



Lab3

*Dynamic Routing
and*

Network Address Translation

Deadline: 2022/03/28 (Mon) 23:59



Outline

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



Objective

- How Linux kernel handle incoming packet
- Observe packet before/after NAT
- Configure NAT rules on routers with iptables
 - Source NAT
 - Destination NAT (Port forwarding)
- Docker setup
- Dynamic routing configuration



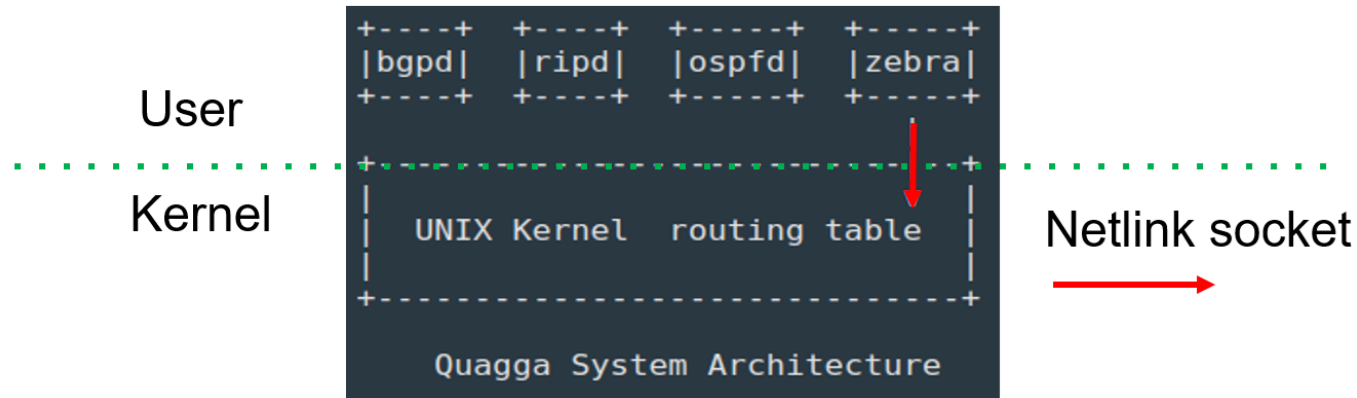
Outline

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



Quagga

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





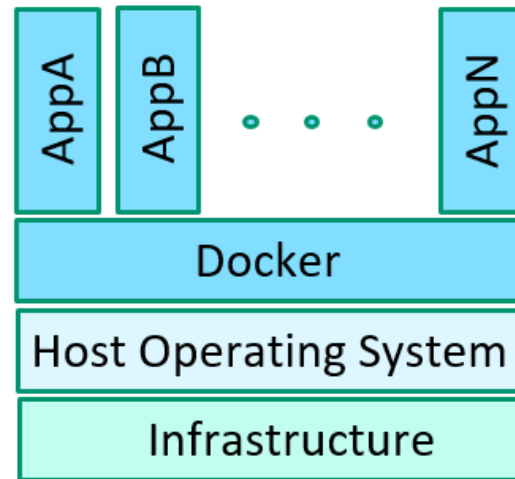
Outline

- Objective
- Quagga
- Docker
 - Docker Object and Docker Registry
 - Docker Installation
 - Docker basic usage
- Dynamic Routing
- iptables overview
- NAT scenarios setup
- Lab requirement
- Appendix



Docker Objects

- Docker images
 - An image is a read-only template with instructions for creating a Docker container
 - An image could be based on another image, with some additional customization
- Docker containers
 - A runnable instance of an image





Docker Registry

- A database of images where users can pull or push images
 - Public (Docker Hub)
 - <https://hub.docker.com/>
 - Private (e.g. your computer)
 - Default path: `/var/docker/lib`



Outline

- Objective
- Quagga
- Docker
 - Docker Object and Docker Registry
 - Docker Installation
 - Docker basic usage
- Dynamic Routing
- iptables overview
- NAT scenarios setup
- Lab requirement
- Appendix



Docker Installation

- Update apt

```
bash$ sudo apt-get update
```

- Install curl for data transfer

```
bash$ sudo apt-get install -y curl
```

- Retrieve Docker installation script and install Docker

```
bash$ sudo curl -ssl https://get.docker.com | sh
```



Docker Permission Setup

- By default, Docker daemon runs as the root user
 - Need root permission (sudo) to run Docker commands
- To run docker commands without root permission
 1. Create a Unix group called docker
 2. Add users to docker group

```
bash$ sudo groupadd docker # add a docker group
```

```
bash$ sudo usermod -aG docker $USER # add $USER into docker group
```

- -a: append the user to the supplemental Groups
- -G: new list of supplemental Groups

3. Log out and log back / Reboot your computer



Outline

- Objective
- Quagga
- Docker
 - Docker object and Docker registry
 - Docker installation
 - Docker basic usage
 - Pull image
 - Build image
 - Docker run
 - Docker exec
 - Docker network



Pull Images from Docker Hub Registry

- Usage

```
bash$ sudo docker pull [NAME]:[TAG]
```

- Example

```
bash$ sudo docker pull ubuntu:18.04
```

- List images

```
bash$ sudo docker images
```

```
jln@ubuntu:~$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
ubuntu	18.04	c090eaba6b94	2 weeks ago
63.3MB			



Outline

- Objective
- Quagga
- Docker
 - Docker object and Docker registry
 - Docker installation
 - Docker basic usage
 - Pull image
 - Build image
 - Docker run
 - Docker exec
 - Docker network



Create Dockerfile

- Create a file named **Dockerfile**
- **Dockerfile**: a text file contains all the commands for Docker to build a container image automatically

– Including Base image

- A base image command
- Other custom commands

Custom commands for building custom image

- Example **Dockerfile**

```
1 FROM ubuntu:18.04
2
3 MAINTAINER jin
4
5 RUN apt-get update
6
7 RUN apt-get install iptables -y
8 RUN apt-get install iputils-ping -y
9 RUN apt-get install net-tools -y
10 RUN apt-get install iproute2 -y
11 RUN apt-get install tcpdump -y
12 RUN apt-get install vim -y
13 RUN apt-get install sudo -y
14 RUN apt-get install git -y
15 RUN apt-get install isc-dhcp-server -y
16 RUN apt-get install isc-dhcp-client -y
17 RUN apt-get install mininet -y
```



Build Docker Image

- Build image

```
bash$ docker build -t [image_name] [path of dockerfile]
```

- Execution steps

1. Load Dockerfile
2. Pull and load base image
3. Run custom commands
4. Save the image to local docker registry



Example: Build Docker Image

- Build a custom image named test

```
jln@ubuntu:~/Desktop$ docker build -t test .
```

- Default filename: Dockerfile

- Show docker images

