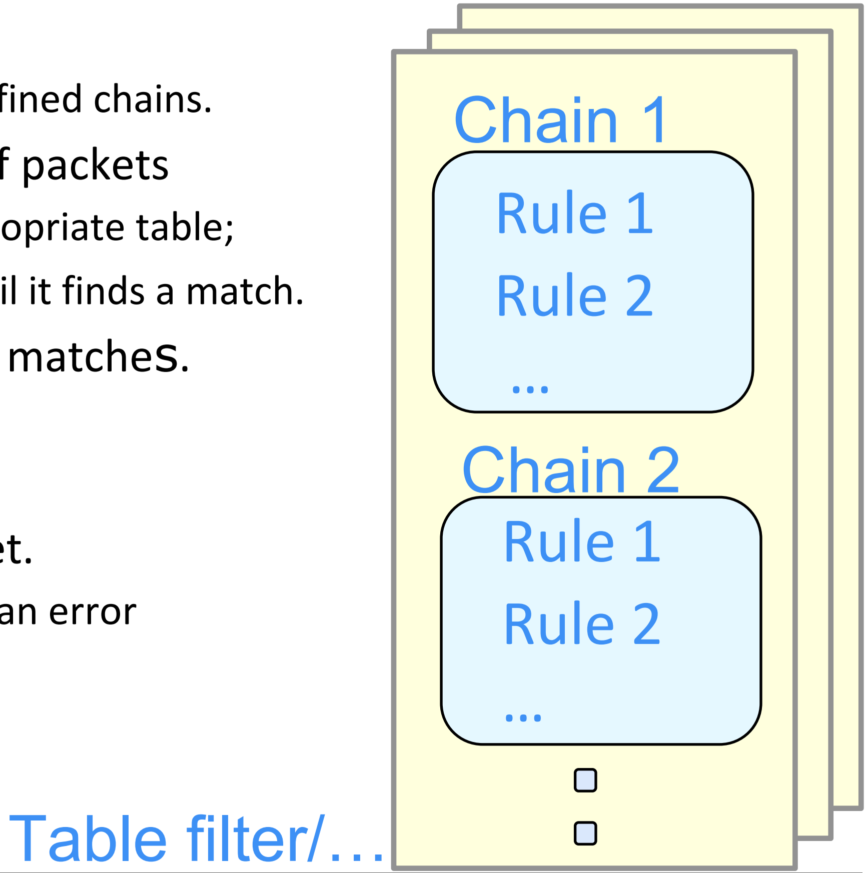
- A user-space utility program for configuring IP packet filter rules of Linux kernel firewall
  - Linux kernel firewall implemented as different **Netfilter modules**.
- Netfilter: a framework inside the Linux kernel that allows kernel modules to register **callback** functions at different locations (hooks) of the Linux network stack.
  - – A **registered callback** function is called back for every packet that traverses the respective hook within the Linux network stack.



Reference:https://en.wikipedia.org/wiki/Netfilter

Component of iptables

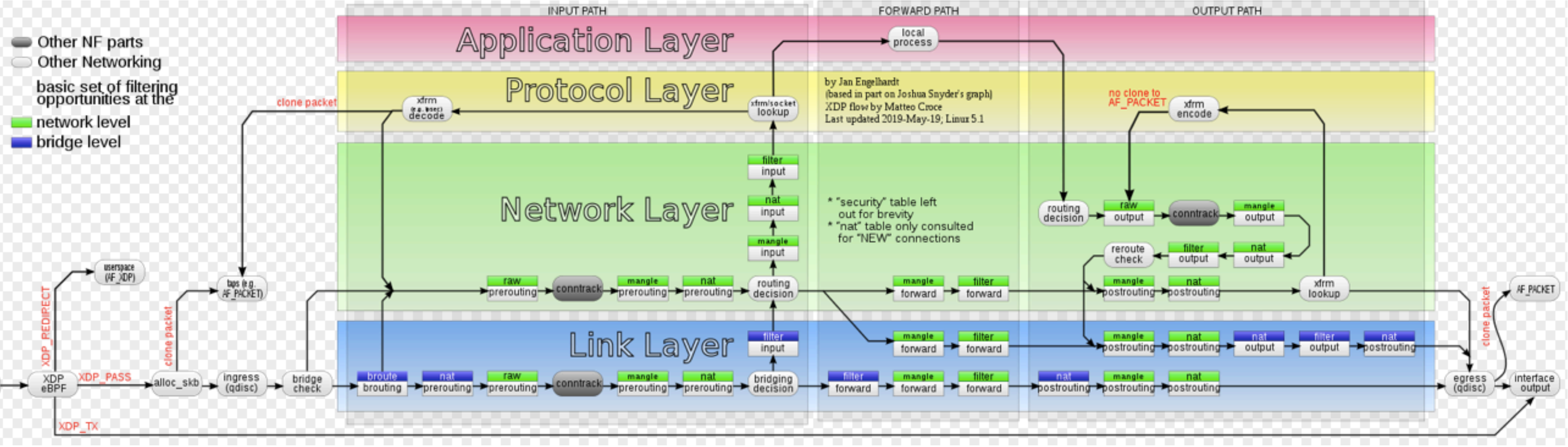
- **Tables:** files that join similar actions.
  - Contains a number of built-in chains or user-defined chains.
- **Chains:** a list of **rules** which can match a set of packets
  - When receives a packet, iptables finds the appropriate table;
  - Then apply the chain of **rules** on the packet until it finds a match.
- **Rules:** specifies what to do with a packet that matches.
  - can block one type of packet, or
  - forward another type of packet.
- **Targets:** a decision of what to do with a packet.
  - Typically, Accept, Drop, or Reject (which sends an error back to the sender)



