



## OPEN VIRTUAL SWITCH LAB SERIES

**Book Version: 09-30-2021**

Principal Investigator: Jorge Crichigno



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

- Lab 1: Introduction to Linux Namespaces and Open vSwitch
- Lab 2: Introduction to Mininet
- Lab 3: Introduction to Open vSwitch
- Lab 4: Open vSwitch Flow Table
- Exercise 1: OpenFlow Basic Operations
- Lab 5: Implementing Routing in Open vSwitch
- Lab 6: Implementing Routing using Multiple Flow Tables
- Exercise 2: Implement Routing using Multiple Flow Tables
- Lab 7: Configuring Stateless Firewall using ACLs
- Lab 8: Configuring Stateful Firewall using Connection Tracking
- Exercise 3: Configuring Stateless and Stateful Firewalls in Open vSwitch
- Lab 9: Quality of Service (QoS)
- Exercise 4: Configuring Quality of Service (QoS)
- Lab 10: Open vSwitch Database Management Protocol (OVSDB)
- Lab 11: Open vSwitch Kernel Datapath
- Lab 12: Implementing Virtual Local Area Network (VLANs) in Open vSwitch
- Lab 13: VLAN trunking in Open vSwitch
- Exercise 5: Configuring Virtual Local Area Network (VLAN)
- Lab 14: Configuring GRE Tunnel
- Lab 15: Configuring IPsec Tunnel



## OPEN VIRTUAL SWITCH

### Lab 1: Introduction to Linux namespaces and Open vSwitch

Document Version: **03-31-2020**



Award 1829698  
“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    Linux namespaces .....	4
1.2    Open vSwitch networking with Linux namespaces.....	4
2    Lab topology.....	5
2.1    Lab settings.....	5
3    Creating Linux namespaces and Open vSwitch .....	6
4    Linking the namespaces to the Open vSwitch .....	9
4.1    Creating <i>veth</i> peers .....	9
4.2    Attaching <i>veth</i> peer to namespaces.....	11
4.3    Attaching <i>veth</i> peer to switch s1.....	13
5    Assigning IP addresses to the hosts.....	15
6    Verifying configuration .....	16
References .....	18

## Overview

This lab discusses the concept of Linux namespaces, which is an isolated network stack in the kernel. Logically, it creates another copy of the network stack with its interfaces, routes, and firewall rules. This lab aims to configure network namespaces required to connect through one instance of Open Virtual Switch (Open vSwitch).

## Objectives

By the end of this lab, you should be able to:

1. Understand the concept of Linux namespaces.
2. Understand the basic operation of an Open vSwitch.
3. Create Linux namespaces, tunnels, Open vSwitches, and ports.
4. Verify the routes between network namespaces.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Creating Linux namespaces and Open vSwitch.
4. Section 4: Linking namespaces to the Open vSwitch.
5. Section 5: Assigning IP addresses on the hosts.
6. Section 6: Verifying configuration.

### 1      Introduction

The increasing number of services based on virtualization presents a significant change in datacenter networking. The migration from physical ports to virtual ports moved a substantial part of the workload to virtual switches in a hypervisor. In addition to what traditional software switches provide, virtual switches have been developed to improve

the performance and to provide advanced features. Some of these features include high flexibility, vendor independence, low costs, and conceptual benefits for switching without Ethernet limitations<sup>1</sup>. By consolidating multiple servers and storage devices into a single host machine, virtualization benefits the organization by reducing physical hardware.

## 1.1 Linux namespaces

Namespaces are a feature that partitions Linux resources. The Linux operating system has a single routing table and a set of network interfaces. In Linux namespaces, you can isolate the global system resources between processes. Any changes to the global resource will only be visible to other processes that are members of the namespace but are invisible to other processes. Linux namespaces provide independent instances of networks that enable network isolation and independent operations. Each network namespace has its own networking devices, IP addresses, routing tables, and firewall rules<sup>1</sup>.

Each network namespace has its own set of IP tables (for both IPv4 and IPv6). These tables can apply different security rules to flows that share the same IP address but have different namespaces and forwarding routes.

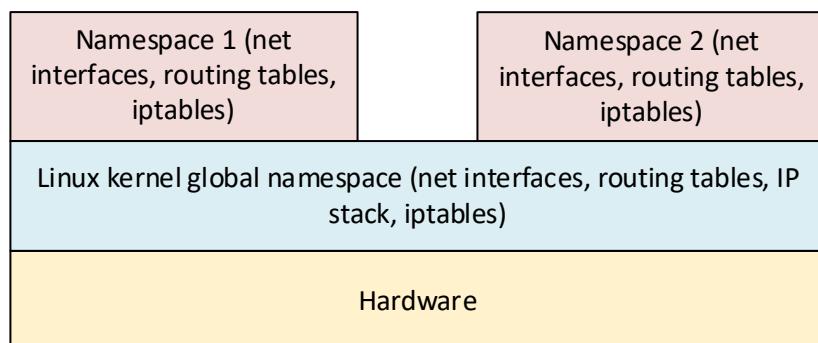


Figure 1. Linux namespace architecture.

Consider Figure 1. This figure shows network isolation and independent operation of multiple network instances. Each network namespace is independent, having individual processes with separate network interfaces, routing tables, and IP tables.

## 1.2 Open vSwitch networking with Linux namespaces

Open vSwitch is an open-source software switch used in virtualized environments. It can forward traffic between different virtual machines (VMs) on the same physical host. It includes most of the physical switch features and provides some advanced features such as NetFlow, IPFIX, and sFlow<sup>2</sup>. It can also operate entirely in userspace without assistance from a kernel module.

The Open vSwitch daemon runs on the root namespace. It listens to netlink event messages from all networking namespaces. Each netlink message contains the network namespace identifier as ancillary data used to match the event to the corresponding port<sup>3</sup>.

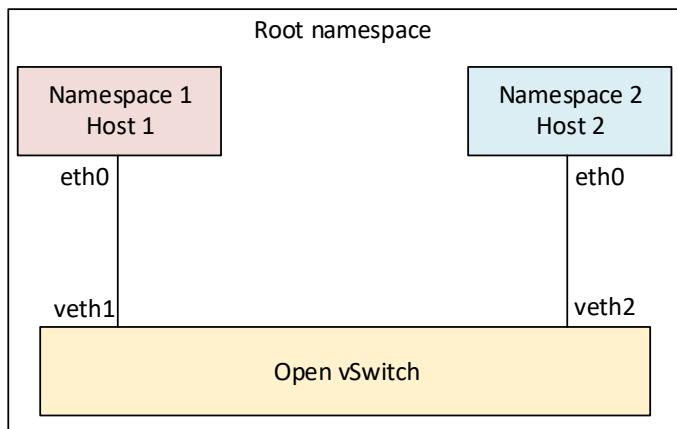


Figure 2. Open vSwitch connected with Linux namespace hosts.

Consider Figure 2. This figure shows two namespaces connected via Open vSwitch. The network interface within namespaces interconnects with Open vSwitch via virtual Ethernet (*veth*) port peer. *Veth* ports are equivalent to pair of physical Ethernet interfaces. These ports are considered as tunnels that run on the link layer (layer 2) and connect Open vSwitch with the namespaces. Each namespace has its own network interface , and the connection with the Open vSwitch is done through these interfaces.

## 2 Lab topology

Consider Figure 3. Two network namespaces, *h1* and *h2*, are linked to one instance of Open vSwitch, *s1*. Switch *s1* acts as a layer 2 switch enabling connectivity between two namespaces. The network environments of hosts *h1* and *h2* are isolated from each other and the root namespace.

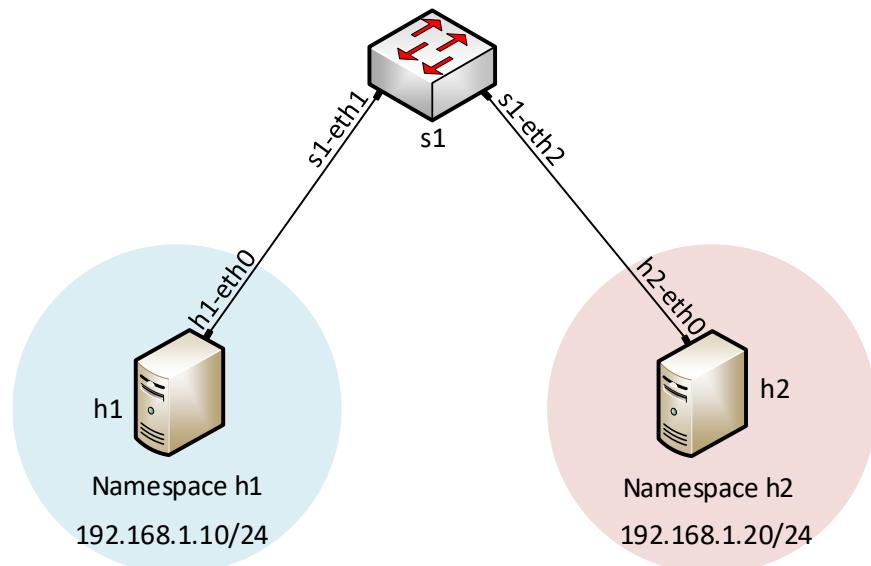


Figure 3. Lab topology.

### 2.1 Lab settings

The hosts should be configured according to Table 2.

Table 2. Topology information.

Host	Interface	IP Address	Subnet
h1	h1-eth0	192.168.1.10	/24
h2	h2-eth0	192.168.1.20	/24

### 3 Creating Linux namespaces and Open vSwitch

In this section, you will create two Linux namespaces called h1 and h2. Additionally, you will create a virtual switch (i.e., Open vSwitch), s1.

**Step 1.** Open the Linux terminal.



Figure 4. Opening Linux terminal.

**Step 2.** Type the following command in the terminal to elevate the entire command session to root privileges. The obtained privileges are necessary to run the commands needed to create Linux namespaces and the Open vSwitch. When prompted for a password, type password, password.

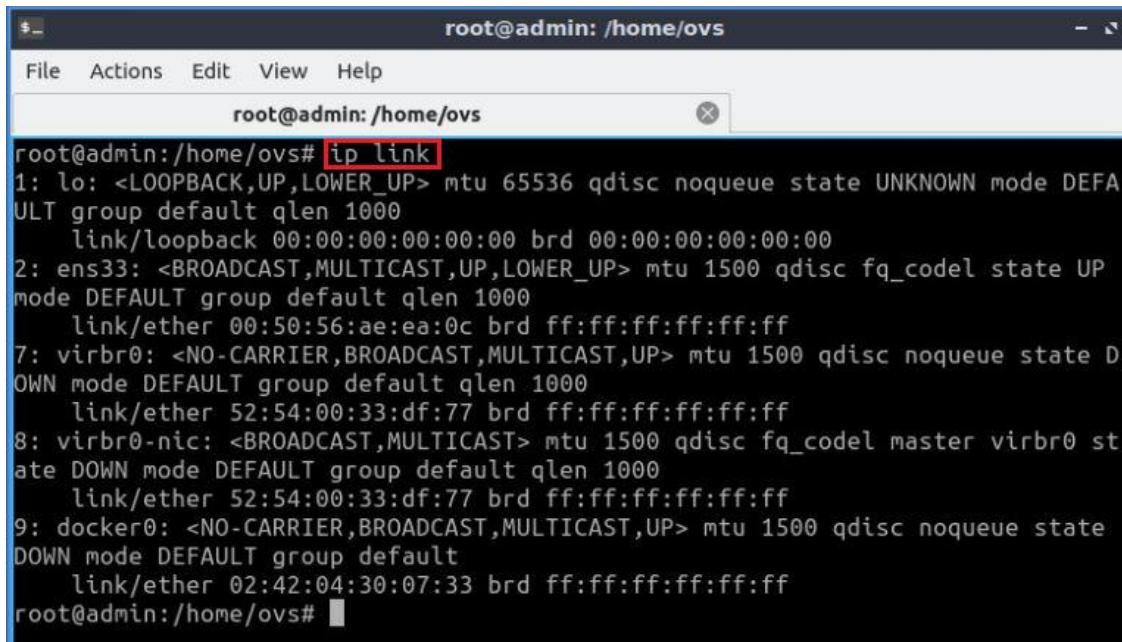
```
sudo su
```

```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo su
[sudo] password for ovs: [REDACTED]
```

Figure 5. Elevating command session to root privileges.

**Step 3.** Type the following command to display all the interfaces that exist in the root namespace.

```
ip link
```

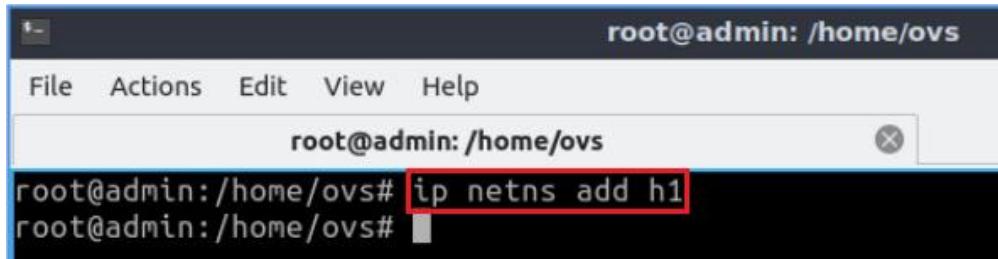


```
root@admin:/home/ovs# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFALULT group default qlen 1000
    link/loopback brd 00:00:00:00:00:00
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 00:50:56:ae:ea:0c brd ff:ff:ff:ff:ff:ff
7: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default qlen 1000
    link/ether 52:54:00:33:df:77 brd ff:ff:ff:ff:ff:ff
8: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN mode DEFAULT group default qlen 1000
    link/ether 52:54:00:33:df:77 brd ff:ff:ff:ff:ff:ff
9: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:04:30:07:33 brd ff:ff:ff:ff:ff:ff
root@admin:/home/ovs#
```

Figure 6. Displaying root namespace interfaces.

**Step 4.** Type the following command to create a namespace, h1.

```
ip netns add h1
```

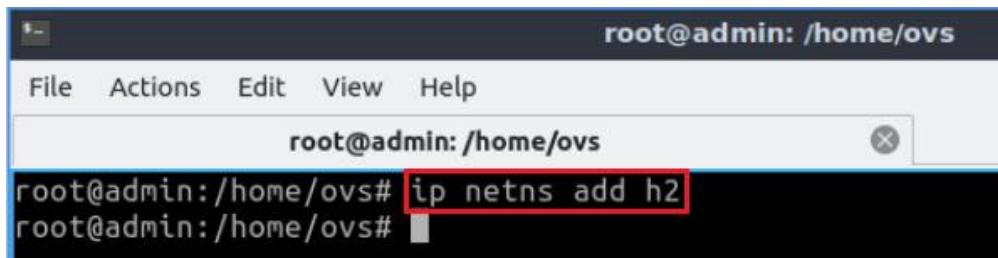


```
root@admin:/home/ovs# ip netns add h1
root@admin:/home/ovs#
```

Figure 7. Creating a namespace.

**Step 5.** Type the following command to create another namespace, h2.

```
ip netns add h2
```

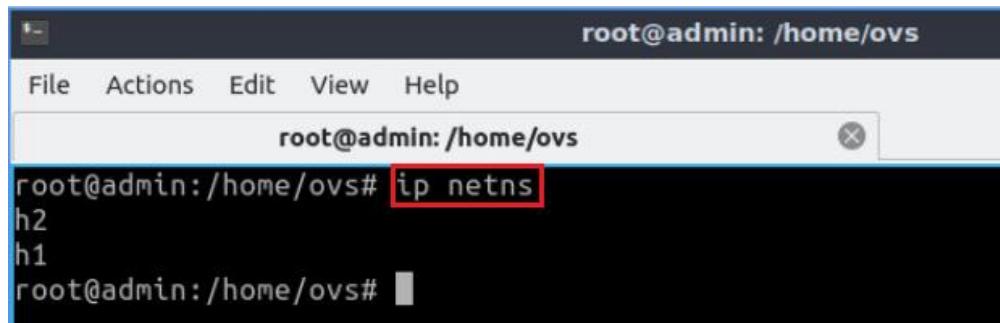


```
root@admin:/home/ovs# ip netns add h2
root@admin:/home/ovs#
```

Figure 8. Creating a namespace.

**Step 6.** Type the following command to display all the available namespaces in the kernel.

```
ip netns
```



The screenshot shows a terminal window titled "root@admin: /home/ovs". It has two tabs: "root@admin: /home/ovs" and "root@admin: /home/ovs# ip netns". The command "ip netns" is being typed in the second tab. The output shows two namespaces: h2 and h1.

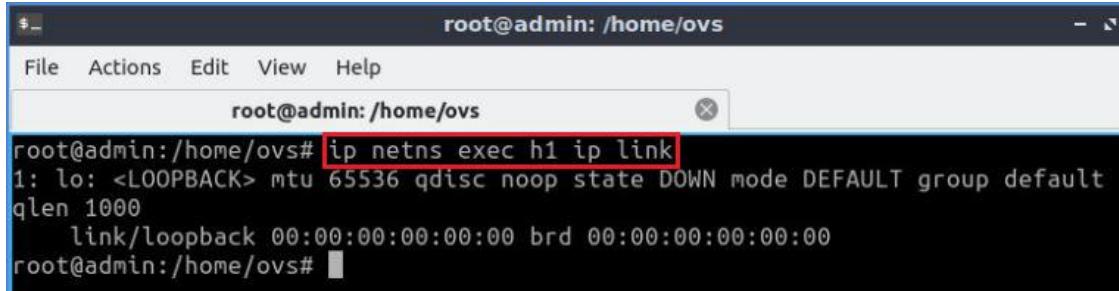
```
root@admin:/home/ovs# ip netns
h2
h1
root@admin:/home/ovs#
```

Figure 9. Displaying all the available namespaces.

Consider the figure above. The figure shows the list of namespaces, h1 and h2.

**Step 7.** Type the following command to display all the interfaces that exist in namespace h1.

```
ip netns exec h1 ip link
```



The screenshot shows a terminal window titled "root@admin: /home/ovs". It has one tab: "root@admin: /home/ovs". The command "ip netns exec h1 ip link" is being typed. The output shows the single loopback interface "1: lo" in namespace h1.

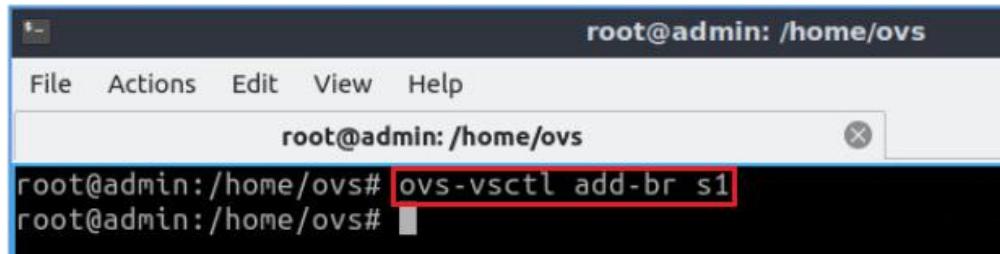
```
root@admin:/home/ovs# ip netns exec h1 ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
root@admin:/home/ovs#
```

Figure 10. Displaying h1 namespace interfaces.

Consider the figure above. Only the loopback interface is listed in the namespace h1.

**Step 8.** Type the following command to create a virtual switch, s1.

```
ovs-vsctl add-br s1
```



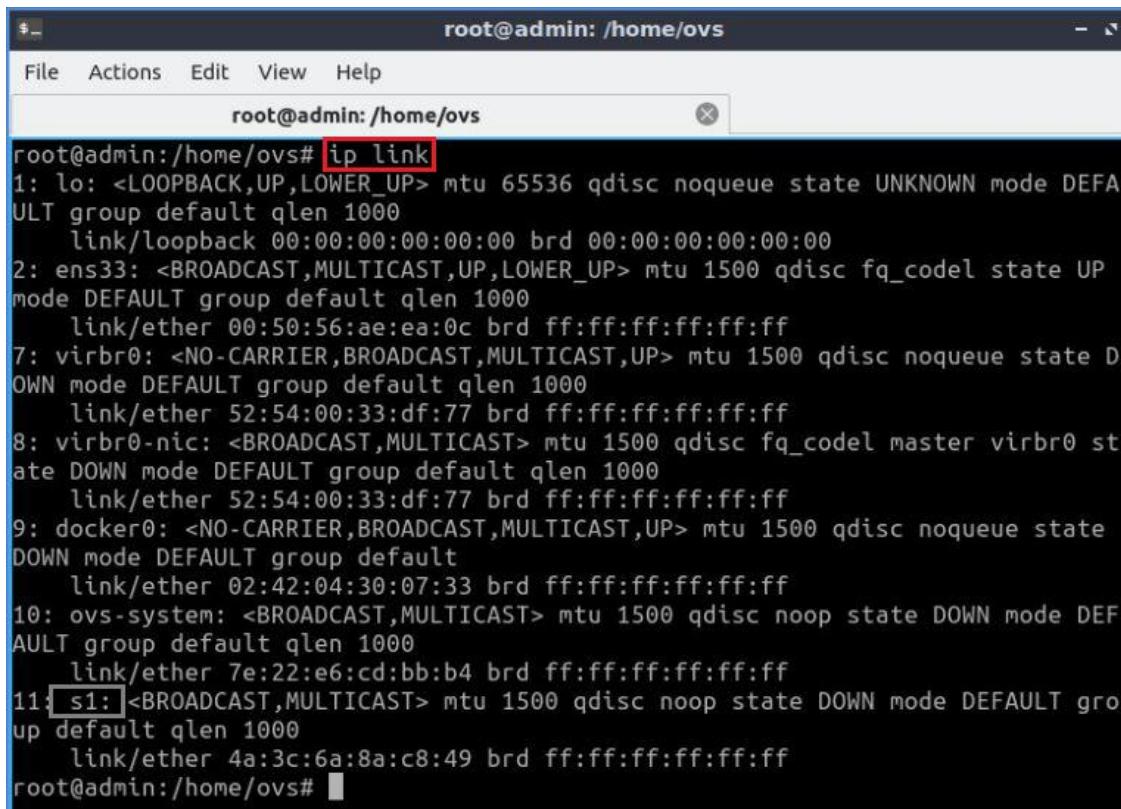
The screenshot shows a terminal window titled "root@admin: /home/ovs". It has one tab: "root@admin: /home/ovs". The command "ovs-vsctl add-br s1" is being typed. The output shows the creation of the virtual switch s1.

```
root@admin:/home/ovs# ovs-vsctl add-br s1
root@admin:/home/ovs#
```

Figure 11. Creating a virtual switch.

**Step 9.** Type the following command to display all the interfaces that exist in the root namespace.

```
ip link
```



```
root@admin:/home/ovs# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFALULT group default qlen 1000
    link/loopback brd 00:00:00:00:00:00
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 00:50:56:ae:ea:0c brd ff:ff:ff:ff:ff:ff
7: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default qlen 1000
    link/ether 52:54:00:33:df:77 brd ff:ff:ff:ff:ff:ff
8: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN mode DEFAULT group default qlen 1000
    link/ether 52:54:00:33:df:77 brd ff:ff:ff:ff:ff:ff
9: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:04:30:07:33 brd ff:ff:ff:ff:ff:ff
10: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFALULT group default qlen 1000
    link/ether 7e:22:e6:cd:bb:b4 brd ff:ff:ff:ff:ff:ff
11: s1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 4a:3c:6a:8a:c8:49 brd ff:ff:ff:ff:ff:ff
root@admin:/home/ovs#
```

Figure 12. Displaying root namespace interfaces.

Consider the figure above. Open vSwitch s1 is added to the root namespace.

At this point, the network namespaces and the Open vSwitch have been created. However, the namespaces are not connected to the Open vSwitch.

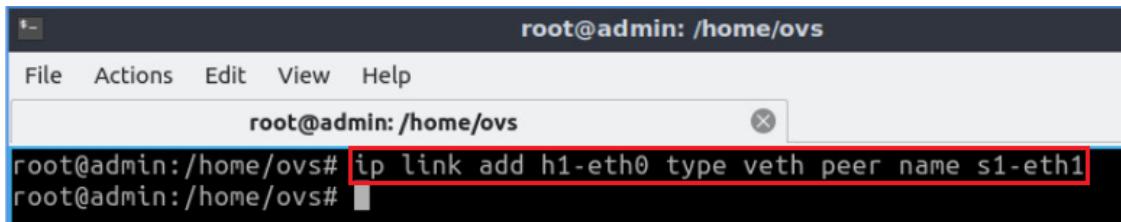
## 4 Linking the namespaces to the Open vSwitch

In this section, in order to link the namespaces to the switch s1, you will create two virtual Ethernet (*veth*) peers. The *veth* peer will act as a tunnel between namespaces and the switch. One end point of the *veth* peer will be attached to the particular namespace whereas, the other end point will be connected to the switch s1.

### 4.1 Creating *veth* peers

**Step 1.** In order to create a *veth* peer, type the following command.

```
ip link add h1-eth0 type veth peer name s1-eth1
```



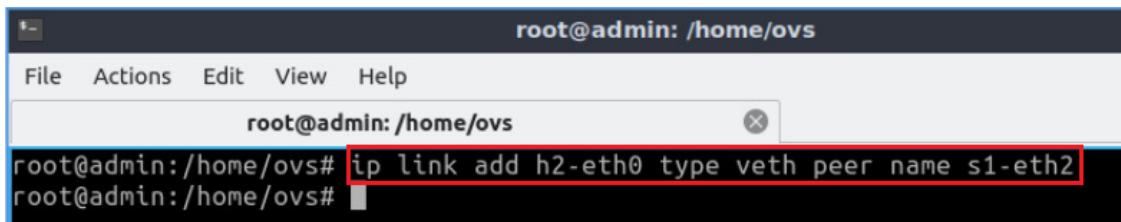
The screenshot shows a terminal window titled "root@admin: /home/ovs". The menu bar includes "File", "Actions", "Edit", "View", and "Help". A tab labeled "root@admin: /home/ovs" is open. The command "ip link add h1-eth0 type veth peer name s1-eth1" is highlighted with a red box. The prompt "root@admin:/home/ovs#" is visible at the bottom.

Figure 13. Creating *veth* peer.

Consider the figure above. The *veth* peer has been created but not connected to the namespaces and the switch. Interface *h1-eth0* will be attached to namespace *h1*, and the other point, *s1-eth1*, will be attached to the switch *s1* in the following section.

**Step 2.** To create a *veth* peer, type the following command.

```
ip link add h2-eth0 type veth peer name s1-eth2
```



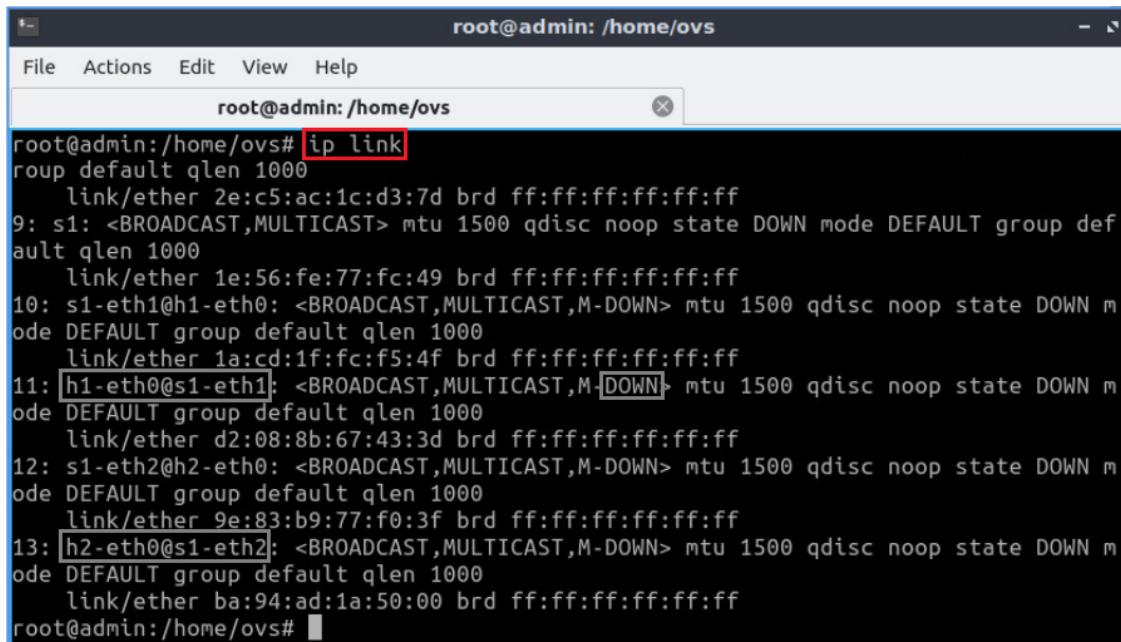
The screenshot shows a terminal window titled "root@admin: /home/ovs". The menu bar includes "File", "Actions", "Edit", "View", and "Help". A tab labeled "root@admin: /home/ovs" is open. The command "ip link add h2-eth0 type veth peer name s1-eth2" is highlighted with a red box. The prompt "root@admin:/home/ovs#" is visible at the bottom.

Figure 14. Creating *veth* peer.

Consider the figure above. The *veth* peer has been created but not connected to the namespaces and the switch. Interface *h2-eth0* will be attached to namespace *h2*, and the other point *s1-eth2* will be attached to the switch *s1* in the following section.

**Step 3.** Type the following command to display all the interfaces available in the root namespace.

```
ip link
```



```
root@admin:/home/ovs# ip link
1: lo: <LOOPBACK,NOQUEUE,NOFCS> mtu 1500 qdisc nofcs state UP mode DEFAULT group default qlen 1
    link/loopback brd ff:ff:ff:ff:ff:ff
2: eth0: <NOQUEUE,NOFCS> mtu 1500 qdisc nofcs state UP mode DEFAULT group default qlen 1
    link/ether 2e:c5:ac:1c:d3:7d brd ff:ff:ff:ff:ff:ff
3: eth1: <NOQUEUE,NOFCS> mtu 1500 qdisc nofcs state UP mode DEFAULT group default qlen 1
    link/ether 1e:56:fe:77:fc:49 brd ff:ff:ff:ff:ff:ff
4: eth2: <NOQUEUE,NOFCS> mtu 1500 qdisc nofcs state UP mode DEFAULT group default qlen 1
    link/ether d2:08:8b:67:43:3d brd ff:ff:ff:ff:ff:ff
5: eth3: <NOQUEUE,NOFCS> mtu 1500 qdisc nofcs state UP mode DEFAULT group default qlen 1
    link/ether 9e:83:b9:77:f0:3f brd ff:ff:ff:ff:ff:ff
6: eth4: <NOQUEUE,NOFCS> mtu 1500 qdisc nofcs state UP mode DEFAULT group default qlen 1
    link/ether ba:94:ad:1a:50:00 brd ff:ff:ff:ff:ff:ff
7: h1-eth0@h1-eth1: <NOQUEUE,NOFCS,BROADCAST,MULTICAST> mtu 1500 qdisc nofcs state DOWN mode DEFAULT group default qlen 1
    link/ether 1a:cd:1f:fc:f5:4f brd ff:ff:ff:ff:ff:ff
8: h2-eth0@h2-eth1: <NOQUEUE,NOFCS,BROADCAST,M-DOWN> mtu 1500 qdisc nofcs state DOWN mode DEFAULT group default qlen 1
    link/ether h2-eth0@h2-eth1 brd ff:ff:ff:ff:ff:ff
9: h1-eth1@h1-eth0: <NOQUEUE,NOFCS,BROADCAST,M-DOWN> mtu 1500 qdisc nofcs state DOWN mode DEFAULT group default qlen 1
    link/ether h1-eth1@h1-eth0 brd ff:ff:ff:ff:ff:ff
10: h2-eth1@h2-eth0: <NOQUEUE,NOFCS,BROADCAST,M-DOWN> mtu 1500 qdisc nofcs state DOWN mode DEFAULT group default qlen 1
    link/ether h2-eth1@h2-eth0 brd ff:ff:ff:ff:ff:ff
11: h1-eth0@h1-eth1: <NOQUEUE,NOFCS,BROADCAST,M-DOWN> mtu 1500 qdisc nofcs state DOWN mode DEFAULT group default qlen 1
    link/ether h1-eth0@h1-eth1 brd ff:ff:ff:ff:ff:ff
12: h2-eth0@h2-eth1: <NOQUEUE,NOFCS,BROADCAST,M-DOWN> mtu 1500 qdisc nofcs state DOWN mode DEFAULT group default qlen 1
    link/ether h2-eth0@h2-eth1 brd ff:ff:ff:ff:ff:ff
13: h1-eth1@h1-eth0: <NOQUEUE,NOFCS,BROADCAST,M-DOWN> mtu 1500 qdisc nofcs state DOWN mode DEFAULT group default qlen 1
    link/ether h1-eth1@h1-eth0 brd ff:ff:ff:ff:ff:ff
14: h2-eth1@h2-eth0: <NOQUEUE,NOFCS,BROADCAST,M-DOWN> mtu 1500 qdisc nofcs state DOWN mode DEFAULT group default qlen 1
    link/ether h2-eth1@h2-eth0 brd ff:ff:ff:ff:ff:ff
root@admin:/home/ovs#
```

Figure 15. Displaying root namespace interfaces.

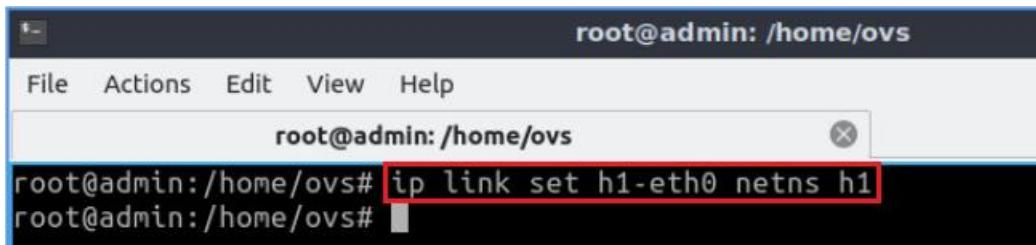
Consider the figure above. Both peers are visible in the root namespace.

## 4.2 Attaching *veth* peer to namespaces

In this section, you will attach *h1-eth0* to namespace *h1* and *h2-eth0* to namespace *h2*.

**Step 1.** Type the following command to link one of the peers, *h1-eth0*, to the namespace *h1*.

```
ip link set h1-eth0 netns h1
```



```
root@admin:/home/ovs# ip link set h1-eth0 netns h1
root@admin:/home/ovs#
```

Figure 16. Linking namespace *h1* to *veth* peer *h1-eth0*.

Another peer *s1-eth1* will be attached to switch *s1* in the following section.

**Step 2.** Type the following command to display all the interfaces available in the root namespace.

```
ip link
```

The screenshot shows a terminal window titled "root@admin: /home/ovs". The command "ip link" is being run. The output lists various network interfaces, including "link/ether 02:42:94:86:29:e2 brd ff:ff:ff:ff:ff:ff" (mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000), "link/ether 2e:c5:ac:1c:d3:7d brd ff:ff:ff:ff:ff:ff" (mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000), "link/ether 1e:56:fe:77:fc:49 brd ff:ff:ff:ff:ff:ff" (mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000), "link/ether 1a:cd:1f:fc:f5:4f brd ff:ff:ff:ff:ff:ff" (link-netns h1), "link/ether 9e:83:b9:77:f0:3f brd ff:ff:ff:ff:ff:ff" (link-netns h1), and "link/ether ba:94:ad:1a:50:00 brd ff:ff:ff:ff:ff:ff". The interface "link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00" is also listed.

```
root@admin:/home/ovs# ip link
1: link/ether 02:42:94:86:29:e2 brd ff:ff:ff:ff:ff:ff
2: link/ether 2e:c5:ac:1c:d3:7d brd ff:ff:ff:ff:ff:ff
3: link/ether 1e:56:fe:77:fc:49 brd ff:ff:ff:ff:ff:ff
4: link/ether 1a:cd:1f:fc:f5:4f brd ff:ff:ff:ff:ff:ff link-netns h1
5: link/ether 9e:83:b9:77:f0:3f brd ff:ff:ff:ff:ff:ff
6: link/ether ba:94:ad:1a:50:00 brd ff:ff:ff:ff:ff:ff
root@admin:/home/ovs#
```

Figure 17. Displaying root namespace interfaces.

Consider the figure above. The interface, *h1-eth0*, is not available in the root namespace anymore.

**Step 3.** Type the following command to display all the interfaces available in the namespace *h1*.

```
ip netns exec h1 ip link
```

The screenshot shows a terminal window titled "root@admin: /home/ovs". The command "ip netns exec h1 ip link" is being run. The output shows the interfaces available in the "h1" namespace, including "link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00" and "link/ether d2:08:8b:67:43:3d brd ff:ff:ff:ff:ff:ff link-netnsid 0". The interface "link/ether 1a:cd:1f:fc:f5:4f brd ff:ff:ff:ff:ff:ff" is also listed.

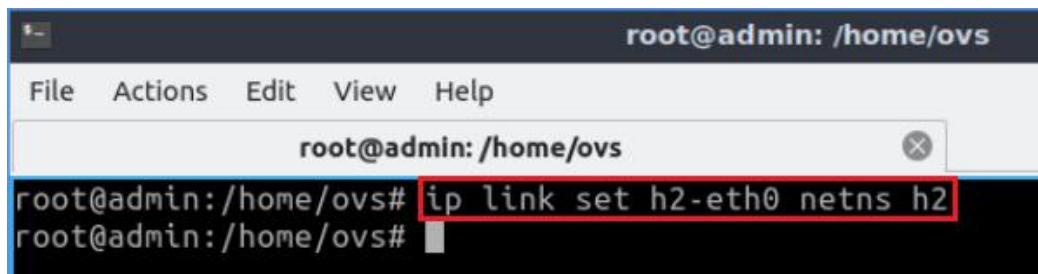
```
root@admin:/home/ovs# ip netns exec h1 ip link
1: link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: link/ether d2:08:8b:67:43:3d brd ff:ff:ff:ff:ff:ff link-netnsid 0
root@admin:/home/ovs#
```

Figure 18. Displaying namespace *h1* interfaces.

Consider the figure above. The interface *h1-eth0* belongs to the namespace *h1*.

**Step 4.** Type the following command to link one of the peers, *h2-eth0*, to the namespace *h2*.

```
ip link set h2-eth0 netns h2
```



```
root@admin: /home/ovs
File Actions Edit View Help
root@admin: /home/ovs
root@admin:/home/ovs# ip link set h2-eth0 netns h2
root@admin:/home/ovs#
```

Figure 19. Linking namespace h2 to *veth* peer *h2-eth0*.

Another peer *s1-eth2* will be attached to switch *s1* in the following section.

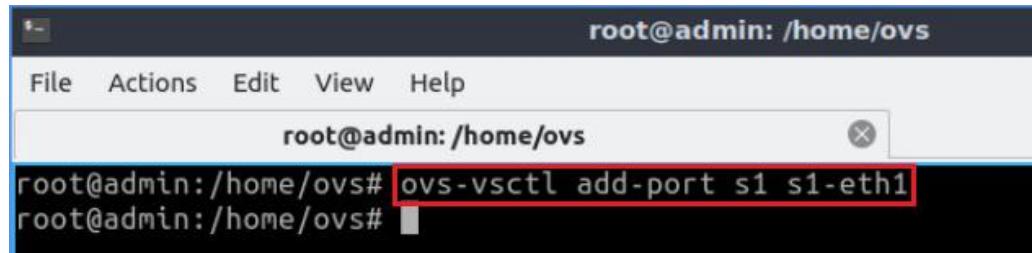
Repeat steps 2 and 3 to verify that interface *h2-eth0* is not in the root namespace anymore.

#### 4.3 Attaching *veth* peer to switch *s1*

In this section, you will link *s1-eth1* and *s1-eth2* to the switch *s1*.

**Step 1.** Type the following command to connect *veth* peer *s1-eth1* to switch *s1*.

```
ovs-vsctl add-port s1 s1-eth1
```

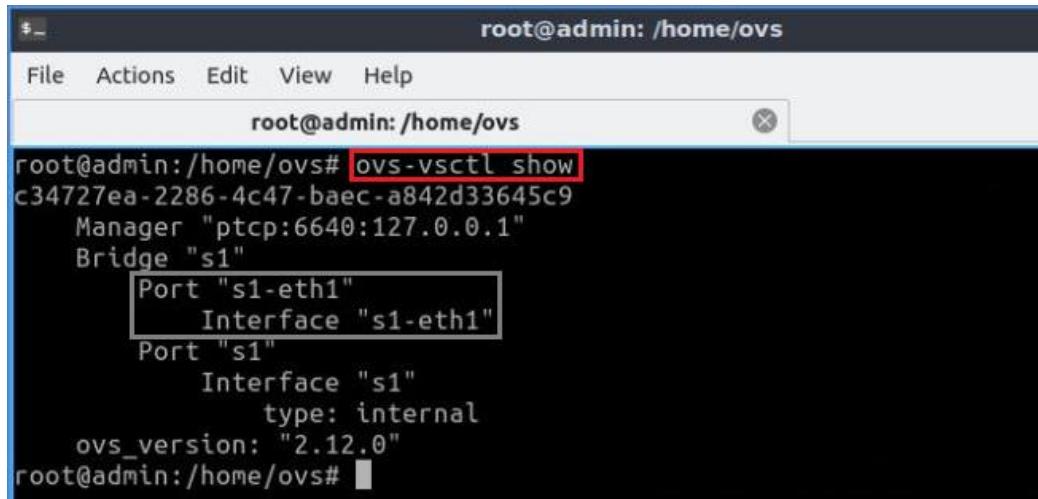


```
root@admin: /home/ovs
File Actions Edit View Help
root@admin: /home/ovs
root@admin:/home/ovs# ovs-vsctl add-port s1 s1-eth1
root@admin:/home/ovs#
```

Figure 20. Linking switch *s1* to *veth* peer *s1-eth1*.

**Step 2.** Type the following command to display switch *s1* configuration.

```
ovs-vsctl show
```



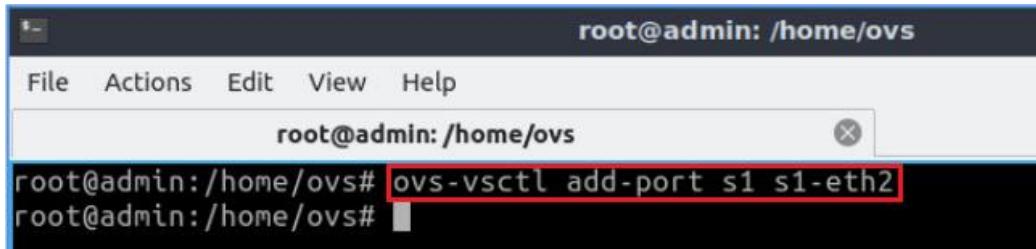
```
root@admin: /home/ovs
File Actions Edit View Help
root@admin: /home/ovs
root@admin:/home/ovs# ovs-vsctl show
c34727ea-2286-4c47-baec-a842d33645c9
  Manager "ptcp:6640:127.0.0.1"
  Bridge "s1"
    Port "s1-eth1"
      Interface "s1-eth1"
    Port "s1"
      Interface "s1"
        type: internal
  ovs_version: "2.12.0"
root@admin:/home/ovs#
```

Figure 21. Displaying Open vSwitch configuration.

Consider the figure above. You will notice that *s1-eth1* is connected to Open vSwitch *s1*.

**Step 3.** Type the following command to connect *veth* peer *s1-eth2* to switch *s1*.

```
ovs-vsctl add-port s1 s1-eth2
```

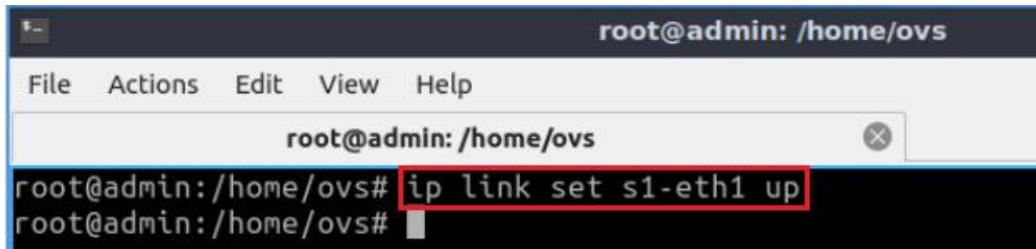


A terminal window titled "root@admin: /home/ovs". The command "ovs-vsctl add-port s1 s1-eth2" is highlighted with a red box. The prompt "root@admin:/home/ovs#" appears twice below the command line.

Figure 22. Linking switch *s1* to *veth* peer *s1-eth2*.

**Step 4.** Type the following command to activate the interface *s1-eth1*.

```
ip link set s1-eth1 up
```



A terminal window titled "root@admin: /home/ovs". The command "ip link set s1-eth1 up" is highlighted with a red box. The prompt "root@admin:/home/ovs#" appears twice below the command line.

Figure 23. Turning up the interface, *s1-eth1*.

**Step 5.** Type the following command to verify the status of the interface *s1-eth1*.

```
ip link
```

```
root@admin:/home/ovs# ip link
1: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 52:54:00:33:df:77 brd ff:ff:ff:ff:ff:ff
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 02:42:dc:d0:02:fd brd ff:ff:ff:ff:ff:ff
3: eth1: <NO-CARRIER,BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN
    link/ether e2:e5:41:f9:f4:ec brd ff:ff:ff:ff:ff:ff
4: eth2: <NO-CARRIER,BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN
    link/ether ca:97:f7:1d:e5:29 brd ff:ff:ff:ff:ff:ff
5: eth3: <NO-CARRIER,BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN
    link/ether aa:43:a4:39:08:4d brd ff:ff:ff:ff:ff:ff
6: eth4: <NO-CARRIER,BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN
    link/ether e6:1c:d8:fd:79:12 brd ff:ff:ff:ff:ff:ff
7: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 52:54:00:33:df:77 brd ff:ff:ff:ff:ff:ff
8: eth0: <NO-CARRIER,BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN
    link/ether 02:42:dc:d0:02:fd brd ff:ff:ff:ff:ff:ff
9: eth1: <NO-CARRIER,BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN
    link/ether e2:e5:41:f9:f4:ec brd ff:ff:ff:ff:ff:ff
10: eth2: <NO-CARRIER,BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN
    link/ether ca:97:f7:1d:e5:29 brd ff:ff:ff:ff:ff:ff
11: eth3: <NO-CARRIER,BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN
    link/ether aa:43:a4:39:08:4d brd ff:ff:ff:ff:ff:ff
12: eth4: <NO-CARRIER,BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN
    link/ether e6:1c:d8:fd:79:12 brd ff:ff:ff:ff:ff:ff
root@admin:/home/ovs#
```

Figure 24. Displaying root namespace interfaces.

Consider the figure above. You will notice the status of the interface *s1-eth1* has changed from *DOWN* to *UP*.

**Step 6.** Type the following command to turn up the interface *s1-eth2*.

```
ip link set s1-eth2 up
```

```
root@admin:/home/ovs# ip link set s1-eth2 up
root@admin:/home/ovs#
```

Figure 25. Turning up the interface, *s1-eth2*.

## 5 Assigning IP addresses to the hosts

**Step 1.** Type the following command to turn up the interface *h1-eth0* in the namespace *h1*.

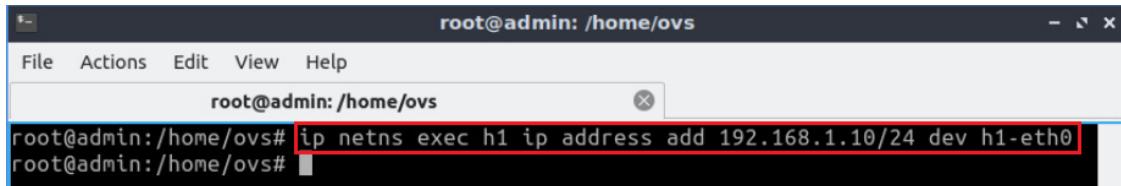
```
ip netns exec h1 ip link set dev h1-eth0 up
```

```
root@admin:/home/ovs# ip netns exec h1 ip link set dev h1-eth0 up
root@admin:/home/ovs#
```

Figure 26. Turning up the interface, *h1-eth0*.

**Step 2.** Type the following command to assign an IP address to the interface *h1-eth0*.

```
ip netns exec h1 ip address add 192.168.1.10/24 dev h1-eth0
```

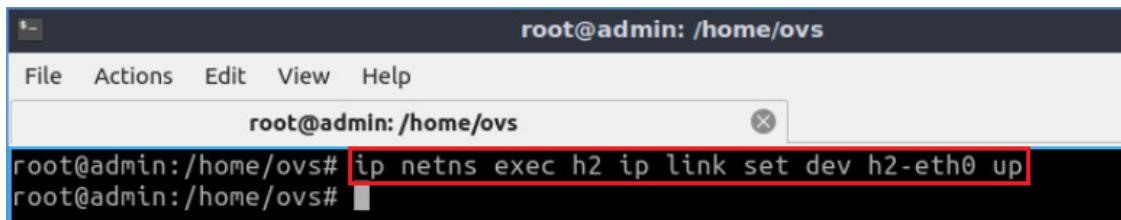


A screenshot of a terminal window titled "root@admin: /home/ovs". The window has a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu is a title bar with "root@admin: /home/ovs". The main area shows a command prompt "root@admin:/home/ovs#". A red box highlights the command "ip netns exec h1 ip address add 192.168.1.10/24 dev h1-eth0". The command is followed by a new line "root@admin:/home/ovs#".

Figure 27. Assigning IP address to the interface *h1-eth0*.

**Step 3.** Type the following command to turn up the interface *h2-eth0* in the namespace *h2*.

```
ip netns exec h2 ip link set dev h2-eth0 up
```

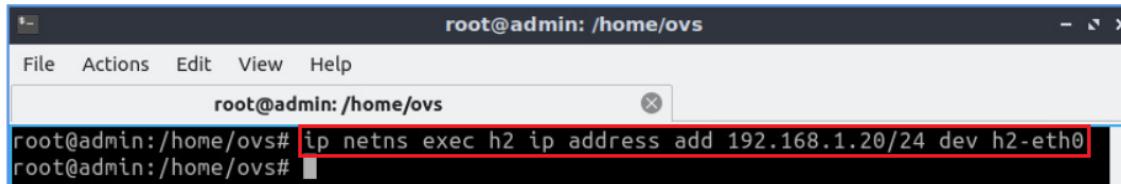


A screenshot of a terminal window titled "root@admin: /home/ovs". The window has a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu is a title bar with "root@admin: /home/ovs". The main area shows a command prompt "root@admin:/home/ovs#". A red box highlights the command "ip netns exec h2 ip link set dev h2-eth0 up". The command is followed by a new line "root@admin:/home/ovs#".

Figure 28. Turning up the interface *h2-eth0*.

**Step 4.** Type the following command to assign an IP address to interface *h2-eth0*.

```
ip netns exec h2 ip address add 192.168.1.20/24 dev h2-eth0
```



A screenshot of a terminal window titled "root@admin: /home/ovs". The window has a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu is a title bar with "root@admin: /home/ovs". The main area shows a command prompt "root@admin:/home/ovs#". A red box highlights the command "ip netns exec h2 ip address add 192.168.1.20/24 dev h2-eth0". The command is followed by a new line "root@admin:/home/ovs#".

Figure 29. Assigning IP address to the interface *h2-eth0*.

## 6 Verifying configuration

**Step 1.** Type the following command to verify the IP address on namespace *h1*. You will verify the interface status is up, and the IP address is 192.168.1.10/24.

```
ip netns exec h1 ip address
```

```
root@admin:/home/ovs# ip netns exec h1 ip address
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
11: h1-eth0@if10: <BROADCAST,MULTICAST UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether d2:08:b6:67:43:3d brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet 192.168.1.10/24 scope global h1-eth0
            valid_lft forever preferred_lft forever
        inet6 fe80::d008:bfff:fe67:433d/64 scope link
            valid_lft forever preferred_lft forever
root@admin:/home/ovs#
```

Figure 30. Verifying IP address for interface *h1-eth0*.

**Step 2.** Type the following command to display the routing table of namespace *h1*. You will notice that network 192.168.1.0/24 is available.

```
ip netns exec h1 ip route
```

```
root@admin:/home/ovs# ip netns exec h1 ip route
192.168.1.0/24 dev h1-eth0 proto kernel scope link src 192.168.1.10
root@admin:/home/ovs#
```

Figure 31. Displaying namespace *h1*'s routing table.

**Step 3.** Type the following command to display the routing table of the root namespace.

```
ip route
```

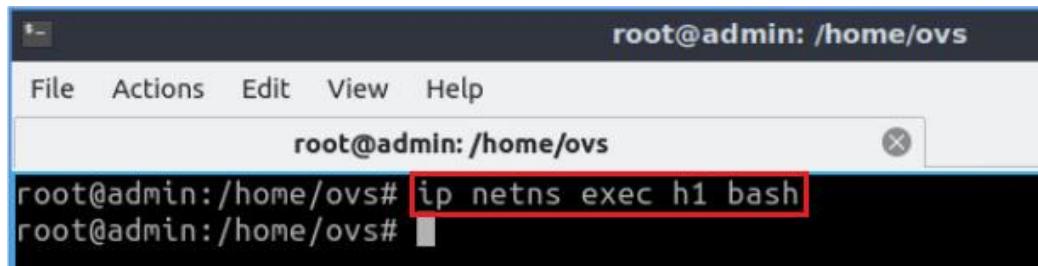
```
root@admin:/home/ovs# ip route
default via 10.173.78.1 dev ens33 proto dhcp metric 100
10.173.78.0/24 dev ens33 proto kernel scope link src 10.173.78.65 metric 100
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown
root@admin:/home/ovs#
```

Figure 32. Displaying root namespace routing table.

Consider the figure above. The root namespace is not aware of network 192.168.1.0/24. This is due to the isolation of the namespaces from the root namespace.

**Step 4.** Type the following command to start a bash shell within namespace *h1*, where you can run all the commands on the particular namespace.

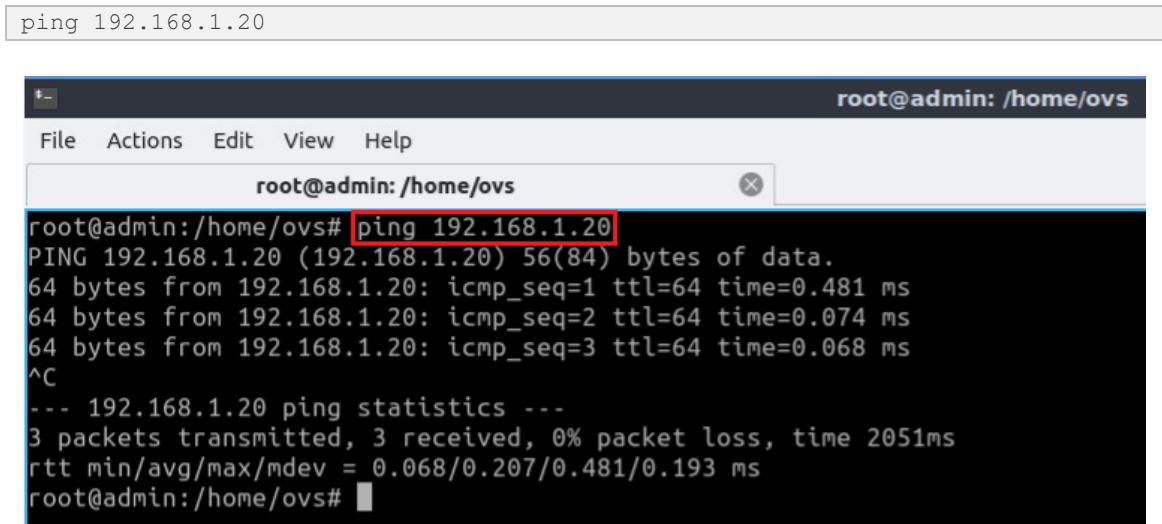
```
ip netns exec h1 bash
```



```
root@admin: /home/ovs
File Actions Edit View Help
root@admin: /home/ovs
root@admin:/home/ovs# ip netns exec h1 bash
root@admin:/home/ovs#
```

Figure 33. Starting a bash shell within namespace *h1*.

**Step 5.** Test the connectivity between namespaces *h1* and *h2* using the `ping` command. To stop the test, press `Ctrl+c`.

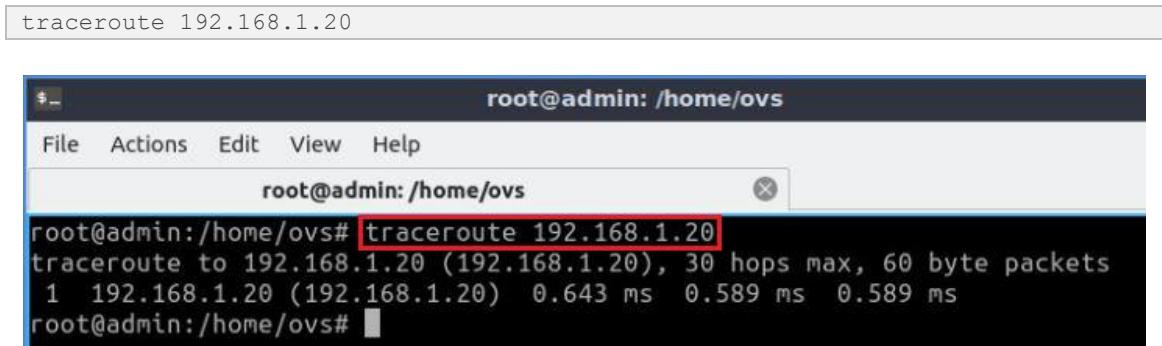


```
ping 192.168.1.20
root@admin: /home/ovs
File Actions Edit View Help
root@admin: /home/ovs
root@admin:/home/ovs# ping 192.168.1.20
PING 192.168.1.20 (192.168.1.20) 56(84) bytes of data.
64 bytes from 192.168.1.20: icmp_seq=1 ttl=64 time=0.481 ms
64 bytes from 192.168.1.20: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 192.168.1.20: icmp_seq=3 ttl=64 time=0.068 ms
^C
--- 192.168.1.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2051ms
rtt min/avg/max/mdev = 0.068/0.207/0.481/0.193 ms
root@admin:/home/ovs#
```

Figure 34. Output of the `ping` command.

The figure shows successful connectivity between two namespaces.

**Step 6.** You can also run a traceroute test between namespaces using the `traceroute` command.



```
traceroute 192.168.1.20
root@admin: /home/ovs
File Actions Edit View Help
root@admin: /home/ovs
root@admin:/home/ovs# traceroute 192.168.1.20
traceroute to 192.168.1.20 (192.168.1.20), 30 hops max, 60 byte packets
 1  192.168.1.20 (192.168.1.20)  0.643 ms  0.589 ms  0.589 ms
root@admin:/home/ovs#
```

Figure 35. Output of the `traceroute` command.

This concludes Lab 1. You can exit the Linux terminal.

## References

1. Toptal, “*Separation anxiety: A tutorial for isolating your system with Linux namespaces*”, [Online]. Available: <https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces>
2. Linux foundation, “*Open vSwitch*”, [Online]. Available: <http://openvSwitch.org>.
3. Linux foundation, “*Open vSwitch*”, [Online]. Available: <https://docs.openvswitch.org/en/latest/topics/networking-namespaces/>
4. RFC 4047, “*The open vSwitch database management protocol*”, Dec 2013.
5. IBM, “*Archived / virtual networking in Linux*”, [Online]. Available: <https://developer.ibm.com/tutorials/l-virtual-networking/>
6. Konstantin Ivanov, “*Containerization with LXC*”, Feb 2017.
7. Mininet walkthrough, [Online]. Available: <http://mininet.org>.
8. Cisco, “*Introduction to Cisco IOS Netflow – A technical overview*”, May 2012.
9. RFC 7011, “*Specification of the IP Flow information export (IPFIX) protocol for the exchange of flow information*”, Sep 2013.
10. sFlow, [Online]. Available: <https://sflow.org/>
11. Red Hat, “*What is virtualization?*”, [Online]. Available: <https://www.redhat.com/en/topics/virtualization/what-is-virtualization>



## OPEN VIRTUAL SWITCH

### Lab 2: Introduction to Mininet

Document Version: **03-31-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction to Mininet .....	3
2    Invoke Mininet using the CLI .....	5
2.1    Invoke Mininet using the default topology.....	5
2.2    Test connectivity .....	9
3    Build and emulate a network in Mininet using the GUI .....	10
3.1    Build the network topology .....	11
3.2    Test connectivity .....	13
3.3    Automatic assignment of IP addresses .....	16
3.4    Save and load a Mininet topology .....	18
4    Configure router r1 .....	19
4.1    Verify end-hosts configuration.....	20
4.2    Configure router's interface.....	21
4.3    Verify router r1 configuration .....	25
4.4    Test connectivity between end-hosts.....	26
References .....	26

## Overview

This lab provides an introduction to Mininet, a virtual testbed used for testing network tools and protocols. It demonstrates how to invoke Mininet from the command-line interface (CLI) utility and build and emulate topologies using a graphical user interface (GUI) application. In this lab, you will use Containernet, a Mininet network emulator fork that uses Docker containers as hosts in emulated network topologies. However, all the concepts covered are bounded to Mininet.

## Objectives

By the end of this lab, you should be able to:

1. Understand what Mininet is and why it is useful for testing network topologies.
2. Invoke Mininet from the CLI.
3. Construct network topologies using the GUI.
4. Save/load Mininet topologies using the GUI.
5. Configure the interfaces of a router using the CLI.

## Lab settings

The information in Table 1 provides the credentials of the machine containing Mininet.

Table 1. Credentials to access Client machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction to Mininet.
2. Section 2: Invoke Mininet using the CLI.
3. Section 3: Build and emulate a network in Mininet using the GUI.
4. Section 4: Configure router r1.

### 1      Introduction to Mininet

Mininet is a virtual testbed enabling the development and testing of network tools and protocols. With a single command, Mininet can create a realistic virtual network on any type of machine (Virtual Machine (VM), cloud-hosted, or native). Therefore, it provides

an inexpensive solution and streamlined development running in line with production networks<sup>1</sup>. Mininet offers the following features:

- Fast prototyping for new networking protocols.
- Simplified testing for complex topologies without the need of buying expensive hardware.
- Realistic execution as it runs real code on the Unix and Linux kernels.
- Open-source environment backed by a large community contributing extensive documentation.

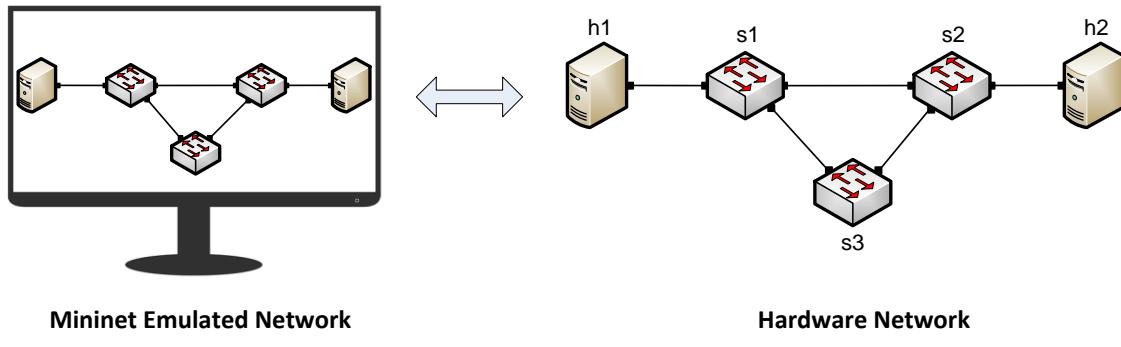


Figure 1. Hardware network vs. Mininet emulated network.

Mininet is useful for development, teaching, and research as it is easy to customize and interact with it through the CLI or the GUI. Mininet was originally designed to experiment with *OpenFlow*<sup>2</sup> and *Open Virtual Network (Open vSwitch)*<sup>3</sup>. This lab, however, only focuses on emulating a simple network environment without Open vSwitch devices.

Mininet's logical nodes can be connected into networks. These nodes are sometimes called containers, or more accurately, *network namespaces*. Containers consume sufficiently fewer resources that networks of over a thousand nodes have created, running on a single laptop. A Mininet container is a process (or group of processes) that no longer has access to all the host system's native network interfaces. Containers are then assigned virtual Ethernet interfaces, which are connected to other containers through a virtual switch<sup>4</sup>. Mininet connects a host and a switch using a virtual Ethernet (veth) link. The veth link is analogous to a wire connecting two virtual interfaces, as illustrated below.

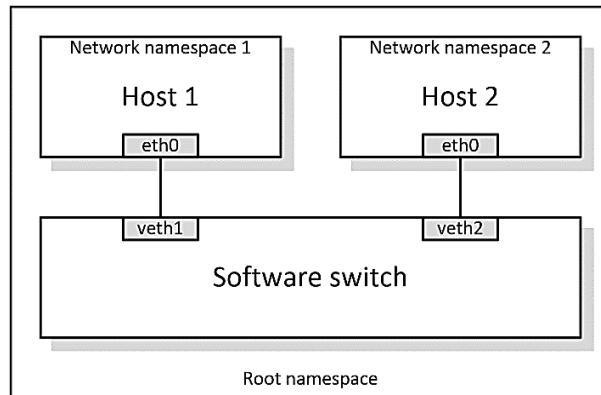


Figure 2. Network namespaces and virtual Ethernet links.

Each container is an independent network namespace, a lightweight virtualization feature that provides individual processes with separate network interfaces, routing tables, and Address Resolution Protocol (ARP) tables.

Mininet provides network emulation opposed to simulation, allowing all network software at any layer to be simply run *as is*, i.e. nodes run the native network software of the physical machine. On the other hand, in a simulated environment applications and protocol implementations need to be ported to run within the simulator before they can be used.

## 2 Invoke Mininet using the CLI

The first step to start Mininet using the CLI is to start a Linux terminal.

### 2.1 Invoke Mininet using the default topology

**Step 1.** Launch a Linux terminal by holding the `Ctrl+Alt+T` keys or by clicking on the Linux terminal icon.

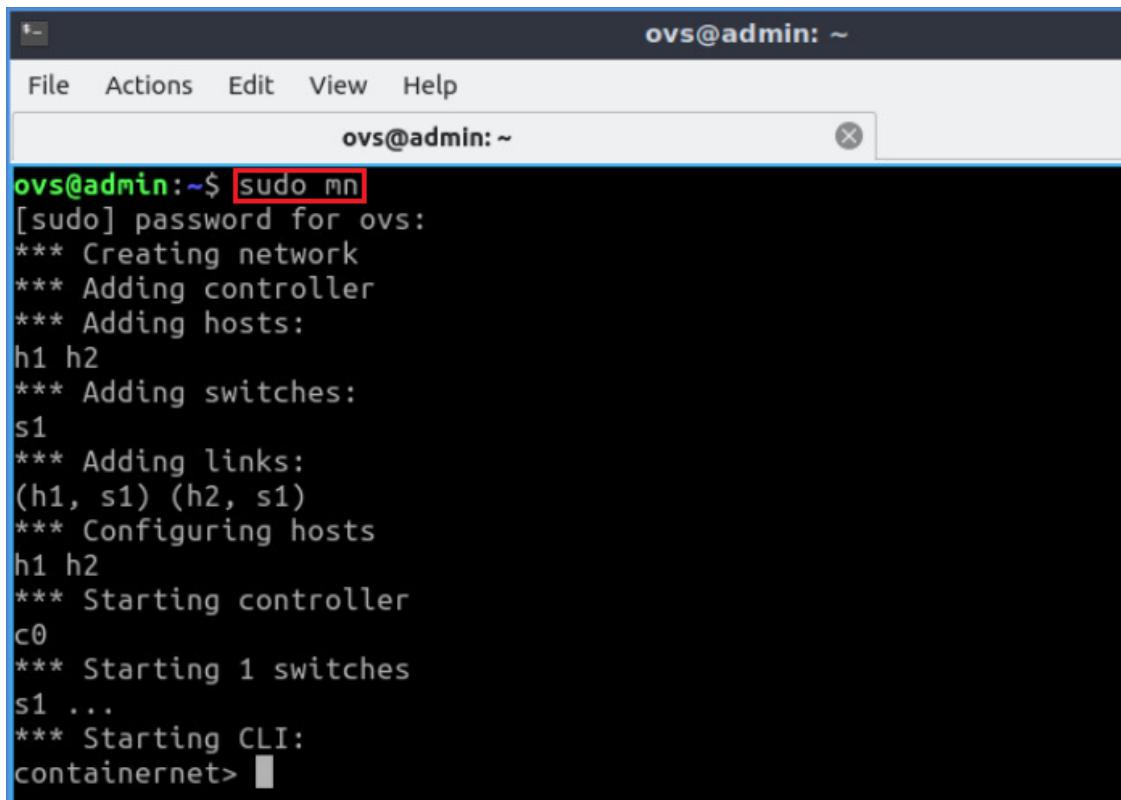


Figure 3. Shortcut to open a Linux terminal.

The Linux terminal is a program that opens a window and permits you to interact with a command-line interface (CLI). A CLI is a program that takes commands from the keyboard and sends them to the operating system for execution.

**Step 2.** To start a minimal topology, enter the command shown below. When prompted for a password, type `password` and hit enter. Note that the password will not be visible as you type it.

```
sudo mn
```



```
ovs@admin:~$ sudo mn
[sudo] password for ovs:
*** Creating network
*** Adding controller
*** Adding hosts
h1 h2
*** Adding switches
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
containernet> █
```

Figure 4. Starting Mininet using the CLI.

The above command starts Mininet with a minimal topology, which consists of a switch connected to two hosts as shown below.

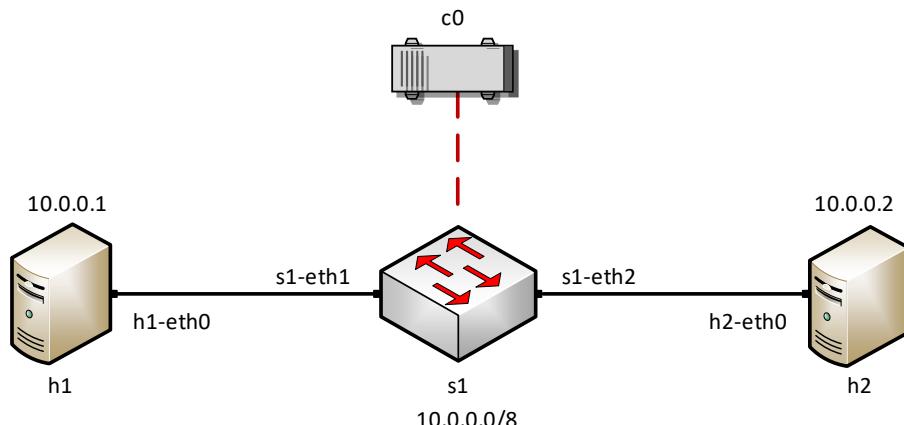


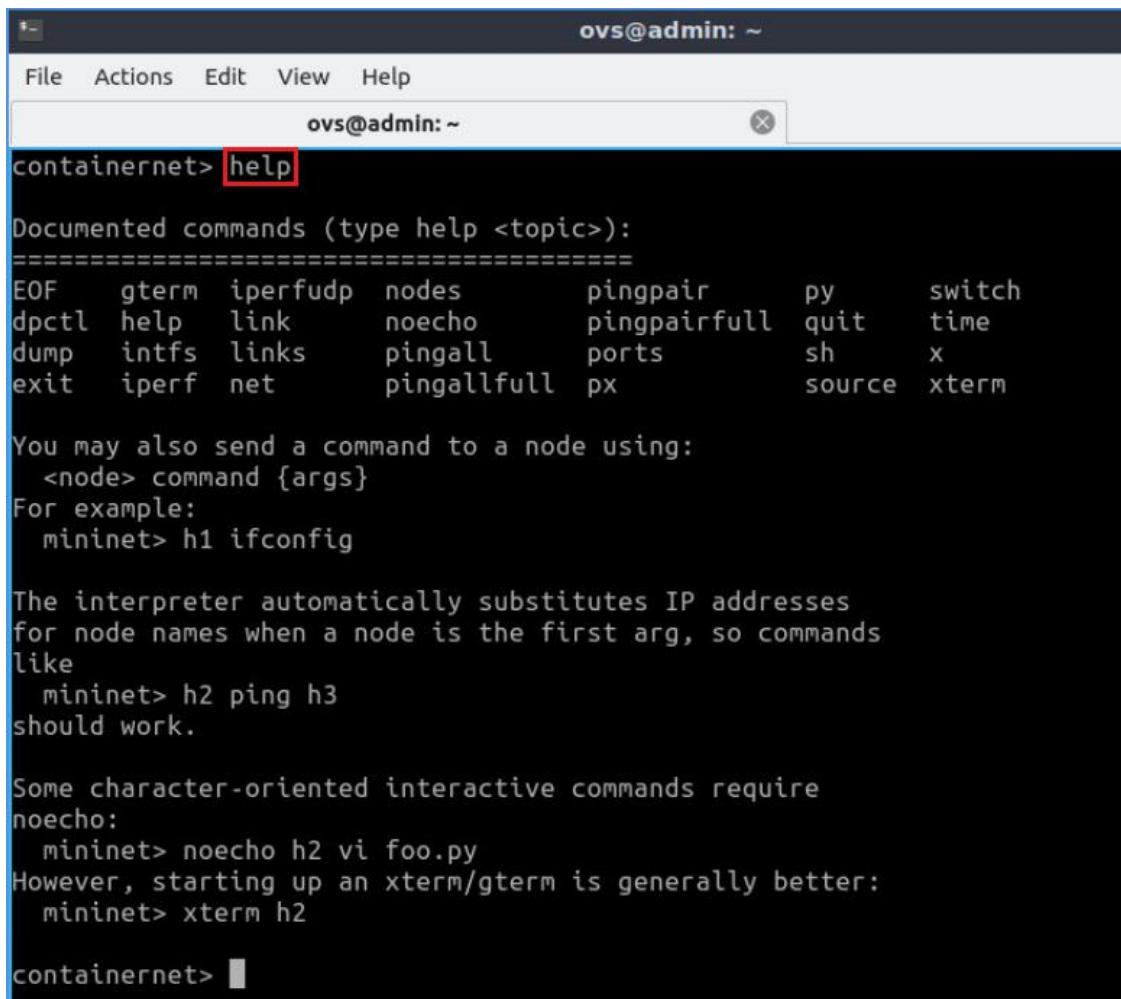
Figure 5. Mininet's default minimal topology.

When issuing the `sudo mn` command, Mininet initializes the topology and launches its command line interface which looks like this:

```
containernet>
```

**Step 3.** To display the list of Mininet CLI commands and examples on their usage, type the following command:

```
help
```



```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
containernet> help
Documented commands (type help <topic>):
=====
EOF      gterm    iperfudp   nodes      pingpair    py      switch
dpctl    help     link       noecho    pingpairfull quit    time
dump     intfs   links      pingall    ports      sh      x
exit     iperf   net       pingallfull px      source  xterm

You may also send a command to a node using:
<node> command {args}
For example:
mininet> h1 ifconfig

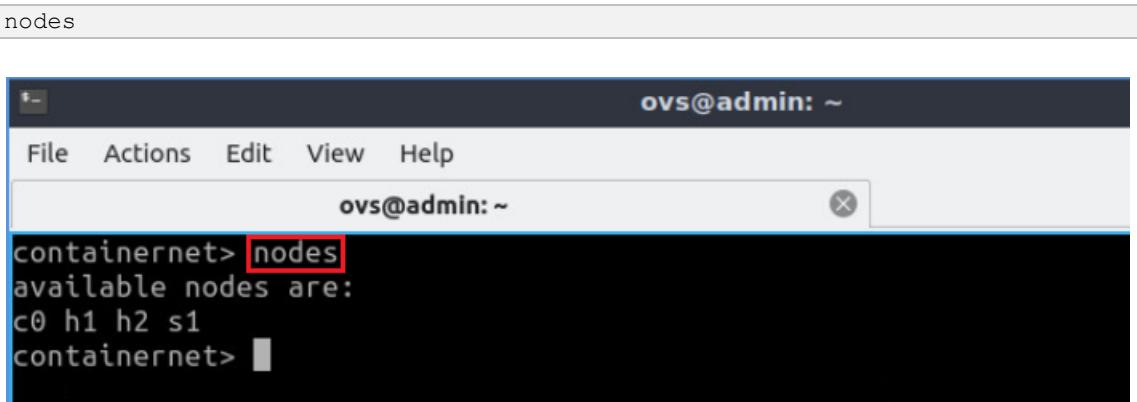
The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
mininet> xterm h2

containernet> 
```

Figure 6. Mininet's `help` command.

**Step 4.** To display the available nodes, type the following command:

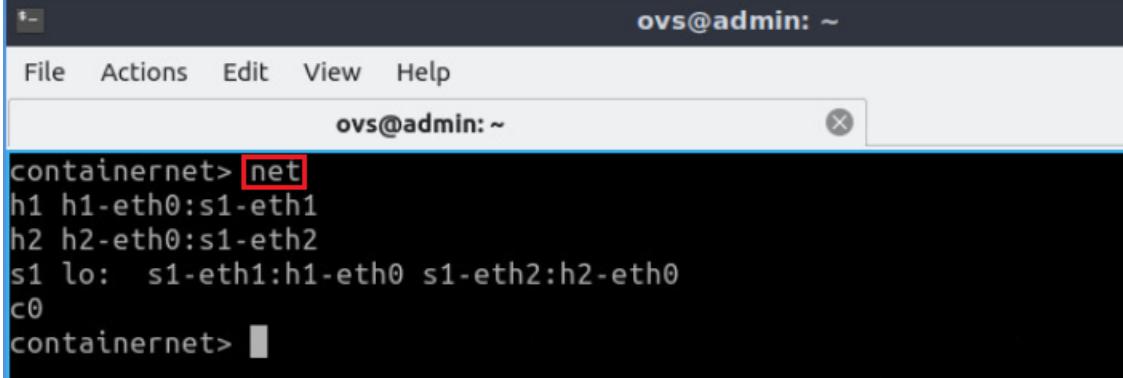


```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
containernet> nodes
available nodes are:
c0 h1 h2 s1
containernet> 
```

Figure 7. Mininet's `nodes` command.

The output of this command shows that there is a controller, two hosts (host h1 and host h2), and a switch (s1).

**Step 5.** It is useful sometimes to display the links between the devices in Mininet to understand the topology. Issue the command shown below to see the available links.



The screenshot shows a terminal window titled "ovs@admin: ~". The command "net" is being typed into the input field, which is highlighted with a red box. The output shows the network topology: host h1 is connected to switch s1 on interface s1-eth1, host h2 is connected to switch s1 on interface s1-eth2, and switch s1 has a loopback interface lo. Controller c0 is also listed.

```
containernet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
containernet>
```

Figure 8. Mininet's `net` command.

The output of this command shows that:

1. Host h1 is connected using its network interface *h1-eth0* to the switch on interface *s1-eth1*.
2. Host h2 is connected using its network interface *h2-eth0* to the switch on interface *s1-eth2*.
3. Switch s1:
  - a. has a loopback interface *lo*.
  - b. connects to *h1-eth0* through interface *s1-eth1*.
  - c. connects to *h2-eth0* through interface *s1-eth2*.
4. Controller c0 is the brain of the network, where it has a global knowledge about the network. A controller instructs the switches on how to forward/drop packets in the network.

Mininet allows you to execute commands on a specific device. To issue a command for a specific node, you must specify the device first, followed by the command.

**Step 6.** To proceed, issue the command:

```
h1 ifconfig
```

```

ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
containernet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0
        ether 56:c1:1f:a7:b4:c5 txqueuelen 1000 (Ethernet)
        RX packets 24 bytes 3196 (3.1 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 3 bytes 270 (270.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        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

containernet> 

```

Figure 9. Output of `h1 ifconfig` command.

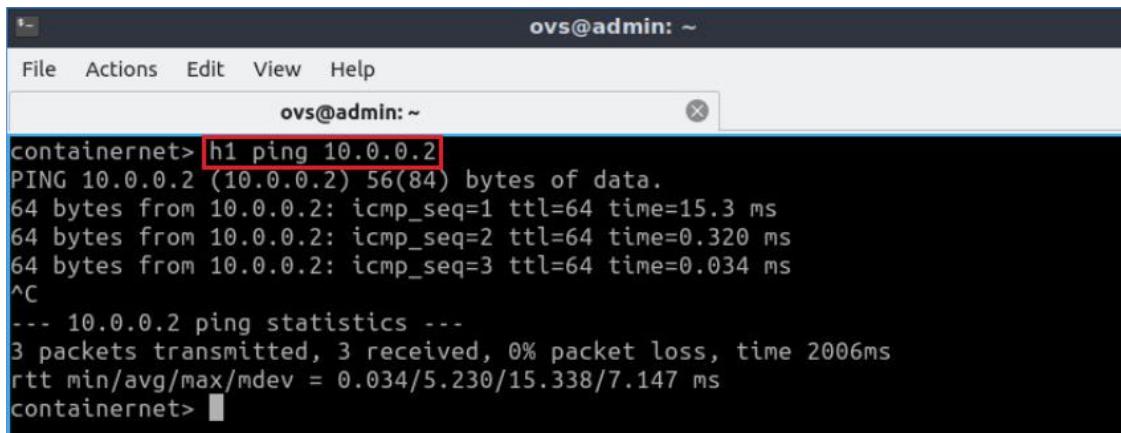
This command executes the `ifconfig` Linux command on host h1. The command shows host h1's interfaces. The display indicates that host h1 has an interface `h1-eth0` configured with IP address 10.0.0.1, and another interface `lo` configured with IP address 127.0.0.1 (loopback interface).

## 2.2 Test connectivity

Mininet's default topology assigns the IP addresses 10.0.0.1/8 and 10.0.0.2/8 to host h1 and host h2, respectively. To test connectivity between them, you can use the command `ping`. The `ping` command operates by sending Internet Control Message Protocol (ICMP) Echo Request messages to the remote computer and waiting for a response. Information available includes how many responses are returned and how long it takes for them to return.

**Step 1.** On the CLI, type the command shown below. This command tests the connectivity between host h1 and host h2. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test. Host h1 (10.0.0.1) sent four packets to host h2 (10.0.0.2) and successfully received the expected responses.

```
h1 ping 10.0.0.2
```

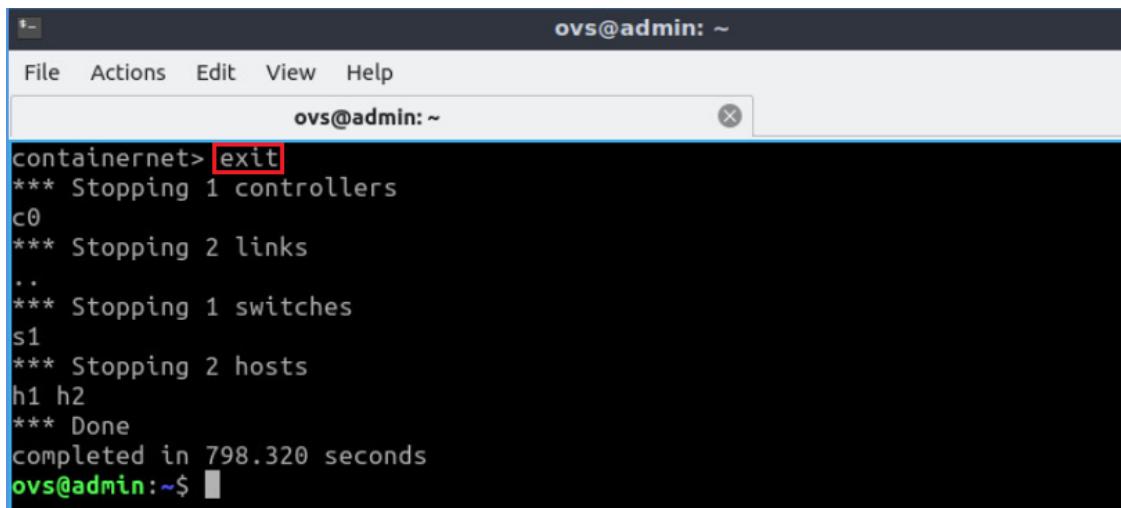


```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
containernet> h1 ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=15.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.320 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.034 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 0.034/5.230/15.338/7.147 ms
containernet> 
```

Figure 10. Connectivity test between host h1 and host h2.

**Step 2.** Stop the emulation by typing the following command:

```
exit
```



```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
containernet> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 798.320 seconds
ovs@admin:~$ 
```

Figure 11. Stopping the emulation using `exit`.

The command `sudo mn -c` is often used on the Linux terminal (not on the Mininet CLI) to clean a previous instance of Mininet (e.g., after a crash).

### 3 Build and emulate a network in Mininet using the GUI

In this section, you will use the application MiniEdit<sup>5</sup> to deploy the topology illustrated below. MiniEdit is a simple GUI network editor for Mininet.

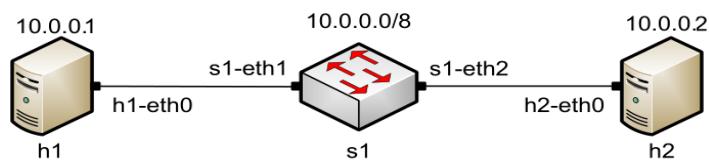


Figure 12. Lab topology.

### 3.1 Build the network topology

**Step 1.** A shortcut to MiniEdit is located on the machine's desktop. Start MiniEdit by clicking on MiniEdit's shortcut. When prompted for a password, type `password`.

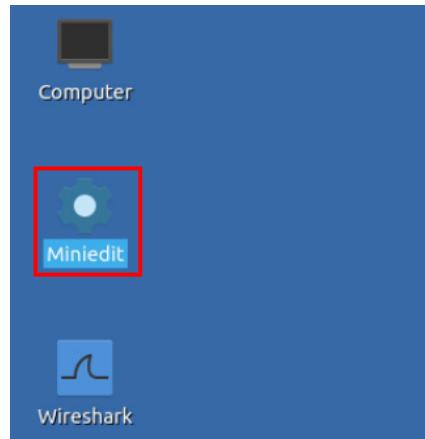


Figure 13. MiniEdit desktop shortcut.

MiniEdit will start, as illustrated below.

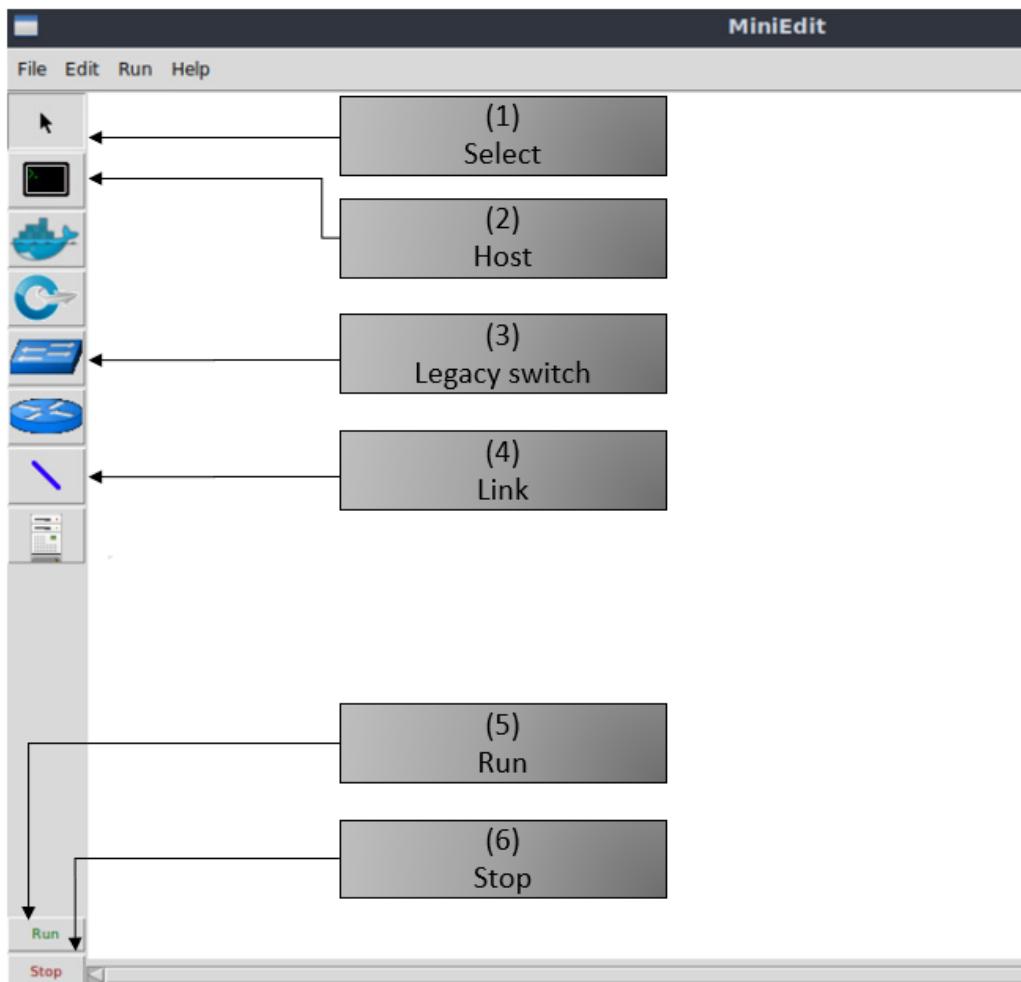


Figure 14. MiniEdit Graphical User Interface (GUI).

The main buttons in this lab are:

1. *Select*: allows selection/movement of the devices. Pressing *Del* on the keyboard after selecting the device removes it from the topology.
2. *Host*: allows addition of a new host to the topology. After clicking this button, click anywhere in the blank canvas to insert a new host.
3. *Legacy switch*: allows addition of a new legacy switch to the topology. After clicking this button, click anywhere in the blank canvas to insert the switch.
4. *Link*: connects devices in the topology (mainly switches and hosts). After clicking this button, click on a device and drag to the second device to which the link is to be established.
5. *Run*: starts the emulation. After designing and configuring the topology, click the run button.
6. *Stop*: stops the emulation.

**Step 2.** To build the topology illustrated in Figure 12, two hosts and one switch must be deployed. Deploy these devices in MiniEdit, as shown below.

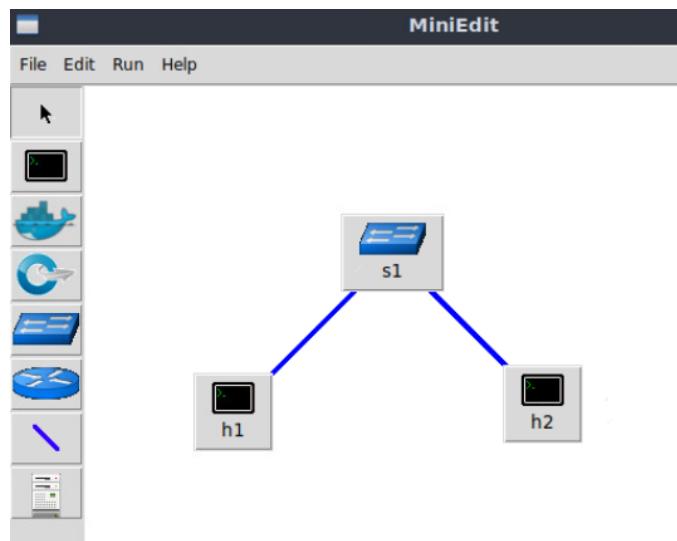


Figure 15. MiniEdit's topology.