```
jln@ubuntu:~/Desktop$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
test	latest	94fa41d40498	11 minutes ago
ubuntu	18.04	c090eaba6b94	2 weeks ago



Outline

- Objective
- Quagga
- Docker
 - Docker object and Docker registry
 - Docker installation
 - Docker basic usage
 - Pull image
 - Build image
 - Docker run
 - Docker exec
 - Docker network



Docker run

- Run a command in a new container
- Usage

```
bash$ sudo docker run [OPTIONS] [IMAGE:TAG] [COMMAND] [ARG]
```

Create and Run a new container

A Command runs in the new container (Optional)

- **Create and Start** a container “sample”

```
bash$ sudo docker run -d -it --name sample Ubuntu:18.04
```

- -d: Detached (like a daemon in background)
- -it: interactive process (like a shell)
- --name: Assign a name to the container



List Docker Containers

- Command to list Docker containers

```
bash$ sudo docker ps -a
```

- “--all”, “-a”: Show all containers

```
jln@ubuntu:~/Desktop$ sudo docker run -d -it --name sample ubuntu:18.04
[sudo] password for jln:
be95d7141f15663ff624d226f08c95a99f7946467ac36b24329cfe7cdf1bf517
jln@ubuntu:~/Desktop$ sudo docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
be95d7141f15	ubuntu:18.04	"/bin/bash"	16 seconds ago
Up 13 seconds		sample	



Outline

- Objective
- Quagga
- Docker
 - Docker object and Docker registry
 - Docker installation
 - Docker basic usage
 - Pull image
 - Build image
 - Docker run
 - Docker exec
 - Docker network



Docker exec

- Execute a command in a **running** container
- Usage

```
bash$ sudo docker exec [OPTIONS] [CONTAINER] [COMMAND]
```

- E.g., Execute *bash* command in a **running** container “sample”

```
jln@ubuntu:~/Desktop$ sudo docker exec -it sample bash
root@be95d7141f15:/#
```



Outline

- Objective
- Quagga
- Docker
 - Docker object and Docker registry
 - Docker installation
 - Docker basic usage
 - Pull image
 - Build image
 - Docker run
 - Docker exec
 - Docker network



Docker network – Create

- Usage

```
bash$ sudo docker network create [OPTIONS] [Network name]
```

[OPTIONS]: Choose the network mode, default mode is *bridge*

- E.g. create a network named **testbr**

```
bash$ sudo docker network create testbr
```

testbr

- List existing docker networks

```
bash$ sudo docker network ls
```

```
jln@ubuntu:~/Desktop$ sudo docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
d8d101d5687d	bridge	bridge	local
1529c9f8db21	host	host	local
d99e273e7e4c	none	null	local
b85856850a7e	testbr	bridge	local



Docker network – Connect

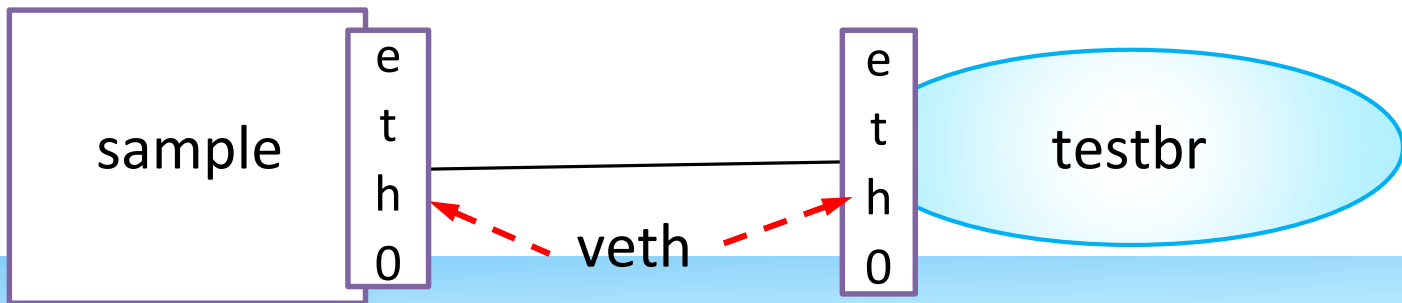
- Connect a container to a network
- Usage

```
bash$ sudo docker network connect [NETWORK] [CONTAINER]
```

- E.g. connect **sample** to **testbr**

```
bash$ sudo docker network connect testbr sample
```

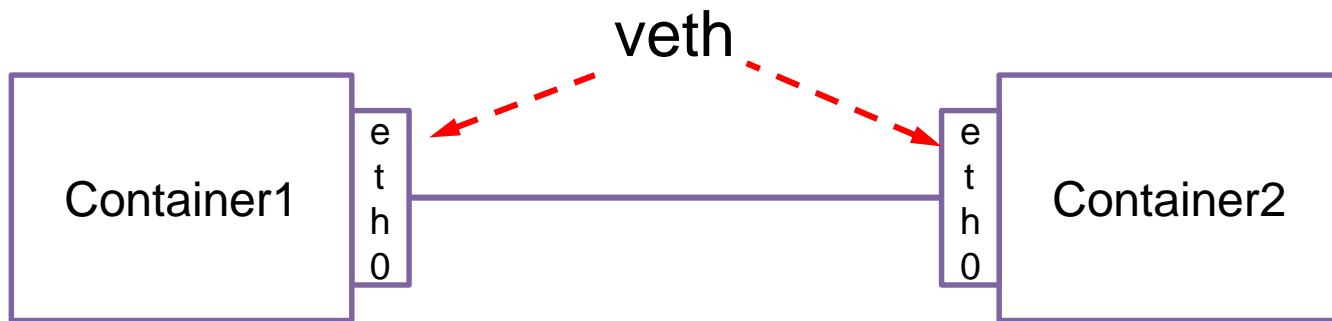
- Docker will create a pair of virtual interfaces (next slide)
 - One for the **testbr** and the other for the **sample** container
- Assigns an IP address to the interface of





Virtual Ethernet Device

- veth: Virtual Ethernet device
 - A local Ethernet tunnel
 - Normally, created in pairs
- Containers could be created without network then manually set veth pairs to connect containers





Create veth Pair without Docker Network

- Create two contains without network

```
bash$ docker run -it --cap-add=NET_ADMIN --name left --net=none --privileged test
```

```
bash$ docker run -it --cap-add=NET_ADMIN --name right --net=none --privileged test
```

- `--cap-add=NET_ADMIN`: Add Linux capabilities to modify network interfaces
- `--privileged`: Give extended privilege to this container
- **test**: custom image