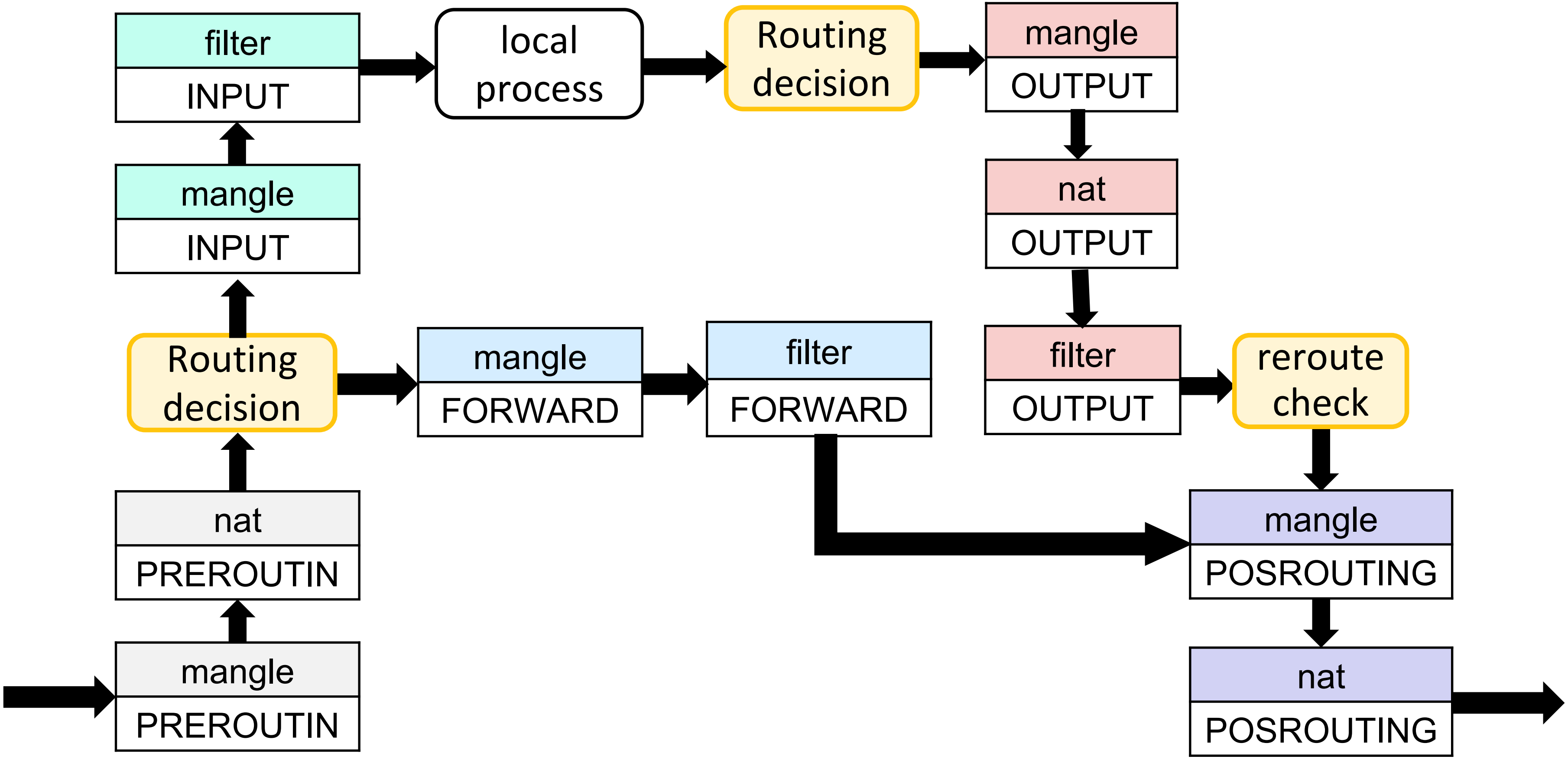
Tables and Chains

- Tables
  - filter: packet filtering, default table
  - nat: NAT operation
  - mangle: add tag on packet (for QoS or load distribution)
  - raw: mainly for exemptions from connection tracking
- **Five Predefined Chains** (mapping to the five available Netfilter hooks)
  - **PREROUTING**: for packets before a routing decision is made.
  - **INPUT**: for packets destined to local sockets
  - **FORWARD**: for packets being routed through the machine.