Use the buttons described in the previous step to add and connect devices. The configuration of IP addresses is described in Step 3.

**Step 3.** Configure the IP addresses of host h1 and host h2. Host h1's IP address is 10.0.0.1/8 and host h2's IP address is 10.0.0.2/8. A host can be configured by holding the right click and selecting properties on the device. For example, host h2 is assigned the IP address 10.0.0.2/8 in the figure below.

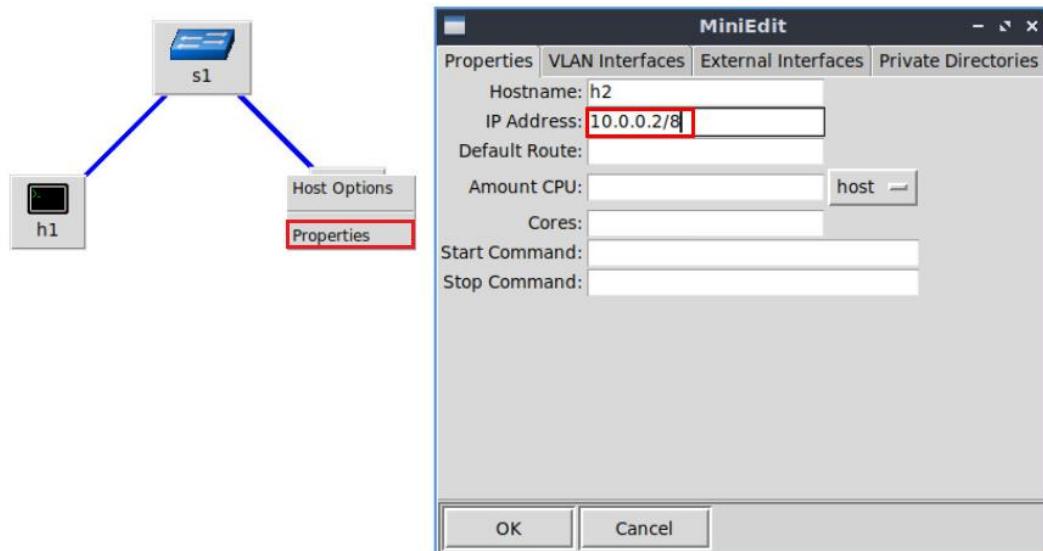


Figure 16. Configuration of a host's properties.

### 3.2 Test connectivity

Before testing the connection between host h1 and host h2, the emulation must be started.

**Step 1.** Click on the *Run* button to start the emulation. The emulation will start and the buttons of the MiniEdit panel will gray out, indicating that they are currently disabled.



Figure 17. Starting the emulation.

**Step 2.** Open a terminal on host h1 by holding the right click on host h1 and selecting *Terminal*. This opens a terminal on host h1 and allows the execution of commands on the host h1. Repeat the procedure on host h2.

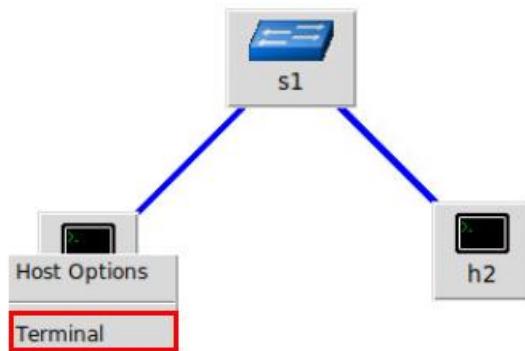


Figure 18. Opening a terminal on host h1.

The network and terminals at host h1 and host h2 will be available for testing.

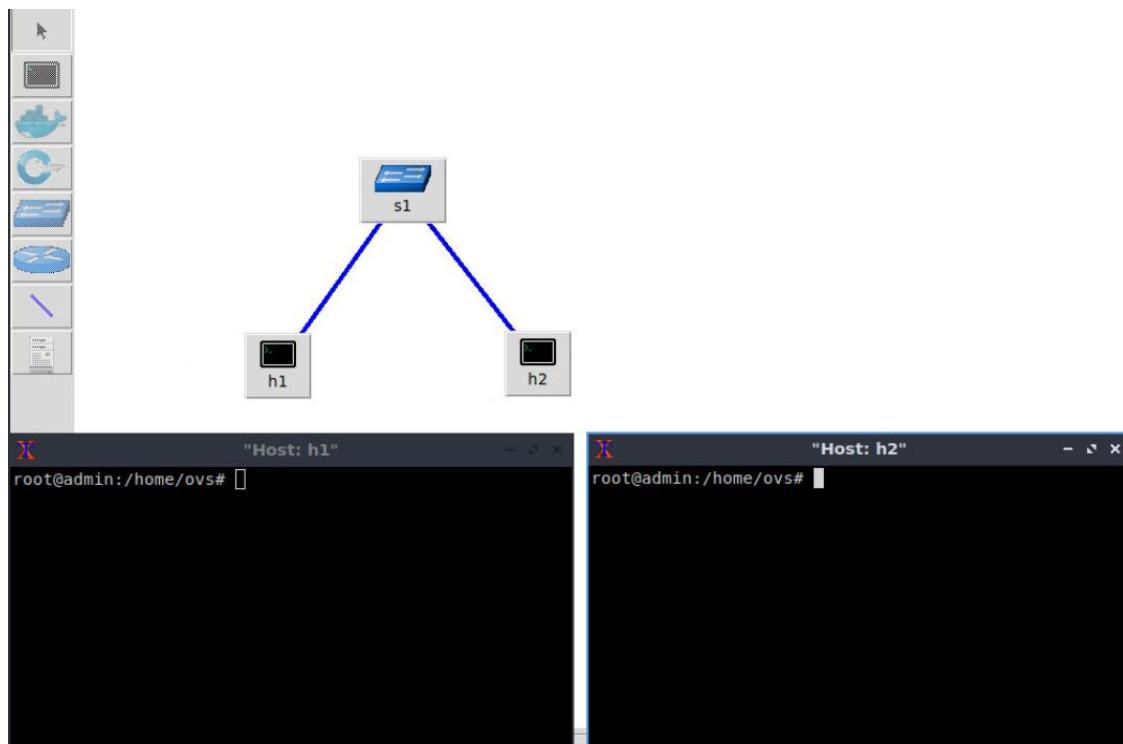
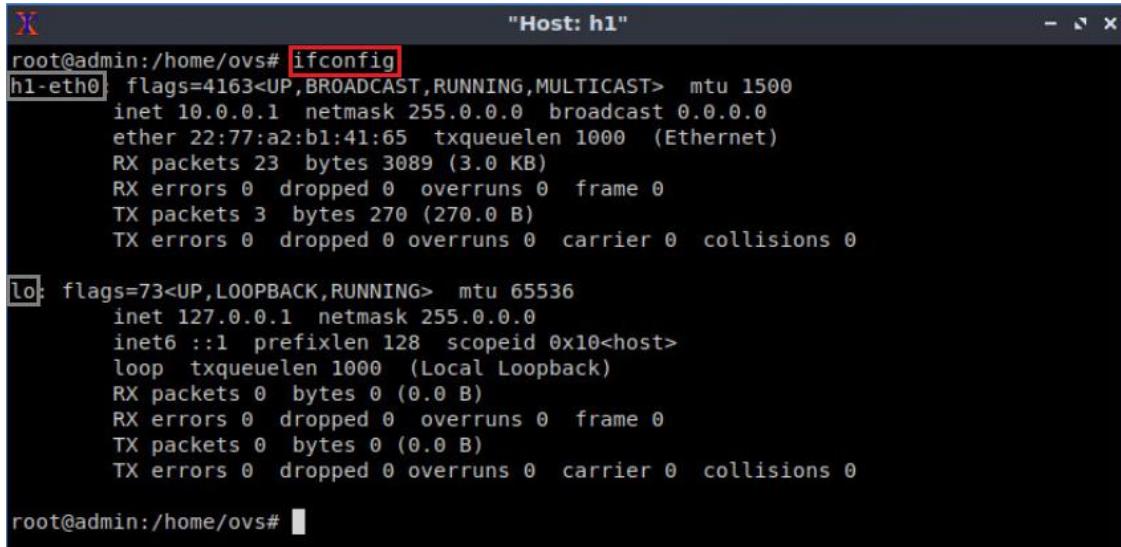


Figure 19. Terminals at host h1 and host h2.

**Step 3.** On host h1's terminal, type the command shown below to display its assigned IP addresses. The interface *h1-eth0* at host h1 should be configured with the IP address 10.0.0.1 and subnet mask 255.0.0.0.

```
ifconfig
```



```
"Host: h1"
root@admin:/home/ovs# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0
          ether 22:77:a2:b1:41:65 txqueuelen 1000 (Ethernet)
            RX packets 23 bytes 3089 (3.0 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 3 bytes 270 (270.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
          inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
            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

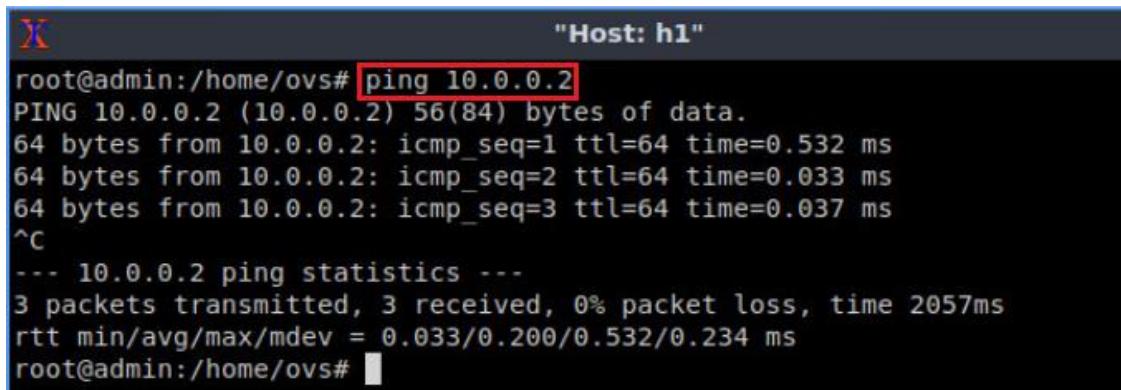
root@admin:/home/ovs#
```

Figure 20. Output of `ifconfig` command on host h1.

Repeat Step 3 on host h2. Its interface `h2-eth0` should be configured with IP address 10.0.0.2 and subnet mask 255.0.0.0.

**Step 4.** On host h1's terminal, type the command shown below. This command tests the connectivity between host h1 and host h2. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test. Host h1 (10.0.0.1) sent six packets to host h2 (10.0.0.2) and successfully received the expected responses.

```
ping 10.0.0.2
```



```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.532 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.033 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.037 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2057ms
rtt min/avg/max/mdev = 0.033/0.200/0.532/0.234 ms
root@admin:/home/ovs#
```

Figure 21. Connectivity test using `ping` command.

**Step 5.** Stop the emulation by clicking on the *Stop* button.



Figure 22. Stopping the emulation.

### 3.3 Automatic assignment of IP addresses

In the previous section, you manually assigned IP addresses to host h1 and host h2. An alternative is to rely on Mininet for an automatic assignment of IP addresses (by default, Mininet uses automatic assignment), which is described in this section.

**Step 1.** Remove the manually assigned IP address from host h1. Hold right-click on host h1, *Properties*. Delete the IP address, leaving it unassigned, and press the *OK* button as shown below. Repeat the procedure on host h2.

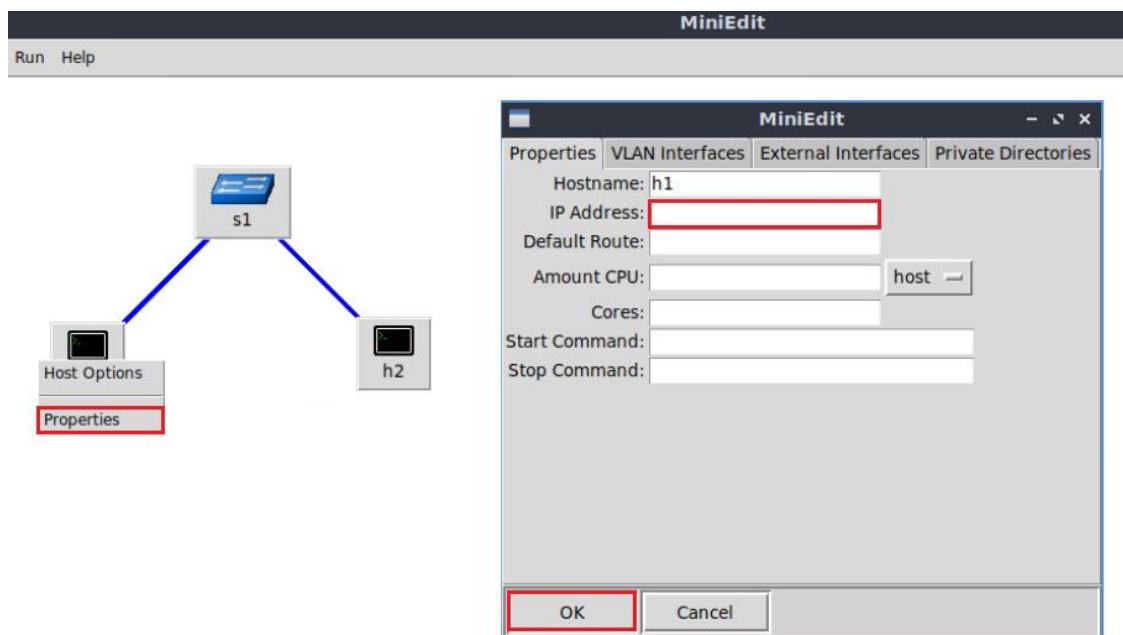


Figure 23. Host h1 properties.

**Step 2.** Click on *Edit, Preferences* button. The default IP base is 10.0.0.0/8. Modify this value to 15.0.0.0/8, and then press the *OK* button.

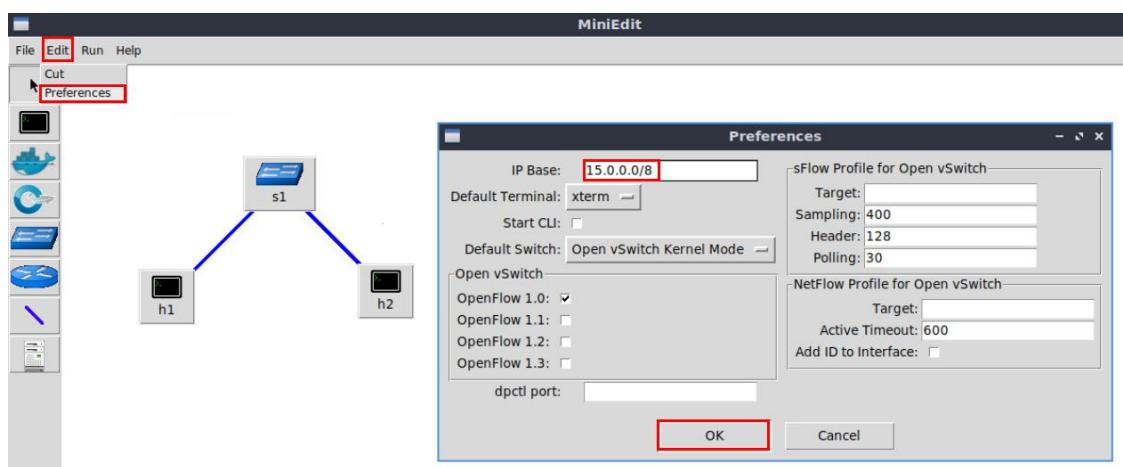


Figure 24. Modification of the IP Base (network address and prefix length).

**Step 3.** Run the emulation again by clicking on the *Run* button. The emulation will start and the buttons of the MiniEdit panel will be disabled.

**Step 4.** Open a terminal on host h1 by holding the right click on host h1 and selecting Terminal.

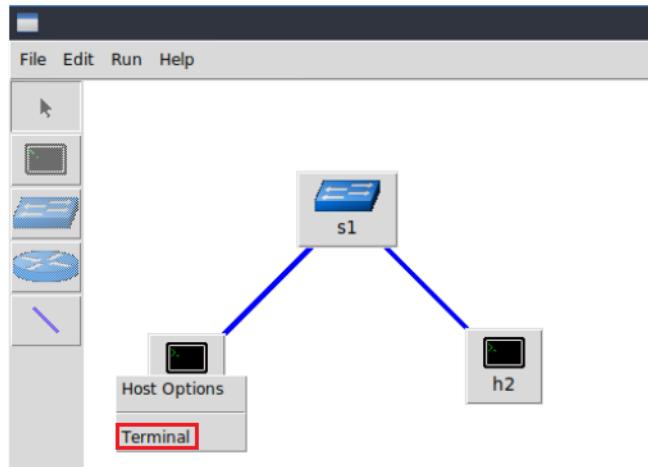


Figure 25. Opening a terminal on host h1.

**Step 5.** Type the command shown below to display the IP addresses assigned to host h1. The interface *h1-eth0* at host h1 now has the IP address 15.0.0.1 and subnet mask 255.0.0.0.

```
ifconfig
```

```
"Host: h1"
root@admin:/home/ovs# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 15.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0
                ether 5e:2f:8f:d4:5b:a3 txqueuelen 1000 (Ethernet)
                RX packets 17 bytes 2447 (2.4 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 3 bytes 270 (270.0 B)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
                loop txqueuelen 1000 (Local Loopback)
                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

root@admin:/home/ovs#
```

Figure 26. Output of `ifconfig` command on host h1.

You can also verify the IP address assigned to host h2 by repeating Steps 4 and 5 on host h2's terminal. The corresponding interface *h2-eth0* at host h2 has now the IP address 15.0.0.2 and subnet mask 255.0.0.0.

**Step 6.** Stop the emulation by clicking on *Stop* button.



Figure 27. Stopping the emulation.

### 3.4 Save and load a Mininet topology

In this section you will save and load a Mininet topology. It is often useful to save the network topology, particularly when its complexity increases. MiniEdit enables you to save the topology to a file.

**Step 1.** Save the current topology by clicking on *File* then *Save*. Provide a name for the topology and save it in the local folder. In this case, we used *myTopology* as the topology name.

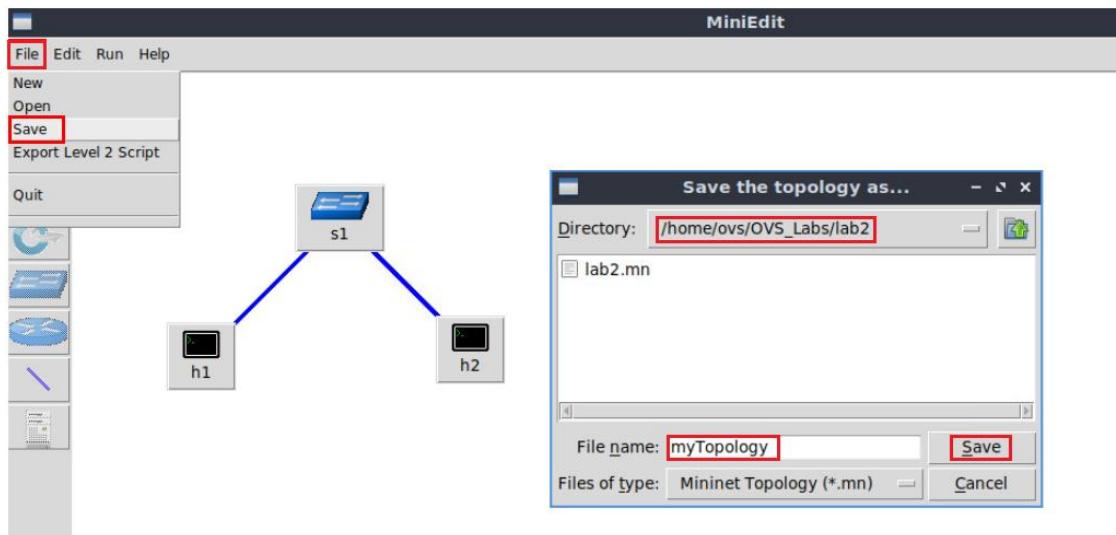


Figure 28. Saving the topology.

**Step 2.** Load the topology by clicking on *File* then *Open*. Search for the topology file called *lab2.mn* and click on *Open*. A new topology will be loaded to MiniEdit.

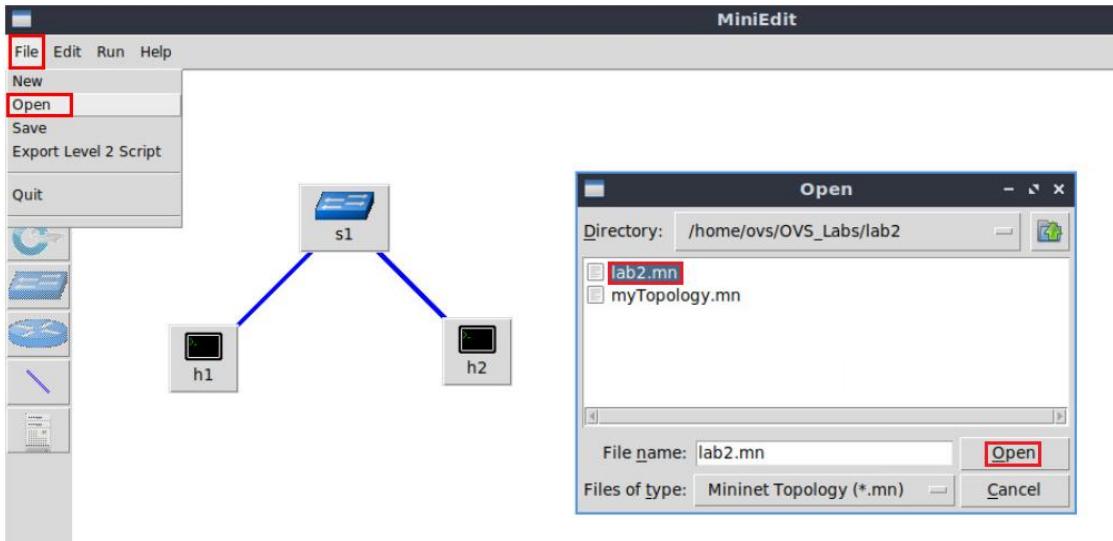


Figure 29. Opening a topology.

#### 4 Configure router r1

In the previous section, you loaded a topology that consists of two networks directly connected to router r1. Consider Figure 30. In this topology two LANs, defined by switch s1 and switch s2 are connected to router r1. Initially, host h1 and host h2 do not have connectivity thus, you will configure router r1's interfaces in order to establish connectivity between the two networks.

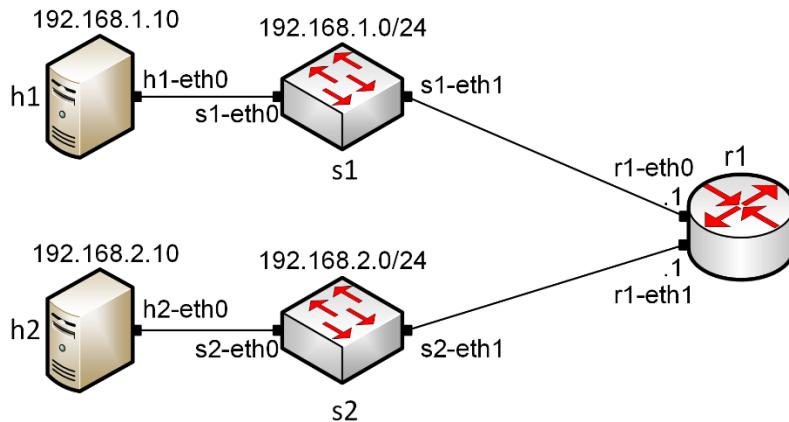


Figure 30. Topology.

Table 2 summarized the IP addresses used to configure router r1 and the end-hosts.

Table 2. Topology information.

Device	Interface	IP Address	Subnet	Default gateway
r1	r1-eth0	192.168.1.1	/24	N/A
	r1-eth1	192.168.2.1	/24	N/A
h1	h1-eth0	192.168.1.10	/24	192.168.1.1

h2	h2-eth0	192.168.2.10	/24	192.168.2.1
----	---------	--------------	-----	-------------

**Step 1.** Click on the *Run* button to start the emulation. The emulation will start and the buttons of the MiniEdit panel will gray out, indicating that they are currently disabled.



Figure 31. Starting the emulation.

#### 4.1 Verify end-hosts configuration

In this section, you will verify that the IP addresses are assigned according to Table 2. Additionally, you will check routing information.

**Step 1.** Hold right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

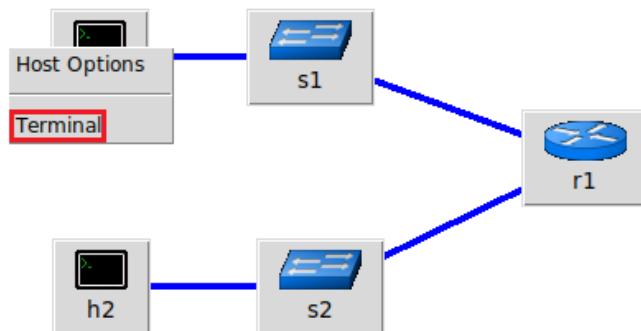


Figure 32. Opening a terminal on host h1.

**Step 2.** In host h1 terminal, type the command shown below to verify that the IP address was assigned successfully. You will verify that host h1 has two interfaces, *h1-eth0* configured with the IP address 192.168.1.10 and the subnet mask 255.255.255.0 and, the loopback interface *lo* configured with the IP address 127.0.0.1.

```
ifconfig
```

```
"Host: h1"
root@admin:/home/ovs# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.10 netmask 255.255.255.0 broadcast 0.0.0.0
        ether c6:60:40:0f:d3:07 txqueuelen 1000 (Ethernet)
        RX packets 16 bytes 2248 (2.2 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 3 bytes 270 (270.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        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

root@admin:/home/ovs# "
```

Figure 33. Output of `ifconfig` command.

**Step 3.** In host h1 terminal, type the command shown below to verify that the default gateway IP address is 192.168.1.1.

```
"Host: h1"
root@admin:/home/ovs# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref  Use Iface
default         192.168.1.1   0.0.0.0         UG    0      0      0 h1-eth0
192.168.1.0     0.0.0.0       255.255.255.0   U     0      0      0 h1-eth0
root@admin:/home/ovs# "
```

Figure 34. Output of `route` command.

**Step 4.** In order to verify host 2 default route, proceed similarly by repeating from step 1 to step 3 in host h2 terminal. Similar results should be observed.

## 4.2 Configure router's interface

**Step 1.** In order to configure router r1, hold right-click on router r1 and select *Terminal*.

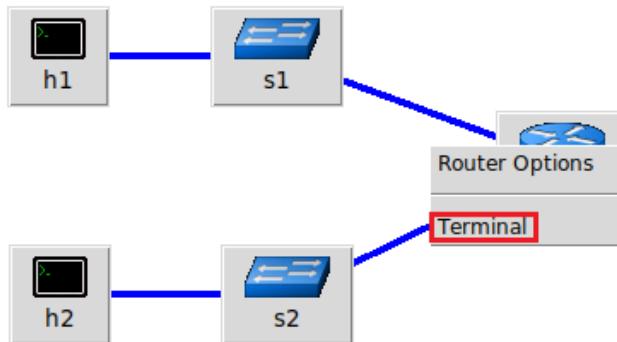


Figure 35. Opening a terminal on router r1.

**Step 2.** In this step, you will start zebra daemon, which is a multi-server routing software that provides TCP/IP based routing protocols. The configuration will not be working if you do not enable zebra daemon initially. In order to start the zebra, type the following command:

```
zebra
```

```
"Host: r1"
root@admin:/etc/routers/r1# zebra
```

Figure 36. Starting zebra daemon.

**Step 3.** After initializing zebra, vtysh should be started in order to provide all the CLI commands defined by the daemons. To proceed, issue the following command:

```
vtysh
```

```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

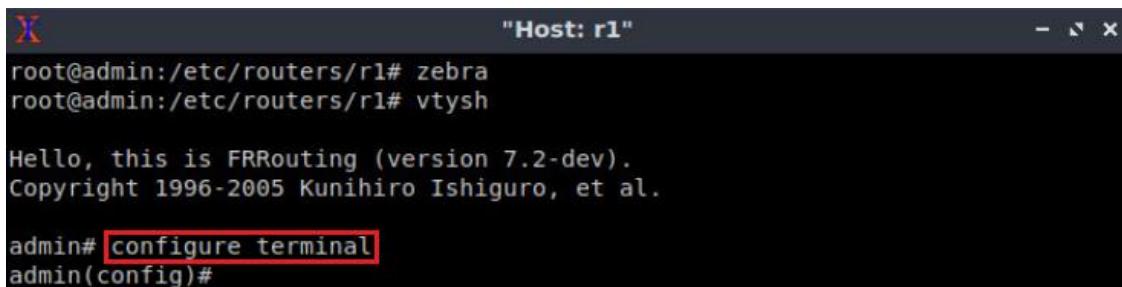
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin#
```

Figure 37. Starting vtysh on router r1.

**Step 4.** Type the following command in the router r1 terminal to enter in configuration mode.

```
configure terminal
```



```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

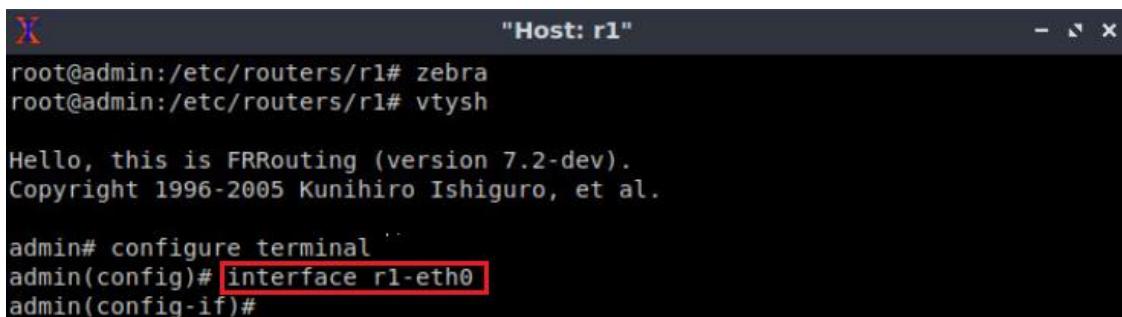
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)#
```

Figure 38. Entering in configuration mode.

**Step 5.** Type the following command in the router r1 terminal to configure interface *r1-eth0*.

```
interface r1-eth0
```



```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

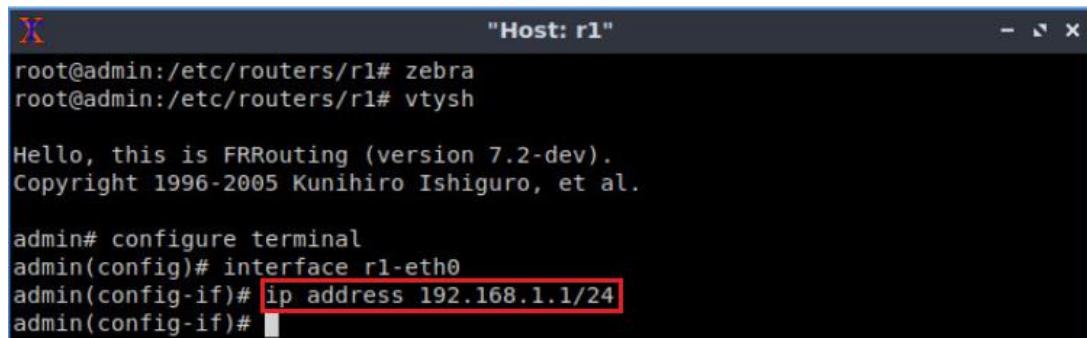
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# interface r1-eth0
admin(config-if)#
```

Figure 39. Configuring interface *r1-eth0*.

**Step 6.** Type the following command on router r1 terminal to configure the IP address of the interface *r1-eth0*.

```
ip address 192.168.1.1/24
```



```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

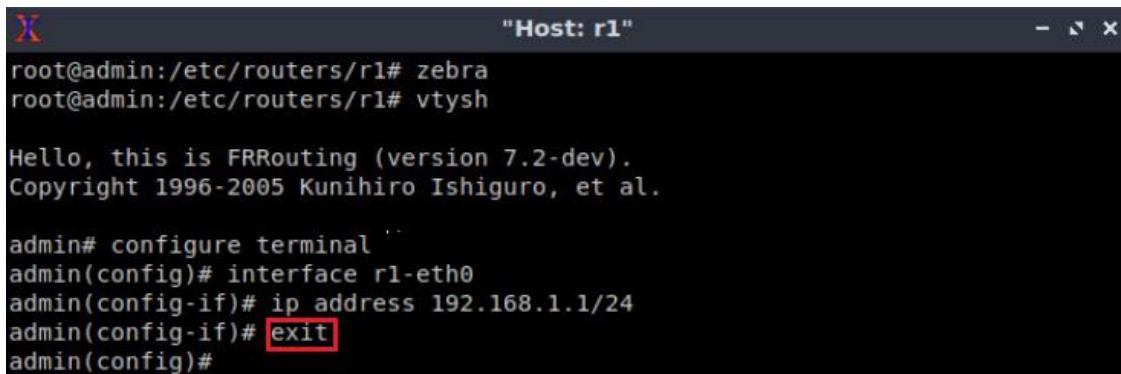
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# interface r1-eth0
admin(config-if)# ip address 192.168.1.1/24
admin(config-if)#
```

Figure 40. Configuring an IP address to interface *r1-eth0*.

**Step 7.** Type the following command exit from interface *r1-eth0* configuration.

```
exit
```



```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

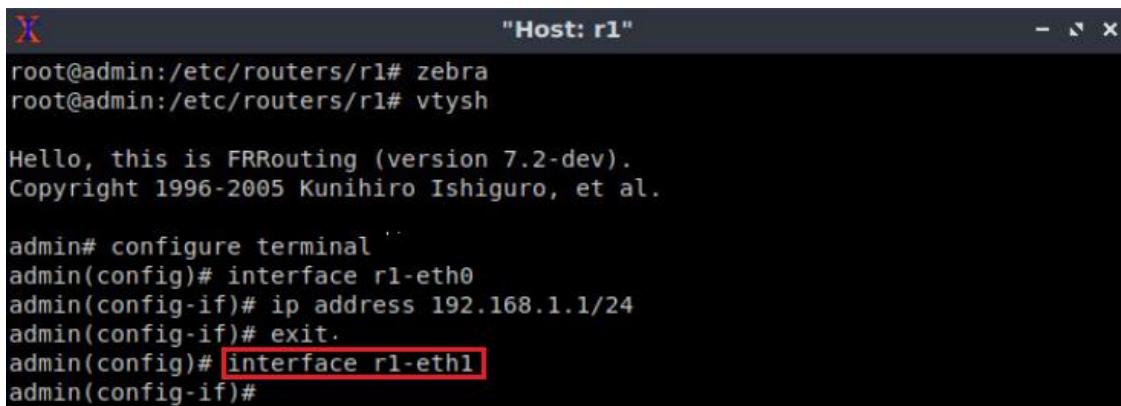
admin# configure terminal
admin(config)# interface r1-eth0
admin(config-if)# ip address 192.168.1.1/24
admin(config-if)# exit
admin(config)#

```

Figure 41. Exiting from configuring interface *r1-eth0*.

**Step 8.** Type the following command on router r1 terminal to configure the interface *r1-eth1*.

```
interface r1-eth1
```



```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

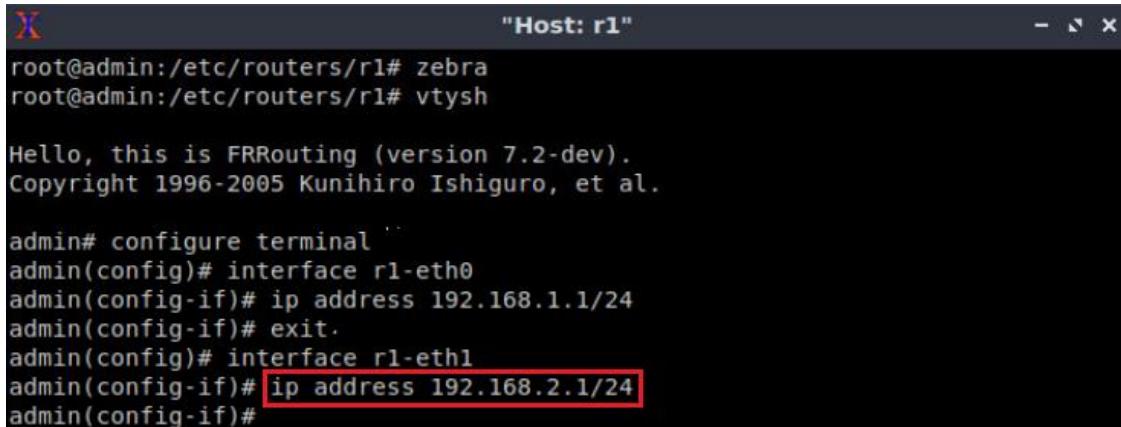
admin# configure terminal
admin(config)# interface r1-eth0
admin(config-if)# ip address 192.168.1.1/24
admin(config-if)# exit
admin(config)# interface r1-eth1
admin(config-if)#

```

Figure 42. Configuring interface *r1-eth1*.

**Step 9.** Type the following command on router r1 terminal to configure the IP address of the interface *r1-eth1*.

```
ip address 192.168.2.1/24
```



```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

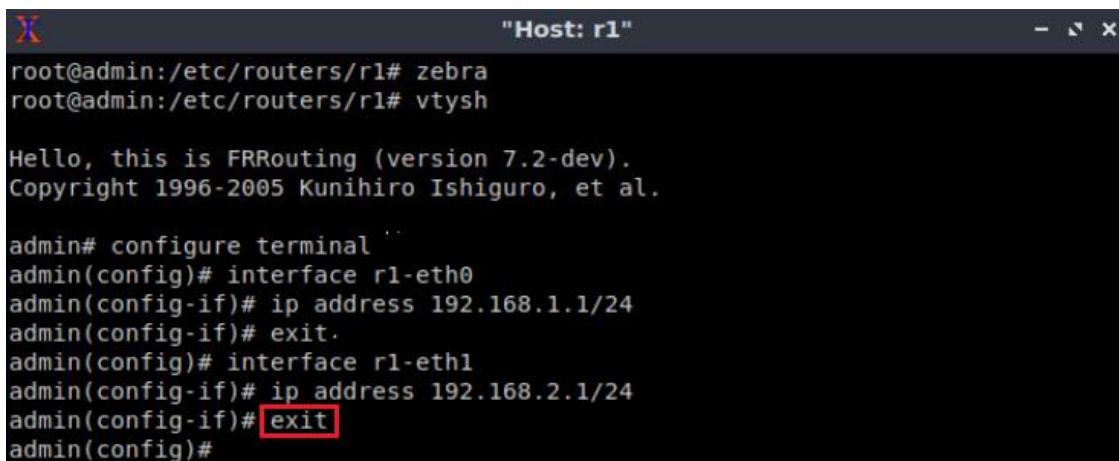
admin# configure terminal
admin(config)# interface r1-eth0
admin(config-if)# ip address 192.168.1.1/24
admin(config-if)# exit
admin(config)# interface r1-eth1
admin(config-if)# ip address 192.168.2.1/24
admin(config-if)#

```

Figure 43. Configuring an IP address to interface *r1-eth1*.

**Step 10.** Type the following command to exit from *r1-eth1* interface configuration.

```
exit
```



The terminal window shows the FRRouting configuration process. It starts with 'zebra' and 'vtysh' commands. Then it enters configuration mode ('configure terminal') and configures two interfaces: 'r1-eth0' and 'r1-eth1'. Both interfaces have their IP addresses set to 192.168.1.1/24 and 192.168.2.1/24 respectively. Finally, it exits configuration mode twice ('exit') and returns to the 'admin#' prompt.

```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

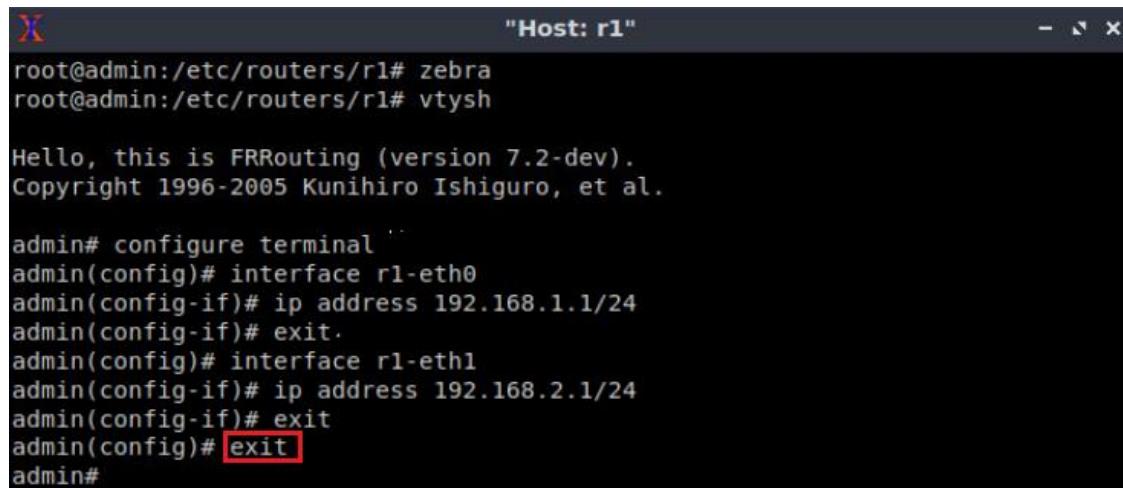
admin# configure terminal
admin(config)# interface r1-eth0
admin(config-if)# ip address 192.168.1.1/24
admin(config-if)# exit
admin(config)# interface r1-eth1
admin(config-if)# ip address 192.168.2.1/24
admin(config-if)# exit
admin(config)#
```

Figure 44. Exiting from configuring interface *r1-eth1*.

#### 4.3 Verify router r1 configuration

**Step 1.** Exit from router r1 configuration mode issuing the following command:

```
exit
```



This terminal window is identical to Figure 44, showing the configuration of 'r1-eth0' and 'r1-eth1'. The 'exit' command is highlighted with a red box in the configuration mode section.

```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

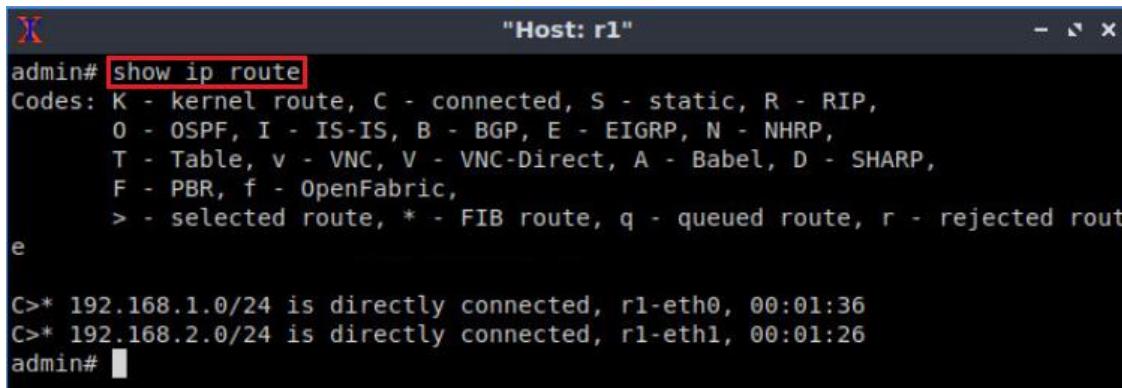
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# interface r1-eth0
admin(config-if)# ip address 192.168.1.1/24
admin(config-if)# exit
admin(config)# interface r1-eth1
admin(config-if)# ip address 192.168.2.1/24
admin(config-if)# exit
admin(config)# exit
admin#
```

Figure 45. Exiting from configuration mode.

**Step 2.** Type the following command on router r1 terminal to verify the routing information of router r1. It will be showing all the directly connected networks.

```
show ip route
```



```
"Host: r1"
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       0 - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected route
C>* 192.168.1.0/24 is directly connected, r1-eth0, 00:01:36
C>* 192.168.2.0/24 is directly connected, r1-eth1, 00:01:26
admin#
```

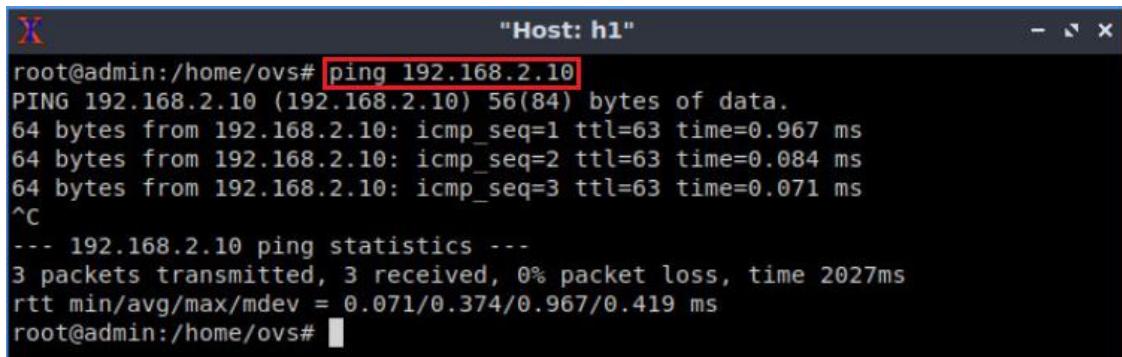
Figure 46. Displaying routing information of router r1.

#### 4.4 Test connectivity between end-hosts

In this section you will run a connectivity test between host h1 and host h2.

**Step 1.** In host h1 terminal type the command shown below. Notice that according to Table 2, the IP address 192.168.2.10 is assigned to host h2. To stop the test press **ctrl+c**

```
ping 192.168.2.10
```



```
"Host: h1"
root@admin:/home/ovs# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
64 bytes from 192.168.2.10: icmp_seq=1 ttl=63 time=0.967 ms
64 bytes from 192.168.2.10: icmp_seq=2 ttl=63 time=0.084 ms
64 bytes from 192.168.2.10: icmp_seq=3 ttl=63 time=0.071 ms
^C
--- 192.168.2.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.071/0.374/0.967/0.419 ms
root@admin:/home/ovs#
```

Figure 47. Connectivity test between host h1 and host h2.

This concludes Lab 2. Stop the emulation and then exit out of MiniEdit and Linux terminal.

## References

1. Mininet walkthrough. [Online]. Available: <http://Mininet.org>.
2. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow,” ACM SIGCOMM computer Communication review, vol. 38, no. 2, p. 69, 2008.
3. Linux foundation, [Online]. Available: <http://openvSwitch.org>.
4. P. Dordal, “An introduction to computer networks.”. [Online]. Available: <https://intronetworks.cs.luc.edu/>.
5. B. Lantz, G. Gee, “MiniEdit: a simple network editor for Mininet,” 2013. [Online]. Available: <https://github.com/Mininet/Mininet/blob/master/examples>.



## OPEN VIRTUAL SWITCH

### Lab 3: Introduction to Open vSwitch

Document Version: **06-29-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    Introduction to Open vSwitch .....	4
2    Lab topology.....	5
2.1    Lab settings.....	6
2.2    Loading a topology .....	6
3    Querying switches and daemon information .....	8
4    Open vSwitch fail-modes .....	10
5    Inspecting the state of the switch .....	15
References .....	17

## Overview

This lab introduces Open Virtual Switch (Open vSwitch, or OVS), an open-source software switch used in virtualization environments. This lab aims to show the basic features of Open vSwitch through the utility tools (`ovs-vsctl`, `ovs-ofctl`, `ovs-appctl`). Such tools allow querying and configuring the switch's daemon, as well as administering the switch state which includes flow table entries.

## Objectives

By the end of this lab, you should be able to:

1. Understand the architecture of an Open vSwitch.
2. Explore Open vSwitch utility tools.
3. Understand the different modes of an Open vSwitch.
4. Display and modify basic parameters of the switch daemon.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Querying switches and daemon information.
4. Section 4: Open vSwitch fail-modes.
5. Section 5: Inspecting the state of the switch.

### 1      **Introduction**

Virtualization has changed the way of computing over the past few years. It can combine various physical networks into one virtual network. Virtualization provides simplicity, agility, and security by automating many of the data center network processes. By

consolidating multiple servers and storage devices into a single host machine, virtualization benefits the organization by reducing physical hardware.

Software switches form an integral part of any virtualized computing setup. They provide network access for virtual machines (VMs) by linking virtual and physical network interfaces. The deployment of software switches in virtualized environments has led to the term virtual switches and paved the way for the mainstream adoption of software switches<sup>2</sup>, which did not receive much attention before. Virtual switches have been developed to meet the requirements in a virtualized environment. In addition to the traditional benefits of software switches such as high flexibility, vendor independence, low costs, and conceptual benefits for switching without Ethernet limitations, these devices focus on the performance and provide advanced features<sup>1</sup>.

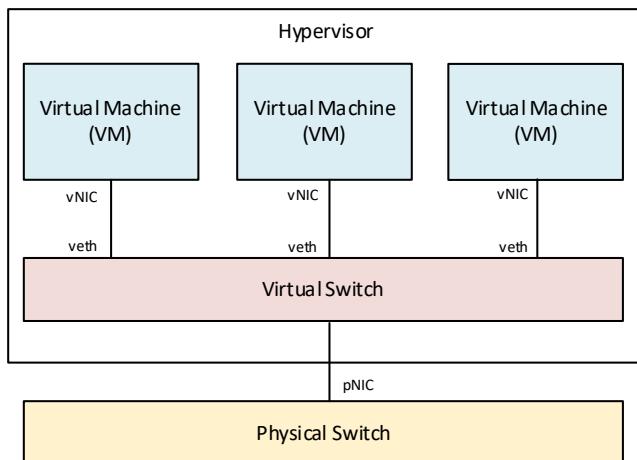


Figure 1. Virtual switch architecture.

Consider Figure 1. The figure shows virtual switch architecture where VMs are attached to a virtual switch through virtual Network Interface Cards (vNICs). Virtual switch uses virtual Ethernet link (veth) to connect VMs. The hypervisor permits communication to the physical networking infrastructure by attaching the server's physical NICs (pNICs) to the hypervisor's logical infrastructure, permitting efficient communication among VMs within the hypervisor as well as efficient communication to the external network.

## 1.1 Introduction to Open vSwitch

Open vSwitch is an open-source software switch in virtual environments. Open vSwitch is able to forward traffic between different VMs on either the same physical host or on different hosts. It includes most of the features provided by regular switches, and provides advanced features such as NetFlow, IPFIX, and sFlow<sup>1</sup>. It can also operate entirely in userspace without assistance from a kernel module, at the cost of degradation in the performance. The most vital components of Open vSwitch are the switch daemon (ovs-vswitchd) and a kernel module<sup>5</sup>.

Figure 2 depicts the architecture of Open vSwitch, all the components are described below:

**Kernel Module:** The kernel module is designed for handling switching and tunneling. When a packet is received, it looks for a match in the kernel table. When a match is found, required actions are executed. Otherwise, packets are sent to the userspace.

**Open vSwitch Database:** The Open vSwitch Database (OVSDB) contains switch configuration and keeps track of created and modified interfaces. All the configuration is stored on persistent storage and survives a reboot. The OVSDB-server communicates with ovs-vswitchd and the controller using the OVSDB management protocol.

**Open vSwitch Daemon:** The Open vSwitch daemon (ovs-vswitchd) is one of the major components of Open vSwitch. It communicates with the controller using OpenFlow and with the OVSDB-server through the OVSDB management protocol. The ovs-vswitchd communicates with the kernel module over netlink (a Linux kernel interface used for creating a connection between userspace processes and the kernel). It supports multiple independent data paths, known as bridges.

**Controller:** The controller is remotely connected to switches and OVSDB-server. It controls the switch and manages the routing.

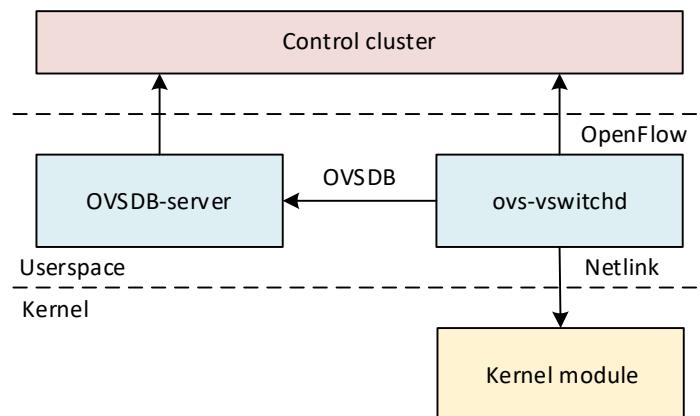


Figure 2. Open vSwitch architecture.

## 2 Lab topology

Consider Figure 3. The topology consists of two hosts and one switch. The hosts belong to the network 10.0.0.0/8.

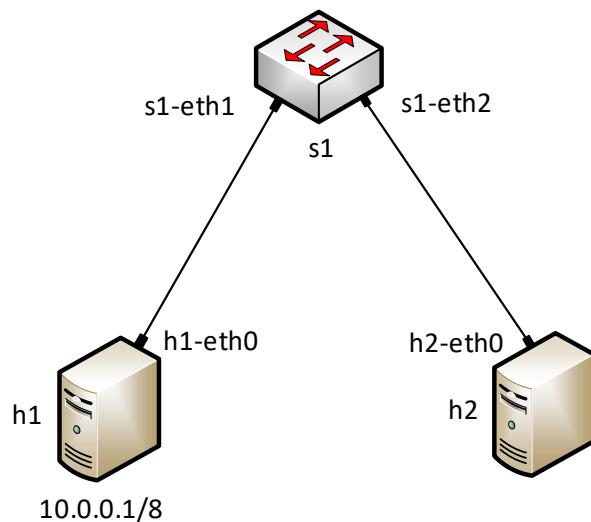


Figure 3. Lab topology.

## 2.1 Lab settings

The hosts are configured according to Table 2.

Table 2. Topology information.

Host	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8

## 2.2 Loading a topology

**Step 1.** Start by launching MiniEdit by clicking on the desktop's shortcut. When prompted for a password, type `password`.

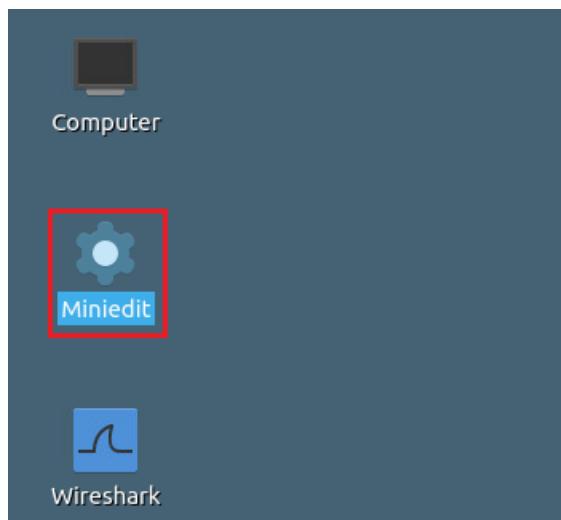


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then *open* to load the lab's topology. Locate the *lab3.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab3* and click on *Open*.

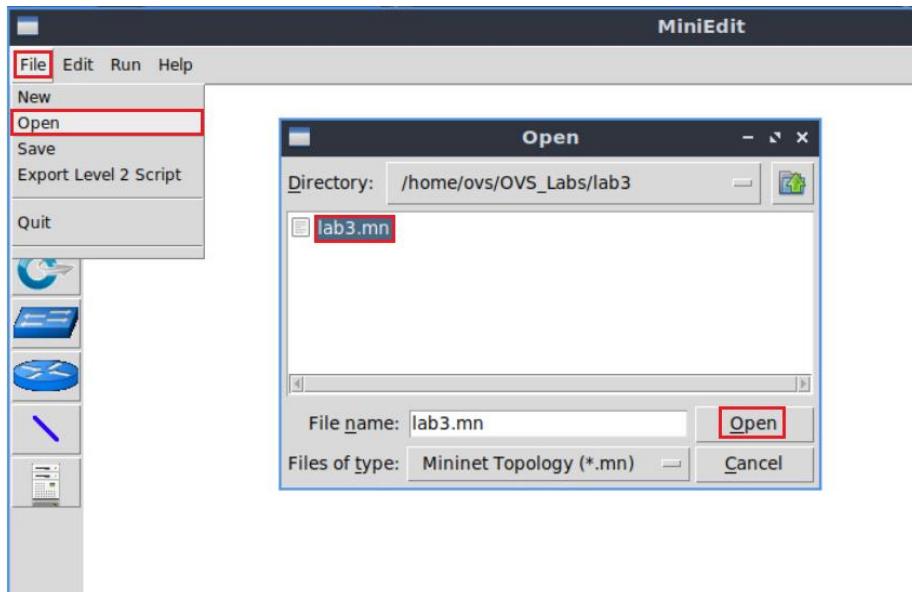


Figure 5. MiniEdit's Open dialog.

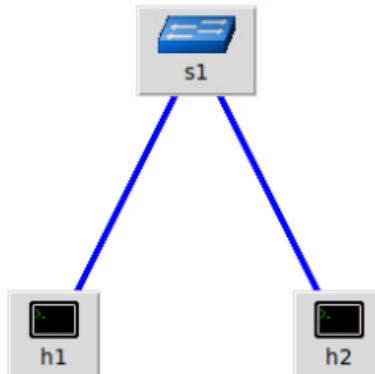


Figure 6. MiniEdit's topology.

**Step 3.** To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 7. Starting the emulation.

**Step 4.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

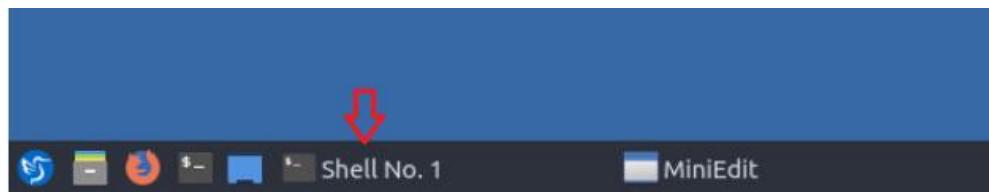


Figure 8. Opening Mininet's terminal.

**Step 5.** Issue the following command to display the links between the interfaces in the topology.

```
links
```

A screenshot of the 'Shell No. 1' terminal window. The user has entered the command 'links'. The output shows two network links: 'h1-eth0<->s1-eth1' and 's1-eth2<->h2-eth0'. Both links are labeled '(OK OK)'.

Figure 9. Displaying network interfaces.

In Figure 9, the link displayed within the gray box indicates that interface *eth0* of host h1 connects to interface *eth1* of switch s1 (i.e., *h1-eth0<->s1-eth1*).

### 3 Querying switches and daemon information

In this section, you will use `ovs-vsctl` command line tool to query the Open vSwitch daemon for information pertaining to the running switches. You will also query general information such as the installed Open vSwitch version.

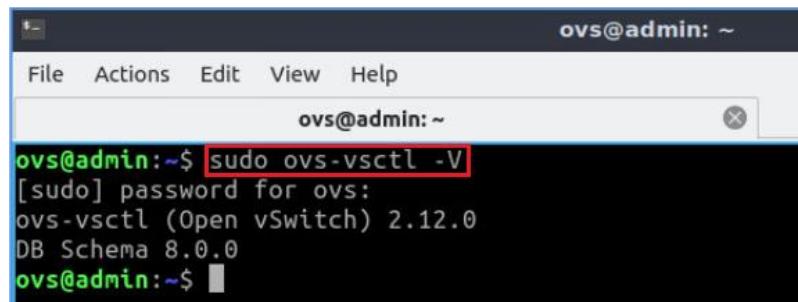
**Step 1.** Open the Linux terminal.



Figure 10. Opening Linux terminal.

**Step 2.** In order to show the version of the installed Open vSwitch on the system, type the following command. When prompted for a password, type `password`. You can verify that the installed version of Open vSwitch is 2.12.0.

```
sudo ovs-vsctl -V
```

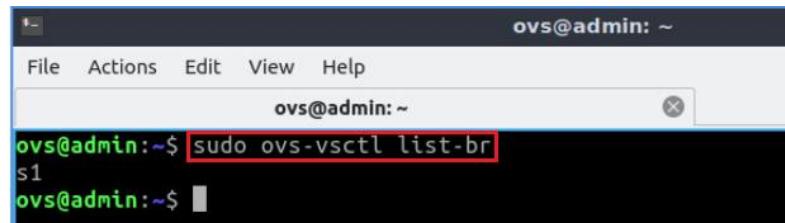


```
ovs@admin:~$ sudo ovs-vsctl -V
[sudo] password for ovs:
ovs-vsctl (Open vSwitch) 2.12.0
DB Schema 8.0.0
ovs@admin:~$
```

Figure 11. Verifying Open vSwitch version.

**Step 3.** To list all the switches that are currently running in your system, type the following command. Note that a switch is also referred to as a bridge.

```
sudo ovs-vsctl list-br
```



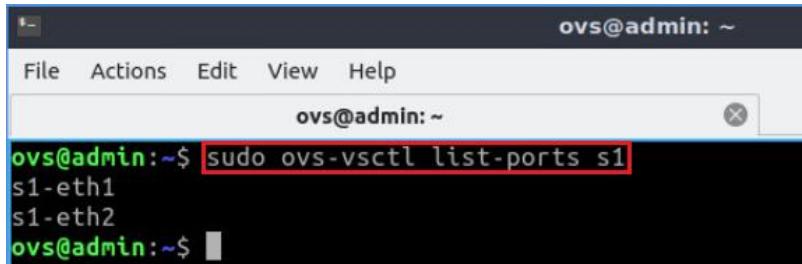
```
ovs@admin:~$ sudo ovs-vsctl list-br
s1
ovs@admin:~$
```

Figure 12. Verifying Open vSwitch list.

The figure above shows that there is a single switch *s1* that is running in the system. The output verifies the lab's topology which consists of a single switch.

**Step 4.** Type the following command to list all the ports within switch *s1*. You will see that two ports are connected to the switch. These ports connect hosts *h1* and *h2* to the switch.

```
sudo ovs-vsctl list-ports s1
```



```
ovs@admin:~$ sudo ovs-vsctl list-ports s1
s1-eth1
s1-eth2
ovs@admin:~$
```

Figure 13. Listing switch *s1* ports.

**Step 5.** The ports that were returned in the previous command, which correspond to regular interfaces in Linux, can also be inspected by using the `ifconfig` tool as follows.

```
ifconfig s1-eth1; ifconfig s1-eth2
```

```

ovs@admin:~$ ifconfig s1-eth1; ifconfig s1-eth2
s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet6 fe80::d48b:c4ff:fecc:49c5 prefixlen 64 scopeid 0x20<link>
          ether d6:8b:c4:cb:49:c5 txqueuelen 1000 (Ethernet)
            RX packets 3 bytes 270 (270.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 339 bytes 68785 (68.7 KB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet6 fe80::5a:41:7:c6:ca:a5 prefixlen 64 scopeid 0x20<link>
          ether 5a:41:7:c6:ca:a5 txqueuelen 1000 (Ethernet)
            RX packets 3 bytes 270 (270.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 339 bytes 68785 (68.7 KB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ovs@admin:~$ 

```

Figure 14. Displaying Open vSwitch interfaces through `ifconfig`.

The figure above confirms that the switches' interfaces are visible in the operating system. This is useful because we can use management, monitoring, and traffic shaping tools, and apply them on the switch's interfaces. For instance, we can use the Network Emulator (`netem`) with the traffic control tool (`tc`) to emulate various network conditions such as delays, packet losses, jitter, etc.

**Step 6.** The switch daemon (`ovs-vswitchd`) stores its configuration in a database. Type the following command to print a brief overview of the database contents.

```
sudo ovs-vsctl show
```

```

ovs@admin:~$ sudo ovs-vsctl show
c34727ea-2286-4c47-baec-a842d33645c9
  Manager "ptcp:6640:127.0.0.1"
  Bridge "s1"
    fail_mode: standalone
    Port "s1"
      Interface "s1"
        type: internal
    Port "s1-eth1"
      Interface "s1-eth1"
    Port "s1-eth2"
      Interface "s1-eth2"

```

Figure 15. Printing an overview of database contents.

The figure above displays the bridge name (i.e., switch `s1`) and its connected interfaces (`s1`, `s1-eth1`, and `s1-eth2`). The interface `s1` is internal and can be used to assign an IP address to the bridge itself. The other interfaces connect the hosts `h1` and `h2` to the switch. Finally, the `fail_mode` is set to `standalone`, which we describe next.

#### 4 Open vSwitch fail-modes

Open vSwitch maintains flow tables that are consulted to determine how to forward traffic. The flow tables entries are typically populated by a controller. A controller is a centralized software that provides services to the forwarding planes. It might be the case where there is no connection to the controller, and hence, tables would not be populated, causing packets to be dropped at the switch.

Open vSwitch offers the option to operate in a standalone fail-mode. This means that if no connection to the controller is possible, or if messages have not been received from the controller for a time interval, Open vSwitch will take over responsibility for setting up flows. Consequently, the switch will operate as a regular MAC-learning switch. In this section, we will display and modify the fail-mode of the switch and understand its impact on traffic forwarding.

**Step 1.** By default, when adding a LegacySwitch from Miniedit, the Open vSwitch will operate in *standalone* fail-mode. When adding an OpenFlow switch from Miniedit, the Open vSwitch will operate in *secure* fail-mode. The figure below shows the LegacySwitch and the OpenFlow icons in MiniEdit and their corresponding fail-modes. Note that both switches are in fact OpenFlow switches.

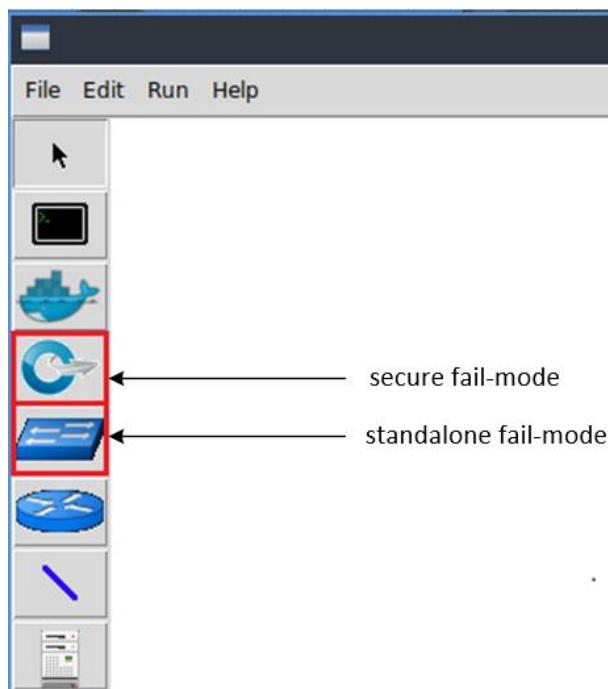
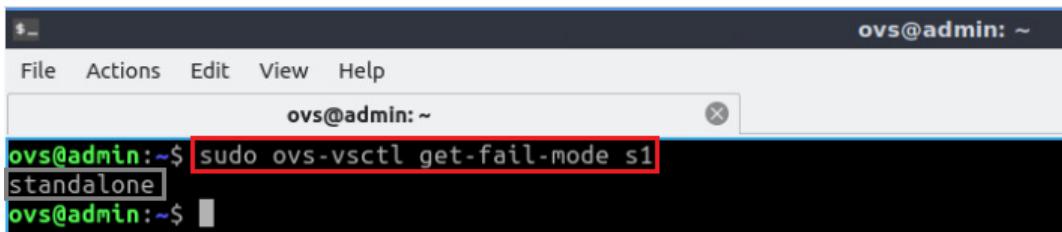


Figure 16. LegacySwitch and OpenFlow icons in Miniedit and their corresponding fail-modes.

**Step 2.** In this step, we will verify that the switch is operating as a standalone. Issue the command below to display the fail-mode of switch s1.

```
sudo ovs-vsctl get-fail-mode s1
```



```
ovs@admin:~$ sudo ovs-vsctl get-fail-mode s1
standalone
ovs@admin:~$
```

Figure 17. Displaying switch s1 operating mode.

The figure above confirms that the switch is running in standalone fail-mode.

**Step 3.** Hold right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

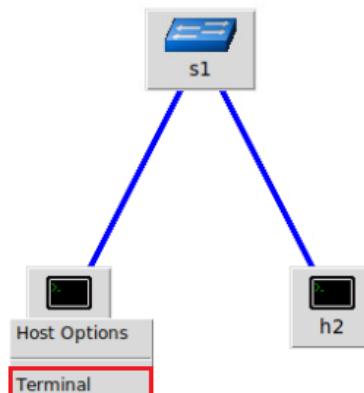
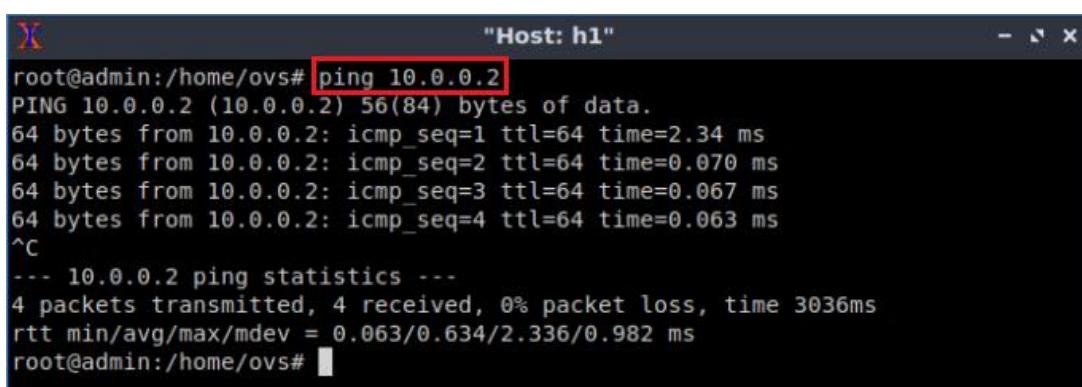


Figure 18. Opening host h1 terminal.

**Step 4.** Run a connectivity test between hosts h1 and h2 to check if the switch is successfully forwarding packets without the help of a controller. Type the command below on the terminal of host h1 to issue a ping test to host h2. Press **ctrl+d** to stop the test.

```
ping 10.0.0.2
```



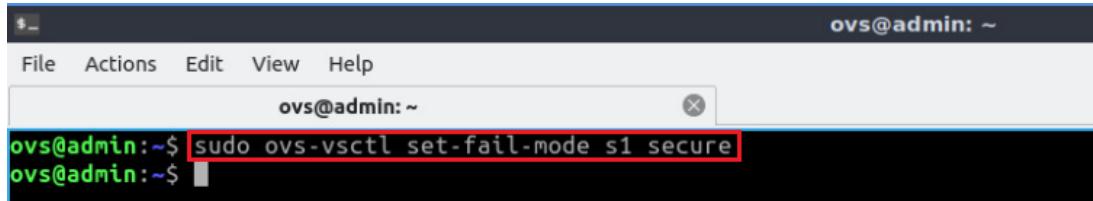
```
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.34 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.063 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3036ms
rtt min/avg/max/mdev = 0.063/0.634/2.336/0.982 ms
root@admin:/home/ovs#
```

Figure 19. Running a connectivity test between h1 and h2.

The following figure shows a successful connectivity test. This means that the switch is acting as a MAC-learning switch and can forward packets without having the controller to populate its flow tables.

**Step 5.** In this step, we will modify the switch fail-mode to secure. The secure fail-mode causes the switch not to set up flows on its own when the controller connection fails. Issue the command below to modify the fail-mode of switch s1 to secure.

```
sudo ovs-vsctl set-fail-mode s1 secure
```

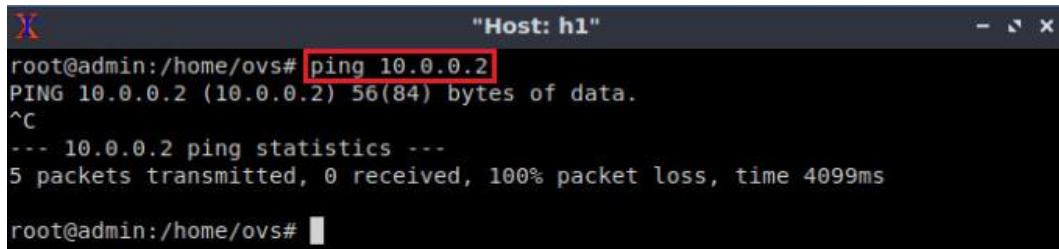


A screenshot of a terminal window titled "ovs@admin: ~". The window has a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu is a toolbar with icons for "File", "Actions", "Edit", "View", and "Help". The main area shows a command-line interface with the prompt "ovs@admin: ~\$". A red box highlights the command "sudo ovs-vsctl set-fail-mode s1 secure". The command is followed by a new line "ovs@admin: ~\$".

Figure 20. Changing the switch's mode from fail-mode to secure-mode.

**Step 6.** Run a connectivity test between hosts h1 and h2 to check if the switch is successfully forwarding packets without the help of a controller. Open the terminal of host h1 and type the command below to issue a ping test to host h2. Press **ctrl+c** to stop the test.

```
ping 10.0.0.2
```



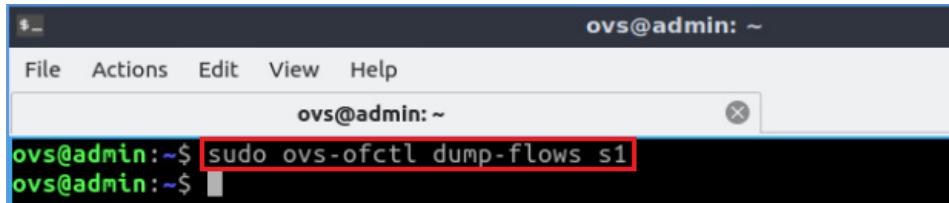
A screenshot of a terminal window titled "Host: h1". The window has a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu is a toolbar with icons for "File", "Actions", "Edit", "View", and "Help". The main area shows a command-line interface with the prompt "root@admin:/home/ovs#". A red box highlights the command "ping 10.0.0.2". The output shows the ping command being run, followed by statistics: "5 packets transmitted, 0 received, 100% packet loss, time 4099ms". The command is followed by a new line "root@admin:/home/ovs#".

Figure 21. Running a connectivity test between h1 and h2.

The figure above shows that there is no connectivity between hosts h1 and h2. Since we did not configure a controller in this lab, and since the switch is set to the secure fail-mode, the tables in the switch are not getting populated. Consequently, packets arriving to the switch are getting dropped.

**Step 7.** Type the following command to dump the flow table of switch s1.

```
sudo ovs-ofctl dump-flows s1
```



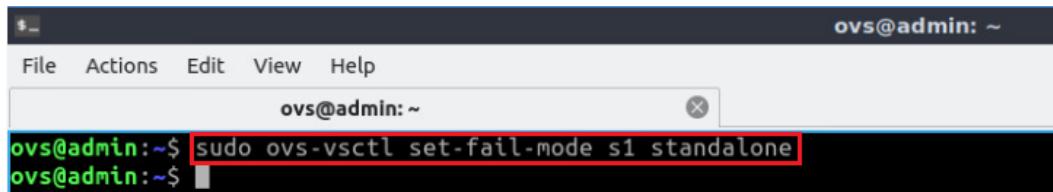
A screenshot of a terminal window titled "ovs@admin: ~". The window has a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu is a toolbar with icons for "File", "Actions", "Edit", "View", and "Help". The main area shows a command-line interface with the prompt "ovs@admin: ~\$". A red box highlights the command "sudo ovs-ofctl dump-flows s1". The command is followed by a new line "ovs@admin: ~\$".

Figure 22. Showing the active flow entries of switch s1.

Consider the figure above. The flow table of switch s1 is empty which indicates that switch s1 does not know how to forward the traffic.

**Step 8.** Change the fail-mode of the switch back to standalone by issuing the following command.

```
sudo ovs-vsctl set-fail-mode s1 standalone
```

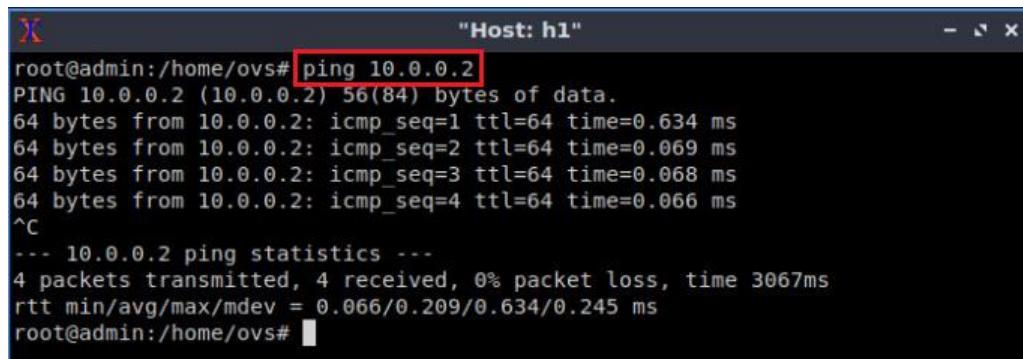


The screenshot shows a terminal window titled "ovs@admin: ~". The command "sudo ovs-vsctl set-fail-mode s1 standalone" is highlighted with a red box. The terminal prompt "ovs@admin:~\$" appears twice at the bottom.

Figure 23. Changing back the switch's mode from secure-mode to fail-mode.

**Step 9.** Verify that the connectivity between hosts h1 and h2 is restored by issuing the following command on host h1 terminal.

```
ping 10.0.0.2
```



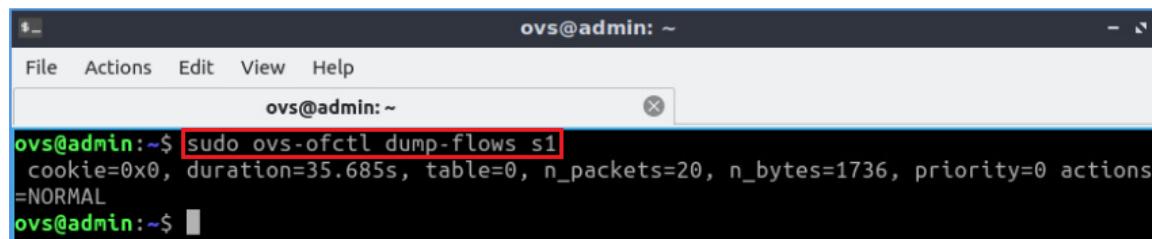
The screenshot shows a terminal window titled "Host: h1". The command "ping 10.0.0.2" is highlighted with a red box. The terminal shows the ping statistics for four packets transmitted to 10.0.0.2, with a round-trip time (rtt) ranging from 0.066 to 0.634 ms. The terminal prompt "root@admin:/home/ovs#" appears at the bottom.

Figure 24. Running a connectivity test between h1 and h2.

The figure above confirms that the connectivity between the hosts h1 and h2 is restored.

**Step 10.** Type the following command to dump the flow table of switch s1.

```
sudo ovs-ofctl dump-flows s1
```



The screenshot shows a terminal window titled "ovs@admin: ~". The command "sudo ovs-ofctl dump-flows s1" is highlighted with a red box. The output shows a single flow entry with a cookie of 0x0, a duration of 35.685 seconds, and 20 packets processed. The actions listed are NORMAL. The terminal prompt "ovs@admin:~\$" appears at the bottom.

Figure 25. Showing the active flow entries of switch s1.

The figure above shows the following information:

- **cookie:** a 64-bit number associated with a certain flow. The cookie field can be specified when adding/modifying flows. If not specified (as in the scenario above), the default cookie value of 0x0 is used.
- **duration:** the number of seconds since the entry was added.

- table: the table number (0 in this case). Table 0 corresponds to the first table in the pipeline and is matched against its entries when a packet arrives to the switch.
- n\_packets: the number of packets that had a *hit* on the flow entry.
- n\_bytes: the total number of bytes of packets that *hit* the flow entry.
- priority: a number between 0 and 65535 that indicates the priority of match in comparison to other matching entries. A higher value will match before a lower one.

actions: the action to take when a flow hits the entry. In the scenario above, the action is NORMAL, indicating that the packet is subject to the device's normal L2/L3 processing. This is why we have connectivity between the hosts h1 and h2.

## 5 Inspecting the state of the switch

In this section, you will use `ovs-ofctl` command line tool to inspect the state of the switch. The state in this context includes capabilities, configuration, and table entries.

**Step 1.** Issue the command below to display information pertaining to switch s1.

```
sudo ovs-ofctl show s1
```

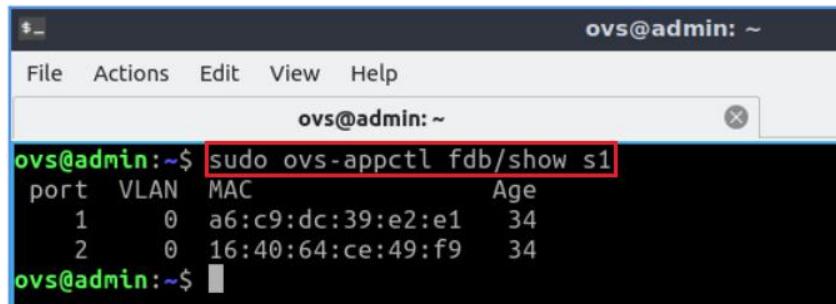
```
ovs@admin:~$ sudo ovs-ofctl show s1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst
mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
1(s1-eth1): addr:b2:5d:16:6b:5e:7f
    config: 0
    state: 0
    current: 10GB-FD COPPER
    speed: 10000 Mbps now, 0 Mbps max
2(s1-eth2): addr:46:4a:bc:77:80:c6
    config: 0
    state: 0
    current: 10GB-FD COPPER
    speed: 10000 Mbps now, 0 Mbps max
LOCAL(s1): addr:56:01:d5:01:9d:4a
    config: PORT_DOWN
    state: LINK_DOWN
```

Figure 26. Showing switch s1 information.

Consider the figure above. Switch s1 has three interfaces. Each interface displays the Media Access Control (MAC) address (`addr`) and other information, such as the current state of the switch.

**Step 2.** It is possible to inspect the learned MAC entries on the switch. Type the following command to display the MAC addresses that were learned on switch s1.

```
sudo ovs-appctl fdb/show s1
```



The screenshot shows a terminal window titled 'ovs@admin: ~'. The command 'sudo ovs-appctl fdb/show s1' is run, displaying a table of learned MAC addresses:

port	VLAN	MAC	Age
1	0	a6:c9:dc:39:e2:e1	34
2	0	16:40:64:ce:49:f9	34

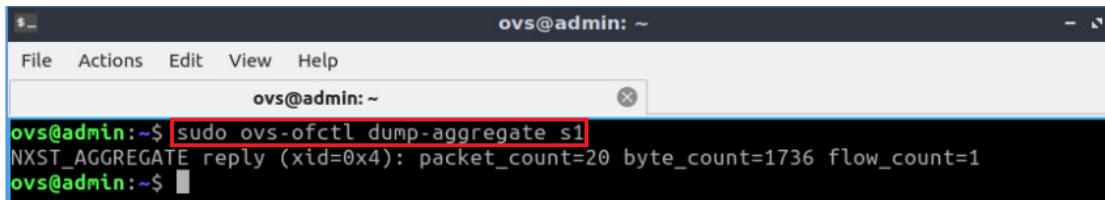
Figure 27. Verifying MAC addresses on switch s1.

Consider the figure above. The table contains each MAC address/VLAN pair learned by switch s1, along with the port on which it was learned. Aging time defines the period an entry is in the table, in seconds. The default VLAN ID is set to 0. The MAC addresses of ports 1 and 2 shown in the output above refer to the MAC address of hosts h1 and h2, respectively.

Perform the connectivity test between hosts h1 and h2 again if you do not get the expected output since MAC entries are deleted after a certain period of time.

**Step 3.** Type the following command to display flow information for all interfaces. This command aggregates the stats of all ports and display the totals.

```
sudo ovs-ofctl dump-aggregate s1
```



The screenshot shows a terminal window titled 'ovs@admin: ~'. The command 'sudo ovs-ofctl dump-aggregate s1' is run, displaying the following output:

```
NXST_AGGREGATE reply (xid=0x4): packet_count=20 byte_count=1736 flow_count=1
```

Figure 28. Displaying flow information in switch s1.

Consider the figure above. The figure shows the information about all interfaces (total packets, bytes, drop, errors). You will notice the number of total packets (20), bytes (1736) received in switch s1.

**Step 4.** Type the following command to display statistics for the flow tables in switch s1.

```
sudo ovs-ofctl dump-tables s1
```

```
ovs@admin:~$ sudo ovs-ofctl dump-tables s1
OFPST_TABLE reply (xid=0x2):
table 0:
active=1, [lookup=20] [matched=18]
max_entries=1000000
matching:
  in_port: exact match or wildcard
  eth_src: exact match or wildcard
  eth_dst: exact match or wildcard
  eth_type: exact match or wildcard
  vlan_vid: exact match or wildcard
  vlan_pcp: exact match or wildcard
  ip_src: exact match or wildcard
  ip_dst: exact match or wildcard
  nw_proto: exact match or wildcard
  nw_tos: exact match or wildcard
  tcp_src: exact match or wildcard
  tcp_dst: exact match or wildcard

table 1:
active=0, lookup=0, matched=0
(same features)
```

Figure 29. Displaying flow tables in switch s1.

Consider the figure above. By default, Open vSwitch uses OpenFlow table 0 to save the flows. The figure shows the number of lookup (20) and the number of matches (18) found in table 0.

You might get different output depending on how many packets were transferred between hosts h1 and h2.

This concludes Lab 3. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. Paul Emmerich, Daniel Raumer, Florian Wohlfart, Georg Carle, “*Performance characteristics of virtual switching*”, IEEE 3<sup>rd</sup> International conference on cloud networking (CloudNet), 2014.
2. Linux foundation, “*Open vSwitch*”, [Online]. Available: <http://openvSwitch.org>.
3. RFC 7047, “*The open vSwitch database management protocol*”, Dec 2013.
4. IBM, “*Virtual networking in Linux*”, [Online]. Available: <https://developer.ibm.com/tutorials/l-virtual-networking/>
5. B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, S. Shenker, “*Extending networking into the virtualization layer*”, 8<sup>th</sup> ACM Workshop on Hot Topics in Networks (HotNets-VIII). New York City, NY, 2009.
6. Mininet walkthrough, [Online]. Available: <http://mininet.org>.



## OPEN VIRTUAL SWITCH

### Lab 4: Open vSwitch Flow Table

Document Version: **06-29-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    Introduction to OpenFlow.....	4
1.2    OpenFlow architecture .....	4
1.3    OpenFlow flow table and packet matching .....	5
2    Lab topology.....	6
2.1    Lab settings.....	6
2.2    Loading a topology .....	6
2.3    Loading the configuration file .....	8
3    Verifying IP addresses on the hosts.....	9
4    Enabling traditional switch forwarding operation.....	11
5    Enabling traffic forwarding using layer 1 data.....	13
6    Enabling traffic forwarding using layer 2 data.....	14
7    Enabling traffic forwarding using layer 3 data.....	16
8    Enabling traffic forwarding using layer 4 data.....	18
9    Setting match priority .....	20
References .....	24

## Overview

This lab discusses the concept of OpenFlow, a protocol designed to manage and direct traffic among routers and switches manufactured by various vendors. This lab aims to demonstrate how to manage flows manually in an Open Virtual Switch (Open vSwitch) connected to two emulated hosts.

## Objectives

By the end of this lab, you should be able to:

1. Understand the behavior of the OpenFlow protocol.
2. Inspect the flow table entries.
3. Enable packet forwarding by inserting flow entries manually.
4. Set match priorities over existing flows.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Verifying IP addresses on the hosts.
4. Section 4: Enabling traditional switch forwarding operation.
5. Section 5: Enabling traffic forwarding using layer 1 data.
6. Section 6: Enabling traffic forwarding using layer 2 data.
7. Section 7: Enabling traffic forwarding using layer 3 data.
8. Section 8: Enabling traffic forwarding using layer 4 data.
9. Section 9: Setting match priority.

## 1      Introduction

Traditional routing schemes do not change flow paths once the paths are selected. Due to network topology's irregularity and the randomness of traffic, some switches have more flows passing through than other switches. Therefore, these switches' flow-table utilization is usually higher than others, which becomes the bottleneck of the network<sup>1</sup>. OpenFlow addresses bottlenecks to high performance and scalability. In a traditional switch, packet forwarding (the data plane) and routing (the control plane) occur on the same device. While in OpenFlow, the data plane is decoupled from the control plane, providing more efficiency and operational agility<sup>7</sup>.

## 1.1 Introduction to OpenFlow

OpenFlow is the communication protocol between OpenFlow switches and controllers. An OpenFlow switch communicates with a controller which is responsible for managing the switch via the OpenFlow protocol. An OpenFlow switch consists of one or more flow tables which perform packet lookups and forwarding operations. Using the OpenFlow protocol, the controller can manipulate the flow entries in flow tables.

Consider Figure 1. There is an OpenFlow controller that communicates with one or more OpenFlow switches. When a packet arrives at a switch, the switch looks for matched flow entries in the flow tables and executes the corresponding lists of actions. If no match is found, the packet is sent to the controller. The controller responds with a new flow entry for handling the packet.

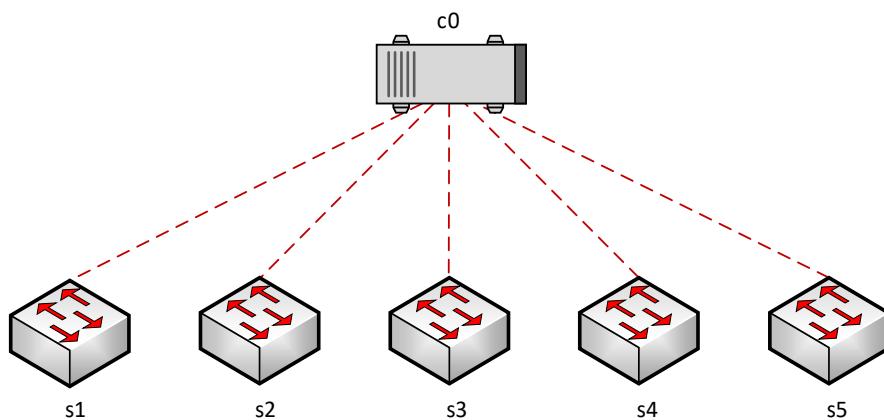


Figure 1. OpenFlow components.

## 1.2 OpenFlow architecture

OpenFlow switches perform packet forwarding using the packet-matching function within the flow table. Thus, once a packet arrives at the switch, the latter will look up in its flow table and check if there is a match. Consequently, the switch will decide which action to take based on the flow table<sup>8</sup>. The action could be:

- Forward the packet to another port.