- Create veth pair

```
bash$ sudo ip link add leftVeth type veth peer name rightVeth
```



veth Pair connect to Container

- Set veth pairs into containers **left** and **right**

```
bash$ sudo ip link set leftVeth netns $(sudo docker inspect -f '{{.State.Pid}}' left)
```

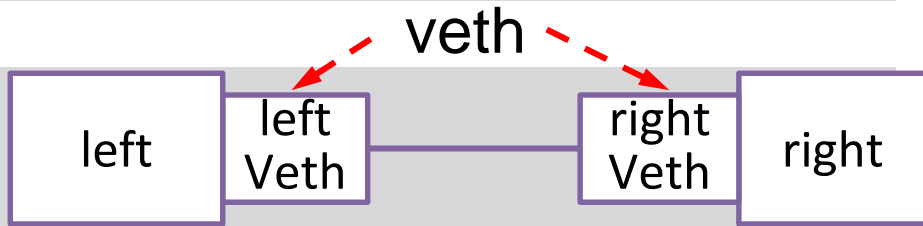
```
bash$ sudo ip link set rightVeth netns $(sudo docker inspect -f '{{.State.Pid}}' right)
```

- Set veth pair up

```
bash$ ip link set leftVeth up #left
```

```
bash$ ip link set rightVeth up #right
```

```
bash$ ifconfig #left
```



```
root@1c27c6b58ca7:/# ip link set leftVeth up
root@1c27c6b58ca7:/# ifconfig
leftVeth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    ether 96:43:84:b0:5a:e7  txqueuelen 1000  (Ethernet)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```



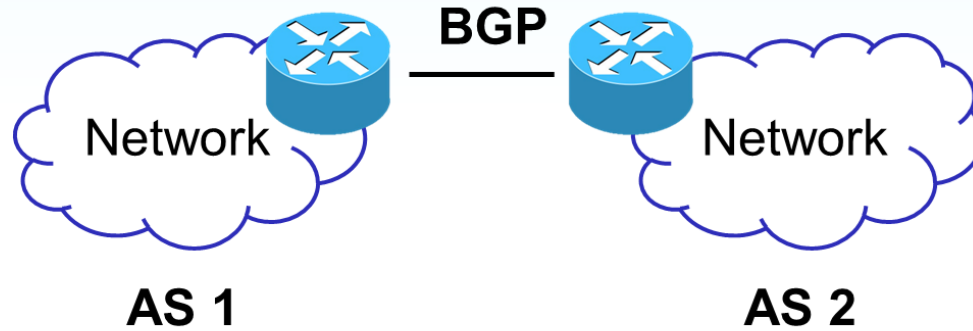
Outline

- Objective
- Quagga
- Docker
- **Dynamic Routing**
 - **Example Scenario**
 - Setup example
- iptables overview
- NAT scenarios setup
- Lab requirement
- Appendix



Example Scenario

- Interconnection of two networks



- BGP: Border Gateway Protocol
 - AS: Autonomous System



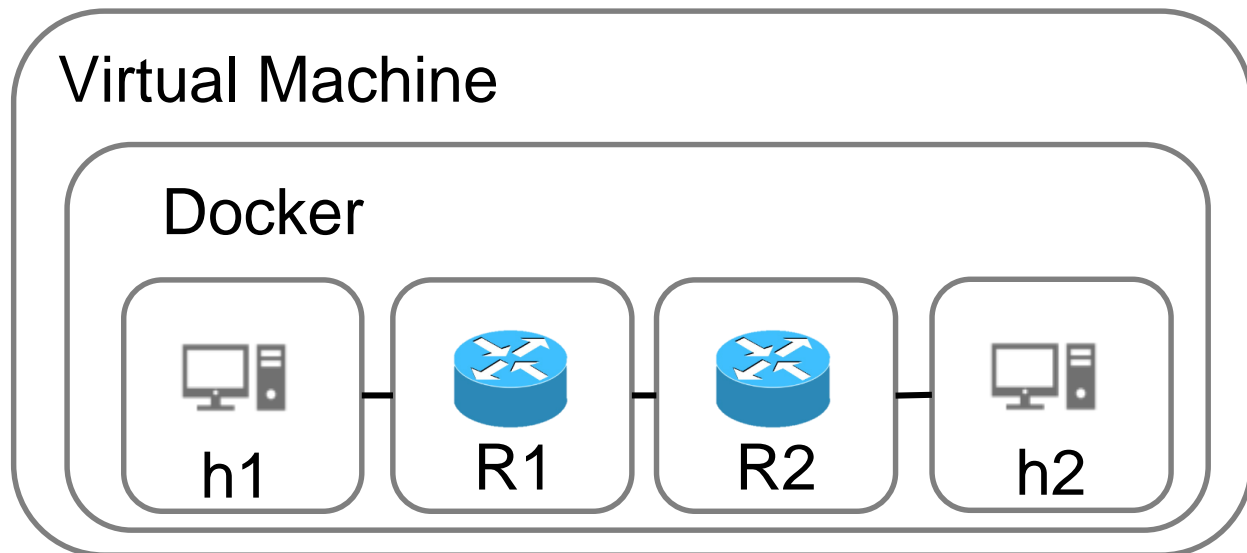
Outline

- Objective
- Quagga
- Docker
- **Dynamic Routing**
 - Example Scenario
 - **Setup example scenario**
- iptables overview
- NAT scenarios setup
- Lab requirement
- Appendix



Steps to Setup Example Scenario

1. Create Containers
2. Set up Container Networks
3. Configure Host Gateways
4. Setup Routers





Step 1 – Create Containers (1/2)

- We use Ubuntu 16.04 for all hosts and routers
- Create a Container with Ubuntu as OS

```
bash$ sudo docker run --privileged --cap-add NET_ADMIN \  
--cap-add NET_BROADCAST -d -it \  
--name <ContainerName> ubuntu:16.04
```

- NET_BROADCAST: Make socket broadcast, and listen to multicasts.



Step 1. Create Containers – example

- Create container for a host h1 (h2)

```
bash$ sudo docker run --privileged --cap-add NET_ADMIN \  
--cap-add NET_BROADCAST -d -it \  
--name h1 ubuntu:16.04
```

- Create container for a virtual router R1 (R2)

```
bash$ sudo docker run --privileged --cap-add NET_ADMIN \  
--cap-add NET_BROADCAST -d -it \  
--name R1 ubuntu:16.04
```



h1



R1



Step 2 – Setup Container Networks (1/3)

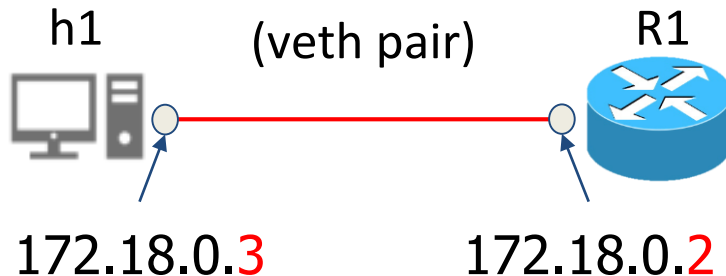
- Create a veth pair

veth