  - **OUTPUT**: for locally-generated packets.
  - **POSTROUTING**: for packets about to go out after Routing decision has been made.

Netfilter Packet Flow



Simplified Netfilter Network Layer Packet Flow



Outline

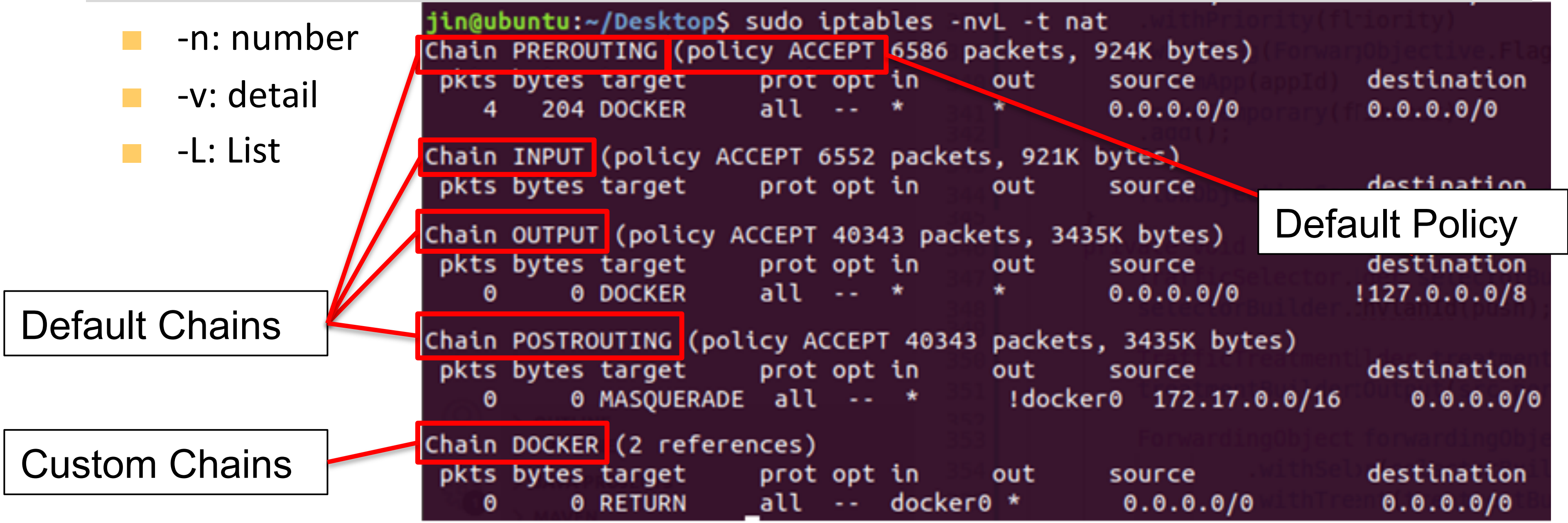
- Objective
- Quagga
- Dynamic Routing
- iptables
  - Overview
  - Basic Operations
- NAT scenarios
- Lab requirement
- Appendix

iptables – Chains and Rules

- Show all chains and rules in a table

```
bash$ sudo iptables -nvL -t [table name]
```