- Drop the packet.
- Pass the packet to the controller.

The flows can be installed manually within the switch if there is no controller connected to it. The flows are installed in Open vSwitch daemon (Open vSwitch-vSwitchd) that controls the switch and implements the OpenFlow protocol. `ovs-ofctl` command line tool is required for monitoring and administering switches that support OpenFlow protocol.

Figure 2 shows the basic functions of an OpenFlow switch and its relationship to a controller. When the data plane does not match the incoming packet, it sends a `packet_in` message to the controller. The control plane runs routing and switching protocols and other logic to determine the forwarding tables and logic in the data plane. Consequently, when the controller has a data packet to forward through the switch, it uses the OpenFlow `packet_out` message. The flow entry is then stored in the flow table located in the switch. If there is no controller connected to the switch, the switch will look up into its flow table and takes action based on the flow entries manually stored in the switch. If there is no match in the flow table, the switch will drop the packet.

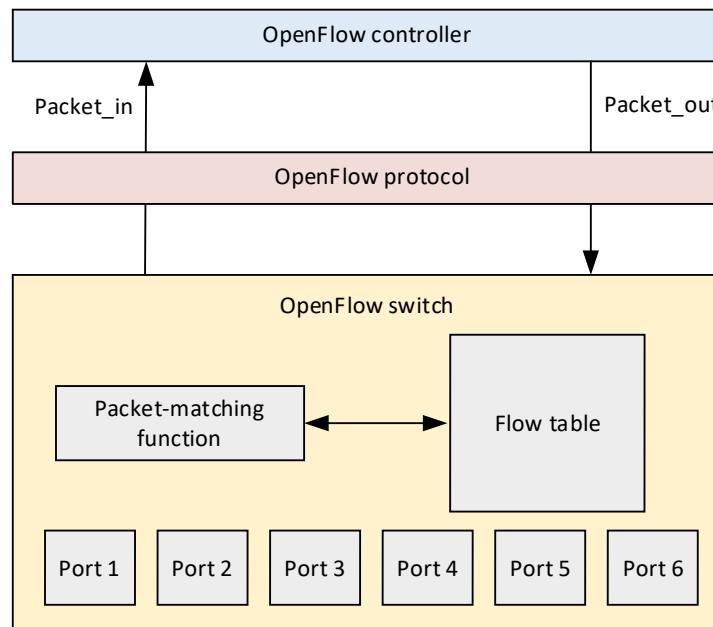


Figure 2. OpenFlow packet forwarding architecture.

### 1.3 OpenFlow flow table and packet matching

Each flow table contains a set of flow entries that consist of match fields, counters, and a set of instructions. An Open vSwitch may contain more than one flow table. The switch starts matching at the first flow table and continues to check additional flow tables to find a match. By default, all the flow entries are stored in the first table (table 0), if the table number is not specified for an entry. Packets match against the packet header fields such as switch input port, VLAN ID, Ethernet source/destination addresses, IP source/destination addresses, IP protocol, source/destination ports. If a matching entry

is found in the table, the instructions associated with that specific flow entry are executed<sup>8</sup>.

Flow entries match packets in priority order, higher priority entries must match before lower priority ones. If multiple flow entries have the same priority, the switch will choose any order.

## 2 Lab topology

Consider Figure 3. The topology consists of two hosts and one switch. All the hosts belong to the same network, 10.0.0.0/8.

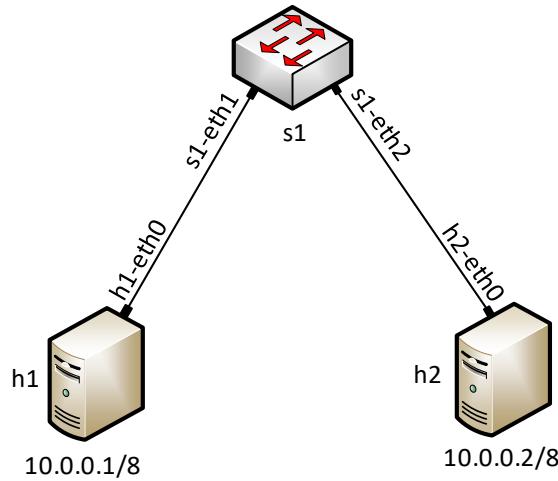


Figure 3. Lab topology.

### 2.1 Lab settings

The hosts are configured according to Table 2.

Table 2. Topology information.

Host	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8

### 2.2 Loading a topology

**Step 1.** Start by launching MiniEdit by clicking on desktop's shortcut. When prompted for a password, type `password`.

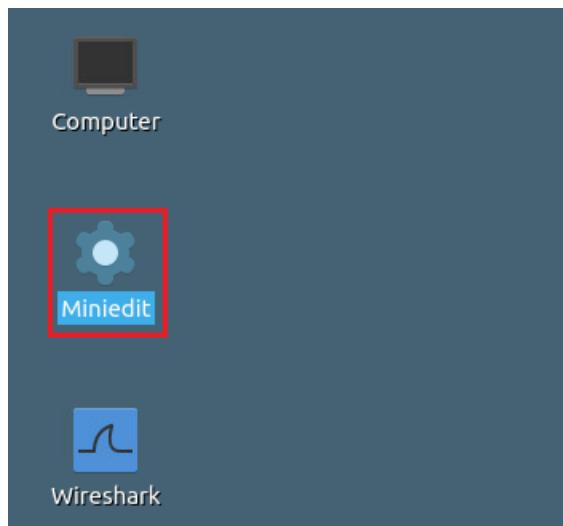


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then *open* to load the lab's topology. Locate the *Lab4.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab4* and click on *Open*.

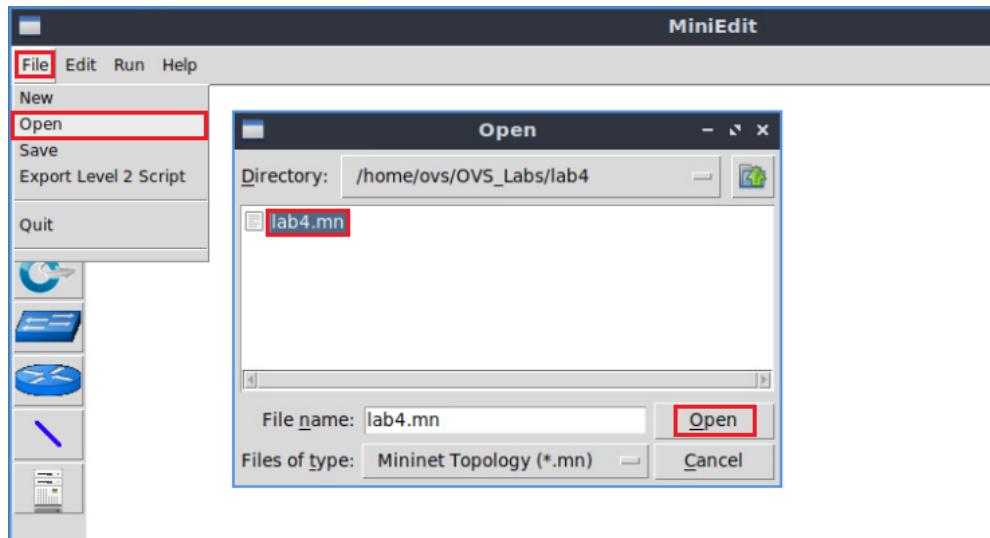


Figure 5. MiniEdit's Open dialog.

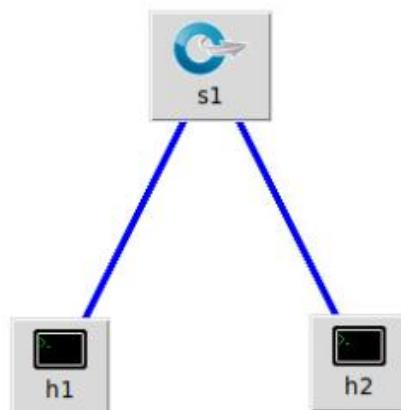


Figure 6. MiniEdit's topology.

**Step 3.** To proceed with the emulation, click on the *Run* button located in the lower left-hand side.



Figure 7. Starting the emulation.

**Step 4.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

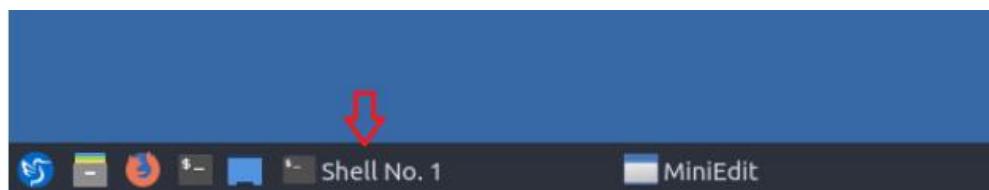


Figure 8. Opening Mininet's terminal.

**Step 5.** Issue the following command to display the interface names and connections.

```
links
```

A screenshot of the Mininet terminal window titled 'Shell No. 1'. The user has entered the command 'links' and the output shows network connections: 'h1-eth0<->s1-eth1' (OK OK) and 's1-eth2<->h2-eth0' (OK OK). The line 'links' is highlighted with a red box.

Figure 9. Displaying network interfaces.

In Figure 9, the link displayed within the gray box indicates that interface eth0 of host h1 connects to interface eth1 of switch s1 (i.e.,  $h1\text{-}eth0 <-> s1\text{-}eth1$ ).

## 2.3 Loading the configuration file

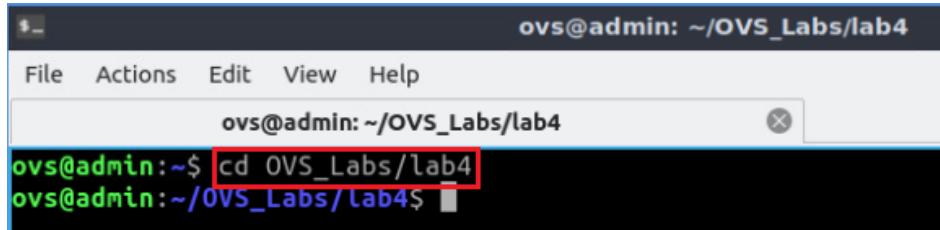
**Step 1.** Open the Linux terminal.



Figure 10. Opening Linux terminal.

**Step 2.** Click on the Linux's terminal and navigate into *OVS\_Labs/lab4* directory by issuing the following command.

```
cd OVS_Labs/lab4
```

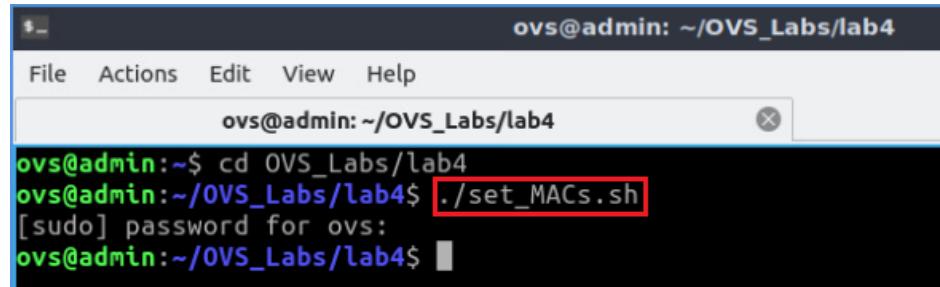


The screenshot shows a terminal window titled "ovs@admin: ~/OVS\_Labs/lab4". The user has typed "cd OVS\_Labs/lab4" at the prompt. The command is highlighted with a red box. The terminal window has a dark background with white text and a light gray header bar.

Figure 11. Entering to the *OVS\_Labs/lab4* directory.

**Step 3.** This folder contains a configuration file that will assign easy-to-read Media Access Control (MAC) addresses to the hosts' interfaces. To execute the shell script, type the following command. When prompted for a password, type password.

```
./set_MACs.sh
```

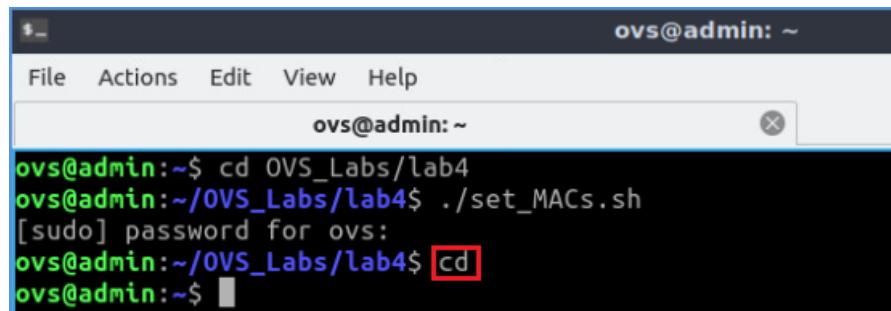


The screenshot shows a terminal window titled "ovs@admin: ~/OVS\_Labs/lab4". The user has typed "./set\_MACs.sh" at the prompt. The command is highlighted with a red box. A password prompt "[sudo] password for ovs:" is visible above the command line. The terminal window has a dark background with white text and a light gray header bar.

Figure 12. Executing the shell script to load the configuration.

**Step 4.** Type the following command to exit from the lab4 directory and go back to the home directory.

```
cd
```



The screenshot shows a terminal window titled "ovs@admin: ~". The user has typed "cd" at the prompt. The command is highlighted with a red box. The terminal window has a dark background with white text and a light gray header bar.

Figure 13. Exiting from the directory.

### 3 Verifying IP addresses on the hosts

In this section, you will verify that the IP addresses on the hosts are assigned according to table 2.

**Step 1.** Hold right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

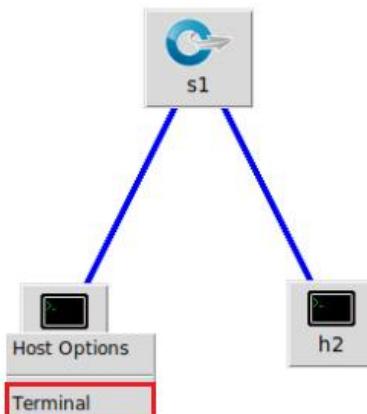


Figure 14. Opening a terminal on host h1.

**Step 2.** In host h1 terminal, type the following command to verify that the IP address was assigned successfully. You will verify the host interface *h1-eth0* is configured with the IP address 10.0.0.1 and the subnet mask 255.0.0.0. You will also verify the MAC address, 00:00:00:00:00:01.

```
ifconfig
```

```
"Host: h1"
root@admin:/home/ovs# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0
        ether 00:00:00:00:00:01 txqueuelen 1000 (Ethernet)
            RX packets 60 bytes 11002 (11.0 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 4 bytes 356 (356.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
            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

root@admin:/home/ovs#
```

Figure 15. Verifying the IP address, subnet mask, and MAC address of the host.

**Step 3.** In host h2 terminal, type the following command to verify that the IP address was assigned successfully. You will verify the host interface *h2-eth0* is configured with the IP address 10.0.0.2 and the subnet mask 255.0.0.0. You will also verify the MAC address, 00:00:00:00:00:02.

```
ifconfig
```

```
"Host: h2"
root@admin:/home/ovs# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet [10.0.0.2] netmask 255.0.0.0 broadcast 0.0.0.0
        ether [00:00:00:00:00:02] txqueuelen 1000 (Ethernet)
        RX packets 71 bytes 13371 (13.3 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 3 bytes 270 (270.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        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

root@admin:/home/ovs# "
```

Figure 16. Verifying the IP address, subnet mask, and MAC address of the host.

**Step 4.** On host h1 terminal, test the connectivity between host h1 and host h2 using the `ping` command. To stop the test, press `Ctrl+d`.

```
ping 10.0.0.2
```

```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 0 received, +6 errors, 100% packet loss, time 6135ms
pipe 4
root@admin:/home/ovs# "
```

Figure 17. Output of `ping` command.

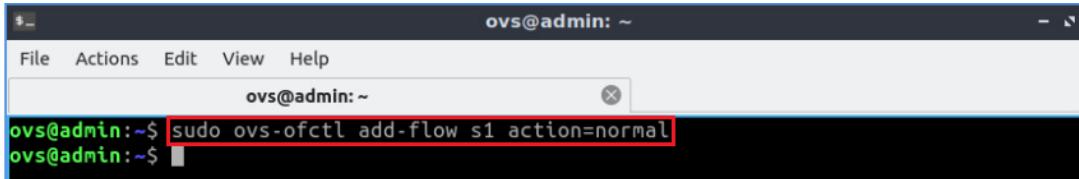
Hosts cannot ping each other as the *fail\_mode* of the switch is *secure* and the flow table in the switch is empty at this point. In secure fail\_mode, the flow table in the switch will not be populated if there is no connection to the controller.

## 4 Enabling traditional switch forwarding operation

In this section, you will enable the traditional switch forwarding operation in switch s1.

**Step 1.** Type the following command to add a flow in switch s1. The `ovs-ofctl` program is a command-line tool for monitoring and administering OpenFlow switches. An action of a flow indicates an action to take when a packet matches the flow entry. A *normal* action allows the device to conduct normal layer 2/layer 3 packet processing.

```
sudo ovs-ofctl add-flow s1 action=normal
```

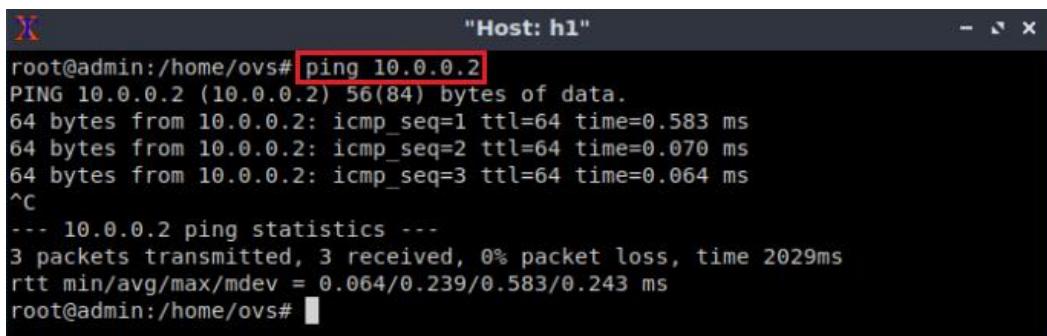


```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 action=normal
ovs@admin:~$
```

Figure 18. Adding normal flow in switch s1.

**Step 2.** On host h1 terminal, test the connectivity between host h1 and host h2 using the `ping` command.

```
ping 10.0.0.2
```



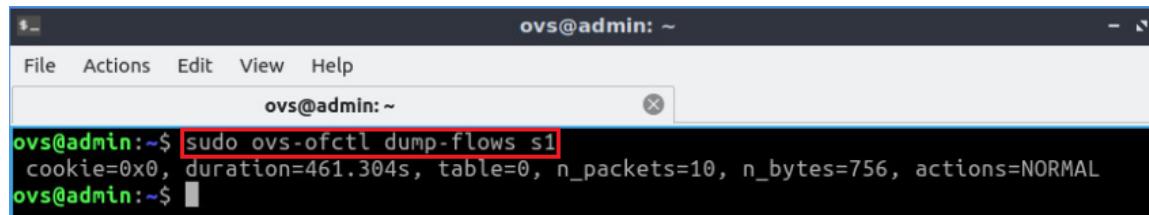
```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.583 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.064 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2029ms
rtt min/avg/max/mdev = 0.064/0.239/0.583/0.243 ms
root@admin:/home/ovs#
```

Figure 19. Output of `ping` command.

The figure shows a successful connectivity test. To stop the test, press `Ctrl+c`.

**Step 3.** Type the following command to verify the flow installation. This command prints the flow table entries in switch s1. The output depicts the configuration parameters when the forwarding action is set to normal.

```
sudo ovs-ofctl dump-flows s1
```



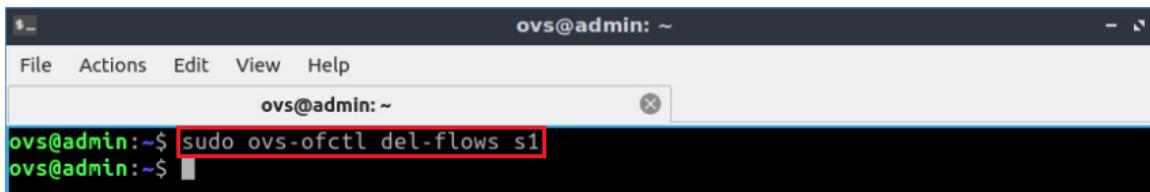
```
ovs@admin:~$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=461.304s, table=0, n_packets=10, n_bytes=756, actions=NORMAL
ovs@admin:~$
```

Figure 20. Verifying flow in switch s1.

Consider the figure above. The normal action allows the switch to conduct normal layer 2/layer 3 packet processing.

**Step 4.** Type the following command to delete all existing flows in switch s1 so that you can add another flow entry in the following section.

```
sudo ovs-ofctl del-flows s1
```



```
ovs@admin:~$ sudo ovs-ofctl del-flows s1  
ovs@admin:~$
```

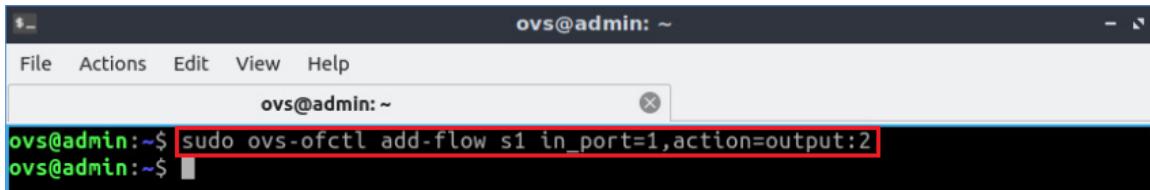
Figure 21. Deleting existing flows from switch s1.

## 5 Enabling traffic forwarding using layer 1 data

In this section, you will work at the physical ports level. You will program the switch so that everything that comes at switch s1 from port 1 is sent out to port 2, and vice versa.

**Step 1.** Type the following command to add a flow in switch s1. The command indicates the traffic coming from port 1 (*s1-eth1*) has to be forwarded to port 2 (*s1-eth2*).

```
sudo ovs-ofctl add-flow s1 in_port=1,action=output:2
```

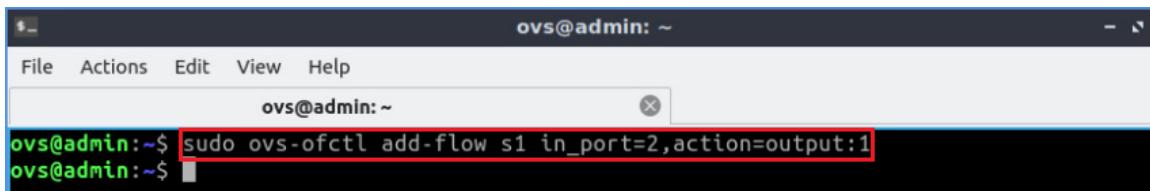


```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 in_port=1,action=output:2  
ovs@admin:~$
```

Figure 22. Adding a port forwarding flow entry.

**Step 2.** Type the following command to add a flow in switch s1. The command indicates the traffic coming from port 2 (*s1-eth2*) has to be forwarded to port 1 (*s1-eth1*).

```
sudo ovs-ofctl add-flow s1 in_port=2,action=output:1
```

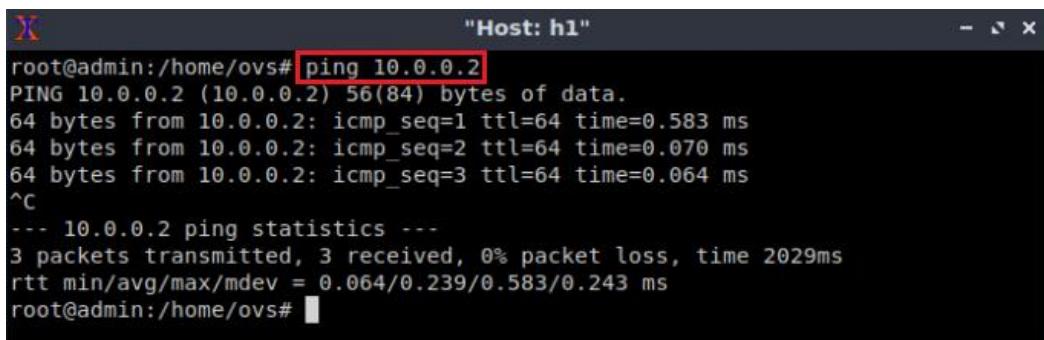


```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 in_port=2,action=output:1  
ovs@admin:~$
```

Figure 23. Adding a port forwarding flow entry.

**Step 3.** On host h1 terminal, test the connectivity between host h1 and host h2 using the `ping` command.

```
ping 10.0.0.2
```



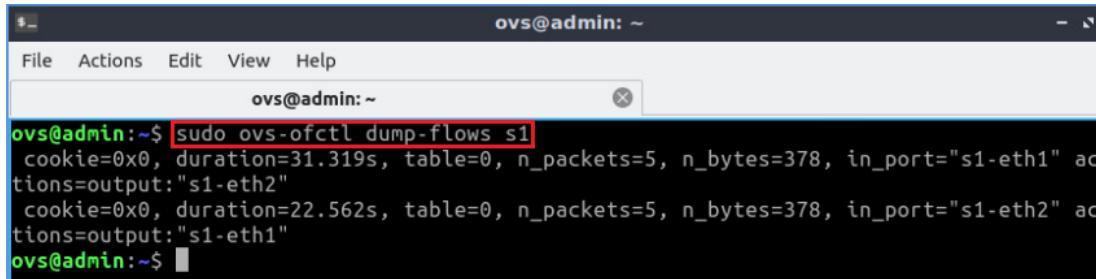
```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.583 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.064 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2029ms
rtt min/avg/max/mdev = 0.064/0.239/0.583/0.243 ms
root@admin:/home/ovs#
```

Figure 24. Output of `ping` command.

The figure shows a successful connectivity test. To stop the test, press `Ctrl+c`.

**Step 4.** Type the following command to verify the flow installation This command prints the flow table entries in switch s1.

```
sudo ovs-ofctl dump-flows s1
```



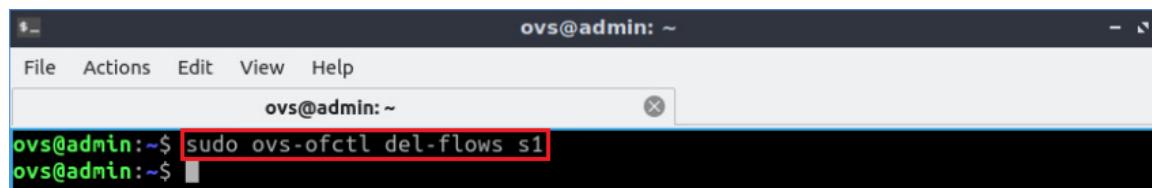
```
ovs@admin:~$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=31.319s, table=0, n_packets=5, n_bytes=378, in_port="s1-eth1" actions=output:"s1-eth2"
cookie=0x0, duration=22.562s, table=0, n_packets=5, n_bytes=378, in_port="s1-eth2" actions=output:"s1-eth1"
ovs@admin:~$
```

Figure 25. Verifying flows in switch s1.

You will notice two flow entries installed on the switch. Packets coming from port *s1-eth1* are sent out to port *s1-eth1*, and vice versa.

**Step 5.** Type the following command to delete all existing flows in switch s1 so that you can add another flow entry in the following section.

```
sudo ovs-ofctl del-flows s1
```



```
ovs@admin:~$ sudo ovs-ofctl del-flows s1
ovs@admin:~$
```

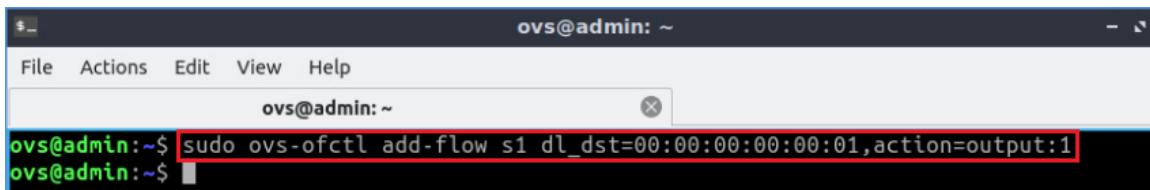
Figure 26. Deleting existing flows from switch s1.

## 6 Enabling traffic forwarding using layer 2 data

In this section, you will create flow entries based on the MAC addresses of the hosts.

**Step 1.** Type the following command to insert a flow entry in switch s1.

```
sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:01,action=output:1
```



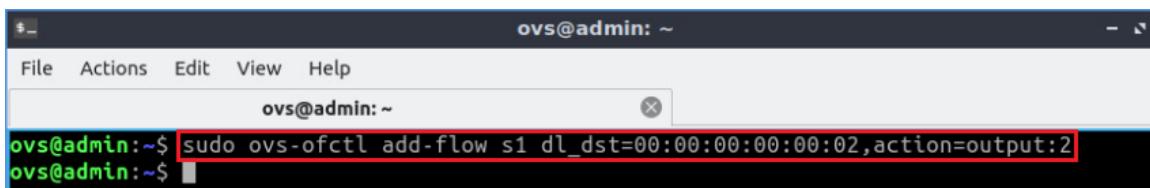
```
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:01,action=output:1
ovs@admin:~$
```

Figure 27. Adding a MAC based flow entry.

Consider the figure above. The flow specifies that the switch will match against MAC destination address. Traffic going to the destination host h1 will be forwarded to switch port *s1-eth1*.

**Step 2.** Type the following command to insert a flow entry in switch s1.

```
sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:02,action=output:2
```



```
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ovs-ofctl add-flow s1 dl_dst=00:00:00:00:00:02,action=output:2
ovs@admin:~$
```

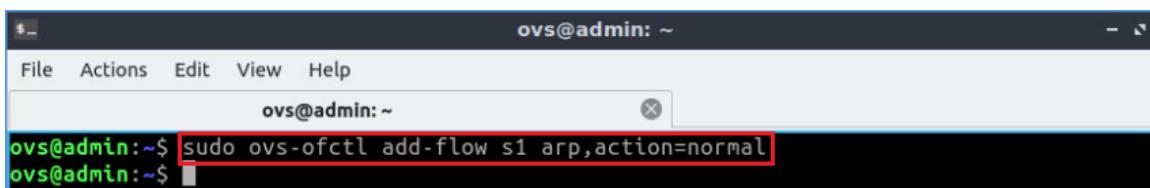
Figure 28. Adding a MAC based flow entry.

Consider the figure above. The flow specifies that the switch will match against MAC address. Traffic going to the destination host h2 will be forwarded to switch port *s1-eth2*.

Hosts cannot ping at this moment since Address Resolution Protocol (ARP) is required to find out the MAC address of a different host. You will add a flow that allows ARP requests in the following step in order to get successful connectivity between hosts.

**Step 3.** Type the following command to add a flow to allow ARP requests.

```
sudo ovs-ofctl add-flow s1 arp,action=normal
```



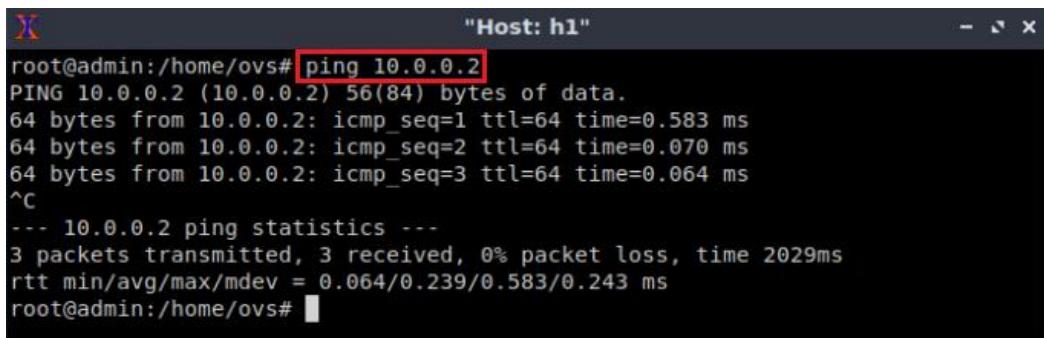
```
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ovs-ofctl add-flow s1 arp,action=normal
ovs@admin:~$
```

Figure 29. Adding a flow to allow ARP requests.

Consider the figure above. The command adds a flow that sends ARP requests to all the switch ports.

**Step 4.** In host h1 terminal, test the connectivity between host h1 and host h2 using the `ping` command.

```
ping 10.0.0.2
```



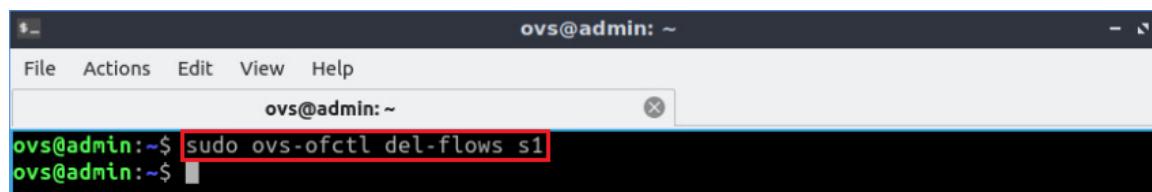
```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.583 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.064 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2029ms
rtt min/avg/max/mdev = 0.064/0.239/0.583/0.243 ms
root@admin:/home/ovs#
```

Figure 30. Output of `ping` command.

The figure shows a successful connectivity test. To stop the test, press `Ctrl+c`.

**Step 5.** Type the following command to delete all existing flows in the switch s1 so that you can add another flow entry in the following section.

```
sudo ovs-ofctl del-flows s1
```



```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ovs-ofctl del-flows s1
ovs@admin:~$
```

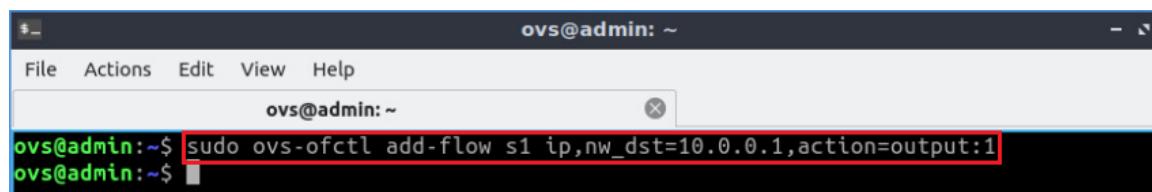
Figure 31. Deleting existing flows from switch s1.

## 7 Enabling traffic forwarding using layer 3 data

In this section, you will create flow entries based on IP addresses.

**Step 1.** Type the following command to insert a flow entry in switch s1.

```
sudo ovs-ofctl add-flow s1 ip,nw_dst=10.0.0.1,action=output:1
```



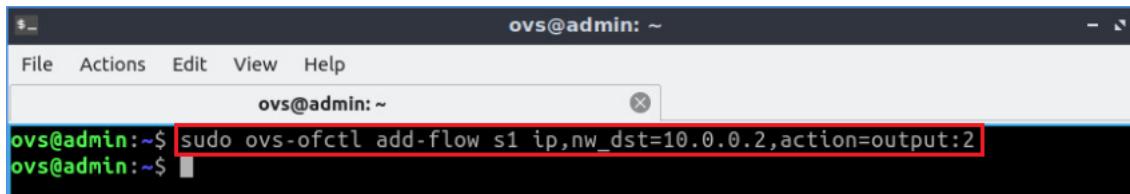
```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ovs-ofctl add-flow s1 ip,nw_dst=10.0.0.1,action=output:1
ovs@admin:~$
```

Figure 32. Adding an IP-based flow entry.

Consider the figure above. The flow specifies that the switch will match against destination IP 10.0.0.1. Traffic going to the destination 10.0.0.1 will be forwarded to switch port `s1-eth1`.

**Step 2.** Type the following command to insert a flow entry in switch s1.

```
sudo ovs-ofctl add-flow s1 ip,nw_dst=10.0.0.2,action=output:2
```



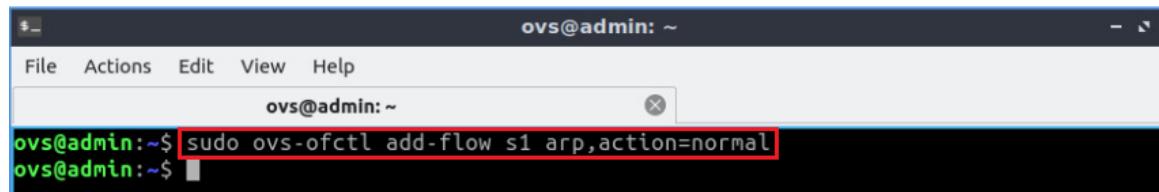
```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 ip,nw_dst=10.0.0.2,action=output:2  
ovs@admin:~$
```

Figure 33. Adding an IP-based flow entry.

Consider the figure above. The flow specifies that the switch will match against destination IP 10.0.0.2. Traffic going to the destination 10.0.0.2 will be forwarded to switch port *s1-eth2*.

**Step 3.** Type the following command to add a flow to allow ARP requests.

```
sudo ovs-ofctl add-flow s1 arp,action=normal
```

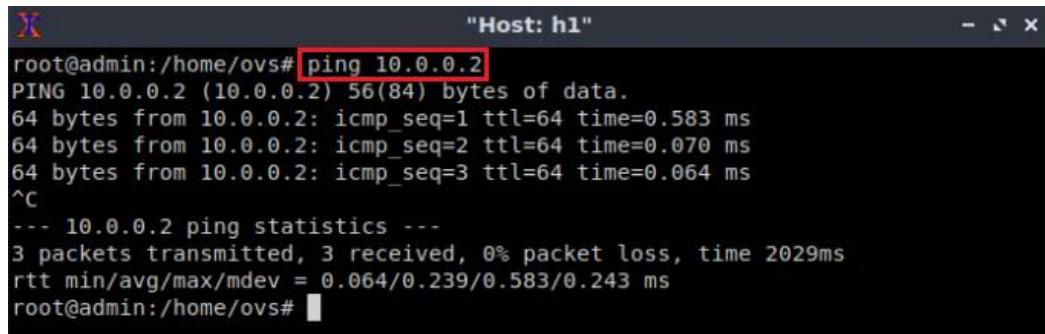


```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 arp,action=normal  
ovs@admin:~$
```

Figure 34. Adding an ARP flow entry.

**Step 4.** In host h1 terminal, test the connectivity between host h1 and host h2 using the `ping` command.

```
ping 10.0.0.2
```



```
"Host: h1"  
root@admin:/home/ovs# ping 10.0.0.2  
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.  
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.583 ms  
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.070 ms  
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.064 ms  
^C  
--- 10.0.0.2 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2029ms  
rtt min/avg/max/mdev = 0.064/0.239/0.583/0.243 ms  
root@admin:/home/ovs#
```

Figure 35. Output of `ping` command.

The figure shows a successful connectivity test. To stop the test, press `Ctrl+c`.

**Step 5.** Type the following command to verify the flow installation in switch s1.

```
sudo ovs-ofctl dump-flows s1
```

```
ovs@admin:~$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=41.727s, table=0, n_packets=3, n_bytes=294, ip,nw_dst=10.0.0.1 actions=output:"s1-eth1"
cookie=0x0, duration=30.975s, table=0, n_packets=3, n_bytes=294, ip,nw_dst=10.0.0.2 actions=output:"s1-eth2"
cookie=0x0, duration=21.055s, table=0, n_packets=4, n_bytes=168, arp actions=NORMAL
ovs@admin:~$
```

Figure 36. Verifying flow in switch s1.

Consider the figure above. IP based flow entries are installed in the flow table.

**Step 6.** Type the following command to delete all existing flows in switch s1 so that you can add another flow entry in the following section.

```
sudo ovs-ofctl del-flows s1
```

```
ovs@admin:~$ sudo ovs-ofctl del-flows s1
ovs@admin:~$
```

Figure 37. Deleting existing flows from switch s1.

## 8 Enabling traffic forwarding using layer 4 data

In this section, you will work at the application layer. A simple python web server will be executed in host h2, and host h1 will connect to that server that runs at port 80.

**Step 1.** Type the following command to start an HTTP server in host h2 named *SimpleHTTPServer* which is listening to port 80.

```
python -m SimpleHTTPServer 80 &
```

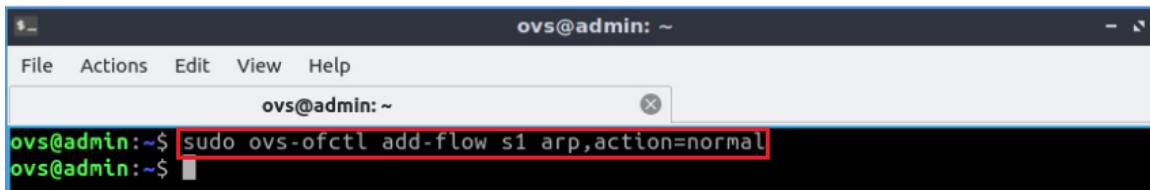
```
"Host: h2"
root@admin:/home/ovs# python -m SimpleHTTPServer 80 &
[2] 23184
[1]  Exit 1                  python -m HTTPserver 80
root@admin:/home/ovs# Serving HTTP on 0.0.0.0 port 80 ...
```

Figure 38. Starting a python web server.

Consider the figure above. Host h2 is serving as an HTTP server, listening to port 80.

**Step 2.** Type the following command to insert an ARP flow in switch s1.

```
sudo ovs-ofctl add-flow s1 arp,action=normal
```



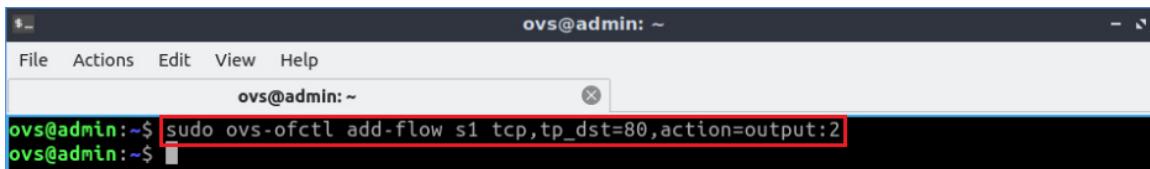
```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 arp,action=normal  
ovs@admin:~$
```

Figure 39. Adding a flow to allow ARP requests.

Consider the figure above. The command adds a flow that sends ARP requests to all the switch ports.

**Step 3.** Type the following command to add a flow that forwards all TCP traffic with destination port 80 ( $tp\_dst=80$ ), to the switch port  $s1\text{-}eth2$ .

```
sudo ovs-ofctl add-flow s1 tcp,tp_dst=80,action=output:2
```

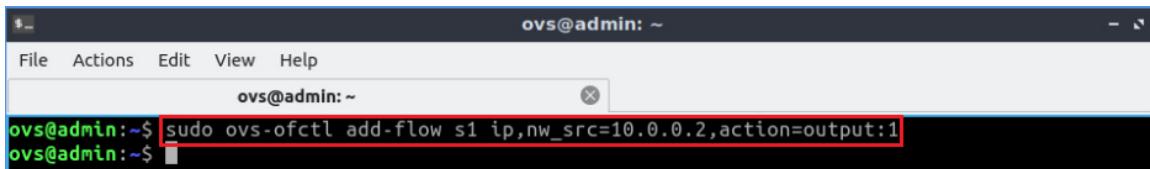


```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 tcp,tp_dst=80,action=output:2  
ovs@admin:~$
```

Figure 40. Adding a flow entry.

**Step 4.** Type the following command to add a flow in switch s1.

```
sudo ovs-ofctl add-flow s1 ip,nw_src=10.0.0.2,action=output:1
```



```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 ip,nw_src=10.0.0.2,action=output:1  
ovs@admin:~$
```

Figure 41. Adding a flow entry.

Consider the figure above. The IP based flow will be matched against source address. If there is a match, the traffic will be forwarded to switch port  $s1\text{-}eth1$ .

**Step 5.** In host h1 terminal, issue an HTTP request to host h2 using `curl` command.

```
curl 10.0.0.2
```

```
"Host: h1"
root@admin:/home/ovs# curl 10.0.0.2
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a>
<li><a href=".bash_logout">.bash_logout</a>
<li><a href=".bashrc">.bashrc</a>
<li><a href=".bashrc.bak">.bashrc.bak</a>
<li><a href=".cache/">.cache/</a>
<li><a href=".config/">.config/</a>
<li><a href=".dbus/">.dbus/</a>
<li><a href=".gnupg/">.gnupg/</a>
<li><a href=".history_frr">.history_frr</a>
<li><a href=".karaf/">.karaf/</a>
<li><a href=".local/">.local/</a>
<li><a href=".mininet_history">.mininet_history</a>
<li><a href=".mozilla/">.mozilla/</a>
<li><a href=".new.sh.swo">.new.sh.swo</a>
```

Figure 42. Output of `curl` command.

Consider the figure above. This is a basic example of `curl` command that simulates a GET request for a website URL. This command shows the output of the HTTP response from host h2 in HTML format.

**Step 6.** In host h2 terminal, see the output for the `curl` command.

```
"Host: h2"
root@admin:/home/ovs# python -m SimpleHTTPServer 80 &
[2] 23184
[1]  Exit 1                  python -m HTTPserver 80
root@admin:/home/ovs# Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.1 - - [22/Jun/2021 13:42:37] "GET / HTTP/1.1" 200 -
```

Figure 43. Output of `curl` command.

The figure shows the response sending to host h1 (10.0.0.1).

**Step 7.** Type the following command to delete all existing flows in switch s1 so that you can add another flow entry in the following section.

```
sudo ovs-ofctl del-flows s1
```

```
ovs@admin: ~
ovs@admin: ~
ovs@admin:~$ sudo ovs-ofctl del-flows s1
ovs@admin:~$
```

Figure 44. Deleting existing flows from switch s1.

## 9 Setting match priority

In OpenFlow, packets are matched against flow entries based on prioritization. A priority has a value of 0 to 65535. Higher priority entries must match before lower priority ones. In this section, you will add a flow that has priority over an existing flow and verify how the higher priority works over the lower ones.

**Step 1.** Type the following command to add a MAC based flow entry in switch s1 with a priority. The command indicates that the traffic going to the destination host h2 will be forwarded to the port *s1-eth2*.

```
sudo ovs-ofctl add-flow s1  
priority=400,dl_dst=00:00:00:00:00:02,action=output:2
```

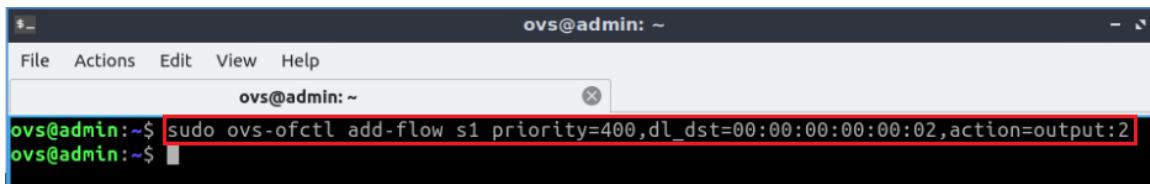


Figure 45. Adding a MAC based flow entry.

**Step 2.** Type the following command to insert a MAC based flow entry in switch s1 with a priority. The command indicates that the traffic going to the destination host h1 will be forwarded to the port *s1-eth1*.

```
sudo ovs-ofctl add-flow s1  
priority=400,dl_dst=00:00:00:00:00:01,action=output:1
```

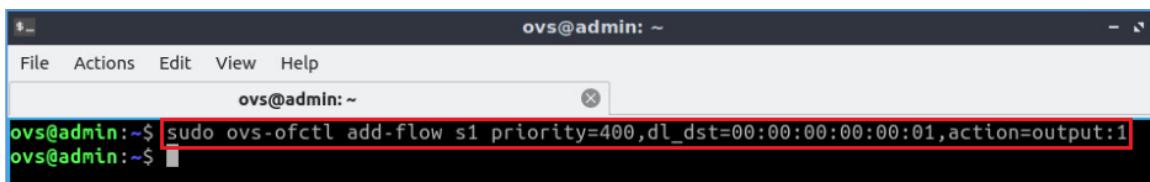


Figure 46. Adding a MAC based flow entry.

**Step 3.** Type the following command to add an IP based flow entry in switch s1 with higher priority. The command indicates that the traffic going to the destination host h2 will be dropped.

```
sudo ovs-ofctl add-flow s1 ip,priority=500,nw_dst=10.0.0.2,action=drop
```

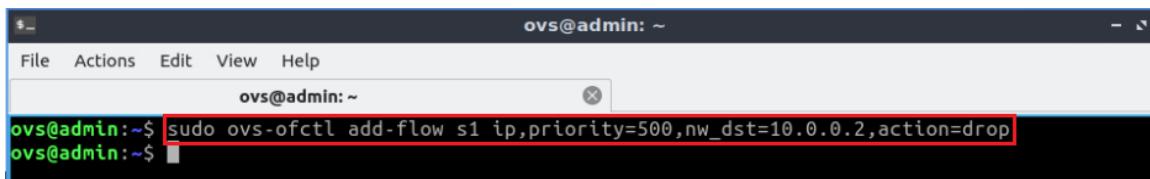
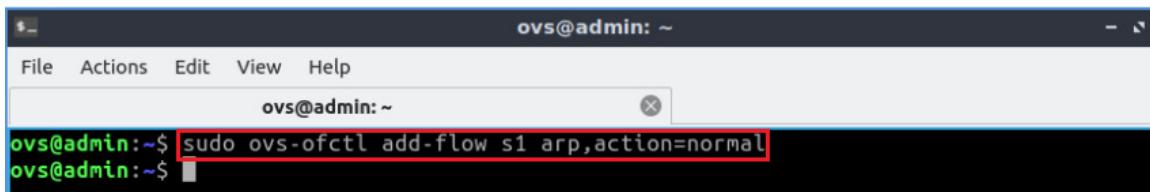


Figure 47. Adding a flow in switch s1.

Consider the command above. The priority has been set to 500, greater than the previous flow (400). Based on the flow definition, there will be no connectivity between hosts h1 and h2 since the flow has a higher priority.

**Step 4.** Type the following command to insert an ARP flow in switch s1.

```
sudo ovs-ofctl add-flow s1 arp,action=normal
```

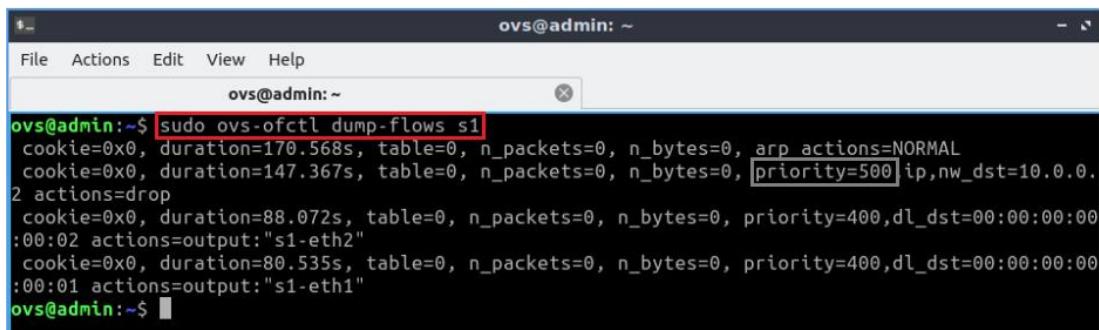


A screenshot of a terminal window titled "ovs@admin: ~". The command "sudo ovs-ofctl add-flow s1 arp,action=normal" is being typed into the terminal. The command is highlighted with a red box.

Figure 48. Adding a flow to allow ARP requests.

**Step 5.** Type the following command to verify flow installation in switch s1.

```
sudo ovs-ofctl dump-flows s1
```



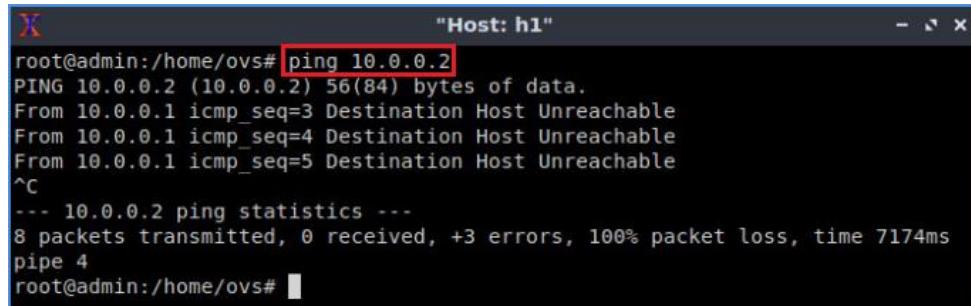
A screenshot of a terminal window titled "ovs@admin: ~". The command "sudo ovs-ofctl dump-flows s1" is being run. The output shows several flows installed on interface s1. One flow has a priority of 500 and is for ARP requests. Another flow has a priority of 400 and is for IP traffic destined to 10.0.0.0. Both flows have actions of drop.

Figure 49. Verifying flows in switch s1.

The figure above shows flows with different priorities.

**Step 6.** In host h1 terminal, test the connectivity between hosts h1 and h2 using the `ping` command. There is no connectivity between hosts h1 and h2 since the switch is dropping the traffic. To stop the test, press `Ctrl+c`.

```
ping 10.0.0.2
```

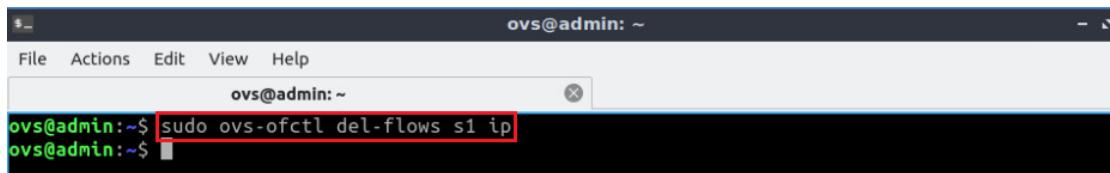


A screenshot of a terminal window titled "Host: h1". The command "ping 10.0.0.2" is being run. The output shows three ICMP echo requests being sent to 10.0.0.2, all of which are dropped by the switch. The host receives no replies. The command is highlighted with a red box.

Figure 50. Output of `ping` command.

**Step 7.** Type the following command to delete an existing flow with higher priority.

```
sudo ovs-ofctl del-flows s1 ip
```



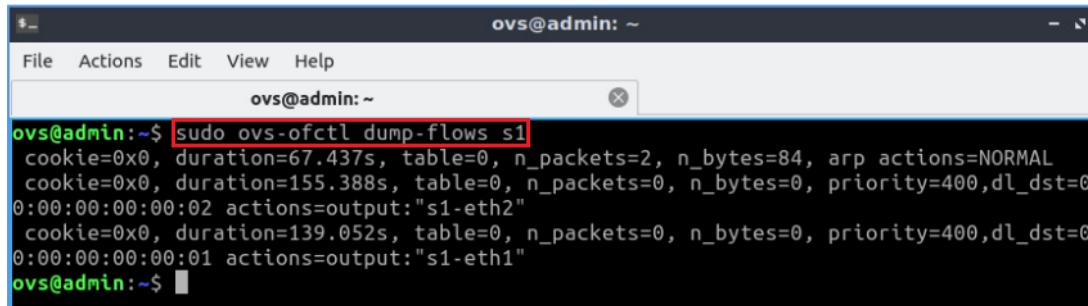
```
ovs@admin:~$ sudo ovs-ofctl del-flows s1 ip
ovs@admin:~$
```

Figure 51. Deleting an existing flow from switch s1.

Consider the figure above. We can delete flows from the flow table based on match priorities. The command indicates to delete all the IP based flows. The IP based flow with priority 500 should be deleted from the flow table.

**Step 8.** Type the following command to verify flow installation in switch s1.

```
sudo ovs-ofctl dump-flows s1
```



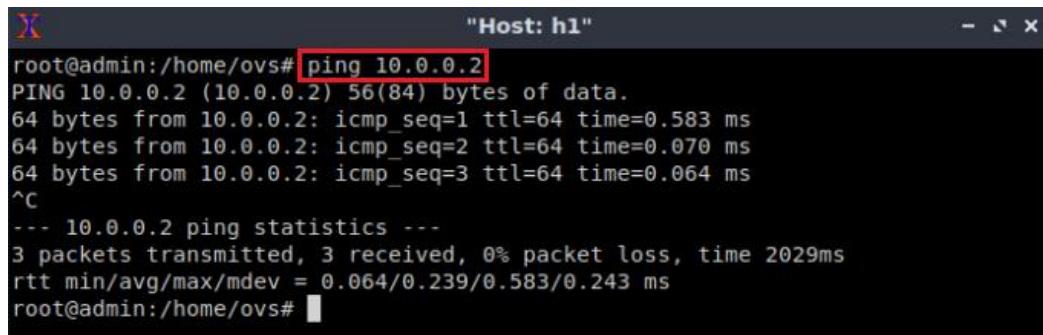
```
ovs@admin:~$ Sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=67.437s, table=0, n_packets=2, n_bytes=84, arp actions=NORMAL
cookie=0x0, duration=155.388s, table=0, n_packets=0, n_bytes=0, priority=400,dl_dst=0
0:00:00:00:00:02 actions=output:"s1-eth2"
cookie=0x0, duration=139.052s, table=0, n_packets=0, n_bytes=0, priority=400,dl_dst=0
0:00:00:00:00:01 actions=output:"s1-eth1"
ovs@admin:~$
```

Figure 52. Verifying flows in switch s1.

Consider the figure above. The IP based flow with priority 500 does not exist in the flow table anymore. Only MAC based flows with priority 400 are present in the table. Hosts h1 and h2 should have successful connectivity between them at this point.

**Step 9.** In host h1 terminal, test the connectivity between host h1 and host h2 using the `ping` command.

```
ping 10.0.0.2
```



```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.583 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.070 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.064 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2029ms
rtt min/avg/max/mdev = 0.064/0.239/0.583/0.243 ms
root@admin:/home/ovs#
```

Figure 53. Output of `ping` command.

The figure shows a successful connectivity test. To stop the test, press `Ctrl+c`.

This concludes Lab 4. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. Zehua Guo, Yang Xu, Ruoyan Liu, Andrey Gushchin, Kuan-yin Chen, Anwar Walid, H. Jonathan Chao, “*Balancing flow table occupancy and link utilization in software-defined networks*”, Dec 2018.
2. Linux Foundation, “*Open vSwitch*”, [Online]. Available: <http://openvSwitch.org>.
3. RFC 7047, “*The open vSwitch database management protocol*”, Dec 2013.
4. IBM, “*Archived / Virtual networking in Linux*”, [Online]. Available: <https://developer.ibm.com/tutorials/l-virtual-networking/>
5. Mininet walkthrough, [Online]. Available: <http://mininet.org>.
6. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. “*OpenFlow: enabling innovation in campus networks.*” ACM SIGCOMM Computer Communication Review 38, no.2 (2008):69-74.
7. Aria Zhu, “*OpenFlow switch: what is it and how does it work?*”, [Online]. Available: <https://medium.com/@AriaZhu/openflow-switch-what-is-it-and-how-does-it-work-7589ea7ea29c#:~:text=OpenFlow%20switch%20is%20designed%20to,hardware%20it%27s%20intended%20to%20control>.
8. Open Networking Foundation, “*OpenFlow Switch Specification*”, [Online]. Available: <https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf>



## OPEN VIRTUAL SWITCH

### Exercise 1: OpenFlow Basic Operations

Document Version: **09-17-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

1	Exercise topology .....	3
1.1	Topology settings .....	3
1.2	Credentials .....	3
2	Deliverables.....	4

## 1 Exercise topology

Consider Figure 1. The topology consists of three hosts and three switches. All the hosts belong to the same network, 10.0.0.0/8.

The goal of this exercise is to manage the flow entries in Open vSwitches using `ovs-ofctl` command line tool.

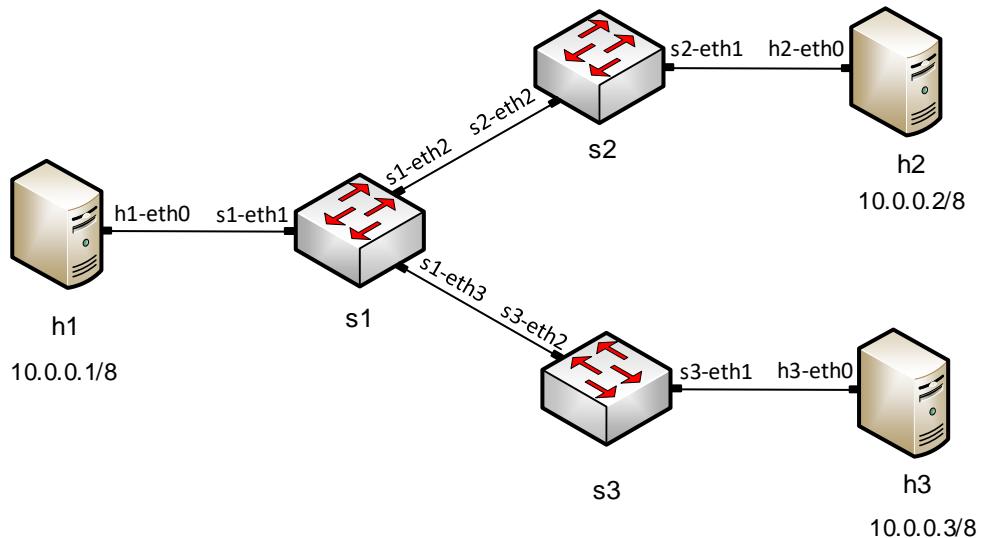


Figure 1. Exercise topology.

### 1.1 Topology settings

The devices are already configured according to Table 1.

Table 1. Topology information.

Host	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8
h3	h3-eth0	10.0.0.3	/8

### 1.2 Credentials

The information in Table 2 provides the credentials to access the Client's virtual machine.

Table 2. Credentials to access the Client's virtual machine.

Device	Account	Password

Client	admin	password
--------	-------	----------

## 2 Deliverables

Follow the steps below to complete the exercise.

- a) Start MiniEdit by clicking on MiniEdit's shortcut. Load the topology *Exercise1.mn* located at *~/OVS\_Labs/Exercise1* as shown in the figure below.

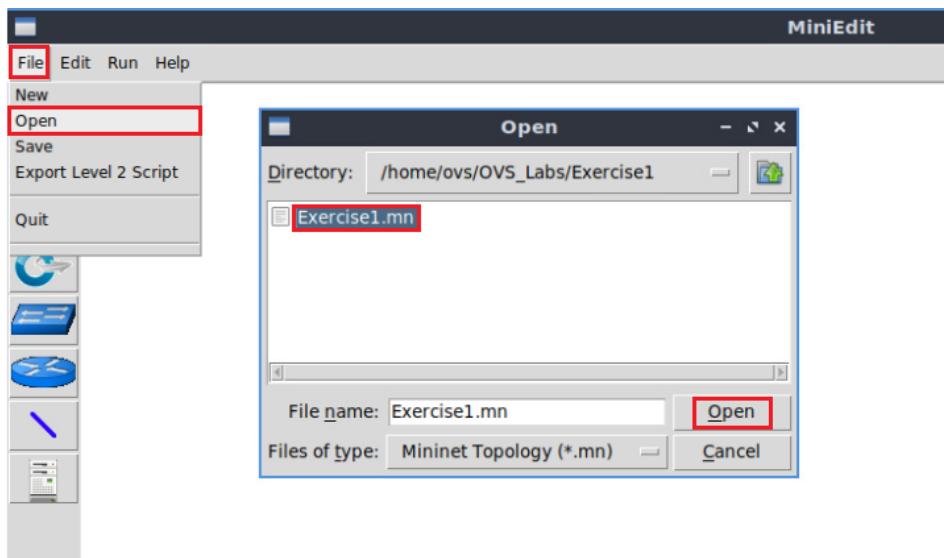


Figure 2. Loading the topology file in Mininet.

- b) Run the emulation in Mininet.
- c) In the Mininet terminal, launch the command that displays the interface names and connections of the current topology. Verify that links conform to the topology in Figure 1.
- d) *~/OVS\_Labs/Exercise1* folder contains a script *set\_MACs.sh* responsible for loading the MAC addresses. Execute the script using the following command:

```
cd OVS_Labs/Exercise1
```

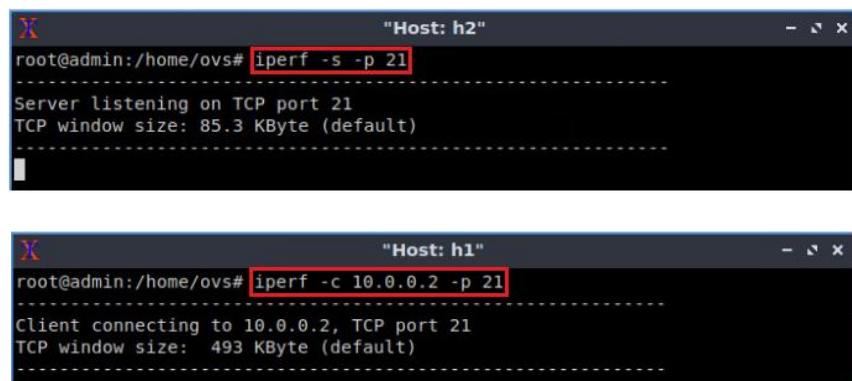
```
./set_MACs.sh
```

- e) Configure port based forwarding in switches s2 and s3. Traffic generated from host h1 should be forwarded to host h2/h3, and vice versa.
- f) In switch s1, create flow entries based on the MAC addresses of the hosts. Verify connectivity between the hosts using the `ping` command.
- g) Delete all the flow entries in switch s1. Create flow entries based on destination IP addresses. Verify connectivity between the hosts using the `ping` command.

**h)** Delete all the flow entries in switch s1.

**i)** Consider hosts h2 and h3 as FTP and HTTP servers, respectively. In switch s1, create flow entries to match against TCP port numbers (port 21 for FTP, port 80 for HTTP). Whenever there is a match against port 21, the switch will forward the traffic to host h2. If there is a match against port 80, the destination host is h3.

**j)** Verify the configuration between hosts h1 and h2. Run an iperf test where host h2 is running as an FTP server and host h1 is running as an FTP client. Following figures show how to run an iperf test between two hosts.



The image shows two terminal windows side-by-side. The left window is titled "Host: h2" and shows the command "iperf -s -p 21" being run. The output indicates that the server is listening on TCP port 21 with a default TCP window size of 85.3 KByte. The right window is titled "Host: h1" and shows the command "iperf -c 10.0.0.2 -p 21" being run. The output indicates that the client is connecting to host 10.0.0.2 on TCP port 21 with a default TCP window size of 493 KByte.

```
"Host: h2"
root@admin:/home/ovs# iperf -s -p 21
-----
Server listening on TCP port 21
TCP window size: 85.3 KByte (default)

-----
```

```
"Host: h1"
root@admin:/home/ovs# iperf -c 10.0.0.2 -p 21
-----
Client connecting to 10.0.0.2, TCP port 21
TCP window size: 493 KByte (default)
```

**k)** Run an iperf test where host h3 is running as an HTTP server (port 80) and host h1 is running as a client.



## OPEN VIRTUAL SWITCH

### Lab 5: Implementing Routing in Open vSwitch

Document Version: **07-13-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    Routing in Open vSwitch .....	4
2    Lab topology.....	5
2.1    Lab settings.....	5
2.2    Loading a topology .....	6
2.3    Loading the configuration file .....	8
3    Verifying IP addresses on the hosts .....	9
4    Enabling routing in switch s1 .....	13
5    Enabling routing in switch s2 .....	14
6    Verifying configuration .....	16
References .....	21

## Overview

This lab aims to demonstrate how to manually configure routing tables in Open vSwitch to enable packet forwarding between different networks.

## Objectives

By the end of this lab, the student should be able to:

1. Understand the concept of routing.
2. Understand OpenFlow protocol.
3. Understand how Open vSwitch implements routing between two different networks.
4. Configure manual flows to enable packet forwarding between different networks.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Verifying IP addresses on the hosts.
4. Section 4: Enabling routing in switch s1.
5. Section 5: Enabling routing in switch s2.
6. Section 6: Verifying configuration.

## 1      Introduction

A router is responsible for delivering packets across different networks. It requires a routing protocol to exchange routing information with other routers<sup>1</sup>. This information changes over time-based on the network configuration. If any link fails, routers can pick a new route to a particular network. An end device cannot communicate directly with devices outside of the local network. When a host sends a packet to another host on a

different network, the packet is forwarded to the default gateway, which routes traffic out of the local network.

## 1.1 Routing in Open vSwitch

Open vSwitch relies on OpenFlow protocol to implement routing. OpenFlow switches perform packet forwarding using the packet-matching function within the flow table. Thus, once a packet arrives at the switch, the latter will look up in its flow table and check if there is a match. Consequently, the switch will decide which action to take based on the flow table<sup>11</sup>. The action could be:

- Forward the packet out to another port.
- Drop the packet.
- Pass the packet to the controller.

The flows can be installed manually within the switch if there is no controller connected to the switch. The flows are installed in the Open vSwitch daemon (Open vSwitch-vSwitchd) that controls the switch and implements the OpenFlow protocol. `ovs-ofctl` command-line tool is required for monitoring and administering switches that support OpenFlow protocol.

Each flow table contains a set of flow entries that consist of match fields, counters, and a set of instructions. An Open vSwitch may contain more than one flow table. The switch starts matching at the first flow table and continues to check additional flow tables to find a match. By default, all the flow entries are stored in the first table (table 0) if the table number is not specified for an entry. Packets match against the packet header fields such as switch input port, VLAN ID, Ethernet source/destination addresses, IP source/destination addresses, IP protocol, source/destination ports. If a matching entry is found in a table, the instructions associated with that specific flow entry are executed<sup>11</sup>.

Figure 1 shows the basic functions of an OpenFlow switch and its relationship to a controller. When the data plane does not match the incoming packet, it sends a *packet\_in* message to the controller. The control plane runs routing and switching protocols and other logic to determine the forwarding tables and logic in the data plane. Consequently, when the controller has a data packet to forward out through the switch, it uses the OpenFlow *packet\_out* message. The flow entry is then stored in the flow table located in the switch. If there is no controller connected to the switch, the switch will look up in its flow table and takes action based on the flow entries manually stored in the switch. If there is no match in the flow table, the switch will drop the packet.

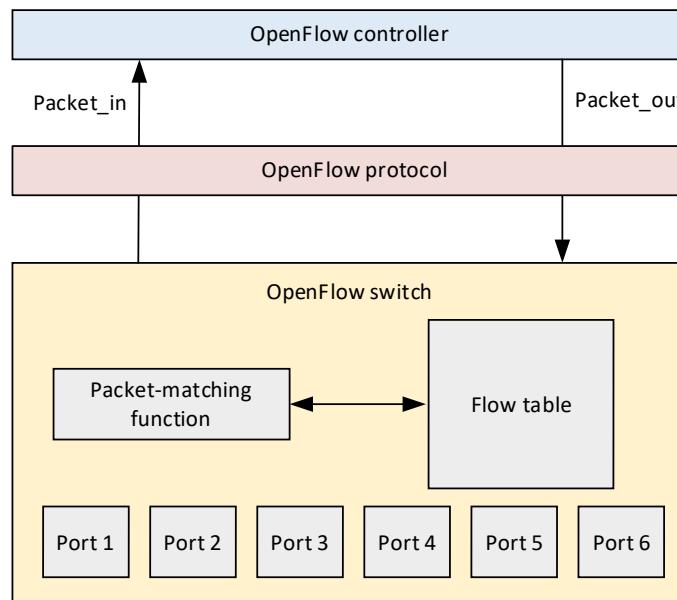


Figure 1. OpenFlow packet forwarding architecture.

## 2 Lab topology

Consider Figure 2. The topology consists of two end-hosts and two switches. Hosts h1 and h2 belong to the networks 192.168.1.0/24 and 192.168.2.0/24, respectively.

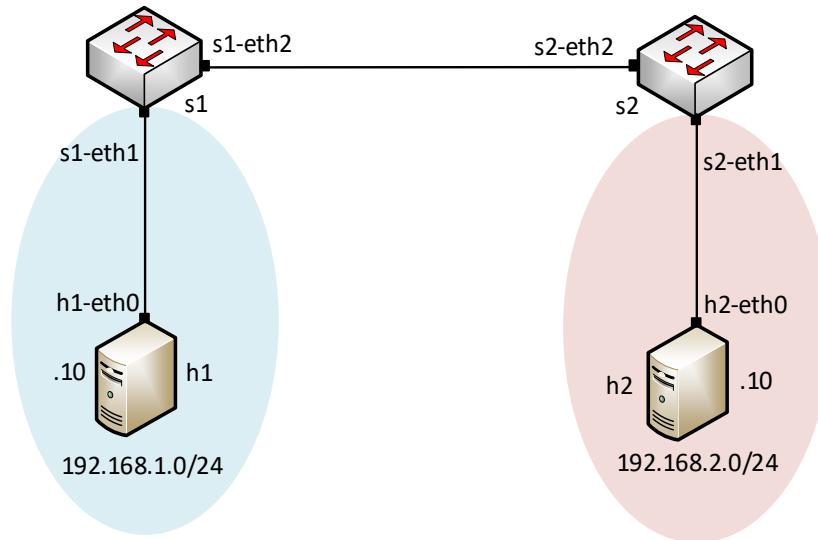


Figure 2. Lab topology.

### 2.1 Lab settings

The devices are already configured according to Table 2.

Table 2. Topology information.

Device	Interface	IP Address	Subnet	Default gateway
h1	h1-eth0	192.168.1.10	/24	192.168.1.1
h2	h2-eth0	192.168.2.10	/24	192.168.2.1

## 2.2 Loading a topology

**Step 1.** Start by launching MiniEdit by clicking on desktop's shortcut. When prompted for a password, type password.

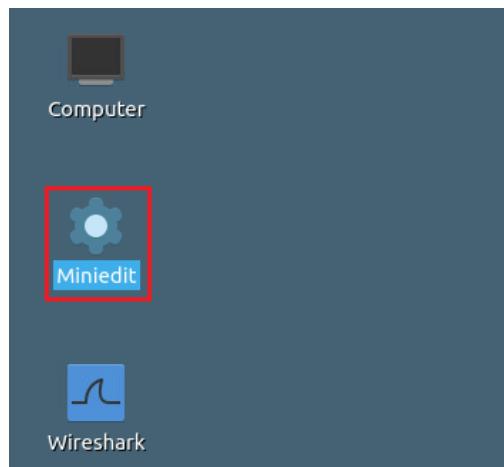


Figure 3. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then *open* to load the lab's topology. Locate the *Lab5.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab5* and click on *Open*.

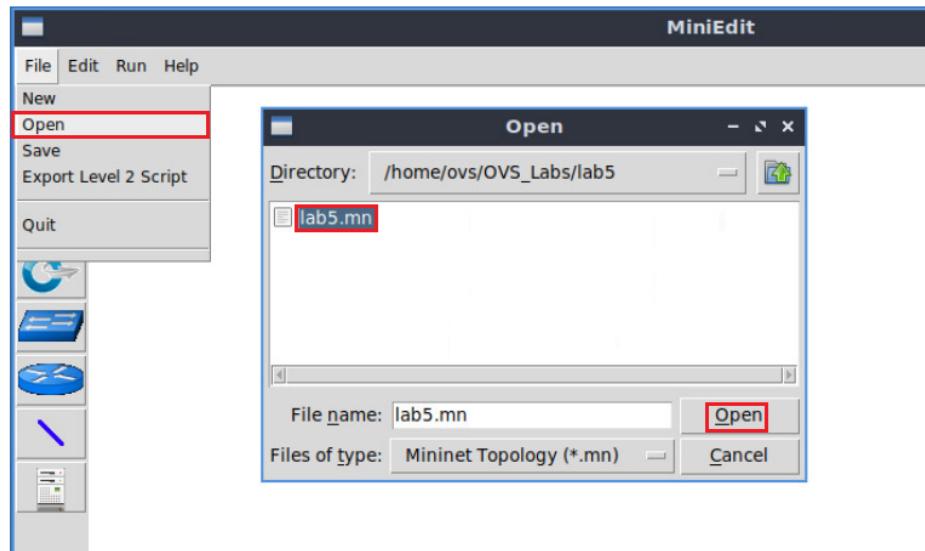


Figure 4. MiniEdit's Open dialog.

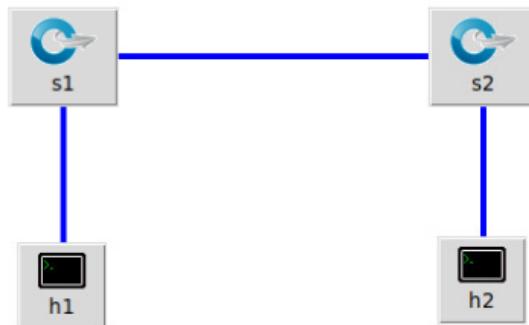


Figure 5. MiniEdit's topology.

**Step 3.** To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 6. Starting the emulation.

**Step 4.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

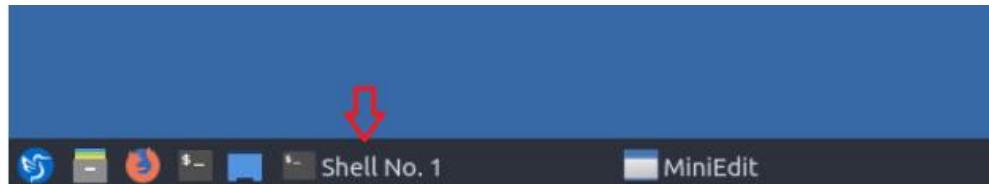


Figure 7. Opening Mininet's terminal.

**Step 5.** Issue the following command to display the interface names and connections.

A screenshot of the 'Shell No. 1' terminal window from Mininet. The window has a dark background and a light gray border. At the top, there is a menu bar with options: File, Actions, Edit, View, Help. Below the menu bar, the title 'Shell No. 1' is displayed. The main area of the window shows a command-line interface. A user has entered the command 'links' and pressed Enter. The terminal then displays the following output:

```
container@net:~$ links
s1-eth1<->h1-eth0 (OK OK)
s2-eth1<->h2-eth0 (OK OK)
s1-eth2<->s2-eth2 (OK OK)
container@net:~$
```

Figure 8. Displaying network interfaces.

In Figure 8, the link displayed within the gray box indicates that interface eth1 of switch s1 connects to interface eth0 of host h1 (i.e.,  $s1\text{-}eth1 <\rightarrow> h1\text{-}eth0$ ).

## 2.3 Loading the configuration file

**Step 1.** Open the Linux terminal.



Figure 9. Opening Linux terminal.

**Step 2.** Navigate into *OVS\_Labs/lab5* directory by issuing the following command. This folder contains a configuration file and the script responsible for loading the configuration. The configuration file will assign the MAC addresses to the hosts' interfaces. The `cd` command is short for change directory followed by an argument that specifies the destination directory.

```
cd OVS_Labs/lab5
```

Figure 10. Entering to the *OVS\_Labs/lab5* directory.

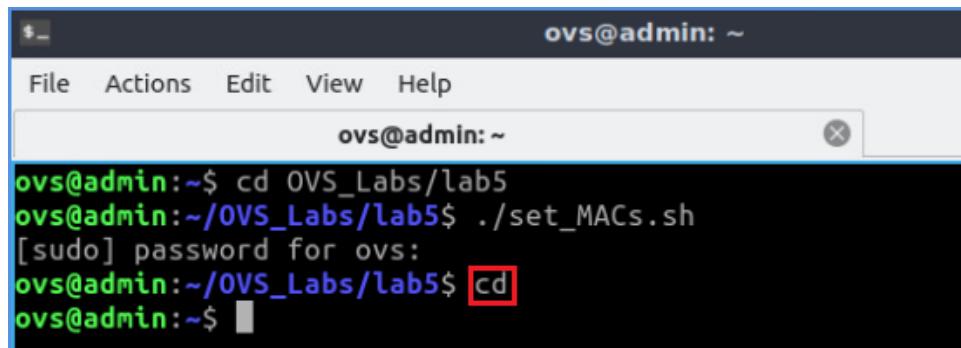
**Step 3.** To execute the shell script, type the following command. The argument of the program corresponds to the configuration zip file that will be loaded in all the hosts and switches in order to set the manual MAC address. When prompted for a password, type `password`.

```
./set_MACs.sh
```

Figure 11. Executing the shell script to load the configuration.

**Step 4.** Type the following command to exit from the lab5 directory.

```
cd
```



The screenshot shows a terminal window titled 'ovs@admin: ~'. The menu bar includes 'File', 'Actions', 'Edit', 'View', and 'Help'. Below the menu is another terminal window titled 'ovs@admin: ~'. The command 'cd' is being typed into this window, with the red box highlighting the 'cd' command. The background of the terminal window shows a command-line interface with 'cd OVS\_Labs/lab5' and '[sudo] password for ovs:'.

Figure 12. Exiting from the directory.

### 3 Verifying IP addresses on the hosts

In this section, you will verify that IP addresses on the hosts are assigned according to table 2.

**Step 1.** Hold right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

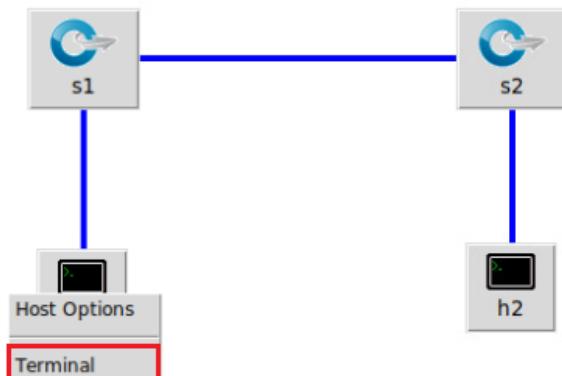
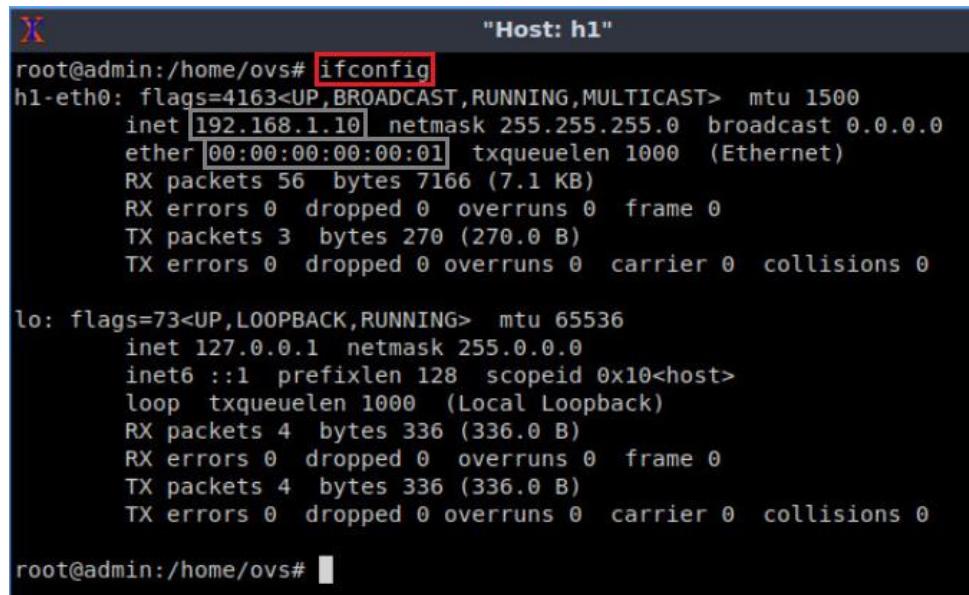


Figure 13. Opening a terminal on host h1.

**Step 2.** In host h1 terminal, type the following command to verify that the IP address was assigned successfully. You will verify the host interface, *h1-eth0* configured with the IP address 192.168.1.10 and the subnet mask 255.255.255.0. You will also verify the MAC address, 00:00:00:00:00:01.

```
ifconfig
```



```
"Host: h1"
root@admin:/home/ovs# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.10 netmask 255.255.255.0 broadcast 0.0.0.0
        ether 00:00:00:00:00:01 txqueuelen 1000 (Ethernet)
        RX packets 56 bytes 7166 (7.1 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 3 bytes 270 (270.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

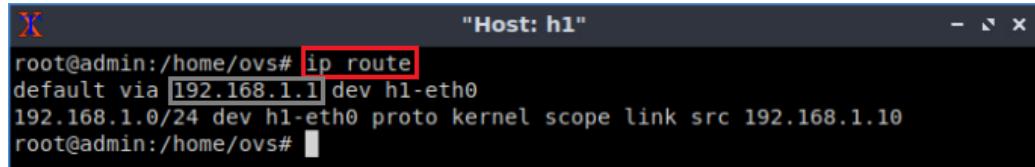
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        RX packets 4 bytes 336 (336.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 4 bytes 336 (336.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@admin:/home/ovs#
```

Figure 14. Verifying IP and MAC addresses on the host.

**Step 3.** On host h1 terminal, type the following command to verify that the default gateway IP address is 192.168.1.1.

```
ip route
```

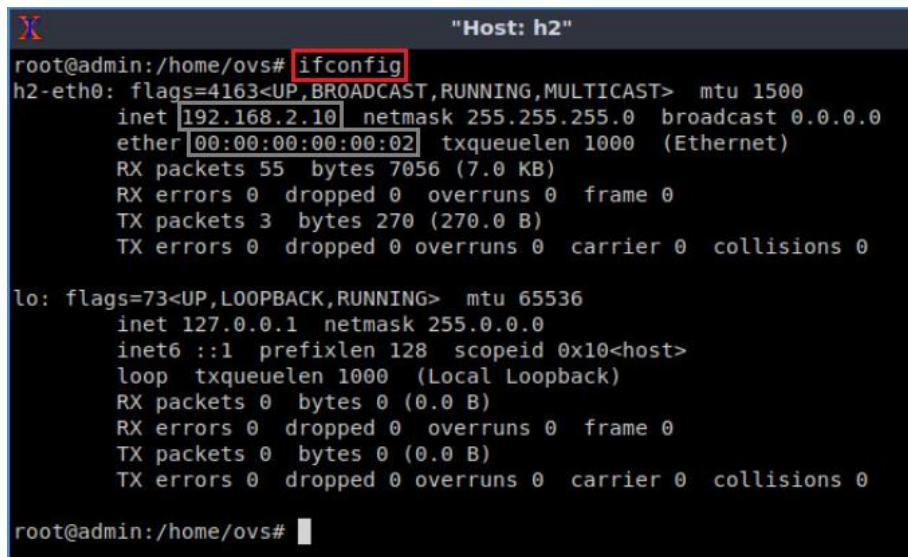


```
"Host: h1"
root@admin:/home/ovs# ip route
default via 192.168.1.1 dev h1-eth0
192.168.1.0/24 dev h1-eth0 proto kernel scope link src 192.168.1.10
root@admin:/home/ovs#
```

Figure 15. Verifying default gateway on the host.

**Step 4.** On host h2 terminal, type the following command to verify that the IP address was assigned successfully. You will verify the host interface, *h2-eth0* configured with the IP address 192.168.2.10 and the subnet mask 255.255.255.0. You will also verify the MAC address, 00:00:00:00:00:02.

```
ifconfig
```



```
"Host: h2"
root@admin:/home/ovs# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.10 netmask 255.255.255.0 broadcast 0.0.0.0
        ether 00:00:00:00:00:02 txqueuelen 1000 (Ethernet)
            RX packets 55 bytes 7056 (7.0 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 3 bytes 270 (270.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

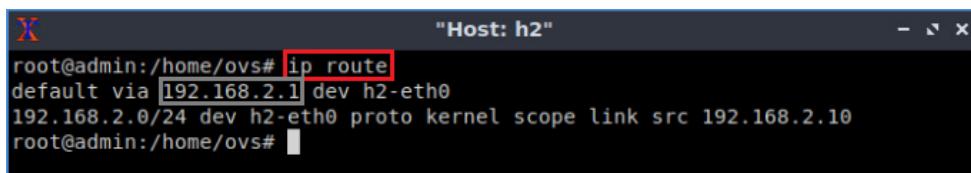
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@admin:/home/ovs#
```

Figure 16. Verifying IP and MAC addresses on the host.

**Step 5.** Type the following command to verify that the default gateway IP address is 192.168.2.1.

```
ip route
```

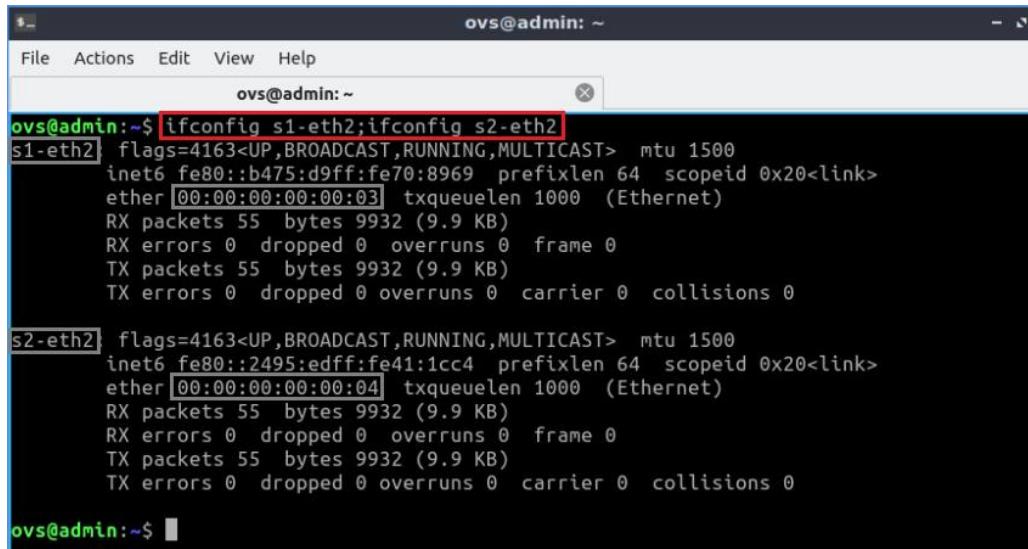


```
"Host: h2"
root@admin:/home/ovs# ip route
default via 192.168.2.1 dev h2-eth0
192.168.2.0/24 dev h2-eth0 proto kernel scope link src 192.168.2.10
root@admin:/home/ovs#
```

Figure 17. Verifying default gateway on the host.

**Step 6.** On the Linux terminal, type the following command to verify the MAC addresses of the switches. You will verify the switch interfaces, s1-eth2 and s2-eth2 are configured with MAC addresses 00:00:00:00:00:03 and 00:00:00:00:00:04, respectively.

```
ifconfig s1-eth2;ifconfig s2-eth2
```



```
ovs@admin: ~
File Actions View Help
ovs@admin: ~
ovs@admin:~$ ifconfig s1-eth2;ifconfig s2-eth2
s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::b475:dfff:fe70:8969 prefixlen 64 scopeid 0x20<link>
        ether 00:00:00:00:00:03 txqueuelen 1000 (Ethernet)
            RX packets 55 bytes 9932 (9.9 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 55 bytes 9932 (9.9 KB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

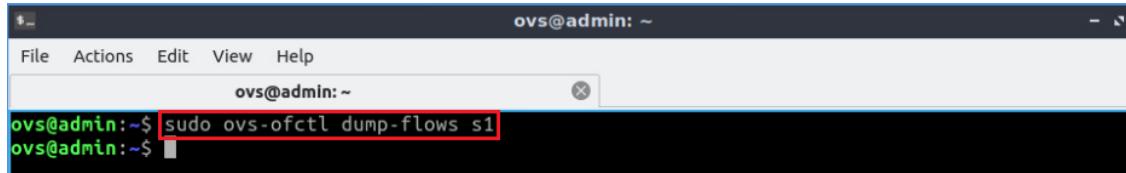
s2-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::2495:edff:fe41:1cc4 prefixlen 64 scopeid 0x20<link>
        ether 00:00:00:00:00:04 txqueuelen 1000 (Ethernet)
            RX packets 55 bytes 9932 (9.9 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 55 bytes 9932 (9.9 KB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ovs@admin:~$
```

Figure 18. Verifying MAC addresses of the switches.