```
bash$ sudo ip link add h1R1veth type veth peer name R1h1veth
```

- Connect containers h1 and R1 with veth pair

```
bash$ sudo ip link set R1h1veth netns $(docker inspect -f {{.State.Pid}} R1)
bash$ sudo ip link set h1R1veth netns $(docker inspect -f {{.State.Pid}} h1)
```





Step 2 – Setup Container Networks (2/3)

- Set IP addresses of network interfaces

```
bash$ docker exec h1 ip addr add 172.18.0.3 dev h1R1veth
bash$ docker exec R1 ip addr add 172.18.0.2 dev R1h1veth
bash$ docker exec h1 ip link set h1R1veth up
bash$ docker exec R1 ip link set R1h1veth up
```



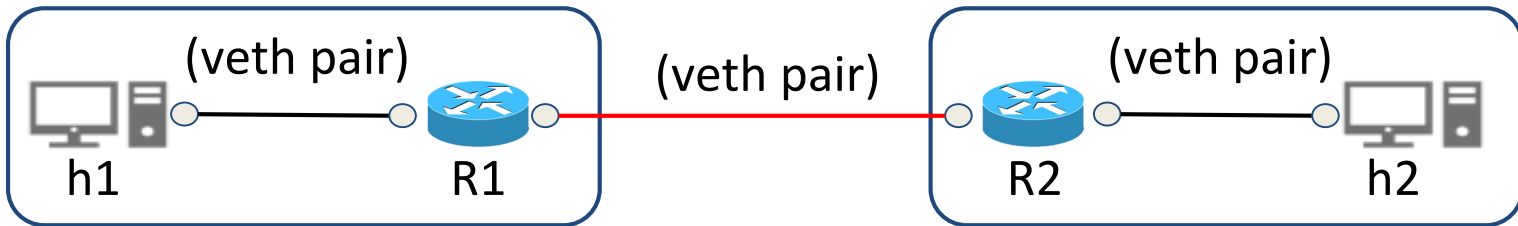
Step 2 – Setup Container Networks (3/3)

- Connect two domains
 - Create inter domain link

```
bash$ sudo ip link add R1R2veth type peer name R2R1veth
```

- Connect containers R1 and R2 to veth pair

```
bash$ sudo ip link set R1R2veth netns $(docker inspect -f {{.State.Pid}} R1)
bash$ sudo ip link set R2R1veth netns $(docker inspect -f {{.State.Pid}} R2)
```





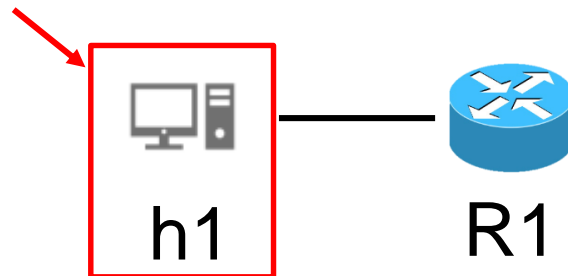
Step3 – Configure Host Gateways (1/2)

- Run bash on h1 (h2)

```
bash$ sudo docker exec -it h1 bash
```

- Install net-tools and iproute2 on h1 (h2)

```
/# apt-get update  
/# apt-get install -y net-tools  
/# apt-get install -y iproute2
```





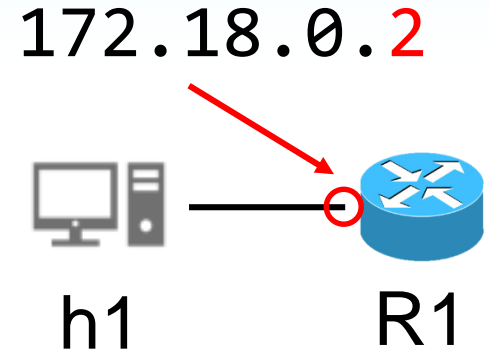
Step3 – Configure Host Gateways (2/2)

- Set R1 (R2) as default gateway of h1 (h2)

```
/# ip route del default  
/# ip route add default via 172.18.0.2
```

- Check route in h1 (h2)

```
/# ip route show
```



```
root@f4ac43b5f92a:/# ip r  
default via 172.18.0.2 dev h1R1veth  
172.17.0.0/16 dev eth0 proto kernel scope link src 172.17.0.2  
172.18.0.0/16 dev h1R1veth proto kernel scope link src 172.18.0.1
```



Step 4 – Setup Routers (1/6)

4.1 Install vim and quagga on R1 (R2)

- Run bash on R1 (R2)

```
bash$ sudo docker exec -it R1 bash
```

```
/# apt-get update  
/# apt-get install -y vim  
/# apt-get install -y quagga
```




Step 4 – Setup Routers (2/6)

4.2 Enable IP forwarding of R1 (R2)

- Edit system control configuration file

```
/# vim /etc/sysctl.conf
```

- Uncomment “net.ipv4.ip_forward=1” in sysctl.conf
 - 1: enable
 - 0: disable
- Run sysctl to load the configuration

```
/# sysctl -p
```



Step 4 – Setup Routers (3/6)

4.3 Enable routing function of Quagga

- Edit Quagga daemons on R1 (R2)

```
/# vim /etc/quagga/daemons
```

- Enable zebra and bgpd daemons
 - Change zebra and bgpd to yes

zebra=no bgpd=no ospfd=no ospf6d=no ripd=no ripngd=no isisd=no babeld=no	→	zebra=yes bgpd=yes ospfd=no ospf6d=no ripd=no ripngd=no isisd=no babeld=no
---	---	---



Step 4 – Setup Routers (4/6)

4.4 Set Hostname and Password of Zebra on R1 (R2)

- Edit configuration file zebra.conf of Quagga on R1 (R2)

```
/# vim /etc/quagga/zebra.conf
```

- Add router name and password in zebra configuration file

```
hostname R1zebra (R2zebra)  
password vRouter  
log stdout
```

- Hostname for identifying zebra on R1 or R2 (for shell prompt)
- Password for user access verification



Step 4 – Setup Routers (5/6)

4.5 Set BGP configuration of routers