  - -n: number
  - -v: detail
  - -L: List



```
jin@ubuntu:~/Desktop$ sudo iptables -nvL -t nat
Chain PREROUTING (policy ACCEPT 6586 packets, 924K bytes)
 pkts bytes target     prot opt in     out     source               destination
    4   204 DOCKER     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain INPUT (policy ACCEPT 6552 packets, 921K bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 40343 packets, 3435K bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DOCKER     all  --  *      *       0.0.0.0/0           !127.0.0.0/8

Chain POSTROUTING (policy ACCEPT 40343 packets, 3435K bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MASQUERADE  all  --  *      !docker0  172.17.0.0/16       0.0.0.0/0

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 RETURN     all  --  docker0 *       0.0.0.0/0            0.0.0.0/0
```

iptables – Matching Fields and Actions

- Matching fields

```
pkts bytes target   prot opt in   out   source      destination
   0     0 DOCKER   all  --  *    *     0.0.0.0/0   0.0.0.0/0    ADDRTYPE match dst-type LOCAL
```

- Actions
  - Target indicate
    - Actions of rules like DNAT/ SNAT/ MASQUERADE
    - Jump to another chains

```
Chain POSTROUTING (policy ACCEPT 38271 packets, 3243K bytes)
 pkts bytes target     prot opt in  out       source          destination
    0     0 MASQUERADE  all  --  *   !docker0  172.17.0.0/16   0.0.0.0/0
```

```
pkts bytes target  prot opt in  out  source      destination
   0     0 DOCKER  all  --  *   *    0.0.0.0/0   0.0.0.0/0    ADDRTYPE match dst-type LOCAL
```

Iptables – Adding Rules

- Add rules to a chain of a table

```
bash$ sudo iptables -t [table] -A [chain] [match field] -j [Actions]
```

  - -A: append
  - -I: insert

- Example:

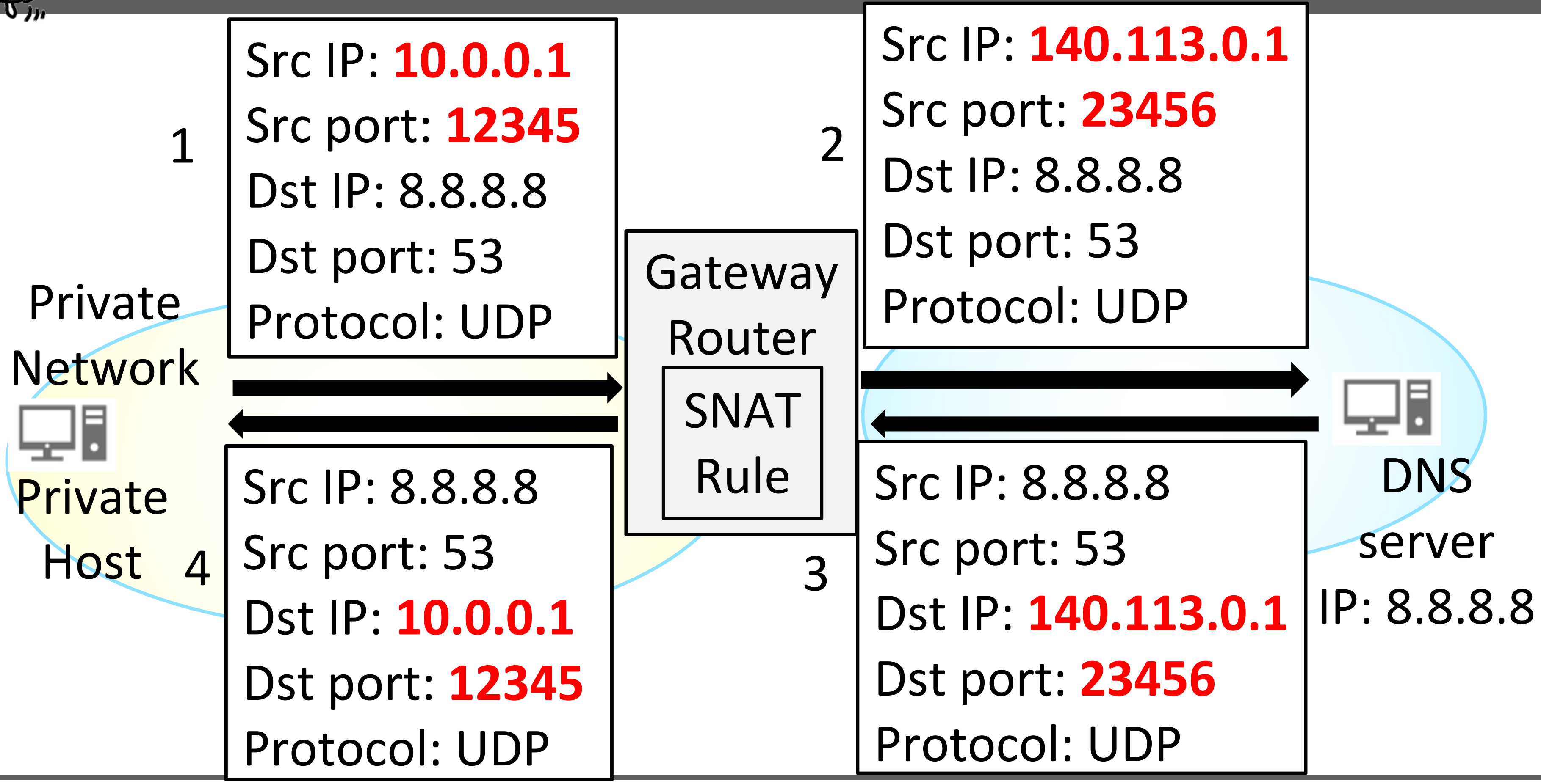
```
bash$ sudo iptables -t nat -A POSTROUTING -s 172.20.0.0/16 -d
172.87.0.0/16 -o eth2 -j SNAT –to-source 140.113.194.239
```