**Step 7.** On the Linux terminal, type the following command to verify the flow installation on switch s1. The fail-mode of the switch is *secure*, and the flow table of the switch is empty at this point.

```
sudo ovs-ofctl dump-flows s1
```

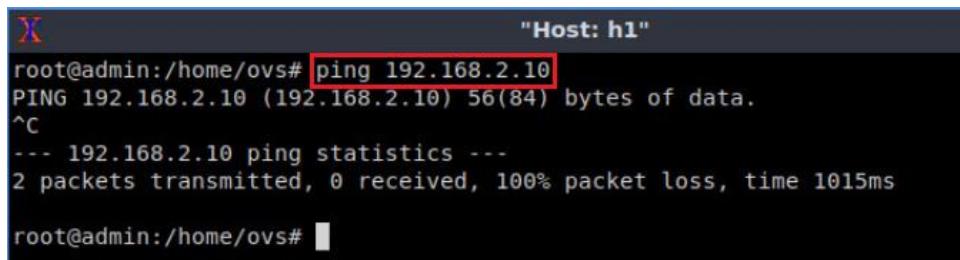


A screenshot of a terminal window titled "ovs@admin: ~". The window has a menu bar with File, Actions, Edit, View, and Help. Below the menu is a toolbar with icons for File, Actions, Edit, View, Help, and a close button. The main area shows a command-line interface with the prompt "ovs@admin:~\$". A red box highlights the command "sudo ovs-ofctl dump-flows s1". The command is followed by its output: "ovs@admin:~\$".

Figure 19. Verifying flow on switch s1.

**Step 8.** Test the connectivity between hosts h1 and host h2 using the `ping` command. There is no connectivity between hosts since switch s1 does not know how to process the traffic. To stop the test, press `Ctrl+c`.

```
ping 192.168.2.10
```

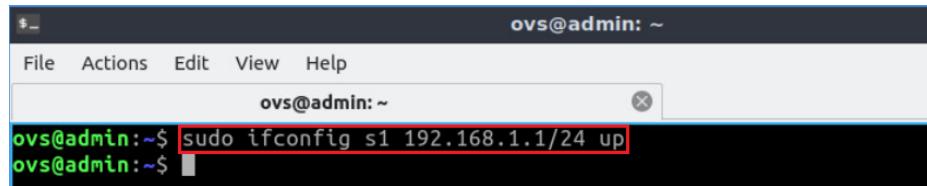


A screenshot of a terminal window titled "'Host: h1'". The window has a title bar with "Host: h1" and a menu bar with File, Actions, Edit, View, Help. Below the menu is a toolbar with icons for File, Actions, Edit, View, Help, and a close button. The main area shows a command-line interface with the prompt "root@admin:/home/ovs#". A red box highlights the command "ping 192.168.2.10". The command is followed by its output:  
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.  
^C  
--- 192.168.2.10 ping statistics ---  
2 packets transmitted, 0 received, 100% packet loss, time 1015ms  
root@admin:/home/ovs#

Figure 20. Output of `ping` command.

**Step 9.** On the Linux terminal, type the following command to add an IP address to the switch, s1. The IP address is the gateway (192.168.1.1) of host h1.

```
sudo ifconfig s1 192.168.1.1/24 up
```

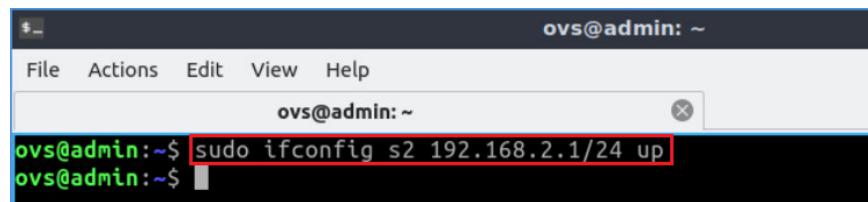


A screenshot of a terminal window titled "ovs@admin: ~". The window has a menu bar with File, Actions, Edit, View, Help. Below the menu is a toolbar with icons for File, Actions, Edit, View, Help, and a close button. The main area shows a command-line interface with the prompt "ovs@admin:~\$". A red box highlights the command "sudo ifconfig s1 192.168.1.1/24 up". The command is followed by its output: "ovs@admin:~\$".

Figure 21. Adding IP address in the switch.

**Step 10.** Type the following command to add an IP address to the switch, s2. The IP address is the gateway (192.168.2.1) of host h2.

```
sudo ifconfig s2 192.168.2.1/24 up
```



```
ovs@admin:~$ sudo ifconfig s2 192.168.2.1/24 up
ovs@admin:~$
```

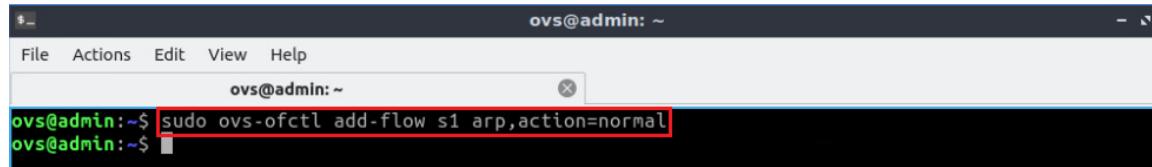
Figure 22. Adding IP address in the switch.

#### 4 Enabling routing in switch s1

In this section, you will configure routing in switch s1. You will configure the switch so that traffic coming from host h1 is sent out to switch s2. Additionally, you will add flows so that switch s1 will be responsible for delivering any data received from switch s2 to the required destination (host h1).

**Step 1.** Type the following command to manually insert an ARP flow entry in the switch s1.

```
sudo ovs-ofctl add-flow s1 arp,action=normal
```



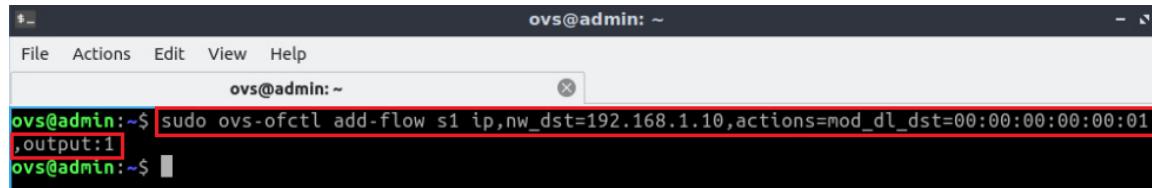
```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 arp,action=normal
ovs@admin:~$
```

Figure 23. Manually adding a flow entry.

Consider the figure above. The flow is for the ARP request. The command adds a flow that sends ARP requests to all the switch ports.

**Step 2.** Type the following command to manually insert a flow entry in switch s1 for the traffic going to the destination host h1 (192.168.1.10).

```
sudo ovs-ofctl add-flow s1
ip,nw_dst=192.168.1.10,actions=mod_dl_dst=00:00:00:00:00:01,output:1
```



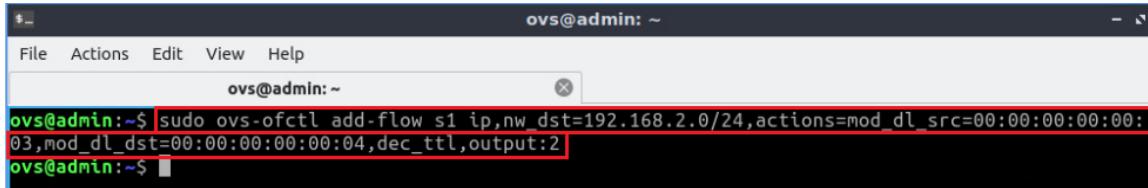
```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 ip,nw_dst=192.168.1.10,actions=mod_dl_dst=00:00:00:00:00:01
, output:1
ovs@admin:~$
```

Figure 24. Manually adding a flow entry.

Consider the figure above. Whenever switch s1 receives any traffic going to the destination host h1 (192.168.1.10), the traffic will be forwarded to port *s1-eth1*. `mod_dl_dst` is responsible for changing the MAC address to the correct MAC address of host h1 (00:00:00:00:00:01).

**Step 3.** Type the following command to manually insert a flow entry in switch s1 for the traffic going to the destination network 192.168.2.0/24.

```
sudo ovs-ofctl add-flow s1  
ip,nw_dst=192.168.2.0/24,actions=mod_dl_src=00:00:00:00:00:03,mod_dl_dst=00:00:  
00:00:00:04,dec_ttl,output:2
```



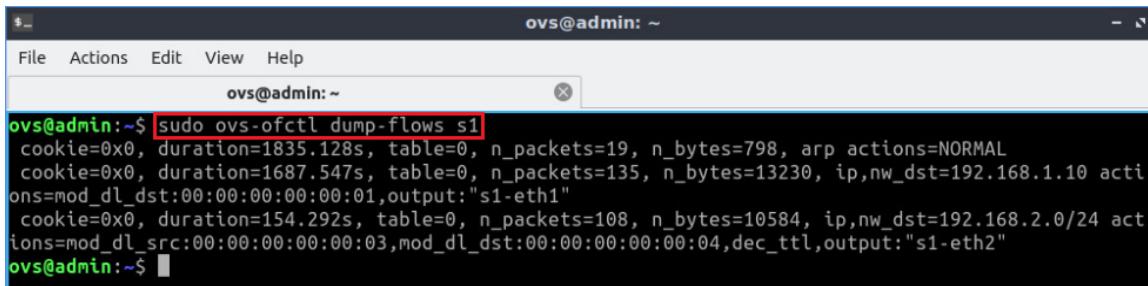
```
File Actions Edit View Help  
ovs@admin: ~  
ovs@admin:~$ sudo ovs-ofctl add-flow s1 ip,nw_dst=192.168.2.0/24,actions=mod_dl_src=00:00:00:00:03,mod_dl_dst=00:00:00:00:04,dec_ttl,output:2  
ovs@admin:~$
```

Figure 25. Manually adding a flow entry.

Consider the figure above. Whenever switch s1 receives traffic going to the destination network 192.168.2.0/24, the traffic will be forwarded to port s1-eth2. The source and destination MAC will be changed to 00:00:00:00:03 (s1-eth2) and 00:00:00:00:04 (s2-eth2). The TTL value will be decreased by one.

**Step 4.** Type the following command to verify the flow installation. This command prints the flow table entries in switch s1.

```
sudo ovs-ofctl dump-flows s1
```



```
File Actions Edit View Help  
ovs@admin: ~  
ovs@admin:~$ sudo ovs-ofctl dump-flows s1  
cookie=0x0, duration=1835.128s, table=0, n_packets=19, n_bytes=798, arp actions=NORMAL  
cookie=0x0, duration=1687.547s, table=0, n_packets=135, n_bytes=13230, ip,nw_dst=192.168.1.10 actions=mod_dl_dst:00:00:00:00:01, output:"s1-eth1"  
cookie=0x0, duration=154.292s, table=0, n_packets=108, n_bytes=10584, ip,nw_dst=192.168.2.0/24 actions=mod_dl_src:00:00:00:00:03,mod_dl_dst:00:00:00:00:04,dec_ttl,output:"s1-eth2"  
ovs@admin:~$
```

Figure 26. Verifying flow table in switch s1.

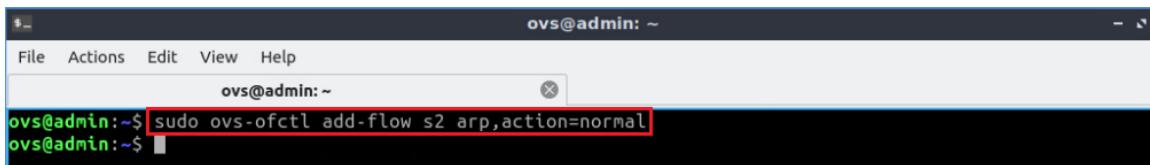
Consider the figure above. You will notice all the manual flows have been installed in the flow table.

## 5 Enabling routing in switch s2

In this section, you will configure routing in switch s2. You will configure the switch so that traffic coming from host h2 is sent out to switch s1. Additionally, you will install flows so that switch s2 will be responsible for delivering any data receiving from switch s1 to the required destination (host h2).

**Step 1.** Type the following command to manually insert an ARP flow entry in switch s2.

```
sudo ovs-ofctl add-flow s2 arp,action=normal
```



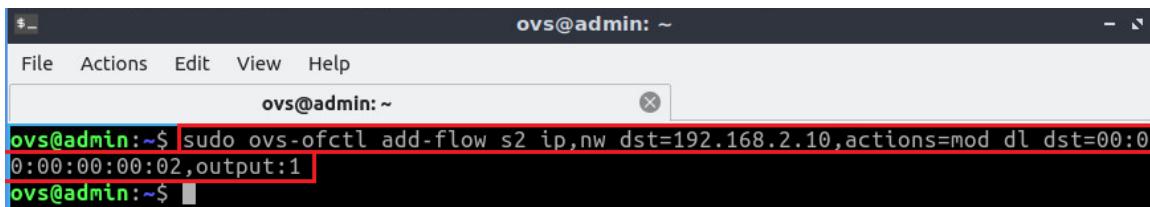
```
ovs@admin:~$ sudo ovs-ofctl add-flow s2 arp,action=normal  
ovs@admin:~$
```

Figure 27. Manually adding a flow entry.

Consider the figure above. The flow is for the ARP request. The command adds a flow that sends ARP requests to all the switch ports.

**Step 2.** Type the following command to manually insert a flow entry in switch s2 for the traffic going to the destination host h2 (192.168.2.10).

```
sudo ovs-ofctl add-flow s2  
ip,nw_dst=192.168.2.10,actions=mod_dl_dst=00:00:00:00:00:02,output:1
```



```
ovs@admin:~$ sudo ovs-ofctl add-flow s2 ip,nw_dst=192.168.2.10,actions=mod_dl_dst=00:00:00:00:00:02,output:1  
ovs@admin:~$
```

Figure 28. Manually adding a flow entry.

Consider the figure above. Whenever switch s2 receives any traffic going to the destination host h2 (192.168.2.10), the traffic will be forwarded to port s2-eth1. `mod_dl_dst` is responsible for changing the MAC address to the correct MAC address of host h2 (00:00:00:00:00:02).

**Step 3.** Type the following command to manually insert a flow entry in switch s2 for the traffic going to the destination network 192.168.1.0/24.

```
sudo ovs-ofctl add-flow s2  
ip,nw_dst=192.168.1.0/24,actions=mod_dl_src=00:00:00:00:00:04,mod_dl_dst=00:00:00:00:00:03,dec_ttl,output:2
```



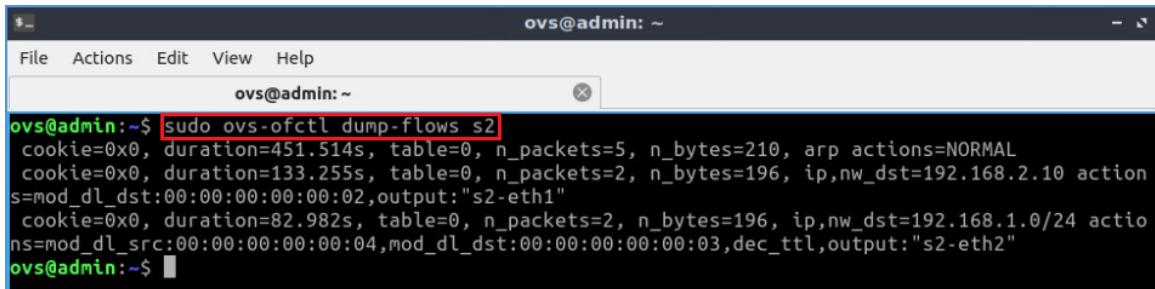
```
ovs@admin:~$ sudo ovs-ofctl add-flow s2 ip,nw_dst=192.168.1.0/24,actions=mod_dl_src=00:00:00:00:00:04,mod_dl_dst=00:00:00:00:00:03,dec_ttl,output:2  
ovs@admin:~$
```

Figure 29. Manually adding a flow entry.

Consider the figure above. Whenever switch s2 receives traffic going to the destination network 192.168.1.0/24, the traffic will be forwarded to port s2-eth2. The source and destination MAC will be changed to 00:00:00:00:00:04 (s2-eth2) and 00:00:00:00:00:03 (s1-eth2). The TTL value will be decreased by one.

**Step 4.** Type the following command to verify the flow installation. This command prints the OpenFlow table entries in switch s2.

```
sudo ovs-ofctl dump-flows s2
```



```
ovs@admin:~$ sudo ovs-ofctl dump-flows s2
cookie=0x0, duration=451.514s, table=0, n_packets=5, n_bytes=210, arp actions=NORMAL
cookie=0x0, duration=133.255s, table=0, n_packets=2, n_bytes=196, ip,nw_dst=192.168.2.10 actions=mod_dl_dst:00:00:00:00:02, output:"s2-eth1"
cookie=0x0, duration=82.982s, table=0, n_packets=2, n_bytes=196, ip,nw_dst=192.168.1.0/24 actions=mod_dl_src:00:00:00:00:00:04,mod_dl_dst:00:00:00:00:03,dec_ttl,output:"s2-eth2"
ovs@admin:~$
```

Figure 30. Verifying flow table in switch s2.

Consider the figure above. You will notice all the manual flows have been installed in the flow table.

## 6 Verifying configuration

**Step 1.** On the Linux terminal, start the Wireshark packet analyzer by issuing the following command. A new window will emerge.

```
sudo wireshark
```

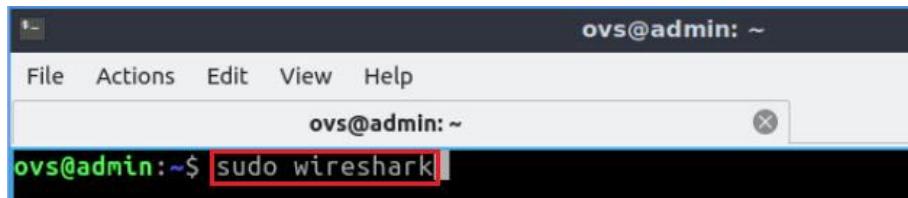


Figure 31. Starting Wireshark packet analyzer.

**Step 2.** Click on interface *s1-eth1* then, click on the icon located on the upper left-hand side to start capturing packets on this interface.

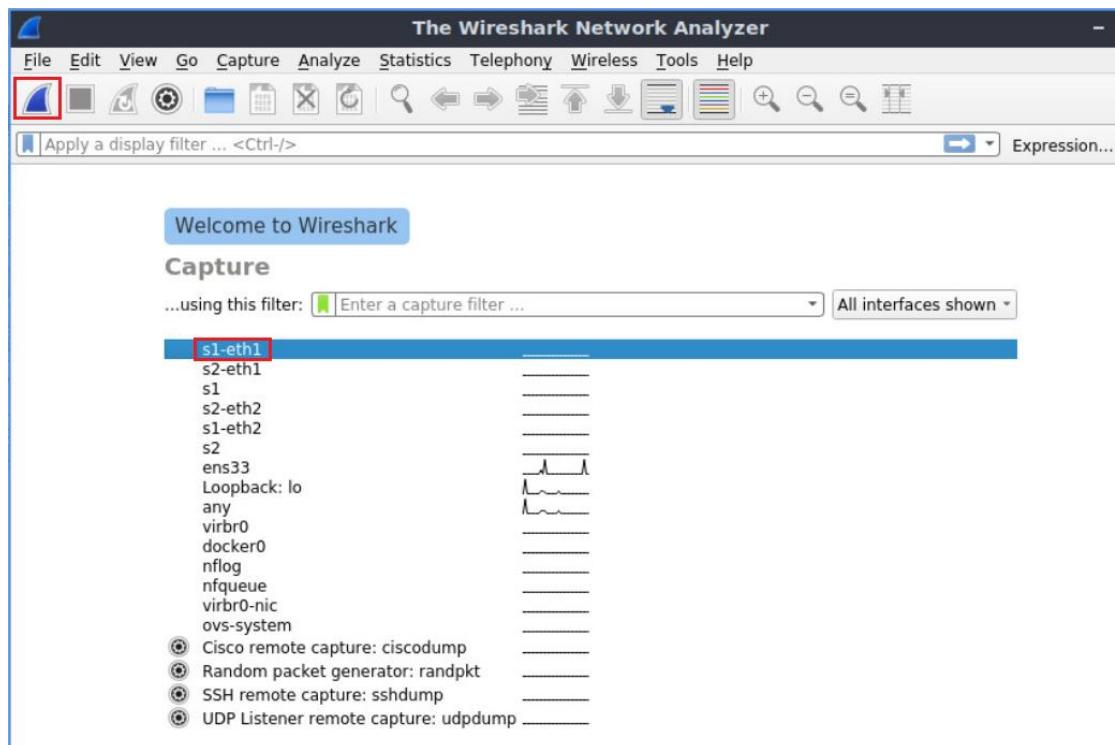


Figure 32. Starting packet capture.

**Step 3.** On the Linux terminal, go to the file option and open a new terminal tab (showed in the following figure) or press Ctrl+Shift+T.

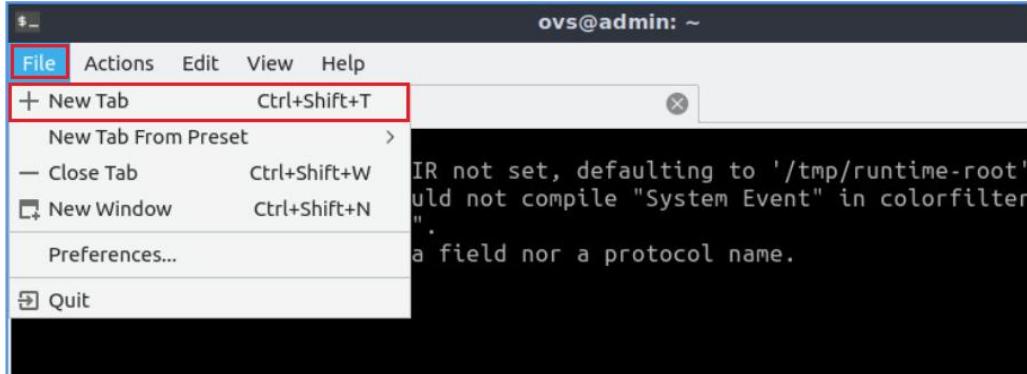


Figure 33. Opening a new terminal.

**Step 4.** On the Linux terminal, start the Wireshark packet analyzer again by issuing the following command. A new window will emerge. When prompted for a password, type password.

```
sudo wireshark
```

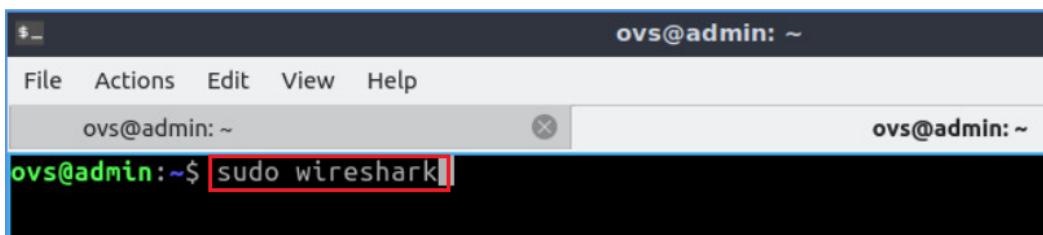


Figure 34. Starting Wireshark packet analyzer.

**Step 5.** Click on interface *s1-eth2* then, click on the icon located on the upper left-hand side to start capturing packets on this interface.

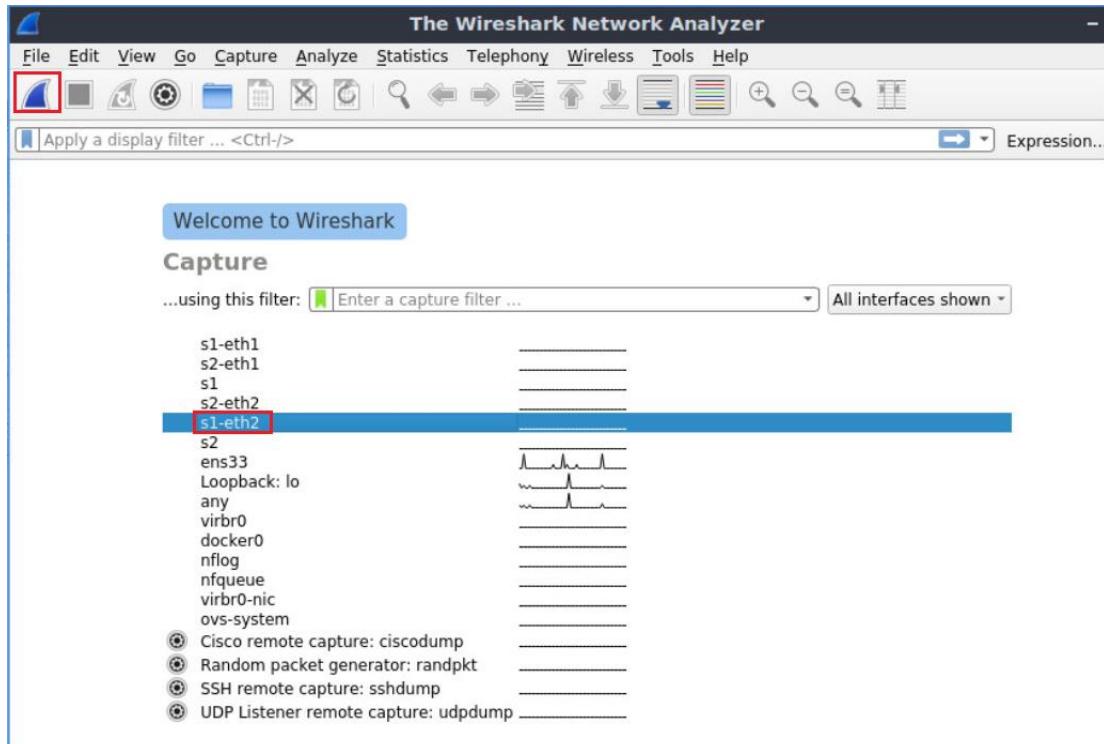


Figure 35. Starting packet capture.

**Step 6.** Test the connectivity between host h1 and host h2 using the `ping` command. To stop the test, press `Ctrl+c`.

```
ping 192.168.2.10
```

```
"Host: h1"
root@admin:/home/ovs# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
64 bytes from 192.168.2.10: icmp_seq=1 ttl=64 time=1.09 ms
64 bytes from 192.168.2.10: icmp_seq=2 ttl=64 time=0.062 ms
64 bytes from 192.168.2.10: icmp_seq=3 ttl=64 time=0.068 ms
^C
--- 192.168.2.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2017ms
rtt min/avg/max/mdev = 0.062/0.407/1.091/0.483 ms
root@admin:/home/ovs#
```

Figure 36. Output of `ping` command.

The figure shows a successful connectivity test.

**Step 7.** Verify packet capturing on interface *s1-eth1*. Click on any ICMP packet having source IP 192.168.1.10. Click on the arrow located on the leftmost side of the field called *Ethernet II*. A list will be displayed.

## Lab 5: Implementing Routing in Open vSwitch

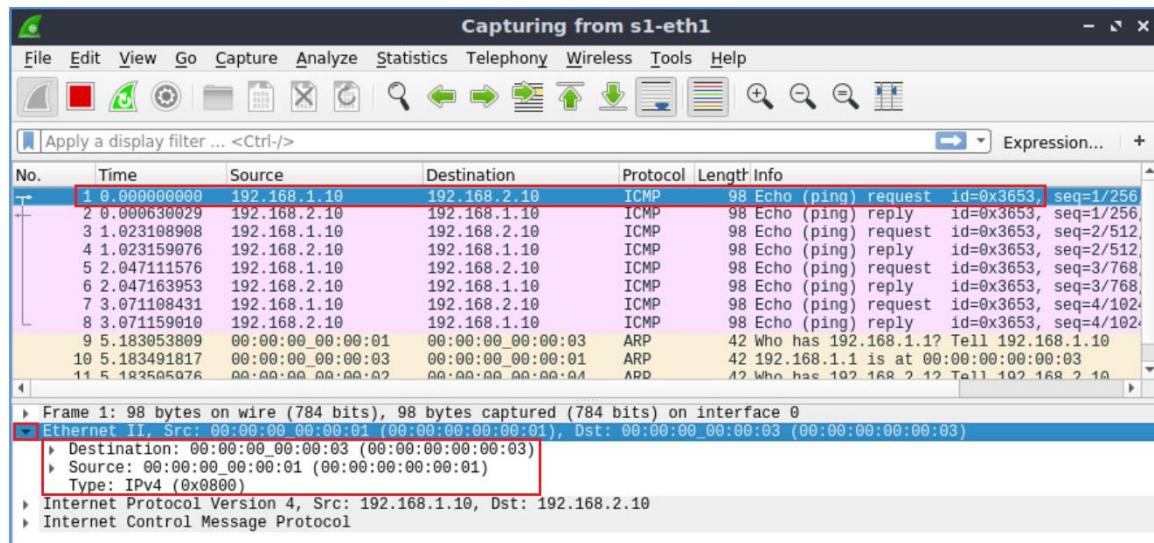


Figure 37. Verifying MAC addresses.

Consider the figure above. Whenever traffic is forwarding from host h1 to the interface *s1-eth1*, the source MAC address is 00:00:00:00:00:01 (host h1) and the destination MAC address is 00:00:00:00:00:03 (*s1-eth2*).

**Step 8.** Verify packet capturing on interface *s1-eth1*. Click on the arrow located on the leftmost side of the field called *Internet Protocol Version 4*. A list will be displayed.

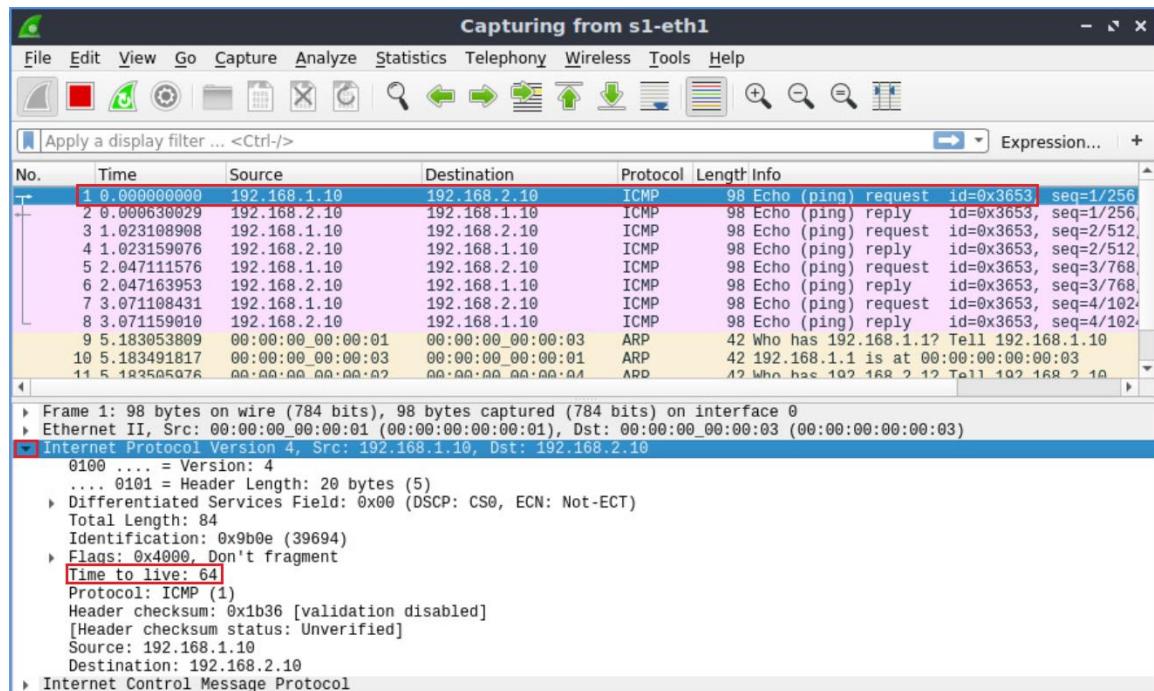


Figure 38. Verifying TTL value.

Consider the figure above. You will notice that the Time To Live (TTL) value is 64.

**Step 9.** Verify packet capturing on interface *s1-eth2*. Click on any ICMP packet having source IP 192.168.1.10. Click on the arrow located on the leftmost side of the field called *Ethernet II*. A list will be displayed.

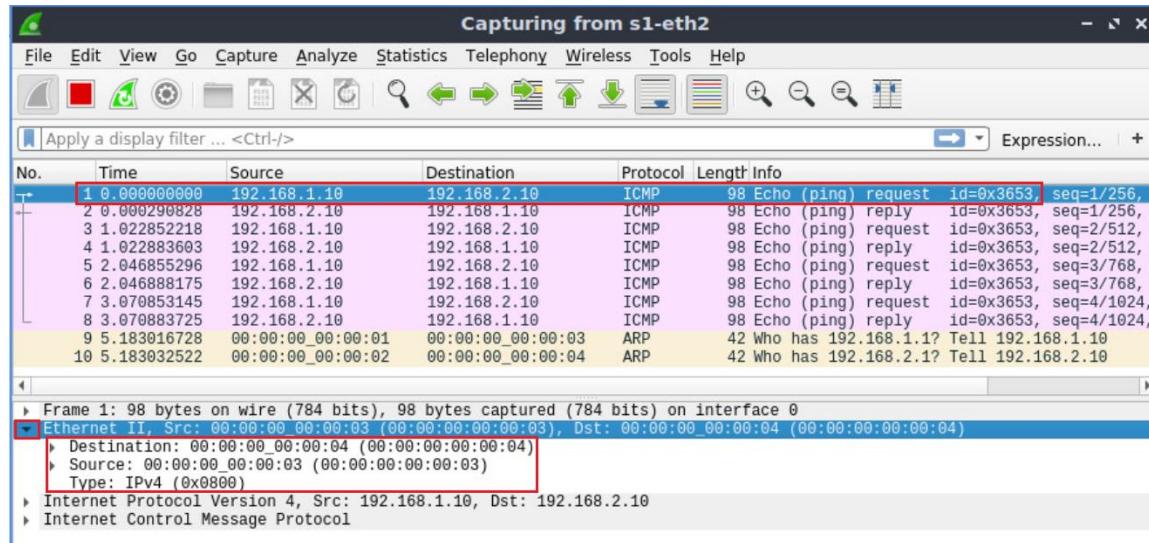


Figure 39. Verifying MAC addresses.

Consider the figure above. Whenever the traffic is forwarded from interface *s1-eth2* to the interface *s2-eth2*, the source MAC address is 00:00:00:00:00:03 (*s1-eth2*) and the destination MAC address is 00:00:00:00:00:04 (*s2-eth2*).

**Step 10.** Verify packet capturing on interface *s1-eth2*. Click on the arrow located on the leftmost side of the field called *Internet Protocol Version 4*. A list will be displayed.

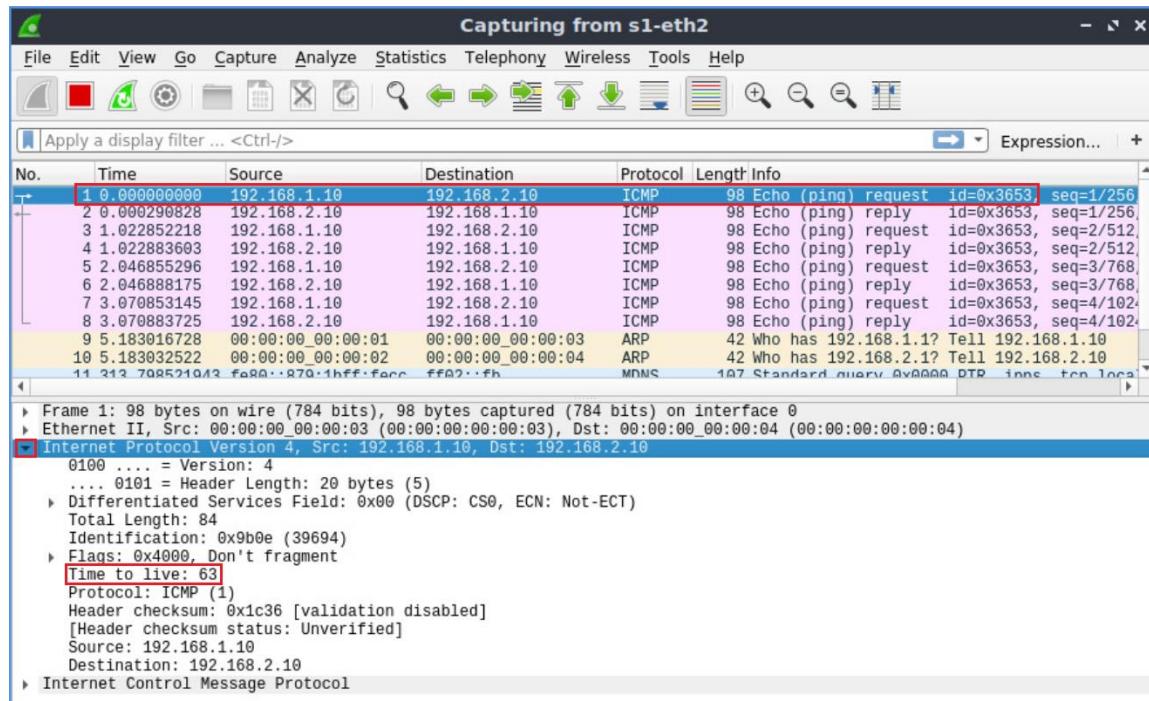


Figure 40. Verifying TTL value.

Consider the figure above. You will notice the TTL value has been decreased by one (63).

**Step 11.** Now you will use another tool provided by Open vSwitch to trace the packet as it traverses the switch. `ovs-appctl` is the tracing tool that can be used to determine what is happening with packets as they go through the data plane processing (e.g., modified fields, output port, etc.). Type the command below to issue a packet with the following fields' values:

- Input port: 1
- Network layer protocol: IP
- Source IP address: 192.168.1.10
- Destination IP address: 192.168.2.10
- Source MAC address: 00:00:00:00:00:01
- Time-to-live (TTL): 64

```
sudo ovs-appctl ofproto/trace s1
in_port=1,ip,nw_src=192.168.1.10,nw_dst=192.168.2.10,dl_src=00:00:00:00:00:01,
nw_ttl=64
```

```
ovs@admin:~$ sudo ovs-appctl ofproto/trace s1 in_port=1,ip,nw_src=192.168.1.10,nw_dst=192.168.2.10,dl_src=00:00:00:00:00:01,nw_ttl=64
Flow: ip,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:00,nw_src=192.168.1.10,nw_dst=192.168.2.10,
nw_proto=0,nw_tos=0,nw_ecn=0,nw_ttl=64
bridge("s1")
-----
0. ip,nw_dst=192.168.2.0/24, priority 32768
    mod_dl_src:00:00:00:00:00:03
    mod_dl_dst:00:00:00:00:00:04
    dec_ttl
    output:2
Final flow: ip,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04,nw_src=192.168.1.10,nw_dst=192.168.2.0/24,nw_ttl=63
Megaflow: recirc_id=0,eth,ip,in_port=1,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:00,nw_dst=192.168.2.0/24,nw_ttl=64,nw_f
rag=no
Datapath actions: set(eth(src=00:00:00:00:00:03,dst=00:00:00:00:00:04)),1
ovs@admin:~$
```

Figure 41. Tracing the packet processing.

Consider the figure above. The first gray box shows the flow fields' values before the flow is sent to the switch. Note that if a field is not specified (e.g., dl\_src, or the source MAC address), the tool uses the value 0 for that field. The values here match those specified in the command by the user. The second gray box shows the rule that was matched in the flow table. The third gray box shows the final flow fields' values after the packet was processed in the switch. The values here match those that were shown in the Wireshark capture of the previous steps.

This concludes Lab 5. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. Router Freak, “*Understanding Network Routing Protocols*”, April 2010, [online]. Available: <https://www.routerfreak.com/network-fundamentals/page/4/>
2. Linux Foundation, “*Open vSwitch*”, [Online]. Available: <http://openvSwitch.org>.

3. B.Pfaff, B. Davie, Ed, “*The Open vSwitch Database Management Protocol*”, RFC 7047, Dec 2013.
4. IBM, “*Archived / Virtual networking in Linux*”, [Online]. Available: <https://developer.ibm.com/tutorials/l-virtual-networking/>
5. Mininet walkthrough, [Online]. Available: <http://mininet.org>.
6. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. “*OpenFlow: enabling innovation in campus networks.*” ACM SIGCOMM Computer Communication Review 38, no.2 (2008):69-74.
7. PicOS Documentation, [Online]. Available: <https://docs.pica8.com/pages/viewpage.action?pageId=3083175>
8. Cisco, “*IP Addressing: ARP Configuration Guide*”, [Online]. Available: [https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr\\_arp/configuration/15-s/arp-15-s-book/Configuring-Address-Resolution-Protocol.html](https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_arp/configuration/15-s/arp-15-s-book/Configuring-Address-Resolution-Protocol.html)
9. Grandmetric, “*How does a switch work?*”, [Online]. Available: <https://www.grandmetric.com/2018/03/08/how-does-switch-work-2/>
10. Juniper Networks, “*Layer 2 Networking*”, [Online]. Available: [https://www.juniper.net/documentation/en\\_US/junos/topics/topic-map/layer-2-understanding.html](https://www.juniper.net/documentation/en_US/junos/topics/topic-map/layer-2-understanding.html)
11. Open Networking Foundation, “*OpenFlow Switch Specification*”, [Online]. Available: <https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf>



## OPEN VIRTUAL SWITCH

### Lab 6: Implementing Routing using multiple Flow Tables

Document Version: **08-17-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    Open vSwitch Routing .....	4
1.2    OpenFlow pipeline processing .....	4
2    Lab topology.....	5
2.1    Lab settings.....	5
2.2    Loading a topology .....	5
2.3    Loading the configuration file .....	7
3    Verifying IP addresses on the hosts .....	9
4    Table 0 - Classifier .....	13
5    Table 1 – Layer 3 Forwarding.....	15
6    Table 2 – Layer 2 Forwarding.....	16
7    Verifying configuration .....	18
References .....	20

## Overview

This lab aims to demonstrate how to manually configure multiple flow tables to enable packet forwarding between different networks.

## Objectives

By the end of this lab, the student should be able to:

1. Understand OpenFlow pipeline processing.
2. Understand how Open vSwitch implements routing between two different networks.
3. Configure multiple flow tables to enable packet forwarding between different networks.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Verifying IP addresses on the hosts.
4. Section 4: Table 0 – Classifier.
5. Section 5: Table 1 – Layer 3 Forwarding.
6. Section 6: Table 2 – Layer 2 Forwarding.
7. Section 7: Verifying configuration.

### 1      **Introduction**

OpenFlow has been widely used for the low-cost configuration and optimization of traffic flows in Data-center and Campus networks since it decouples the control and data forwarding plane. The packets of traffic flow are processed against the flow entry in the flow table where a flow entry is identified by the match fields and priority. The structure

of match fields is becoming more complex in the new Internet architecture, new application type, and new media format. Thus, the single flow table of OpenFlow implementation can lead to fast storage space growth, and finally cause table-overflow. Multiple flow tables can address this problem to improve network performance<sup>5</sup>.

## 1.1 Open vSwitch Routing

Open vSwitch relies on OpenFlow protocol to implement routing. OpenFlow switches perform packet forwarding using the packet-matching function within the flow table. Thus, once a packet arrives at the switch, the latter will look up in its flow table and check if there is a match. Consequently, the switch will decide which action to take based on the flow table<sup>4</sup>. The action could be:

- Forward the packet out to another port.
- Drop the packet.
- Pass the packet to the controller.

The flows can be installed manually within the switch if there is no controller connected to it. The flows are installed in the Open vSwitch daemon (Open vSwitch-vSwitchd) that controls the switch and implements the OpenFlow protocol. `ovs-ofctl` command-line tool is required for monitoring and administering switches that support OpenFlow protocol.

## 1.2 OpenFlow pipeline processing

An Open vSwitch may contain more than one flow table which is referred to as OpenFlow pipeline. Each flow table contains a set of flow entries that consist of match fields, counters, and a set of instructions. The flow tables of an OpenFlow switch are sequentially numbered, starting at 0. The packet is first matched with flow entries of the first flow table, which is flow table 0. A flow entry can only direct a packet to a flow table number that is greater than its own flow table number. Packets match against the packet header fields such as switch input port, Virtual Local Area Network (VLAN) ID, Ethernet source/destination addresses, IP source/destination addresses, IP protocol, source/destination ports. When a packet matches a flow entry, the OpenFlow switch updates the action set for the packet and passes the packet to the next flow table. When pipeline processing stops, the packet is processed with its associated action set and usually forwarded<sup>4</sup>.

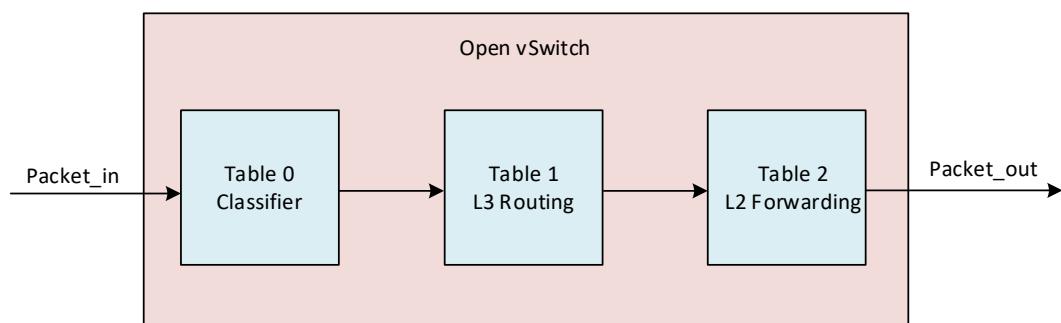


Figure 1. OpenFlow forwarding workflow.

Figure 1 shows OpenFlow pipeline processing. For an incoming packet, matching starts in table 0 and check all the entries in the flow table. Each flow entry contains a set of instructions that are executed when a packet matches an entry. These instructions result in changes to the packet, action set and/or pipeline processing. Table 0 is referred to as classifier. For any IP packet, the switch is instructed to check the next table (table 1). In table 1, routing will be placed based on the destination IP address. The source and destination MACs are modified and Time-To-Live (TTL) is decreased. In table 2, Layer 2 lookup will be placed, and the packet will be forwarded out to the correct port.

## 2 Lab topology

Consider Figure 2. The topology consists of two end-hosts and two switches. Hosts h1 and h2 belong to the networks 192.168.1.0/24 and 192.168.2.0/24, respectively.

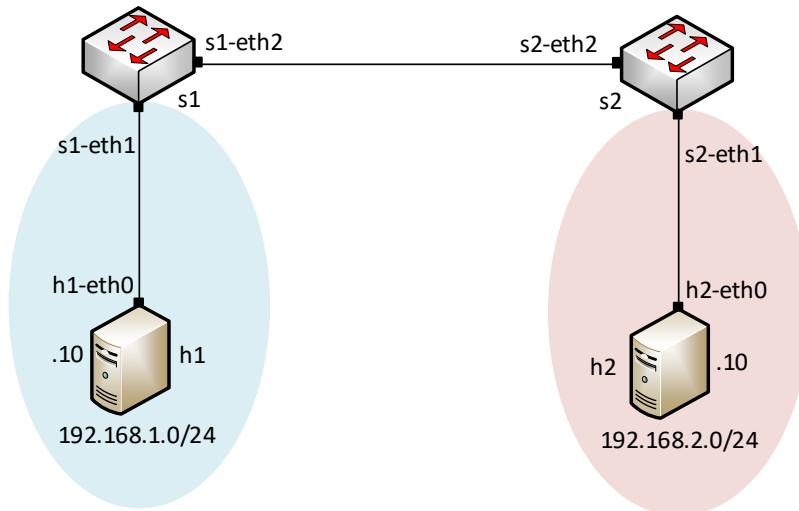


Figure 2. Lab topology.

### 2.1 Lab settings

The devices are already configured according to Table 2.

Table 2. Topology information.

Device	Interface	IP Address	Subnet	Default gateway
h1	h1-eth0	192.168.1.10	/24	192.168.1.1
h2	h2-eth0	192.168.2.10	/24	192.168.2.1

### 2.2 Loading a topology

**Step 1.** Start by launching MiniEdit by clicking on desktop's shortcut. When prompted for a password, type `password`.

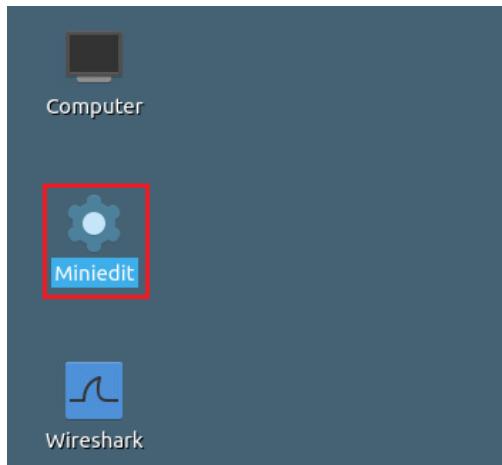


Figure 3. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then *open* to load the lab's topology. Locate the *Lab6.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab6* and click on *Open*.

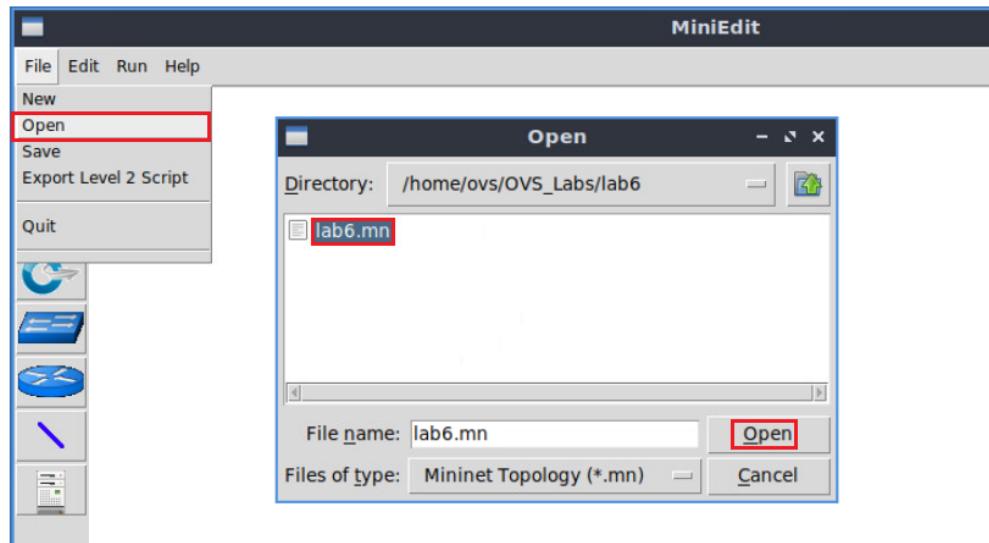


Figure 4. MiniEdit's Open dialog.

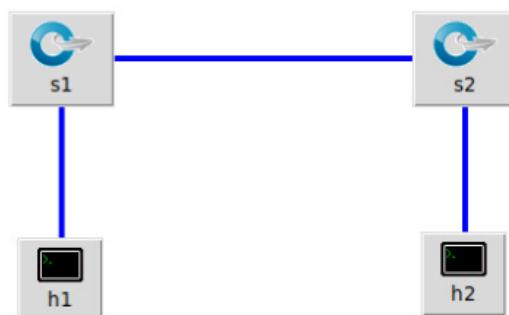


Figure 5. MiniEdit's topology.

**Step 3.** To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 6. Starting the emulation.

**Step 4.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

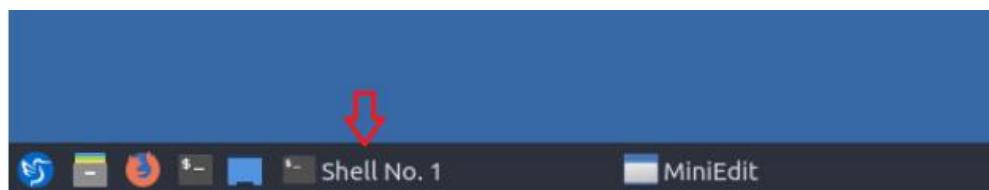


Figure 7. Opening Mininet's terminal.

**Step 5.** Issue the following command to display the interface names and connections.

A screenshot of the 'Shell No. 1' terminal window. The user has entered the command 'links' and pressed Enter. The terminal displays the following output:

```
containernet> links
s1-eth1<->h1-eth0 (OK OK)
s2-eth1<->h2-eth0 (OK OK)
s1-eth2<->s2-eth2 (OK OK)
containernet>
```

The first line of the output, 's1-eth1<->h1-eth0 (OK OK)', is highlighted with a red box.

Figure 8. Displaying network interfaces.

In Figure 8, the link displayed within the gray box indicates that interface eth1 of switch s1 connects to interface eth0 of host h1 (i.e.,  $s1\text{-}eth1 \leftrightarrow h1\text{-}eth0$ ).

## 2.3 Loading the configuration file

**Step 1.** Open the Linux terminal.



Figure 9. Opening Linux terminal.

**Step 2.** Click on the Linux terminal and navigate into *OVS\_Labs/lab6* directory by issuing the following command. This folder contains a configuration file and the script responsible for loading the configuration. The configuration file will assign the MAC addresses to the hosts' interfaces. The `cd` command is short for change directory followed by an argument that specifies the destination directory.

```
cd OVS_Labs/lab6
```

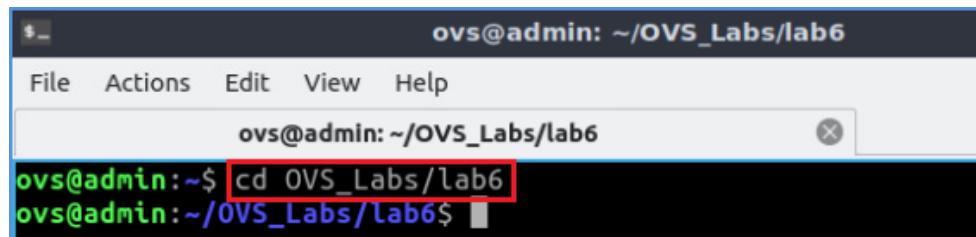


Figure 10. Entering to the *OVS\_Labs/lab6* directory.

**Step 3.** To execute the shell script, type the following command. The argument of the program corresponds to the configuration zip file that will be loaded in all the hosts and switches in order to set the manual MAC address. When prompted for a password, type `password`.

```
./set_MACs.sh
```

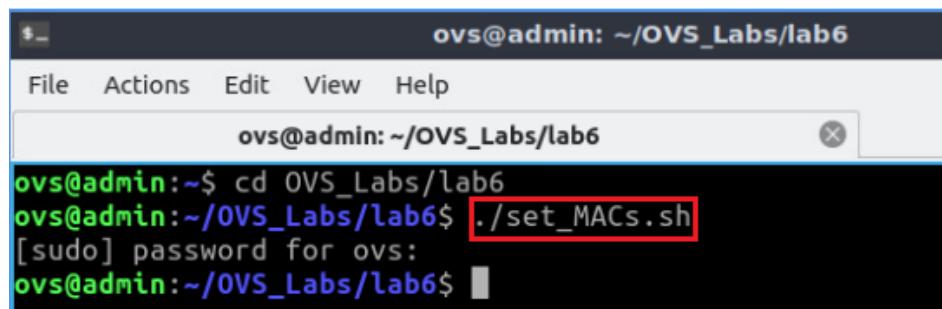
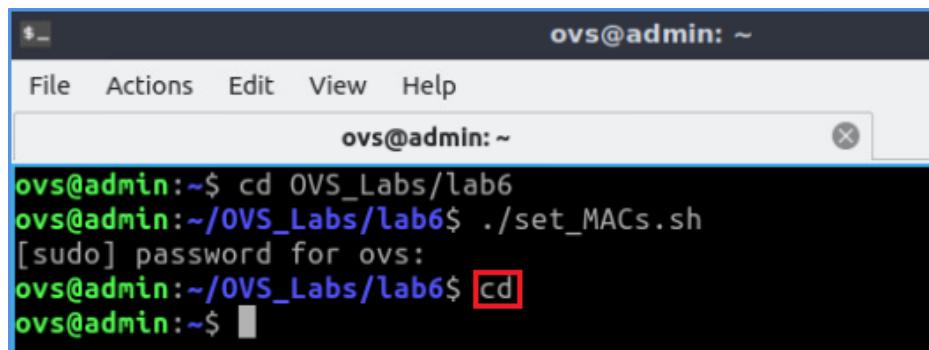


Figure 11. Executing the shell script to load the configuration.

**Step 4.** Type the following command to exit from the lab6 directory.

```
cd
```



```
ovs@admin:~$ cd OVS_Labs/lab6
ovs@admin:~/OVS_Labs/lab6$ ./set_MACs.sh
[sudo] password for ovs:
ovs@admin:~/OVS_Labs/lab6$ cd
ovs@admin:~$
```

Figure 12. Exiting from the directory.

### 3 Verifying IP addresses on the hosts

In this section, you will verify that IP addresses on the hosts are assigned according to table 2.

**Step 1.** Hold right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

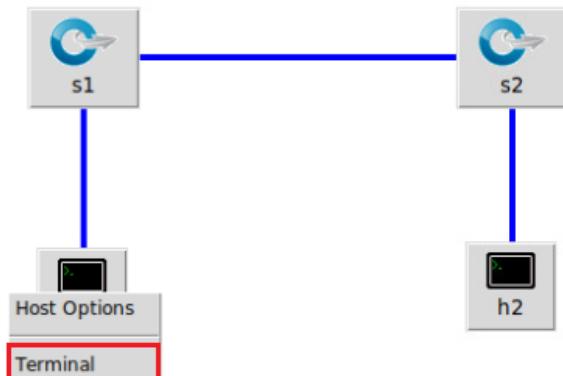
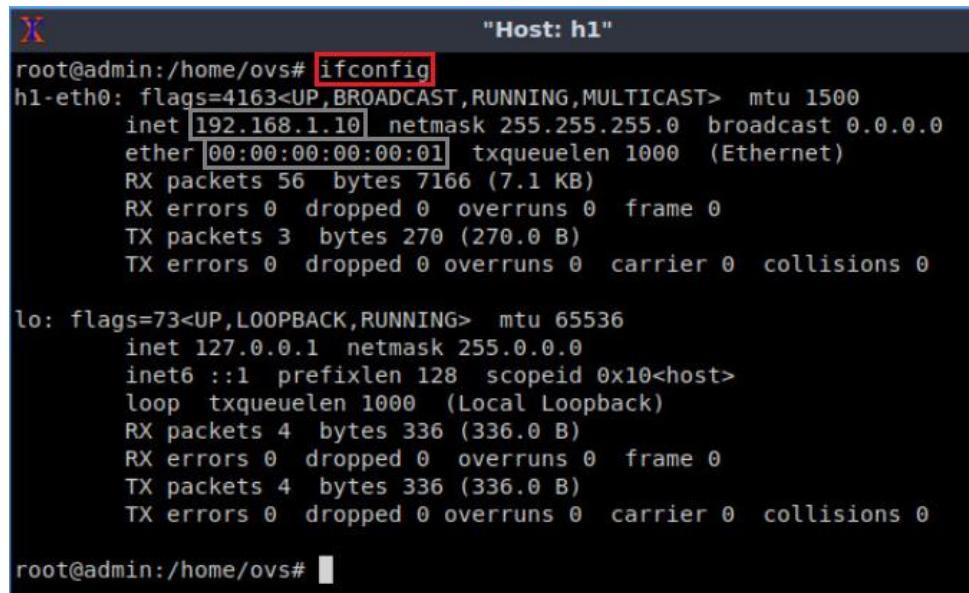


Figure 13. Opening a terminal on host h1.

**Step 2.** In host h1 terminal, type the following command to verify that the IP address was assigned successfully. You will verify the host interface, *h1-eth0* configured with the IP address 192.168.1.10 and the subnet mask 255.255.255.0. You will also verify the MAC address, 00:00:00:00:00:01.

```
ifconfig
```



```
"Host: h1"
root@admin:/home/ovs# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.10 netmask 255.255.255.0 broadcast 0.0.0.0
        ether 00:00:00:00:00:01 txqueuelen 1000 (Ethernet)
        RX packets 56 bytes 7166 (7.1 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 3 bytes 270 (270.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

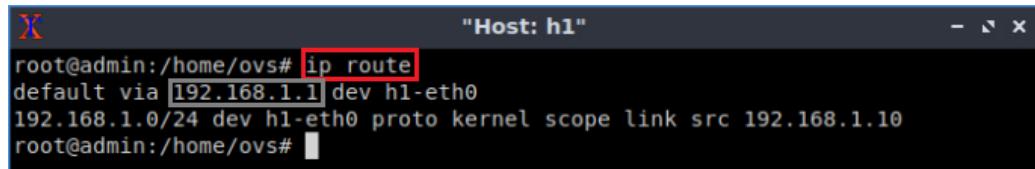
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        RX packets 4 bytes 336 (336.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 4 bytes 336 (336.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@admin:/home/ovs#
```

Figure 14. Verifying IP and MAC addresses on the host.

**Step 3.** On host h1 terminal, type the following command to verify that the default gateway IP address is 192.168.1.1.

```
ip route
```

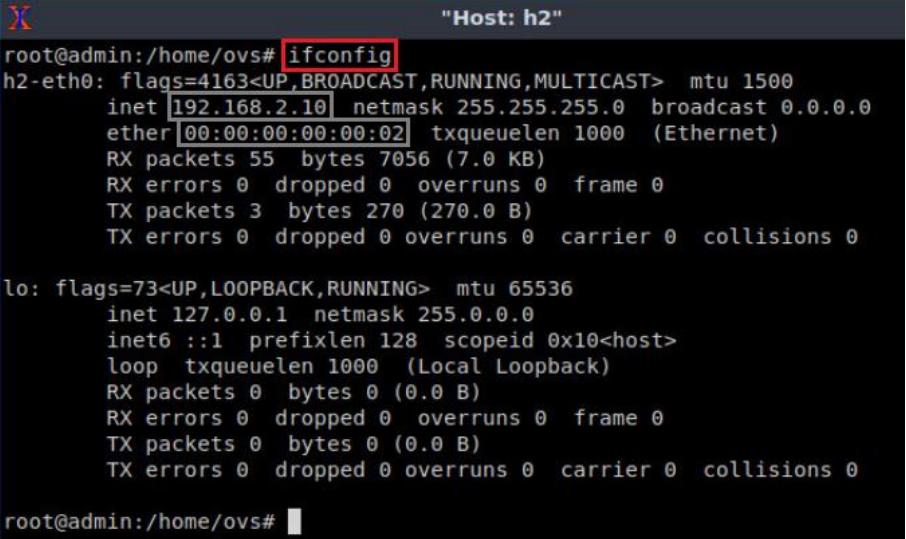


```
"Host: h1"
root@admin:/home/ovs# ip route
default via 192.168.1.1 dev h1-eth0
192.168.1.0/24 dev h1-eth0 proto kernel scope link src 192.168.1.10
root@admin:/home/ovs#
```

Figure 15. Verifying default gateway on the host.

**Step 4.** On host h2 terminal, type the following command to verify that the IP address was assigned successfully. You will verify the host interface, *h2-eth0* configured with the IP address 192.168.2.10 and the subnet mask 255.255.255.0. You will also verify the MAC address, 00:00:00:00:00:02.

```
ifconfig
```



```
"Host: h2"
root@admin:/home/ovs# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.10 netmask 255.255.255.0 broadcast 0.0.0.0
        ether 00:00:00:00:00:02 txqueuelen 1000 (Ethernet)
            RX packets 55 bytes 7056 (7.0 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 3 bytes 270 (270.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

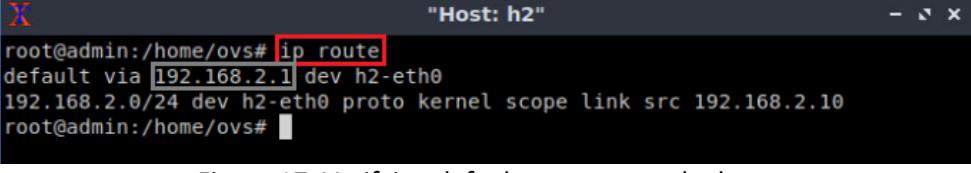
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@admin:/home/ovs#
```

Figure 16. Verifying IP and MAC addresses on the host.

**Step 5.** Type the following command to verify that the default gateway IP address is 192.168.2.1.

```
ip route
```



```
"Host: h2"
root@admin:/home/ovs# ip route
default via 192.168.2.1 dev h2-eth0
192.168.2.0/24 dev h2-eth0 proto kernel scope link src 192.168.2.10
root@admin:/home/ovs#
```

Figure 17. Verifying default gateway on the host.

**Step 6.** On the Linux terminal, type the following command to verify the MAC addresses of the switches. You will verify the switch interfaces, s1-eth1, s2-eth1, s1-eth2 and s2-eth2 are configured with MAC addresses 00:00:00:00:00:03, 00:00:00:00:00:04, 00:00:00:00:00:05 and 00:00:00:00:00:06, respectively.

```
ifconfig
```

```

ovs@admin:~$ ifconfig
s1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet6 fe80::8a5:30ff:fea5:e567 prefixlen 64 scopeid 0x20<link>
        ether 00:00:00:00:00:03 txqueuelen 1000 (Ethernet)
          RX packets 3 bytes 270 (270.0 B)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 22 bytes 2999 (2.9 KB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s1-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet6 fe80::3cfa:a8ff:fea4:bf39 prefixlen 64 scopeid 0x20<link>
        ether 00:00:00:00:00:05 txqueuelen 1000 (Ethernet)
          RX packets 22 bytes 2999 (2.9 KB)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 22 bytes 2999 (2.9 KB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

s2-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet6 fe80::2cf1:e7ff:fe78:f59e prefixlen 64 scopeid 0x20<link>
        ether 00:00:00:00:00:04 txqueuelen 1000 (Ethernet)
          RX packets 3 bytes 270 (270.0 B)
          RX errors 0 dropped 0 overruns 0 frame 0

s2-eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet6 fe80::ac03:1fff:fe0a:455c prefixlen 64 scopeid 0x20<link>
        ether 00:00:00:00:00:06 txqueuelen 1000 (Ethernet)
          RX packets 22 bytes 2999 (2.9 KB)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 22 bytes 2999 (2.9 KB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Figure 18. Verifying MAC addresses of the switches.

**Step 7.** On the Linux terminal, type the following command to verify the flow installation on switch s1. The fail-mode of the switch is *secure*, and the flow table of the switch is empty at this point.

```
sudo ovs-ofctl dump-flows s1
```

```

ovs@admin:~$ sudo ovs-ofctl dump-flows s1
ovs@admin:~$ 

```

Figure 19. Verifying flow on switch s1.

**Step 8.** Test the connectivity between hosts h1 and host h2 using the `ping` command. There is no connectivity between hosts since switch s1 does not know how to process the traffic. To stop the test, press `Ctrl+c`.

```
ping 192.168.2.10
```

```
"Host: h1"
root@admin:/home/ovs# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
^C
--- 192.168.2.10 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1015ms
root@admin:/home/ovs#
```

Figure 20. Output of `ping` command.

**Step 9.** On the Linux terminal, type the following command to add an IP address to the switch, s1. The IP address is the gateway (192.168.1.1) of host h1.

```
sudo ifconfig s1 192.168.1.1/24 up
```

```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ifconfig s1 192.168.1.1/24 up
ovs@admin:~$
```

Figure 21. Adding IP address in the switch.

**Step 10.** Type the following command to add an IP address to the switch, s2. The IP address is the gateway (192.168.2.1) of host h2.

```
sudo ifconfig s2 192.168.2.1/24 up
```

```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ifconfig s2 192.168.2.1/24 up
ovs@admin:~$
```

Figure 22. Adding IP address in the switch.

#### 4 Table 0 - Classifier

In this section, you will configure table 0 which is the default table in Open vSwitch. Address Resolution Protocol (ARP) is handled in table 0. For the routing part, whenever there is match against IP, the switch will use other tables to look for further instructions to forward the traffic.

**Step 1.** Type the following command to manually insert an ARP flow entry in the switch s1.

```
sudo ovs-ofctl add-flow s1 "table=0,arp,action=normal"
```



```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=0,arp,action=normal"
```

Figure 23. Manually adding a flow entry.

Consider the figure above. The flow is for the ARP request. The command adds a flow that sends ARP requests to all the switch ports.

**Step 2.** Type the following command to manually insert a flow entry in switch s1. If there is a match against IP in table 0, the switch will check table 1 for further instruction.

```
sudo ovs-ofctl add-flow s1 "table=0,ip,action=goto_table=1"
```



```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=0,ip,action=goto_table=1"
```

Figure 24. Manually adding a flow entry.

Consider the figure above. If there is a match against IP in table 0, the switch will check table 1 for further instruction.

**Step 3.** Type the following command to manually insert an ARP flow entry in the switch s2.

```
sudo ovs-ofctl add-flow s2 "table=0,arp,action=normal"
```



```
ovs@admin:~$ sudo ovs-ofctl add-flow s2 "table=0,arp,action=normal"
```

Figure 25. Manually adding a flow entry.

Consider the figure above. The flow is for the ARP request. The command adds a flow that sends ARP requests to all the switch ports.

**Step 4.** Type the following command to manually insert a flow entry in switch s2.

```
sudo ovs-ofctl add-flow s2 "table=0,ip,action=goto_table=1"
```



```
ovs@admin:~$ sudo ovs-ofctl add-flow s2 "table=0,ip,action=goto_table=1"
```

Figure 26. Manually adding a flow entry.

Consider the figure above. If there is a match against IP in table 0, the switch will check table 1 for further instruction.

## 5 Table 1 – Layer 3 Forwarding

In this section, you will modify the source and destination MAC addresses, Decrement TTL and push to table 2 for forwarding.

**Step 1.** Type the following command to manually insert a flow entry in switch s1 for the traffic going to the destination host h1 (192.168.1.10).

```
sudo ovs-ofctl add-flow s1
"table=1,ip,nw_dst=192.168.1.10,action=mod_dl_src=00:00:00:00:00:05,
mod_dl_dst=00:00:00:00:00:01,dec_ttl,goto_table=2"
```

Figure 27. Manually adding a flow entry.

Consider the figure above. Whenever switch s1 receives any traffic and the destination host is 192.168.1.10, the source and destination MAC addresses are modified to 00:00:00:00:00:05 and 00:00:00:00:00:01, respectively. TTL value will be decreased by one. The switch will check table 2 for further instruction.

**Step 2.** Type the following command to manually insert a flow entry in switch s1 for the traffic going to the destination network 192.168.2.0/24.

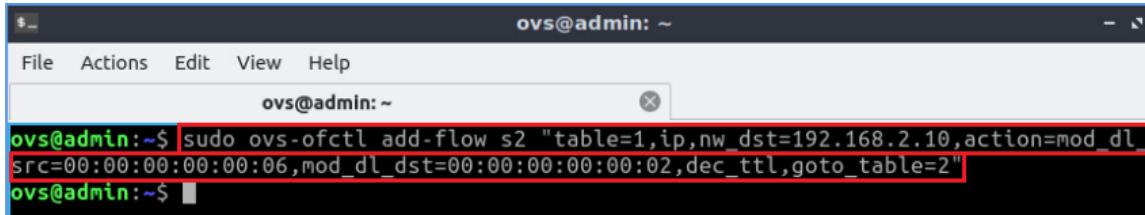
```
sudo ovs-ofctl add-flow s1
"table=1,ip,nw_dst=192.168.2.0/24,action=mod_dl_src=00:00:00:00:00:01,
mod_dl_dst=00:00:00:00:00:05,dec_ttl,goto_table=2"
```

Figure 28. Manually adding a flow entry.

Consider the figure above. Whenever switch s1 receives any traffic and the destination network is 192.168.2.0/24, the source and destination MAC addresses are modified to 00:00:00:00:00:01 and 00:00:00:00:00:05, respectively. TTL value will be decreased by one. The switch will check table 2 for further instruction.

**Step 3.** Type the following command to manually insert a flow entry in switch s2 for the traffic going to the destination host h2 (192.168.2.10).

```
sudo ovs-ofctl add-flow s2  
"table=1,ip,nw_dst=192.168.2.10,action=mod_dl_src=00:00:00:00:00:06,  
mod_dl_dst=00:00:00:00:00:02,dec_ttl,goto_table=2"
```



```
ovs@admin:~$ sudo ovs-ofctl add-flow s2 "table=1,ip,nw_dst=192.168.2.10,action=mod_dl_src=00:00:00:00:00:06,mod_dl_dst=00:00:00:00:00:02,dec_ttl,goto_table=2"  
ovs@admin:~$
```

Figure 29. Manually adding a flow entry.

Consider the figure above. Whenever switch s2 receives any traffic and the destination host is 192.168.2.10, the source and destination MAC addresses are modified to 00:00:00:00:00:06 and 00:00:00:00:00:02, respectively. TTL value will be decreased by one. The switch will check table 2 for further instruction.

**Step 4.** Type the following command to manually insert a flow entry in switch s2 for the traffic going to the destination network 192.168.1.0/24.

```
sudo ovs-ofctl add-flow s2  
"table=1,ip,nw_dst=192.168.1.0/24,action=mod_dl_src=00:00:00:00:00:02,  
mod_dl_dst=00:00:00:00:00:06,dec_ttl,goto_table=2"
```



```
ovs@admin:~$ sudo ovs-ofctl add-flow s2 "table=1,ip,nw_dst=192.168.1.0/24,action=mod_dl_src=00:00:00:00:00:02,mod_dl_dst=00:00:00:00:00:06,dec_ttl,goto_table=2"  
ovs@admin:~$
```

Figure 30. Manually adding a flow entry.

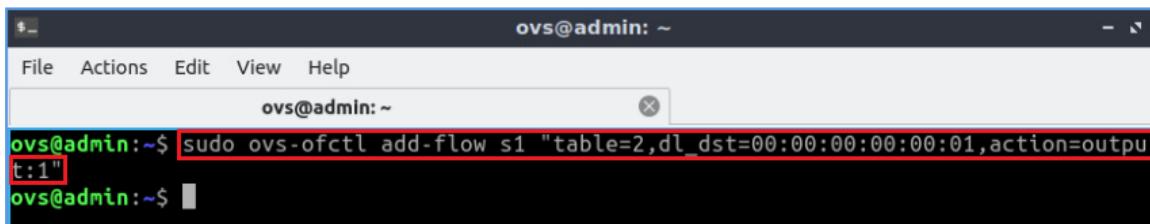
Consider the figure above. Whenever switch s2 receives any traffic and the destination network is 192.168.1.0/24, the source and destination MAC addresses are modified to 00:00:00:00:00:02 and 00:00:00:00:00:06, respectively. TTL value will be decreased by one. The switch will check table 2 for further instruction.

## 6 Table 2 – Layer 2 Forwarding

In this section, you will specify the forwarding ports for each destination MAC address.

**Step 1.** Type the following command to manually insert a flow entry in switch s1 to specify the forwarding port for destination MAC address 00:00:00:00:00:01.

```
sudo ovs-ofctl add-flow s1 "table=2,dl_dst=00:00:00:00:00:01,action=output:1"
```



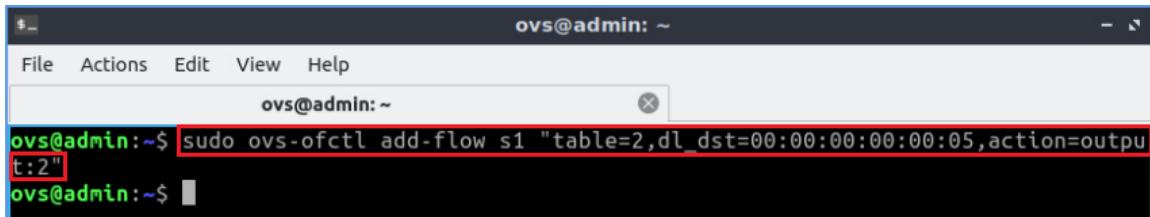
```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=2,dl_dst=00:00:00:00:01,action=output:1"
ovs@admin:~$
```

Figure 31. Manually adding a flow entry.

Consider the figure above. To forward any packet to the destination MAC 00:00:00:00:01, switch s1 will use interface *s1-eth1*.

**Step 2.** Type the following command to manually insert a flow entry in switch s1 to specify the forwarding port for destination MAC address 00:00:00:00:05.

```
sudo ovs-ofctl add-flow s1 "table=2,dl_dst=00:00:00:00:05,action=output:2"
```



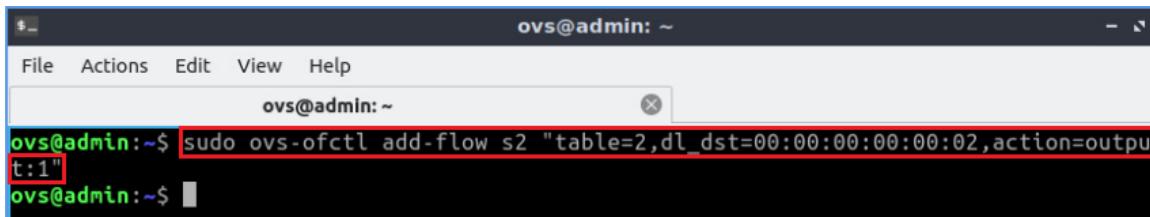
```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=2,dl_dst=00:00:00:00:05,action=output:2"
ovs@admin:~$
```

Figure 32. Manually adding a flow entry.

Consider the figure above. To forward any packet to the destination MAC 00:00:00:00:05, switch s1 will use interface *s1-eth2*.

**Step 3.** Type the following command to manually insert a flow entry in switch s2 to specify the forwarding port for destination MAC address 00:00:00:00:02.

```
sudo ovs-ofctl add-flow s2 "table=2,dl_dst=00:00:00:00:02,action=output:1"
```



```
ovs@admin:~$ sudo ovs-ofctl add-flow s2 "table=2,dl_dst=00:00:00:00:02,action=output:1"
ovs@admin:~$
```

Figure 33. Manually adding a flow entry.

Consider the figure above. To forward any packet to the destination MAC 00:00:00:00:02, switch s2 will use interface *s2-eth1*.

**Step 4.** Type the following command to manually insert a flow entry in switch s2 to specify the forwarding port for destination MAC address 00:00:00:00:06.

```
sudo ovs-ofctl add-flow s2 "table=2,dl_dst=00:00:00:00:06,action=output:2"
```

```
ovs@admin:~$ sudo ovs-ofctl add-flow s2 "table=2,dl_dst=00:00:00:00:00:06,action=output:2"
ovs@admin:~$
```

Figure 34. Manually adding a flow entry.

Consider the figure above. To forward any packet to the destination MAC 00:00:00:00:00:06, switch s2 will use interface *s2-eth2*.

## 7 Verifying configuration

**Step 1.** Type the following command to verify the flow installation. This command prints the flow table entries in switch s1.

```
sudo ovs-ofctl dump-flows s1
```

```
ovs@admin:~$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=834.693s, table=0, n_packets=0, n_bytes=0, arp actions=NORMAL
cookie=0x0, duration=805.568s, table=0, n_packets=3, n_bytes=261, ip actions=resubmit(,1)
cookie=0x0, duration=692.431s, table=1, n_packets=0, n_bytes=0, ip,nw_dst=192.168.1.1
0 actions=mod_dl_src:00:00:00:00:05,mod_dl_dst:00:00:00:00:01,dec_ttl,resubmit(,2)
cookie=0x0, duration=588.351s, table=1, n_packets=0, n_bytes=0, ip,nw_dst=192.168.2.0
/24 actions=mod_dl_src:00:00:00:00:01,mod_dl_dst:00:00:00:00:05,dec_ttl,resubmit(,2)
cookie=0x0, duration=271.583s, table=2, n_packets=0, n_bytes=0, dl_dst=00:00:00:00:00:00
:01 actions=output:"s1-eth1"
cookie=0x0, duration=198.517s, table=2, n_packets=0, n_bytes=0, dl_dst=00:00:00:00:00:00
:05 actions=output:"s1-eth2"
ovs@admin:~$
```

Figure 35. Verifying flow table in switch s1.

Consider the figure above. You will notice there are three different tables and flows associated with the tables.

Resubmit (,table\_id) refers to goto\_table=table\_id.

**Step 2.** Type the following command to verify the flow installation. This command prints the flow table entries in switch s2.

```
sudo ovs-ofctl dump-flows s2
```

```

ovs@admin:~$ sudo ovs-ofctl dump-flows s2
cookie=0x0, duration=879.915s, table=0, n_packets=0, n_bytes=0, arp actions=NORMAL
cookie=0x0, duration=848.884s, table=0, n_packets=3, n_bytes=261, ip actions=resubmit(,1)
cookie=0x0, duration=540.994s, table=1, n_packets=0, n_bytes=0, ip,nw_dst=192.168.2.1
0 actions=mod_dl_src:00:00:00:00:00:06,mod_dl_dst:00:00:00:00:00:02,dec_ttl,resubmit(,2)
cookie=0x0, duration=453.318s, table=1, n_packets=0, n_bytes=0, ip,nw_dst=192.168.1.0
/24 actions=mod_dl_src:00:00:00:00:00:02,mod_dl_dst:00:00:00:00:00:06,dec_ttl,resubmit(,2)
cookie=0x0, duration=189.459s, table=2, n_packets=0, n_bytes=0, dl_dst=00:00:00:00:00:00
:02 actions=output:"s2-eth1"
cookie=0x0, duration=130.865s, table=2, n_packets=0, n_bytes=0, dl_dst=00:00:00:00:00:00
:06 actions=output:"s2-eth2"
ovs@admin:~$ 

```

Figure 36. Verifying flow table in switch s2.

Consider the figure above. You will notice all the manual flows have been installed in the flow tables.

**Step 3.** Test the connectivity between host h1 and host h2 using the `ping` command.