```
/# vim /etc/quagga/bgpd.conf
```

```
! BGP configuration for R1
```

```
!
```

```
hostname R1bgp  
password vRouter
```

```
!
```

```
router bgp 65000
```

```
  bgp router-id 172.20.0.2
```

```
  timers bgp 3 9
```

```
  neighbor 172.20.0.3 remote-as 65001
```

```
  neighbor 172.20.0.3 ebgp-multihop
```

```
  neighbor 172.20.0.3 timers connect 5
```

```
  neighbor 172.20.0.3 advertisement-interval 5
```

```
  network 172.18.0.0/16
```

```
!
```

```
log stdout
```

ASN 65000

172.18.0.0/16



h1



R1

172.20.0.2

ASN 65001

172.20.0.3



R2



h2

ASN 65000 —

ASN 65001 —

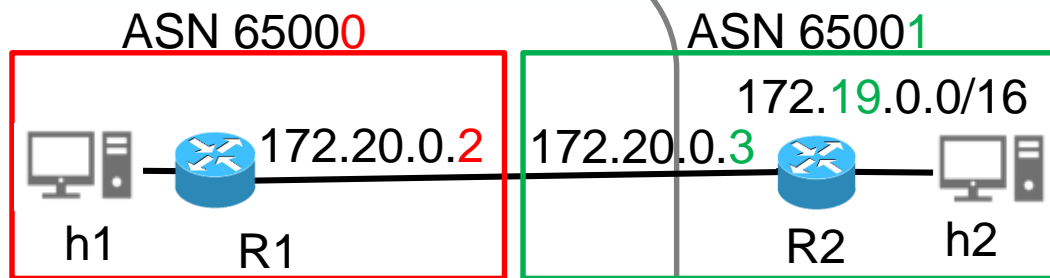


Step 4 – Setup Routers (6/6)

- Edit configuration file bgpd.conf of Quagga on R2

```
/# vim /etc/quagga/bgpd.conf
```

```
! BGP configuration for R2
!
hostname R2bgp
password vRouter
!
router bgp 65001
  bgp router-id 172.20.0.3
  timers bgp 3 9
  neighbor 172.20.0.2 remote-as 65000
  neighbor 172.20.0.2 ebgp-multihop
  neighbor 172.20.0.2 timers connect 5
  neighbor 172.20.0.2 advertisement-interval 5
  network 172.19.0.0/16
!
log stdout
```



ASN 65000 —

ASN 65001 —



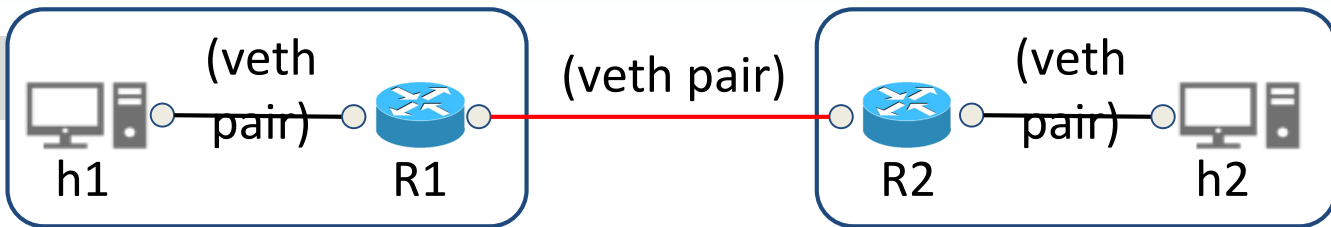
Check Route (1/3)

- Restart Quagga on R1 and R2

```
/# /etc/init.d/quagga restart
```

- Check Route

```
/# ip r
```



```
root@13d4701fa1a8:/# ip route
default via 172.17.0.1 dev eth0
172.17.0.0/16 dev eth0 proto kernel scope link src 172.17.0.4
172.18.0.0/16 dev R1h1veth proto kernel scope link src 172.18.0.2
172.19.0.0/16 via 172.20.0.3 dev R1R2veth proto zebra
172.20.0.0/16 dev R1R2veth proto kernel scope link src 172.20.0.2
```



Check Route (2/3)

- Telnet R1 zebra daemons (on port 2601)

```
/# apt-get install -y telnet  
/# telnet localhost 2601
```

```
User Access Verification
```

```
Password:
```

```
R1zebra> |
```

- Show bgp route in R1zebra

```
R1zebra> show ip route bgp
```

```
R1zebra> 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,  
> - selected route, * - FIB route
```

```
B>* 172.19.0.0/16 [20/0] via 172.20.0.3, R1R2veth, 00:05:55
```



Check Route (3/3)

- Telnet R1 bgpd daemons (on port 2605)

```
/# telnet localhost 2605
```

```
User Access Verification
```

```
Password:
```

```
R1bgp> |
```

- Show R1 bgp summary

```
R1bgp> show ip bgp summary
```

```
R1bgp> show ip bgp summary
```

```
BGP router identifier 172.20.0.2, local AS number 65000
```

```
RIB entries 3, using 336 bytes of memory
```

```
Peers 1, using 4568 bytes of memory
```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
172.20.0.3	4	65001	150	153	0	0	0	00:07:22	1

