Final Project

SDN-Enabled Virtual Routers

Deadline: 2022/01/10 (Mon) 23:59

Outline

- Introduction
  - Linux Namespaces
    - Network Namespace
  - Veth
- ONOS vRouter App
- Environment Setup Example
- Project Requirements
- Reference

Outline

- Introduction
  - Linux Namespaces
    - Network Namespace
  - Veth
- ONOS vRouter App
- Environment Setup Example
- Project Requirements
- Reference

Linux Namespaces

- A namespace wraps a global system resource in an abstraction
  - Feature of Linux kernel that partitions kernel resources
    - E.g., network, storage, etc.
- Eight kinds of namespaces since kernel version 5.6:
  - **Network (net)**
  - Mount (mnt)
  - Process ID (pid)
  - Interprocess Communication (ipc)
  - UTS (Unix Time-Sharing)
  - User ID (user)
  - Control group (cgroup)
  - Time

Ref: https://en.wikipedia.org/wiki/Linux_namespaces

Network Namespace

- Network namespaces virtualize the network stack
- Each network namespace will have it own:
  - Interfaces
  - IP address set
  - Routing table
  - Firewall
  - …
- Command to list current network namespaces:
  - Use *“ip netns”* command

```
demo@SDN-NFV:~$ sudo ip netns ls
1524 (id: 5)
1345 (id: 4)
1658 (id: 6)
32205 (id: 3)
1171 (id: 0)
```

Outline

- Introduction
  - Linux Namespaces
    - Network Namespace
  - Veth
- ONOS vRouter App
- Environment Setup Example
- Project Requirements
- Reference

Linux Veth

- The ***veth*** devices are virtual Ethernet devices
- Veth devices are always created in interconnected pairs
- Act as tunnels between network namespaces
  - Just like creating a bridge to a physical device in another network namespace
- Create a veth pair:

```
/# ip link add <p1-name> type veth peer name <p2-name>
```

  - ***p1-name*** and ***p2-name***: names assigned to the two connected end points

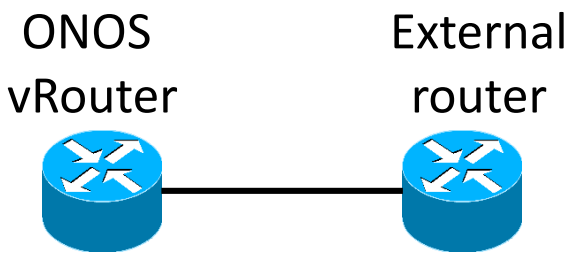
- An veth pair

Outline

- Introduction
- ONOS vRouter App
- Environment Setup Example
- Project Requirements
- Reference

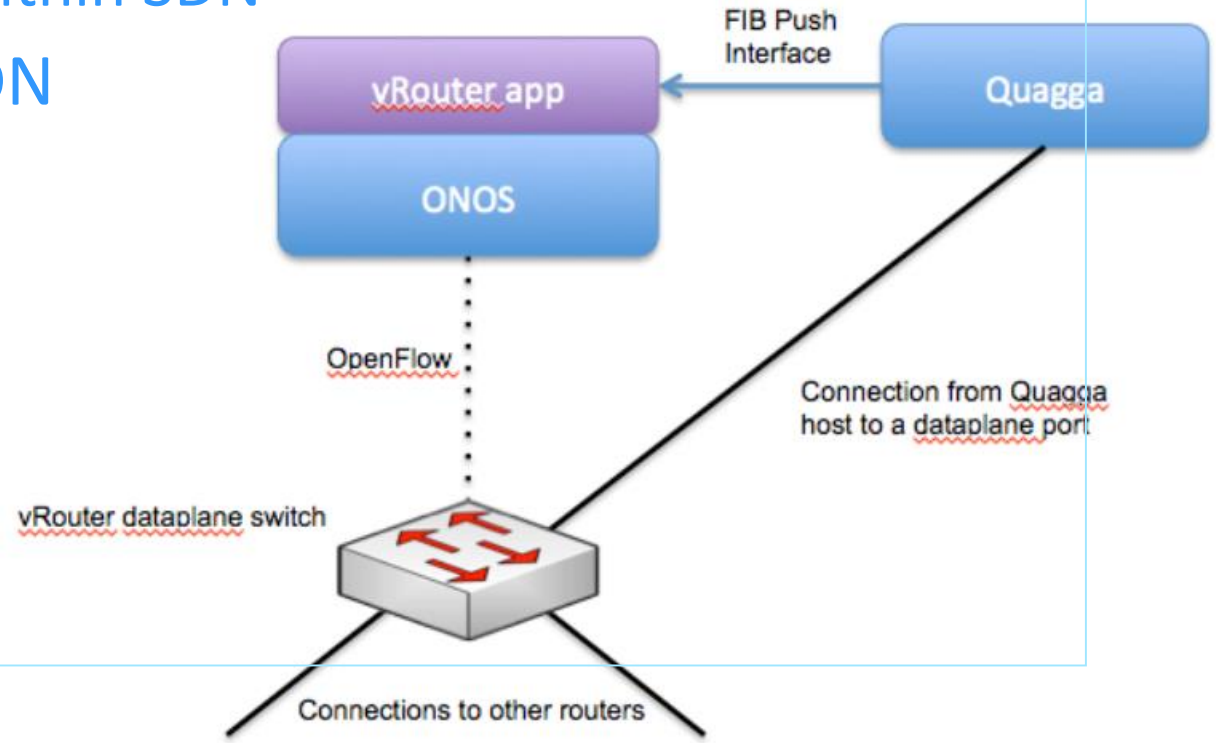
ONOS vRouter App

- vRouter is an ONOS application
- vRouter was designed for ONOS CORD project
  - Service as the gateway between CORD infrastructure and the upstream network
  - To provide Internet access to services within SDN
- The external router views the entire SDN network as a single router



- View from external router

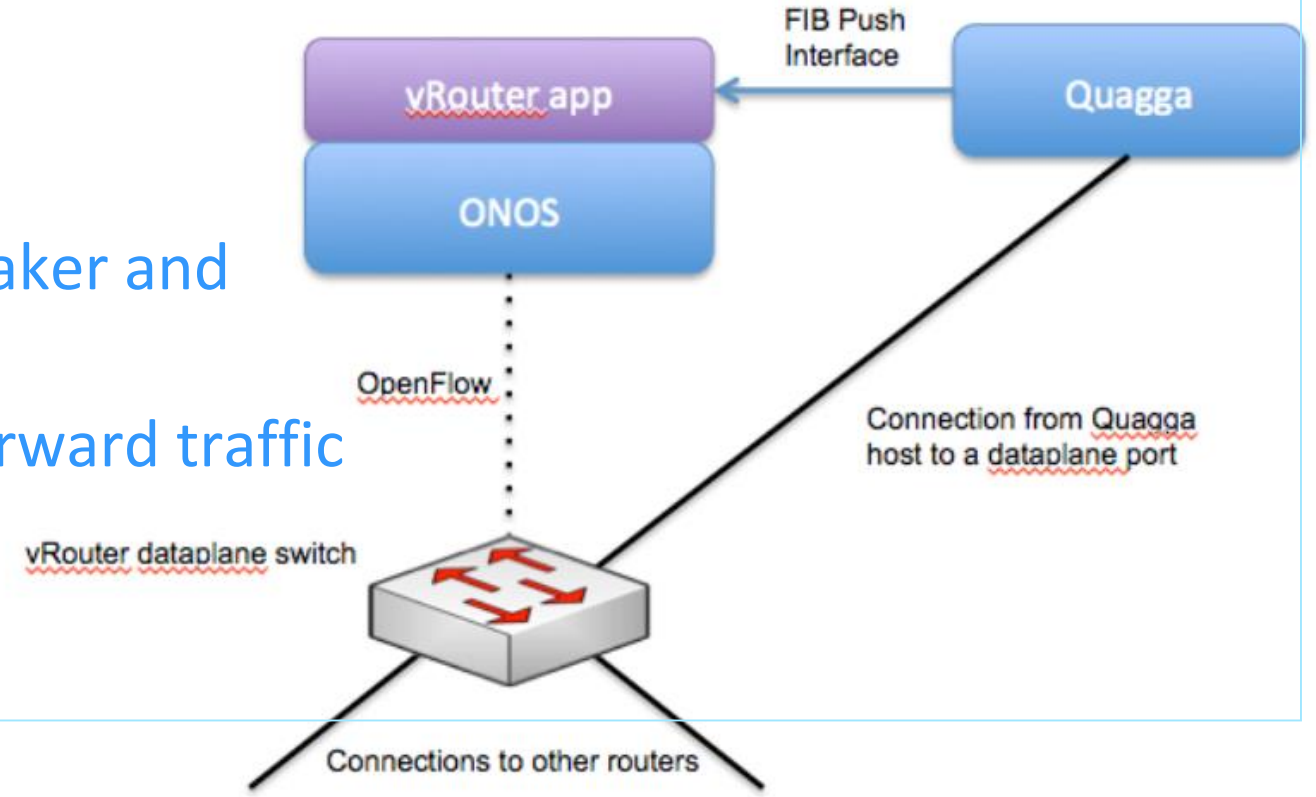
- High level architecture



Design of ONOS vRouter

- Control plane:
  - Use **Quagga** as speaker to speak routing protocol with external routers
    - E.g. BGP speaker
  - Quagga (speaker) pushes learned routes to ONOS via FIB Push Interface (FPM*)
    - FIB: Forwarding Information Base
- Data plane:
  - Set an edge switch sits between speaker and external routers
  - vRouter installs flow rules on it to forward traffic