```
ping 192.168.2.10
```

```

"Host: h1"
root@admin:/home/ovs# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
64 bytes from 192.168.2.10: icmp_seq=1 ttl=64 time=1.09 ms
64 bytes from 192.168.2.10: icmp_seq=2 ttl=64 time=0.062 ms
64 bytes from 192.168.2.10: icmp_seq=3 ttl=64 time=0.068 ms
^C
--- 192.168.2.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2017ms
rtt min/avg/max/mdev = 0.062/0.407/1.091/0.483 ms
root@admin:/home/ovs# 

```

Figure 37. Output of `ping` command.

The figure shows a successful connectivity test. To stop the test, press `Ctrl+C`.

**Step 4.** Now you will use another tool provided by Open vSwitch to trace the packet as it traverses the switch. `ovs-appctl` is the tracing tool that can be used to determine what is happening with packets as they go through the data plane processing (e.g., modified fields, output port, etc.). Type the command below to issue a packet with the following fields' values:

- Network layer protocol: IP
- Source IP address: 192.168.1.10
- Destination IP address: 192.168.2.10
- Source MAC address: 00:00:00:00:00:01
- Time-to-live (TTL): 64

```

sudo ovs-appctl ofproto/trace s1
ip,nw_src=192.168.1.10,nw_dst=192.168.2.10,dl_src=00:00:00:00:00:01,nw_ttl=64

```

```

ovs@admin:~$ sudo ovs-appctl ofproto/trace s1 ip,nw_src=192.168.1.10,nw_dst=192.168.2.10,dl_src=00:00:00:00:00:01,nw_ttl=64
Flow: ip,in_port=ANY,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:00,nw_src=192.168.1.10,nw_dst=192.168.2.10,nw_proto=0,nw_tos=0,nw_ecn=0,nw_ttl=64
bridge("s1")
0. ip, priority 32768
    resubmit(,1)
1. ip,nw_dst=192.168.2.0/24, priority 32768
    mod_dl_src:00:00:00:00:00:01
    mod_dl_dst:00:00:00:00:00:05
    dec_ttl
    resubmit(,2)
2. dl_dst:00:00:00:00:00:05, priority 32768
    output:2

Final flow: ip,in_port=ANY,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05,nw_src=192.168.1.10,nw_dst=192.168.2.10,nw_proto=0,nw_tos=0,nw_ecn=0,nw_ttl=63
Megaflow: recirc_id=0,eth,ip,in_port=ANY,dl_dst=00:00:00:00:00:00,nw_dst=192.168.2.0/24,nw_ttl=64,nw_frag=no
Datapath actions: set(eth(dst=00:00:00:00:00:05)),1
ovs@admin:~$ 

```

Figure 38. Tracing the packet processing.

Consider the figure above. The first gray box shows the flow fields' values before the flow is sent to the switch. Note that if a field is not specified (e.g., dl\_dst, or the destination MAC address), the tool uses the value 0 for that field. The values here match those specified in the command by the user. The second gray box shows the rules that were matched in the flow tables. The third gray box shows the final flow fields' values after the packet was processed in the switch.

This concludes Lab 6. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. Linux Foundation, “Open vSwitch”, [Online]. Available: <http://openvSwitch.org>.
2. B.Pfaff, B. Davie, Ed, “The Open vSwitch Database Management Protocol”, RFC 7047, Dec 2013.
3. Mininet walkthrough, [Online]. Available: <http://mininet.org>.
4. Open Networking Foundation, “OpenFlow Switch Specification”, [Online]. Available: <https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf>
5. Zhi Chen, Yulei Wu, Jingguo Ge, “A new lookup model for multiple flow tables of OpenFlow with implementation and optimization considerations”, Sep 2014.



## OPEN VIRTUAL SWITCH

### Exercise 2: Implementing Routing using Multiple Flow Tables

Document Version: **09-22-2021**



Award 1829698  
“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers”

## Contents

1	Exercise topology .....	3
1.1	Topology settings .....	3
1.2	Credentials .....	3
2	Deliverables.....	4

## 1 Exercise topology

The goal of this exercise is to manually configure multiple flow tables to enable packet forwarding between different networks.

Consider Figure 1. The topology consists of two end-hosts and four switches. Hosts h1 and h2 belong to the networks 192.168.1.0/24 and 192.168.2.0/24, respectively. Bandwidth for interface  $s1\text{-}eth2$  is set to 1 Gbps. High volume traffic more than 1 Gbps will pass through interface  $s1\text{-}eth3$ .

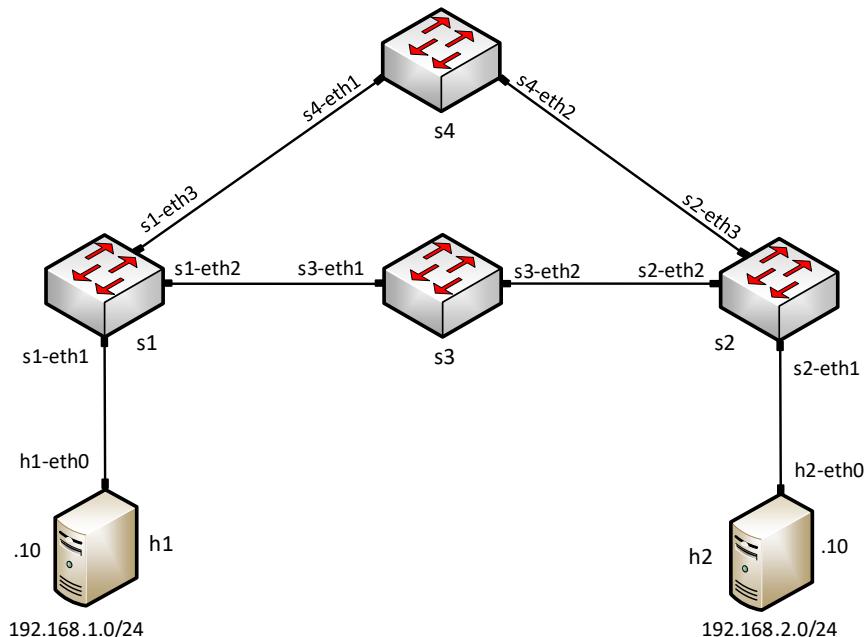


Figure 1. Exercise topology.

### 1.1 Topology settings

The devices are already configured according to Table 1.

Table 1. Topology information.

Host	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8

### 1.2 Credentials

The information in Table 2 provides the credentials to access the Client's virtual machine.

Table 2. Credentials to access the Client's virtual machine.

Device	Account	Password
Client	admin	password

## 2 Deliverables

Follow the steps below to complete the exercise.

- a) Start MiniEdit by clicking on MiniEdit's shortcut. Load the topology *Exercise2.mn* located at *~/OVS\_Labs/Exercise2* as shown in the figure below.

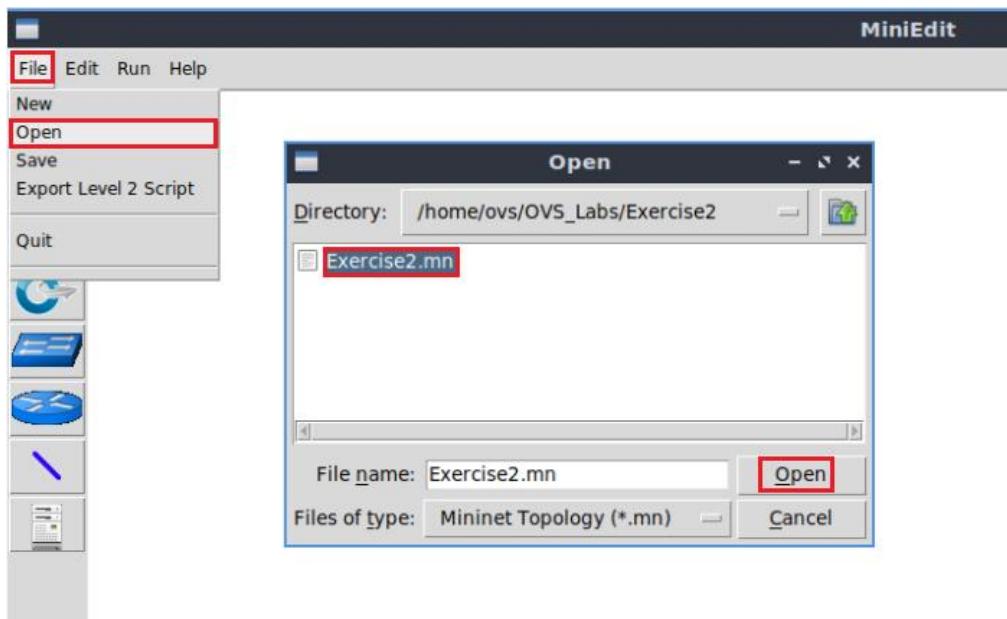


Figure 2. Loading the topology file in Mininet.

- b) Run the emulation in Mininet.
- c) In the Mininet terminal, launch the command that displays the interface names and connections of the current topology. Verify that links conform to the topology in Figure 1.
- d) *~/OVS\_Labs/Exercise2* folder contains two scripts *set\_MACs.sh* and *config\_switches.sh* responsible for loading the MAC addresses and the configuration needed for the exercise. Execute the script using the following command:

```
cd OVS_Labs/Exercise2
```

```
./set_MACs.sh
```

```
./config_switches.sh
```

- e) Add IP addresses to the switches s1 (192.168.1.1/24) and s2 (192.168.2.1/24).

- f) Enable the traditional switch forwarding operation in switches s3 and s4.
- g) Configure table 0 as classifier in switches s1 and s2. Set ARP flow to normal action. For any other traffic, switches s1 and s2 will check table 1.
- h) Configure table 1 for Layer 3 forwarding in switches s1 and s2. You will configure the switches so that IP traffic will take path s1<->s3<->s2. For any TCP traffic, the path is s1<->s4<->s2.
- i) Configure table 2 in switches s1 and s2 for Layer 2 forwarding.
- j) Verify the connectivity between hosts h1 and h2 using `ping` command.
- k) Run an iperf3 test between the hosts where host h2 is acting as an FTP server and h1 is an FTP client. Explain the result. What is the throughput?
- l) Verify the configuration using `ovs-appctl` command to trace the packet as it traverses switch s1.
- Issue one packet with Network layer protocol: IP, Source IP address: 192.168.1.10, Destination IP address: 192.168.2.10, Source MAC address: 00:00:00:00:00:01 and Time-to-live (TTL): 64.
  - Issue another packet with Network layer protocol: TCP, TCP port=21, Source IP address: 192.168.1.10, Destination IP address: 192.168.2.10, Source MAC address: 00:00:00:00:00:01 and Time-to-live (TTL): 64.

Explain the results. Which path does the switch take for each packet?



## OPEN VIRTUAL SWITCH

### Lab 7: Configuring Stateless Firewall using Access Control Lists (ACLs)

Document Version: **09-07-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    Access Control Lists (ACLs) .....	4
1.2    ACL implementation in Open vSwitch.....	5
2    Lab topology.....	6
2.1    Lab settings.....	6
2.2    Loading a topology .....	7
2.3    Loading the configuration file .....	9
3    Verifying IP addresses on the hosts .....	10
4    Configuring ACLs in switch s1 .....	11
5    Verifying configuration .....	13
References .....	15

## Overview

This lab discusses the concept of Access Control Lists (ACLs), a set of rules that acts as a firewall to control incoming and outgoing traffic. The lab aims to configure ACLs in Open Virtual Switch (Open vSwitch) that controls whether to accept or deny traffic based on the source and destination IP addresses.

## Objectives

By the end of this lab, you should be able to:

1. Understand the concept of a firewall.
2. Understand the concept of ACL.
3. Explain the ACL implementation in OpenFlow.
4. Configure and verify ACLs in Open vSwitch.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Verifying IP addresses on the hosts.
4. Section 4: Configuring ACLs in switch s1.
5. Section 5: Configuring traditional forwarding in the switch.
6. Section 6: Verifying configuration.

## 1      Introduction

Network security is one of the most important aspects to consider while protecting company assets when working over the Internet. An efficient and robust network security

system is essential to protect online information and data since no network is immune to attacks. Network attacks can cause huge financial and operational losses to any organization<sup>1</sup>.

Firewalls are a primary line of defense in network security. Network designers use firewalls to shield networks from unauthorized users. A firewall is used to establish a barrier between a trusted network and an untrusted network. It keeps destructive and disruptive forces out and controls the incoming and outgoing network traffic supported by predetermined security rules<sup>2</sup>. A firewall protects many potentially exploitable internal programs from danger by limiting the traffic that crosses the network boundary to only authorized traffic.

## 1.1 Access Control Lists (ACLs)

ACLs are widely utilized in computer networking and network security to mitigate network attacks and control network traffic. It performs packet filtering and consists of sequential series of statements called Access Control Entry (ACE). Each ACE specifies a matching criterion and an action that can be either permit or deny. Packet filtering can restrict users' access and devices to a network, providing a measure of security and saving network resources by reducing traffic<sup>3</sup>. ACLs can allow one host to access a part of the network and prevent another host from accessing the same area. The matching criteria refer to packets match against the packet header fields such as Ethernet source/destination addresses, IP source/destination addresses, IP protocol, source/destination ports. A router configured with an ACL extracts the source address from the packet header. The router starts at the top of the ACL and compares the address to each ACE sequentially. When a match is made, the router carries out the instruction, either permitting or denying the packet. After a match is made, the remaining ACEs in the ACL are not analyzed. If the source IP address does not match any ACEs in the ACL, the packet is discarded.

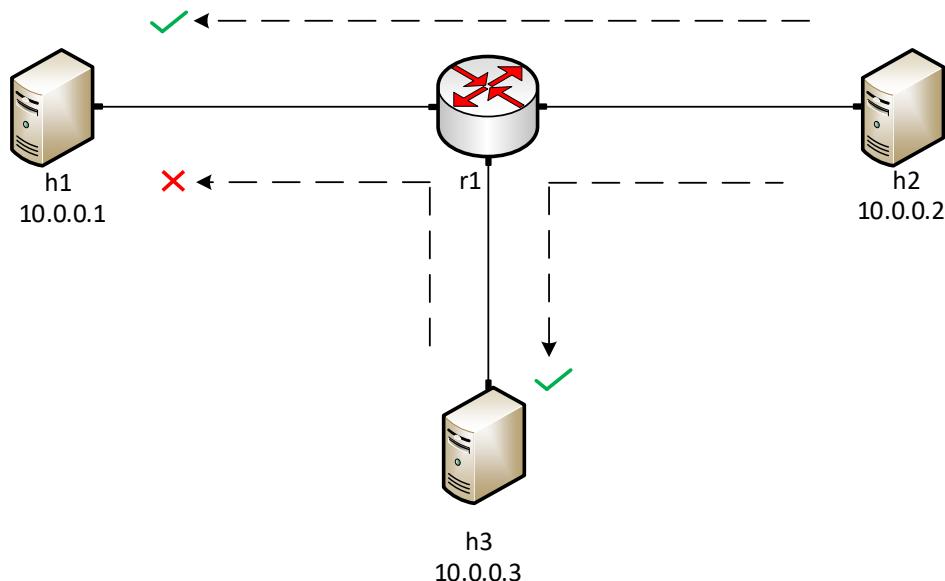


Figure 1. Packet filtering using ACLs.

Consider Figure 1. There are three hosts connected to router r1. ACLs are configured in router r1 to deny connection from host h3 to host h1 and allow any other traffic. Router r1 will allow any other traffic.

Table 2. ACL table.

Source	Destination	Protocol	Access
10.0.0.3	10.0.0.1	any	Deny
any	any	any	permit

Consider Table 2. The table contains ACLs running in router r1. If host h3 (10.0.0.3) generates traffic for the destination host h1 (10.0.0.1), router r1 will analyze the table, and the first entry will be a match. Since the match has been found, the router will discard the packet. If host h2 wants to communicate with host h1, the second entry will be a match, and the router will permit the traffic. If the second entry is considered as the first entry (permit any source), whenever host h3 wants to communicate with host h1, the first entry will be the match and the traffic will be allowed. Therefore, it is important to order all ACL statements from most specific (deny traffic generated by host h3 for the destination host h1) to least specific (permit any traffic) since the match is sequential.

## 1.2 ACL implementation in Open vSwitch

In an OpenFlow-based firewall, the rules are pre-installed onto the switch's flow table. Each packet header is checked against the firewall rule from highest to lowest priority. In Open vSwitch, packets match against the packet header fields such as switch input port, Virtual Local Area Network (VLAN) ID, Ethernet source/destination addresses, IP source/destination addresses, IP protocol, source/destination ports. If a matching entry is found in a table, the instructions associated with that specific flow entry are executed. Any unmatched packets are dropped<sup>4</sup>.

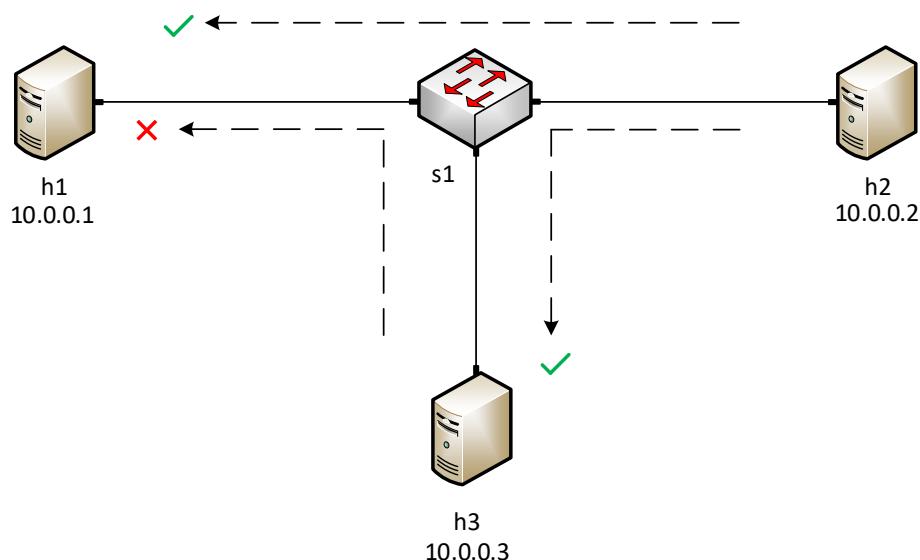


Figure 2. Packet filtering in Open vSwitch.

Consider Figure 2. Three hosts are connected to an Open vSwitch, s1. Switch s1 will act as a packet filtering firewall. All the flows are installed in the switch. If host h3 (10.0.0.3) generates traffic for the destination host h1 (10.0.0.1), switch s1 will analyze the flow tables and the traffic will be blocked. All other traffic will be allowed.

Table 3. Open vSwitch ACL table.

Priority	Table	Source	Destination	Protocol	Action
40000	0	10.0.0.3	10.0.0.1	any	drop
32768	0	any	any	any	Goto_table=1

Table 4. Open vSwitch Forwarding table.

Priority	Table	Source	Destination	Protocol	Action
32768	1	any	any	any	Normal

Consider Table 3 and Table 4. The tables contain firewall rules in switch s1. If host h3 wants to communicate with host h1, switch s1 will analyze the table based on the priority and the first entry from table 0 will be a match, and the traffic will be dropped. For any other traffic, the second entry will be a match, and the switch will check table 1 to forward the traffic.

## 2 Lab topology

Consider Figure 3. The topology consists of three hosts and one switch. Hosts h1, h2, and h3 belong to the same network, 10.0.0.0/8.

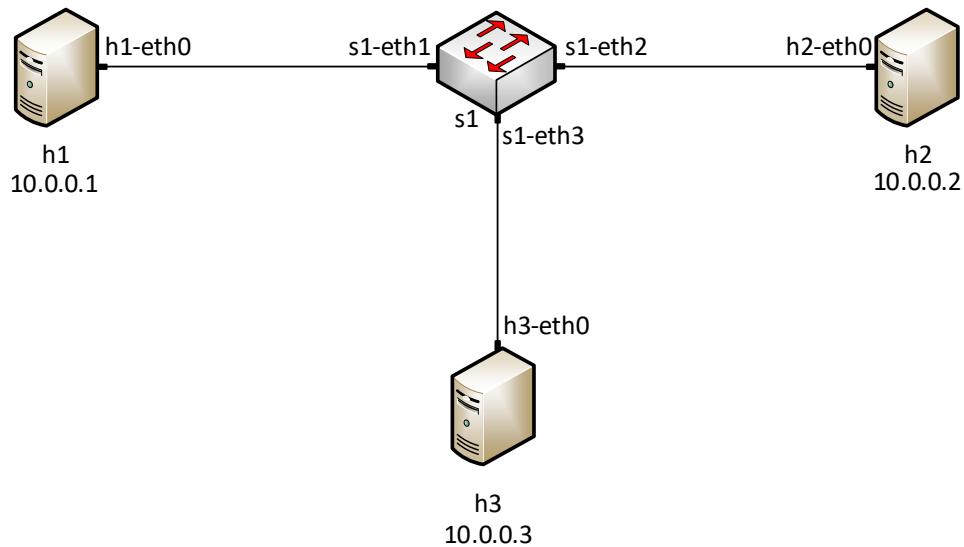


Figure 3. Lab topology.

### 2.1 Lab settings

The hosts are configured according to Table 4.

Table 4. Topology information.

Host	Interface	IP Address	Subnet	Default gateway
h1	h1-eth0	10.0.0.1	/8	N/A
h2	h2-eth0	10.0.0.2	/8	N/A
h3	h3-eth0	10.0.0.3	/8	N/A

## 2.2 Loading a topology

**Step 1.** Start by launching MiniEdit by clicking on the desktop's shortcut. When prompted for a password, type `password`.

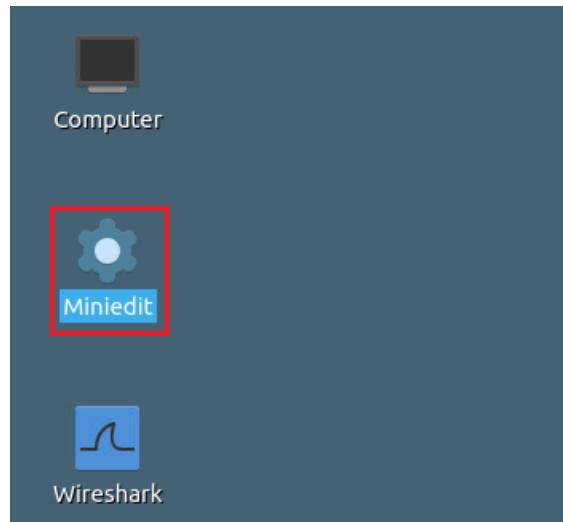


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File*, then *open* to load the lab's topology. Locate the *lab7.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab7* and click on *Open*.

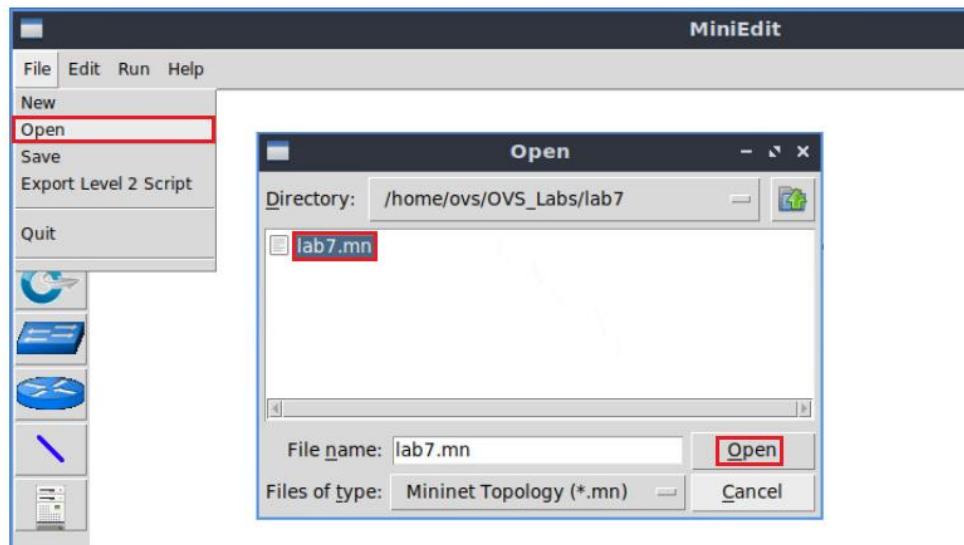


Figure 5. MiniEdit's Open dialog.

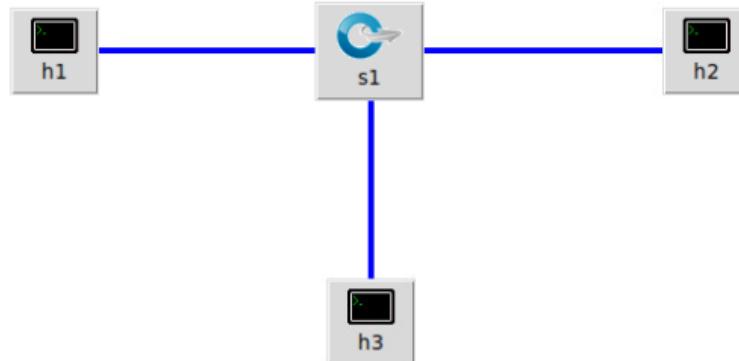


Figure 6. MiniEdit's topology.

**Step 3.** To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 7. Starting the emulation.

**Step 4.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

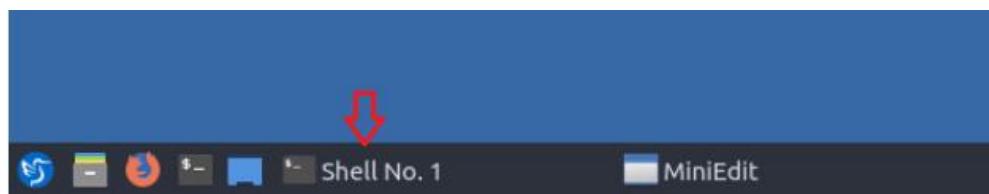


Figure 8. Opening Mininet's terminal.

**Step 5.** Issue the following command to display the interface names and connections.

```
links
```

```
containernet> links
h1-eth0<->s1-eth1 (OK OK)
s1-eth2<->h2-eth0 (OK OK)
s1-eth3<->h3-eth0 (OK OK)
containernet>
```

Figure 9. Displaying network interfaces.

In Figure 9, the link displayed within the gray box indicates that interface *eth0* of host *h1* connects to interface *eth1* of host *s1* (i.e., *h1-eth0<->s1-eth1*).

### 2.3 Loading the configuration file

**Step 1.** Open the Linux terminal.



Figure 10. Opening Linux terminal.

**Step 2.** Click on the Linux terminal and navigate into *OVS\_Labs/lab7* directory by issuing the following command. This folder contains a configuration file and the script responsible for loading the configuration. The configuration file will assign the Media Access Control (MAC) addresses to the hosts' interfaces. The `cd` command is short for change directory, followed by an argument that specifies the destination directory.

```
cd OVS_Labs/lab7
```

```
ovs@admin: ~$ cd OVS_Labs/lab7
ovs@admin: ~/OVS_Labs/lab7$
```

Figure 11. Entering to the *OVS\_Labs/lab7* directory.

**Step 3.** To execute the shell script, type the following command. The program's argument corresponds to a file that will be loaded in all the hosts to set a manual MAC address. When prompted for a password, type `password`.

```
./set_MACs.sh
```

Figure 12. Executing the shell script to load the configuration.

**Step 4.** Type the following command to exit the lab directory.

```
cd
```

Figure 13. Exiting from the lab directory.

### 3 Verifying IP addresses on the hosts

In this section, you will verify that IP and MAC addresses on the hosts are assigned according to table 4.

**Step 1.** Hold right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

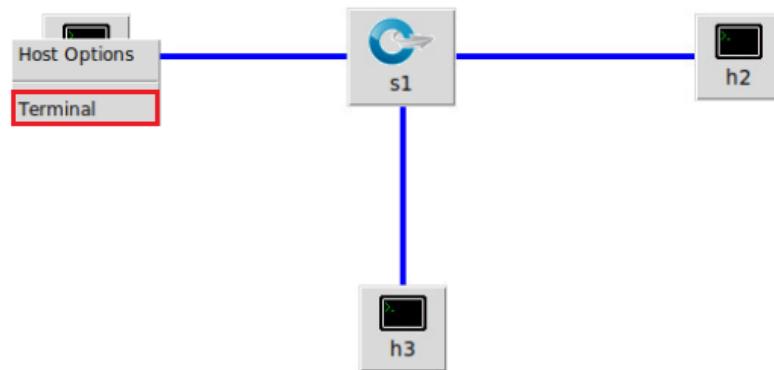


Figure 14. Opening a terminal on host h1.

**Step 2.** In host h1 terminal, type the following command to verify that the IP address was assigned successfully. You will verify the host interface, *h1-eth0* configured with the IP address 10.0.0.1 and the subnet mask 255.0.0.0. You will also verify the MAC address, 00:00:00:00:00:01.

```
ifconfig
```

```
"Host: h1"
root@admin:/home/ovs# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0
                ether 00:00:00:00:00:01 txqueuelen 1000  (Ethernet)
                RX packets 21 bytes 2910 (2.9 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 3 bytes 270 (270.0 B)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000  (Local Loopback)
            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

root@admin:/home/ovs# "
```

Figure 15. Verifying IP and MAC address in the host.

**Step 3.** You can also repeat steps 1-2 to verify hosts h2 and h3. You will notice the MAC addresses of hosts h2 and h3 are 00:00:00:00:00:02 and 00:00:00:00:00:03, respectively.

#### 4 Configuring ACLs in switch s1

In this section, you will configure switch s1 so that the switch blocks the traffic when the source address is 10.0.0.3 and the destination address is 10.0.0.1.

**Step 1.** Type the following command to manually insert a flow into switch s1.

```
sudo ovs-ofctl add-flow s1
"table=0,priority=40000,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1,action=drop"
```

```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=0,priority=40000,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1,action=drop"
[sudo] password for ovs:
ovs@admin:~$
```

Figure 16. Manually adding a flow entry.

Consider the figure above. Whenever host h3 (10.0.0.3) generates traffic for the destination host h1 (10.0.0.1), switch s1 will drop the traffic. The priority value for this entry is 40000.

**Step 2.** Type the following command to manually insert a flow into switch s1.

```
sudo ovs-ofctl add-flow s1 "table=0,action=goto_table=1"
```

```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=0,action=goto_table=1"
ovs@admin:~$
```

Figure 17. Manually adding a flow entry.

Consider the figure above. Switch s1 is instructed to check table 1 for further instruction if there is any traffic. The priority value for this entry is 32768 which is the default priority value in Open vSwitch.

Note that, there are two flow entries in table 0. Packets will be matched against flow entries based on priority. Whenever host h3 (10.0.0.3) generates traffic for the destination host h1 (10.0.0.1), the entry with higher priority (40000) in table 0 will be a match and the traffic will be dropped. For any other traffic, flow entry with default priority (32768) will be a match and the switch will check table 1 for further instruction.

**Step 3.** Type the following command to manually insert a flow into switch s1. A *normal* action allows the device to conduct normal layer 2/layer 3 packet processing like a traditional switch.

```
sudo ovs-ofctl add-flow s1 "table=1,action=normal"
```

```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=1,action=normal"
ovs@admin:~$
```

Figure 18. Adding normal flow in switch s1.

There are multiple ways to configure flow tables. You can configure flow tables based on different protocols, source/destination addresses and ports. For simplicity, *normal* action has been used in this lab. To explore more about OpenFlow implementation, you can check out labs 10, 11, 12 of this series.

## 5 Verifying configuration

**Step 1.** Type the following command to verify the flow installation. This command prints the flow table entries in switch s1.

```
sudo ovs-ofctl dump-flows s1
```

```
ovs@admin:~$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=373.016s, table=0, n_packets=0, n_bytes=0, priority=40000, ip,nw_src=192.168.3.0/24,nw_dst=192.168.1.0/24 actions=drop
cookie=0x0, duration=273.293s, table=0, n_packets=2, n_bytes=194, actions=resubmit(,1)
cookie=0x0, duration=212.376s, table=1, n_packets=0, n_bytes=0, actions=NORMAL
ovs@admin:~$
```

Figure 19. Verifying flow table in switch s3.

Consider the figure above. You will notice there are two different tables and flows associated with the tables.

Resubmit (,table\_id) refers to goto\_table=table\_id.

**Step 2.** Test the connectivity between host h1 and host h2 using the `ping` command.

```
ping 10.0.0.2
```

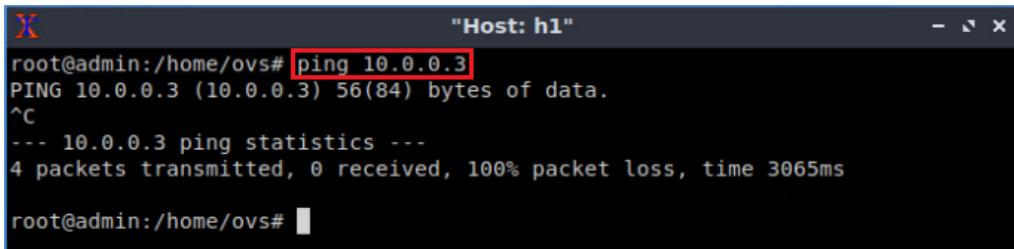
```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.17 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.073 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2031ms
rtt min/avg/max/mdev = 0.073/0.439/1.167/0.514 ms
root@admin:/home/ovs#
```

Figure 20. Output of `ping` command.

The figure shows a successful connectivity test. To stop the test, press `Ctrl+c`.

**Step 3.** Test the connectivity between host h1 and host h3 using the `ping` command.

```
ping 10.0.0.3
```



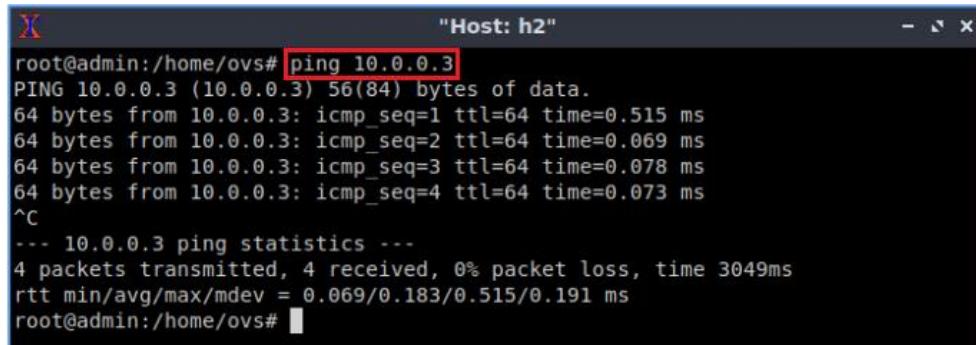
```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3065ms
root@admin:/home/ovs#
```

Figure 21. Output of `ping` command.

You will notice that there is no connectivity between hosts h1 and h3. To stop the test, press `Ctrl+c`.

**Step 4.** Test the connectivity between host h2 and host h3 using the `ping` command.

```
ping 10.0.0.3
```



```
"Host: h2"
root@admin:/home/ovs# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.515 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.069 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.078 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.073 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3049ms
rtt min/avg/max/mdev = 0.069/0.183/0.515/0.191 ms
root@admin:/home/ovs#
```

Figure 22. Output of `ping` command.

The figure shows a successful connectivity test. To stop the test, press `Ctrl+c`.

**Step 5.** Now you will use another tool provided by Open vSwitch to trace the packet as it traverses the switch. `ovs-appctl` is the tracing tool that can be used to determine what is happening with packets as they go through the data plane processing (e.g., modified fields, output port, etc.). Type the command below to issue a packet with the following fields' values:

- Network layer protocol: IP
- Source IP address: 10.0.0.1
- Destination IP address: 10.0.0.3
- Source MAC address: 00:00:00:00:00:01
- Destination MAC address: 00:00:00:00:00:03

```
sudo ovs-appctl ofproto/trace s1
ip,nw_src=10.0.0.1,nw_dst=10.0.0.3,dl_src=00:00:00:00:00:01,
dl_dst=00:00:00:00:00:03
```

```
ovs@admin:~$ sudo ovs-appctl ofproto/trace s1 ip,nw_src=10.0.0.1,nw_dst=10.0.0.3,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03
00:03
Flow: ip,in_port=ANY,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_pro
to=0,nw_tos=0,nw_ecn=0,nw_ttl=0
bridge("s1")
0. priority 32768
    resubmit(,1)
1. priority 32768
    NORMAL
        -> forwarding to learned port

Final flow: unchanged
Megaflow: recirc_id=0,eth,ip,in_port=ANY,vlan_tci=0x0000/0x1fff,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03,nw_dst=10.0
.0.2/31,nw_frag=no
Datapath actions: 6
ovs@admin:~$
```

Figure 23. Tracing the packet processing.

Consider the figure above. The gray box shows the rules that were matched in the flow tables. There was a match in table 0 and the switch was instructed to check table 1. From table 1, the switch chooses the normal action, and the packet is forwarded to the destination port.

**Step 6.** Type the following command to trace another packet as it traverses switch s1.

```
sudo ovs-appctl ofproto/trace s1
ip,nw_src=10.0.0.3,nw_dst=10.0.0.1,dl_src=00:00:00:00:00:03,
dl_dst=00:00:00:00:00:01
```

```
ovs@admin:~$ sudo ovs-appctl ofproto/trace s1 ip,nw_src=10.0.0.3,nw_dst=10.0.0.1,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01
00:01
Flow: ip,in_port=ANY,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_pro
to=0,nw_tos=0,nw_ecn=0,nw_ttl=0
bridge("s1")
0. ip,nw_src=10.0.0.3,nw_dst=10.0.0.1, priority 40000
    drop

Final flow: unchanged
Megaflow: recirc_id=0,eth,ip,in_port=ANY,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_frag=no
Datapath actions: drop
ovs@admin:~$
```

Figure 24. Tracing the packet processing.

Consider the figure above. Whenever switch s1 receives traffic where the source address is 10.0.0.3 and the destination address is 10.0.0.1, the traffic will be dropped in table 0.

Switch s1 will allow traffic generated by host h1 for the destination host h3. Whenever host h3 is generating reply packet for the destination host h1, the traffic will be blocked.

This concludes Lab 7. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. Cisco, “What is network security?”, [Online], Available: <https://www.cisco.com/c/en/us/products/security/what-is-network-security.html>

2. Comodo antivirus, “*What is firewall and types of firewall*”, May 2020.
3. U.S Patent, “*Method for configuring ACLs on network device based on flow information*”, Oct 2012.
4. Open networking foundation, “*OpenFlow switch specification*”, Sep 2012.
5. James F. Kurose, Keith W. Ross, “*Computer networking: A top-down approach*”, 7<sup>th</sup> edition, 2017.
6. Cisco, “*Security configuration guide: access control lists, Cisco IOS XE release 3S*”, [Online]. Available: [https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec\\_data\\_acl/configuration/xe-3s/sec-data-acl-xe-3s-book/sec-acl-named.html](https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_data_acl/configuration/xe-3s/sec-data-acl-xe-3s-book/sec-acl-named.html)
7. Linux foundation, “*Open vSwitch*”, [Online]. Available: <http://openvSwitch.org>.
8. RFC 7047, “*The Open vSwitch database management protocol*”, Dec 2013.
9. IBM, “*Virtual networking in Linux*”, [Online]. Available: <https://developer.ibm.com/tutorials/l-virtual-networking/>
10. Mininet walkthrough, [Online]. Available: <http://mininet.org>.



## OPEN VIRTUAL SWITCH

### Lab 8: Configuring Stateful Firewall using Connection Tracking (conntrack)

Document Version: **09-13-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    Stateful Firewall.....	4
1.2    Connection Tracking System (conntrack).....	5
2    Lab topology.....	6
2.1    Lab settings.....	6
2.2    Loading a topology .....	6
2.3    Loading the configuration file .....	8
3    Verifying IP addresses on the hosts .....	10
4    Configuring stateful firewall in switch s1.....	11
5    Verifying configuration .....	13
6    Monitoring conntrack state table .....	16
References .....	19

## Overview

This lab introduces the concept of stateful firewall, a firewall that monitors the full state of active network connections. This lab also introduces the Connection Tracking system (conntrack), a feature of Linux kernel responsible for monitoring the state of the connection. This lab aims to configure a stateful firewall in Open Virtual Switch (Open vSwitch) using conntrack.

## Objectives

By the end of this lab, you should be able to:

1. Explain the limitation of stateless firewall.
2. Understand the concept of stateful firewall.
3. Understand the concept of Connection Tracking (conntrack).
4. Explore conntrack features.
5. Configure stateful firewall in Open vSwitch using conntrack.
6. Monitor conntrack state table to verify the configuration.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Verifying IP addresses on the hosts.
4. Section 4: Configuring conntrack in the switches.
5. Section 5: Verifying configuration.
6. Section 6: Monitoring conntrack state table.

## 1      Introduction

The main goal of a firewall is to manage network traffic across different areas of a given local network. It establishes a barrier between a trusted and an untrusted network. Within the stateless firewall, the filtering actions (accepting or rejecting packets) are taken according to a set of static configuration rules. These rules are based on the information contained within the packet itself, like network addresses (source and destination), ports, and protocols. The main advantage of stateless firewalls is their filtering operations speed. However, since they cannot keep track of the state of a connection, they fail at handling some vulnerabilities that benefit from the position of a packet within existing streams of traffic<sup>1</sup>.

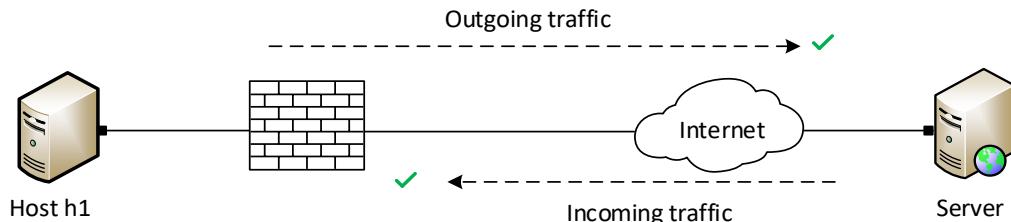


Figure 1. Stateless Firewall.

Consider Figure 1. Traffic originated from host h1 is permitted. To get a reply from the server, it will allow the incoming packet. Stateless firewalls only examine the packet header. On the contrary, stateful firewalls keep track of the state of active network connections while analyzing incoming traffic. Since the filtering action cannot differentiate if the traffic is trusted or not, it is easy for attackers to perform IP spoofing because the packet headers can be easily forged.

## 1.1 Stateful Firewall

A stateful firewall keeps track of the state of the network connections for applying the firewall policy. It is important to monitor the state and context of network communications because this information can be used to identify threats—either based on where they are coming from, where they are going, or the content of their data packets.

This stateful inspection within the firewall occurs at layers 3 and layer 4 of the Open System Interconnected (OSI) model. Whenever a packet is to be sent across the firewall, information about the source and destination IP addresses, port numbers, and connection status are stored within the state table, which is used to determine which network packets should be allowed through the firewall<sup>3</sup>. A stateful firewall can refuse to accept any packet from a remote host to a local host unless the local host has previously sent a packet to the remote host<sup>2</sup>. It provides a better fine-grained filtering capability and protects against more complex attacks, like denial of service (DOS) and IP Spoofing.

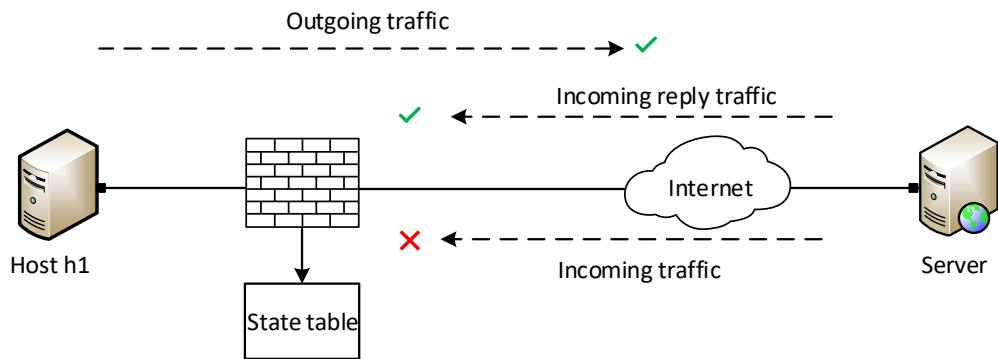


Figure 2. Stateful Firewall.

Consider Figure 2. Traffic originating from host h1 is permitted and inspected as it travels toward the public network (e.g., the Internet). Inspected traffic returning from the public network and associated with traffic that originated from host h1 is permitted. When the connection ends, that opening is closed. Traffic originating from the public network and traveling to host h1 will be blocked.

## 1.2 Connection Tracking System (conntrack)

In Linux, Iptables is a command-line firewall utility that uses policy to allow or block traffic. Iptables in the firewall works by interacting with the Netfilter which is a packet filtering framework in Linux<sup>6</sup>.

Connection tracking (conntrack) is one of the main features of the Linux kernel's networking stack, allowing the kernel to keep track of all network connections or flows. Conntrack stores information about the state of a connection (TCP/UDP/ICMP) in a state table that contains the source and destination IP addresses, port number pairs, protocol types, state, and timeout. A more intelligent filtering policy can be defined with this information. This state table is used by iptables to accept/drop packets based on the characteristic of the connection they belong to<sup>6</sup>.

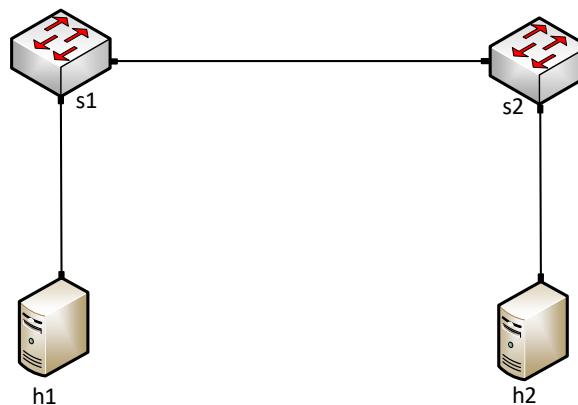


Figure 3. Open vSwitch conntrack.

Consider Figure 3. Manual flows and conntrack are configured in the switches. Switch s1 will allow all outgoing traffic (new) and permits only reply traffic from switch s2 (established). If host h1 generates a new packet for the destination host h2, switch s1 will

store the flow as a new connection and send a request to host h2. Host h2 will reply, and switch s1 will monitor the flow. Since it is the reply packet, the state of the connection is established and allowed through the firewall. Any new packet generated from host h2 for the destination host h1 will be dropped.

## 2 Lab topology

Consider Figure 4. The topology consists of three hosts and one switch. Hosts h1, h2, and h3 belong to the same network, 10.0.0.0/8.

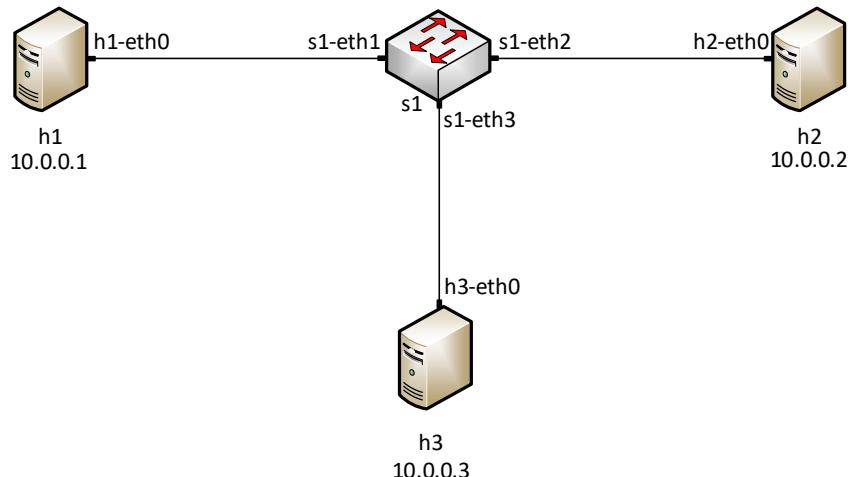


Figure 4. Lab topology.

### 2.1 Lab settings

The hosts are configured according to Table 2.

Table 2. Topology information.

Host	Interface	IP Address	Subnet	Default gateway
h1	h1-eth0	10.0.0.1	/8	N/A
h2	h2-eth0	10.0.0.2	/8	N/A
h3	h3-eth0	10.0.0.3	/8	N/A

### 2.2 Loading a topology

**Step 1.** Start by launching MiniEdit by clicking on the desktop's shortcut. When prompted for a password, type `password`.

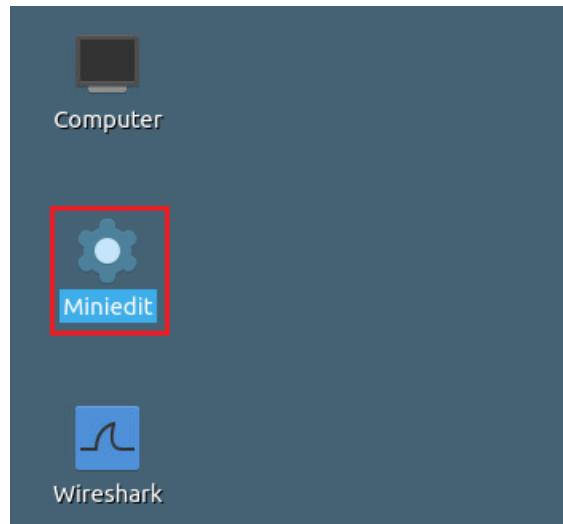


Figure 5. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File*, then *open* to load the lab's topology. Locate the *lab8.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab8* and click on *Open*.

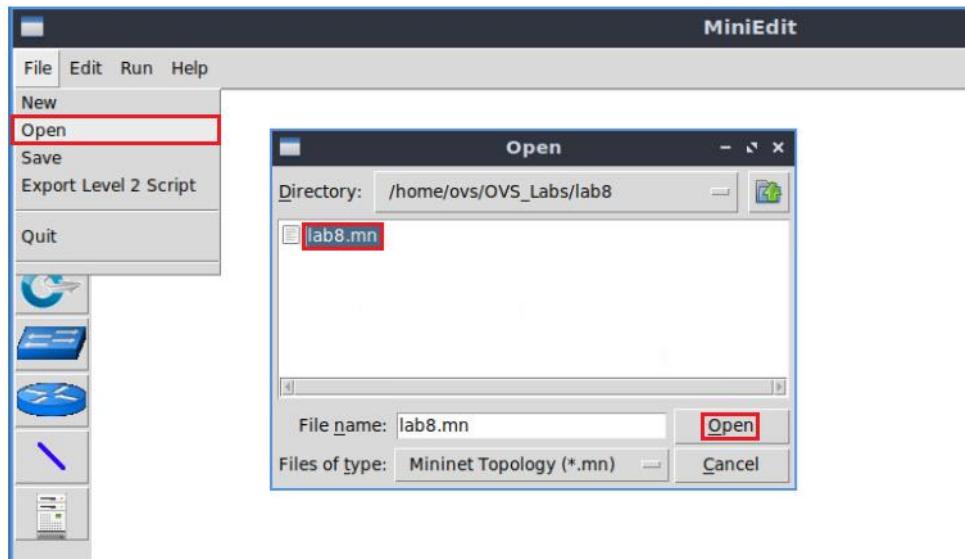


Figure 6. MiniEdit's Open dialog.

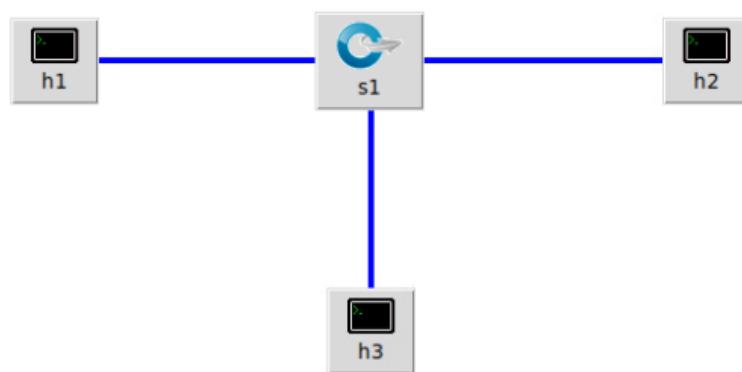


Figure 7. MiniEdit's topology.

**Step 3.** To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 8. Starting the emulation.

**Step 4.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

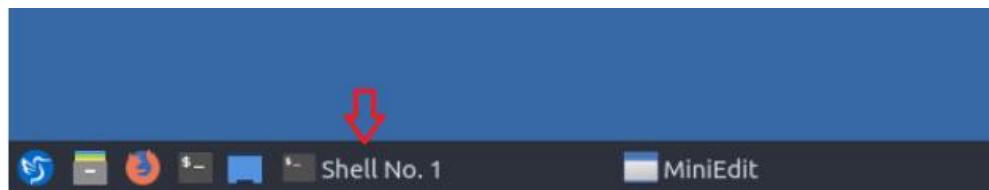


Figure 9. Opening Mininet's terminal.

**Step 5.** Issue the following command to display the interface names and connections.

```
links
```

A screenshot of the 'Shell No. 1' terminal window. The command 'links' has been entered and its output is displayed. The output shows four network links: 'h1-eth0<->s1-eth1' (OK OK), 's1-eth2<->h2-eth0' (OK OK), 's1-eth3<->h3-eth0' (OK OK), and 'containernet>'. A red box highlights the word 'links' in the command line. The terminal window has a dark background with white text.

Figure 10. Displaying network interfaces.

In Figure 10, the link displayed within the gray box indicates that interface *eth0* of host *h1* connects to interface *eth1* of host *s1* (i.e., *h1-eth0<->s1-eth1*).

## 2.3 Loading the configuration file

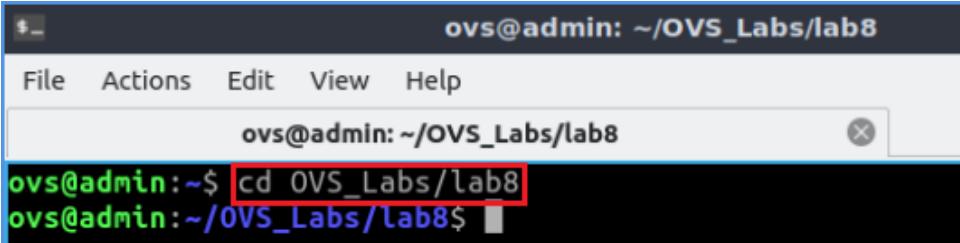
**Step 1.** Open the Linux terminal.



Figure 11. Opening Linux terminal.

**Step 2.** Click on the Linux terminal and navigate into *OVS\_Labs/lab8* directory by issuing the following command. This folder contains a configuration file and the script responsible for loading the configuration. The configuration file will assign the Media Access Control (MAC) addresses to the hosts' interfaces. The `cd` command is short for change directory, followed by an argument that specifies the destination directory.

```
cd OVS_Labs/lab8
```

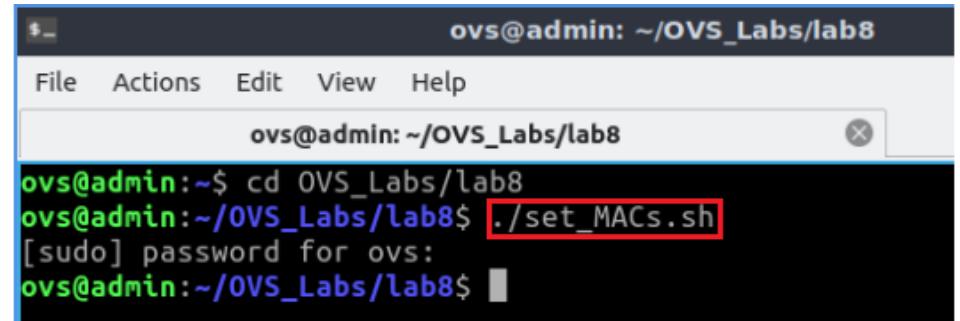


The screenshot shows a terminal window titled 'ovs@admin: ~/OVS\_Labs/lab8'. The menu bar includes 'File', 'Actions', 'Edit', 'View', and 'Help'. The title bar also displays 'ovs@admin: ~/OVS\_Labs/lab8'. The terminal prompt is 'ovs@admin:~\$'. A red box highlights the command 'cd OVS\_Labs/lab8' as it is being typed. The command has been completed and the new directory path 'ovs@admin:~/OVS\_Labs/lab8\$' is shown in blue.

Figure 12. Entering to the *OVS\_Labs/lab8* directory.

**Step 3.** To execute the shell script, type the following command. The program's argument corresponds to a file that will be loaded in all the hosts to set a manual MAC address. When prompted for a password, type `password`.

```
./set_MACs.sh
```

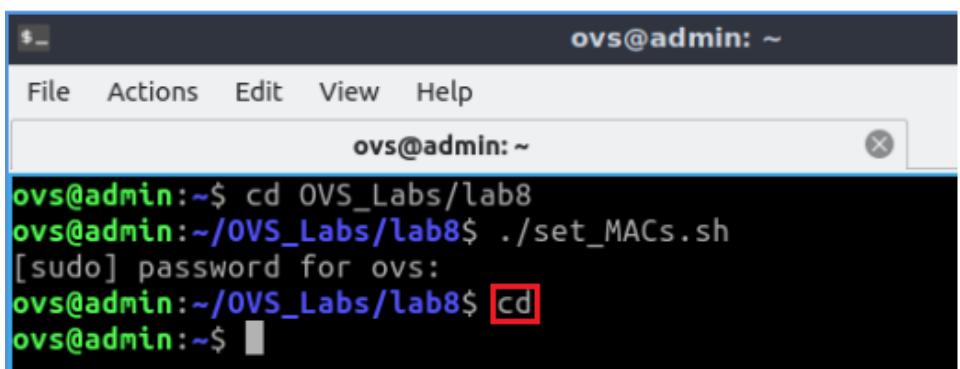


The screenshot shows a terminal window titled 'ovs@admin: ~/OVS\_Labs/lab8'. The menu bar includes 'File', 'Actions', 'Edit', 'View', and 'Help'. The title bar also displays 'ovs@admin: ~/OVS\_Labs/lab8'. The terminal prompt is 'ovs@admin:~\$'. A red box highlights the command './set\_MACs.sh' as it is being typed. Below the prompt, the text '[sudo] password for ovs:' is visible, indicating a password is required. The command has been completed and the new directory path 'ovs@admin:~/OVS\_Labs/lab8\$' is shown in blue.

Figure 13. Executing the shell script to load the configuration.

**Step 4.** Type the following command to exit the lab8 directory.

```
cd
```



The screenshot shows a terminal window titled 'ovs@admin: ~'. The menu bar includes 'File', 'Actions', 'Edit', 'View', and 'Help'. The title bar also displays 'ovs@admin: ~'. The terminal prompt is 'ovs@admin:~\$'. A red box highlights the command 'cd' as it is being typed. The command has been completed and the new directory path 'ovs@admin:~\$' is shown in blue.

Figure 14. Exiting from the lab directory.

### 3 Verifying IP addresses on the hosts

In this section, you will verify that IP addresses on the hosts are assigned according to table 2.

**Step 1.** Hold right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

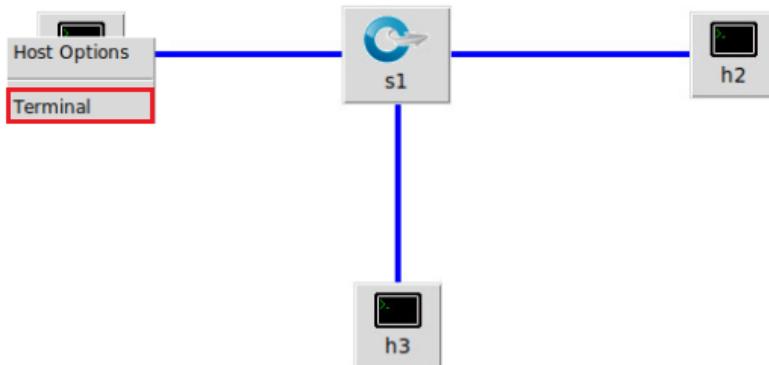


Figure 15. Opening a terminal on host h1.

**Step 2.** In host h1 terminal, type the following command to verify that the IP address was assigned successfully. You will verify the host interface, *h1-eth0* configured with the IP address 10.0.0.1 and the subnet mask 255.0.0.0. You will also verify the MAC address, 00:00:00:00:00:01.

```

ifconfig

```

```

root@admin:/home/ovs# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0
          ether 00:00:00:00:00:01 txqueuelen 1000  (Ethernet)
            RX packets 21 bytes 2910 (2.9 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 3 bytes 270 (270.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
          loop txqueuelen 1000  (Local Loopback)
            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

root@admin:/home/ovs#
  
```

Figure 16. Verifying IP and MAC address in the host.

**Step 3.** You can also repeat steps 1-2 to verify IP and MAC addresses in hosts h2 and h3. You will notice the MAC addresses of hosts h2 and h3 are 00:00:00:00:00:02 and 00:00:00:00:00:03, respectively.

## 4 Configuring stateful firewall in switch s1

In this section, you will configure switch s1 so that the switch blocks the traffic generated from host h3 (10.0.0.3) for the destination host h1 (10.0.0.1). Host h3 will be able to send reply if any traffic is generated from host h1.

**Step 1.** Type the following command to manually insert a flow into switch s1.

```
sudo ovs-ofctl add-flow s1 "table=0,action=normal"
```

```
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=0,action=normal"
ovs@admin:~$
```

Figure 17. Manually adding a flow entry.

Consider the figure above. A *normal* action allows the device to conduct normal layer 2/layer 3 packet processing like a traditional switch. The priority value for this entry is 32768 which is the default priority value in Open vSwitch.

**Step 2.** Type the following command to manually insert a flow into switch s1.

```
sudo ovs-ofctl add-flow s1
"table=0,priority=40000,ip,nw_src=10.0.0.1,nw_dst=10.0.0.3,ct_state=-trk,action=ct(table=1)"
```

```
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=0,priority=40000,ip,nw_src=10.0.0.1,
nw_dst=10.0.0.3,ct_state=-trk,action=ct(table=1)"
ovs@admin:~$
```

Figure 18. Manually adding a flow entry.

Consider the figure above. The `ct_state` parameter refers to the state of the conntrack. The `-trk` parameter is used so that if host h1 (10.0.0.1) generates traffic for the destination host h3 (10.0.0.3), conntrack will start tracking the packet, and the switch will check table 1 for further instruction. The flow entry has a higher priority value which is 40000.

**Step 3.** Type the following command to manually insert a flow into switch s1.

```
sudo ovs-ofctl add-flow s1
"table=0,priority=40000,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1,ct_state=-trk,action=ct(table=1)"
```

```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=0,priority=40000,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1,ct_state=-trk,action=ct(table=1)"
ovs@admin:~$
```

Figure 19. Manually adding a flow entry.

Consider the figure above. If host h3 (10.0.0.3) generates traffic for the destination host h1 (10.0.0.1), conntrack will start tracking the packet, and the switch will check table 1 for further instruction. The flow entry has a higher priority value which is 40000.

**Step 4.** Type the following command to manually insert a flow into switch s1.

```
sudo ovs-ofctl add-flow s1
"table=1,ip,nw_src=10.0.0.1,nw_dst=10.0.0.3,ct_state=+trk+new,action=ct(commit)
,3"
```

```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=1,ip,nw_src=10.0.0.1,nw_dst=10.0.0.3,ct_state=+trk+new,action=ct(commit),3"
ovs@admin:~$
```

Figure 20. Manually adding a flow entry.

Consider the figure above. If host h1 generates traffic for destination host h3, switch s1 will forward the traffic to port 3, (s1-eth3). If any host generates a new packet for any destination host, the traffic is considered as new. The `ct` action sends the packet through the connection tracker. The `commit` parameter is used so that conntrack module will store information for a certain time beyond the lifetime of packets. If host h1 wants to send data to destination host h3, the traffic will be monitored by conntrack, and the information will be stored.

**Step 5.** Type the following command to manually insert a flow into switch s1.

```
sudo ovs-ofctl add-flow s1
"table=1,ip,nw_src=10.0.0.1,nw_dst=10.0.0.3,ct_state=+trk+est,action=output:3"
```

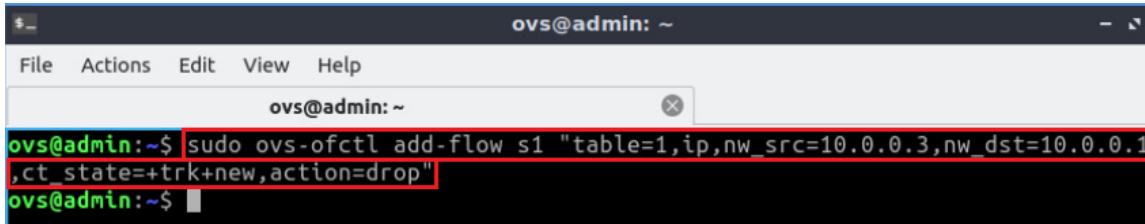
```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=1,ip,nw_src=10.0.0.1,nw_dst=10.0.0.3,ct_state=+trk+est,action=output:3"
ovs@admin:~$
```

Figure 21. Manually adding a flow entry.

Consider the figure above. If host h1 generates reply traffic for destination host h3, switch s1 will forward the traffic to port 3, (s1-eth3). If any host generates a reply packet for any destination host, the traffic is considered as `est` (established).

**Step 6.** Type the following command to manually insert a flow into switch s1.

```
sudo ovs-ofctl add-flow s1  
"table=1,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1,ct_state=+trk+new,action=drop"
```



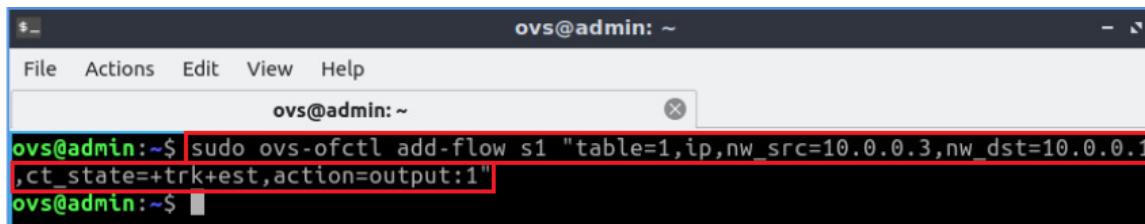
```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=1,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1,  
,ct_state=+trk+new,action=drop"  
ovs@admin:~$
```

Figure 22. Manually adding a flow entry.

Consider the figure above. If host h3 generates new traffic for destination host h1, switch s1 will drop the traffic.

**Step 7.** Type the following command to manually insert a flow into switch s1.

```
sudo ovs-ofctl add-flow s1  
"table=1,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1,ct_state=+trk+est,action=output:1"
```



```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 "table=1,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1,  
,ct_state=+trk+est,action=output:1"  
ovs@admin:~$
```

Figure 23. Manually adding a flow entry.

Consider the figure above. If host h3 generates reply traffic for destination host h1, switch s1 will forward the traffic to port 1, (s1-eth1).

## 5 Verifying configuration

In this section, you will verify the firewall configuration.

**Step 1.** Type the following command to verify the flow installation. This command prints the flow table entries in switch s1.

```
sudo ovs-ofctl dump-flows s1
```

```
ovs@admin:~$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=7231.410s, table=0, n_packets=3, n_bytes=294, priority=40000,
ct_state=-trk,ip,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=ct(table=1)
cookie=0x0, duration=7139.884s, table=0, n_packets=4, n_bytes=392, priority=40000,
ct_state=-trk,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=ct(table=1)
cookie=0x0, duration=7255.959s, table=0, n_packets=20, n_bytes=1288, actions=NORMA
L
cookie=0x0, duration=4308.732s, table=1, n_packets=1, n_bytes=98, ct_state=+new+tr
k,ip,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=ct(commit),output:"s1-eth3"
cookie=0x0, duration=4289.640s, table=1, n_packets=1, n_bytes=98, ct_state=+new+tr
k,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=drop
cookie=0x0, duration=4296.857s, table=1, n_packets=2, n_bytes=196, ct_state=+est+tr
k,ip,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=output:"s1-eth3"
cookie=0x0, duration=4281.844s, table=1, n_packets=3, n_bytes=294, ct_state=+est+tr
k,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=output:"s1-eth1"
ovs@admin:~$
```

Figure 24. Verifying flows in switch s1.

Consider the figure above. You will notice all the flows added in switch s1.

**Step 2.** In h1 terminal, test the connectivity between host h1 and host h2 using the `ping` command.

```
ping 10.0.0.2
```

```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.594 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.068 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.079 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.082 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3050ms
rtt min/avg/max/mdev = 0.068/0.205/0.594/0.224 ms
root@admin:/home/ovs#
```

Figure 25. Output of `ping` command.

The result in the figure above shows a successful connectivity between the hosts. To stop the test, press `Ctrl+c`.

**Step 3.** In h1 terminal, test the connectivity between host h1 and host h3 using the `ping` command.

```
ping 10.0.0.3
```

```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.810 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.332 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.086 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.081 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3064ms
rtt min/avg/max/mdev = 0.081/0.327/0.810/0.296 ms
root@admin:/home/ovs#
```

Figure 26. Output of `ping` command.

The result in the figure above shows a successful connectivity between the hosts. To stop the test, press `Ctrl+c`.

**Step 4.** Type the following command to show conntrack state table for the IP address 10.0.0.1.

```
sudo ovs-dpctl dump-conntrack | grep "10.0.0.1"
```

```
ovs@admin:~$ sudo ovs-dpctl dump-conntrack | grep "10.0.0.1"
icmp,orig=(src=10.0.0.1,dst=10.0.0.3,id=41501,type=8,code=0),reply=(src=10.0.0.3,dst=10.0.0.1,id=41501,type=0,code=0)
ovs@admin:~$
```

Figure 27. Displaying conntrack state table.

Consider the figure above. The figure shows the state of conntrack state table. Source host h1 generated a ping request (ICMP packet) for the destination host h3, 10.0.0.3 where id=41501, host h1 receives a reply from the destination host h3 using the same id (41501).

Verify conntrack state table right after pinging since the table information is deleted after a certain period of time. If you do not get expected result, run the test again and see the conntrack table.

**Step 5.** In host h3 terminal, test the connectivity between host h3 and host h1 using the `ping` command.

```
ping 10.0.0.1
```

```
"Host: h3"
root@admin:/home/ovs# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
^C
--- 10.0.0.1 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3052ms
root@admin:/home/ovs#
```

Figure 28. Output of `ping` command.

To stop the test, press `Ctrl+C`.

Consider the figure above. Host h3 cannot communicate with host h1 since switch s1 drops incoming traffic generated from host h3 for destination host h1.

## 6 Monitoring conntrack state table

In this section, you will run a TCP test and verify how conntrack monitors each flow and store the information in the state table.

**Step 1.** Go to file option and open a new terminal tab (showed in the following figure) or press `Ctrl+Shift+T`.

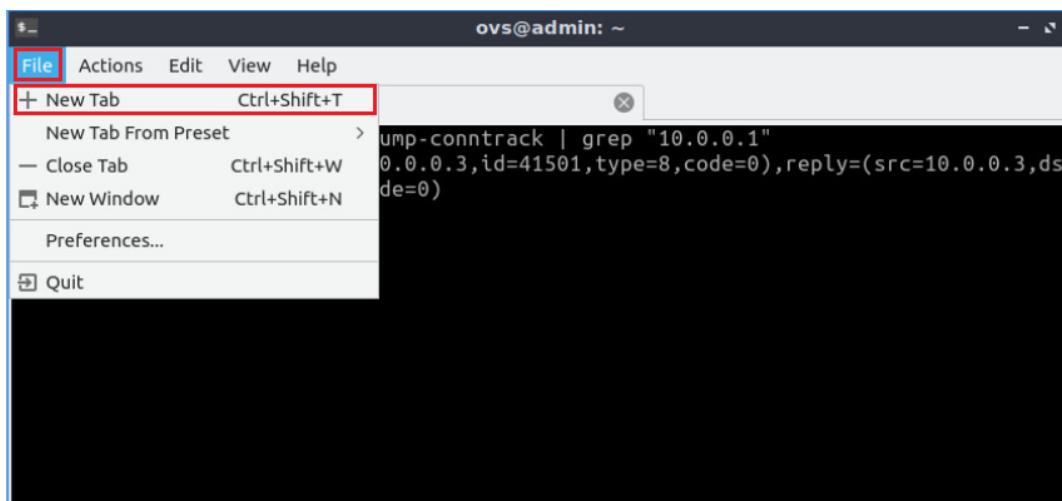


Figure 29. Opening a new terminal.

**Step 2.** Type the following command to show conntrack event updates for IP address 10.0.0.1. When prompted for a password, type `password`.

```
sudo conntrack -E | grep "10.0.0.1"
```

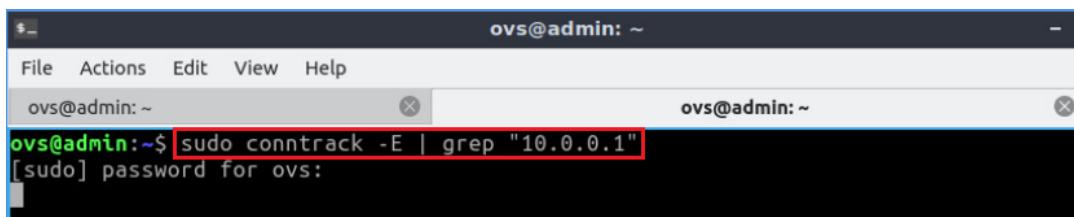
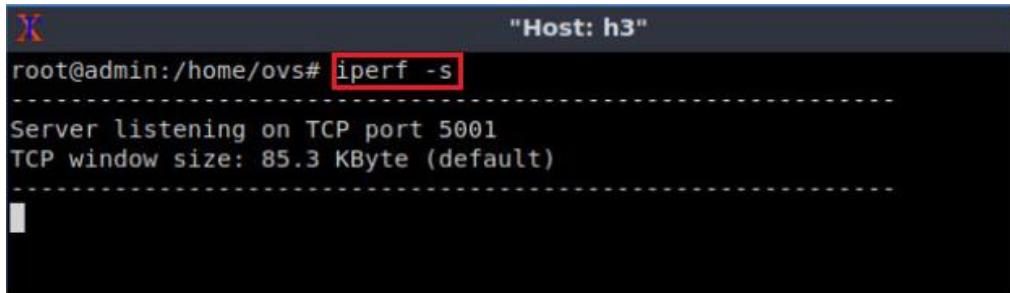


Figure 30. Displaying conntrack events.

Consider the figure above. Currently, there is no event for the IP address 10.0.0.1.

**Step 3.** In h3 terminal, type the following command to run the host in server mode.

```
iperf -s
```



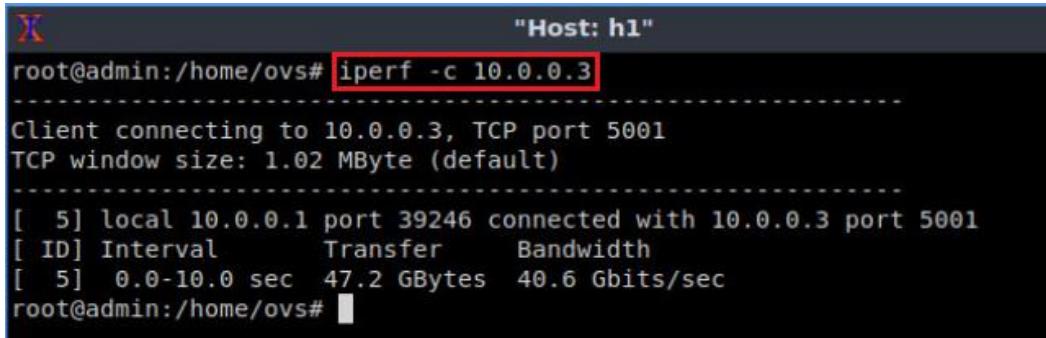
The terminal window shows the command "iperf -s" being run by root at host h3. The output indicates that the server is listening on TCP port 5001 with a default TCP window size of 85.3 KByte.

Figure 31. Running host h2 in server mode.

Consider the figure above. The figure shows that host h3 is acting as a server and listening to TCP port 5001.

**Step 4.** In h1 terminal, type the following command to run the host in client mode and run a TCP test between hosts h1 and h3.

```
iperf -c 10.0.0.3
```



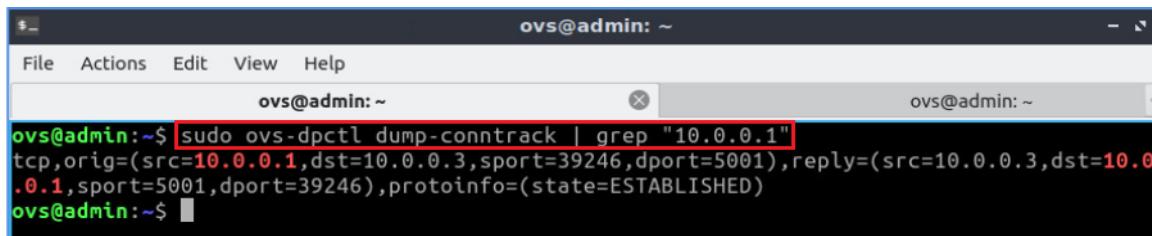
The terminal window shows the command "iperf -c 10.0.0.3" being run by root at host h1. The output shows a client connecting to host h3 (IP 10.0.0.3) on port 5001. The test results show a bandwidth of 40.6 Gbits/sec over a 10.0 sec interval.

Figure 32. Running host h1 in client mode.

Consider the figure above. The figure shows that host h1 is acting as a client and a TCP test has been done between the client and the server.

**Step 5.** Go back to the Linux terminal. Type the following command immediately after running the TCP test to show the state of the conntrack.

```
sudo ovs-dpctl dump-conntrack | grep "10.0.0.1"
```



```
ovs@admin:~$ sudo ovs-dpctl dump-conntrack | grep "10.0.0.1"
tcp,orig=(src=10.0.0.1,dst=10.0.0.3,sport=39246,dport=5001),reply=(src=10.0.0.3,dst=10.0.0.1,sport=5001,dport=39246),protoinfo=(state=ESTABLISHED)
ovs@admin:~$
```

Figure 33. Displaying conntrack state table.

Consider the figure above. The figure shows the TCP test. You will notice that source h1 generates the TCP session using source port 39246 and destination port 5001. This flow will be stored. Whenever host h3 is sending a reply to host h1 using source port 5001 and destination port 39246, switch s1 will allow the traffic since the source, and destination ports are the same.

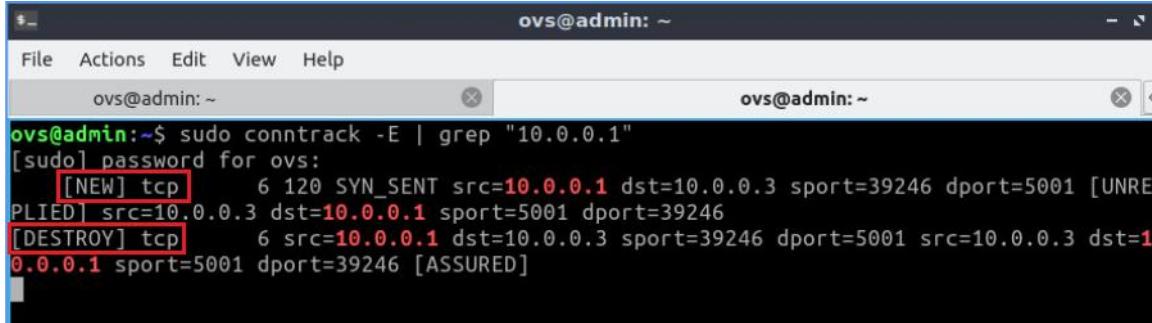
You might notice different port number since the port numbers are generated randomly.

You need to verify the state table immediately after running the TCP test to see the ESTABLISHED connection. Otherwise, the established connection will be changed to TIME\_WAIT if the TCP session is complete.

If you encounter this, you can rerun the TCP test and see the results.

**Step 6.** In host h3 terminal, press **Ctrl+c** to stop the TCP session.

**Step 7.** Go to another terminal to verify the conntrack events for the IP address 10.0.0.1.



```
ovs@admin:~$ sudo conntrack -E | grep "10.0.0.1"
[sudo] password for ovs:
[NEW] tcp      6 120 SYN_SENT src=10.0.0.1 dst=10.0.0.3 sport=39246 dport=5001 [UNREPLIED]
[DESTROY] tcp    6 src=10.0.0.1 dst=10.0.0.3 sport=39246 dport=5001 src=10.0.0.3 dst=10.0.0.1 sport=5001 dport=39246 [ASSURED]
```

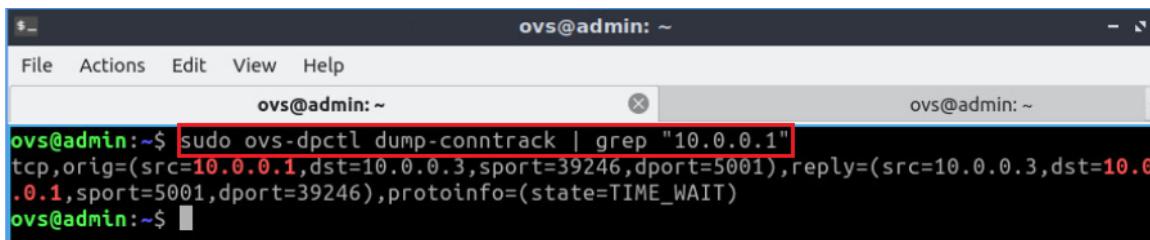
Figure 34. Displaying conntrack events.

Consider the figure above. The figure shows that a new TCP session was generated. **Unreplied** refers to nonpersistent flow. Once the session is stopped, the flow will be destroyed. You will notice that the session has been destroyed at the end.

The destroy session may take a bit longer (2-3 minutes) to appear.

**Step 8.** Type the following command to show the state of the conntrack.

```
sudo ovs-dpctl dump-conntrack | grep "10.0.0.1"
```



```
ovs@admin:~$ sudo ovs-dpctl dump-conntrack | grep "10.0.0.1"
tcp,orig=(src=10.0.0.1,dst=10.0.0.3,sport=39246,dport=5001),reply=(src=10.0.0.3,dst=10.0.0.1,sport=5001,dport=39246),protoinfo=(state=TIME_WAIT)
ovs@admin:~$
```

Figure 35. Displaying conntrack state table.

Consider the figure above. Once the session is completed, the conntrack state will be changed. The figure shows that the state has been changed to `TIME_WAIT` (ending of a TCP session).

This concludes Lab 8. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Martinez, J. Cabot, “Management of statefull firewall misconfiguration”, 2013.
2. M. Gouda, A. Liu, “A model of stateful firewalls and its properties”, 2005.
3. Cisco, “The Cisco learning network”, [Online]. Available: <https://learningnetwork.cisco.com/s/question/0D53i00000Ksup8/stateful-firewall-overview>
4. Red Hat, “How connection tracking in Open vSwitch helps OpenStack performance”, July 2016.
5. ScienceDirect “Stateful inspection”, 2020. [Online]. Available: <https://www.sciencedirect.com/topics/computer-science/stateful-inspection#:~:text=However%20a%20stateful%20firewall%20also,ends%2C%20that%20opening%20is%20closed>.
6. J. Ellingwood, “A deep dive into iptables and netfilter architecture”, Aug 2015.
7. Linux foundation, “Open vSwitch”, [Online]. Available: <http://openvSwitch.org>.
8. Mininet walkthrough, [Online]. Available: <http://mininet.org>.



## OPEN VIRTUAL SWITCH

### **Exercise 3: Configuring Stateless and Stateful Firewalls in Open vSwitch**

Document Version: **09-23-2021**



Award 1829698  
“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput Networks for Big Science Data Transfers”

## Contents

1	Exercise topology .....	3
1.1	Topology settings .....	3
1.2	Credentials .....	3
2	Deliverables.....	4

## 1 Exercise topology

Consider Figure 1. The topology consists of four hosts and two switches. All the hosts belong to the same network, 10.0.0.0/8.

The goal of this lab is to implement stateless and stateful firewalls in switch s1.

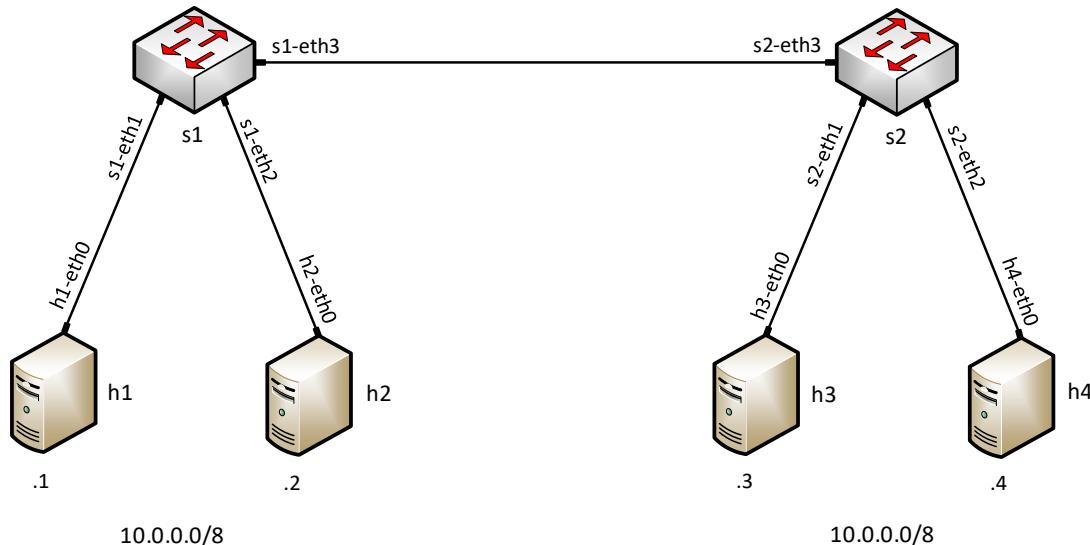


Figure 1. Exercise topology.

### 1.1 Topology settings

The devices are already configured according to Table 1.

Table 1. Topology information.

Host	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8
h3	h3-eth0	10.0.0.3	/8
h4	h4-eth0	10.0.0.4	/8

### 1.2 Credentials

The information in Table 2 provides the credentials to access the Client's virtual machine.

Table 2. Credentials to access the Client's virtual machine.

Device	Account	Password

Client	admin	password
--------	-------	----------

## 2 Deliverables

Follow the steps below to complete the exercise.

- a) Start MiniEdit by clicking on MiniEdit's shortcut. Load the topology *Exercise3.mn* located at *~/OVS\_Labs/Exercise3* as shown in the figure below.

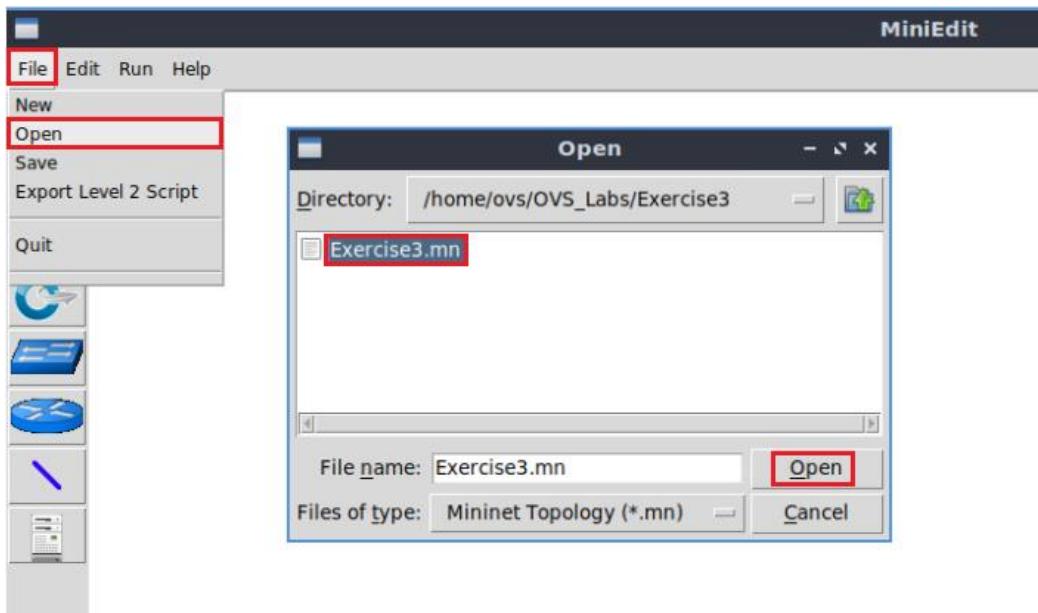


Figure 2. Loading the topology file in Mininet.

- b) Run the emulation in Mininet.
- c) In the Mininet terminal, launch the command that displays the interface names and connections of the current topology. Verify that links conform to the topology in Figure 1.
- d) Enable the traditional switch forwarding operation in switch s2.
- e) Configure table 0 in switch s1.
- Enable the traditional switch forwarding operation.
  - Add a flow entry with a higher priority to block traffic coming from host h4 to h2.
- f) Verify the stateless firewall configuration using `ping` and `ovs-appctl` commands. Explain the result. What happens when host h4 generates traffic for the destination host h2? Do you see successful connectivity between other hosts?
- g) Configure table 0 in switch s1 for stateful firewall. Add flow entries with higher priorities and start tracking packets where source and destination IP addresses are

10.0.0.1 and 10.0.0.3 and vice versa. The switch will be instructed to check table 1 for further instructions.

**h)** Configure table 1 in switch s1 so that new traffic generated from host h3 for the destination host h1 will be blocked. Host h3 will be able to send reply if any traffic is generated from host h1.

**i)** Verify the stateful firewall configuration using `ping` command. Explain the result. What happens when host h1 generates traffic for the destination host h3? What happens when host h3 generates traffic for the destination host h1?



## OPEN VIRTUAL SWITCH

### Lab 9: Configuring Quality of Service (QoS) in Open vSwitch

Document Version: **07-22-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    Introduction to QoS.....	4
1.2    Hierarchical Token Bucket (HTB) algorithm .....	4
1.3    QoS shaping and policing .....	5
1.4    QoS metering .....	6
2    Lab topology.....	6
2.1    Lab settings.....	6
2.2    Loading a topology .....	7
2.3    Verifying default traffic rate.....	9
3    Configuring QoS policing in switch s1.....	10
4    Configuring Single-Rate Two-Colors Traffic Policing in switch s1.....	13
5    Verifying metering configuration.....	14
6    Configuring QoS shaping in switch s1.....	17
7    Verifying QoS shaping configuration .....	19
7.1    Verifying QoS and traffic rate for individual hosts.....	19
7.2    Verifying traffic rate for competing hosts.....	22
References .....	24

## Overview

This lab introduces Quality of Service (QoS), a set of protocols aimed at manipulating the traffic, such that network devices forward it according to the application's requirement (e.g., low latency, low loss rate, bounded jitter, and guaranteed bandwidth). The lab's focus is to configure QoS in Open Virtual Switch (Open vSwitch), where the switch will establish the bandwidth allocation for each flow.

## Objectives

By the end of this lab, you should be able to:

1. Understand the concept of QoS.
2. Explore QoS mechanisms.
3. Understand how QoS works in Open vSwitch.
4. Configure and verify the QoS shaping method in Open vSwitch to limit the traffic flow.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Configuring QoS policing in switch s1.
4. Section 4: Configuring Single-Rate Two-Colors Traffic Policing in switch s1.
5. Section 5: Verifying metering configuration.
6. Section 6: Configuring QoS shaping in switch s1.
7. Section 7: Verifying QoS shaping configuration.

## 1      Introduction

When a router receives an IP packet, it looks for the destination in the routing table and forwards the packet toward the destination. All the packets are served in the same order as they arrive in the queue. This method is known as First-In-First-Out (FIFO). When data and voice traffic are transmitted simultaneously, the router will enqueue packets waiting to be transmitted. But the queue is limited. Once the queue is full, the router will start dropping packets<sup>1</sup>.

## 1.1 Introduction to QoS

QoS manages network resources by prioritizing specific types of data to maximize the experience of end-users. In a switched packet network, all the traffic is treated in the same way independently of the services that they are carrying. Using QoS, we can define how forwarding devices can manage different types of packets. QoS allows organizations to use their existing bandwidths more efficiently<sup>2</sup>.

QoS prioritizes packets in a way that ensures that the bandwidth meets the traffic requirements<sup>3</sup>. For instance, packets belonging to a video call would be prioritized over packets belonging to an email or a file download because a video call is a more synchronous form of communication and needs to happen in real-time, whereas emailing is not necessarily time-sensitive. If packets are dropped or delayed during a video chat, the end-user may experience jitter or latency. On the other hand, if packets are delayed in the emailing process, they can still be sent after, and the end-user will not experience any lapse in service. The QoS classifier reads the packet header to prioritize packets and determines that a packet is related to video streaming and prioritizes it over less time-sensitive packets. QoS allows organizations to use their existing bandwidths more efficiently.

## 1.2 Hierarchical Token Bucket (HTB) algorithm

Consider Figure 1. The HTB algorithm can control the outbound bandwidth on a given link by defining several slower links. The user specifies how to slice the physical link and defines the bandwidth for each of the slices. HTB shapes traffic using the Token Bucket Filter (TBF) algorithm, which does not depend on interface characteristics and does not need to know the underlying bandwidth of the outgoing interface.

The classes are configured as a tree according to relationships of traffic aggregations. Only leaf classes have a queue to buffer the packets that belong to the class. Children's classes borrow bandwidth from their parents when the configured rate is exceeded. A child will continue to attempt to borrow bandwidth until it reaches ceil, which is the maximum bandwidth available for that class. Under each class, the user can specify other queueing disciplines, namely Token Bucket Filter (TBF), Stochastic Fair Queuing (SFQ), Controlled Delay (CoDel), etc. It will depend on the service that such class is intended to provide. If there is no queueing discipline, HTB sets First-in, First-out (FIFO) as the default queueing discipline.

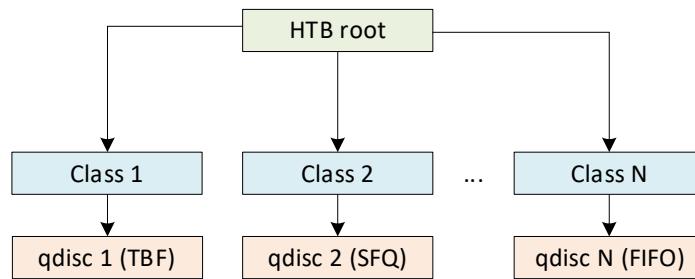


Figure 1. Hierarchical Token Bucket Structure.

### 1.3 QoS shaping and policing

Open vSwitch supports traffic policing and shaping. Policing is a method that drops packets received more than the configured rate. Shaping is a technique that allows higher-priority traffic to flow at an optimal level even when the link is overutilized. Shaping is used for outbound traffic where policing is required to apply for inbound traffic<sup>4</sup>. Egress shaping limits the rate at which traffic is allowed to transmit from a physical interface. On the other hand, ingress policing limits the rate at which traffic will be received on a switch.

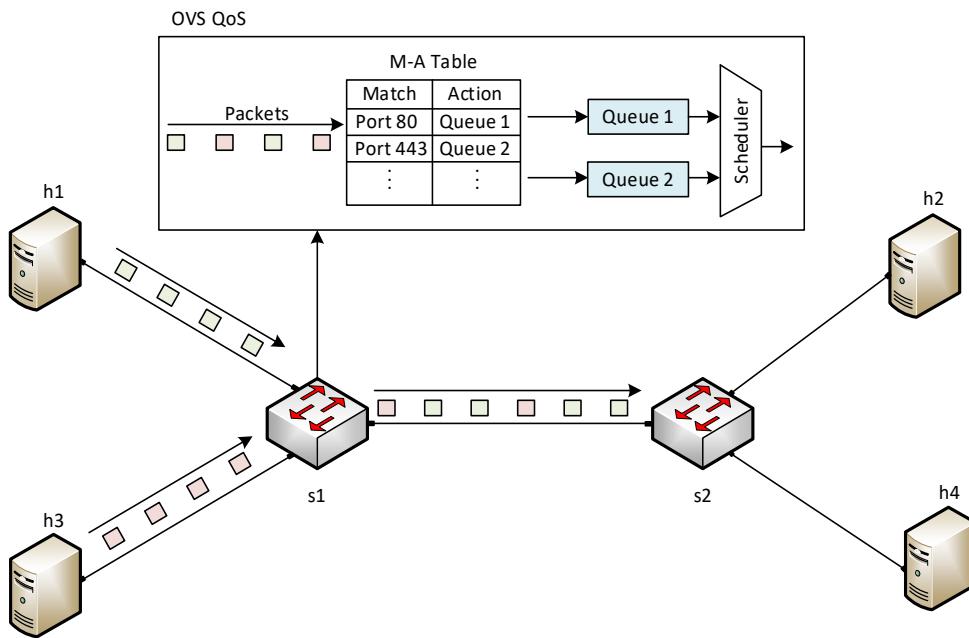


Figure 2. QoS queues for different data types.

Consider Figure 2. The QoS shaping is implemented by separating in different queues. Once the corresponding type of traffic occupies the queues, a scheduler will decide which order and rate the packets are sent. In this example, Qos is enabled within switch s1, and traffic is matching against port numbers. Whenever the port number is 80, instructions associated with Queue 1 will be executed.

## 1.4 QoS metering

Metering and color marking is another tool which is used in QoS policing to increase granularity. The tool allows to measure the traffic arrival rate and assigns different colors to the traffic according to the rate. Metering compares the actual rate of the traffic with Committed information rate (CIR) which is the guaranteed rate and Peak information rate (PIR) which is the maximum allowed traffic rate<sup>7</sup>. It measures the traffic comparing these two values and marks the traffic with colors that identify whether the traffic is *in-contract* or *out-of-contract*:

- Green: Traffic rate is below the CIR and is *in-contract*.
- Yellow: Traffic rate falls between CIR and PIR and is *out-of-contract*.
- Red: Traffic is above PIR and is *out-of-contract*.

Though Open vSwitch supports three color marking, our version of Open vSwitch only supports two color marking (green and red).

## 2 Lab topology

Consider Figure 3. The topology consists of two switches and four end-hosts. Hosts h2 and h4 are acting as File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP) servers, whereas hosts h1 and h3 are competing clients.

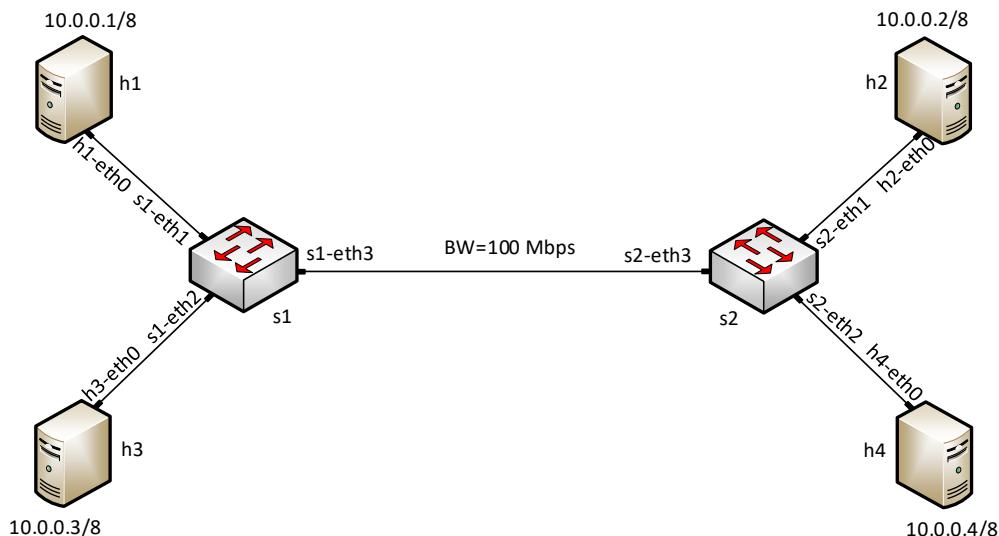


Figure 3. Lab topology.

## 2.1 Lab settings

The hosts are configured according to Table 2.

Table 2. Topology information.

Hosts	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8
h3	h3-eth0	10.0.0.3	/8
h4	h4-eth0	10.0.0.4	/8

## 2.2 Loading a topology

**Step 1.** Start by launching MiniEdit by clicking on the desktop's shortcut. When prompted for a password, type `password`.

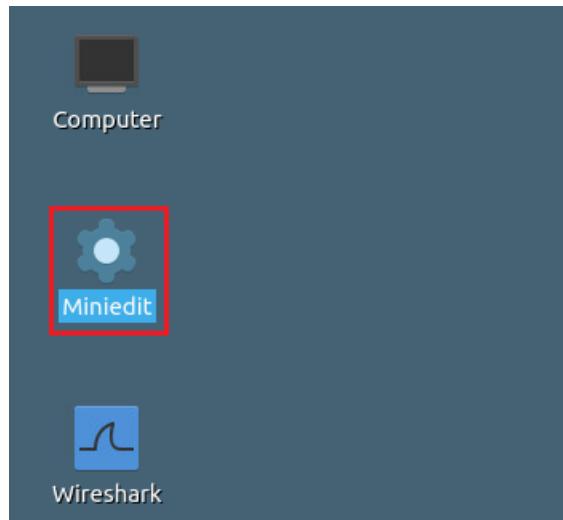


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File*, then *open* to load the lab's topology. Locate the *Lab9.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab9* and click on *Open*.

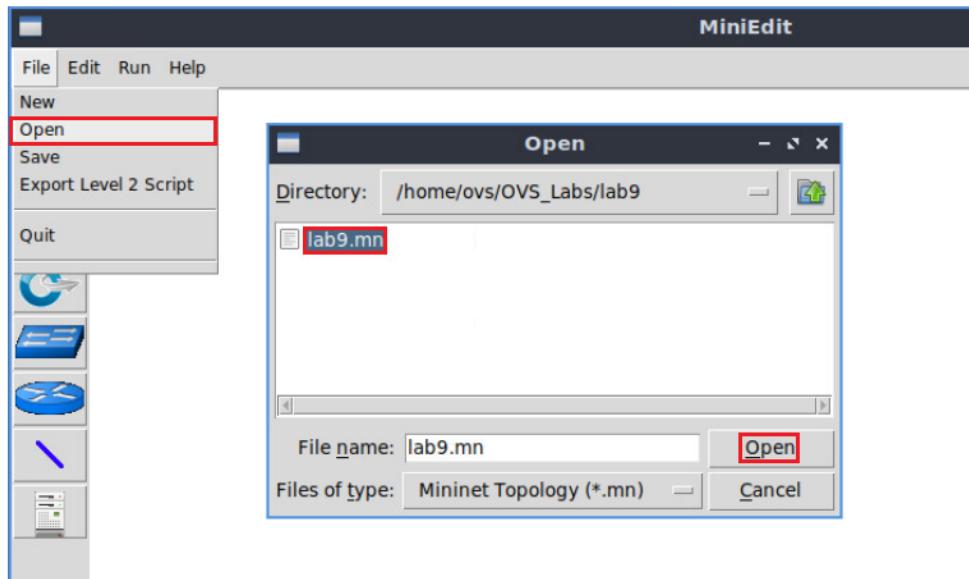


Figure 5. MiniEdit's open dialog.

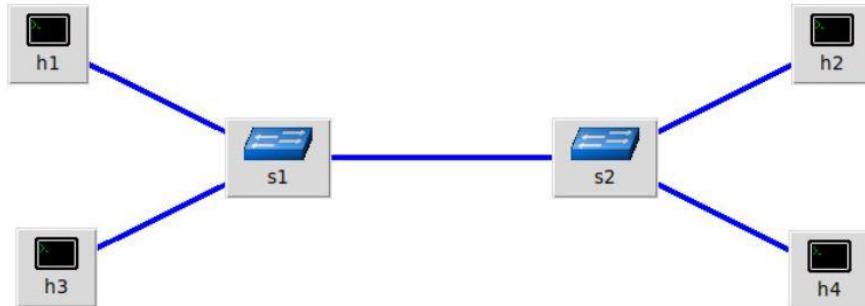


Figure 6. MiniEdit's topology.

**Step 3.** Click on the *Run* button to start the emulation. The emulation will start, and the MiniEdit panel buttons will gray out, indicating that they are currently disabled.



Figure 7. Starting the emulation.

**Step 4.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

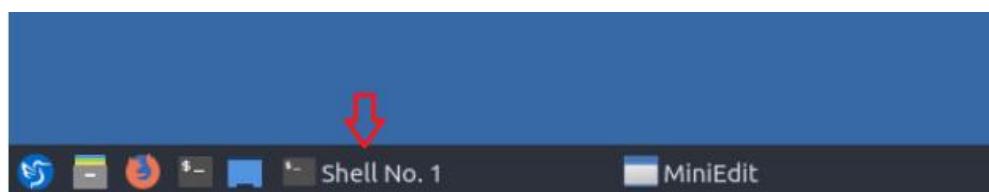


Figure 8. Opening Mininet's terminal.

**Step 5.** Issue the following command to display the interface names and connections.