```
Total number of neighbors 1
```




Outline

- Objective
- Quagga
- Docker
- Dynamic Routing
- **Iptables**
 - Overview
 - Basic usage
- NAT scenarios
- Lab requirement
- Appendix



iptables overview

- **iptables:**

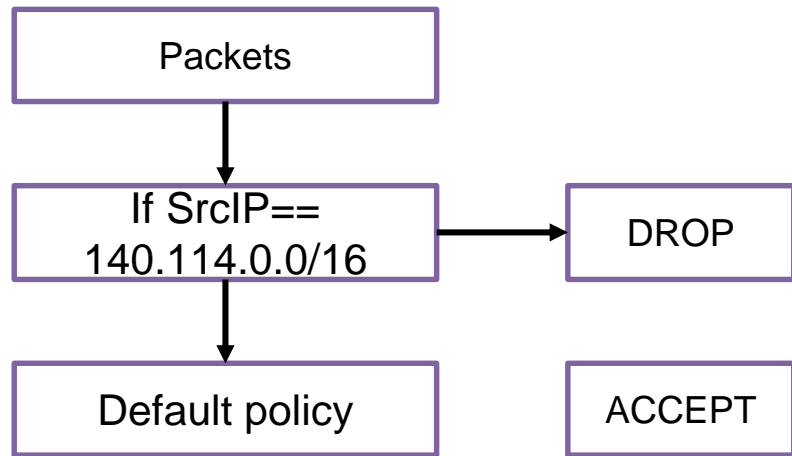
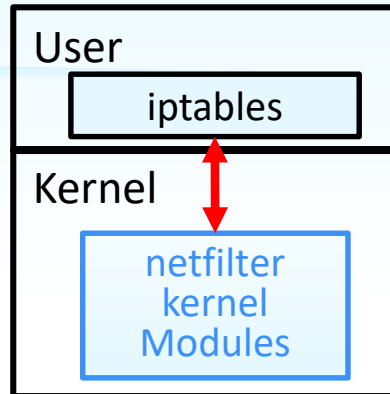
A user-space utility program that allows a system administrator to configure **Linux kernel firewall**

- **Linux kernel firewall** implemented as different **Netfilter modules**

- **Netfilter:**

A framework provided by Linux kernel to implemented various networking operations with some match-action rules

- Network address translation
- Packet filtering
 - e.g., Filtering packet from NTHU





Tables and Chains in iptables

- iptables contains several tables to implement networking operations
- Linux has at least three basic tables
 - Filter: Packet filtering
 - NAT: Network Address translation
 - Mangle: Special tags on packets
- Each table contains several chains
 - Each chain contains match-actions rules to perform networking operation
 - Different chains will be applied in a order in Linux kernel

Ref: <https://zh.wikipedia.org/wiki/Iptables#/media/File:Netfilter-packet-flow.svg>



Table and chain in iptables

- Linux basic tables and their default chains

iptables

Filter

Chain: INPUT
Rule 1

...

Chain: OUTPUT
Rule 1

...

Chain: FORWARD
Rule1

...

NAT

Chain: PREROUTING
Rule 1

...

Chain: OUTPUT
Rule 1

...

Chain: POSTROUTING
Rule1

...

Mangle

Chain:
PREROUTING
Rule 1

...

Chain: OUTPUT
Rule 1

...

Custom tables (option)

Chain: custom chain
Rule 1

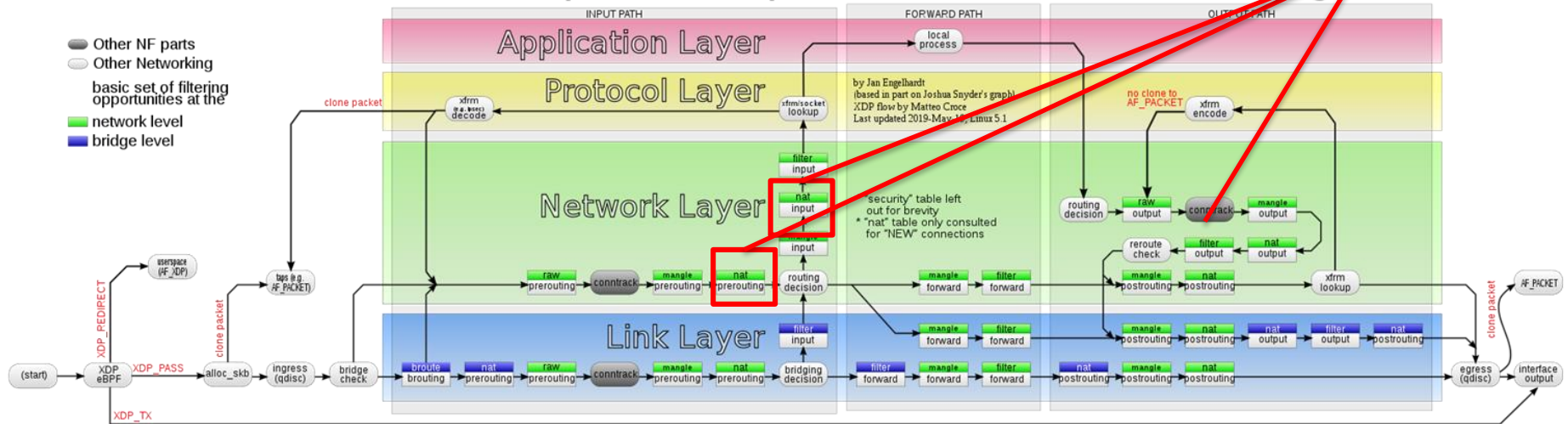


Packet flow in Netfilter

- Applied order example:
 - Chains of NAT table

NAT
Chain: PREROUTING Rule 1 ...
Chain: OUTPUT Rule 1 ...
Chain: POSTROUTING Rule1 ...

Packet flow in Netfilter and General Networking





Outline

- Objective
- Quagga
- Docker
- Dynamic Routing
- **Iptables**
 - Overview
 - **Basic usage**
- NAT scenarios
- Lab requirement
- Appendix



iptables basic usage – add rule

- Append rules to a chain of a table

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

- Example:
 - `sudo iptables -t nat -A POSTROUTING -s 172.20.0.0/16 -d 172.87.0.0/16 -o eth2 -j SNAT --to-source 172.20.0.1`
 - Append a SNAT rules to change source address in chain POSTROUTING of NAT table if packet matched following fields
 - source addresses: 172.20.0.0/16
 - destination addresses: 172.87.0.0/16
 - output interface: eth2
 - action: SNAT



iptables basic usage – list rule

- Show all chains and rules in a table

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

Default
Chains