  - Append a SNAT rules chain POSTROUTING of NAT table if packet matched following fields
    - source address is in 172.20.0.0/16
    - destination address is in 172.87.0.0/16
    - the output interface is eth2

Outline

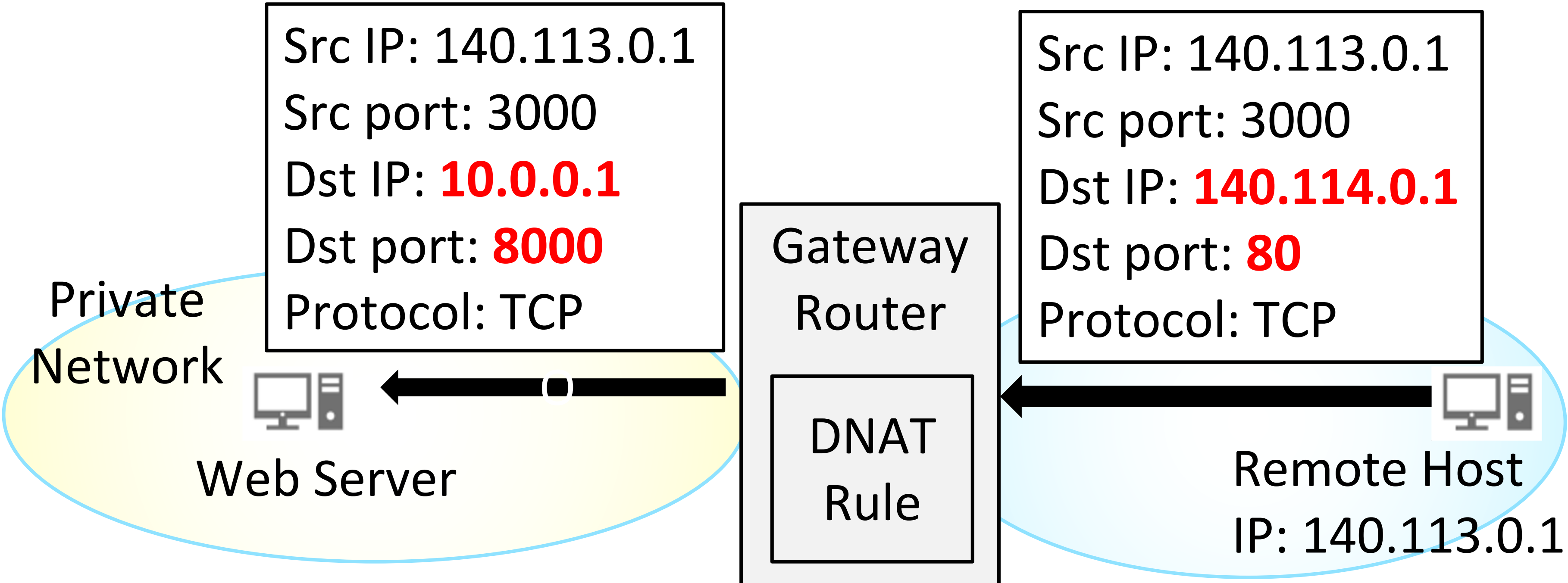
- Objective
- Quagga
- Dynamic Routing
- iptables overview
- NAT scenarios
  - Source NAT
  - Destination NAT
- Lab requirement
- Appendix