```
links
```

```
containernet> links
h1-eth0<->s1-eth1 (OK OK)
h3-eth0<->s1-eth2 (OK OK)
h2-eth0<->s2-eth1 (OK OK)
h4-eth0<->s2-eth2 (OK OK)
s1-eth3<->s2-eth3 (OK OK)
containernet>
```

Figure 9. Displaying network interfaces.

In Figure 9, the link displayed within the gray box indicates that interface *eth0* of host *h1* connects to interface *eth1* of switch *s1* (i.e., *h1-eth0<->s1-eth1*).

## 2.3 Verifying default traffic rate

**Step 1.** To open the host *h2* terminal, hold right-click on host *h2* and select *Terminal*.

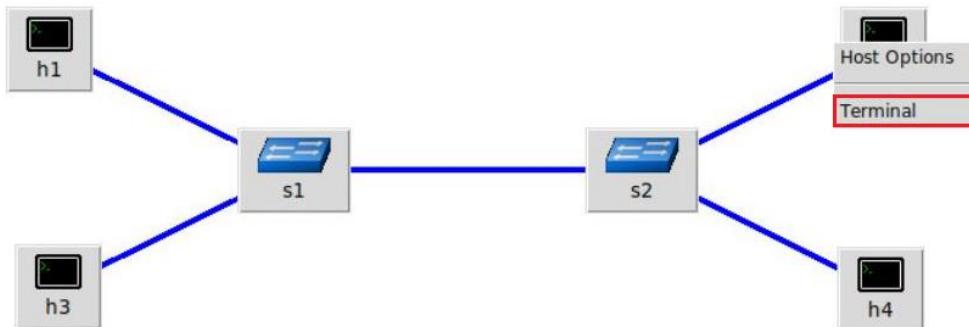


Figure 10. Opening a terminal on host h2.

**Step 2.** In the host *h2* terminal, type the following command to run the host as an FTP server using port 21.

```
iperf3 -s -p 21
```

```
"Host: h2"
root@admin:/home/ovs# iperf3 -s -p 21
-----
Server listening on 21
-----
```

Figure 11. Running host h2 in server mode.

**Step 3.** In order to open host *h1* terminal, hold right-click on host *h1* and select *Terminal*.

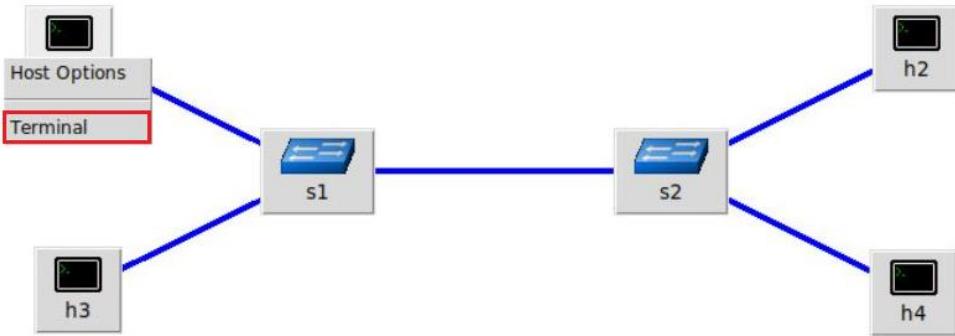


Figure 12. Opening a terminal on host h1.

**Step 4.** In the host h1 terminal, type the following command to run the host in client mode and run a throughput test between hosts h1 and h2.

```
iperf3 -c 10.0.0.2 -p 21
```

A terminal window titled "Host: h1" displays the output of an iperf3 test. The command entered was "iperf3 -c 10.0.0.2 -p 21". The output shows multiple data transfer intervals (ID 7) from host 10.0.0.1 to host 10.0.0.2. The throughput is consistently around 43.9 Gbytes/sec. The test ends with "iperf Done.".

ID	Interval	Transfer	Bitrate	Retr	Cwnd
7	0.00-1.00	4.97 GBytes	42.7 Gbits/sec	0	1017 KBytes
7	1.00-2.00	5.07 GBytes	43.5 Gbits/sec	0	1.15 MBytes
7	2.00-3.00	5.24 GBytes	45.0 Gbits/sec	0	1.90 MBytes
7	3.00-4.00	5.01 GBytes	43.0 Gbits/sec	0	2.09 MBytes
7	4.00-5.00	4.97 GBytes	42.7 Gbits/sec	0	2.31 MBytes
7	9.00-10.00	5.29 GBytes	45.5 Gbits/sec	0	2.47 MBytes

ID	Interval	Transfer	Bitrate	Retr	sender	receive
7	0.00-10.00	51.2 GBytes	43.9 Gbits/sec	0		
7	0.00-10.00	51.2 GBytes	43.9 Gbits/sec			

Figure 13. Running throughput test between hosts h1 and h2.

Consider the figure above. By default, the bitrate for the test is 43.9 Gbps.

The throughput can vary depending on the resources assigned to the Virtual Machine (VM).

### 3 Configuring QoS policing in switch s1

**Step 1.** Click on the icon below to open the Linux terminal.

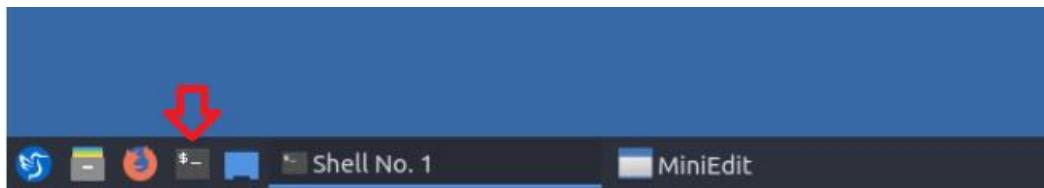


Figure 14. Opening Linux terminal.

**Step 2.** Type the following command to set the ingress policy rate for the interface `s1-eth1` to 10 megabits per second (Mbps) which means the maximum incoming rate is 10 Mbps. If the password is required, type `password`.

```
sudo ovs-vsctl set interface s1-eth1 ingress_policing_rate=10000
```

A screenshot of a terminal window titled "ovs@admin: ~". The window shows a command being run: `sudo ovs-vsctl set interface s1-eth1 ingress_policing_rate=10000`. A red box highlights this command. Below it, a message asks for a password: "[sudo] password for ovs:". The command prompt ends with a red box around the closing bracket.

Figure 15. Configuring QoS policing.

**Step 3.** In the host `h1` terminal, type the following command to run a throughput test between hosts `h1` and `h2`.

```
iperf3 -c 10.0.0.2 -p 21
```

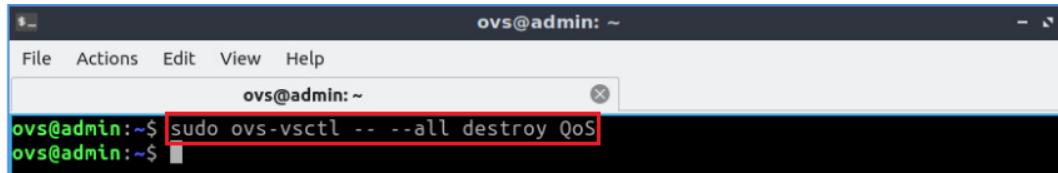
A screenshot of a terminal window titled "Host: h1". The window shows the output of an `iperf3` command: `iperf3 -c 10.0.0.2 -p 21`. The output displays a series of data transfer intervals with their corresponding transfer times, bitrates, and other metrics. A red box highlights the command at the top. Another red box highlights the final summary line where the word "sender" is followed by "receive".

Figure 16. Running throughput test between hosts h1 and h2.

The figure above shows that the bitrate is 10.6 Mbps.

**Step 4.** Type the following command to delete the QoS policy. This command is responsible for deleting all the running QoS configurations.

```
sudo ovs-vsctl -- --all destroy QoS
```

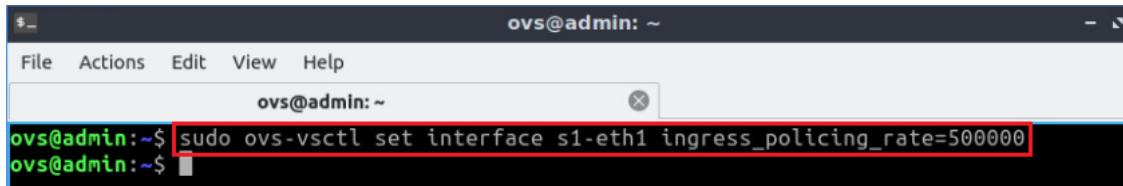


```
ovs@admin:~$ sudo ovs-vsctl -- --all destroy QoS
```

Figure 17. Deleting all QoS.

**Step 5.** Type the following command to set the ingress policy rate for the interface s1-eth1 to 500Mb.

```
sudo ovs-vsctl set interface s1-eth1 ingress_policing_rate=500000
```

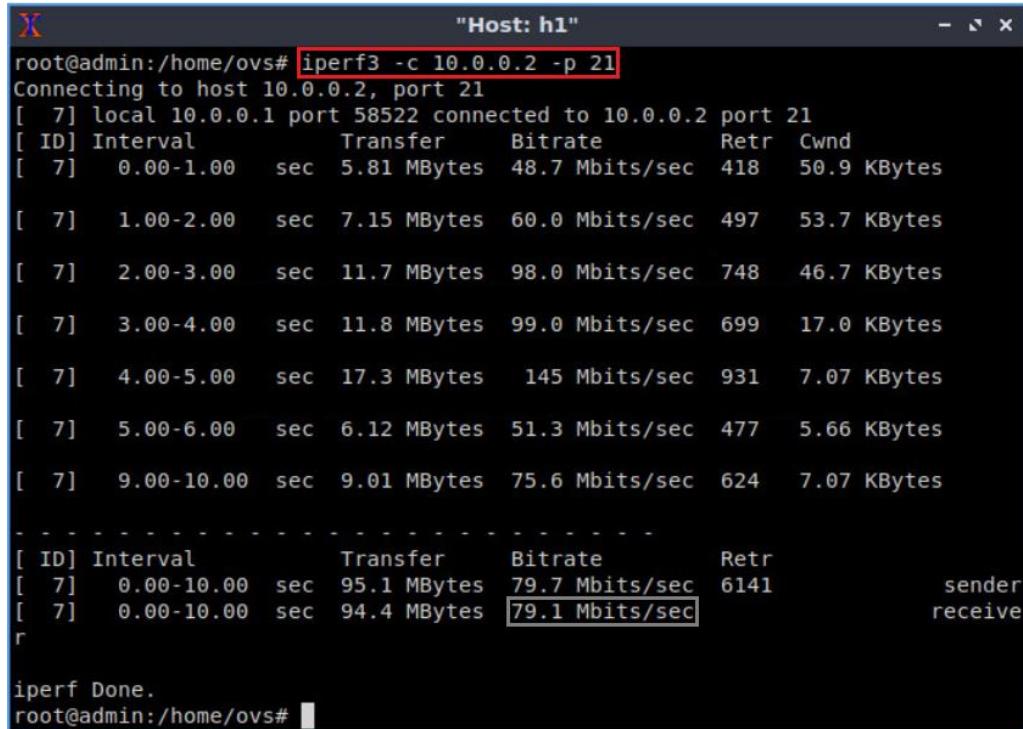


```
ovs@admin:~$ sudo ovs-vsctl set interface s1-eth1 ingress_policing_rate=500000
```

Figure 18. Configuring QoS policing.

**Step 6.** In the host h1 terminal, type the following command to run a throughput test between hosts h1 and h2.

```
iperf3 -c 10.0.0.2 -p 21
```



```
root@admin:/home/ovs# iperf3 -c 10.0.0.2 -p 21
Connecting to host 10.0.0.2, port 21
[ 7] local 10.0.0.1 port 58522 connected to 10.0.0.2 port 21
[ ID] Interval Transfer Bitrate Retr Cwnd
[ 7] 0.00-1.00 sec 5.81 MBytes 48.7 Mbits/sec 418 50.9 KBytes

[ 7] 1.00-2.00 sec 7.15 MBytes 60.0 Mbits/sec 497 53.7 KBytes
[ 7] 2.00-3.00 sec 11.7 MBytes 98.0 Mbits/sec 748 46.7 KBytes
[ 7] 3.00-4.00 sec 11.8 MBytes 99.0 Mbits/sec 699 17.0 KBytes
[ 7] 4.00-5.00 sec 17.3 MBytes 145 Mbits/sec 931 7.07 KBytes
[ 7] 5.00-6.00 sec 6.12 MBytes 51.3 Mbits/sec 477 5.66 KBytes
[ 7] 9.00-10.00 sec 9.01 MBytes 75.6 Mbits/sec 624 7.07 KBytes

[ ID] Interval Transfer Bitrate Retr
[ 7] 0.00-10.00 sec 95.1 MBytes 79.7 Mbits/sec 6141
[ 7] 0.00-10.00 sec 94.4 MBytes 79.1 Mbits/sec

iperf Done.
root@admin:/home/ovs#
```

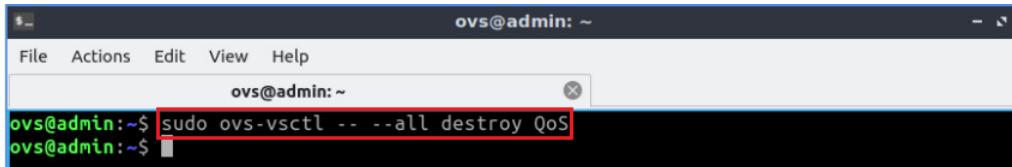
Figure 19. Running throughput test between hosts h1 and h2.

The figure above shows that the bitrate is 79.1 Mbps though the maximum rate is 500 Mbps. This happens because traffic policing interacts poorly with some network protocols

and can have surprising results<sup>4</sup>. This is a limitation of ingress policing and occurs mostly with higher rates like 500 Mbps.

**Step 7.** Type the following command to delete the QoS policy.

```
sudo ovs-vsctl -- --all destroy QoS
```



```
ovs@admin:~$ sudo ovs-vsctl -- --all destroy QoS
ovs@admin:~$
```

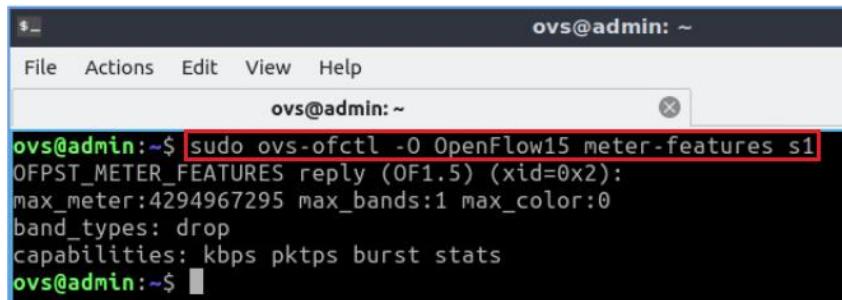
Figure 20. Deleting all QoS.

## 4 Configuring Single-Rate Two-Colors Traffic Policing in switch s1

In this section, you will configure single-rate two-colors traffic policing where the policer meters the traffic stream and classifies packets into two categories, green for conforming packets and red for nonconforming packets.

**Step 1.** Type the following command to show metering features supported by Open vSwitch. You will notice *band\_types: drop* which means it supports single rate two colors metering.

```
sudo ovs-ofctl -O OpenFlow15 meter-features s1
```

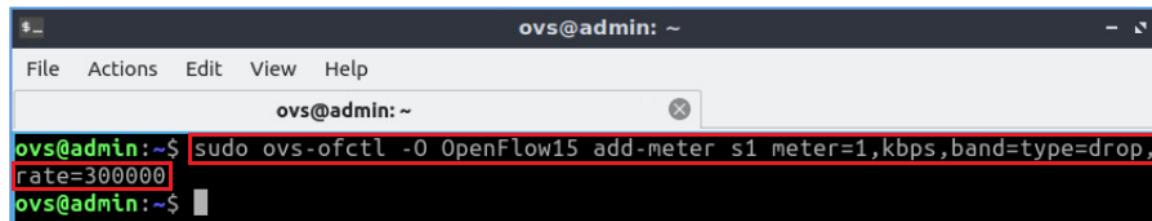


```
ovs@admin:~$ sudo ovs-ofctl -O OpenFlow15 meter-features s1
OFPST_METER_FEATURES reply (OF1.5) (xid=0x2):
max_meter:4294967295 max_bands:1 max_color:0
band_types: drop
capabilities: kbps pkts burst stats
ovs@admin:~$
```

Figure 21. Displaying Open vSwitch metering features.

**Step 2.** Type the following command to add a meter in switch s1 and limit the rate to 300 Mbps.

```
sudo ovs-ofctl -O OpenFlow15 add-meter s1
meter=1, kbps, band=type=drop, rate=300000
```

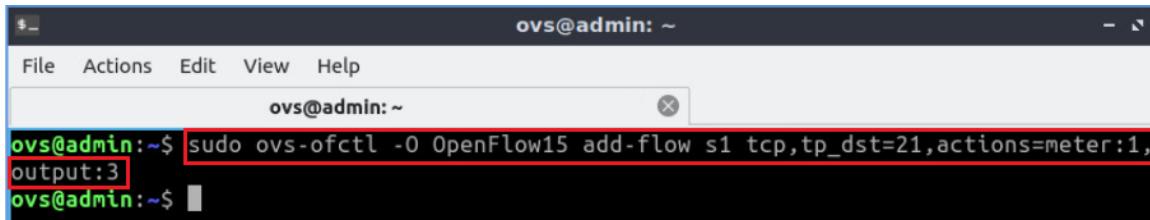


```
ovs@admin:~$ sudo ovs-ofctl -O OpenFlow15 add-meter s1 meter=1, kbps, band=type=drop,
rate=300000
ovs@admin:~$
```

Figure 22. Adding a meter policy to limit ingress rate.

**Step 3.** Type the following command to install a manual flow in switch s1.

```
sudo ovs-ofctl -O OpenFlow15 add-flow s1 tcp,tp_dst=21,actions=meter:1,output:3
```



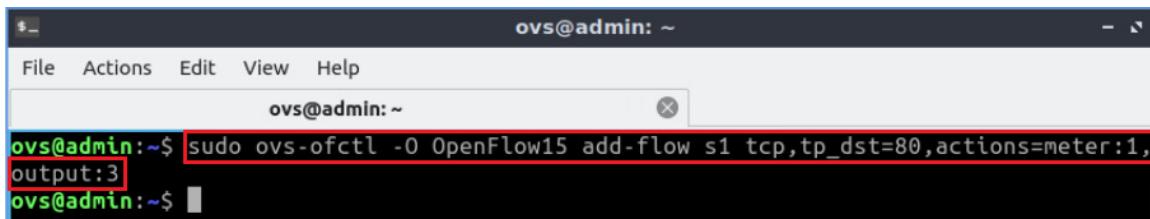
```
ovs@admin:~$ sudo ovs-ofctl -O OpenFlow15 add-flow s1 tcp,tp_dst=21,actions=meter:1,  
output:3  
ovs@admin:~$
```

Figure 23. Installing a manual flow entry.

Consider the figure above. The flow matches on the TCP port (21), meter is set to 1 and the output interface is *s1-eth3*. Whenever the flow matches on the TCP port, switch s1 will limit the rate to 300Mbps.

**Step 4.** Type the following command to install a manual flow in switch s1.

```
sudo ovs-ofctl -O OpenFlow15 add-flow s1 tcp,tp_dst=80,actions=meter:1,output:3
```



```
ovs@admin:~$ sudo ovs-ofctl -O OpenFlow15 add-flow s1 tcp,tp_dst=80,actions=meter:1,  
output:3  
ovs@admin:~$
```

Figure 24. Installing a manual flow entry.

Consider the figure above. The flow matches on the TCP port (80), meter is set to 1 and the output interface is *s1-eth3*.

Since one meter is applied to multiple flow entries, all the flow entries will share the meter rate.

## 5 Verifying metering configuration

**Step 1.** To open the host h4 terminal, hold right-click on host h4 and select Terminal.

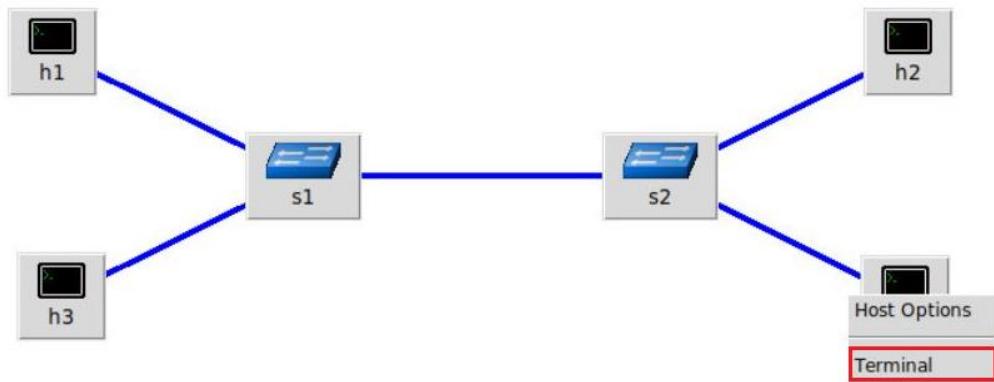


Figure 25. Opening a terminal on host h4.

**Step 2.** In the host h4 terminal, type the following command to run the host as an HTTP server using port 80.

```
iperf3 -s -p 80
```

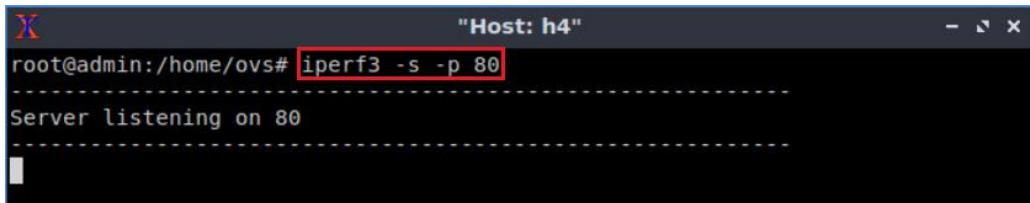


Figure 26. Running host h4 in server mode.

**Step 3.** To open host h3 terminal, hold right-click on the host h3 and select *Terminal*.

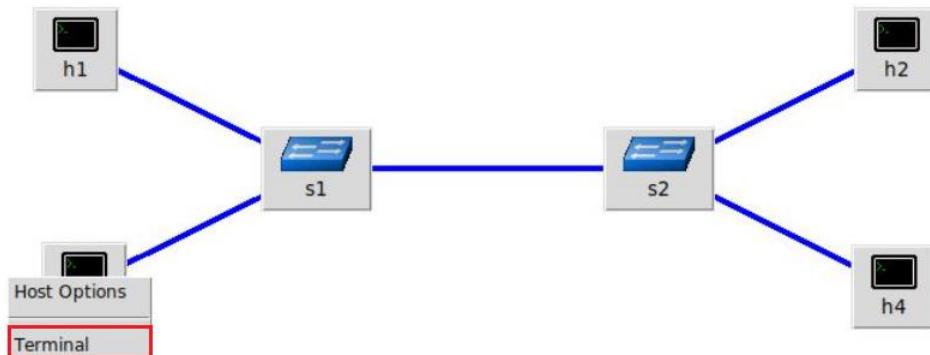


Figure 27. Opening a terminal on host h3.

**Step 4.** In the host h3 terminal, type the following command to run a throughput test between hosts h3 and h4. `-b` refers to the rate which is 300Mbps.

```
iperf3 -c 10.0.0.4 -p 80 -b 300mb
```

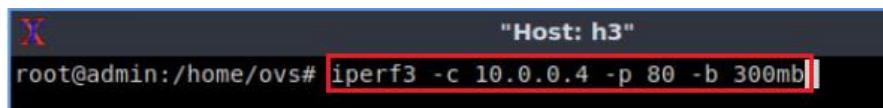
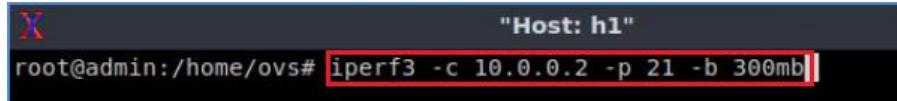


Figure 28. Running throughput test between hosts h3 and h4.

Do not start the test.

**Step 5.** In the host h1 terminal, type the following command to run a throughput test between hosts h1 and h2. **b** refers to the rate which is 300Mbps.

```
iperf3 -c 10.0.0.2 -p 21 -b 300mb
```

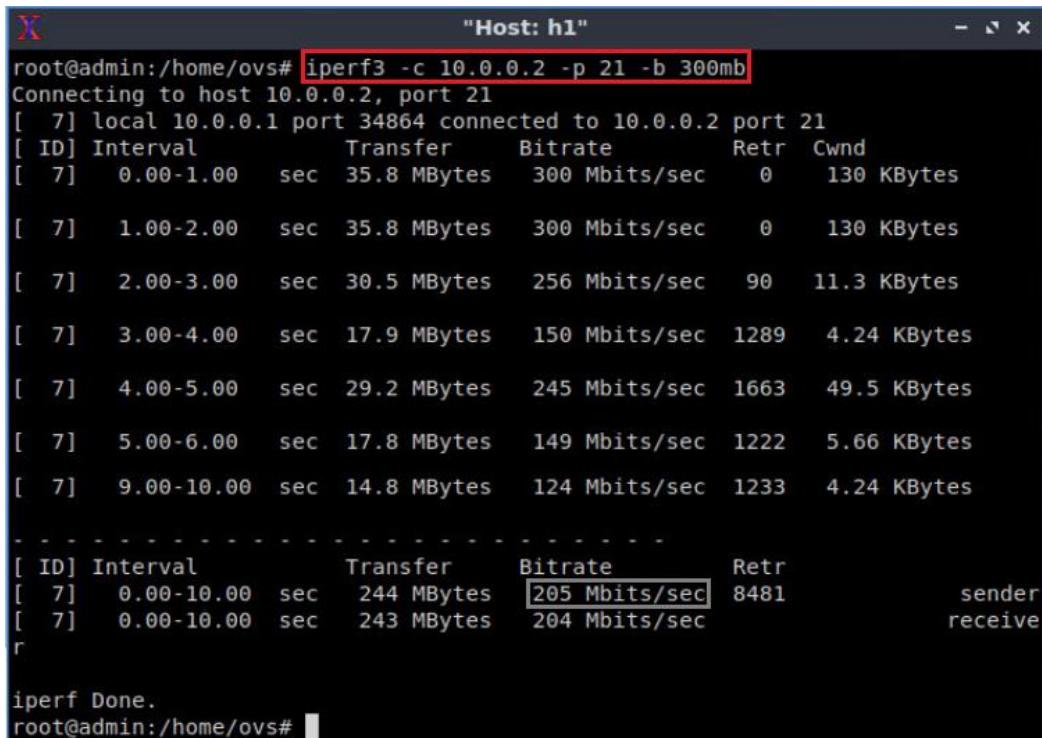


```
"Host: h1"
root@admin:/home/ovs# iperf3 -c 10.0.0.2 -p 21 -b 300mb
```

Figure 29. Running throughput test between hosts h1 and h2.

Execute two commands immediately one after another.

**Step 6.** Execute the command in host h1.



```
"Host: h1"
root@admin:/home/ovs# iperf3 -c 10.0.0.2 -p 21 -b 300mb
Connecting to host 10.0.0.2, port 21
[ 7] local 10.0.0.1 port 34864 connected to 10.0.0.2 port 21
[ ID] Interval      Transfer     Bitrate      Retr  Cwnd
[ 7]  0.00-1.00    sec   35.8 MBytes   300 Mbits/sec    0   130 KBytes
[ 7]  1.00-2.00    sec   35.8 MBytes   300 Mbits/sec    0   130 KBytes
[ 7]  2.00-3.00    sec   30.5 MBytes   256 Mbits/sec   90   11.3 KBytes
[ 7]  3.00-4.00    sec   17.9 MBytes   150 Mbits/sec  1289   4.24 KBytes
[ 7]  4.00-5.00    sec   29.2 MBytes   245 Mbits/sec  1663   49.5 KBytes
[ 7]  5.00-6.00    sec   17.8 MBytes   149 Mbits/sec  1222   5.66 KBytes
[ 7]  9.00-10.00   sec   14.8 MBytes   124 Mbits/sec  1233   4.24 KBytes
[ 7]  0.00-10.00   sec   244 MBytes   205 Mbits/sec  8481
[ 7]  0.00-10.00   sec   243 MBytes   204 Mbits/sec
sender receive
r

iperf Done.
root@admin:/home/ovs#
```

Figure 30. Running throughput test between hosts h1 and h2.

Execute command on host h3 immediately. The test will be running for 10 seconds. The output is summarized in the figure above.

Consider the figure above. In host h1, the bitrate is approximately 200 Mbps.

**Step 7.** Execute the command in host h3.

```

root@admin:/home/ovs# iperf3 -c 10.0.0.4 -p 80 -b 300mb
Connecting to host 10.0.0.4, port 80
[ 7] local 10.0.0.3 port 60620 connected to 10.0.0.4 port 80
[ ID] Interval Transfer Bitrate Retr Cwnd
[ 7] 0.00-1.00 sec 35.8 MBytes 300 Mbits/sec 0 519 KBytes
[ 7] 1.00-2.00 sec 35.8 MBytes 300 Mbits/sec 10 5.66 KBytes
[ 7] 2.00-3.00 sec 12.2 MBytes 103 Mbits/sec 348 9.90 KBytes
[ 7] 3.00-4.00 sec 17.2 MBytes 145 Mbits/sec 826 9.90 KBytes
[ 7] 4.00-5.00 sec 10.2 MBytes 86.0 Mbits/sec 728 18.4 KBytes
[ 7] 5.00-6.00 sec 16.4 MBytes 137 Mbits/sec 978 43.8 KBytes
[ 7] 9.00-10.00 sec 24.1 MBytes 202 Mbits/sec 976 5.66 KBytes
[ ID] Interval Transfer Bitrate Retr
[ 7] 0.00-10.00 sec 203 MBytes 170 Mbits/sec 6952 sender
[ 7] 0.00-10.00 sec 202 MBytes 169 Mbits/sec 6952 receiver
r

iperf Done.
root@admin:/home/ovs#

```

Figure 31. Running throughput test between hosts h3 and h4.

Consider the figure above. Host h3 is competing with host h1 and the bitrate is approx. 170 Mbps. Though switch s1 received 600 Mbps from two hosts, approx. 370 Mbps was allowed and marked as green. The remaining flows were marked as red and dropped by switch s1.

**Step 8.** Type the following command to delete the QoS policy.

```
sudo ovs-vsctl -- --all destroy qos
```

```

File Actions Edit View Help
ovs@admin: ~
ovs@admin: ~
ovs@admin:~$ sudo ovs-vsctl -- --all destroy qos
ovs@admin:~$ 

```

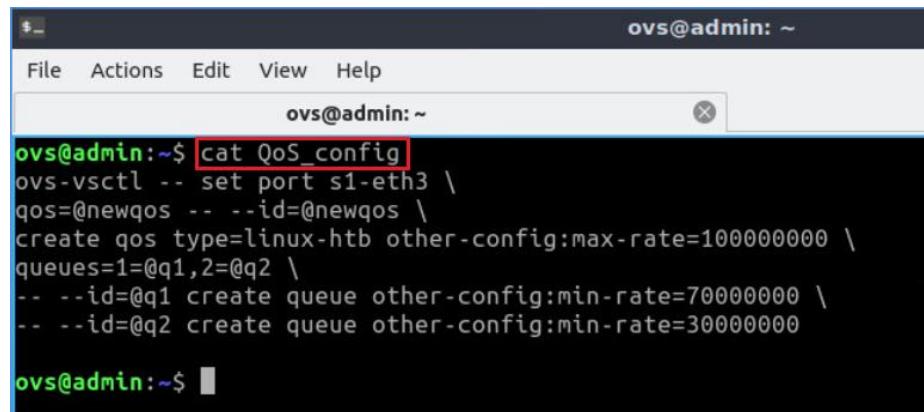
Figure 32. Deleting all QoS.

## 6 Configuring QoS shaping in switch s1

To configure QoS in an Open vSwitch, you will assign maximum bandwidth and create queues with different priorities. The next step is to add these queues to the ports on the switch s1 where you want to implement the QoS. Finally, you will map the required queue ID with the flow. For simplicity, you will use a script to configure QoS shaping in switch s1.

**Step 1.** To visualize the QoS commands used in the lab, type the following command:

```
cat QoS_config
```



```
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ cat QoS_config
ovs-vsctl -- set port s1-eth3 \
qos=@newqos -- --id=@newqos \
create qos type=linux-htb other-config:max-rate=100000000 \
queues=1=@q1,2=@q2 \
-- --id=@q1 create queue other-config:min-rate=70000000 \
-- --id=@q2 create queue other-config:min-rate=30000000
ovs@admin:~$
```

Figure 33. Displaying QoS commands.

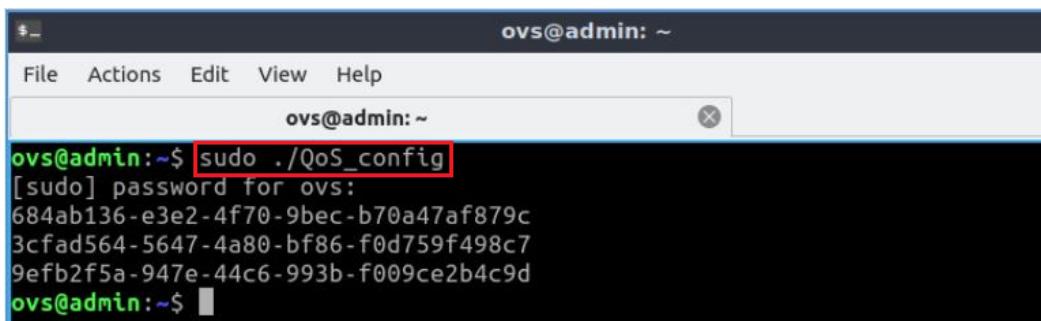
Following commands will be executed in the script:

```
ovs-vsctl -- set port s1-eth3 \
qos=@newqos -- --id=@newqos \
create qos type=linux-htb other-config:max-rate=100000000 \
queues=1=@q1,2=@q2 \
-- --id=@q1 create queue other-config:min-rate=70000000 \
-- --id=@q2 create queue other-config:min-rate=30000000
```

In the figure above, a QoS shaping is created with a maximum data transfer rate of 100 Mbps and attached to port *s1-eth3*. Two queues, q1 with queue *id* = 1 and q2 with queue id = 2 have been created. Queue 1 (q1) has a minimum transfer rate of 70 Mbps. q2 has a minimum transfer rate of 30 Mbps. The QoS method is egress only which means these rates will be applied when the packets are being forwarded out from the port.

**Step 2.** To configure QoS in switch s1, type the following command to execute the script.

```
sudo ./QoS_config
```

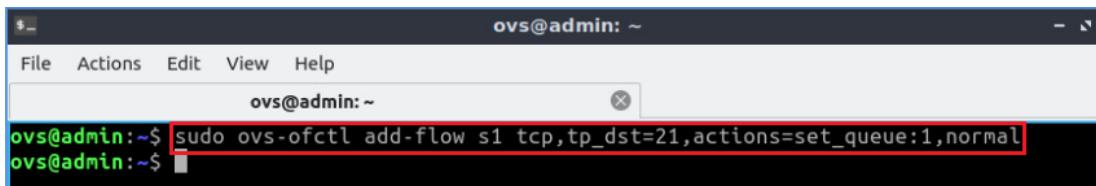


```
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ./QoS_config
[sudo] password for ovs:
684ab136-e3e2-4f70-9bec-b70a47af879c
3cfad564-5647-4a80-bf86-f0d759f498c7
9efb2f5a-947e-44c6-993b-f009ce2b4c9d
ovs@admin:~$
```

Figure 34. Configuring QoS in switch s1.

**Step 3.** Type the following command to install a manual flow entry in switch s1.

```
sudo ovs-ofctl add-flow s1 tcp,tp_dst=21,actions=set_queue:1,normal
```



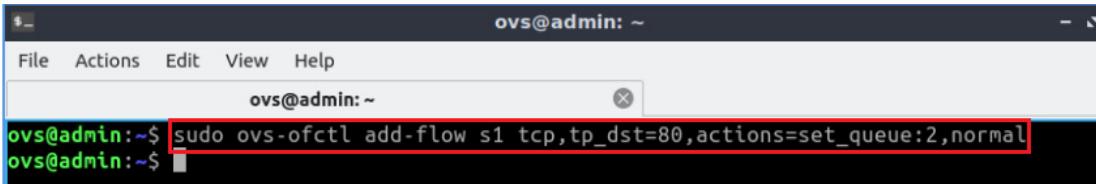
```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 tcp,tp_dst=21,actions=set_queue:1,normal  
ovs@admin:~$
```

Figure 35. Adding a manual flow in the switch.

Consider the figure above. The flow matches on the TCP port (21), sets up the appropriate queue (q1) and executes the *normal* action, which performs the traditional switching operation.

**Step 4.** Type the following command to install a manual flow entry in switch s1.

```
sudo ovs-ofctl add-flow s1 tcp,tp_dst=80,actions=set_queue:2,normal
```



```
ovs@admin:~$ sudo ovs-ofctl add-flow s1 tcp,tp_dst=80,actions=set_queue:2,normal  
ovs@admin:~$
```

Figure 36. Adding a manual flow in the switch.

Consider the figure above. The flow matches on the TCP port (80), sets up the appropriate queue (q2), and executes the *normal* action, which performs the traditional switching operation.

## 7 Verifying QoS shaping configuration

In this section, you will verify the QoS shaping configuration. In section 7.1, you will verify the traffic rate for individual hosts. In section 7.2, you will verify that host h1 will get higher priority when competing with host h3.

### 7.1 Verifying QoS and traffic rate for individual hosts

**Step 1.** Type the following command to verify QoS configuration for the interface *s1-eth3*.

```
sudo ovs-appctl -t ovs-vswitchd qos/show s1-eth3
```

```
ovs@admin:~$ sudo ovs-appctl -t ovs-vswitchd qos/show s1-eth3
QoS: s1-eth3 linux-htb
max-rate: 100000000

Default:
  burst: 12512
  min-rate: 12000
  max-rate: 100000000
  tx_packets: 0
  tx_bytes: 0
  tx_errors: 0

Queue 1:
  burst: 12512
  min-rate: 70000000
  max-rate: 100000000
  tx_packets: 0
  tx_bytes: 0
  tx_errors: 0

Queue 2:
  burst: 12512
  min-rate: 30000000
  max-rate: 100000000
  tx_packets: 0
  tx_bytes: 0
```

Figure 37. Verifying QoS configuration.

Consider the figure above. It lists all the information regarding QoS queues. You will notice two queues, q1 and q2 are added to the list. By default, Open vSwitch uses the default queue where the *min\_rate* is 12,000 and *max\_rate* is 100,000,000.

**Step 2.** In the host h1 terminal, type the following command to run a throughput test between hosts h1 and h2. At this point, host h1 should be able to utilize maximum bandwidth, 100 Mbps.

```
iperf3 -c 10.0.0.2 -p 21
```

```
"Host: h1"
root@admin:/home/ovs# iperf3 -c 10.0.0.2 -p 21
Connecting to host 10.0.0.2, port 21
[ 7] local 10.0.0.1 port 47868 connected to 10.0.0.2 port 21
[ ID] Interval      Transfer     Bitrate      Retr  Cwnd
[ 7]  0.00-1.00  sec   12.2 MBytes   103 Mbits/sec   0   157 KBytes
[ 7]  1.00-2.00  sec   11.6 MBytes   97.0 Mbits/sec   0   163 KBytes
[ 7]  2.00-3.00  sec   11.2 MBytes   93.8 Mbits/sec   0   163 KBytes
[ 7]  3.00-4.00  sec   11.6 MBytes   97.0 Mbits/sec   0   163 KBytes
[ 7]  4.00-5.00  sec   11.2 MBytes   93.8 Mbits/sec   0   163 KBytes
[ 7]  9.00-10.00 sec   11.2 MBytes   93.8 Mbits/sec   0   163 KBytes
-
[ ID] Interval      Transfer     Bitrate      Retr
[ 7]  0.00-10.00 sec   115 MBytes   96.3 Mbits/sec   0
[ 7]  0.00-10.01 sec   114 MBytes   95.6 Mbits/sec
sender receive
iperf Done.
root@admin:/home/ovs#
```

Figure 38. Running throughput test between hosts h1 and h2.

The figure above shows that the bitrate is ~95.6 Mbps.

**Step 3.** To open the host h4 terminal, hold right-click on host h4 and select Terminal.

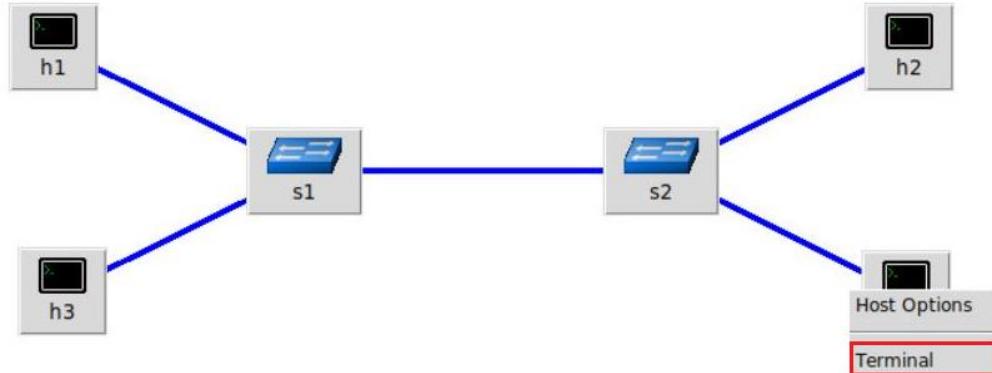


Figure 39. Opening a terminal on host h4.

**Step 4.** In the host h4 terminal, type the following command to run the host as an HTTP server using port 80.

```
iperf3 -s -p 80
```

```
"Host: h4"
root@admin:/home/ovs# iperf3 -s -p 80
-----
Server listening on 80
-----
```

Figure 40. Running host h4 in server mode.

**Step 5.** To open host h3 terminal, hold right-click on the host h3 and select *Terminal*.

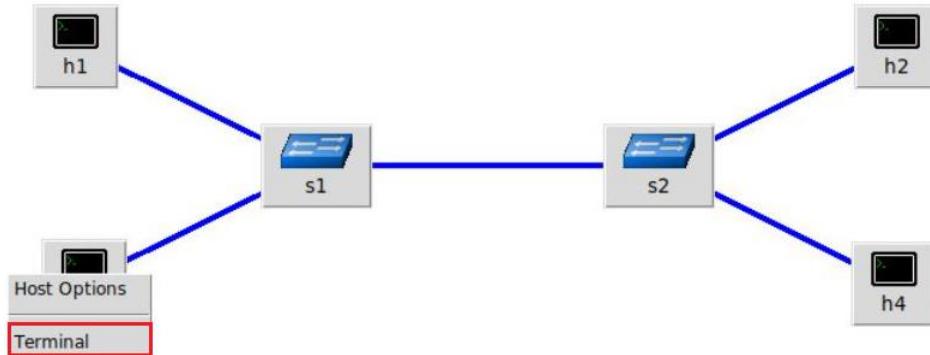


Figure 41. Opening a terminal on host h3.

**Step 6.** In the host h3 terminal, type the following command to run the host in client mode and run a throughput test between hosts h3 and h4. At this point, host h3 should be able to utilize maximum bandwidth, 100 Mbps.

```
iperf3 -c 10.0.0.4 -p 80
```

The terminal window shows the output of the iperf3 command. The title bar says "Host: h3". The command entered was "iperf3 -c 10.0.0.4 -p 80". The output shows multiple data transfer intervals with their respective transfer times, transfer sizes, bitrates, and retransmissions. The final summary shows a sender bitrate of 95.6 Mbytes/sec and a receiver bitrate of 96.2 Mbytes/sec.

```
"Host: h3"
root@admin:/home/ovs# iperf3 -c 10.0.0.4 -p 80
Connecting to host 10.0.0.4, port 80
[ 7] local 10.0.0.3 port 41848 connected to 10.0.0.4 port 80
[ ID] Interval           Transfer     Bitrate      Retr  Cwnd
[ 7]  0.00-1.00   sec   12.1 MBytes   102 Mbits/sec   0   130 KBytes
[ 7]  1.00-2.00   sec   11.5 MBytes   96.4 Mbits/sec   0   130 KBytes
[ 7]  2.00-3.00   sec   11.2 MBytes   93.8 Mbits/sec   0   130 KBytes
[ 7]  3.00-4.00   sec   11.5 MBytes   96.4 Mbits/sec   0   130 KBytes
[ 7]  4.00-5.00   sec   11.5 MBytes   96.4 Mbits/sec   0   130 KBytes
[ 7]  9.00-10.00  sec   11.5 MBytes   96.4 Mbits/sec   0   130 KBytes
[ 7]  10.00-10.01 sec   115 MBytes  96.2 Mbits/sec   0
[ 7]  10.00-10.01 sec   114 MBytes  95.6 Mbits/sec   0
sender receive
iperf Done.
root@admin:/home/ovs#
```

Figure 42. Running throughput test between hosts h3 and h4.

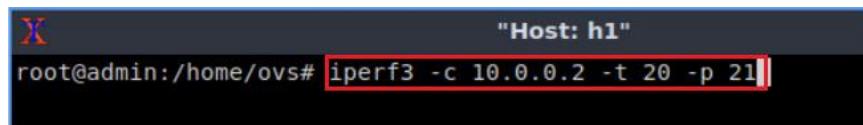
The figure above shows that the bitrate is ~95.6 Mbps.

## 7.2 Verifying traffic rate for competing hosts

**Step 1.** In the host h1 terminal, type the following command to run a throughput test between hosts h1 and h2 for 20 seconds.

```
iperf3 -c 10.0.0.2 -t 20 -p 21
```

Do not start the test.

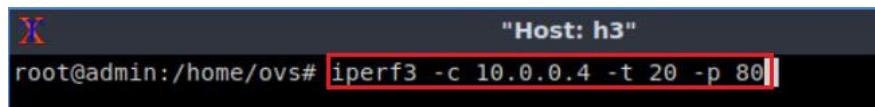


```
"Host: h1"
root@admin:/home/ovs# iperf3 -c 10.0.0.2 -t 20 -p 21
```

Figure 43. Running throughput test between hosts h1 and h2.

**Step 2.** In the host h3 terminal, type the following command to run a throughput test between hosts h3 and h4 for 20 seconds.

```
iperf3 -c 10.0.0.4 -t 20 -p 80
```

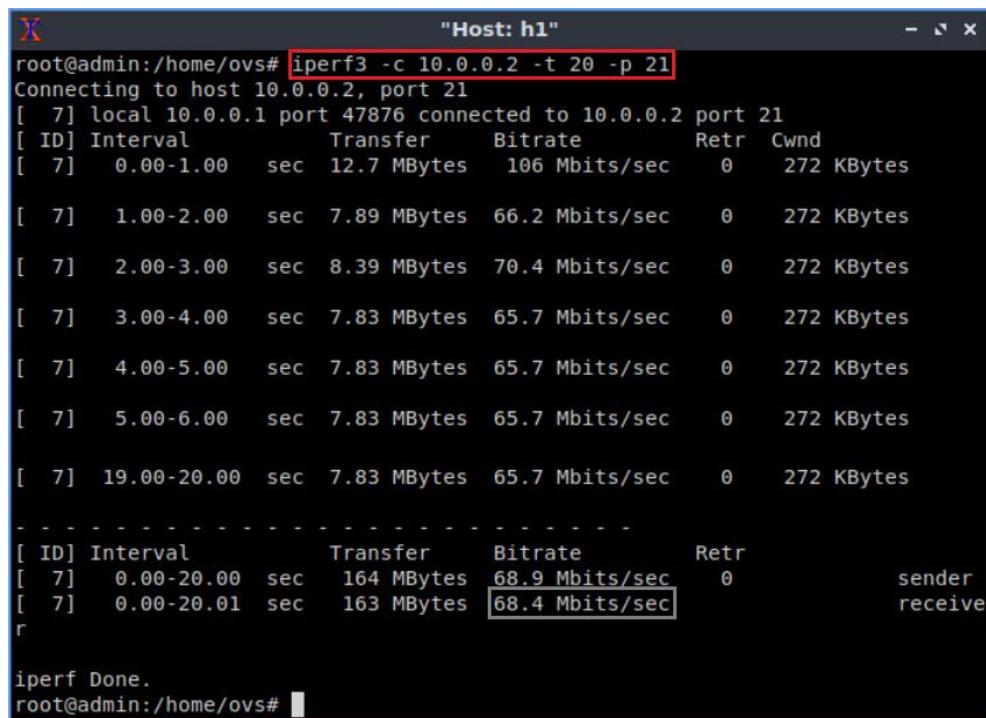


```
"Host: h3"
root@admin:/home/ovs# iperf3 -c 10.0.0.4 -t 20 -p 80
```

Figure 44. Running throughput test between hosts h3 and h4.

Execute two commands immediately one after another.

**Step 3.** Execute the command in host h1.



```
"Host: h1"
root@admin:/home/ovs# iperf3 -c 10.0.0.2 -t 20 -p 21
Connecting to host 10.0.0.2, port 21
[ 7] local 10.0.0.1 port 47876 connected to 10.0.0.2 port 21
[ ID] Interval      Transfer     Bitrate    Retr  Cwnd
[ 7]  0.00-1.00   sec   12.7 MBytes   106 Mbits/sec   0   272 KBytes
[ 7]  1.00-2.00   sec   7.89 MBytes   66.2 Mbits/sec   0   272 KBytes
[ 7]  2.00-3.00   sec   8.39 MBytes   70.4 Mbits/sec   0   272 KBytes
[ 7]  3.00-4.00   sec   7.83 MBytes   65.7 Mbits/sec   0   272 KBytes
[ 7]  4.00-5.00   sec   7.83 MBytes   65.7 Mbits/sec   0   272 KBytes
[ 7]  5.00-6.00   sec   7.83 MBytes   65.7 Mbits/sec   0   272 KBytes
[ 7]  19.00-20.00  sec   7.83 MBytes   65.7 Mbits/sec   0   272 KBytes

[ ID] Interval      Transfer     Bitrate    Retr
[ 7]  0.00-20.00  sec   164 MBytes   68.9 Mbits/sec   0
[ 7]  0.00-20.01  sec   163 MBytes   68.4 Mbits/sec

iperf Done.
root@admin:/home/ovs#
```

Figure 45. Running throughput test between hosts h1 and h2.

Execute command on host h3 immediately. The test will be running for 20 seconds. The output is summarized in the figure above.

Consider the figure above. In host h1, the first bitrate is approximately 100 Mbps. Whenever host h3 started sharing the link with host h1, the bitrate dropped to approximately 70 Mbps (68.4 Mbps for the figure above).

**Step 4.** Execute the command in host h3.

```

root@admin:/home/ovs# iperf3 -c 10.0.0.4 -t 20 -p 80
Connecting to host 10.0.0.4, port 80
[ 7] local 10.0.0.3 port 41856 connected to 10.0.0.4 port 80
[ 7] ID] Interval Transfer Bitrate Retr Cwnd
[ 7] 0.00-1.00 sec 3.68 MBytes 30.9 Mbits/sec 0 66.5 KBytes
[ 7] 1.00-2.00 sec 3.54 MBytes 29.7 Mbits/sec 0 66.5 KBytes
[ 7] 2.00-3.00 sec 3.36 MBytes 28.1 Mbits/sec 0 66.5 KBytes
[ 7] 3.00-4.00 sec 3.54 MBytes 29.7 Mbits/sec 0 66.5 KBytes
[ 7] 4.00-5.00 sec 3.36 MBytes 28.1 Mbits/sec 0 66.5 KBytes
[ 7] 18.00-19.00 sec 3.48 MBytes 29.2 Mbits/sec 0 96.2 KBytes
[ 7] 19.00-20.00 sec 11.8 MBytes 99.0 Mbits/sec 0 201 KBytes
[ 7] 0.00-20.00 sec 77.3 MBytes 32.4 Mbits/sec 0
[ 7] 0.00-20.01 sec 76.4 MBytes 32.0 Mbits/sec
r
iperf Done.
root@admin:/home/ovs#

```

Figure 46. Running throughput test between hosts h3 and h4.

Consider the figure above. Host h3 is competing with host h1 and the bitrate is approx. 30 Mbps (32.0 Mbps for the figure above) since host h1 gets higher priority than host h3.

This concludes Lab 9. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. Network lessons, “*Introduction to QoS*”, [Online], Available: <https://networklessons.com/quality-of-service/introduction-qos-quality-service>
2. T. Szigeti, R. Barton, C. Hatting, K. Briley, “*End-to-End QoS Network Design*”, 2<sup>nd</sup> Edition.
3. TechTarget, “*Quality of service (QoS)*”, [Online], Available: [https://searchunifiedcommunications.techtarget.com/definition/QoS-Quality-of-Service#:~:text=Quality%20of%20service%20\(QoS\)%20refers,and%20jitter%20on%20a%20network.&text=Organizations%20can%20reach%20a%20QoS,jitter%20buffer%20and%20traffic%20shaping](https://searchunifiedcommunications.techtarget.com/definition/QoS-Quality-of-Service#:~:text=Quality%20of%20service%20(QoS)%20refers,and%20jitter%20on%20a%20network.&text=Organizations%20can%20reach%20a%20QoS,jitter%20buffer%20and%20traffic%20shaping).
4. Linux foundation, “*Quality of service (QoS)*”, [Online], Available: <https://docs.openvswitch.org/en/latest/faq/qos/#:~:text=Q%3A%20Does%20OVS%20support%20Quality,excess%20of%20the%20configured%20rate>.
5. Palo alto networks, “*What is Quality of Service*”, [Online], Available: <https://www.paloaltonetworks.com/cyberpedia/what-is-quality-of-service-qos>
6. Mininet walkthrough, [Online]. Available: <http://mininet.org>.
7. Juniper Networks, “*Learn about Quality of Service (QoS)*”, [Online]. Available: [https://www.juniper.net/documentation/en\\_US/learn-about/LA\\_QoS.pdf](https://www.juniper.net/documentation/en_US/learn-about/LA_QoS.pdf)



## OPEN VIRTUAL SWITCH

### Exercise 4: Configuring Quality of Service (QoS)

Document Version: **09-27-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

1	Exercise topology .....	3
1.1	Topology settings .....	3
1.2	Credentials .....	3
2	Deliverables.....	4

## 1 Exercise topology

Consider Figure 1. The topology consists of six hosts and two switches. All the hosts belong to the same network, 10.0.0.0/8.

The goal of this lab is to configure Quality of Service (QoS) in Open vSwitch.

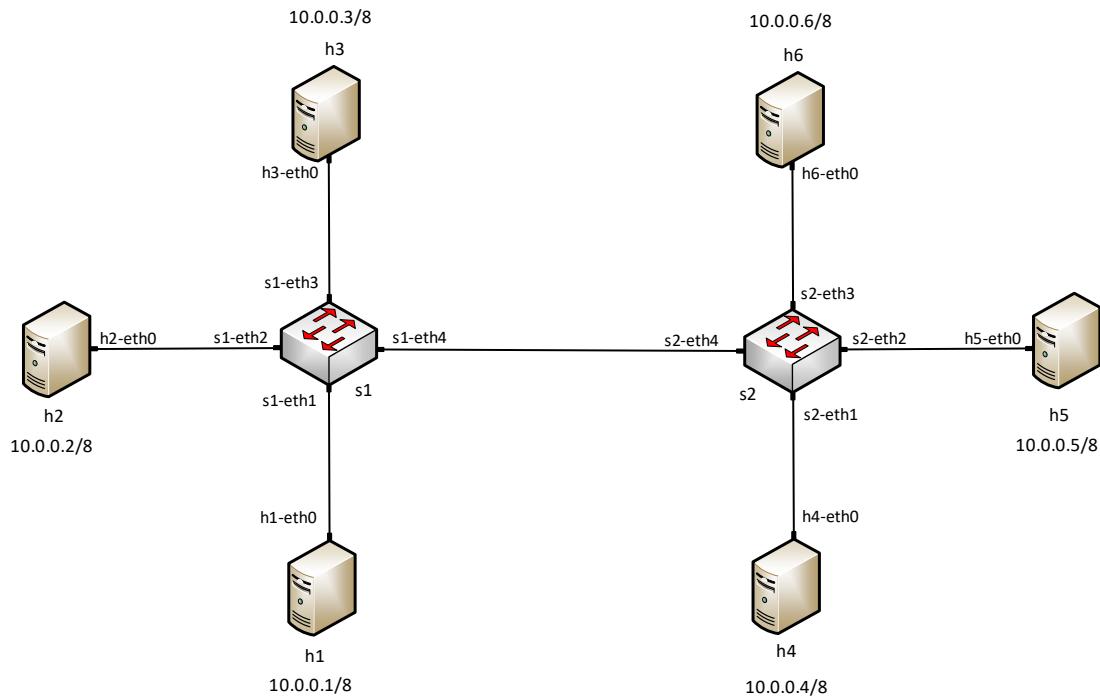


Figure 1. Exercise topology.

### 1.1 Topology settings

The devices are already configured according to Table 1.

Table 1. Topology information.

Host	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8
h3	h3-eth0	10.0.0.3	/8
h4	h4-eth0	10.0.0.4	/8
h5	h5-eth0	10.0.0.5	/8
h6	h6-eth0	10.0.0.6	/8

### 1.2 Credentials

The information in Table 2 provides the credentials to access the Client's virtual machine.

Table 2. Credentials to access the Client's virtual machine.

Device	Account	Password
Client	admin	password

## 2 Deliverables

Follow the steps below to complete the exercise.

- a) Start MiniEdit by clicking on MiniEdit's shortcut. Load the topology *Exercise4.mn* located at *~/OVS\_Labs/Exercise4* as shown in the figure below.

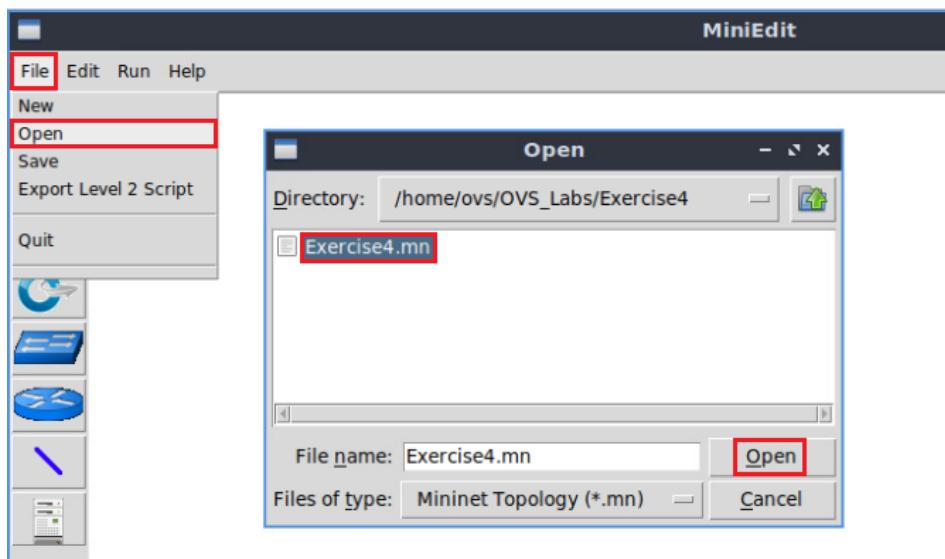


Figure 2. Loading the topology file in Mininet.

- b) Run the emulation in Mininet.
- c) Set the ingress policy rate for the interface *s1-eth1* to 20 megabits per second (Mbps). Verify the configuration by running a throughput test between hosts *h1* and *h4*. Explain the result. What is the throughput?
- d) Delete the QoS policy.
- e) Add a meter in switch *s1* and limit the rate to 500 Mbps.
- f) Add a flow entry in switch *s1* which matches on TCP port (21), add the meter to the action.

- g)** Add a flow entry in switch s1 which matches on source IP address (10.0.0.2), add the meter to the action.
- h)** Add a flow entry in switch s1 which matches on destination IP address (10.0.0.6), add the meter to the action.
- i)** Run a throughput test between hosts h1 and h4 using tcp port 21 for 20 seconds. Run a throughput test between hosts h2 and h5 for 20 seconds. Run a throughput test between hosts h3 and h6 for 20 seconds. Explain the result. Explain the throughput?

Run all the test immediately one after another.

- j)** Delete the QoS policy.
- k)** Configure QoS shaping with a maximum data transfer rate of 100 Mbps and attach to port s1-eth4. Create three queues, q1 with queue *id* = 1, q2 with queue id = 2 and q3 with queue id = 3. Configure q1, q2 and q3 with minimum transfer rate of 80, 10 and 10 Mbps, respectively.
- l)** Add a flow entry in switch s1 which matches on TCP port (21), set queue, q1 in the action and execute normal forwarding.
- m)** Add a flow entry in switch s1 which matches on source IP address (10.0.0.2), set queue, q2 in the action and execute normal forwarding.
- n)** Add a flow entry in switch s1 which matches on destination IP address (10.0.0.6), set queue, q3 in the action and execute normal forwarding.
- o)** Repeat **i)** and explain the result. What is the throughput?
- p)** Delete flow entry that matches against TCP. Use the following command to delete the flow:

```
sudo ovs-ofctl del-flows s1 tcp
```

- q)** Run a throughput test between hosts h2 and h5 for 20 seconds. Run a throughput test between hosts h3 and h6 for 20 seconds. Explain the result. What is the throughput?

Run all the test immediately one after another.



## OPEN VIRTUAL SWITCH

### Lab 10: Open vSwitch Database (OVSDB)

Document Version: **09-15-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    Traditional Database .....	4
1.2    OVSDB in Open vSwitch .....	4
2    Lab topology.....	5
2.1    Lab settings.....	6
2.2    Loading a topology .....	6
3    Visualizing and monitoring OVSDB using ovs-vsctl tool .....	9
4    Visualizing and monitoring OVSDB using ovsdb-client tool.....	11
References .....	17

## Overview

This lab discusses the concept of Open Virtual Switch Database Management Protocol (OVSDB), a network-accessible database system used for configuring and monitoring Open vSwitch. The lab aims to demonstrate how to manage and manipulate OVSDB using command-line interface (CLI) tools.

## Objectives

By the end of this lab, you should be able to:

1. Understand the concept of a database.
2. Understand OVSDB.
3. Understand Open vSwitch CLI tools.
4. Visualize and monitor OVSDB using ovs-vsctl tool.
5. Access, edit, and manage OVSDB using ovsdb-client tool.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Visualizing and monitoring OVSDB using ovs-vsctl tool.
4. Section 4: Visualizing and monitoring OVSDB using ovsdb-client tool.

### 1      Introduction

Open vSwitch is an open-source software switch designed to be used as a virtual switch which is open to programmatic extension. OVSDB was introduced to create a modern, programmatic management protocol interface<sup>3</sup>. Other than Open vSwitch, it is supported by more switch platforms, such as Cumulus, Arista, and Dell. By supporting OVSDB, these

vendors are integrating their hardware platforms with SDN and network virtualization solutions.

## 1.1 Traditional Database

A database is a combination of data entries and the Database Management Systems (DBMS). A database is stored in a file (entity) or a set of files<sup>1</sup>. Each file has a list of records that consists of one or more fields. Fields are the data storage units, and each field contains information about the entity described by the database. Records are also organized into tables that have information about relationships between various fields.

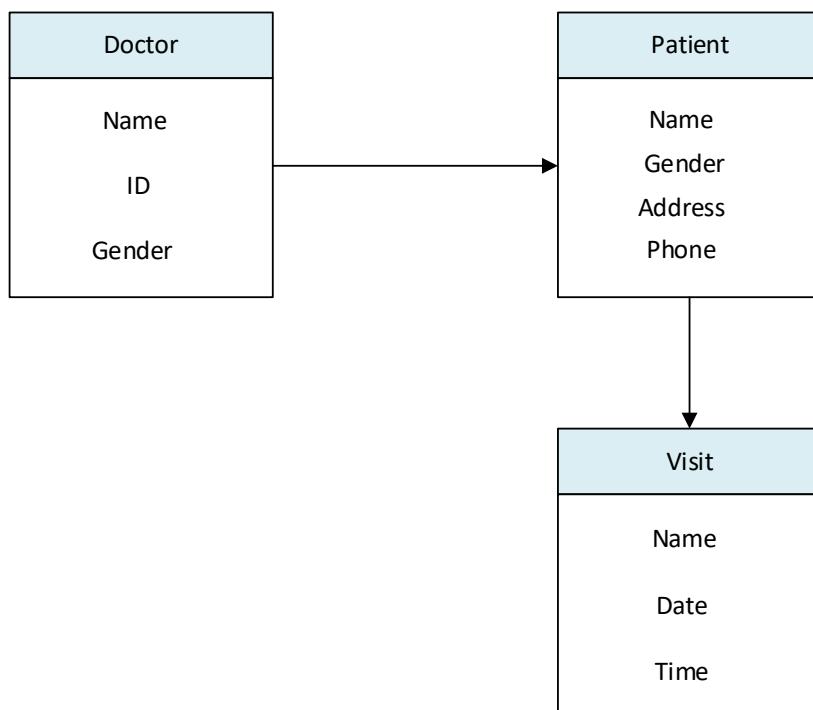


Figure 1. Database entry for patient visit.

The figure above shows a database entry for a patient visit. There are three entities, Doctor, Patient, and Visit. If a patient wants to visit a doctor, the record will be stored in these three tables. The patient information is stored in the patient entity, where the patient's name, gender, address, and phone number are also stored. The doctor who is going to serve the patient will have his information stored in the database. The entity Visit will have the name of the patient, name, and date of the visit stored in the database.

## 1.2 OVSDB in Open vSwitch

OVSDB is a network-accessible database system. OVSDB schema specifies the tables in a database. The types of columns can include data, uniqueness, and referential integrity constraints. OVSDB clients and servers use JavaScript Object Notation (JSON) based protocol to communicate with each other. OVSDB allows clients to monitor the content of the database. If any monitored portion of the database changes, the server notifies the

client about the modification (if any row was added or modified or deleted)<sup>2</sup>. Thus, OVSDB clients can easily keep track of the newest contents of any part of the database.

The primary use of OVSDB is to configure and monitor Open vSwitch daemon `ovs-vswitchd`, using the schema that holds the configuration for one Open vSwitch daemon.

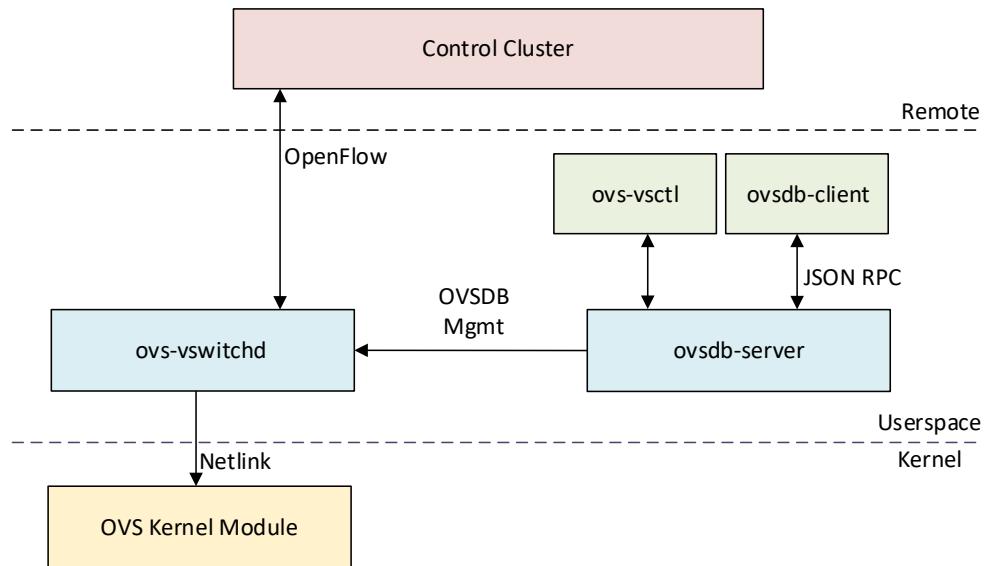


Figure 2. Open vSwitch Architecture.

Figure 2 illustrates the architecture of Open vSwitch. The `ovs-vswitchd` and the `ovsdb-server` are within the userspace. The CLI tool, `ovsdb-client` is responsible for communicating with `ovsdb-server` using JSON RPC protocol. The tool `ovs-vsctl` is used to configure bridges, ports, and tunnels. All the information is stored in the `ovsdb-server`. `ovs-vswitchd` is connected to `ovsdb-server` using the OVSDB management protocol to retrieve all the information. The switch configuration is stored on a persistent storage and survives a reboot.

## 2 Lab topology

Consider Figure 3. The topology consists of two switches and two hosts. Two switches are connected to each other. Each switch is connected to an end-host. They all belong to a single network (10.0.0.0/8).

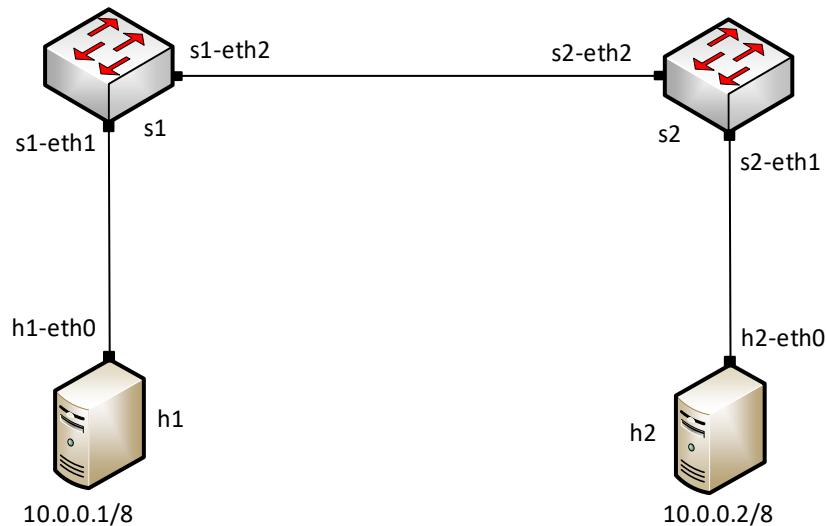


Figure 3. Lab topology.

## 2.1 Lab settings

The hosts are configured according to Table 2.

Table 2. Topology information.

Host	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8

## 2.2 Loading a topology

**Step 1.** Start by launching MiniEdit by clicking on the desktop's shortcut. When prompted for a password, type `password`.

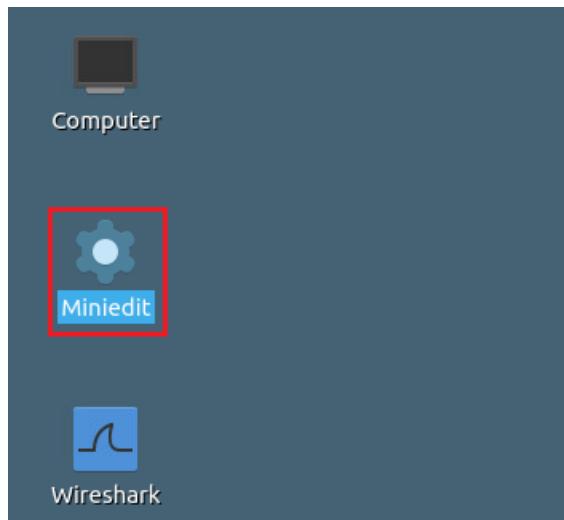


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File*, then *open* to load the lab's topology. Locate the *lab10.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab10* and click on *Open*.

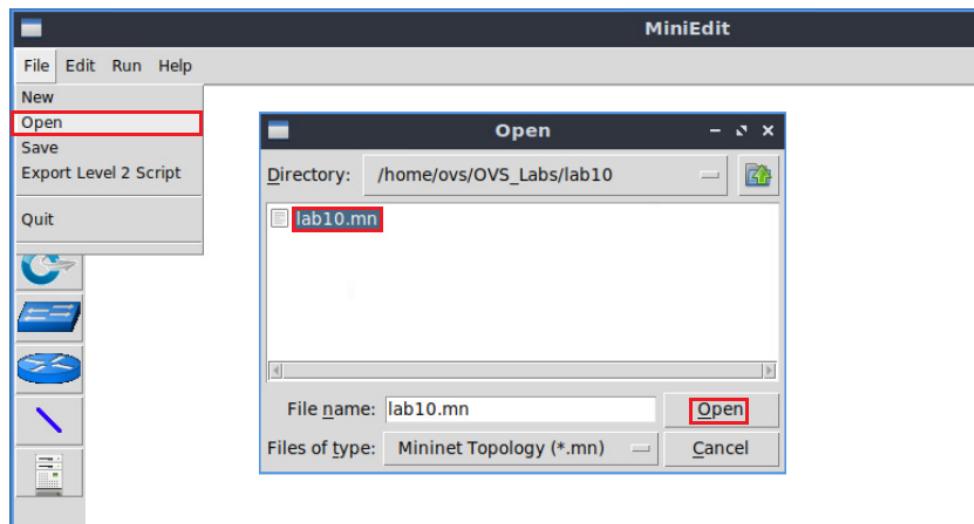


Figure 5. MiniEdit's Open dialog.



Figure 6. MiniEdit's topology.

**Step 3.** To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 7. Starting the emulation.

**Step 4.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

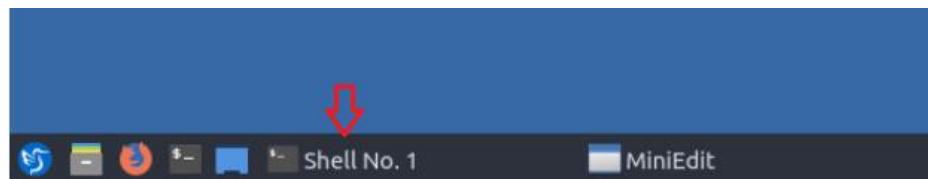


Figure 8. Opening Mininet's terminal.

**Step 5.** Issue the following command to display the interface names and connections.

```
links
```

```
Shell No. 1
File Actions Edit View Help
Shell No. 1
containernet> links
s1-eth1<->h1-eth0 (OK OK)
s2-eth1<->h2-eth0 (OK OK)
s1-eth2<->s2-eth2 (OK OK)
containernet>
```

A screenshot of a terminal window titled 'Shell No. 1'. The window has a menu bar with File, Actions, Edit, View, and Help. Below the menu is a title bar with 'Shell No. 1' and a close button. The main area of the terminal shows the command 'links' being run, followed by the output: 's1-eth1<->h1-eth0 (OK OK)', 's2-eth1<->h2-eth0 (OK OK)', and 's1-eth2<->s2-eth2 (OK OK)'. The entire terminal window is enclosed in a gray border.

Figure 9. Displaying network interfaces.

In Figure 9, the link displayed within the gray box indicates that interface eth1 of switch s1 is connected to interface eth0 of host h1 (i.e.,  $s1\text{-}eth1 <-> h1\text{-}eth0$ ).

**Step 6.** Hold right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

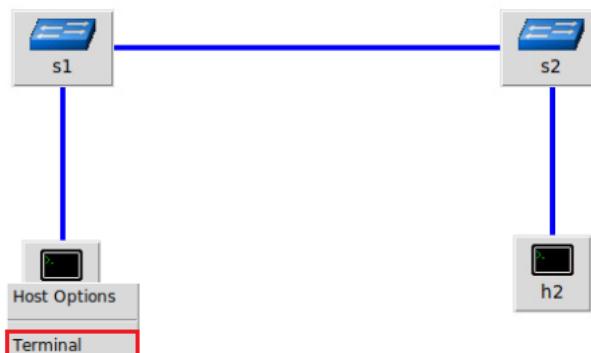
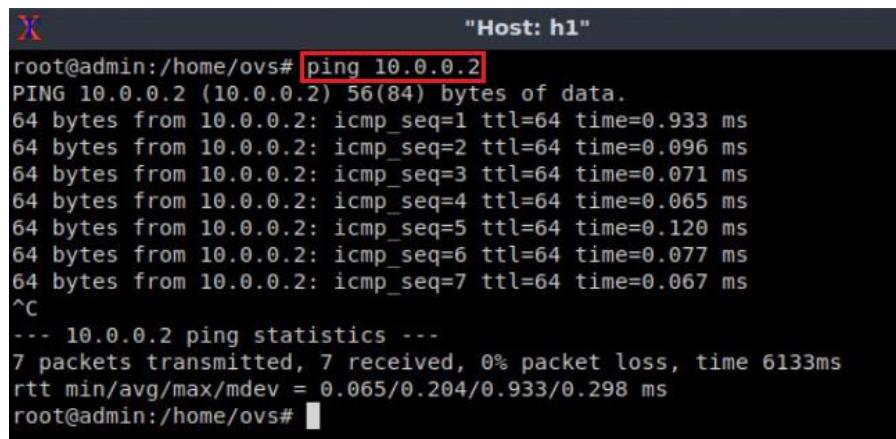


Figure 10. Opening a terminal on host h1.

**Step 7.** Test the connectivity between hosts h1 and h2 using the `ping` command. In host h1, type the command specified below. The following figure shows a successful test.

```
ping 10.0.0.2
```



```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.933 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.096 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.120 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.067 ms
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6133ms
rtt min/avg/max/mdev = 0.065/0.204/0.933/0.298 ms
root@admin:/home/ovs#
```

Figure 11. Output of the `ping` command.

Press `ctrl+c` to stop the test.

### 3 Visualizing and monitoring OVSDB using ovs-vsctl tool

In this section, you will visualize how `ovs-vsctl` manages the switch through interaction with `ovsdb-server`.

**Step 1.** Open the Linux terminal.



Figure 12. Opening Linux terminal.

**Step 2.** Type the following command to show all the available commands for `ovs-vsctl` tool. When prompted for a password, type `password`.

```
sudo ovs-vsctl -h
```

```

ovs@admin:~$ sudo ovs-vsctl -h
[sudo] password for ovs:
ovs-vsctl: ovs-vswitchd management utility
usage: ovs-vsctl [OPTIONS] COMMAND [ARG...]

Open vSwitch commands:
  init           initialize database, if not yet initialized
  show           print overview of database contents
  emer-reset     reset configuration to clean state

Bridge commands:
  add-br BRIDGE      create a new bridge named BRIDGE
  add-br BRIDGE PARENT VLAN   create new fake BRIDGE in PARENT on VLAN
  del-br BRIDGE     delete BRIDGE and all of its ports
  list-br          print the names of all the bridges
  br-exists BRIDGE    exit 2 if BRIDGE does not exist
  br-to-vlan BRIDGE   print the VLAN which BRIDGE is on
  br-to-parent BRIDGE  print the parent of BRIDGE
  br-set-external-id BRIDGE KEY VALUE  set KEY on BRIDGE to VALUE

```

Figure 13. Printing a brief help message.

Consider the figure above. The figure shows all the available commands to configure the Open vSwitch. In the following step, you will use the `show` command to print an overview of the database contents.

**Step 3.** Type the following command to print a brief overview of the database contents.

```

sudo ovs-vsctl show
c34727ea-2286-4c47-baec-a842d33645c9
  Manager "ptcp:6640:127.0.0.1"
  Bridge "s1"
    fail_mode: standalone
    Port "s1-eth2"
      Interface "s1-eth2"
    Port "s1"
      Interface "s1"
        type: internal
    Port "s1-eth1"
      Interface "s1-eth1"
  Bridge "s2"
    fail_mode: standalone
    Port "s2"
      Interface "s2"
        type: internal
    Port "s2-eth2"
      Interface "s2-eth2"

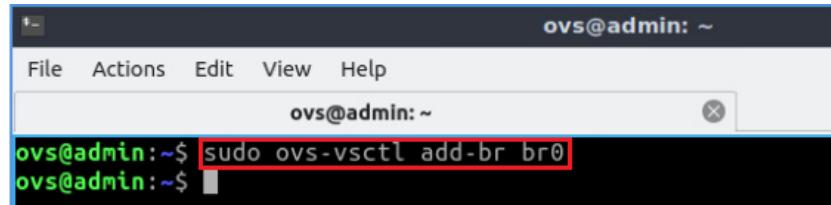
```

Figure 14. Printing an overview of the database contents.

Consider the figure above. The figure displays two bridges (s1 and s2) and the interfaces attached to the bridges. *Fail\_mode* is set to *standalone* since no controller is connected, and switches are responsible for forwarding packets.

**Step 4.** Type the following command to add a bridge *br0*.

```
sudo ovs-vsctl add-br br0
```

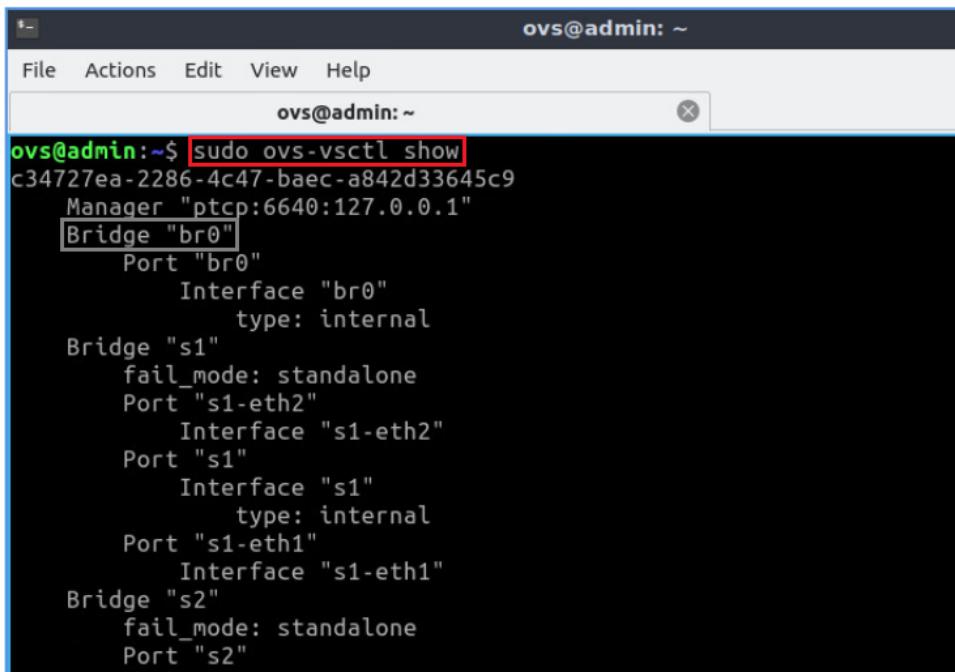


The screenshot shows a terminal window titled 'ovs@admin: ~'. The command 'sudo ovs-vsctl add-br br0' is highlighted with a red box. The output shows the command being run and then the prompt 'ovs@admin: ~\$'.

Figure 15. Adding a bridge.

**Step 5.** Type the following command to print a brief overview of the database contents.

```
sudo ovs-vsctl show
```



The screenshot shows a terminal window titled 'ovs@admin: ~'. The command 'sudo ovs-vsctl show' is highlighted with a red box. The output displays the OVSDB database structure, including bridges (br0, s1, s2), ports (br0, s1-eth2, s1-eth1), and interfaces (br0, s1-eth2, s1-eth1). The output is as follows:

```
c34727ea-2286-4c47-baec-a842d33645c9
  Manager "ptcp:6640:127.0.0.1"
  Bridge "br0"
    Port "br0"
      Interface "br0"
        type: internal
  Bridge "s1"
    fail_mode: standalone
    Port "s1-eth2"
      Interface "s1-eth2"
    Port "s1"
      Interface "s1"
        type: internal
    Port "s1-eth1"
      Interface "s1-eth1"
  Bridge "s2"
    fail_mode: standalone
    Port "s2"
```

Figure 16. Printing an overview of database contents.

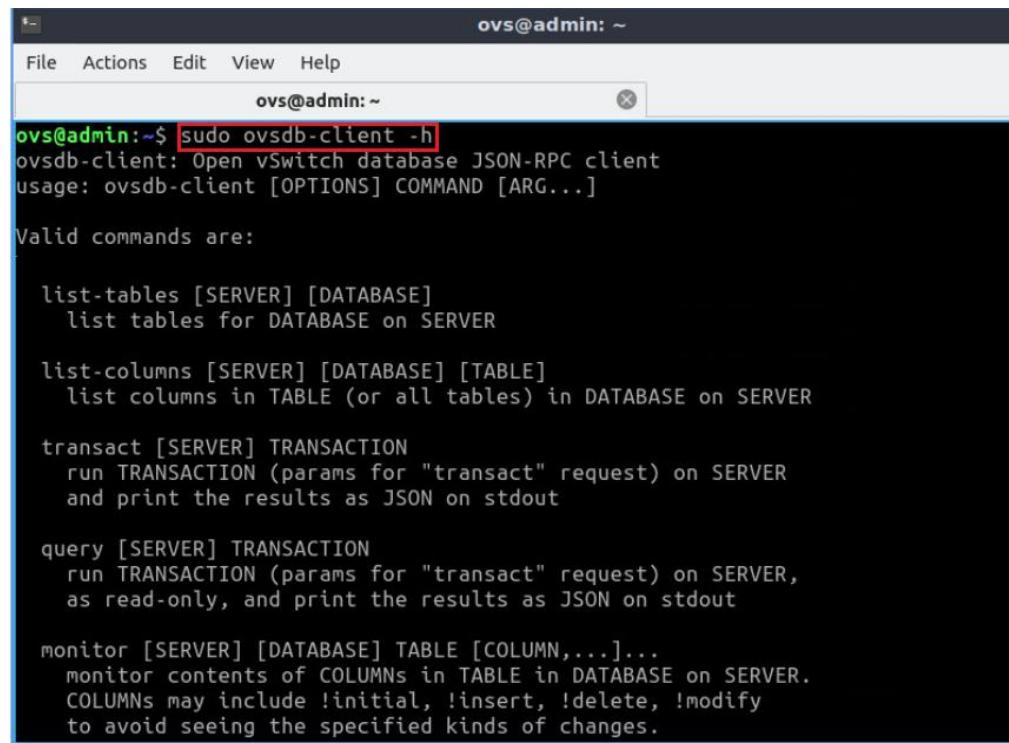
Consider the figure above. The figure shows the new bridge *br0* is added to the database.

#### 4 Visualizing and monitoring OVSDB using ovsdb-client tool

In this section, you will visualize how the client can access, edit, and manage the `ovsdb-server` using `ovsdb-client` tool.

**Step 1.** Type the following command to show all the available commands for `ovsdb-client` tool.