```
root@a232042897b2:/# sudo iptables -t nat -nvL
Chain PREROUTING (policy ACCEPT 4 packets, 323 bytes)
  pkts bytes target    prot opt in      out     source   destination
Chain INPUT (policy ACCEPT 1 packets, 71 bytes)
  pkts bytes target    prot opt in      out     source   destination
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target    prot opt in      out     source   destination
Chain POSTROUTING (policy ACCEPT 3 packets, 252 bytes)
  pkts bytes target    prot opt in      out     source   destination
    0      0 SNAT          all  --  *      h1R1veth 172.20.0.0/16 172.87.0.0/16
```

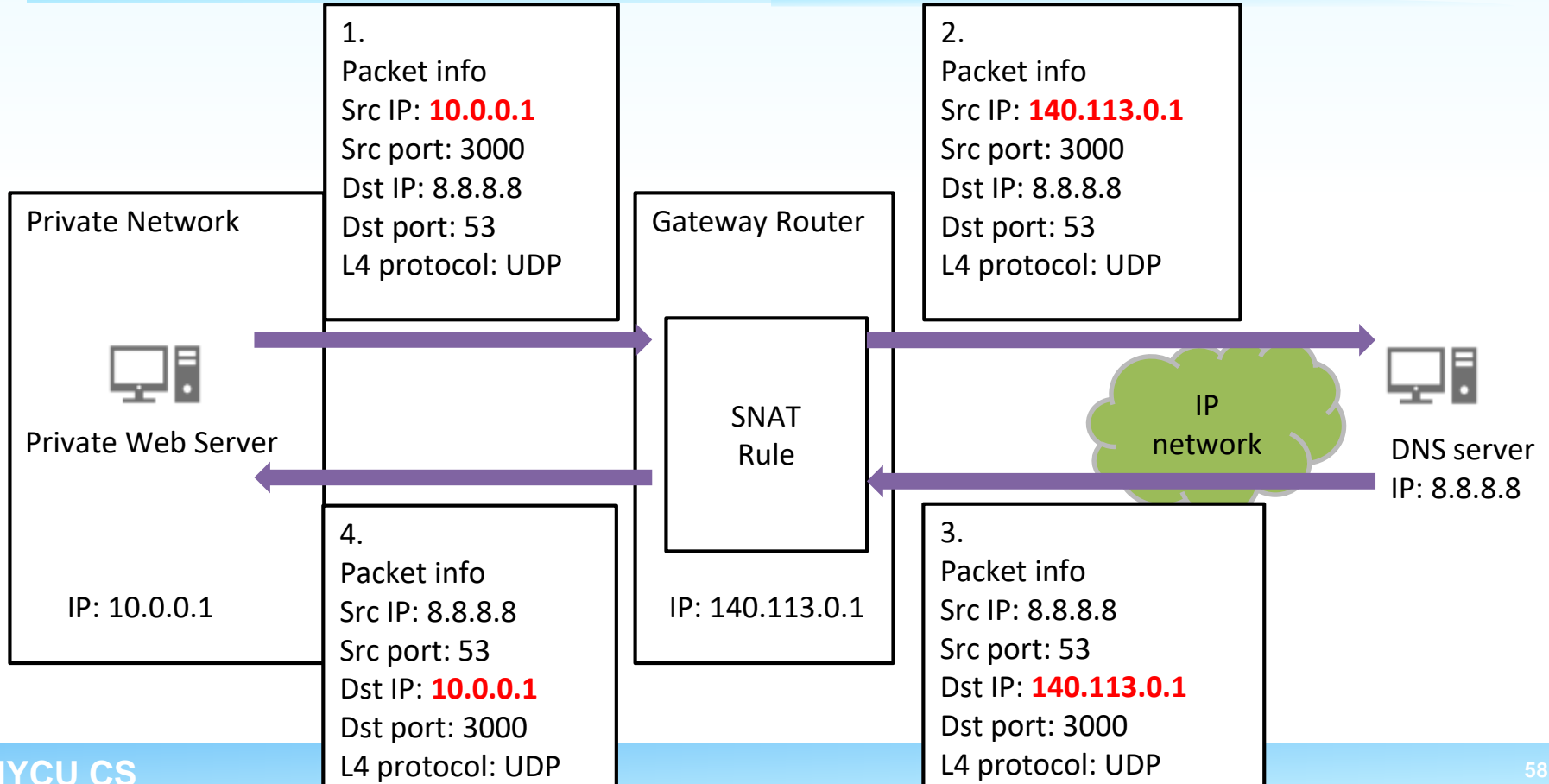



Outline

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



Source NAT





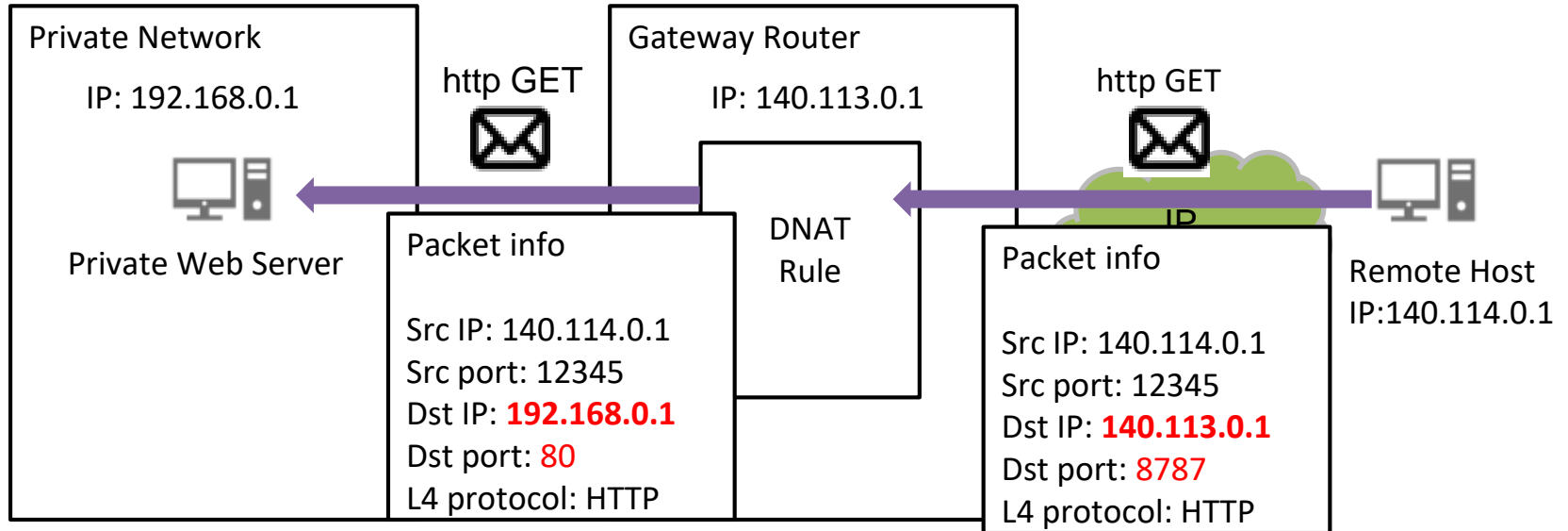
Outline

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



Destination NAT

- System administrator should add some rules in DNAT Rule Table
 - Provide a specific port for remote host to request
 - Every packet send to this port will forward to a specific port of host in private network





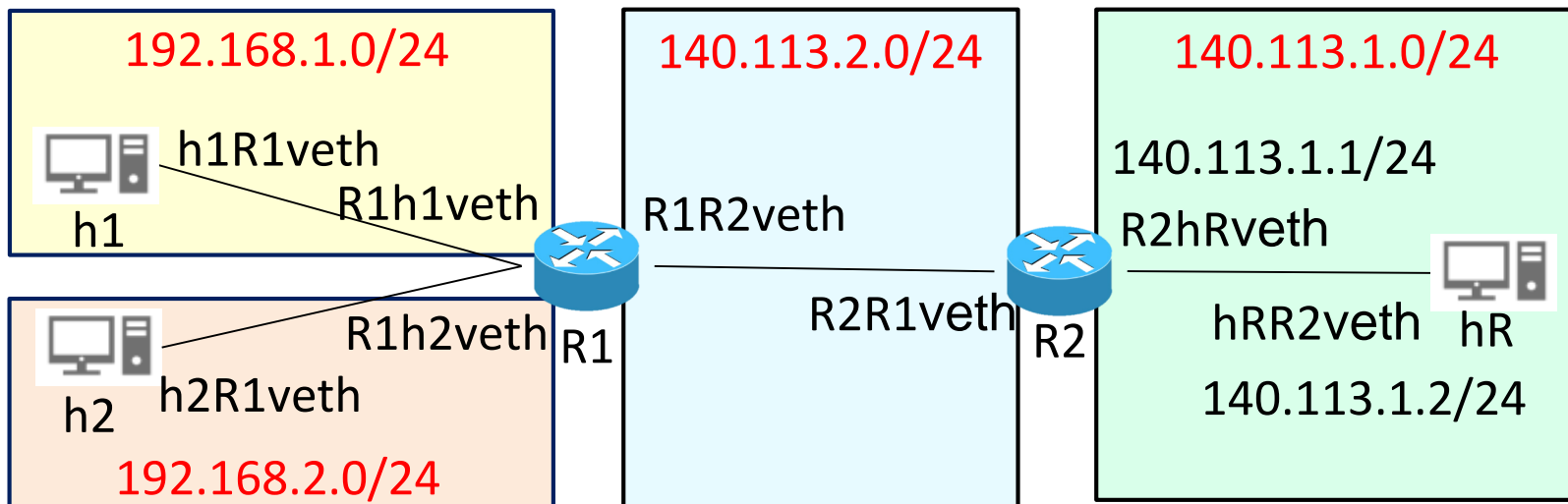
Outline

- Objective
- Quagga
- Docker
- Dynamic Routing
- iptables overview
- NAT scenarios
- Lab requirement
 - Lab topology
 - About submission



Lab Topology

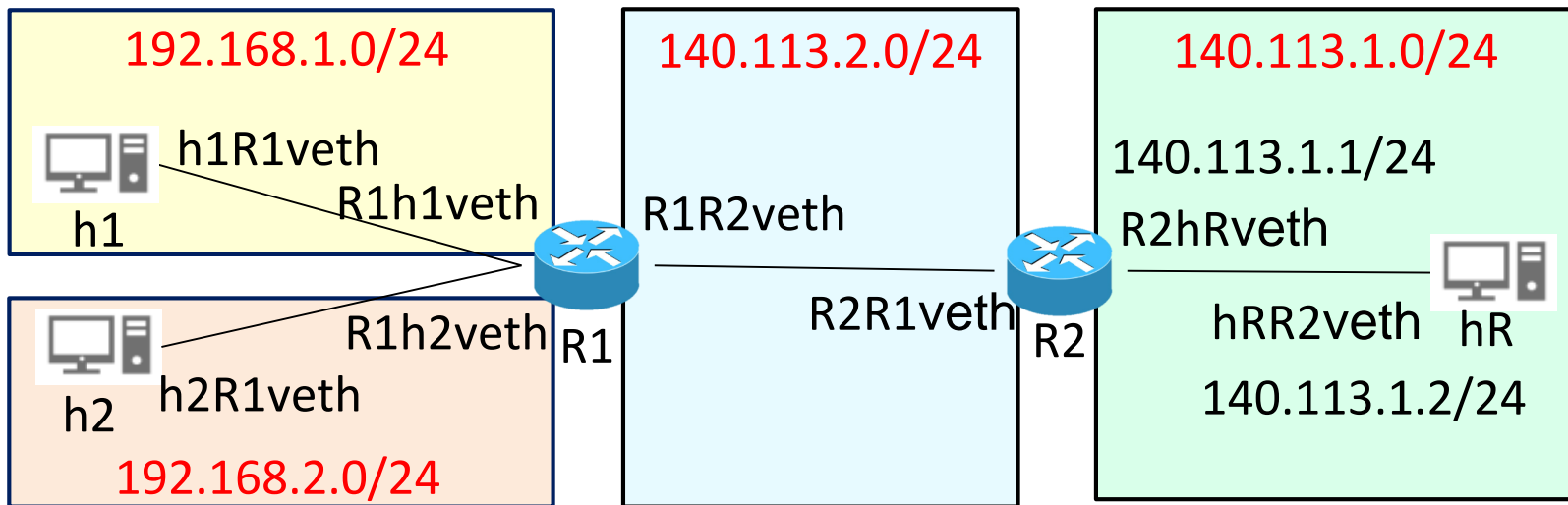
- Build this topology with containers
 - Two private network behind R1, each with one host (h1 and h2)
 - A remote Host hR
 - R1 and R2 use BGP to exchange routing information





Part2: Source NAT

- Let node R1 perform Source NAT with corresponding iptables NAT table rules
 - Modify 192.168.1.0/24 to 140.113.2.30
 - Modify 192.168.2.0/24 to 140.113.2.40





Part1: Setup Environment