Source NAT



1
Src IP: 10.0.0.1
Src port: 12345
Dst IP: 8.8.8.8
Dst port: 53
Protocol: UDP
2
Src IP: 140.113.0.1
Src port: 23456
Dst IP: 8.8.8.8
Dst port: 53
Protocol: UDP
Private Network
Private Host
Gateway Router
SNAT Rule
DNS server
IP: 8.8.8.8
4
Src IP: 8.8.8.8
Src port: 53
Dst IP: 10.0.0.1
Dst port: 12345
Protocol: UDP
3
Src IP: 8.8.8.8
Src port: 53
Dst IP: 140.113.0.1
Dst port: 23456
Protocol: UDP

Destination NAT

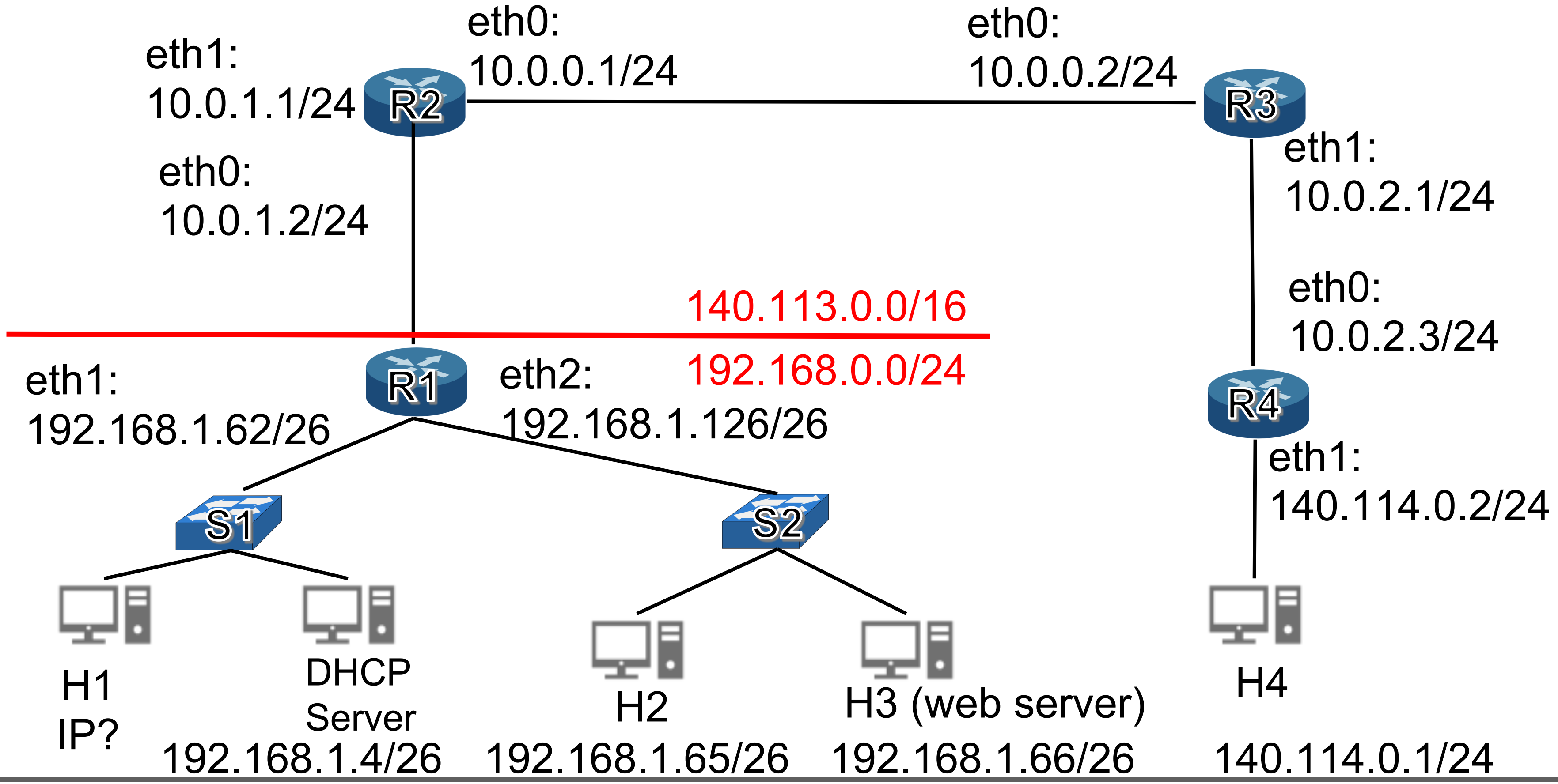


Internal IP:Port	External IP:Port
10.0.0.1:8000	140.114.0.1:80

Outline

- Objective
- Quagga
- Dynamic Routing
- iptables overview
- NAT scenarios
- Lab requirement
  - Topology
  - Part1: dynamic routing
  - Part2: configure NAT rules
  - About submit