- High level architecture



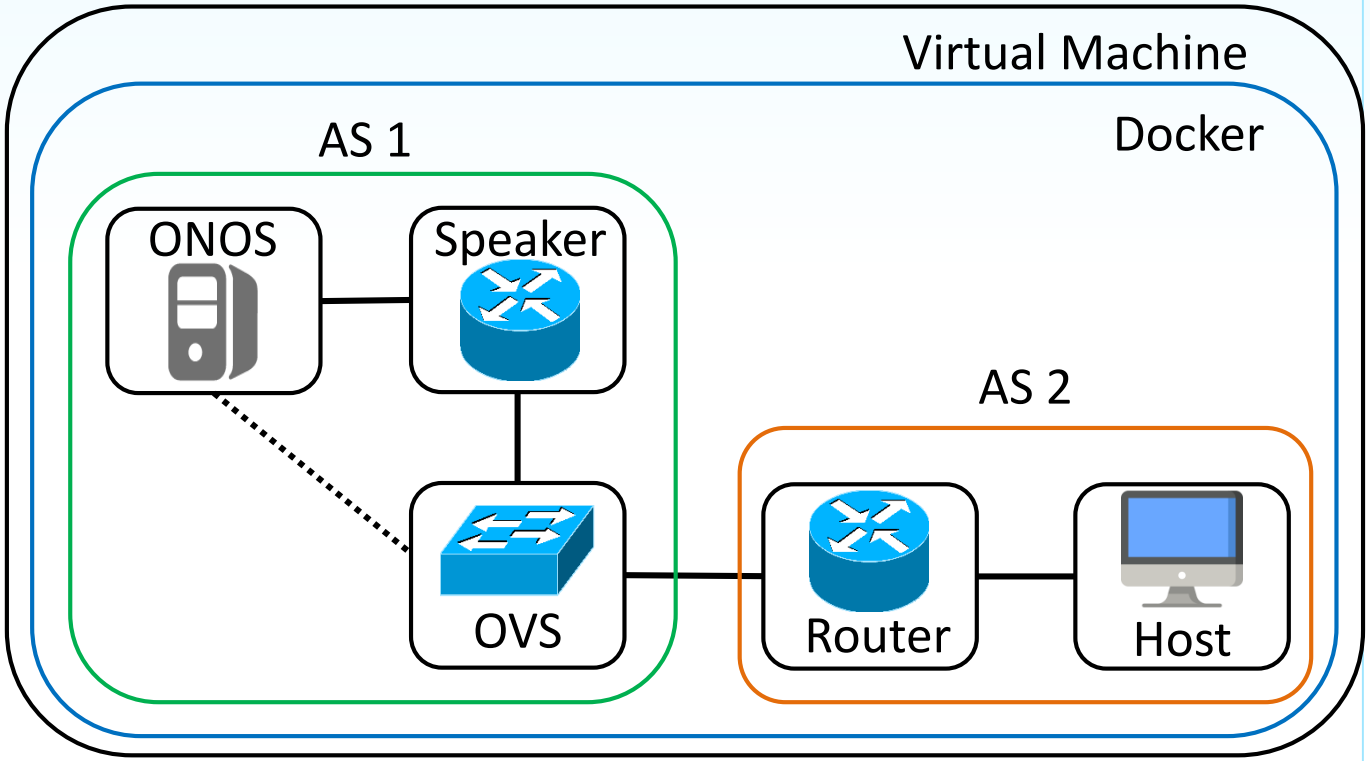
*FPM: Forwarding Plane Manager

Outline

- Introduction
- ONOS vRouter App
- Environment Setup Example
- Project Requirements
- Reference

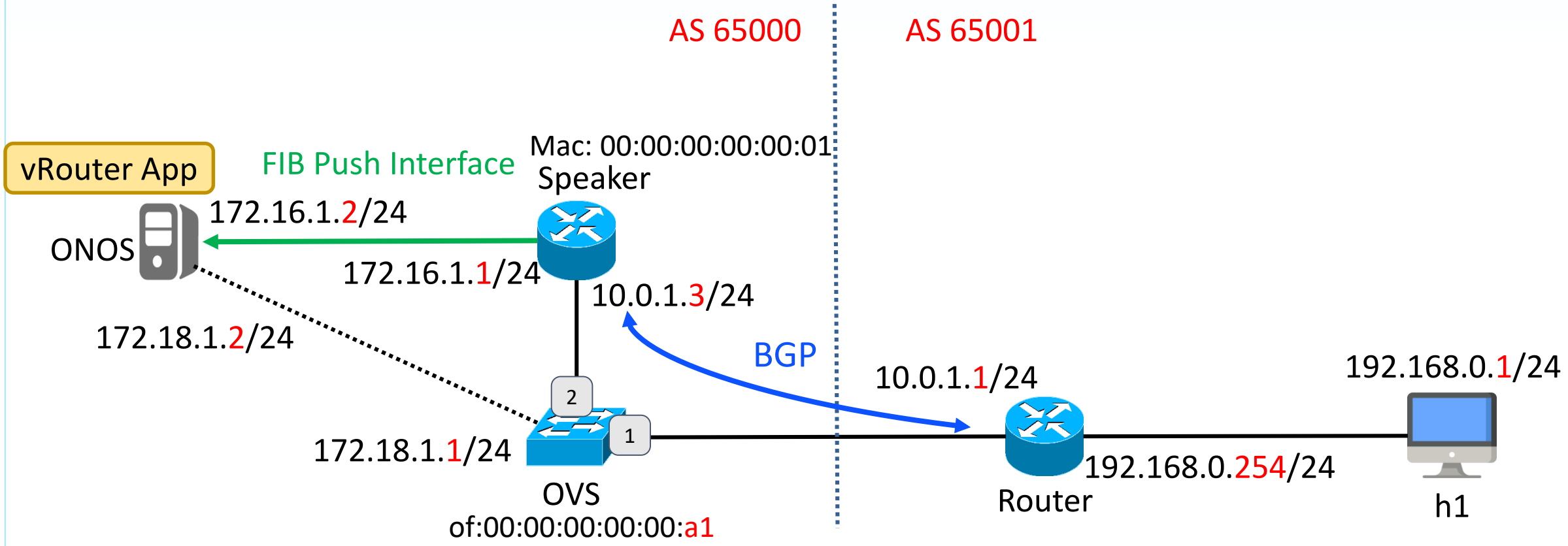
Environment

- Virtual Machine
  - Ubuntu 16.04
- Docker Images
  - Hosts and external routers:
    - ubuntu:16.04
  - ONOS:
    - onosproject/onos:2.2.0
  - OVS:
    - openshift/openvswitch:latest
  - Speaker:
    - johnny3644186/onos_vrouter (image provided by TA)
      - With Quagga pre-installed and configured with *"--enable-fpm"* flag



Scenario

- An example scenario
- Speaker has BGP session with external router



Environment Setup

1. Create Docker Containers
2. Setup Networks among Containers
3. Configure Host Gateway
4. Setup OVS
5. Configure Speaker
6. Configure External Router
7. Activate ONOS vRouter App
8. Create ONOS Network Configuration file
9. Upload ONOS Network Configuration
10. Check Results
11. Clean Up the environment

1. Create Docker Containers

- Create container for ONOS

```
~$ sudo docker run -it -d -e DISPLAY=$DISPLAY \
       -v /tmp/.X11-unix:/tmp/.X11-unix \
       -p 8181:8181 -p 8101:8101 -p 6653:6653 \
       --privileged --cap-add NET_ADMIN --cap-add NET_BROADCAST \
       --cap-add SYS_MODULE \
       --name ONOS onosproject/onos:2.2.0
```