- Part1 Requirement: (Use Screenshot to justify your answers)
 - Topology setup (20%)
 - List all containers interfaces
 - Each container is 5 points
 - Quagga info (20%)
 - Show R1 and R2 bgp summary (5% + 5%)
 - R1 and R2 bgpd.conf (5% + 5%)



Part2: Source NAT

- Part2 Requirement:
 - Reachability: (10%)
 - h1 and h2 can ping hR or not
 - Take screenshot and explain your answers
 - Source NAT rules (20%)
 - Invoke 2 terminals of R1
 - tcpdump on R1R2veth and R1h1veth with commands
 - tcpdump -i R1h1veth -eXX
 - tcpdump -i R1R2veth -eXX
 - Take screen shot to show packet bytes before/after NAT rules



Part3: Destination NAT (1/2)

- Run http servers on h1 and h2 respectively
 - h1> python -m SimpleHTTPServer 8080
 - h2> python -m SimpleHTTPServer 9090
- Set DNAT rules on node R1
- Send http requests from host hR
 - hR> curl 140.113.2.1:[port A]
 - hR> curl 140.113.2.1:[port B]
 - DNAT rules should forward requests to h1_IP:8080 and h2_IP:9090, respectively



Part3: Destination NAT (2/2)

- Part 3 requirement:
 - Take screen shots to show the results of `hR curl h1` and `h2`, respectively (10%)
 - Destination NAT rules (20%)
 - Invoke 2 terminals of R1
 - `tcpdump` on `R1R2veth` and `R1h1veth` with commands
 - `tcpdump -i R1R2veth`
 - `tcpdump -i R1h1veth`
 - Take screen shots to show packet bytes before and after DNAT



About Submission

- Files
 - A report: lab3_<studentID>.pdf
 - bgp_R1.conf
 - bgp_R2.conf
 - Part1, Part 2 and Part3 Question Answers
- Submission
 - Zip the report, bgp_R1.conf and bgp_R2.con into a zip file
 - Zip fule Name: lab3_<studentID>.zip
 - Wrong file name or format subjects to 10 points deduction



Appendix

- iptables man page
 - <https://linux.die.net/man/8/iptables>



Q&A