Lab Topology



Part1: Topology Creation and Configuration

- Download topology.py from e3
- Edit bgp and zebra configuration files for routers
  - bgp_<router>.conf
  - zebra.conf
- Execute topology.py

```
bash$ sudo python topology.py
```

- Routing tables before enabling BGP

```
mininet> r1 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 r1-eth0
192.168.1.0     0.0.0.0         255.255.255.192 U     0      0        0 r1-eth1
192.168.1.64    0.0.0.0         255.255.255.192 U     0      0        0 r1-eth2
mininet> r2 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 r2-eth0
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 r2-eth1
mininet> r3 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 r3-eth0
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 r3-eth1
mininet> r4 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 r4-eth0
140.114.0.0     0.0.0.0         255.255.255.0   U     0      0        0 r4-eth1
mininet>
```

Part1: BGP packet cpaturing

- Run wireshark on node r2 and r3 to capture BGP packets

```
mininet> r2 wireshark & #listen at r2-eth0
mininet> r3 wireshark & #listen at r3-eth0
mininet> r3 wireshark & #listen at r3-eth1
```

- Run BGP and Zebra daemons on every router nodes ([r1-r4])

```
mininet> r1 zebra -f ./configs/zebra.conf -d -i /var/run/quagga/zebra1.pid
mininet> r1 bgpd -f ./configs/bgp_r1.conf -d -i /var/run/quagga/bgpd1.pid
```

Part1: Check routing tables after enabling BGP

- Check routing tables again

```
mininet> r1 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 r1-eth0
140.114.0.0     10.0.1.1        255.255.0.0     UG    20     0        0 r1-eth0
192.168.1.0     0.0.0.0         255.255.255.192 U     0      0        0 r1-eth1
192.168.1.64    0.0.0.0         255.255.255.192 U     0      0        0 r1-eth2
mininet> r2 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 r2-eth0
10.0.1.0        0.0.0.0         255.255.255.0   U     0      0        0 r2-eth1
140.113.0.0     10.0.1.2        255.255.0.0     UG    20     0        0 r2-eth1
140.114.0.0     10.0.0.2        255.255.0.0     UG    20     0        0 r2-eth0
mininet> r3 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 r3-eth0
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 r3-eth1
140.113.0.0     10.0.0.1        255.255.0.0     UG    20     0        0 r3-eth0
140.114.0.0     10.0.2.3        255.255.0.0     UG    20     0        0 r3-eth1
mininet> r4 route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.2.0        0.0.0.0         255.255.255.0   U     0      0        0 r4-eth0
140.113.0.0     10.0.2.1        255.255.0.0     UG    20     0        0 r4-eth0
140.114.0.0     0.0.0.0         255.255.255.0   U     0      0        0 r4-eth1
```

Part1: Question

1. Take routing tables screenshot before/after on [r1-r4] (10%)
2. Telnet zebra and bgpd daemons of [r1-r4] and take screenshots of routes in zebra and bgpd daemons. (10%)
3. Capture BGP packets from wireshark and take screenshot to verify your answer for the following questions (20%)

3-1. Show BGP packets (OPEN, UPDATE, KEEP ALIVE) exchanged by r2 and r3

3-2. What will happen to the routing table if you set r4-eth0 down?