  - -e: Set environment variable
  - -v: mount volumes (directory mapping)
  - -p: port mapping (forwarding)

1. Create Docker Containers

- Create container for Router (h1)

```
~$ sudo docker run –it –d --privileged --cap-add NET_ADMIN \
       --cap-add NET_BROADCAST --cap-add SYS_MODULE \
       --name Router Ubuntu:16.04
```

- Create container for Speaker

```
~$ sudo docker run –it –d –e DISPLAY=$DISPLAY \
       -v /tmp/.X11-unix/:/tmp/.X11-unix
       --privileged --cap-add NET_ADMIN \
       --cap-add NET_BROADCAST --cap-add SYS_MODULE \
       --name Speaker johnny3644186/onos_vrouter
```

- Create container for OVS

```
~$ sudo docker run –it –d --privileged --cap-add NET_ADMIN \
       --cap-add NET_BROADCAST --cap-add SYS_MODULE \
       --name OVS openshift/openvswitch:latest
```

1. Create Docker Containers

- After creating docker containers, we need to disable access control for VM's X server to allow us launch wireshark in Speaker later.
- Use the following command to allow access:

```
~$ xhost +
```

- The result should be like below:

```
demo@SDN-NFV:~$ xhost +
access control disabled, clients can connect from any host
```

- Run bash on Speaker:

```
~$ sudo docker exec –it Speaker bash
```

- Install Wireshark on Speaker:

```
/# apt-get update
/# apt-get install –y wireshark
```

2. Setup Networks among Containers

a. Network Namespaces of Docker Containers
b. Create Soft (Symbolic) Links for Container Network Namespaces
c. Create Veth Pairs between Containers
d. Create Docker Network between Router and Host
   - Create a Docker Network
   - Pre-Install Needed Tools
   - Connect Containers to Docker Network

2(a) Docker Container Network Namespace

- By convention, a named network namespace is an object at *"/var/run/netns/<NS_NAME>"*
  - "ip netns" gets network namespaces under this directory *(/var/run/netns/)*

- Create a netns **"ns1"** and list ns

```
winlab@server168:~$ sudo ip netns add ns1
winlab@server168:~$ ip netns ls
ns1
```

- Display files under /var/run/netns/

```
winlab@server168:/var/run/netns$ ls
ns1
```

- However, Docker puts container ns under ***"/var/run/docker/netns/"*** instead
  - Thus, "ip netns" cannot find container namespaces under its default directory

- Create a container **"test"** and list ns again

```
root@server168:~# docker run -it -d --privileged \
> --cap-add NET_ADMIN --name test ubuntu:16.04
bafb165d0bcc199286a3c9fa05f21b9b0551a616eb8936e8e0c57992227102cf
root@server168:~# ip netns ls
ns1
```

No container's ns in result

- Container ns under /var/run/docker/netns/

```
root@server168:~# cd /var/run/docker/netns/
root@server168:/var/run/docker/netns# ls
82995c30d2c9
```

Container test's ns

2(a) Docker Container Network Namespace

- Actually, the container network namespace file is at *"/proc/<PID>/ns/net"*
  - <PID>: container's process ID

```
root@server168:/var/run/docker/netns# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
bafb165d0bcc        ubuntu:16.04        "/bin/bash"         8 minutes ago       Up 8 minutes                            test
root@server168:/var/run/docker/netns# cd
root@server168:~# docker inspect -f '{{.State.Pid}}' bafb165d0bcc
44159
```

Container test's PID

- Container test's net namespace under /proc/44159/ns/:

```
root@server168:/proc/44159/ns# ls
cgroup  ipc  mnt  net  pid  pid_for_children  user  uts
```

- To manipulate container network namespace by "ip netns", we need to create a symbolic link to *"/proc/<PID>/ns/net"* under *"/var/run/netns/"*

2(b) Create Soft Link for Container Namespace

1. Get container ID:

```
~$ sudo docker ps -a
```

```
winlab@server168:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
8359acda4661        ubuntu:16.04        "/bin/bash"         5 seconds ago       Up 4 seconds                            Router
```

2. Use container ID to get its PID:

```
~$ sudo docker inspect –f '{{.State.Pid}}' <CONTAINER_ID>
```

```
winlab@server168:~$ sudo docker inspect -f '{{.State.Pid}}' 8359acda4661
46248
```

3. Create directory “/var/run/netns/” if it doesn’t exist:

```
~$ sudo mkdir –p /var/run/netns
```

4. Create soft link to link container’s ns to directory /var/run/netns/:

```
~$ sudo ln –s /proc/<PID>/ns/net /var/run/netns/<PID>
```

```
winlab@server168:/var/run/netns$ sudo ln -s /proc/46248/ns/net /var/run/netns/46248
```

2(c) Create Veth Pairs between Containers

- Create veth paris between **all** containers
  - **Except** link between Router and h1 (Use Docker network later)
- Example: Create a veth pair to connect OVS and Router
  - Create a veth pair:

```
~$ sudo ip link add vethRouterOvs type veth peer name vethOvsRouter
```

  - Set vethRouterOvs to Router:

```
~$ sudo ip link set vethRouterOvs netns <Router_PID>
```

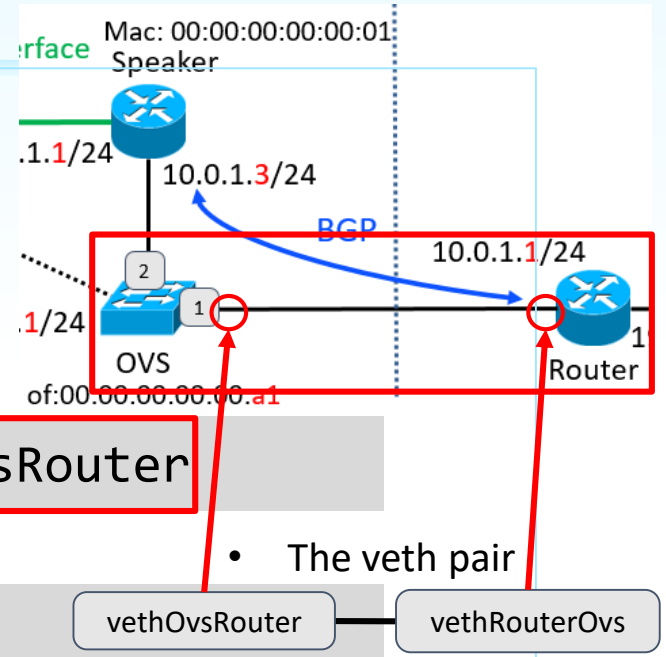
  - Set vethOvsRouter to OVS:

```
~$ sudo ip link set vethOvsRouter netns <OVS_PID>
```

  - Bring veth interfaces up respectively:

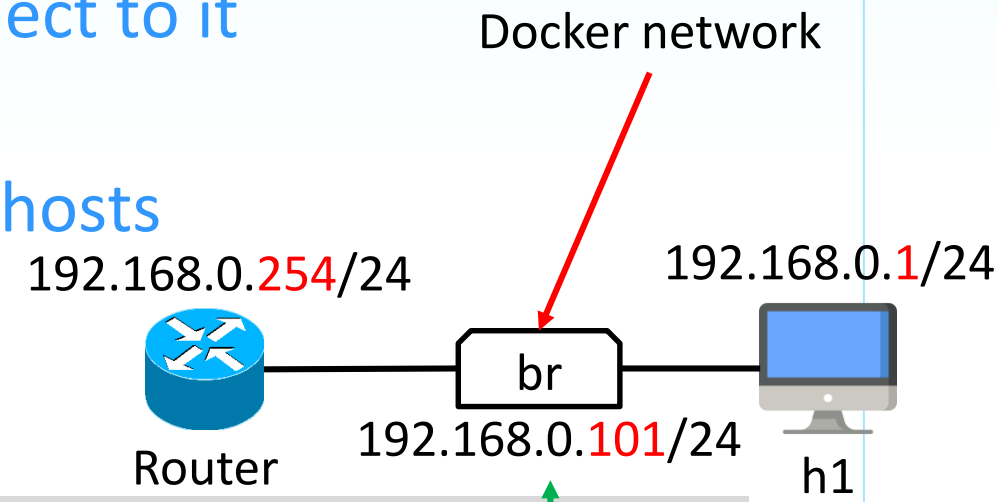
```
~$ sudo ip netns exec <Router_PID> ip link set dev vethRouterOvs up
~$ sudo ip netns exec <OVS_PID> ip link set dev vethOvsRouter up
```