```
sudo ovsdb-client -h
```



```
ovs@admin:~$ sudo ovsdb-client -h
ovsdb-client: Open vSwitch database JSON-RPC client
usage: ovsdb-client [OPTIONS] COMMAND [ARG...]

Valid commands are:

list-tables [SERVER] [DATABASE]
    list tables for DATABASE on SERVER

list-columns [SERVER] [DATABASE] [TABLE]
    list columns in TABLE (or all tables) in DATABASE on SERVER

transact [SERVER] TRANSACTION
    run TRANSACTION (params for "transact" request) on SERVER
    and print the results as JSON on stdout

query [SERVER] TRANSACTION
    run TRANSACTION (params for "transact" request) on SERVER,
    as read-only, and print the results as JSON on stdout

monitor [SERVER] [DATABASE] TABLE [COLUMN,...]...
    monitor contents of COLUMNs in TABLE in DATABASE on SERVER.
    COLUMNs may include !initial, !insert, !delete, !modify
    to avoid seeing the specified kinds of changes.
```

Figure 17. Printing a brief help message.

Consider the figure above. It shows all the available commands to manipulate the database server. The list-tables command connects to the server and prints a table listing each table's name within the database. The list-columns command connects to the server and prints a table listing each column's name and type. If the table is specified, only columns in that particular table are listed; otherwise, it includes all the tables.

**Step 2.** Type the following command to show the list of the database entity. The following command is responsible for connecting to the server and printing a table that lists each table's name within the database.

```
sudo ovsdb-client list-tables
```

```
ovs@admin:~$ sudo ovsdb-client list-tables
Table
-----
Controller
Bridge
Queue
IPFIX
NetFlow
Open_vSwitch
QoS
Port
sFlow
SSL
Flow_Sample_Collector_Set
Mirror
Flow_Table
Interface
AutoAttach
Manager
```

Figure 18. Listing database entities.

Consider the figure above. The command prints all the database entities (i.e., Tables). Each table contains all the components to configure the entity. *Bridge* table contains all the information about the bridges running currently. Port table will list all the information about the ports. Open\_vSwitch contains information about all the tables.

**Step 3.** Type the following command to show the elements of *Bridge* table. The following command prints out all the columns in the bridge table in a list format. Bridge table refers to the configuration for a bridge within an Open vSwitch. `-f list` refers to the list format.

```
sudo ovsdb-client -f list list-columns Bridge
```

Column	Type
name	: "string"
flood_vlans	: ["key": {"maxInteger": 4095, "minInteger": 0, "type": "integer"}, "max": 4096, "min": 0]
auto_attach	: ["key": {"refTable": "AutoAttach", "type": "uuid"}, "min": 0]
ports	: ["key": {"refTable": "Port", "type": "uuid"}, "max": "unlimited", "min": 0]
stp_enable	: "boolean"
rstp_enable	: "boolean"

Figure 19. Listing Bridge table elements.

Consider the figure above. The figure shows the elements of the table, including the type of each column. The first column refers to the bridge name, and the type for the name will be a string value. *Flood\_vlans* refers to the VLAN configuration where VLAN ID can be set up to 4,096 integers, in the range 0 to 4,095.

**Step 4.** Type the following command to show the *Bridge* table from the database in a list format. The following command connects to the server, retrieves all of the data, and prints it as a table series.

```
sudo ovsdb-client -f list dump Bridge
```

```
ovs@admin:~$ sudo ovsdb-client -f list dump Bridge
Bridge table
_uuid      : 8d339383-6d20-496f-8266-4f0cca3ef7e3
auto_attach   : []
controller    : []
datapath_id   : "0000000000000001"
datapath_type : ""
datapath_version: "<unknown>"
external_ids  : {}
fail_mode     : standalone
flood_vlans   : []
name          : "s1"
netflow       : []

_uuid      : ace17ae2-af18-4ccb-bcce-7e18028a8889
auto_attach   : []
controller    : []
datapath_id   : "0000000000000002"
datapath_type : ""
datapath_version: "<unknown>"
external_ids  : {}
fail_mode     : standalone
flood_vlans   : []
name          : "s2"
netflow       : []

_uuid      : 3c862c03-629f-4e5d-a3ea-c20f0425653a
auto_attach   : []
controller    : []
datapath_id   : "0000022c863c5d4e"
datapath_type : ""
datapath_version: "<unknown>"
external_ids  : {}
fail_mode     : []
flood_vlans   : []
name          : "br0"
netflow       : []
```

Figure 20. Printing Open vSwitch data from the server.

Consider the figure above. The figure shows the bridges *br0*, *s1*, and *s2*, including the *name*, *uuid*, *datapath\_id*, and other columns related to the *Bridge* table.

**Step 5.** Type the following command to monitor the port table in real-time.

```
sudo ovsdb-client monitor Port
```

```

ovs@admin:~$ sudo ovsdb-client monitor Port
row                                         action  bond_active_slave bond_downdelay bond
_fake_iface bond_mode bond_updelay cvlans external_ids fake_bridge interfaces
                                lacp mac name      other_config protected qos rstp_statist
ics rstp_status statistics status tag trunks vlan_mode _version
-----
-----
-----
5f6328f7-1eb2-4023-a43b-b24a0e10f8d5 initial []          0      false
e          []          0      []      {}      [5cf90882-abec-
4fc0-92d1-d92b7c0d8bd1] []      []      "s1-eth2" {}
          {}      {}      []      []      []
f06293f52
b47bf50f-b042-46c7-8ba4-6f2ffd210979 initial []
e          []          0      []      {}      [754649aa-4939-
4a72-a27d-6c4d11b2ea42] []      []      "s2-eth1" {}
          {}      {}      []      []      []
88038b3b8
e02f673c-5086-4a31-8c7c-2be7be168915 initial []
e          []          0      []      {}      [52406288-26dd-
449c-9f87-3fc9a7883f89] []      []      "s1-eth1" {}
          {}      {}      []      []      []

```

Figure 21. Monitoring Port table.

Consider the figure above. The command allows to monitor all the ports. Whenever a monitored portion of the database changes, the server notifies the client about the modification.

**Step 6.** Click on the *File* option and select *+ New Tab* or press `Ctrl+Shift+T`.

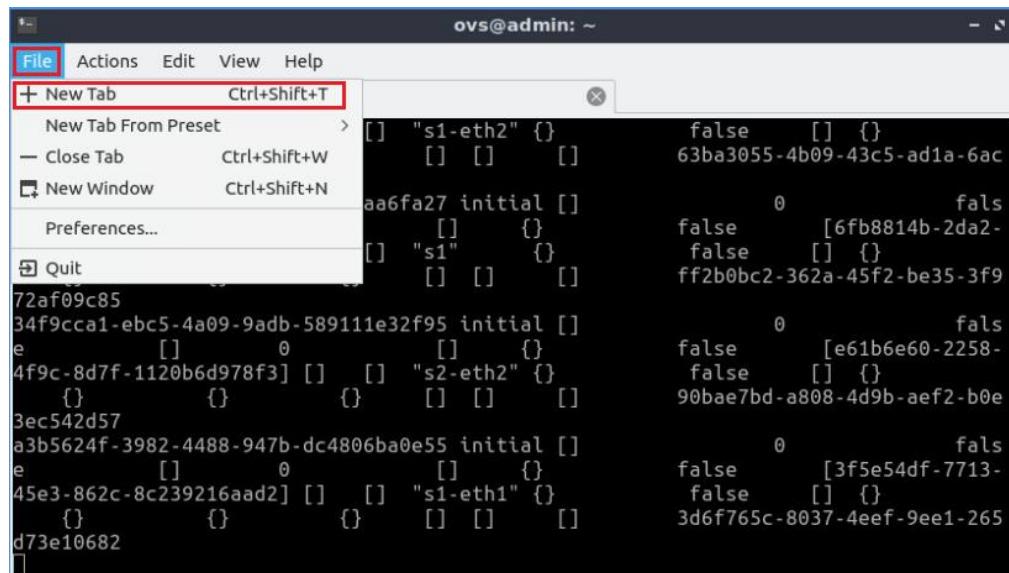


Figure 22. Opening a new terminal tab.

**Step 7.** Type the following command to delete port *s1-eth1*. When prompted for a password, type `password`.

```
sudo ovs-vsctl del-port s1 s1-eth1
```

```
ovs@admin:~$ sudo ovs-vsctl del-port s1 s1-eth1
[sudo] password for ovs:
ovs@admin:~$
```

Figure 23. Deleting a port from the database.

**Step 8.** Go back to the previous terminal. You will see a notification showing that port *s1-eth1* has been deleted.

```
ovs@admin:~$ ovs-vsctl show
row                                     action bond_active_slave bond_downdelay bond_
fake_iface bond_mode bond_updelay cvlans external_ids fake_bridge interfaces
      lacp mac name     other_config protected qos rstp_statisti
cs rstp_status statistics tag trunks vlan_mode _version
-----
-----
```

0fa8251c-2392-4afc-bf40-08595bf7ca16	delete	[]	0	false
ab5-8476-cecd6d65180e	[ ]	[ ]	[ ]	[ ]
a8672eaf	"s1-eth1"	{}	[ ]	[ ]

Figure 24. Monitoring Port table.

**Step 9.** In host h1 terminal, test the connectivity between hosts h1 and h2 using the `ping` command. In host h1, type the command specified below.

```
ping 10.0.0.2
```

```
X                                         "Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1018ms
root@admin:/home/ovs#
```

Figure 25. Output of the `ping` command.

Press `ctrl+c` to stop the test.

Consider the figure above. The figure shows that hosts h1 and h2 are not pingable since port, *s1-eth1* has been deleted.

**Step 10.** In the Linux terminal, type the following command to run a script. The `ovsdb-client` will send raw JSON requests to show the information about the bridges. If required, the password is `password`.

```
sudo ./show_Bridge.cmd
```

```
ovs@admin:~$ sudo ./show_Bridge.cmd
The command is: sudo ovsdb-client -v transact '[{"Open_vSwitch", {"op": "select", "table": "Bridge", "where": []}}]'
The output is:
  Name: br0
    uuid: "4c69834e-aaed-4e2b-a6a7-959ece257e0e"
    "datapath_id": "00004e83694c2b4e",
  Name: s1
    uuid: "f887b461-c931-490d-868e-999508b8c1e8"
    "datapath_id": "0000000000000001",
    fail_mode: standalone
  Name: s2
    uuid: "0f5fc164-1269-4a18-a046-75145b0153a7"
    datapath_id: "0000000000000002"
    fail_mode: standalone
ovs@admin:~$
```

Figure 26. Displaying database information from bridge table.

Consider the figure above. The figure shows all the data stored in the *Bridge*. You will notice the list of bridges *br0*, *s1*, and *s2*, including all the data (*uuid*, *datapath\_id*, *fail\_mode*) stored in the database.

The command that the script `show_Bridge.cmd` executes is:

```
sudo ovsdb-client -v transact '[{"Open_vSwitch", {"op": "select", "table": "Bridge", "where": []}}]' > bridge.json
```

This concludes Lab 10. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. C. Singh, “*Introduction to DBMS*”, BeginnersBook, [Online]. Available: <https://beginnersbook.com/2015/04/dbms-introduction/>
2. Linux foundation, “*Open vSwitch*”, [Online]. Available: <http://openvSwitch.org>
3. RFC-7047, “*The Open vSwitch Database Management Protocol*”, Dec 2013.
4. IBM, “*Virtual networking in Linux*”, [Online]. Available: <https://developer.ibm.com/tutorials/l-virtual-networking/>
5. Mininet walkthrough, [Online]. Available: <http://mininet.org>
6. Open vSwitch Manual, “*Open vSwitch database schema*”, [Online]. Available: <http://www.openvswitch.org//ovs-vswitchd.conf.db.5.pdf>
7. Rikvin, “*Importance of data in your business*”, [Online]. Available: <https://www.rikvin.com/blog/why-data-is-important-to-a-business-performance/>

8. Open vSwitch manual, “*ovsdb-client – command-line interface to ovsdb-server*”, [Online]. Available:  
<http://www.openvswitch.org/support/dist-docs/ovsdb-client.1.txt>



## OPEN VIRTUAL SWITCH

### Lab 11: Open vSwitch Kernel Datapath

Document Version: **09-15-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    Open vSwitch kernel datapath.....	4
1.2    Kernel datapath forwarding process.....	4
2    Lab topology.....	5
2.1    Lab settings.....	6
2.2    Loading a topology .....	6
2.3    Load the configuration file .....	8
3    Visualizing kernel datapath features .....	9
4    Conducting connectivity test between hosts and verifying the flow table .....	11
5    Displaying kernel datapath statistics .....	14
References .....	16

## Overview

This lab introduces Open Virtual Switch (Open vSwitch) datapath, a kernel module that is associated with the flow tables populated from the userspace and responsible for packet forwarding. The lab aims to show how to visualize the flows characteristics using the command-line interface (CLI).

## Objectives

By the end of this lab, you should be able to:

1. Understand the architecture of Open vSwitch.
2. Understand the concept of kernel datapath.
3. Explore kernel datapath features using `ovs-dpctl` tool.
4. Monitor flow installation within the kernel flow table using the CLI.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	Admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Visualizing kernel datapath features.
4. Section 4: Conducting connectivity between hosts and verifying the flow table.
5. Section 5: Displaying kernel datapath statistics.

### 1      Introduction

The increasing number of services based on virtualization presents a significant change in datacenter networking. The migration from physical ports to virtual ports moved a significant part of the workload to virtual switches in a hypervisor. Virtual switches have been developed to improve performance and provide advanced features and the

traditional benefits of software switches. Some of these features include high flexibility, vendor independence, low costs, and conceptual benefits for switching without Ethernet limitations<sup>1</sup>. By consolidating multiple servers and storage devices into a single host machine, virtualization benefits the organization by reducing physical hardware.

## 1.1 Open vSwitch kernel datapath

Open vSwitch includes two modules for packet forwarding. Open vSwitch-vswitchd is a daemon in the userspace that controls the switch and implements the OpenFlow protocol. The other is the datapath kernel module, which runs in the kernel. It is directly connected to Open vSwitch-vswitchd via netlink. `ovs-dpctl` is a tool for configuring the switch kernel module.

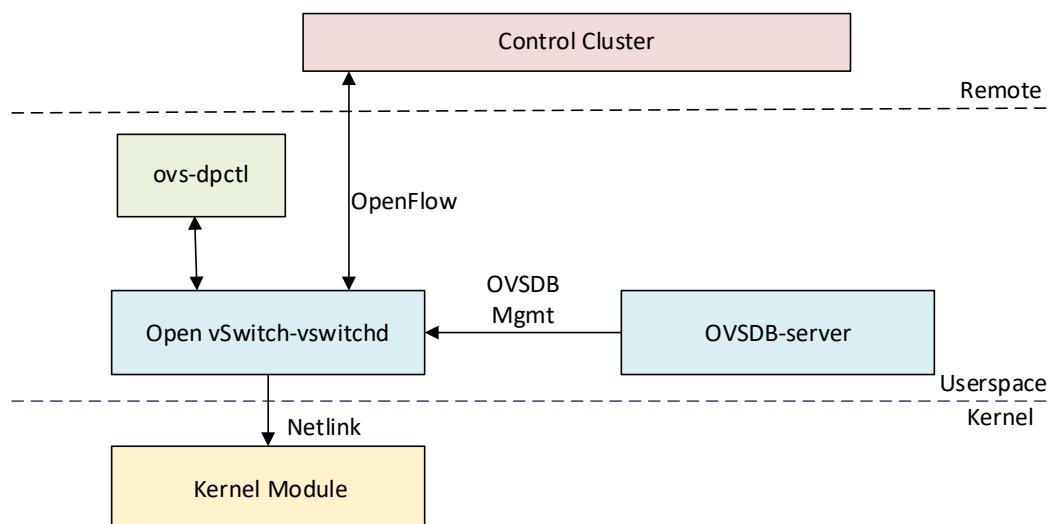


Figure 1. Open vSwitch Architecture.

Figure 1 illustrates the architecture of Open vSwitch. The Open vSwitch-vswitchd and the OVSDB-server lie within the userspace, and the kernel datapath module comes under the kernel space. The kernel module handles functions such as switching and tunneling. When an incoming packet matches an entry in the flow table, the corresponding actions are executed, and the counters are updated. Otherwise, packets are sent to the userspace<sup>4</sup>.

## 1.2 Kernel datapath forwarding process

The datapath is implemented in the kernel module, the primary packet forwarding module of Open vSwitch. The flow table is located in the userspace daemon (Open vSwitch-vswitchd) which is required to make the forwarding decisions. Kernel module also maintains a table known as kernel flow table. Whenever the kernel module receives a frame from an interface, it performs a lookup on its flow table to determine its actions. If the packet does not match any entry in the kernel flow table, the datapath sends the packet to the Open vSwitch-vswitchd. The userspace makes the decision about the actions to be taken against the packet according to OpenFlow entries stored in the flow table. It sends the packet back to the kernel with a list of actions. This process is known as the slow path. The kernel module is instructed to make the same list of actions

whenever a similar flow packet comes in. The action entry is stored in the kernel flow table, which is used to forward subsequent packets, making the forwarding faster (fast path). The kernel module can implement multiple datapaths<sup>8</sup>.

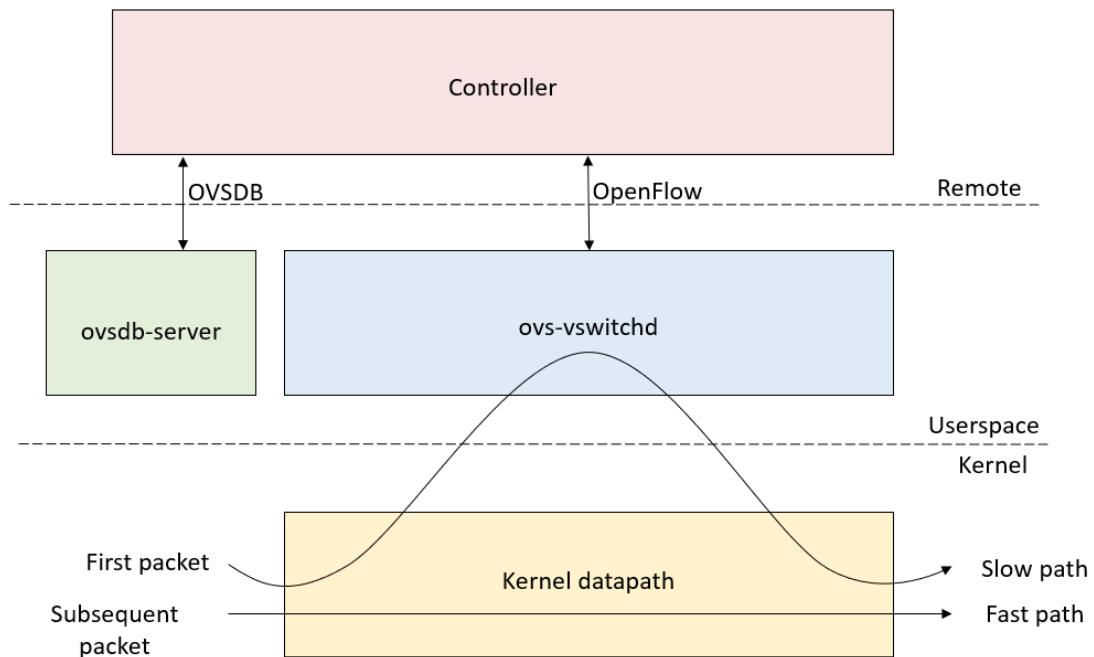


Figure 2. Kernel datapath forwarding.

Figure 2 depicts how Open vSwitch components work together to forward packets. The datapath module receives the packets from a physical network interface controller (NIC) or the virtual NIC of a virtual machine (VM). Since there is no flow entry in the flow table initially, the kernel module directs the packet to the userspace, which sends the forwarding decision for subsequent packets into the kernel.

## 2 Lab topology

Consider Figure 3. Two switches are connected to each other. Each switch is connected to a host. All the hosts belong to the network 10.0.0.0/8.

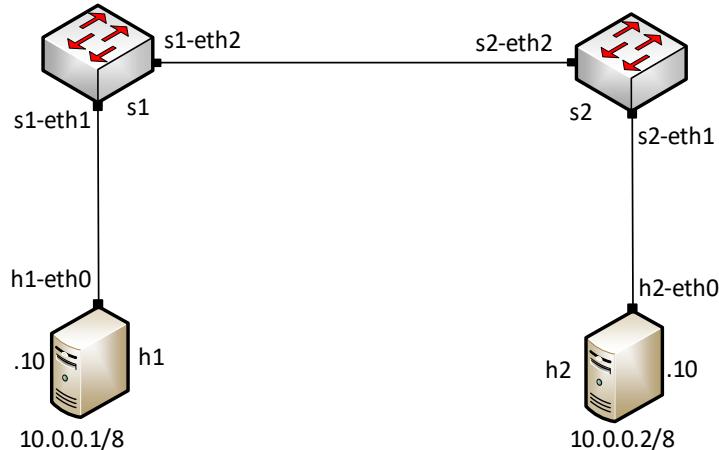


Figure 3. Lab topology.

## 2.1 Lab settings

The hosts are configured according to Table 2.

Table 2. Topology information.

Host	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8

## 2.2 Loading a topology

**Step 1.** Start by launching MiniEdit by clicking on the desktop's shortcut. When prompted for a password, type `password`.

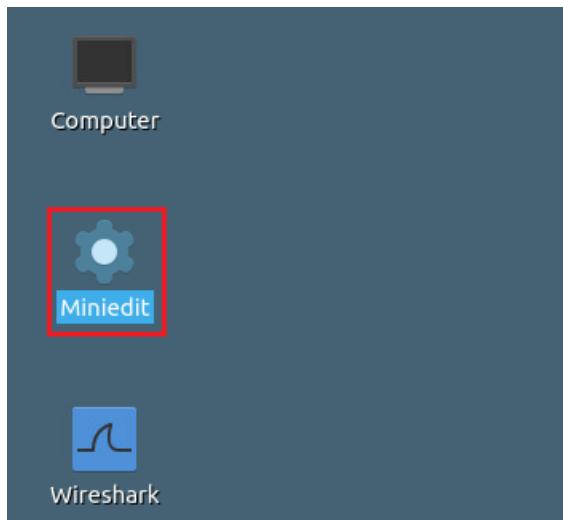


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then, open to load the lab's topology. Locate the *lab11.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab11* and click on *Open*.

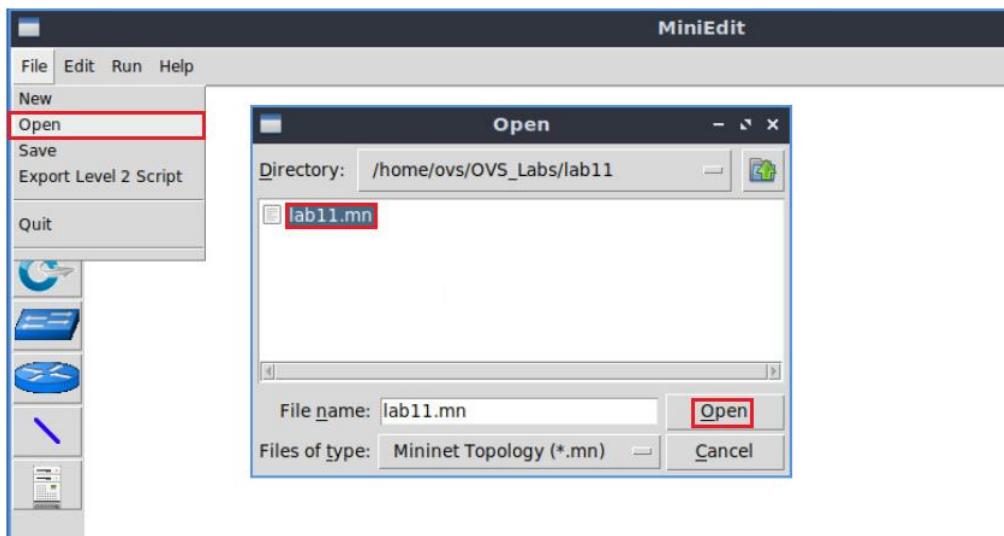


Figure 5. MiniEdit's Open dialog.

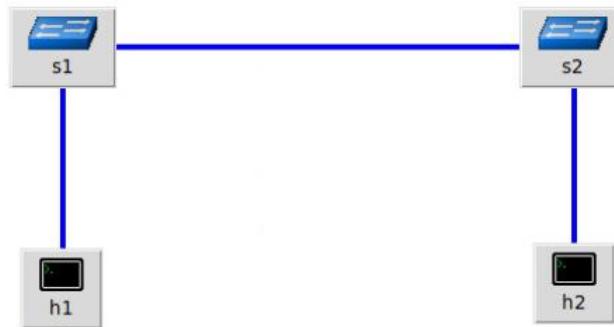


Figure 6. MiniEdit's topology.

**Step 3.** To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 7. Starting the emulation.

**Step 4.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

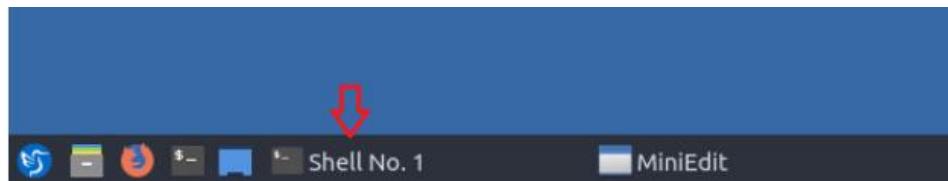
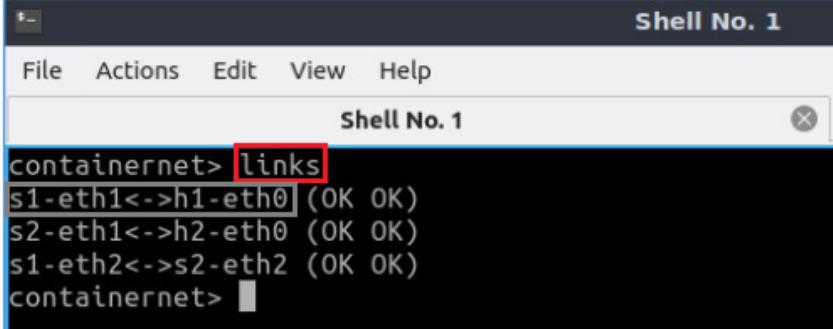


Figure 8. Opening Mininet's terminal.

**Step 5.** Issue the following command to display the interface names and connections.



```
links
Shell No. 1
File Actions Edit View Help
Shell No. 1
containernet> links
s1-eth1<->h1-eth0 (OK OK)
s2-eth1<->h2-eth0 (OK OK)
s1-eth2<->s2-eth2 (OK OK)
containernet>
```

Figure 9. Displaying network interfaces.

In Figure 9, the link displayed within the gray box indicates that interface *eth1* of switch *s1* connects to interface *eth0* of host *h1* (i.e., *s1-eth1<->h1-eth0*).

### 2.3 Load the configuration file

**Step 1.** Open the Linux terminal.

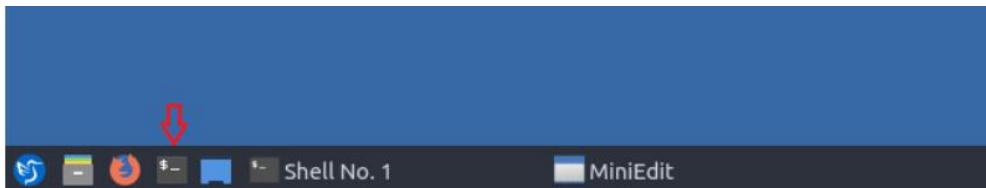
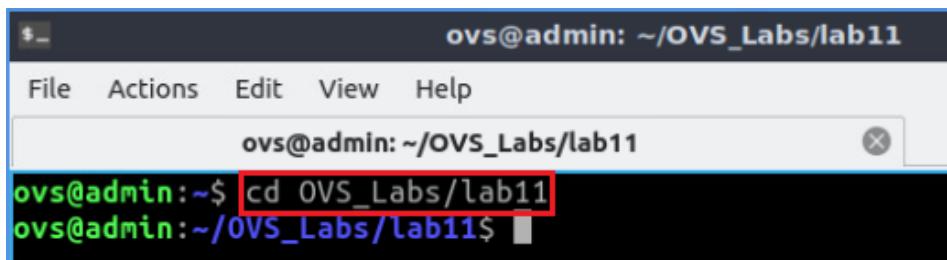


Figure 10. Opening Linux terminal.

**Step 2.** Click on the Linux's terminal and navigate into *OVS\_Labs/lab11* directory by issuing the following command.

```
cd OVS_Labs/lab11
```

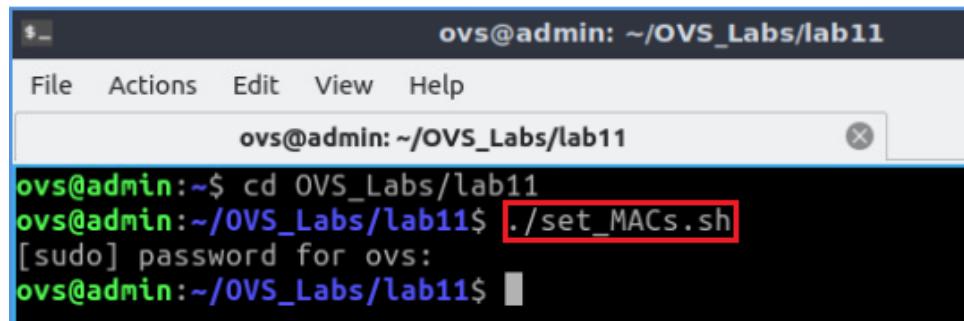


```
ovs@admin: ~/OVS_Labs/lab11
File Actions Edit View Help
ovs@admin: ~/OVS_Labs/lab11
ovs@admin:~$ cd OVS_Labs/lab11
ovs@admin:~/OVS_Labs/lab11$
```

Figure 11. Entering to the *OVS\_Labs/lab11* directory.

**Step 3.** This folder contains a configuration file that will assign easy-to-read Media Access Control (MAC) addresses to the hosts' interfaces. To execute the shell script, type the following command. When prompted for a password, type `password`.

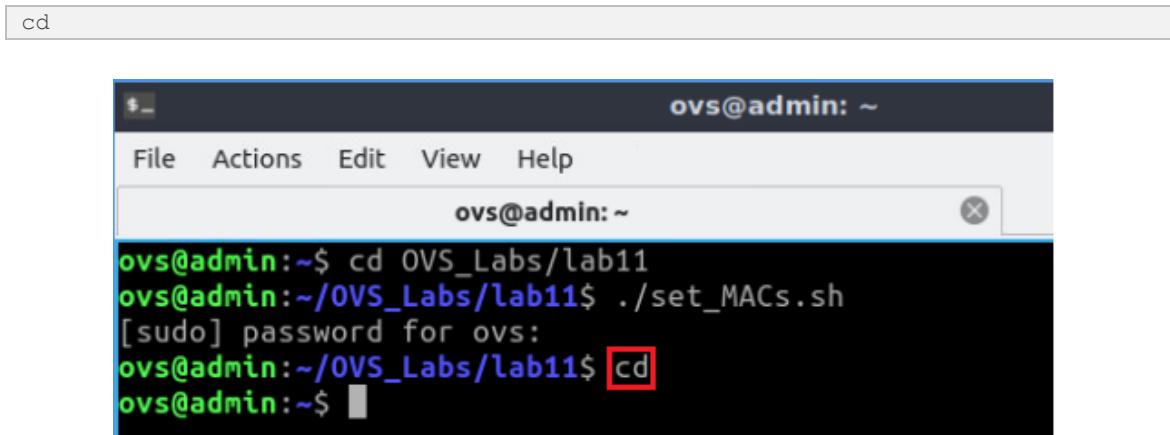
```
./set_MACs.sh
```



```
ovs@admin: ~/OVS_Labs/lab11
File Actions Edit View Help
ovs@admin: ~/OVS_Labs/lab11
ovs@admin:~$ cd OVS_Labs/lab11
ovs@admin:~/OVS_Labs/lab11$ ./set_MACs.sh
[sudo] password for ovs:
ovs@admin:~/OVS_Labs/lab11$
```

Figure 12. Executing the shell script to load the configuration.

**Step 4.** Type the following command to exit from the lab11 directory and go back to the home directory.



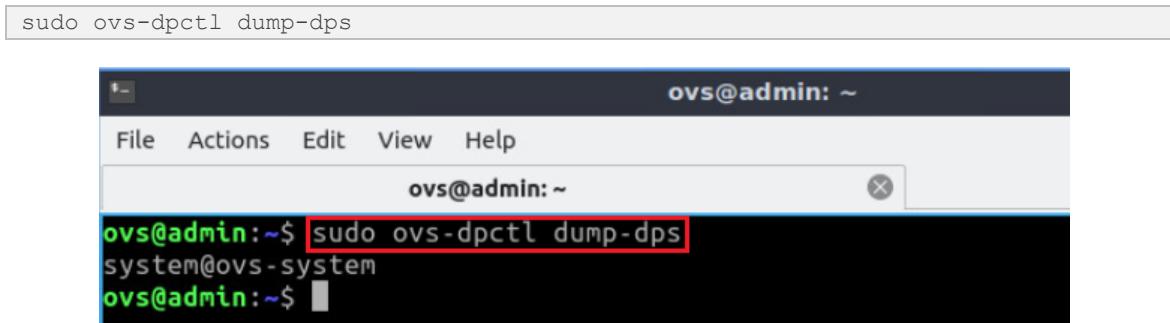
```
cd
$ ...
ovs@admin: ~
File Actions Edit View Help
ovs@admin:~
ovs@admin:~$ cd OVS_Labs/lab11
ovs@admin:~/OVS_Labs/lab11$ ./set_MACs.sh
[sudo] password for ovs:
ovs@admin:~/OVS_Labs/lab11$ cd
ovs@admin:~$
```

Figure 13. Exiting from the lab11 directory.

### 3 Visualizing kernel datapath features

In this section, you will visualize some of the kernel datapath features.

**Step 1.** Type the following command to show available datapaths.



```
sudo ovs-dpctl dump-dps
$ ...
ovs@admin: ~
File Actions Edit View Help
ovs@admin:~
ovs@admin:~$ sudo ovs-dpctl dump-dps
system@ovs-system
ovs@admin:~$
```

Figure 14. Displaying kernel datapath.

Consider the figure above. You will notice the datapath called *system@ovs-system*, which is the default in Open vSwitch and shared by all the bridges. The datapath holds the kernel flow table.

**Step 2.** Type the following command to show all the datapath features for switch s1.

```
sudo ovs-appctl dpif/show-dp-features s1
```

```
ovs@admin:~$ sudo ovs-appctl dpif/show-dp-features s1
Masked set action: Yes
Tunnel push pop: No
Ufid: Yes
Truncate action: Yes
Clone action: Yes
Sample nesting: 10
Conntrack eventmask: Yes
Conntrack clear: Yes
Max dp_hash algorithm: 0
Check pkt length action: No
Max VLAN headers: 2
Max MPLS depth: 1
Recirc: Yes
CT state: Yes
CT zone: Yes
CT mark: Yes
CT label: Yes
CT state NAT: Yes
```

Figure 15. Displaying all datapath features.

Consider the command above. The output of the command shows the features of the datapath including their state. The first feature is *Masked set action* which allows modification of an arbitrary subset of the header bits without requiring the rest to be matched. The second feature refers to *tunneling*, which is disabled at this moment. The third one refers to a unique flow ID (*Ufid*) for each flow in the kernel.

**Step 3.** Type the following command to verify the flow installation. This command prints the flow table entries in switch s1.

```
sudo ovs-ofctl dump-flows s1
```

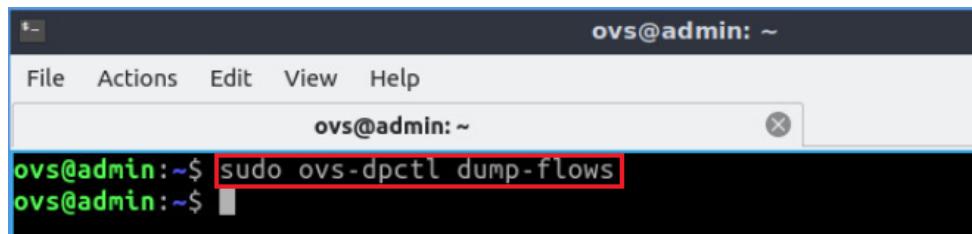
```
ovs@admin:~$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=29.543s, table=0, n_packets=18, n_bytes=2551, priority=0 actions=NORMAL
ovs@admin:~$
```

Figure 16. Verifying flow in switch s1.

Consider the figure above. A normal action allows the device to conduct normal layer 2/layer 3 packet processing. All the flows will match at this point.

**Step 4.** Type the following command to verify the flow installation in the kernel.

```
sudo ovs-dpctl dump-flows
```



```
ovs@admin:~$ sudo ovs-dpctl dump-flows
ovs@admin:~$
```

Figure 17. Verifying flow entry in the kernel.

Consider the figure above. Initially, the flow table will be empty since the hosts did not exchange any packet yet.

#### 4 Conducting connectivity test between hosts and verifying the flow table

In this section, you will perform a connectivity test between the hosts and verify the kernel flow table again. New flows will be added to the flow table once the hosts start exchanging packets.

**Step 1.** Hold right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

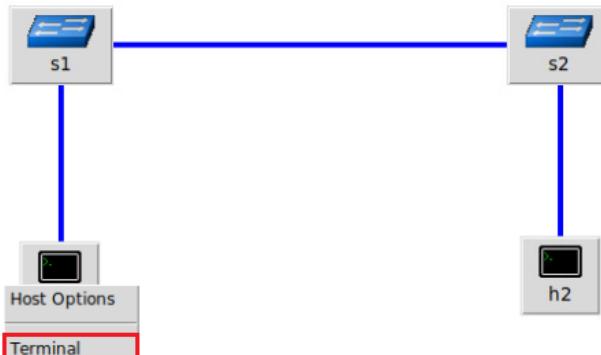


Figure 18. Opening a terminal on host h1.

**Step 2.** Type the following command to verify IP address, subnet mask, and MAC address on host h1.

```
ifconfig
```

```
"Host: h1"
root@admin:/home/ovs# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0
          ether 00:00:00:00:01 txqueuelen 1000 (Ethernet)
            RX packets 47 bytes 6288 (6.2 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 3 bytes 270 (270.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
          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

root@admin:/home/ovs# "
```

Figure 19. Verifying IP address and subnet mask in the host.

Consider the figure above. Interface *h1-eth0* has an IP address 10.0.0.1/8 and a MAC address 00:00:00:00:01.

**Step 3.** Type the following command to verify IP address, subnet mask, and MAC address on host h2.

```
ifconfig
```

```
"Host: h2"
root@admin:/home/ovs# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.0.2 netmask 255.0.0.0 broadcast 0.0.0.0
          ether 00:00:00:00:00:02 txqueuelen 1000 (Ethernet)
            RX packets 46 bytes 6178 (6.1 KB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 3 bytes 270 (270.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
          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

root@admin:/home/ovs# "
```

Figure 20. Verifying IP address and subnet mask in the host.

Consider the figure above. Interface *h2-eth0* has an IP address 10.0.0.2/8 and a MAC address 00:00:00:00:02.

**Step 4.** Test the connectivity between hosts h1 and h2 using the `ping` command. On host h1, type the command specified below. The following figure shows a successful connectivity between two hosts. Do not stop the test.

```
ping 10.0.0.2
```

```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.933 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.096 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.120 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.077 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.067 ms
```

Figure 21. Output of the `ping` command.

**Step 5.** Type the following command to print a summary of configured datapaths.

```
sudo ovs-dpctl show
```

```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ovs-dpctl show
system@ovs-system:
  lookups: hit:75 missed:50 lost:0
  flows: 8
  masks: hit:139 total:2 hit/pkt:1.11
  port 0: ovs-system (internal)
  port 1: s1-eth2
  port 2: s1-eth1
  port 3: s1 (internal)
  port 4: s2-eth2
  port 5: s2-eth1
  port 6: s2 (internal)
ovs@admin:~$
```

Figure 22. Printing a summary of the datapath.

Consider the figure above. The figure shows that eight flows are added to the flow table. You may also notice that each interface has a port number assigned by the kernel.

You may observe a different number of flows.

**Step 6.** On the Linux terminal, type the following command to verify the flow installation in the kernel.

```
sudo ovs-dpctl dump-flows
```

```

ovs@admin:~$ sudo ovs-dpctl dump-flows
recirc_id(0),in_port(5),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:01),eth_type(0x0800),ipv4(frag=no), packets:10, bytes:980, used:0.316s, actions:3
recirc_id(0),in_port(6),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:02),eth_type(0x0800),ipv4(frag=no), packets:10, bytes:980, used:0.316s, actions:4
recirc_id(0),in_port(3),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:02),eth_type(0x0800),ipv4(frag=no), packets:10, bytes:980, used:0.316s, actions:5
recirc_id(0),in_port(4),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:01),eth_type(0x0800),ipv4(frag=no), packets:10, bytes:980, used:0.316s, actions:6
recirc_id(0),in_port(3),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:02),eth_type(0x0806), packets:1, bytes:42, used:5.437s, actions:5
recirc_id(0),in_port(6),eth(src=00:00:00:00:00:01,dst=00:00:00:00:00:02),eth_type(0x0806), packets:0, bytes:0, used:never, actions:4
recirc_id(0),in_port(5),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:01),eth_type(0x0806), packets:1, bytes:42, used:5.437s, actions:3
recirc_id(0),in_port(4),eth(src=00:00:00:00:00:02,dst=00:00:00:00:00:01),eth_type(0x0806), packets:1, bytes:42, used:5.437s, actions:6
ovs@admin:~$ 

```

Figure 23. Verifying flow entry in the kernel.

Consider the figure above. Initially, the kernel flow table was empty, thus, the incoming packet took the slow path. Consequently, flows are sent from the userspace and added to the kernel flow table. The first flow refers to the packets coming from port 5 (*s2-eth1*) having the source MAC address 00:00:00:00:00:02 and destination MAC address 00:00:00:00:00:01. The `eth_type=0x0800` refers to IP packet whereas `eth_type=0x0806` refers to Address Resolution Protocol (ARP) packet.

Open vSwitch deletes kernel flow entries whenever many flows are coming. If you cannot see the flows, perform the connectivity test, and verify the flow table again.

**Step 7.** In host h1 terminal, press `Ctrl+c` to stop the test.

## 5 Displaying kernel datapath statistics

In this section, you will use iPerf3 to generate network traffic between hosts h1 and h2 and inspect some kernel datapath statistics. iPerf3 is a tool for active measurements that supports various features, such as establishing a Transmission Control Protocol (TCP) or User Datagram Protocol (UDP), specifying the sending rate, and launching parallel flows<sup>10</sup>.

**Step 1.** On host h2 terminal, type the following command to run the host in server mode.

```
iperf3 -s
```

```

root@admin:~# iperf3 -s
-----
[  0.00]   0.00 MB/s  0% lost
-----
```

Figure 24. Running host h2 in server mode.

Consider the figure above. The figure shows that host h2 is acting as a server and listening to port 5201.

**Step 2.** In host h1 terminal, type the following command to run an iperf3 test between host h1 and h2, host h1 is running in client mode.

```
iperf3 -c 10.0.0.2
```

```
"Host: h1"
root@admin:/home/ovs# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 7] local 10.0.0.1 port 48138 connected to 10.0.0.2 port 5201
[ ID] Interval      Transfer     Bitrate      Retr  Cwnd
[ 7]  0.00-1.00  sec  5.30 GBytes  45.6 Gbits/sec   0  4.22 MBytes
[ 7]  1.00-2.00  sec  5.05 GBytes  43.4 Gbits/sec   0  4.22 MBytes
[ 7]  2.00-3.00  sec  5.09 GBytes  43.7 Gbits/sec   0  4.22 MBytes
[ 7]  3.00-4.00  sec  5.08 GBytes  43.6 Gbits/sec   0  4.22 MBytes
[ 7]  9.00-10.00 sec  5.10 GBytes  43.8 Gbits/sec   0  4.46 MBytes
[ ID] Interval      Transfer     Bitrate      Retr
[ 7]  0.00-10.00 sec  51.0 GBytes  43.8 Gbits/sec   0
[ 7]  0.00-10.00 sec  51.0 GBytes  43.8 Gbits/sec   0
sender
receive
iperf Done.
root@admin:/home/ovs#
```

Figure 25. Running host h1 in client mode.

**Step 3.** On the Linux terminal, Type the following command to print a summary of the configured datapaths.

```
sudo ovs-dpctl show
```

```
ovs@admin:~$ sudo ovs-dpctl show
system@ovs-system:
    lookups: hit:4594012 missed:59 lost:0
    flows: 10
    masks: hit:8402720 total:4 hit/pkt:1.83
    port 0: ovs-system (internal)
    port 1: s1-eth2
    port 2: s1-eth1
    port 3: s1 (internal)
    port 4: s2-eth2
    port 5: s2-eth1
    port 6: s2 (internal)
ovs@admin:~$
```

Figure 26. Printing a summary of the datapath.

Consider the figure above. The figure shows the number of flows in the kernel flow table. The *hit* tag refers to the number of packets that matched against existing flows. The

*missed* tag refers to the number of packets not matching any existing flow and requires userspace processing. The *lost* tag refers to the number of packets destined for userspace process but subsequently dropped before reaching userspace.

You may observe a different number of flows since Open vSwitch deletes kernel flow entries whenever it encounters many flows.

This concludes Lab 11. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. P. Emmerich, D. Raumer, F. Wohlfart, G. Carle, “*Performance characteristics of virtual switching*”, IEEE 3<sup>rd</sup> International conference on cloud networking (CloudNet), 2014.
2. B. Munch, “*Hype cycle for networking and communications*”, Jul 2013.
3. Linux foundation, “*Open vSwitch*”, [Online]. Available: <http://openvSwitch.org>.
4. B. Pfaff, B. Davie, Ed, “*The open vSwitch database management protocol*”, RFC 7047, Dec 2013.
5. IBM, “*Virtual networking in Linux*”, [Online]. Available: <https://developer.ibm.com/tutorials/l-virtual-networking/>
6. B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, S. Shenker, “*Extending networking into the virtualization layer*”, Jan 2009.
7. Mininet walkthrough, [Online]. Available: <http://mininet.org>.
8. Open vSwitch manual, “*ovs-dpctl – administer Open vSwitch datapaths*”, [Online]. Available: <http://www.openvswitch.org/support/dist-docs/ovs-dpctl.8.txt>
9. Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, “*The design and implementation of Open vSwitch*”, USENIX association, May 2015.
10. iPerf, “*iPerf - The ultimate speed test tool for TCP, UDP and SCTP*”. [Online]. Available: <https://iperf.fr/>



## OPEN VIRTUAL SWITCH

### Lab 12: Implementing Virtual Local Area Networks (VLANs) in Open vSwitch

Document Version: **09-15-2021**



Award 1829698  
“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    VLAN overview .....	4
1.2    VLAN tag architecture .....	4
2    Lab topology.....	5
2.1    Lab settings.....	5
2.2    Loading a topology .....	6
3    Configuring VLANs .....	9
4    Verifying the configuration .....	10
References .....	13

## Overview

This lab introduces Virtual Local Area Network (VLAN), which separates an existing physical network into multiple logical networks. Thus, each VLAN creates its broadcast domain. The lab aims to configure VLAN to isolate network traffic within an emulated environment.

## Objectives

By the end of this lab, you should be able to:

1. Understand the concept of VLAN.
2. Isolate network traffic by setting VLAN tags.
3. Verify VLAN configuration.
4. Analyze the benefits provided of network segmentation using VLANs.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Configuring VLANs.
4. Section 4: Verifying the configuration.

### 1      Introduction

The Local Area Networks (LANs) allow hosts within an organization to be connected together. To ensure the integrity of the information transmitted through the network, the hosts usually rely on IEEE 802.3<sup>1</sup>, known as Ethernet. In modern LANs, the hosts are usually configured according to hierarchy, where each group of hosts is attached to a switched LAN. Each switched LAN is interconnected using a switch that is upper in the

hierarchy. However, this approach presents issues such as the lack of traffic isolation, inefficient use of switches, and managing users more difficult<sup>2</sup>.

## 1.1 VLAN overview

The issues mentioned above are mitigated if the switch supports VLANs<sup>3</sup>. This feature supports multiple VLANs to be created by using a single LAN infrastructure. Thus, hosts in the same VLAN can communicate as if they are connected to a single switch. VLAN traffic isolation allows broadcasting packets such as the ones used by the Address Resolution Protocol (ARP) or the Dynamic Host Configuration Protocol (DHCP) to be no longer required to traverse the whole network, which improves performance and security.

Consider Figure 1. Switch s1 is directly attached to four hosts configured to handle the traffic for two VLANs, VLAN 10 and VLAN 20. These two VLANs are isolated from each other, although they are connected to the same switch and belong to the same network 10.0.0.0/24. The network manager can add or change a host's membership by changing the VLAN tag in switch s1.

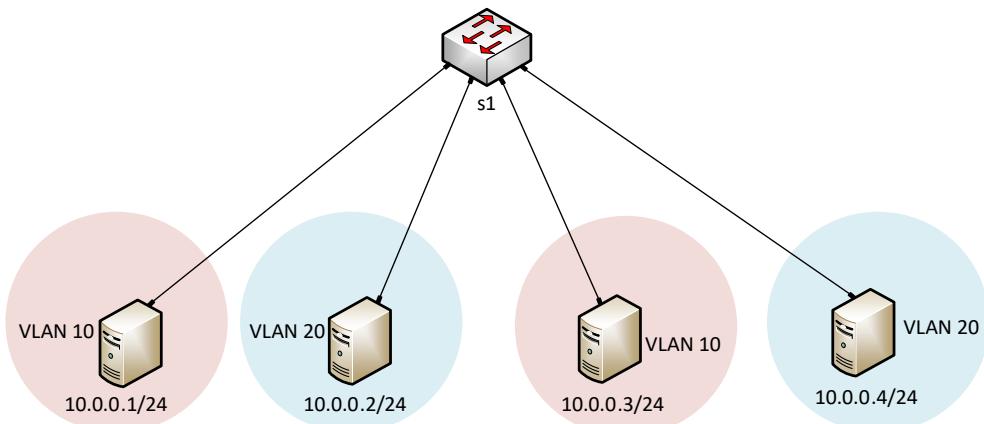


Figure 1. Defining VLAN group.

## 1.2 VLAN tag architecture

The protocol IEEE 802.1Q<sup>1</sup> defines the VLAN frame. The VLAN tag is 4-bytes long and is directly added to the Ethernet frame header. This tag contains the information that associates each frame with a specific VLAN. Additionally, there might be cases where a frame does not belong to any VLAN. In such a situation, the switch associates those packets to default VLAN.

The 4-bytes VLAN tag field is composed of the following fields:

**Type:** A 2-byte value called the tag protocol ID (TPID) value.

**User priority:** A 3-bit value that supports level or service implementation.

**Canonical Format Identifier (CFI):** A 1-bit identifier that indicates whether the VLAN identifier conforms to Ethernet or not.

**VLAN ID (VID):** A 12-bit VLAN identification number that supports up to 4096 VLAN IDs.

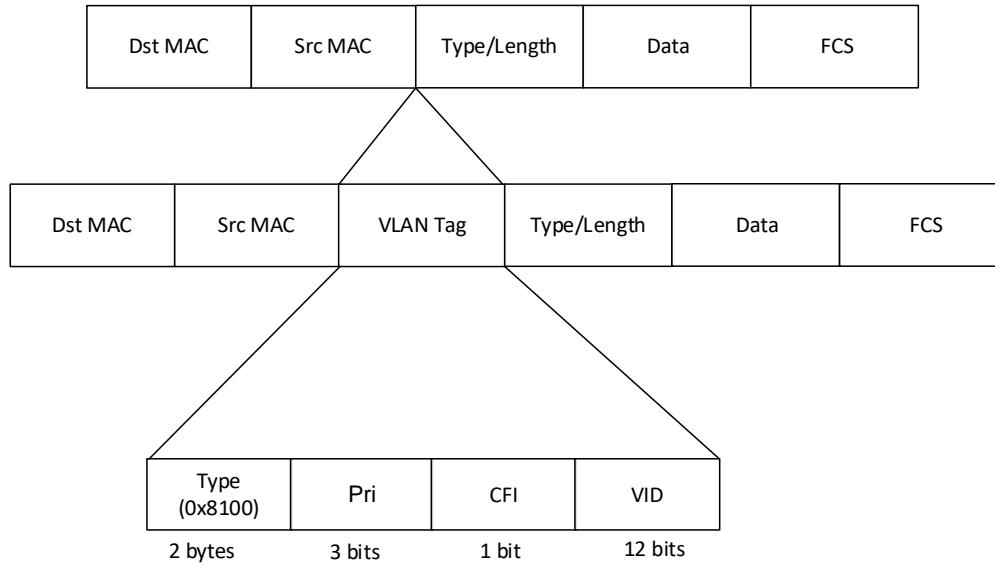


Figure 2. Ethernet 802.1Q frame.

Figure 2 shows the fields of a VLAN tag added to an Ethernet frame.

## 2 Lab topology

Consider Figure 3. The topology consists of four hosts and one switch. All the hosts belong to the network, 10.0.0.0/24. Two different VLANs (VLAN 10 and VLAN 20) are running on the network to isolate the network traffic.

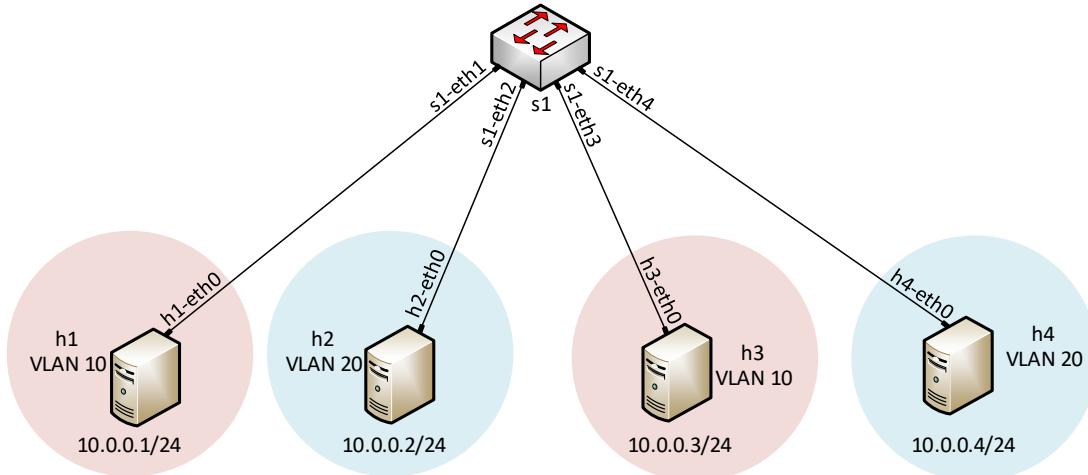


Figure 3. Lab topology.

### 2.1 Lab settings

The hosts are configured according to Table 2.

Table 2. Topology information.

Host	Interface	IP Address	Subnet	VLAN
h1	h1-eth0	10.0.0.1	/24	10
h2	h2-eth0	10.0.0.2	/24	20
h3	h3-eth0	10.0.0.3	/24	10
h4	h4-eth0	10.0.0.4	/24	20

## 2.2 Loading a topology

**Step 1.** Start by launching MiniEdit by clicking on the desktop's shortcut. When prompted for a password, type `password`.

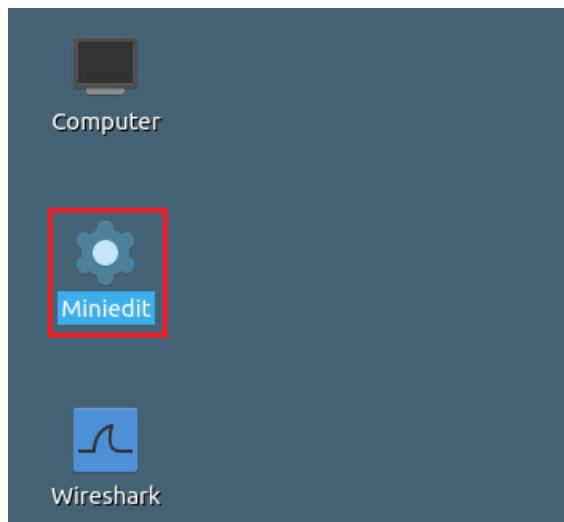


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File*, then *open* to load the lab's topology. Locate the *lab12.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab12* and click on *Open*.

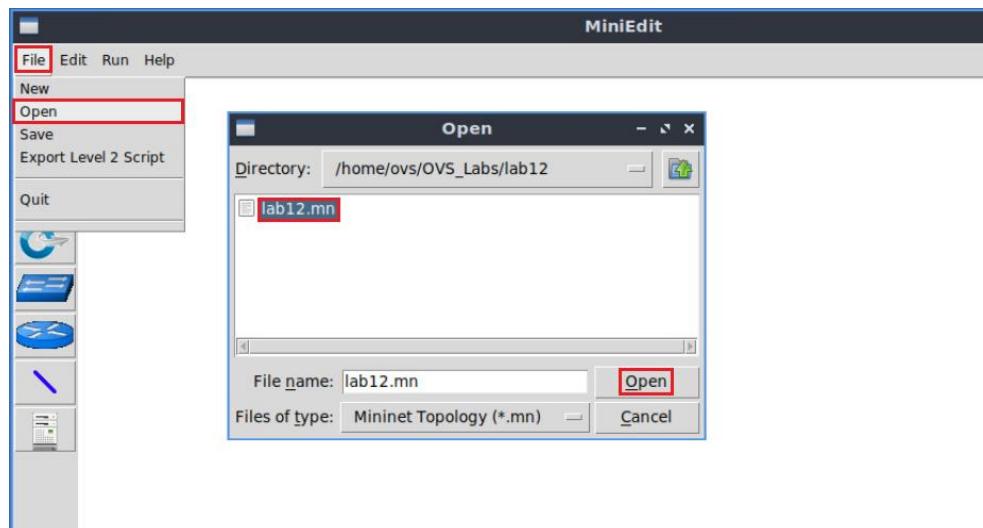


Figure 5. MiniEdit's opening dialog

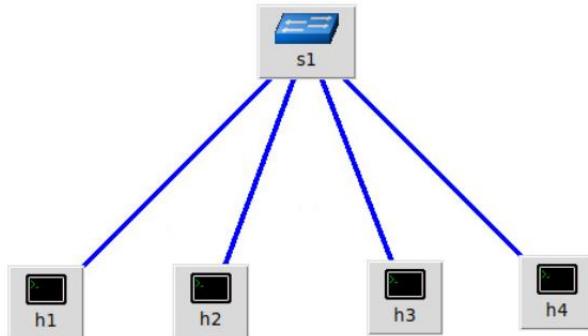


Figure 6. MiniEdit's topology.

**Step 3.** To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 7. Starting the emulation.

**Step 4.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

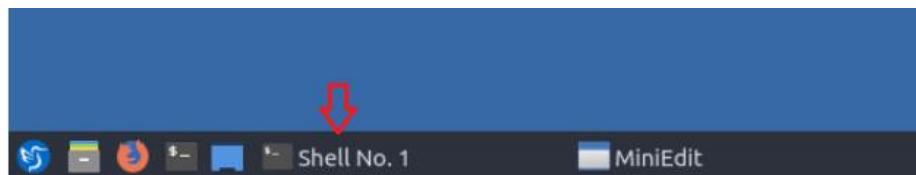


Figure 8. Opening Mininet's terminal.

**Step 5.** Issue the following command to display the interface names and connections.

links

```
containernet> links
h1-eth0<->s1-eth1 (OK OK)
s1-eth2<->h2-eth0 (OK OK)
s1-eth3<->h3-eth0 (OK OK)
s1-eth4<->h4-eth0 (OK OK)
containernet>
```

Figure 9. Displaying network interfaces.

In Figure 9, the link displayed within the gray box indicates that interface *eth0* of host *h1* is connected to interface *eth1* of switch *s1* (i.e., *h1-eth0<->s1-eth1*).

**Step 6.** Hold right-click on host *h1* and select *Terminal*. This opens the terminal of host *h1* and allows the execution of commands on that host.

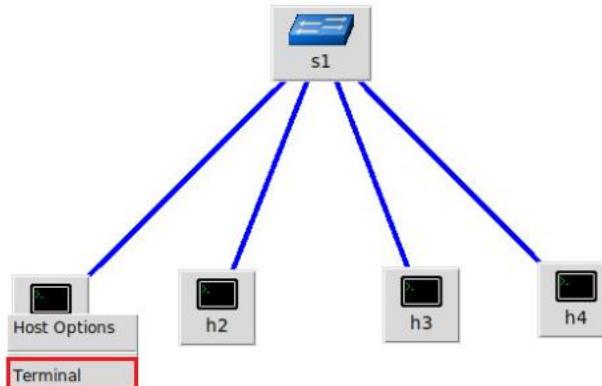
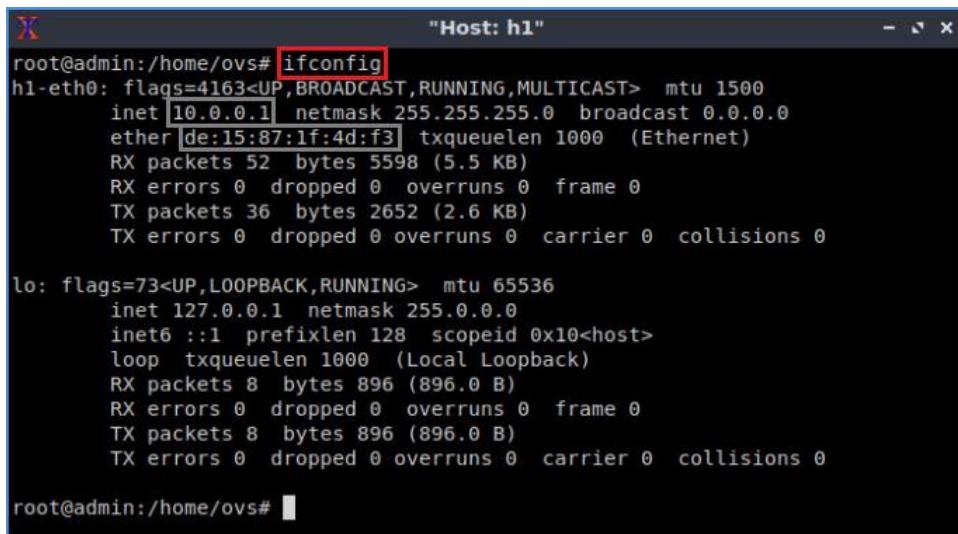


Figure 10. Opening a terminal on host *h1*.

**Step 7.** In host *h1* terminal, type the following command to verify that the IP address was assigned successfully. You will verify the host interface, *h1-eth0* configured with the IP address 10.0.0.1 and the subnet mask 255.255.255.0. You will also verify the MAC address, de:15:87:1f:4d:f3.

ifconfig



```
"Host: h1"
root@admin:/home/ovs# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.0.1 netmask 255.255.255.0 broadcast 0.0.0.0
                ether de:15:87:1f:4d:f3 txqueuelen 1000 (Ethernet)
                RX packets 52 bytes 5598 (5.5 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 36 bytes 2652 (2.6 KB)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
            RX packets 8 bytes 896 (896.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 8 bytes 896 (896.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@admin:/home/ovs#
```

Figure 11. Verifying IP address and subnet mask on the host.

You may notice a different MAC address since MAC addresses are assigned randomly.

**Step 8.** Repeat step 6 and step 7 to verify IP addresses on hosts h2, h3 and h4.

### 3 Configuring VLANs

In this section, you will configure VLANs according to the topology information (Table 2). You will configure the switch ports attached to hosts h1 and h3 to VLAN 10 and switch ports attached to hosts h2 and h4 to VLAN 20.

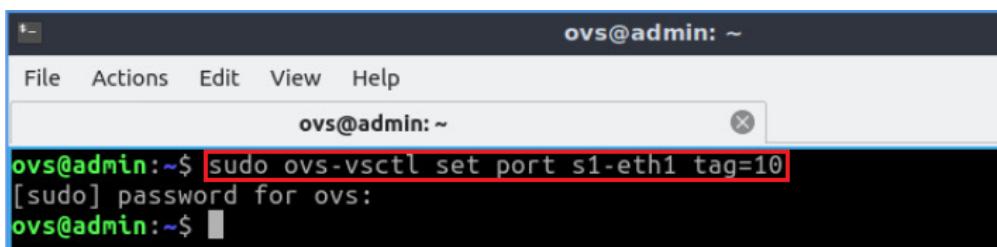
**Step 1.** Open the Linux terminal.



Figure 12. Opening Linux terminal.

**Step 2.** Type the following command to assign port s1-eth1 (connected to host h1) to VLAN 10. When prompted for a password, type `password`.

```
sudo ovs-vsctl set port s1-eth1 tag=10
```



```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ovs-vsctl set port s1-eth1 tag=10
[sudo] password for ovs:
ovs@admin:~$
```

Figure 13. Assigning port s1-eth1 to VLAN 10.

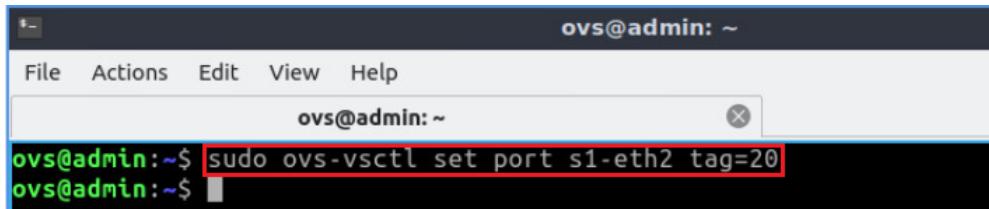
Consider the command above. VLAN tag 10 has been assigned to interface *s1-eth1*.

To remove the tag, you might use the following command:

```
sudo ovs-vsctl remove port s1-eth1 tag 10
```

**Step 3.** Type the following command to assign port *s1-eth2* (connected to host h2) to VLAN 20.

```
sudo ovs-vsctl set port s1-eth2 tag=20
```

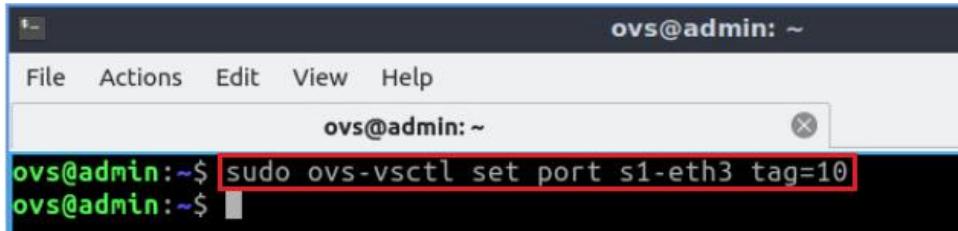


```
ovs@admin:~$ sudo ovs-vsctl set port s1-eth2 tag=20
```

Figure 14. Assigning port *s1-eth2* to VLAN 20.

**Step 4.** Type the following command to assign port *s1-eth3* (connected to host h3) to VLAN 10.

```
sudo ovs-vsctl set port s1-eth3 tag=10
```

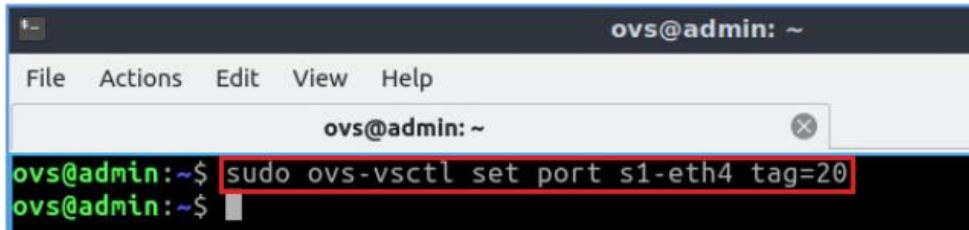


```
ovs@admin:~$ sudo ovs-vsctl set port s1-eth3 tag=10
```

Figure 15. Assigning port *s1-eth3* to VLAN 10.

**Step 5.** Type the following command to assign port *s1-eth4* (connected to host h4) to VLAN 20.

```
sudo ovs-vsctl set port s1-eth4 tag=20
```



```
ovs@admin:~$ sudo ovs-vsctl set port s1-eth4 tag=20
```

Figure 16. Assigning port *s1-eth4* to VLAN 20.

#### 4 Verifying the configuration

In this section, you will verify the VLAN configuration.

**Step 1.** Type the following command to print a brief overview of the database contents.

```
sudo ovs-vsctl show
```

```
ovs@admin:~$ sudo ovs-vsctl show
c34727ea-2286-4c47-baec-a842d33645c9
    Manager "ptcp:6640:127.0.0.1"
    Bridge "s1"
        fail_mode: standalone
        Port "s1-eth4"
            tag: 20
            Interface "s1-eth4"
        Port "s1-eth2"
            tag: 20
            Interface "s1-eth2"
        Port "s1-eth1"
            tag: 10
            Interface "s1-eth1"
        Port "s1"
            Interface "s1"
            type: internal
        Port "s1-eth3"
            tag: 10
            Interface "s1-eth3"
    ovs_version: "2.12.0"
ovs@admin:~$
```

Figure 17. Printing an overview of database contents.

Consider the figure above. Two different domains have been created, and VLAN tags are attached to the interfaces of switch s1.

**Step 2.** Test the connectivity between hosts h1 and h3 using the `ping` command. In host h1, type the command specified below.

```
ping 10.0.0.3
```

```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.554 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.042 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.069 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.067 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3080ms
rtt min/avg/max/mdev = 0.042/0.183/0.554/0.214 ms
root@admin:/home/ovs#
```

Figure 18. Output of the `ping` command.

Press `ctrl+c` to stop the test.

Consider the figure above. The figure shows successful connectivity. Both hosts (h1 and h3) belong to the same VLAN (i.e., VLAN 10).

**Step 3.** Test the connectivity between hosts h1 and h2 using the `ping` command. In host h1, type the command specified below.

```
ping 10.0.0.2
```

```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss
root@admin:/home/ovs# "
```

Figure 19. Output of the `ping` command.

Press `ctrl+c` to stop the test.

Consider the figure above. There is no connectivity between the hosts since they belong to different VLANs.

**Step 4.** Test the connectivity between hosts h2 and h4 using the `ping` command. In host h2, type the command specified below.

```
ping 10.0.0.4
```

```
"Host: h2"
root@admin:/home/ovs# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.609 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.073 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.069 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.065 ms
^C
--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3065ms
rtt min/avg/max/mdev = 0.065/0.204/0.609/0.233 ms
root@admin:/home/ovs# "
```

Figure 20. Output of the `ping` command.

Press `ctrl+c` to stop the test.

Consider the figure above. The figure shows successful connectivity. Both hosts (h2 and h4) belong to the same VLAN (i.e., VLAN 20).

**Step 5.** In the Linux terminal, type the following command to verify the Media Access Control (MAC) address table of switch s1.

```
sudo ovs-appctl fdb/show s1
```

port	VLAN	MAC	Age
3	10	0a:1e:35:16:75:ec	20
1	10	de:15:87:1f:4d:f3	20
2	20	7e:77:ea:e4:2f:fc	2
4	20	5e:40:cd:06:7b:53	2

Figure 21. Verifying MAC addresses on switch s1.

Consider the figure above. The table contains each MAC address/VLAN pair learned by switch s1, along with the port on which it was learned. Aging time defines the period of an entry in the table in seconds. You will notice that each port belongs to a particular VLAN. Ports 1 and 3 are assigned to VLAN 10, whereas the rest of the ports are assigned to VLAN 20.

The MAC table changes every 60 seconds. If all the MAC addresses do not appear in the table initially, repeat the connectivity tests using the `ping` command (step 2 and step 4) and verify the MAC table again.

This concludes Lab 12. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. IEEE Standard, "IEEE 802.3 working group", [Online]. Available: <https://www.ieee802.org/3/>.
2. James F. Kurose, Keith W. Ross, "Computer networking a top-down approach", 6<sup>th</sup> Edition, Mar 2017.
3. IEEE Standard, "IEEE standards for local and metropolitan area networks: virtual bridged local area networks," in *IEEE Std 802.1Q-1998*, March 1999.
4. Juniper Networks, "Bridging and VLANs", Jun 2020.
5. Dhurgham Abdulridha Jawad AL-Khaffaf, "Improving LAN performance based on IEEE802.1Q VLAN switching techniques", Journal of university of babylon, engineering sciences, Vol. (26), No. (1): 2018.
6. Juniper Networks, "Layer 2 Networking", Jun 2020.
7. Linux Foundation, "Open vSwitch", [Online]. Available: <http://openvSwitch.org>.
8. Juniper Networks, "Layer 2 VLANs overview", Dec 2017.
9. IBM, "Archived / Virtual networking in Linux", [Online]. Available: <https://developer.ibm.com/tutorials/l-virtual-networking/>
10. Mininet walkthrough, [Online]. Available: <http://mininet.org>.



## OPEN VIRTUAL SWITCH

### Lab 13: VLAN Trunking in Open vSwitch

Document Version: **09-15-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    VLAN access and trunk mode.....	4
1.2    VLAN trunking .....	4
2    Lab topology.....	5
2.1    Lab settings.....	5
2.2    Loading a topology .....	6
3    Verifying IP addresses on the hosts.....	8
4    Configuring VLANs .....	10
5    Verifying configuration .....	12
References .....	18

## Overview

This lab introduces Virtual Local Area Network (VLAN) trunking, a point-to-point link between two network devices (e.g., switches) that carry frames from more than one VLAN. This lab aims to configure multiple VLANs to isolate network traffic within an emulated environment and verify how trunk ports can differentiate multiple VLANs to forward traffic to the intended destination.

## Objectives

By the end of this lab, you should be able to:

1. Understand the concept of VLAN.
2. Understand the concept of VLAN trunking.
3. Isolate network traffic by using VLAN.
4. Configure an Open vSwitch port to operate as a VLAN trunk.
5. Use Wireshark network analyzer to capture VLAN traffic.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Verifying IP addresses on the hosts.
4. Section 4: Configuring VLANs.
5. Section 5: Verifying configuration.

## 1      Introduction

The switches must have a dedicated port configured to interconnect VLANs. In this way, a more scalable approach is attained for physically distributed networks, but they require to be logically connected. For instance, within a campus network, a department might

operate in different locations. Still, the devices need to be connected to the same VLAN to access the network's features in that department. The standard 802.1Q supports a feature called VLAN trunking to address this requirement.

## 1.1 VLAN access and trunk mode

Ethernet interfaces can be configured either as access ports or trunk ports. An access port can only have a VLAN, and it is required to specify which VLAN will carry the traffic for that interface to configure a port in access mode<sup>11</sup>.

A trunk port can have more than one VLAN configured on an interface. It can carry traffic for multiple VLANs simultaneously.

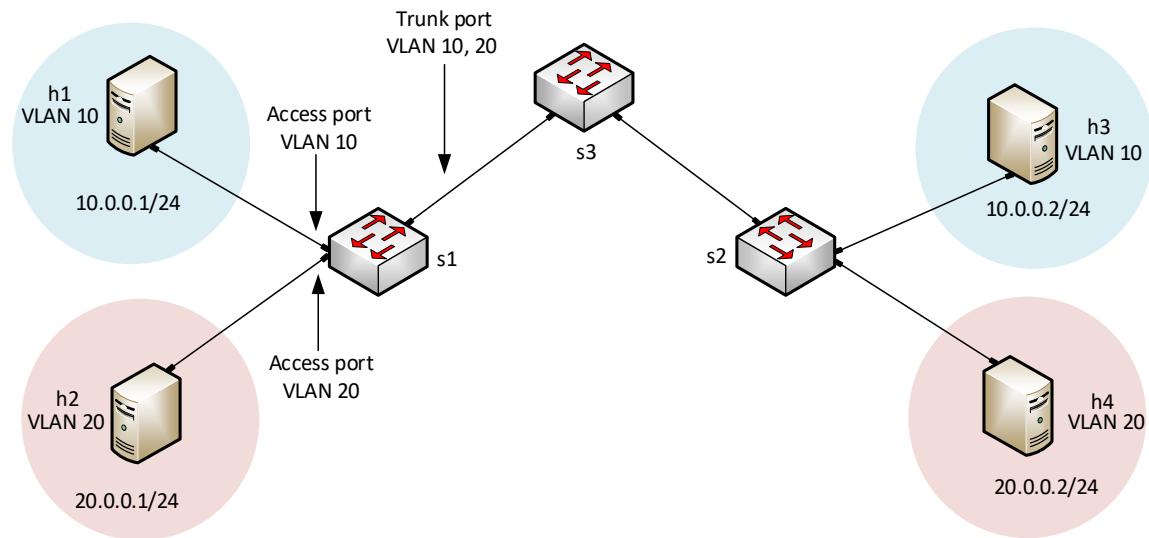


Figure 1. VLAN access and trunk mode.

Figure 1 shows that a single VLAN is configured in access ports, whereas the trunk port can carry VLAN 10 and 20 simultaneously.

## 1.2 VLAN trunking

A VLAN trunk uses a port dedicated to forward frames corresponding to any VLAN so that the VLANs can communicate via a trunking connection. When a switch port is configured to function as a trunk port, it adds unique identification tags defined in the protocol IEEE 802.1Q<sup>2</sup> to the frames to forward traffic between switches. The standard IEEE802.1Q, also referred to as DOT1Q or 1Q, is the networking standard that supports virtual LANs (VLANs) on an IEEE 802.3<sup>3</sup> Ethernet network and, it is the most widely used encapsulation method for VLAN tagging.

Consider Figure 2. Switch s1 and switch s2 are configured with two VLANs (VLAN 10 and VLAN 20). Assume that switches s1 and s2 are in different locations, and they are linked via switch s3. The link that connects switch s1 to switch s3 is configured as a VLAN trunk. Similarly, the link between s3 and s2 is configured as a trunk.

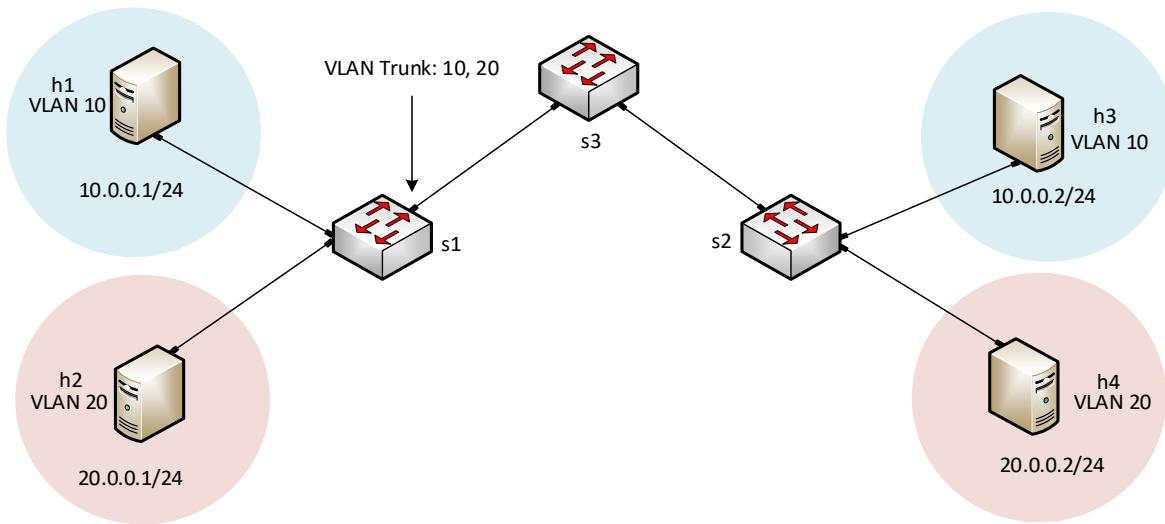


Figure 2. VLAN trunking.

## 2 Lab topology

Consider Figure 3. The topology consists of four hosts and three switches. Hosts h1 and h3 belong to network 10.0.0.0/24 whereas, hosts h2 and h4 belong to network 20.0.0.0/24. Two different VLANs (VLAN 10 and VLAN 20) run on the networks to isolate network traffic.

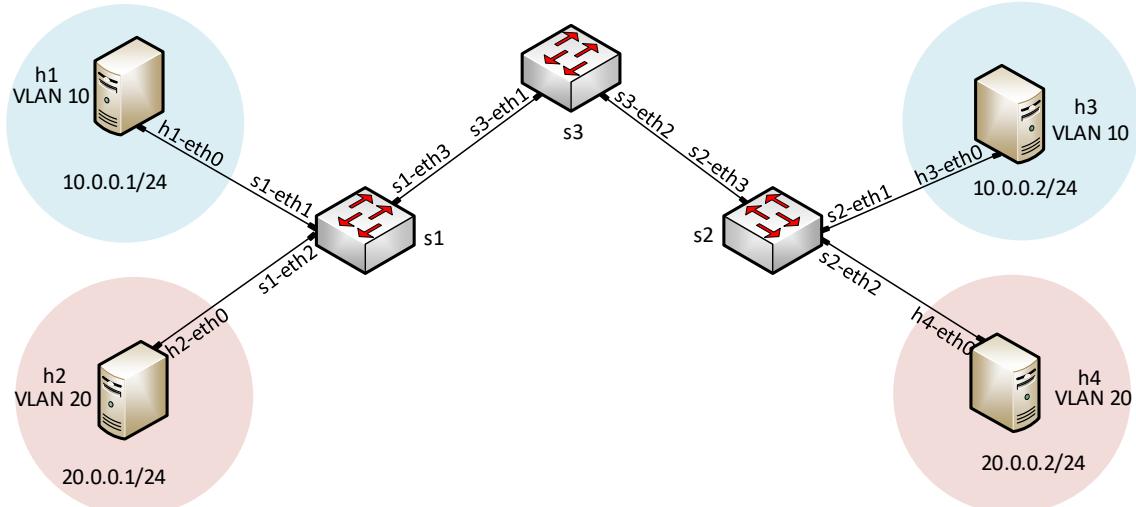


Figure 3. Lab topology.

### 2.1 Lab settings

The hosts are configured according to Table 2.

Table 2. Topology information.

Host	Interface	IP Address	Subnet	VLAN
h1	h1-eth0	10.0.0.1	/24	10
h2	h2-eth0	20.0.0.1	/24	20
h3	h3-eth0	10.0.0.2	/24	10
h4	h4-eth0	20.0.0.2	/24	20

## 2.2 Loading a topology

**Step 1.** Start by launching MiniEdit by clicking on the desktop's shortcut. When prompted for a password, type `password`.

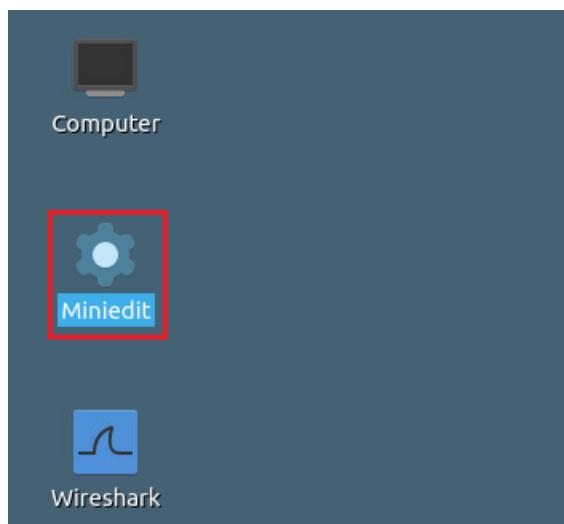


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File*, then *open* to load the lab's topology. Locate *lab13.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab13* and click on *Open*.

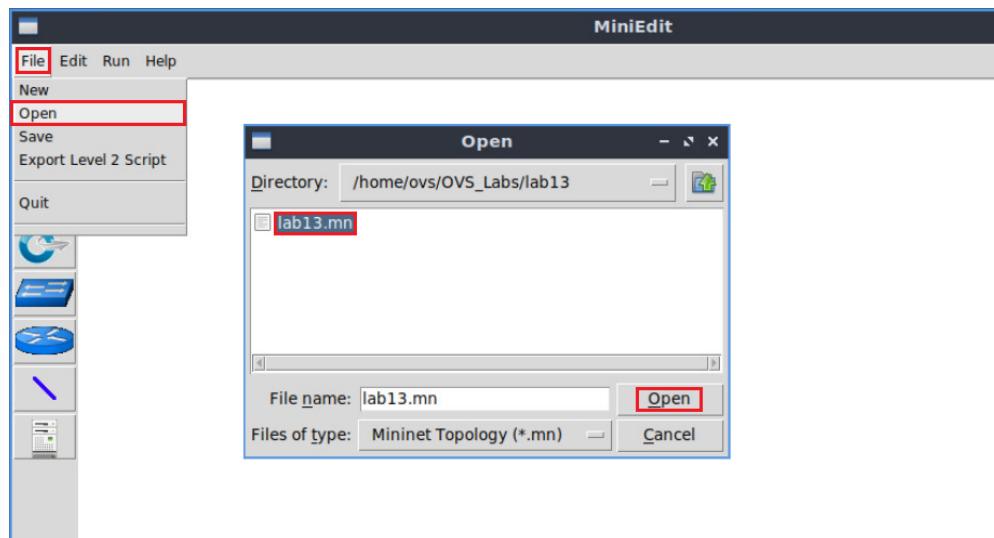


Figure 5. MiniEdit's Open dialog.

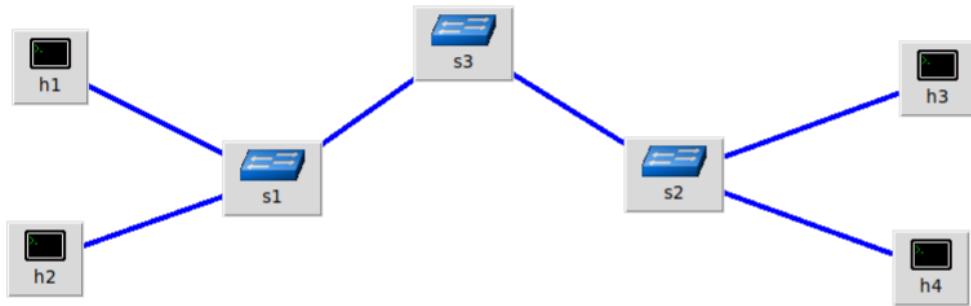


Figure 6. MiniEdit's topology.

**Step 3.** To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 7. Starting the emulation.

**Step 4.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

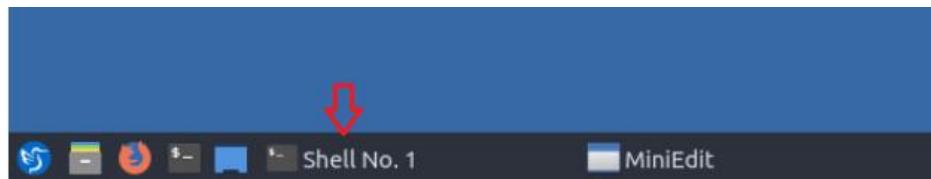


Figure 8. Opening Mininet's terminal.

**Step 5.** Issue the following command to display the interface names and connections.

```
links
```

The screenshot shows a terminal window titled "Shell No. 1". The command "links" is entered, and the output shows various network connections between hosts (h1, h2, h3, h4) and switches (s1, s2, s3). One connection, "h1-eth0<->s1-eth1", is highlighted with a red box. The output is as follows:

```
containernet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
s2-eth1<->h3-eth0 (OK OK)
s2-eth2<->h4-eth0 (OK OK)
s1-eth3<->s3-eth1 (OK OK)
s3-eth2<->s2-eth3 (OK OK)
containernet>
```

Figure 9. Displaying network interfaces.

In Figure 9, the link displayed within the gray box indicates that interface *eth0* of host h1 connects to interface *eth1* of switch s1 (i.e., *h1-eth0<->s1-eth1*).

### 3 Verifying IP addresses on the hosts

In this section, you will verify that IP addresses on the hosts are assigned according to table 2.

**Step 1.** Hold right-click on host h1 and select *Terminal*. This opens the terminal of host h1 and allows the execution of commands on that host.

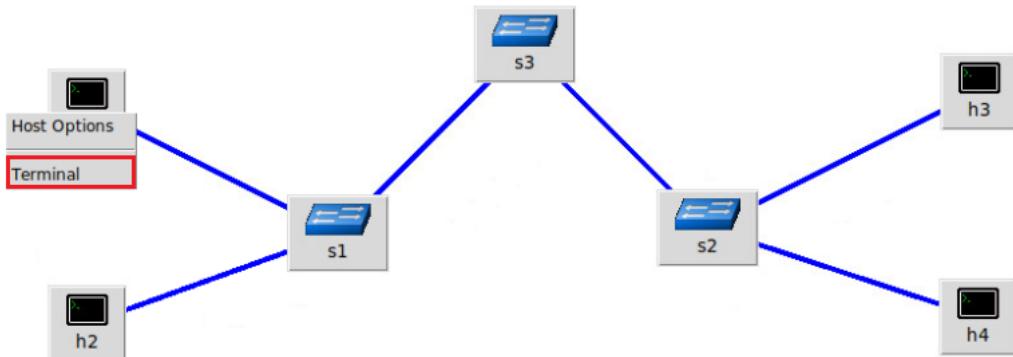
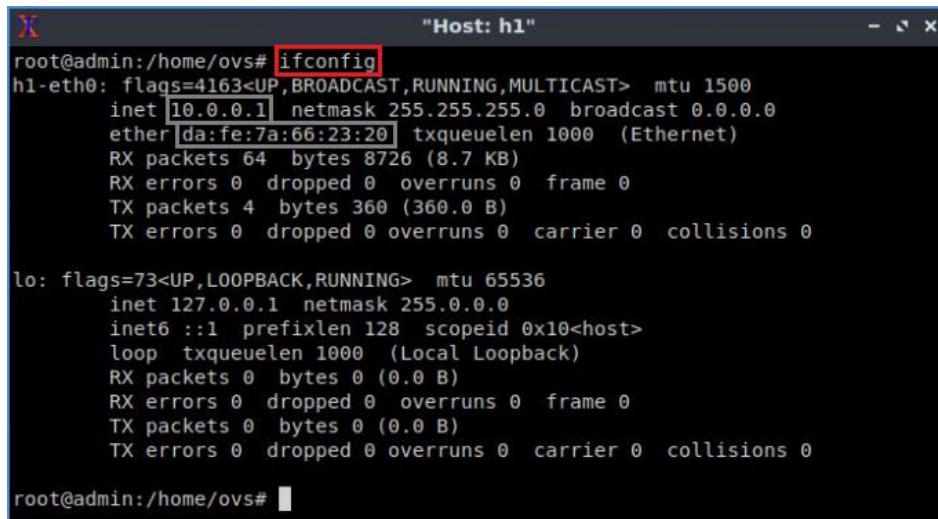


Figure 10. Opening a terminal on host h1.

**Step 2.** In host h1 terminal, type the following command to verify that the IP address was assigned successfully. You will verify the host interface, *h1-eth0* configured with the IP address 10.0.0.1 and the subnet mask 255.255.255.0. You will also verify the MAC address, da:fe:7a:66:23:20.