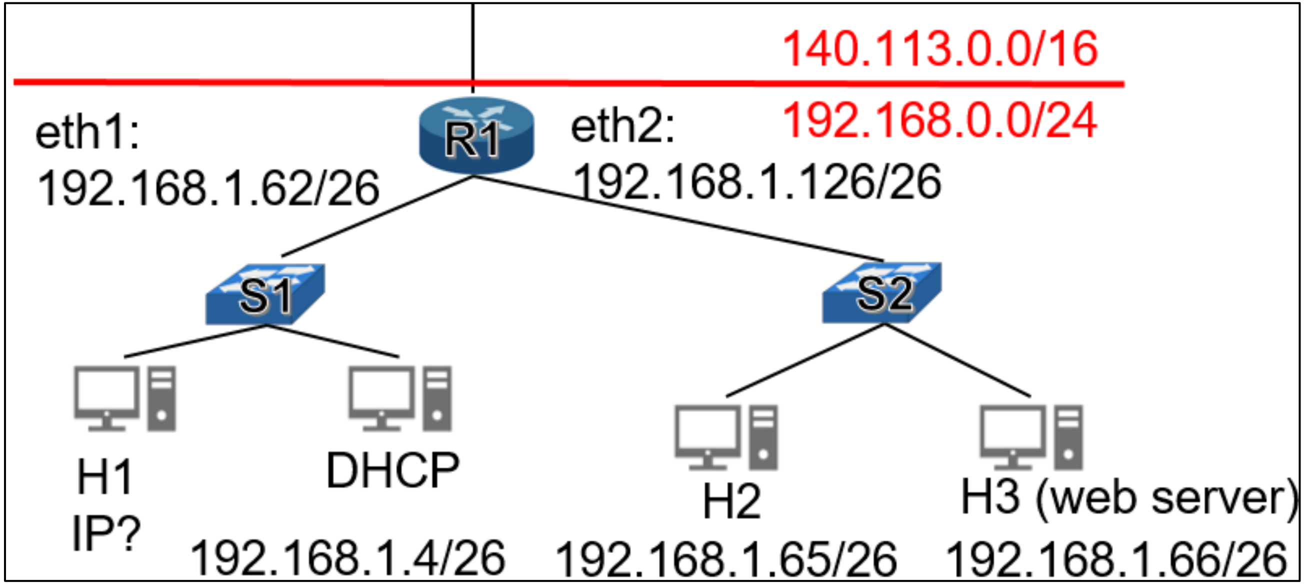
```
mininet> r4 ip link set r4-eth0 down
mininet> [r1-r4] route #check routing tables
```

3-3. How does r3 know r4 is unreachable? Explain how

3-4. How does r2 know r4 is unreachable? Explain how

Part2: Source NAT

- Configure r1 to perform Source NAT with iptables rules
  - Use 140.113.0.30 for network 192.168.1.0/26
  - Use 140.113.0.40 for network 192.168.1.64/26



Part2: Destination NAT

- Run Http server at node h3

```
mininet> h3 python -m SimpleHTTPServer &
```

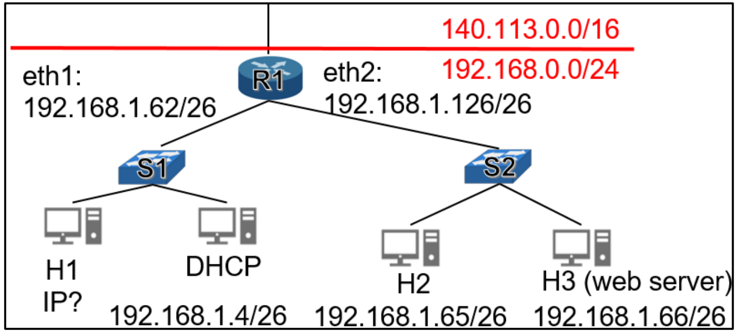
  - SimpleHTTPServer

    Python build-in script, listen at port 8000 by default

- Configure r1 to perform Destination NAT with iptables rules
  - Mapping 140.113.0.40:80 to 192.168.1.66:8000
- h4 sends Http GET to h3

```
mininet> h4 curl 140.113.0.40:80
```

  - 1. Take screenshot of curl result (10%)



Part2: Question

2. Check reachability and take screenshot (10%)

```
mininet> h1 ping h4 -c 1
mininet> h2 ping h4 -c 1
mininet> h3 ping h4 -c 1
```

3. Run wireshark on r1 to take screenshot of input/output packet (10%)
   - Explain the difference of packet headers

```
mininet> r1 wireshark &    #listen at r1-eth0
mininet> r1 wireshark &    #listen at r1-eth1
mininet> r1 wireshark &    #listen at r1-eth2
mininet> h1 ping h4 -c 1
mininet> h2 ping h4 -c 1
```

Report Submission

- Files
  - <StudentID>_topo.py (10%)
    - With NAT configuration
  - bgp_<router>.conf and zebra.conf (20%)
  - A Report: lab3_<studentID>.pdf (70%)
    - Screenshot and answers
- Submission
  - Zip all files into a zip file
    - Name: lab3_<studentID>.zip
    - Wrong filename or format will deduct scores (-5%)

Q&A

THANK
YOU

Appendix

- iptables man page
  - https://linux.die.net/man/8/iptables
- Quagga
  - https://www.quagga.net/docs/quagga.pdf