- For all other containers, same concept as above



2(d) Create Docker Network between Router and Host

- Veth pair is used to connect two containers directly
- However, a router may have many hosts connect to it
  - Veth pair is not suitable for this scenario
- ➢ Use Docker network to connect a router with hosts
- Example:
  - Create a Docker network named “br”:




```
~$ sudo docker network create --subnet=192.168.0.0/24 \
      --gateway=192.168.0.101 br
```

  - --subnet: the subnet IP addresses on this docker network (bridge)
  - --gateway: the IP of the bridge

2(d) Pre-Install Needed Tools on Containers

- After connect to a Docker network, container will use the connected network as the default gateway
  - Need to install needed tools on containers before connect them to a Docker network
- Example: Pre-install needed tools on Router and h1
  - Install tools and Quagga on **Router**:

```
~$ sudo docker exec -it Router apt-get update
~$ sudo docker exec –it Router apt-get install –y \
       net-tools iproute2 iputils-ping \
       vim telnet quagga
```

  - Install tools on **h1**:

```
~$ sudo docker exec -it h1 apt-get update
~$ sudo docker exec –it h1 apt-get install –y \
       net-tools iproute2 iputils-ping
```

2(d) Connect Containers to Docker Network

- Example: Connect Router and h1 to br
  - Connect Router to Docker network (bridge) br:

```
~$ sudo docker network connect --ip 192.168.0.254 br Router
```

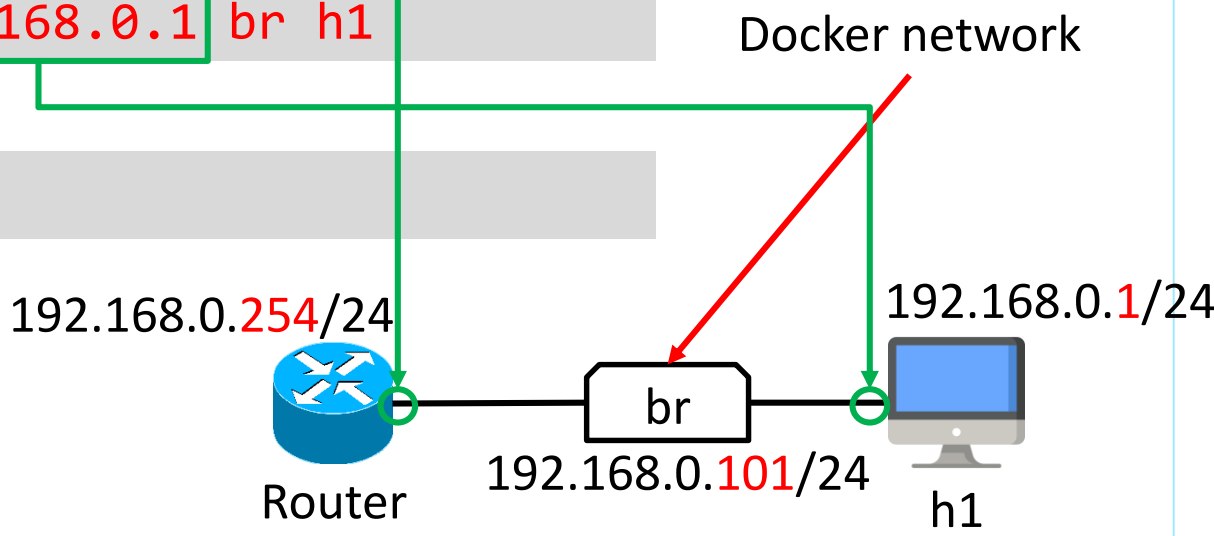
  - Connect h1 to Docker network (bridge) br:

```
~$ sudo docker network connect --ip 192.168.0.1 br h1
```

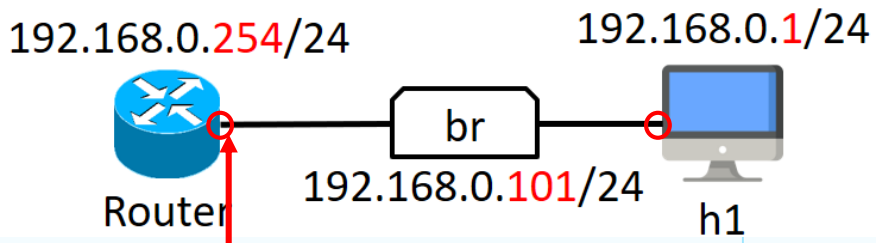
  - Check containers connect to br:

```
~$ sudo docker inspect br
```


```
"Containers": {
    "aa9ddfb7e58bbccd980ed99773551e17ee5ca10fe850c56f960decebee763d0c": {
        "Name": "Router",
        "EndpointID": "48ee055127896b8a28a969e37ba985f2a76bbca0819bed981deff0f6a32a65d4",
        "MacAddress": "02:42:c0:a8:00:fe",
        "IPv4Address": "192.168.0.254/24",
        "IPv6Address": ""
    },
    "f5fd6d44b42ba0ed8b438fe0ef014bf20d63fb14308c042a63e098b272ae376c": {
        "Name": "h1",
        "EndpointID": "a89f3887e2110de17355d534c3e181bb9da0d35bf454742ac5bd57ef79c0b116",
        "MacAddress": "02:42:c0:a8:00:01",
        "IPv4Address": "192.168.0.1/24",
        "IPv6Address": ""
    }
},
```

3. Configure Host Gateway



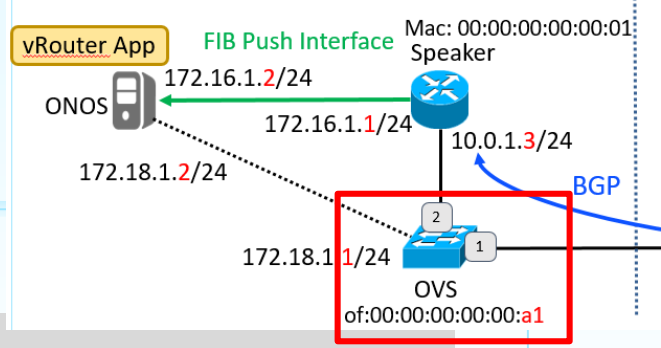
- Assume h1's interface **"eth1"** connects to br
- Set Router's IP as h1's gateway
- Run bash on h1:

```
~$ sudo dokcer exec -it h1 bash
```

- Configure h1's gateway:

```
/# ip route del default  # delete default gateway
/# ip route add default via 192.168.0.254 #Add gateway
```

4. Setup OVS – Create OVS Bridge



- Run bash on OVS:

```
~$ sudo dokcer exec –it OVS bash
```

- Use “ovs-vsctl” to create an OVS bridge named “ovsbr”:

```
/# ovs-vsctl add-br ovsbr \
      -- set bridge ovsbr other-config:datapath-id=0000000000000a1
```

  - other-config: option to specify other configuration on ovs bridge
    - datapath-id: To set DPID of the bridge “ovsbr”
      - ovsbr’s DPID will be “of:0000000000000a1”

- Check dpid:

```
/# ovs-ofctl show ovsbr
```

```
[root@7e2da75701a5 origin]# ovs-ofctl show ovsbr
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000a1
```

4. Setup OVS – Setting Port/Controller

- Assume two veths on OVS were called “vethOvsRouter” and “vethOvsSpeaker” respectively
- Add ports on OVS and specify their OpenFlow port number
  - Specify “vethOvsRouter” as port 1 on OVS:

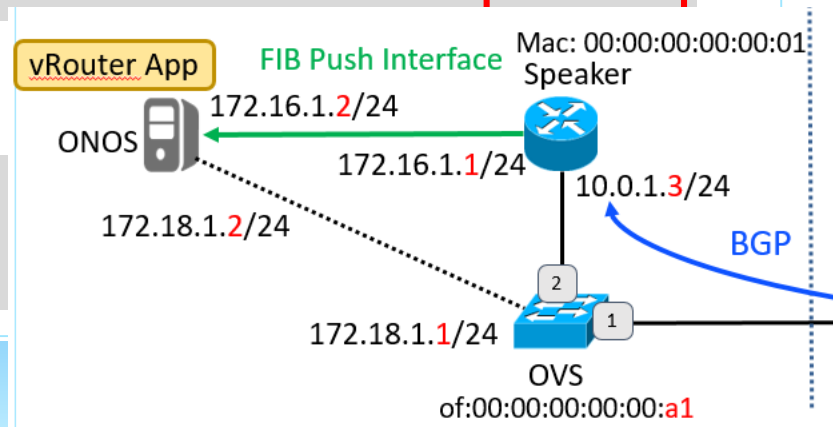
```
/# ovs-vsctl add-port ovsbr vethOvsRouter \
      -- set interface vethOvsRouter ofport_request=1
```

  - Specify “vethOvsSpeaker” as port 2:

```
/# ovs-vsctl add-port ovsbr vethOvsSpeaker \
      -- set interface vethOvsSpeaker ofport_request=2
```

  - Set up controller for ovsbr:

```
/# ovs-vsctl set-controller ovsbr \
      tcp:172.18.1.2:6653
```



5. Configure Speaker – zebra.conf

- Run bash on Speaker:

```
~$ sudo docker exec –it Speaker bash
```

- Edit Quagga zebra.conf on Speaker

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

  – Add router name and password and FPM connection info:

```
! Configuration for zebra
!
hostname zebra
password vRouter
log stdout
!
! ONOS fpm listens on port 2620
!
fpm connection ip 172.16.1.2 port 2620
```

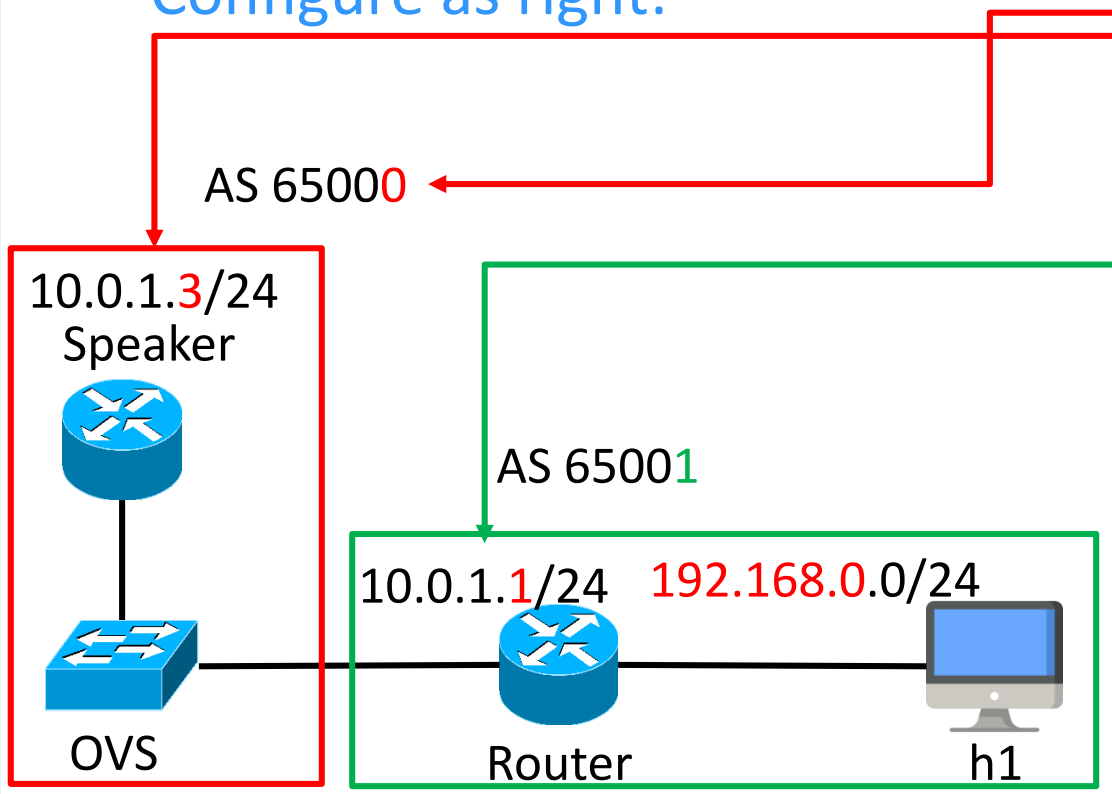
ONOS FPM IP

5. Configure Speaker – bgpd.conf

- Setup BGP configuration on Router
  - Edit Quagga bgpd.conf on Router:

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

  - Configure as right:




```
! BGP configuration for Speaker
!
hostname SpeakerBGP
password vRouter
!
rourouter bgp 65000
  bgp router-id 10.0.1.3
  timers bgp 3 9
  neighbor 10.0.1.1 remote-as 65001
  neighbor 10.0.1.1 ebgp-multihop
  neighbor 10.0.1.1 timers connect 5
  neighbor 10.0.1.1 advertisement-interval 5
  network 20.0.20.0/24
  !
  !redistribute ospf
  !
!
log stdout
```

A dummy network to advertise

5. Configure Speaker – Start Daemons

*VTY: Virtual Teletype Terminal

- Start zebra daemon:

```
/# zebra -d -A 127.0.0.1 --retain
```

  - -d: Run as a daemon
  - -A: Set the VTY* local address to bind to
  - --retain: When program terminates, retain routes added by zebra

- Start bgpd daemon:

```
/# bgpd -d -A 127.0.0.1
```

- Check if zebra, bgpd are listening on expected ports:

```
/# netstat -tnlp
```

```
root@3fe82f335593:/# netstat -tnlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:2601          0.0.0.0:*               LISTEN      21609/zebra
tcp        0      0 127.0.0.1:2605          0.0.0.0:*               LISTEN      21612/bgpd
tcp        0      0 0.0.0.0:179             0.0.0.0:*               LISTEN      21612/bgpd
```

6. Configure External Router (1/4)

- The concept here is the same as in lab 3
- Note: Quagga is already installed on Router at step 2(d)
- Run bash on Router:

```
~$ sudo docker exec -it Router bash
```

- Enable IP forwarding:

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

  - Uncomment “net.ipv4.ip_forward=1” in sysctl.conf

- Reload the configuration:

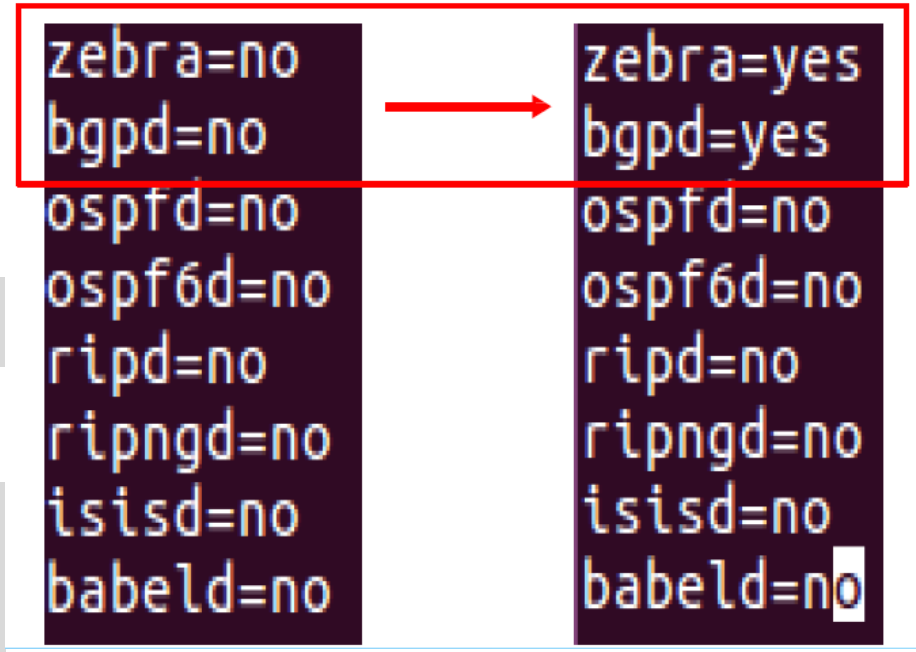
```
/# sysctl -p
```

6. Configure External Router (2/4)

- Enable routing functions of Quagga on Router
  - Edit Quagga daemons :

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

  - Enable zebra and bgpd daemons:
    - Change zebra and bgpd to “yes”



- Edit Quagga zebra.conf on Router

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

  - Add router name and password:

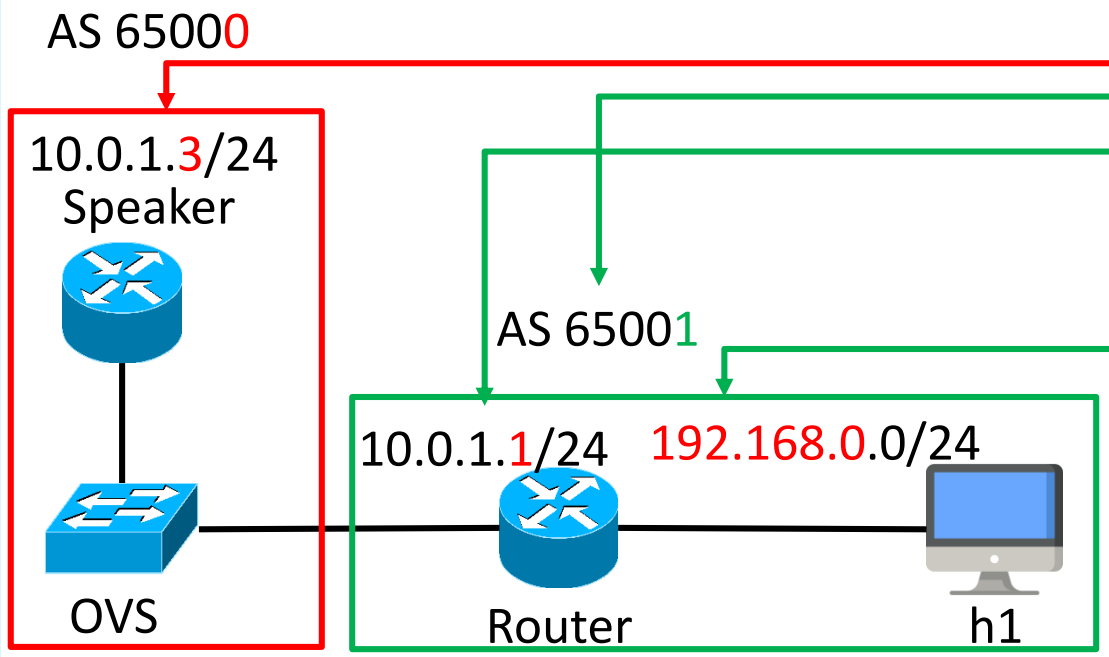
```
hostname zebra
password vRouter
log stdout
```

6. Configure External Router (3/4)

- Setup BGP configuration on Router
  - Edit Quagga bgpd.conf on Router:

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

  - Configure as right:




```
! BGP configuration for router
!
hostname Routerbgp
password vRouter
!
router bgp 65001
  bgp router-id 10.0.1.1
  timers bgp 3 9
  neighbor 10.0.1.3 remote-as 65000
  neighbor 10.0.1.3 ebgp-multihop
  neighbor 10.0.1.3 timers connect 5
  neighbor 10.0.1.3 advertisement-interval 5
  !
  network 192.168.0.0/24
!
log stdout
```

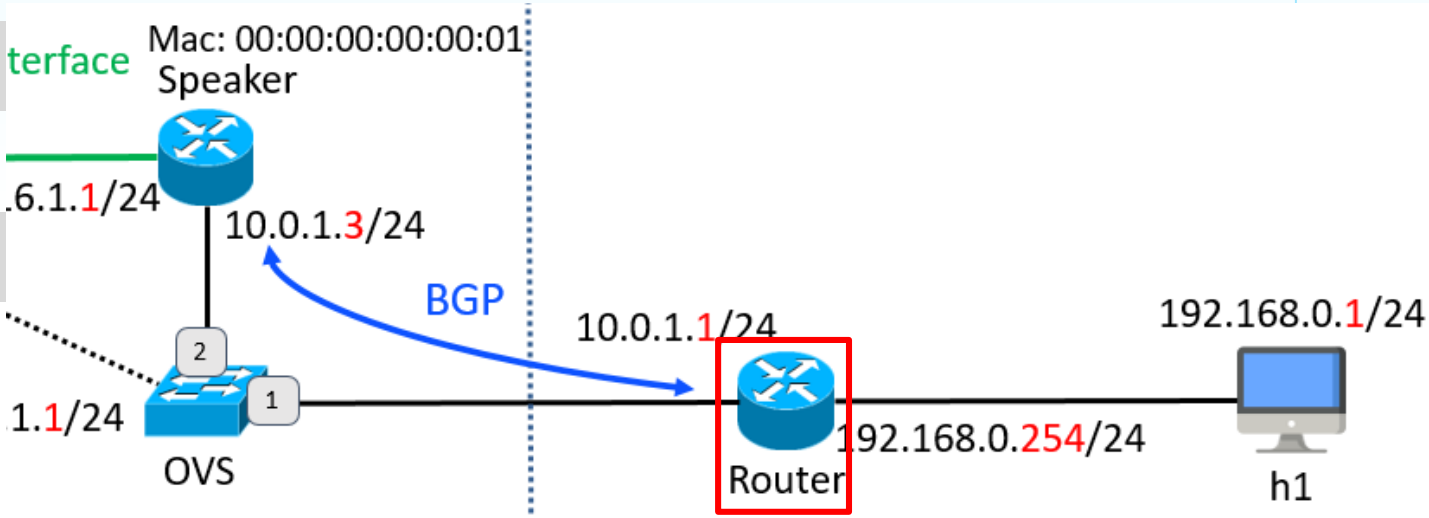
6. Configure External Router (4/4)

- Restart Quagga on Router:

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

- Check route:

```
/# route
```



route learned from speaker

```
root@70d9a448c062:/# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         172.17.0.1      0.0.0.0         UG    0      0        0 eth0
10.0.1.0        *               255.255.255.0   U     0      0        0 vethRouterOvs
20.0.20.0       10.0.1.3        255.255.255.0   UG    0      0        0 vethRouterOvs
172.17.0.0      *               255.255.0.0     U     0      0        0 eth0
192.168.0.0     *               255.255.255.0   U     0      0        0 vethRouterH1
```

7. Activate ONOS vRouter App – Enter ONOS CLI

- Run bash on ONOS:

```
~$ sudo docker exec -it ONOS bash
```

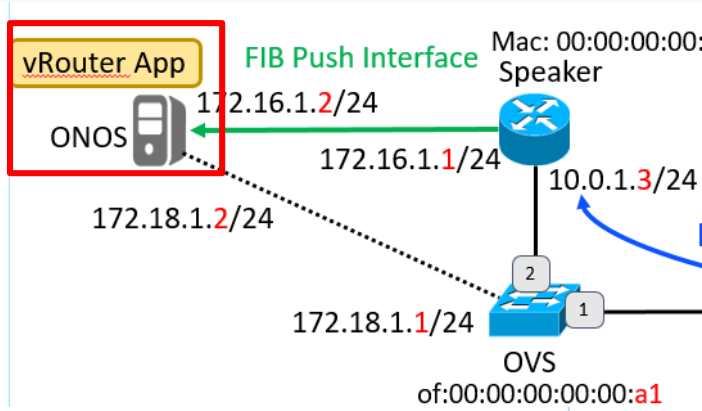
- Install ssh server on ONOS:

```
/# apt-get update
/# apt-get install -y curl net-tools
/# apt-get install -y openssh-server
```

- Enter ONOS CLI:

```
/# ssh -p 8101 karaf@localhost
```

  - Password is also **"karaf"**
  - After login, you will see ONOS CLI prompt like this:




```
root@5bf476860408:~/onos# ssh -p 8101 karaf@localhost
Password authentication
Password:
Welcome to Open Network Operating System (ONOS)!

    ____  _  ______  ____
   / __ \/ |/ / __ \/ __/
  / /_/ /    / /_/ /\ \
  \____/_/|_/\____/___/

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.

karaf@root >
```

7. Activate ONOS vRouter App

- In ONOS CLI, activate OpenFlow suite first
  - Note: The ONOS container doesn't have OpenFlow activated initially

```
karaf@root > app activate org.onosproject.openflow
```

- Then, activate vRouter app:

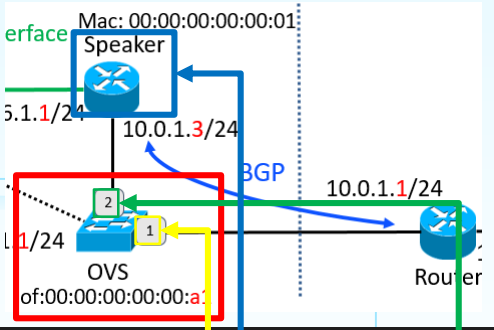
```
karaf@root > app activate vrouter
```

- Check if apps are activated successfully:

```
karaf@root > apps -a -s
*   6 org.onosproject.hostprovider          2.2.0    Host Location Provider
*   9 org.onosproject.optical-model         2.2.0    Optical Network Model
*  10 org.onosproject.drivers               2.2.0    Default Drivers
*  37 org.onosproject.gui2                  2.2.0    ONOS GUI2
*  60 org.onosproject.lldpprovider          2.2.0    LLDP Link Provider
*  77 org.onosproject.route-service         2.2.0    Route Service Server
*  97 org.onosproject.fpm                   2.2.0    FIB Push Manager (FPM) Route Receiver
* 132 org.onosproject.openflow-base         2.2.0    OpenFlow Base Provider
* 133 org.onosproject.openflow              2.2.0    OpenFlow Provider Suite
* 145 org.onosproject.fibinstaller          2.2.0    FIB Installer
* 176 org.onosproject.cpr                   2.2.0    Control Plane Redirect
* 177 org.onosproject.vrouter               2.2.0    Virtual Router
```

8. Create ONOS Network Configuration File

- For example, create a configuration file named “test.json”
- In test.json, there are 3 parts of config:
  - devices:
    - Specify the driver as “softrouter”
  - ports:
    - name: user-defined name for the interface
    - ips: speaker’s IP used to connect to the external router on this subnet
    - mac: speaker’s mac address
  - Apps:
    - controlPlaneConnectPoint: the connectpoint of speaker
    - Interfaces: interfaces connected to external



- test.json

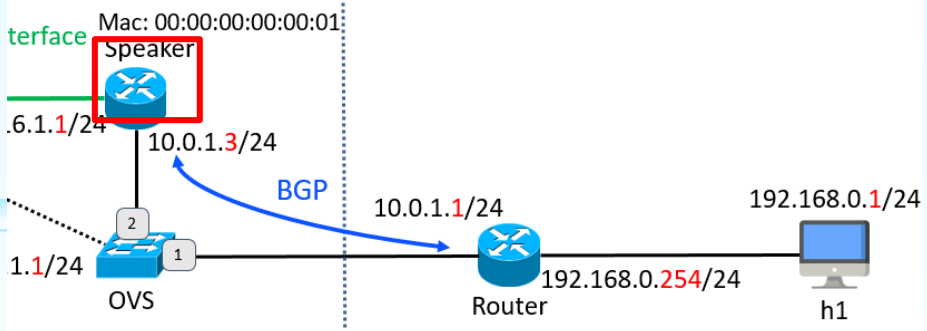
```
{
    "devices" : {
        "of:00000000000000a1" : {
            "basic" : {
                "driver" : "softrouter"
            }
        }
    },
    "ports" : {
        "of:00000000000000a1/1" : {
            "interfaces" : [
                {
                    "name" : "router",
                    "ips"  : [ "10.0.1.3/24" ],
                    "mac"  : "00:00:00:00:00:01"
                }
            ]
        }
    },
    "apps" : {
        "org.onosproject.router" : {
            "router":{
                "controlPlaneConnectPoint" : "of:00000000000000a1/2",
                "ospfEnabled" : "true",
                "interfaces" : ["router"]
            }
        }
    }
}
```

9. Upload ONOS Network Configuration

- In ONOS bash, use *"curl"* to upload test.json via ONOS REST API:

```
/# curl --user onos:rocks –X POST –H “Content-Type: application/json” \
       http://127.0.0.1:8181/onos/v1/network/configuration/ \
       -d @/path/to/test.json
```

10. Check Results



- If all settings are correct, you should be able to:
  - See exchanged BGP routes on both Speaker and Router in zebra or bgpd:
  - In Speaker zebra:

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

B>* 192.168.0.0/24 [20/0] via 10.0.1.1, vethSpeaker1Ovs, 1d20h24m
```

  - In Speaker bgpd:

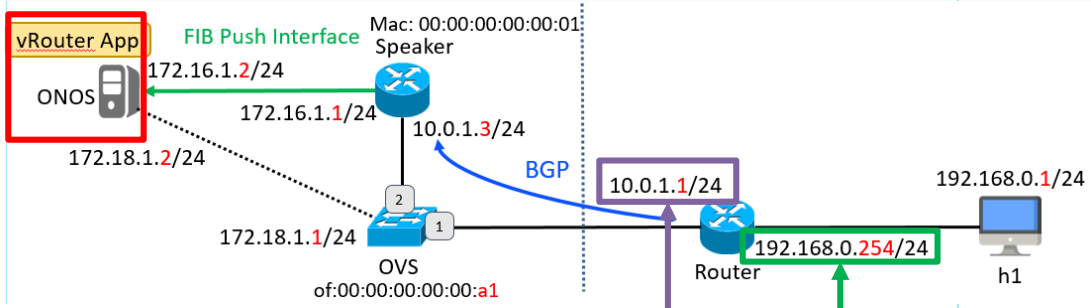
```
Speaker1bgp> show ip bgp summary
BGP router identifier 10.0.1.3, local AS number 65000
RIB entries 3, using 336 bytes of memory
Peers 1, using 4568 bytes of memory

Neighbor        V         AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.0.1.1        4 65001   13912   13919        0    0    0 11:28:11        1

Total number of neighbors 1
```

10. Check Results – vRouter FPM

- Finally, we can check if ONOS vRouter received and stored BGP routes from Speaker FIB push successfully
- In ONOS CLI:

```
karaf@root > routes
```

  - If Speaker has sent routes to ONOS, they should be visible here




```
karaf@root > routes
B: Best route, R: Resolved route

Table: ipv4
B R   Network             Next Hop         Source (Node)
> *   192.168.0.0/24      10.0.1.1         FPM (172.17.0.2)
    Total: 1

Table: ipv6
B R   Network             Next Hop         Source (Node)
    Total: 0
```

ONOS IP on docker0

10 Check Results – ONOS Web GUI

- On ONOS web GUI, we can see related flow rules installed

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	42,071	220	1	ETH_TYPE:ipv4, IPV4_DST:192.168.0.0/24	imm[ETH_SRC:00:00:00:00:00:01, ETH_DST:3A:CE:35:5A:CD:FA, OUTPUT:1], cleared:false	*fibinstaller
Added	0	42,073	32768	0	IN_PORT:1, ETH_DST_MASKED:01:00:5E:00:00:00 /FF:FF:FF:80:00:00, VLAN_VID:None	transition:TABLE:1, cleared:false	*fibinstaller
Added	0	42,073	32768	0	IN_PORT:1, ETH_DST:00:00:00:00:00:01, VLAN_VID:None	transition:TABLE:1, cleared:false	*fibinstaller
Added	0	42,073	32768	0	IN_PORT:2, ETH_DST:00:00:00:00:00:01, VLAN_VID:None	transition:TABLE:1, cleared:false	*fibinstaller
Added	0	42,073	0	0	(No traffic selector criteria for this flow)	imm[NOACTION], cleared:false	*driver.SoftRouterPipeline
Added	0	42,073	0	1	(No traffic selector criteria for this flow)	imm[NOACTION], cleared:false	*driver.SoftRouterPipeline
Added	1	42,073	40002	0	IN_PORT:2, ETH_SRC:00:00:00:00:00:01, ETH_TYPE:arp, VLAN_VID:None, ARP_SPA:10.0.1.3	imm[OUTPUT:CONTROLLER, OUTPUT:1], cleared:false	*cpr
Added	1	42,073	40002	0	IN_PORT:1, ETH_TYPE:arp, VLAN_VID:None	imm[OUTPUT:CONTROLLER, OUTPUT:2], cleared:false	*cpr
Added	25,098	42,073	40001	0	IN_PORT:1, ETH_DST:00:00:00:00:00:01, ETH_TYPE:ipv4, VLAN_VID:None, IPV4_DST:10.0.1.3/32	imm[OUTPUT:2], cleared:false	*cpr
Added	0	42,073	40001	0	IN_PORT:1, ETH_TYPE:ipv4, VLAN_VID:None, IP_PROTO:89	imm[OUTPUT:2], cleared:false	*cpr
Added	24,315	42,073	40001	0	IN_PORT:2, ETH_SRC:00:00:00:00:00:01, ETH_TYPE:ipv4, VLAN_VID:None, IPV4_SRC:10.0.1.3/32	imm[OUTPUT:1], cleared:false	*cpr

Remote subnet rule

Control plane forwarding rules

11. Clean Up the Environment

- After each experiment, we need to clean up the environment on VM
- Remove Containers:

```
~$ sudo docker rm –f <Container-1> <Container-2>
```

```
demo@SDN-NFV:~$ sudo docker rm -f ONOS Speaker OVS Router h1
```

  - -f: force

- List netns under /run/netns:

```
~$ ls –A /run/netns
```

```
demo@SDN-NFV:~$ ls -A /run/netns
28178  28366  28486  28610  28739  28859  28979  29092  29208
```

- Remove all netns if any result list above:

```
~$ sudo rm /run/netns/*
```

- Remove all docker networks:

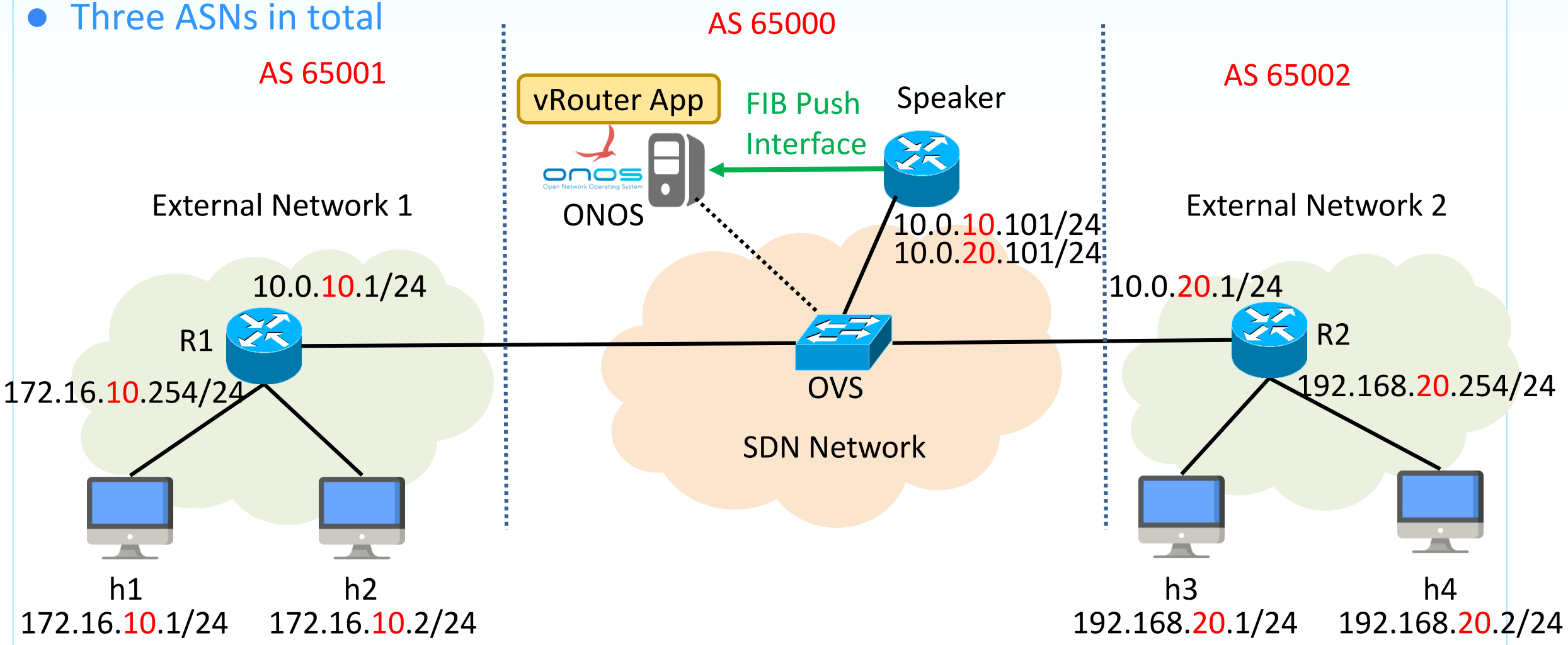
```
~$ sudo docker network rm br
```

Outline

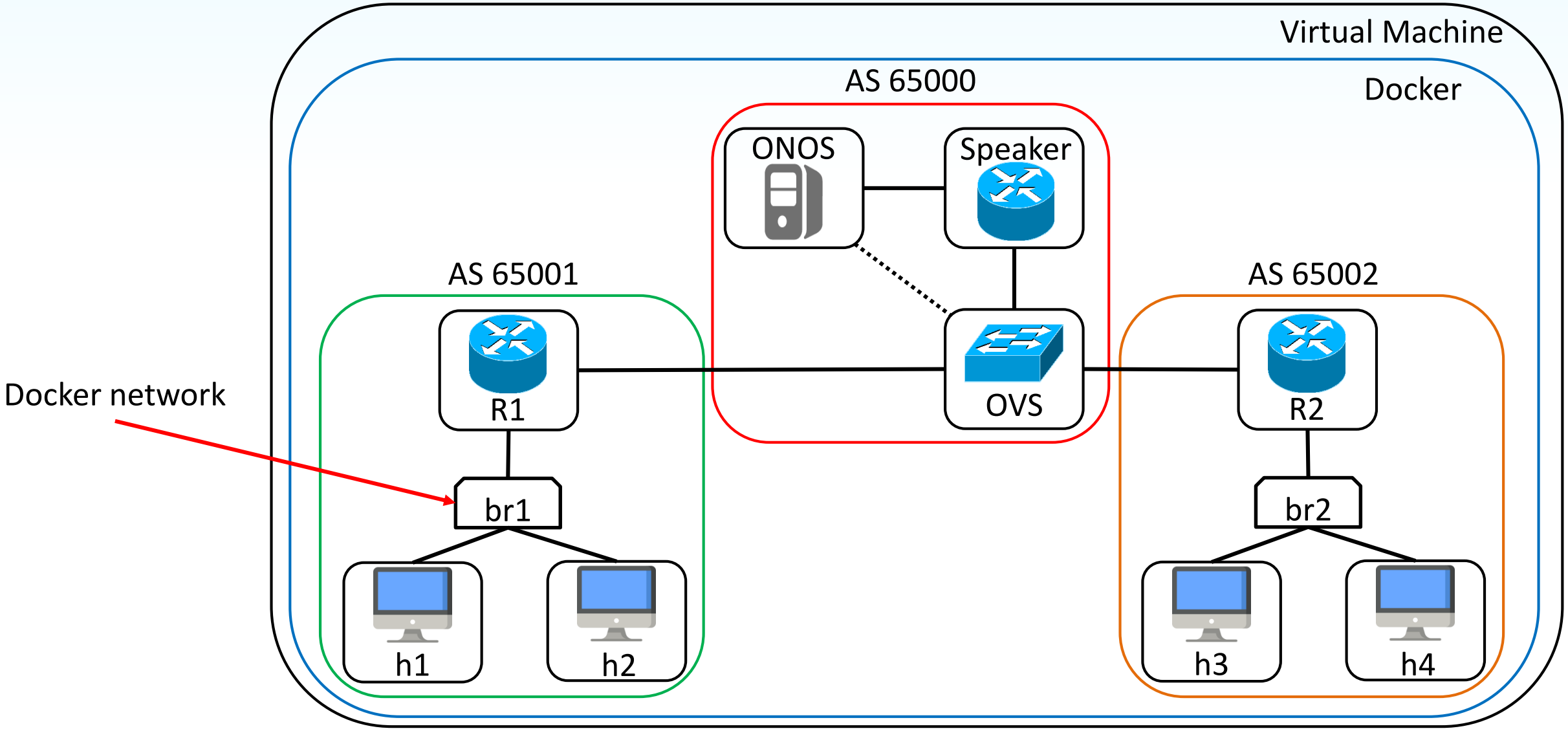
- Introduction
- ONOS vRouter App
- Environment Setup Example
- Project Requirements
  - Scenario
  - Requirements
  - Submission
- Reference

Scenario - Transit Network

- ONOS, routers, speaker, and hosts are all Docker containers in one VM
- Three ASNs in total



Underlay Topology



Report and Demo

- Report (50%)
  - Answer Questions (70%)
    1. Show topology with IP addresses, interfaces and ASNs (10%)
    2. Capture BGP packets (sent/received) on Speaker-OVS link (20%)
       - Use wireshark or tcpdump to capture and show screenshots
    3. Telnet zebra and bgpd daemons on Speaker and show BGP summary routes (10%)
    4. Show command **"routes"** result in ONOS CLI (10%)
    5. h1 can ping h4 successfully (show result with screenshot) (10%)
    6. Write down what you have learned or solved (10%)
  - *<Your_network_config.json>* file for this project (30%)
- Demo (50%)

About Submission

- Files
  - Report: FinalProject_<studentID>.pdf
  - Network configuration: <*Your_network_config.json*> file for this project
    - No naming restriction for json file
- Submission
  - Compress all files to "FinalProject_<studentID>.zip" and upload it to e3
  - Report and zip file with incorrect file name or format subjects to not scoring

References

- Namespaces:
  - https://man7.org/linux/man-pages/man7/namespaces.7.html
  - https://man7.org/linux/man-pages/man8/ip-netns.8.html
- Veth:
  - https://man7.org/linux/man-pages/man4/veth.4.html
- ONOS vRouter:
  - https://wiki.onosproject.org/display/ONOS/vRouter
- ovs-vsctl:
  - http://www.openvswitch.org/support/dist-docs/ovs-vsctl.8.txt
- Docker Network:
  - https://docs.docker.com/engine/reference/commandline/network_create/
  - https://docs.docker.com/engine/reference/commandline/network_connect/