```
ifconfig
```



```
"Host: h1"
root@admin:/home/ovs# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet [10.0.0.1] netmask 255.255.255.0 broadcast 0.0.0.0
        ether [da:fe:7a:66:23:20] txqueuelen 1000 (Ethernet)
        RX packets 64 bytes 8726 (8.7 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 4 bytes 360 (360.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        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

root@admin:/home/ovs#
```

Figure 11. Verifying IP address and subnet mask on the host.

You may notice a different MAC address since MAC addresses are assigned randomly.

**Step 3.** Hold right-click on host h2 and select *Terminal*. This opens the terminal of host h2 and allows the execution of commands on that host.

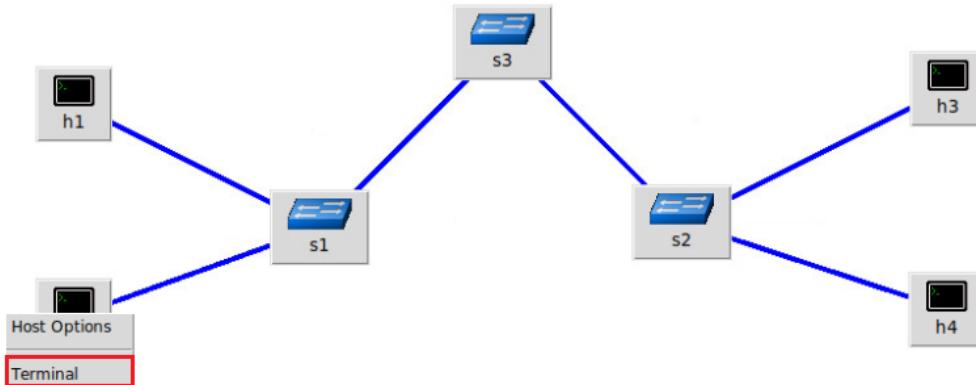
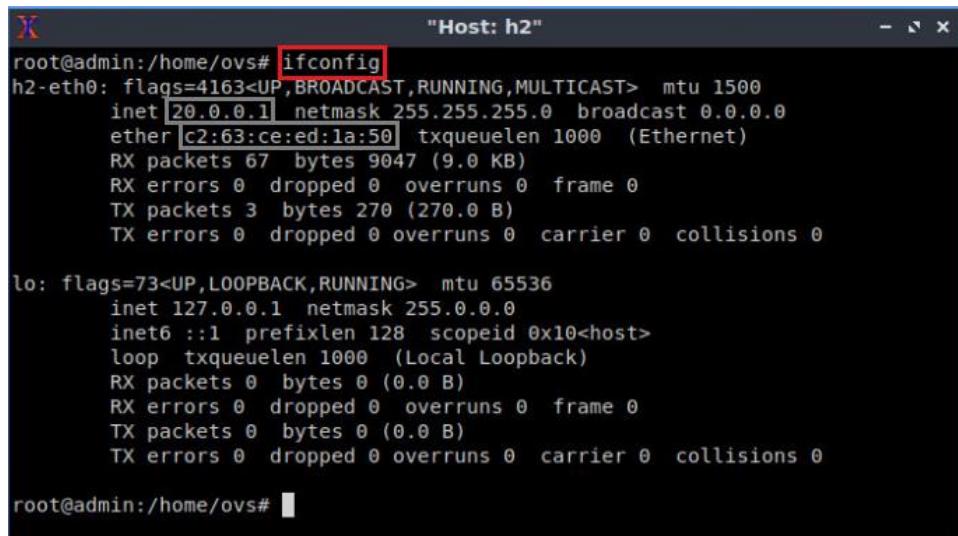


Figure 12. Opening a terminal on host h2.

**Step 4.** In host h2 terminal, type the following command to verify that the IP address was assigned successfully. You will verify the host interface, *h2-eth0* configured with the IP address 20.0.0.1 and the subnet mask 255.255.255.0. You will also verify the MAC address, c2:63:ce:ed:1a:50.

```
ifconfig
```



```
"Host: h2"
root@admin:/home/ovs# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet [20.0.0.1] netmask 255.255.255.0 broadcast 0.0.0.0
                ether [c2:63:ce:ed:1a:50] txqueuelen 1000 (Ethernet)
                RX packets 67 bytes 9047 (9.0 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 3 bytes 270 (270.0 B)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
            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

root@admin:/home/ovs#
```

Figure 13. Verifying IP address and subnet mask on the host.

**Step 5.** Follow step 3 and step 4 to verify IP addresses on hosts h3 and h4.

## 4 Configuring VLANs

In this section, you will configure VLANs according to the topology information (Table 2). You will assign the switch ports attached to hosts h1 and h3 to VLAN 10 and switch ports attached to hosts h2 and h4 to VLAN 20. You will also configure the trunk ports on the switches.

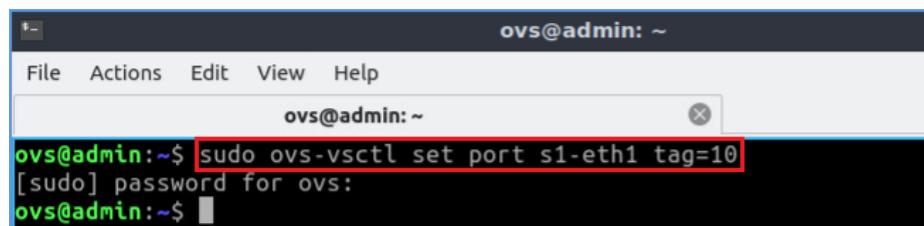
**Step 1.** Open the Linux terminal.



Figure 14. Opening Linux terminal.

**Step 2.** Type the following command to assign port s1-eth1 (connected to h1) to VLAN 10. When prompted for a password, type `password`.

```
sudo ovs-vsctl set port s1-eth1 tag=10
```



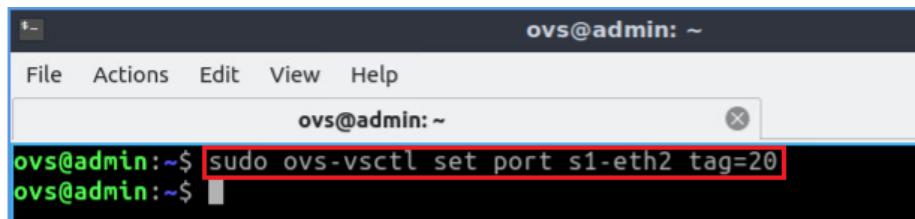
```
ovs@admin: ~
File Actions Edit View Help
ovs@admin: ~
ovs@admin:~$ sudo ovs-vsctl set port s1-eth1 tag=10
[sudo] password for ovs:
ovs@admin:~$
```

Figure 15. Assigning port s1-eth1 to VLAN 10.

Consider the command above. VLAN tag 10 has been assigned to the interface `s1-eth1`.

**Step 3.** Type the following command to assign port *s1-eth2* (connected to h2) to VLAN 20.

```
sudo ovs-vsctl set port s1-eth2 tag=20
```

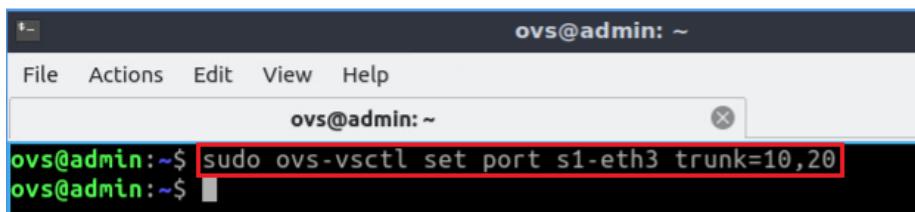


```
ovs@admin:~$ sudo ovs-vsctl set port s1-eth2 tag=20
```

Figure 16. Assigning port *s1-eth2* to VLAN 20.

**Step 4.** Type the following command to configure port *s1-eth3* as a trunk port. The interface is set to trunk mode, where the VLAN traffic can pass through the link. By default, Open vSwitch forwards all VLAN traffic. Using the following command, you are restricting the switch to forward traffic only for particular VLANs.

```
sudo ovs-vsctl set port s1-eth3 trunk=10,20
```



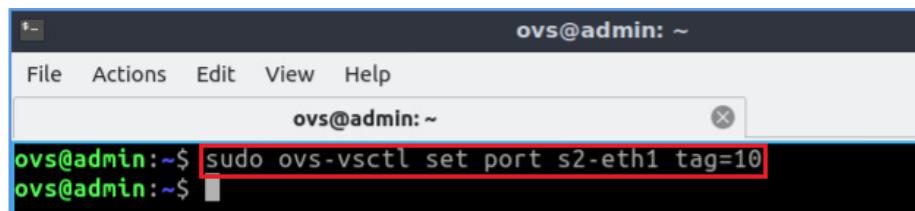
```
ovs@admin:~$ sudo ovs-vsctl set port s1-eth3 trunk=10,20
```

Figure 17. Configuring port *s1-eth3* as a trunk port.

Consider the figure above. Switch s1 will forward traffic belong to VLAN 10 and VLAN 20 only.

**Step 5.** Type the following command to assign port *s2-eth1* (connected to h3) to VLAN 10.

```
sudo ovs-vsctl set port s2-eth1 tag=10
```

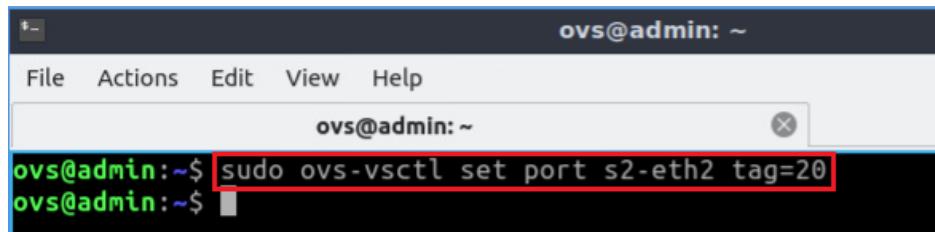


```
ovs@admin:~$ sudo ovs-vsctl set port s2-eth1 tag=10
```

Figure 18. Assigning port *s2-eth1* to VLAN 10.

**Step 6.** Type the following command to assign port *s2-eth2* (connected to h4) to VLAN 20.

```
sudo ovs-vsctl set port s2-eth2 tag=20
```

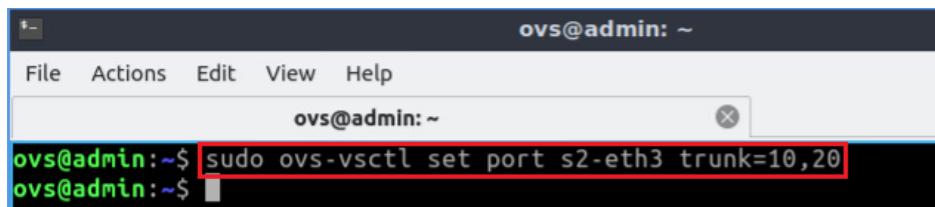


```
ovs@admin:~$ sudo ovs-vsctl set port s2-eth2 tag=20
ovs@admin:~$
```

Figure 19. Assigning port *s2-eth2* to VLAN 20.

**Step 7.** Type the following command to configure port *s2-eth3* as a trunk port. The interface is set to trunk mode, where the VLAN traffic can pass through the link. Switch *s2* will forward traffic belong to VLAN 10 and VLAN 20 only.

```
sudo ovs-vsctl set port s2-eth3 trunk=10,20
```



```
ovs@admin:~$ sudo ovs-vsctl set port s2-eth3 trunk=10,20
ovs@admin:~$
```

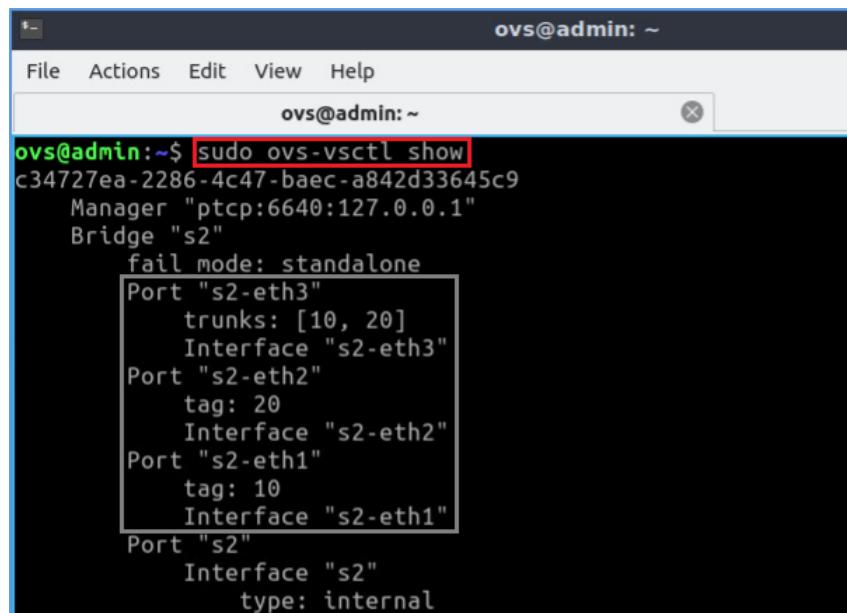
Figure 20. Configuring port *s2-eth3* as a trunk port.

## 5 Verifying configuration

In this section, you will verify the VLAN configuration.

**Step 1.** Type the following command to print a brief overview of the database contents.

```
sudo ovs-vsctl show
```



```
ovs@admin:~$ sudo ovs-vsctl show
c34727ea-2286-4c47-baec-a842d33645c9
    Manager "ptcp:6640:127.0.0.1"
    Bridge "s2"
        fail mode: standalone
        Port "s2-eth3"
            trunks: [10, 20]
            Interface "s2-eth3"
        Port "s2-eth2"
            tag: 20
            Interface "s2-eth2"
        Port "s2-eth1"
            tag: 10
            Interface "s2-eth1"
        Port "s2"
            Interface "s2"
            type: internal
```

Figure 21. Printing an overview of the database contents.

Consider the figure above. Interfaces *s2-eth1* and *s2-eth2* are acting as access ports whereas interface *s2-eth3* is acting as a trunk port.

**Step 2.** In the Linux terminal, start the Wireshark packet analyzer by issuing the following command. A new window will emerge.

```
sudo wireshark
```

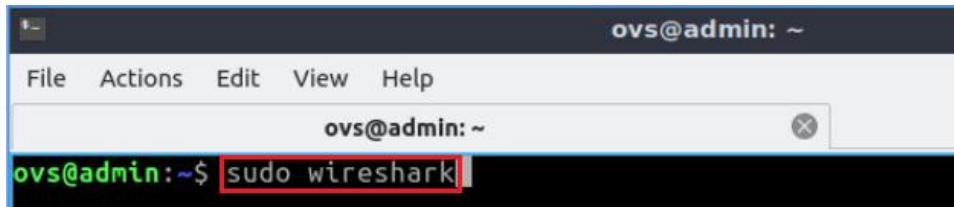


Figure 22. Starting Wireshark packet analyzer.

**Step 3.** Click on interface *s3-eth2* then, click on the icon located on the upper left-hand side to start capturing packets on this interface.

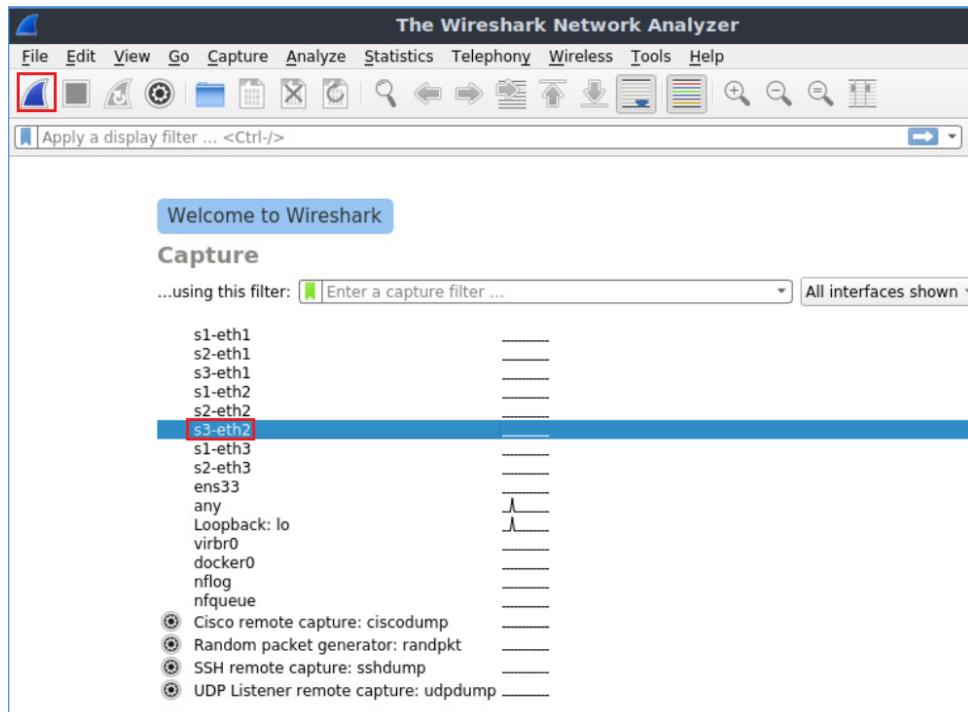
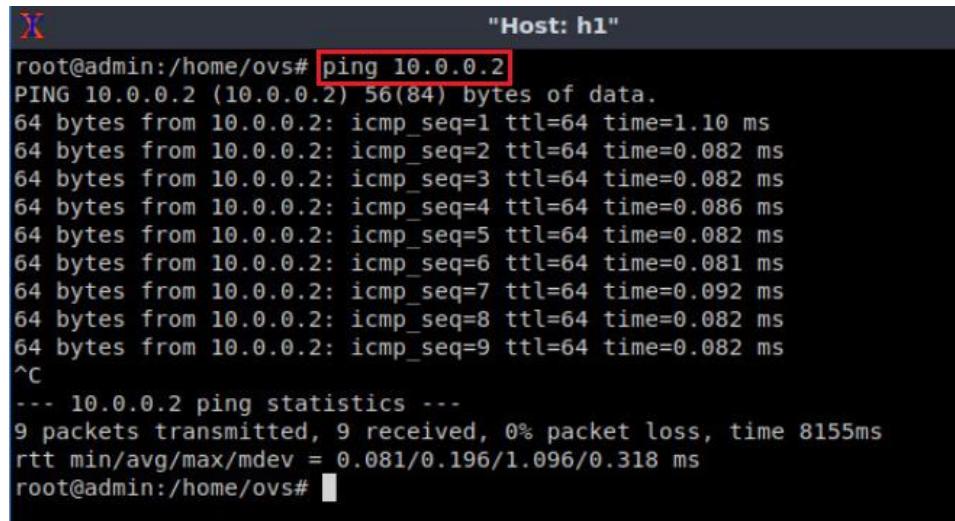


Figure 23. Starting packet capture.

**Step 4.** Test the connectivity between hosts h1 and h3 using the `ping` command. In host h1, type the command specified below. Results will show a successful connectivity test.

```
ping 10.0.0.2
```



```
"Host: h1"
root@admin:/home/ovs# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.10 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.086 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.081 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.092 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.082 ms
^C
--- 10.0.0.2 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8155ms
rtt min/avg/max/mdev = 0.081/0.196/1.096/0.318 ms
root@admin:/home/ovs#
```

Figure 24. Output of the `ping` command.

Press `ctrl+c` to stop the test.

**Step 5.** In Wireshark, the filter box is located on the upper left-hand side, type `vlan` to filter packets containing VLAN tags.

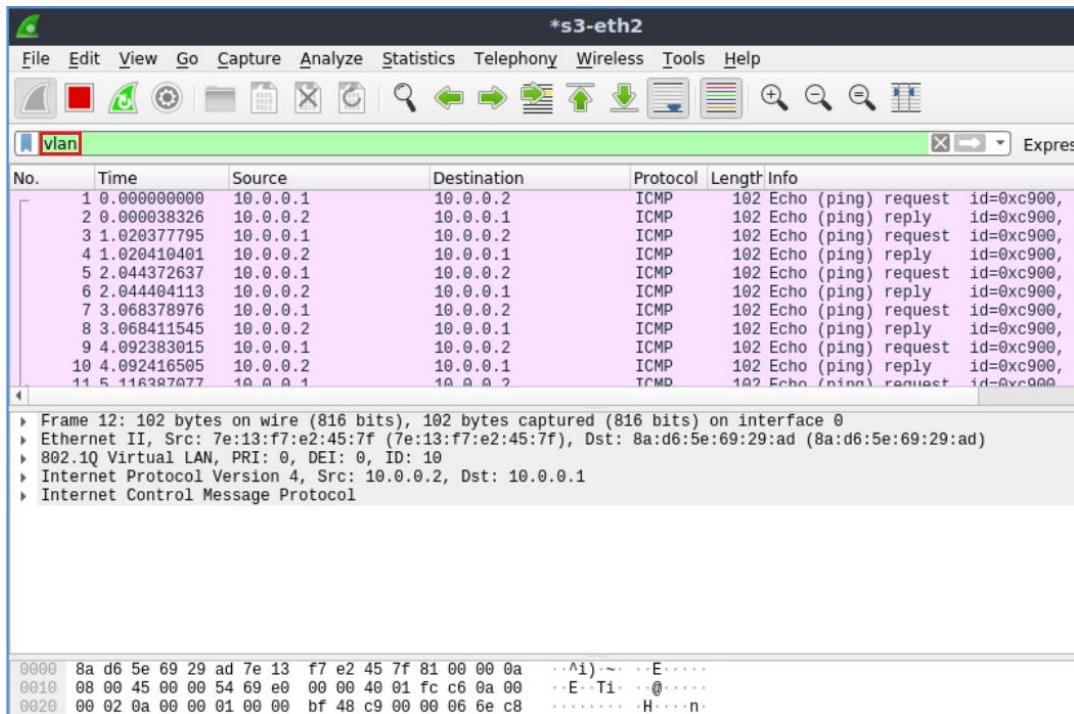


Figure 25. Filtering network traffic.

**Step 6.** Click on the arrow located on the leftmost side of the field called *802.1Q Virtual LAN*. A list will be displayed. Verify that the *VLAN ID* is 10. Notice that such tag corresponds to the packets being forwarded from host h1 to host h3 and vice versa.

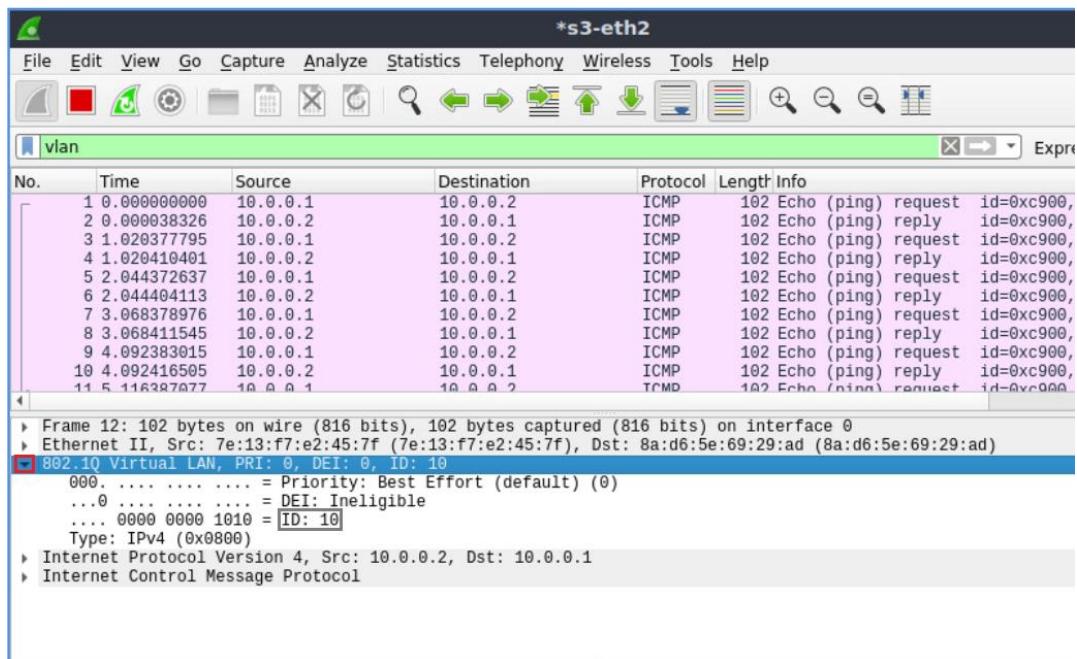


Figure 26. Verifying the VLAN network identifier.

**Step 7.** To stop packet capturing, click on the red button located on the upper left-hand side.

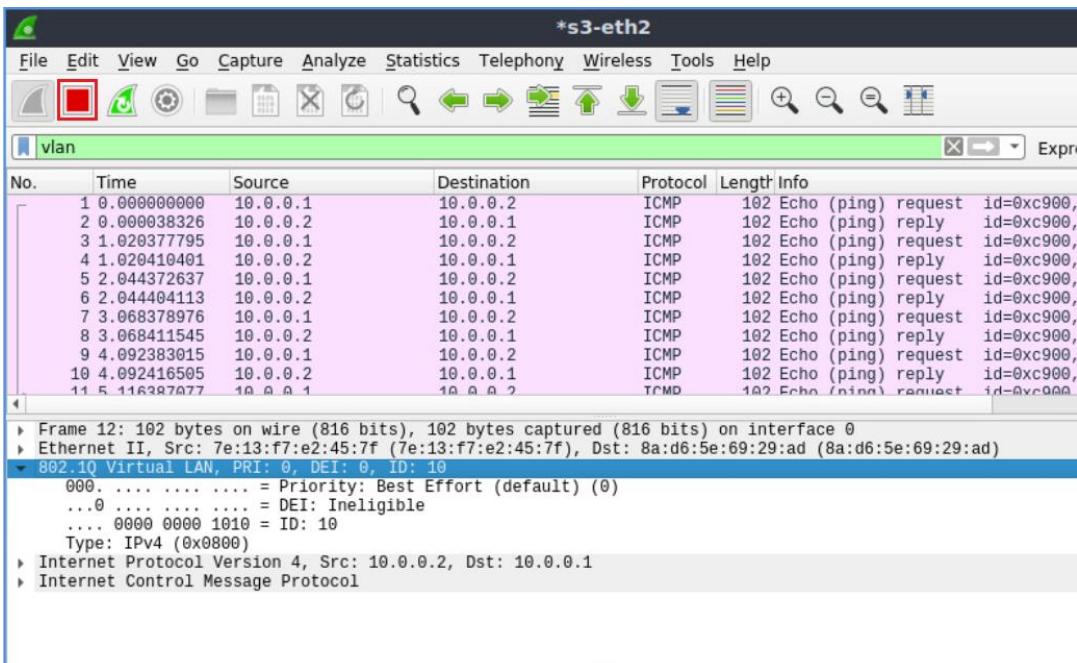


Figure 27. Stopping packet capture.

**Step 8.** In the Wireshark window, start packet capturing by clicking on the button located on the upper left-hand side.

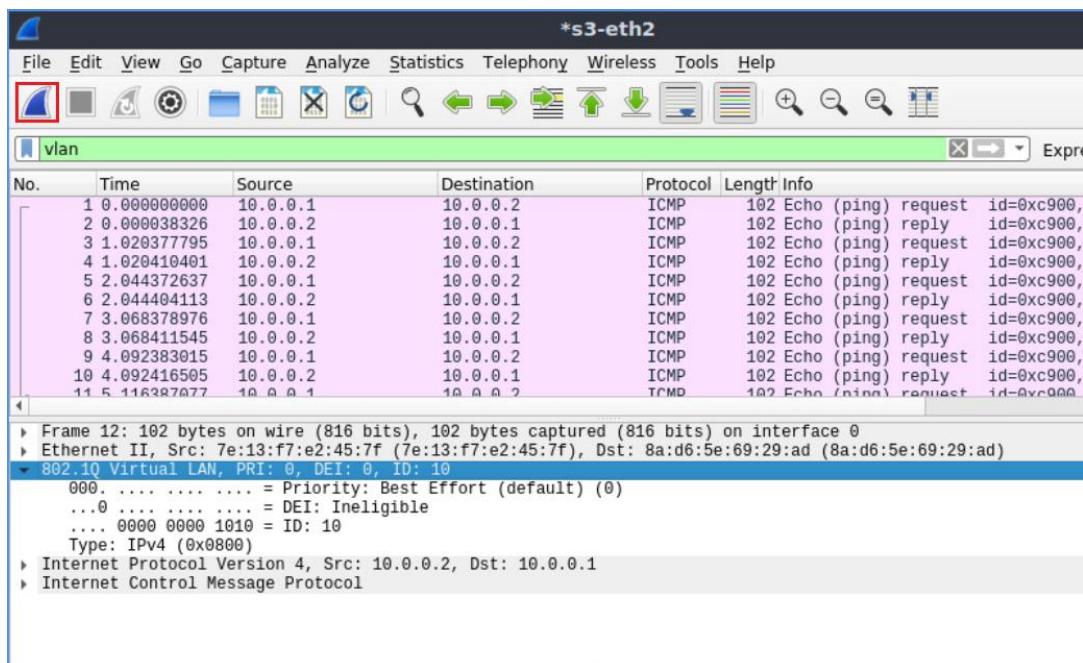


Figure 28. Starting packet capture.

**Step 9.** A notification window will be prompted. Click on *Continue without Saving* to proceed.

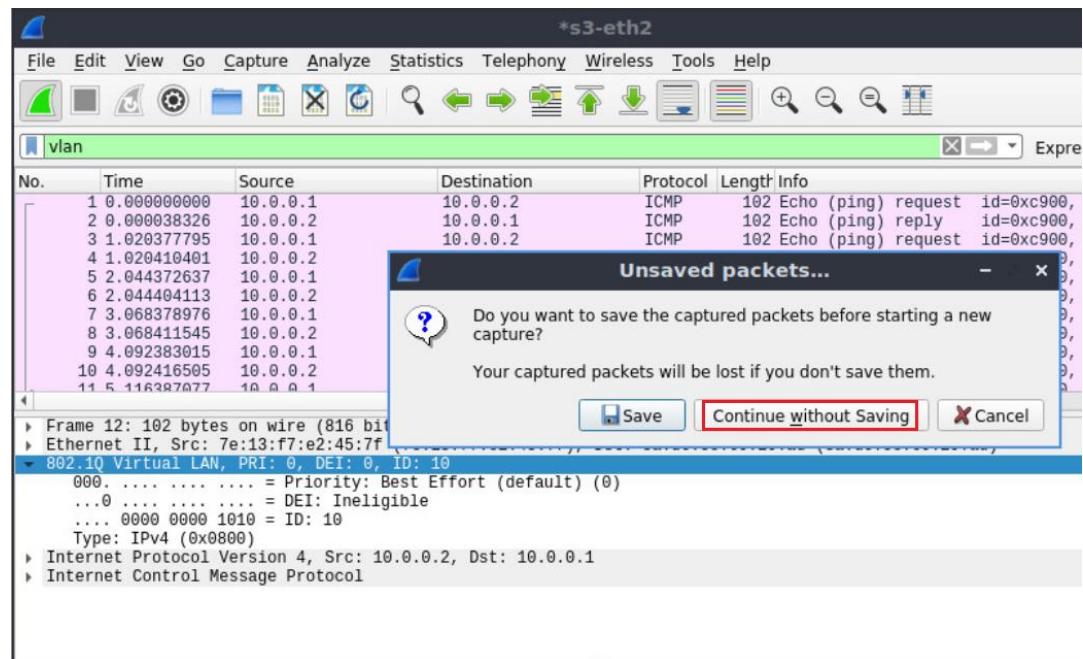


Figure 29. Closing without saving previous packet capture.

**Step 10.** Test the connectivity between hosts h2 and h4 using the `ping` command. In host h2, type the command specified below. Results will show a successful connectivity test.

```
ping 20.0.0.2
```

```

root@admin:/home/ovs# ping 20.0.0.2
PING 20.0.0.2 (20.0.0.2) 56(84) bytes of data.
64 bytes from 20.0.0.2: icmp_seq=1 ttl=64 time=1.13 ms
64 bytes from 20.0.0.2: icmp_seq=2 ttl=64 time=0.080 ms
64 bytes from 20.0.0.2: icmp_seq=3 ttl=64 time=0.079 ms
64 bytes from 20.0.0.2: icmp_seq=4 ttl=64 time=0.082 ms
64 bytes from 20.0.0.2: icmp_seq=5 ttl=64 time=0.060 ms
64 bytes from 20.0.0.2: icmp_seq=6 ttl=64 time=0.083 ms
64 bytes from 20.0.0.2: icmp_seq=7 ttl=64 time=0.082 ms
^C
--- 20.0.0.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6119ms
rtt min/avg/max/mdev = 0.060/0.227/1.129/0.367 ms
root@admin:/home/ovs#

```

Figure 30. Output of the `ping` command.

Press `ctrl+c` to stop the test.

**Step 11.** In the Wireshark window, verify that the VLAN ID is 20. Notice that the tag corresponds to the packets being forwarded from host h2 to h4 and vice versa.

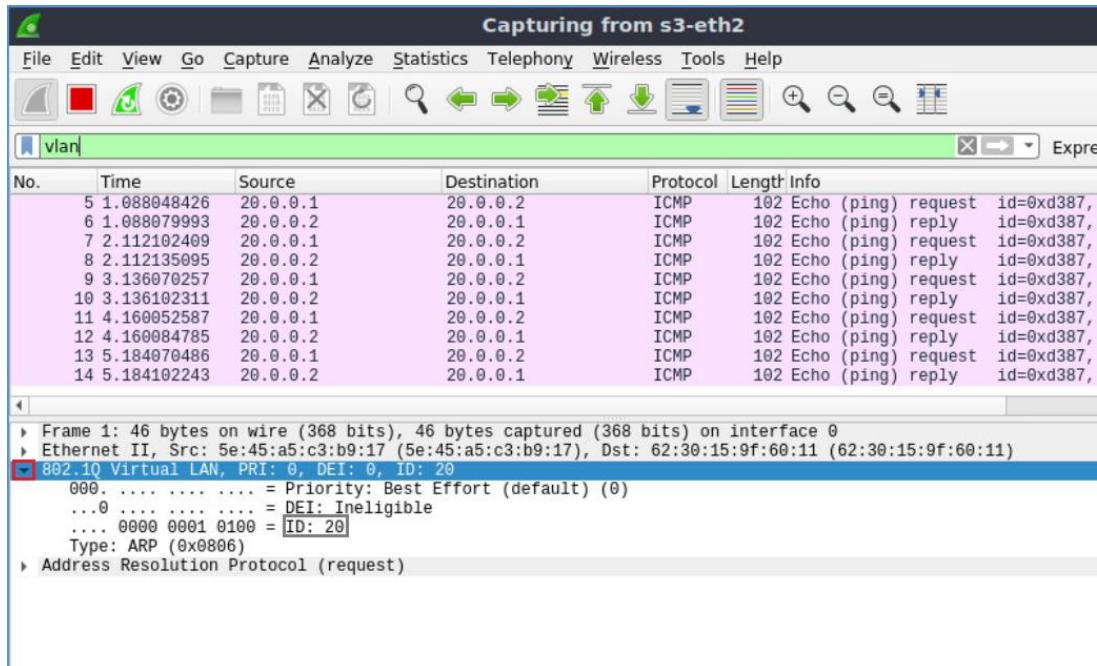


Figure 31. Verifying VLAN network identifier.

**Step 12.** Click on the red button located on the upper left-hand side to stop packet capturing and close Wireshark.

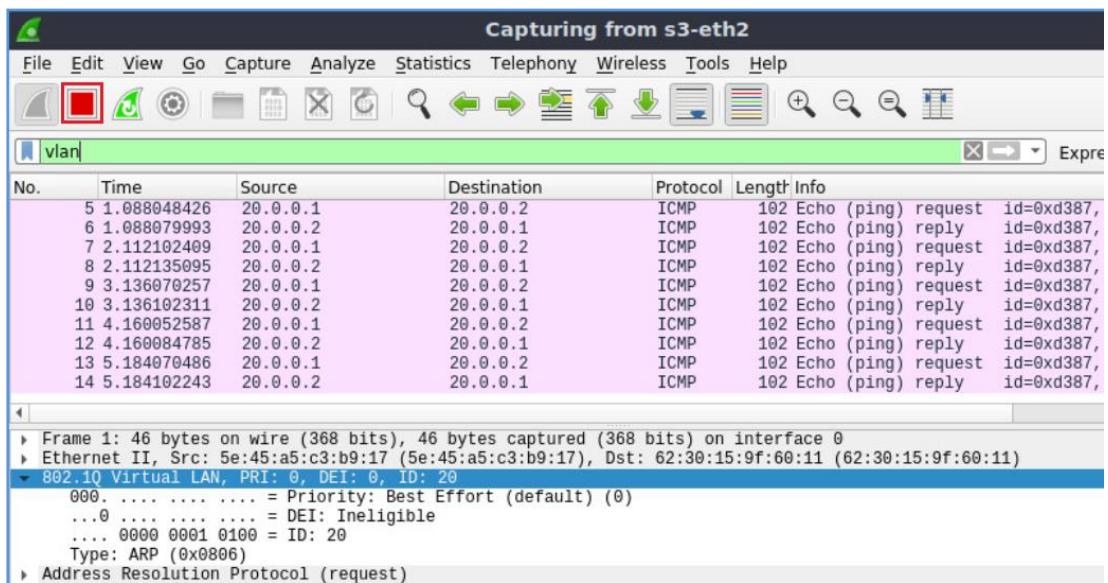


Figure 32. Stopping packet capture.

This concludes Lab 13. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

- IEEE Standard, "IEEE 802.3 working group", [Online]. Available: <https://www.ieee802.org/3/>.
- IEEE Standard, "IEEE standards for local and metropolitan area networks: virtual bridged local area networks," in *IEEE Std 802.1Q-1998*, March 1999.
- James F. Kurose, Keith W. Ross, "Computer networking a top-down approach", 6<sup>th</sup> Edition, Mar 2017.
- Dhurgham Abdulridha Jawad AL-Khaffaf, "Improving LAN performance based on IEEE802.1Q VLAN switching techniques", Journal of university of babylon, engineering sciences, Vol. (26), No. (1): 2018.
- Cisco, "Catalyst 4500 series switch Cisco IOS software configuration guide", Feb 2018.
- Juniper Networks, "Layer 2 networking", Jun 2020.
- Linux Foundation, "Open vSwitch", [Online]. Available: <http://openvSwitch.org>.
- Juniper networks, "Layer 2 VLANs overview", Dec 2017.
- IBM, "Archived / virtual networking in Linux", [Online]. Available: <https://developer.ibm.com/tutorials/l-virtual-networking/>
- Mininet walkthrough, [Online]. Available: <http://mininet.org>.
- Cisco, "Cisco Nexus 5000 series NX-OS configuration guide", July 2017.



## OPEN VIRTUAL SWITCH

### Exercise 5: Configuring VLAN

Document Version: **09-19-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

1	Exercise topology .....	3
1.1	Topology settings .....	3
1.2	Credentials .....	3
2	Deliverables.....	4

## 1 Exercise topology

This goal of this exercise is to configure multiple VLANs to isolate network traffic within an emulated environment.

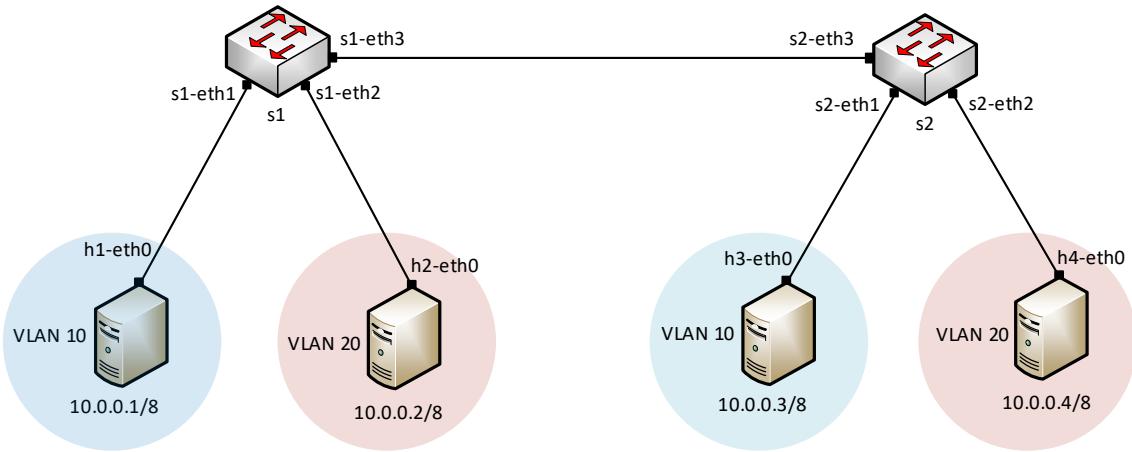


Figure 1. Exercise topology.

### 1.1 Topology settings

The devices are already configured according to Table 1.

Table 1. Topology information.

Host	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8
h3	h3-eth0	10.0.0.3	/8
h4	h4-eth0	10.0.0.4	/8

### 1.2 Credentials

The information in Table 2 provides the credentials to access the Client's virtual machine.

Table 2. Credentials to access the Client's virtual machine.

Device	Account	Password
Client	admin	password

## 2 Deliverables

Follow the steps below to complete the exercise.

- a) Start MiniEdit by clicking on MiniEdit's shortcut. Load the topology *Exercise5.mn* located at *~/OVS\_Labs/Exercise5* as shown in the figure below.

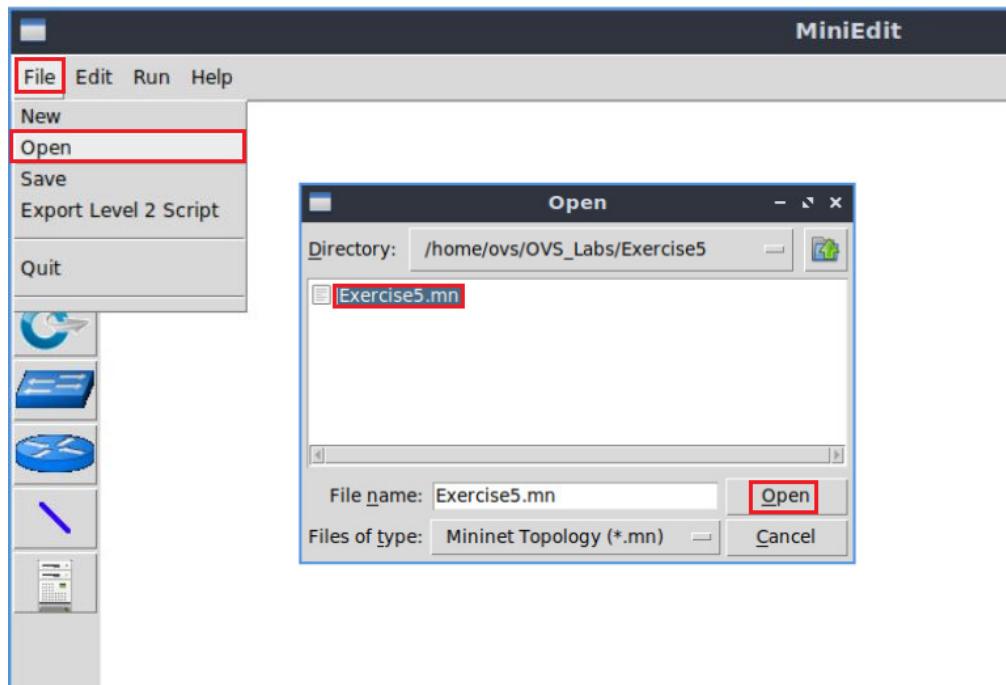


Figure 2. Loading the topology file in Mininet.

- b) Run the emulation in Mininet.
- c) In the Mininet terminal, launch the command that displays the interface names and connections of the current topology. Verify that links conform to the topology in Figure 1.
- d) *~/OVS\_Labs/Exercise5* folder contains a script *config\_switches.sh* responsible for loading the configuration needed for the exercise. Execute the script using the following commands:

```
cd OVS_Labs/Exercise5
```

```
./config_switches.sh
```

- e) Assign switch ports attached to hosts h1 and h3 to VLAN 10 and switch ports attached to hosts h2 and h4 to VLAN 20.
- f) Configure switch ports *s1-eth3* and *s2-eth3* so that traffic for VLAN 10 is allowed (no trunk).
- g) Verify the connectivity using the `ping` command. Explain the result. Why there is no connectivity between hosts h2 and h4?

**h)** Configure switch ports *s1-eth3* and *s2-eth3* so that traffic for VLAN 20 is allowed (no trunk) and repeat **g**).

**i)** Configure switch ports *s1-eth3* and *s2-eth3* so that traffic for both VLANs 10 and 20 are allowed. Verify the connectivity between the hosts.

Configuration for switch ports *s1-eth3* and *s2-eth3* need to be removed before you start **i**).



## OPEN VIRTUAL SWITCH

### Lab 14: Configuring GRE Tunnel

Document Version: **09-16-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    GRE tunnel.....	4
1.2    GRE packet header .....	5
2    Lab topology.....	5
2.1    Lab settings.....	6
2.2    Loading a topology .....	7
2.3    Load the configuration file .....	8
2.4    Run the emulation.....	9
2.5    Verify the configuration .....	10
3    Configuring OSPF on routers.....	13
4    Run Mininet instances within the containers.....	16
5    Configuring GRE tunnel.....	18
6    Verifying GRE configuration.....	21
References .....	24

## Overview

This lab presents Generic Route Encapsulation (GRE), a tunneling protocol that encapsulates a wide variety of network layer protocols inside virtual point-to-point links or point-to-multipoint links over an IP network. This lab aims to configure a GRE tunnel across a Wide Area Network (WAN).

## Objectives

By the end of this lab, you should be able to:

1. Understand the concept of GRE tunnel.
2. Emulate servers by using docker containers.
3. Configure and verify GRE tunnel between two servers.
4. Use Wireshark network analyzer to inspect GRE traffic.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Configuring OSPF on routers.
4. Section 4: Run Mininet instances within the containers.
5. Section 5: Configuring GRE Tunnel.
6. Section 6: Verifying GRE configuration.

### 1      **Introduction**

Most organizations want to use secure and cost-effective ways to interconnect multiple networks, such as allowing branch offices to connect to the headquarters' network. They use Virtual Private Networks (VPNs) to create an end-to-end private network connection over third-party networks, such as the Internet.

Tunneling is a protocol that allows for the secure movement of data from one network to another. It will enable private network communications to be sent across a public network, such as the Internet, through encapsulation. Encapsulation is the process of enclosing one type of packet using another type of packet<sup>2</sup>.

## 1.1 GRE tunnel

GRE is a protocol used for the encapsulation of a network layer protocol within another network layer protocol. Encapsulation is often referred to as tunneling, which means wrapping one data packet within another data packet. GRE encapsulates data packets and redirects them to a device that de-encapsulates them and routes them to their final destination. This allows the source and destination hosts to operate as if they are connected via a virtual point-to-point connection (i.e., a tunnel)<sup>4</sup>. GRE aims to simplify connections between separate networks. GRE tunnels are usually configured between two routers. Each router acts like one end of the tunnel. The routers are set up to send and receive GRE packets directly to each other. Any routers in between those two routers will not open the encapsulated packets; they only reference the headers surrounding the encapsulated packets to forward them<sup>5</sup>.

When a router receives a data packet to be tunneled, the data packet is sent to the tunnel interface. The tunnel interface encapsulates the data in a GRE packet and adds an additional header (outer IP header). The IP packet is forwarded based on the outer IP header. Intermediate routers only forward the encapsulated packet to the destination router. When the destination router receives the IP packet from the tunnel interface, the outer IP header and GRE header are removed<sup>3</sup>. GRE is considered a VPN (Virtual Private Network) since a private network is created by tunneling over a public network.

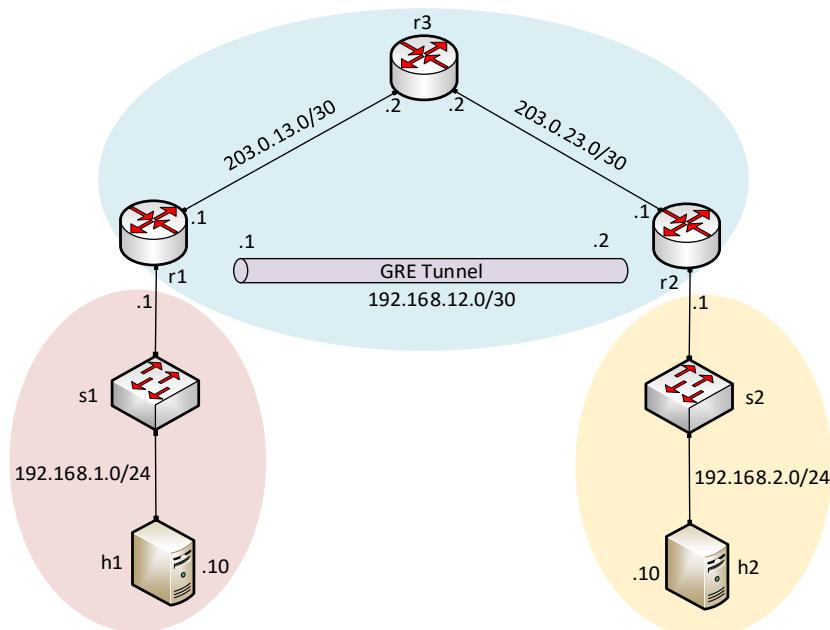


Figure 1. GRE tunnel.

Consider Figure 1. Three routers can communicate with each other. Router r3 does not contain any route to the networks 192.168.1.0/24 and 192.168.2.0/24. GRE tunnel is configured between routers r1 and r2 (network 192.168.12.0/30). Whenever host h1 (192.168.1.10) wants to communicate with host h2 (192.168.2.10), router r1 will change the IP header according to GRE tunnel configuration. Router r1 will change the source IP to 192.168.12.1, and the destination IP will be 192.168.12.2. Router r3 will pass the traffic to the destination router r2. Router r2 will decapsulate the packet, discovers the actual destination IP (192.168.2.10), and delivers the packet to host, h2.

## 1.2 GRE packet header

All data is broken up into smaller pieces called packets while sent over a network, and all packets consist of two parts: the payload and the header. The payload is the actual data, and the header includes the information about where the packet comes from and what group of packets it belongs to. Each network protocol attaches a header to every packet.

GRE adds two headers to each packet: the GRE header (4-bytes long) and outer IP header (20-bytes long). The GRE header indicates the protocol type used by the encapsulated packet. The outer IP header encapsulates the original packet's header and payload. That means a GRE packet uses two IP headers: one for the original packet and one added by the GRE protocol. Only the routers at each end of the GRE tunnel will reference the original IP header<sup>5</sup>.

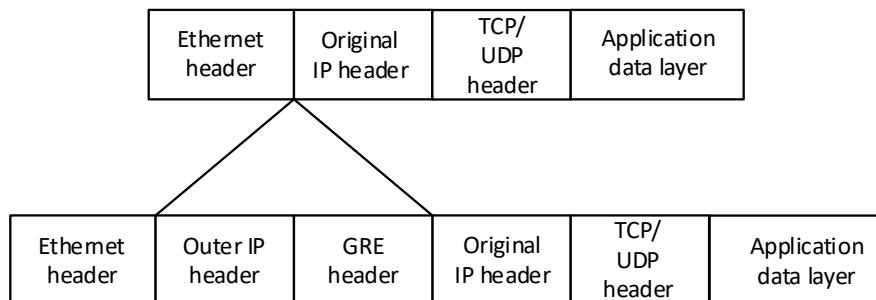


Figure 2. GRE packet header.

## 2 Lab topology

Consider Figure 3. The topology consists of four hosts, two switches, and three routers. The end hosts and switches are running inside Server 1 and Server 2. Those servers are implemented by Docker<sup>6</sup> containers which run Mininet instances. Docker is a platform that uses OS-level virtualization to deliver software packages called container. Routers are supported by the Free-range Routing (FRR) engine. Servers are connected to ISP routers. GRE tunnel is configured between servers (s1 and s2) so that the hosts within the servers assume they are directly connected through the GRE tunnel.

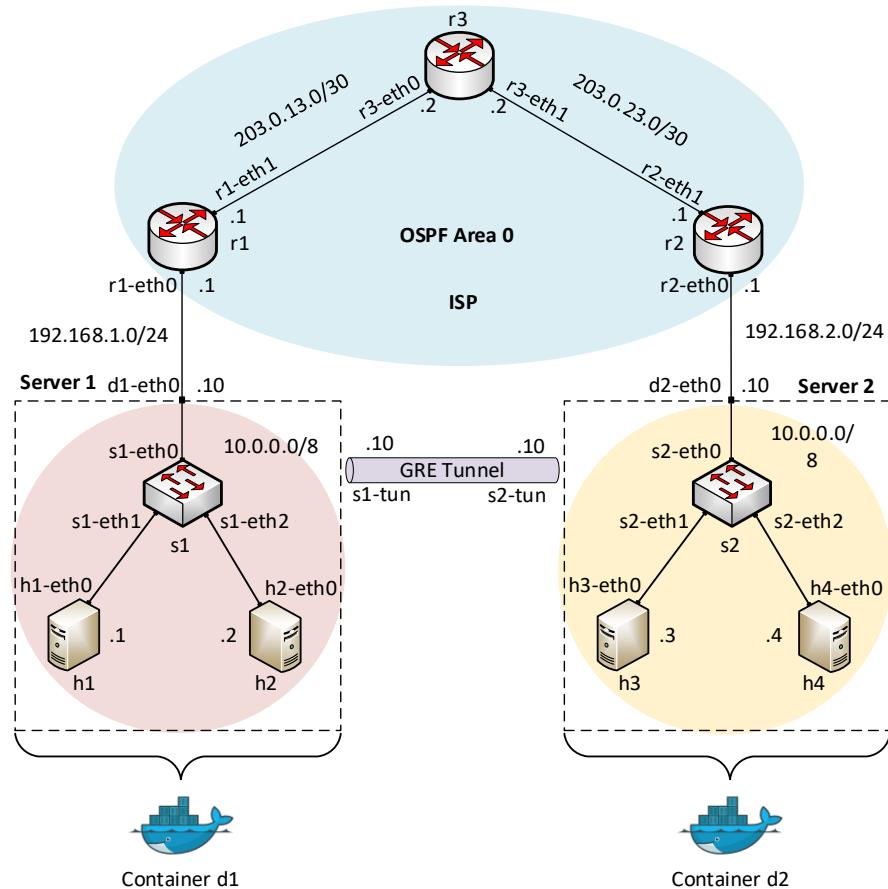


Figure 3. Lab topology.

## 2.1 Lab settings

The devices are configured according to Table 2.

Table 2. Topology information.

Device	Interface	IP Address	Subnet
r1	r1-eth0	192.168.1.1	/24
	r1-eth1	203.0.13.1	/30
r2	r2-eth0	192.168.2.1	/24
	r2-eth1	203.0.23.1	/30
r3	r3-eth0	203.0.13.2	/30
	r3-eth1	203.0.23.2	/30
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8

h3	h3-eth0	10.0.0.3	/8
h4	h4-eth0	10.0.0.4	/8

## 2.2 Loading a topology

In this section, you will open MiniEdit and load the lab topology. MiniEdit provides a Graphical User Interface (GUI) that facilitates the creation and emulation of network topologies in Mininet. This tool has additional capabilities: configuring network elements (i.e., IP addresses, default gateway), saving topologies, and exporting layer 2 models.

**Step 1.** A shortcut to MiniEdit is located on the desktop. Start MiniEdit by clicking on MiniEdit shortcut. When prompted for a password, type `password`.

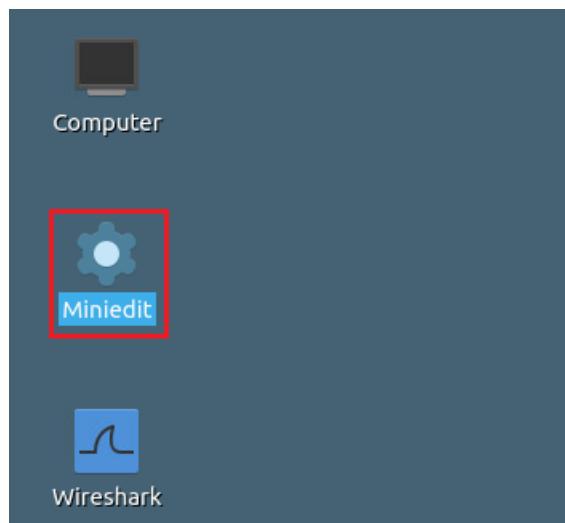


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File*, then *open* to load the lab's topology. Locate the *lab14.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab14* and click on *Open*.

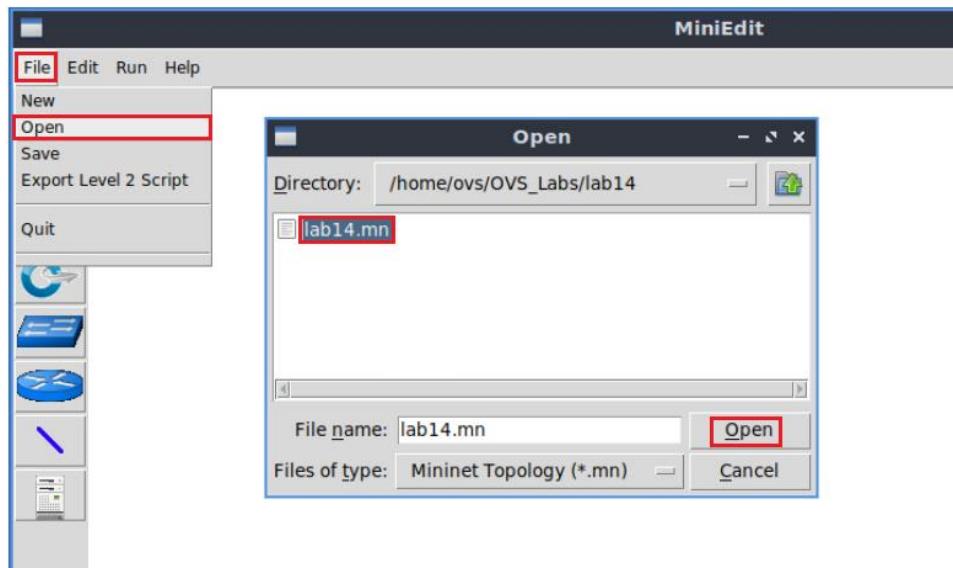


Figure 5. MiniEdit's Open dialog.

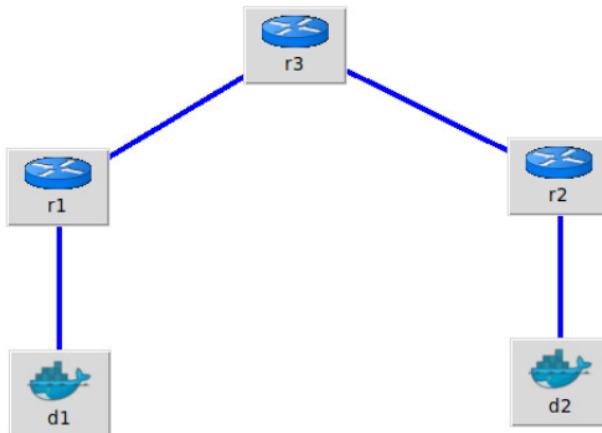


Figure 6. MiniEdit's topology.

### 2.3 Load the configuration file

**Step 1.** Open the Linux terminal.

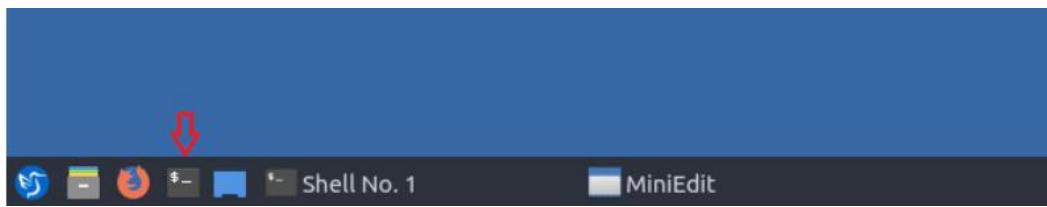
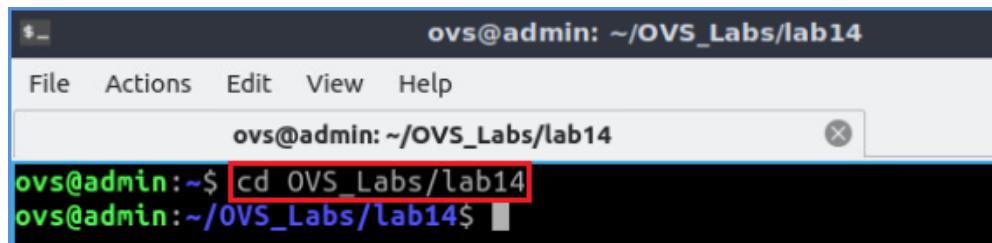


Figure 7. Opening Linux terminal.

**Step 2.** Click on the Linux terminal and navigate into *OVS\_Labs/lab14* directory by issuing the following command. This folder contains a configuration file, and the script is responsible for loading the configuration. The configuration file will assign the IP addresses to the routers' interfaces. The `cd` command is short for change directory, followed by an argument that specifies the destination directory.

```
cd OVS_Labs/lab14
```

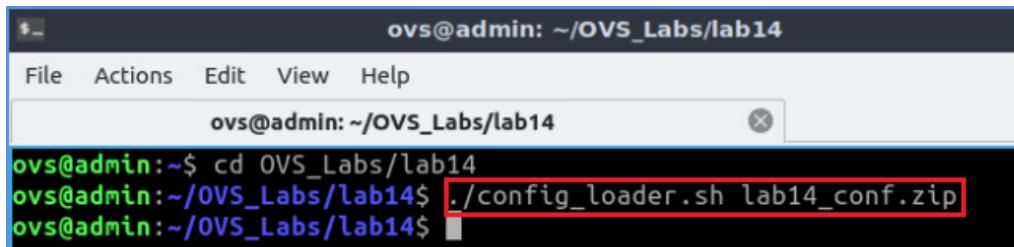


A screenshot of a terminal window titled "ovs@admin: ~/OVS\_Labs/lab14". The window has a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu is a title bar with the same text. The main area shows a command-line interface with the prompt "ovs@admin: ~\$". A red box highlights the command "cd OVS\_Labs/lab14" as it is being typed. The command has been fully typed and is now shown in blue text.

Figure 8. Entering to the *OVS\_Labs/lab14* directory.

**Step 3.** To execute the shell script, type the following command. The program's argument corresponds to the configuration zip file that will be loaded in all the routers in the topology.

```
./config_loader.sh lab14_conf.zip
```

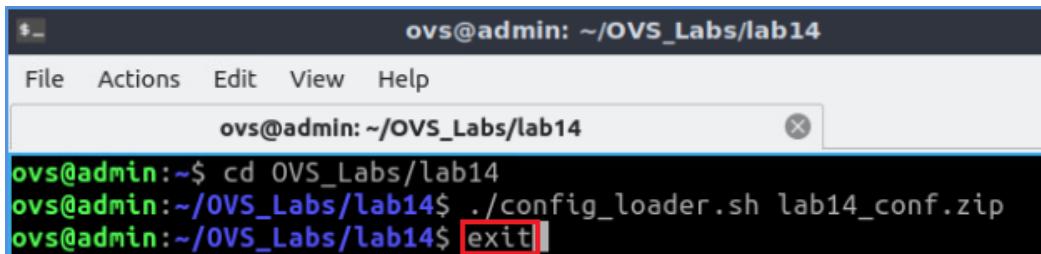


A screenshot of a terminal window titled "ovs@admin: ~/OVS\_Labs/lab14". The window has a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu is a title bar with the same text. The main area shows a command-line interface with the prompt "ovs@admin: ~\$". A red box highlights the command "./config\_loader.sh lab14\_conf.zip" as it is being typed. The command has been fully typed and is now shown in blue text.

Figure 9. Executing the shell script to load the configuration.

**Step 4.** Type the following command to exit the Linux terminal.

```
exit
```



A screenshot of a terminal window titled "ovs@admin: ~/OVS\_Labs/lab14". The window has a menu bar with "File", "Actions", "Edit", "View", and "Help". Below the menu is a title bar with the same text. The main area shows a command-line interface with the prompt "ovs@admin: ~\$". A red box highlights the command "exit" as it is being typed. The command has been fully typed and is now shown in blue text.

Figure 10. Exiting from the terminal.

## 2.4 Run the emulation

**Step 1.** Click on the *Run* button to start the emulation. The emulation will start, and the MiniEdit panel buttons will gray out, indicating that they are currently disabled.



Figure 11. Starting the emulation.

**Step 2.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

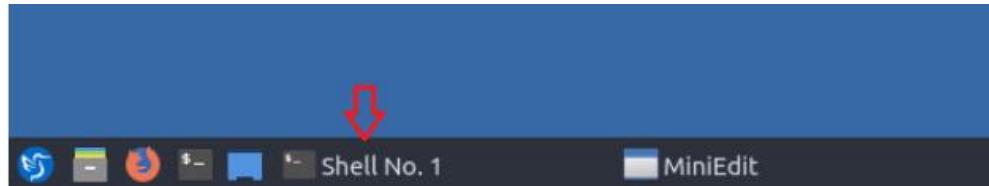


Figure 12. Opening Mininet's terminal.

**Step 3.** Issue the following command to display the interface names and connections.

```
links
```

A screenshot of a terminal window titled 'Shell No. 1'. The window has a menu bar with File, Actions, Edit, View, Help. The title bar also says 'Shell No. 1'. The main area shows the command 'links' being run, followed by its output: 'r1-eth0<->d1-eth0 (OK OK)', 'r2-eth0<->d2-eth0 (OK OK)', 'r1-eth1<->r3-eth0 (OK OK)', 'r3-eth1<->r2-eth1 (OK OK)'. The entire output line is highlighted with a gray box, and the word 'links' in the command line is also highlighted with a red box.

Figure 13. Displaying network interfaces.

In Figure 13, the link displayed within the gray box indicates that interface *eth0* of router *r1* is connected to interface *eth0* of docker container *d1* (i.e., *r1-eth0<->d1-eth0*).

## 2.5 Verify the configuration

**Step 1.** To open router r1 terminal, hold right-click on router r1 and select Terminal.

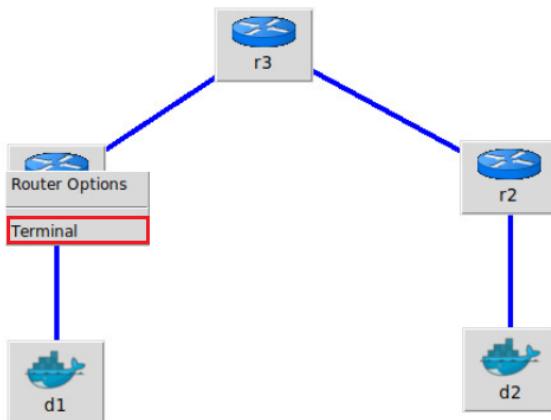


Figure 14. Opening a terminal on router r1.

**Step 2.** In router r1's terminal, you will start zebra daemon, which is a multi-server routing software that provides TCP/IP-based routing protocols. The configuration will not be working if you do not enable zebra daemon initially. To start the zebra, type the following command:

```
zebra
```

```
X "Host: r1"
root@frr-pc:/etc/routers/r1# zebra
root@frr-pc:/etc/routers/r1#
```

Figure 15. Starting zebra daemon.

**Step 3.** After initializing zebra, vtysh should be started to provide all the command-line interface (CLI) commands defined by the daemons. To proceed, issue the following command:

```
vtysh
```

```
X "Host: r1"
root@frr-pc:/etc/routers/r1# zebra
root@frr-pc:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

frr-pc#
```

Figure 16. Starting vtysh on router r1.

**Step 4.** Type the following command in router r1's terminal to verify the routing table of router r1. It will list all the directly connected networks. The routing table of router r1 does not contain any route to the network attached to router r2 (203.0.23.0/30, 192.168.2.0/24). There is no routing protocol configured yet.

```
show ip route
```

```

"Host: r1"
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       0 - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected route
e

C>* 192.168.1.0/24 is directly connected, r1-eth0, 00:00:35
C>* 203.0.13.0/30 is directly connected, r1-eth1, 00:00:35
admin# 

```

Figure 17. Displaying routing table of router r1.

**Step 5.** Router r2 is configured similarly to router r1 but with different IP addresses (see Table 2). Those steps are summarized in the following figure. To proceed, in router r2's terminal, issue the commands depicted below. At the end, you will verify all the directly connected networks of router r2.

```

"Host: r2"
root@admin:/etc/routers/r2# zebra
root@admin:/etc/routers/r2# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       0 - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected route
e

C>* 192.168.2.0/24 is directly connected, r2-eth0, 00:00:07
C>* 203.0.23.0/30 is directly connected, r2-eth1, 00:00:07
admin# 

```

Figure 18. Displaying routing table of router r2.

**Step 6.** Router r3 is configured similarly to router r1 but, with different IP addresses (see Table 2). Those steps are summarized in the following figure. To proceed, in router r3's terminal, issue the commands depicted below. At the end, you will verify all the directly connected networks of router r3.

```

"Host: r3"
root@admin:/etc/routers/r3# zebra
root@admin:/etc/routers/r3# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       0 - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected route
e

C>* 203.0.13.0/30 is directly connected, r3-eth0, 00:00:05
C>* 203.0.23.0/30 is directly connected, r3-eth1, 00:00:05
admin# 

```

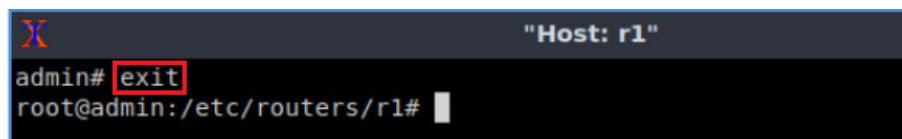
Figure 19. Displaying routing table of router r3.

### 3 Configuring OSPF on routers

At this point, routers are configured with their corresponding IP addresses (see Table 2). However, to provide end-to-end connectivity, it is necessary to enable and configure a routing protocol. In this section, you will configure OSPF as the routing protocol in routers r1, r2, and r3.

**Step 1.** Type the command shown below to close *vtysh*.

```
exit
```



A terminal window titled "Host: r1". The command "admin# exit" is being typed, with the "exit" part highlighted by a red box. The prompt "root@admin:/etc/routers/r1#" is visible at the bottom.

Figure 20. Exiting *vtysh*.

**Step 2.** To enable the OSPF daemon, issue the following command:

```
ospfd
```

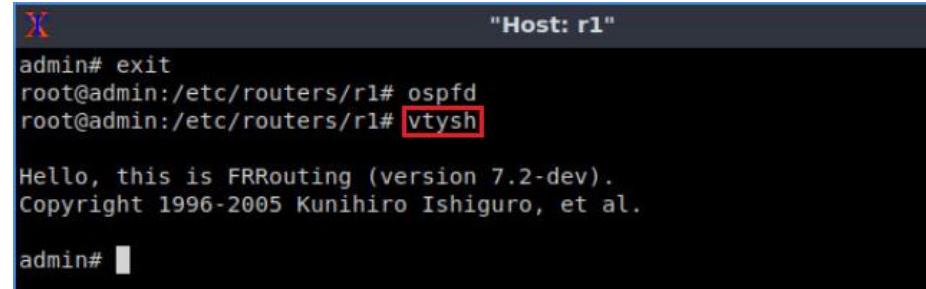


A terminal window titled "Host: r1". The command "root@admin:/etc/routers/r1# ospfd" is being typed, with "ospfd" highlighted by a red box. The prompt "root@admin:/etc/routers/r1#" is visible at the bottom.

Figure 21. Starting the OSPF daemon.

**Step 3.** To enable *vtysh* again, issue the following command:

```
vtysh
```

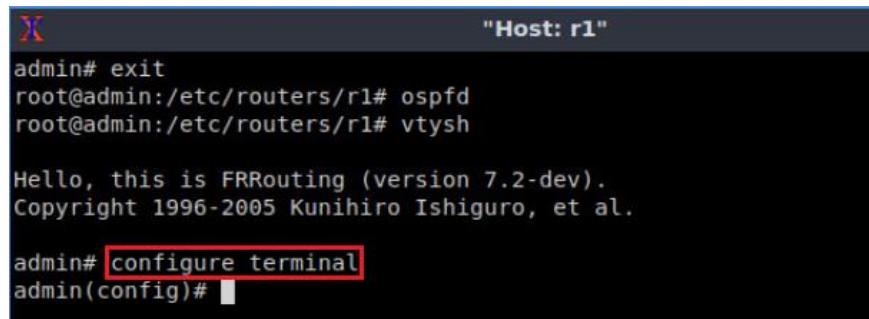


A terminal window titled "Host: r1". The command "root@admin:/etc/routers/r1# vtysh" is being typed, with "vtysh" highlighted by a red box. Below the command, the FRRouting version information is displayed: "Hello, this is FRRouting (version 7.2-dev). Copyright 1996-2005 Kunihiro Ishiguro, et al." The prompt "admin#" is visible at the bottom.

Figure 22. Starting *vtysh*.

**Step 4.** To enable configuration mode in router r1, type the following command:

```
configure terminal
```



```
"Host: r1"
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

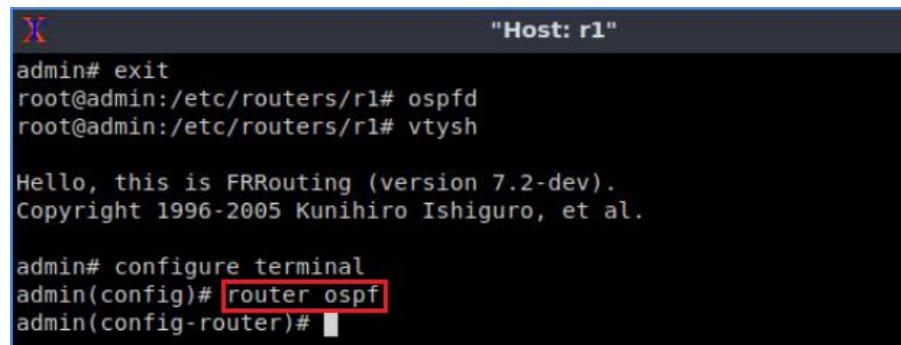
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# 
```

Figure 23. Starting router r1 in configuration mode.

**Step 5.** Issue the following command to configure OSPF in router r1.

```
router ospf
```



```
"Host: r1"
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

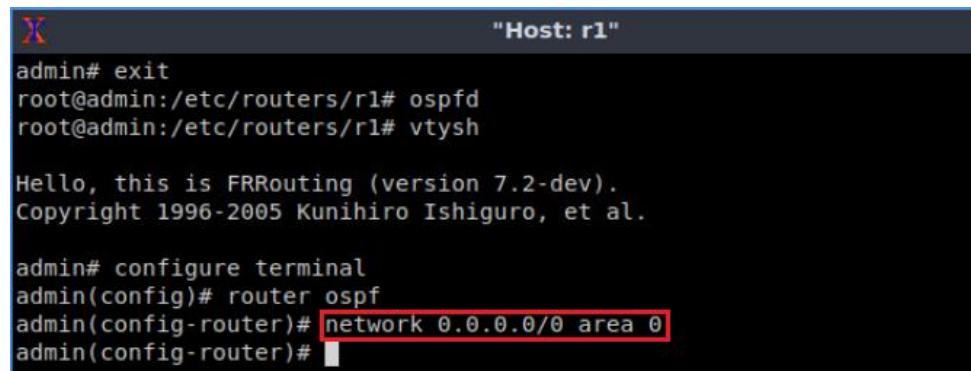
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# router ospf
admin(config-router)# 
```

Figure 24. Configuring OSPF in router r1.

**Step 6.** Type the following command to assign a network and an area to router r1.

```
network 0.0.0.0/0 area 0
```



```
"Host: r1"
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# router ospf
admin(config-router)# network 0.0.0.0/0 area 0
admin(config-router)# 
```

Figure 25. Configuring OSPF network and area settings.

Consider the command above. OSPF will advertise all the networks connected to router r1.

**Step 7.** Issue the following command to end the configuration in router r1.

```
end
```

```
"Host: r1"
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# router ospf
admin(config-router)# network 0.0.0.0/0 area 0
admin(config-router)# end
admin# "
```

Figure 26. Finishing router r1 configuration.

**Step 8.** At this point, router r1 has configured OSPF. Proceed similarly in router r2 by following from step 1 to step 7. All those steps are summarized in the figure below.

```
"Host: r2"
admin# exit
root@admin:/etc/routers/r2# ospfd
root@admin:/etc/routers/r2# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# router ospf
admin(config-router)# network 0.0.0.0/0 area 0
admin(config-router)# end
admin# "
```

Figure 27. Summary of OSPF configuration in router r2.

**Step 9.** Proceed similarly in router r3 by following from step 1 to step 7. All those steps are summarized in the figure below.

```
"Host: r3"
admin# exit
root@admin:/etc/routers/r3# ospfd
root@admin:/etc/routers/r3# vtysh

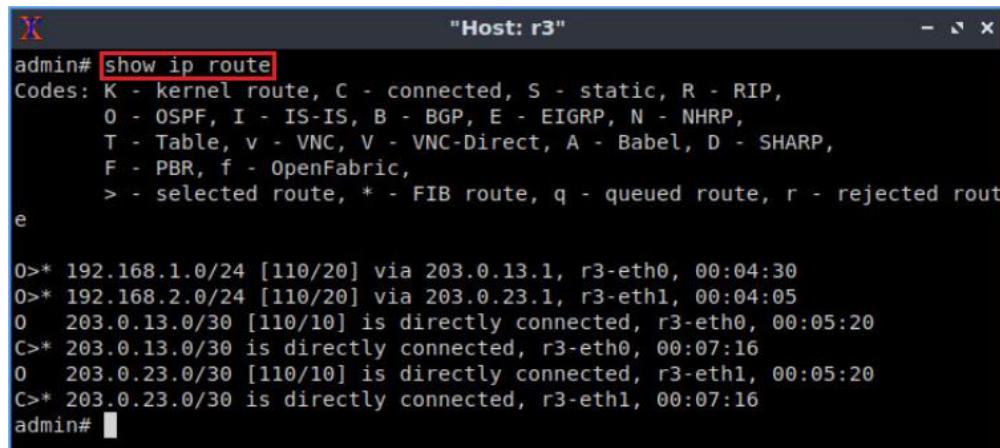
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# router ospf
admin(config-router)# network 0.0.0.0/0 area 0
admin(config-router)# end
admin# "
```

Figure 28. Summary of OSPF configuration in router r3.

**Step 10.** Type the following command in router the r3 terminal to verify the routing table of the router r3. You will notice that the routing table of router r3 is not aware of the network 10.0.0.0/8.

```
show ip route
```



```
"Host: r3"
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected route
e

0>* 192.168.1.0/24 [110/20] via 203.0.13.1, r3-eth0, 00:04:30
0>* 192.168.2.0/24 [110/20] via 203.0.23.1, r3-eth1, 00:04:05
0   203.0.13.0/30 [110/10] is directly connected, r3-eth0, 00:05:20
C>* 203.0.13.0/30 is directly connected, r3-eth0, 00:07:16
0   203.0.23.0/30 [110/10] is directly connected, r3-eth1, 00:05:20
C>* 203.0.23.0/30 is directly connected, r3-eth1, 00:07:16
admin#
```

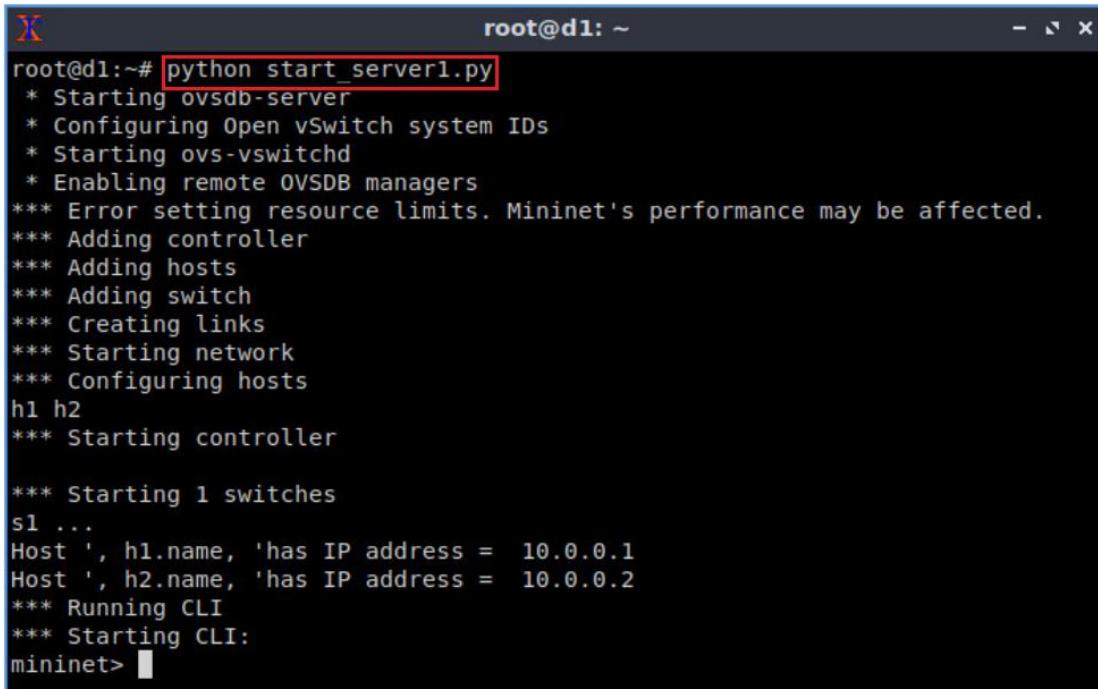
Figure 29. Displaying routing table of router r3.

#### 4 Run Mininet instances within the containers

The following section shows the steps to start Mininet in the containers and display the configuration files on the CLI.

**Step 1.** To open d1 container terminal, hold right-click on d1 and select Terminal. Type the following command to start Mininet. A topology that consists of two hosts connected to a switch is started.

```
python start_server1.py
```



```
root@d1:~# python start_server1.py
* Starting ovsdb-server
* Configuring Open vSwitch system IDs
* Starting ovs-vswitchd
* Enabling remote OVSDB managers
*** Error setting resource limits. Mininet's performance may be affected.
*** Adding controller
*** Adding hosts
*** Adding switch
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
Host ', h1.name, 'has IP address = 10.0.0.1
Host ', h2.name, 'has IP address = 10.0.0.2
*** Running CLI
*** Starting CLI:
mininet>
```

Figure 30. Starting a Mininet instance within container d1.

The figure shows a Mininet topology running within container d1. The information about the hosts is summarized after starting the switch s1.

**Step 2.** Run the following command to display the devices contained in the topology:

```
links
```

```
X root@d1: ~
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
mininet> █
```

Figure 31. Displaying the links between the devices in container d1.

The figure above shows that host h1 and switch s1 are connected via the interface pair *h1-eth0<->s1-eth1*. Similarly, host h2 is connected to switch s1 (*h2-eth0<->s1-eth2*).

**Step 3.** In container d2, type the following command to start a Mininet topology within the container. A topology that consists of two hosts connected to a switch is started.

```
python start_server2.py
```

```
X root@d2: ~# python start_server2.py
* Starting ovsdb-server
* Configuring Open vSwitch system IDs
* Starting ovs-vswitchd
* Enabling remote OVSDB managers
*** Error setting resource limits. Mininet's performance may be affected.
*** Adding controller
*** Adding hosts
*** Adding switch
*** Creating links
*** Starting network
*** Configuring hosts
h3 h4
*** Starting controller

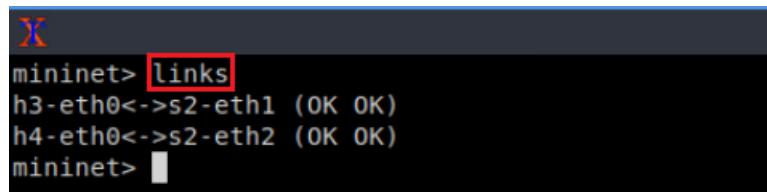
*** Starting 1 switches
s2 ...
Host h3 has IP address = 10.0.0.3
Host h4 has IP address = 10.0.0.4
*** Running CLI
*** Starting CLI:
mininet> █
```

Figure 32. Starting a Mininet instance within container d2.

The figure above starts a Mininet instance in container d2. Also, the information about the hosts is summarized after starting the switch s2.

**Step 4.** Run the following command to display the devices contained in the topology:

```
links
```



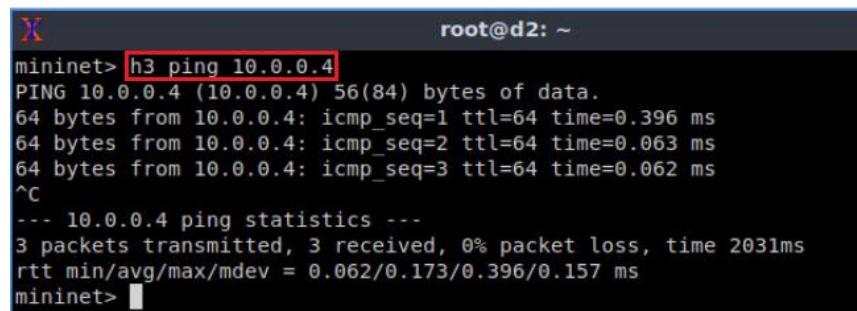
```
X
mininet> links
h3-eth0<->s2-eth1 (OK OK)
h4-eth0<->s2-eth2 (OK OK)
mininet> █
```

Figure 33. Displaying the links between the devices in container d2.

The figure shows that host h3 and switch s2 are connected via the interface pair *h3-eth0<->s2-eth1*. Similarly, host h4 is connected to switch s2 (*h4-eth0<->s2-eth2*).

**Step 5.** In container d2's terminal, issue the following command to verify the connectivity between host h3 and host h4.

```
h3 ping 10.0.0.4
```



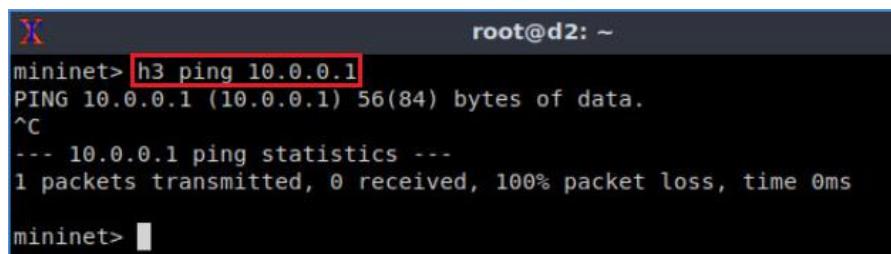
```
X                                     root@d2: ~
mininet> h3 ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.396 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.062 ms
^C
--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2031ms
rtt min/avg/max/mdev = 0.062/0.173/0.396/0.157 ms
mininet> █
```

Figure 34. Performing a connectivity test between host h3 and host h4.

The results will show a successful connectivity test since they belong to the same server. To stop the test, press **ctrl+c**.

**Step 6.** In container d1's terminal, issue the following command to verify the connectivity between host h3 and host h1.

```
h3 ping 10.0.0.1
```



```
X                                     root@d2: ~
mininet> h3 ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
^C
--- 10.0.0.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
mininet> █
```

Figure 35. Performing a connectivity test between host h3 and host h1.

There is no connectivity between hosts h3 and h1 since they are hosted on different servers. To stop the test, press **ctrl+c**.

## 5 Configuring GRE tunnel

In this section, you will configure GRE tunnel so that hosts from the different servers can communicate via a virtual tunnel (GRE tunnel).

**Step 1.** Type the following command to display all the interfaces within the docker container.

```
sh ifconfig
```

```
root@d1: ~
mininet> sh ifconfig
d1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.1.10 netmask 255.255.255.0 broadcast 0.0.0.0
                ether 66:52:19:56:7a:16 txqueuelen 1000 (Ethernet)
                RX packets 9262 bytes 723142 (723.1 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 162 bytes 13362 (13.3 KB)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.17.0.2 netmask 255.255.0.0 broadcast 172.17.255.255
                ether 02:42:ac:11:00:02 txqueuelen 0 (Ethernet)
                RX packets 170 bytes 17126 (17.1 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 112 bytes 10646 (10.6 KB)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

gre_sys: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 65000
        ether 42:e6:d9:91:8c:c9 txqueuelen 1000 (Ethernet)
        RX packets 50 bytes 3668 (3.6 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 69 bytes 4032 (4.0 KB)
```

Figure 36. Displaying interfaces within container d1.

The figure above shows that the interface *d1-eth0* has the IP address 192.168.1.10.

**Step 2.** Type the following command to display all the interfaces within the docker container.

```
sh ifconfig
```

```
root@d2: ~
mininet> sh ifconfig
d2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.2.10 netmask 255.255.255.0 broadcast 0.0.0.0
                ether fe:fd:e5:e0:b3:b1 txqueuelen 1000 (Ethernet)
                RX packets 9283 bytes 724752 (724.7 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 171 bytes 14068 (14.0 KB)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.17.0.3 netmask 255.255.0.0 broadcast 172.17.255.255
                ether 02:42:ac:11:00:03 txqueuelen 0 (Ethernet)
                RX packets 167 bytes 16820 (16.8 KB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 103 bytes 9870 (9.8 KB)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

gre_sys: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 65000
        ether 16:58:7a:df:f1:44 txqueuelen 1000 (Ethernet)
        RX packets 33 bytes 2338 (2.3 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 87 bytes 5768 (5.7 KB)
```

Figure 37. Displaying interfaces within container d2.

The figure above shows that the interface *d2-eth0* has the IP address 192.168.2.10.

**Step 3.** In this step, you will configure a GRE tunnel endpoint that will enable outgoing traffic from switch s1 to the external network. A script is written to facilitate this process. To execute the script, type the following command.

```
sh ./gre_tunnel_cmd1.cmd
```

The screenshot shows a terminal window titled 'root@d1: ~'. The command 'sh ./gre\_tunnel\_cmd1.cmd' is entered, followed by its output: 'The following command is executed: ovs-vsctl add-port s1 s1-tun -- set interface s1-tun type=gre option:remote\_ip=192.168.2.10'. The command and its output are highlighted with a red box.

Figure 38. Configuring GRE tunnel in switch s1.

Consider the figure above. The figure shows the command executed for creating the tunnel. Switch s1 will create a virtual port *s1-tun*. The interface is set to GRE type so that the port *s1-tun* will perform as an end of the GRE tunnel. The remote IP for s1 is 192.168.2.10, which is the IP address of container d2.

**Step 4.** Type the following command in docker d1 to verify the GRE configuration in switch s1.

```
sh ovs-vsctl show
```

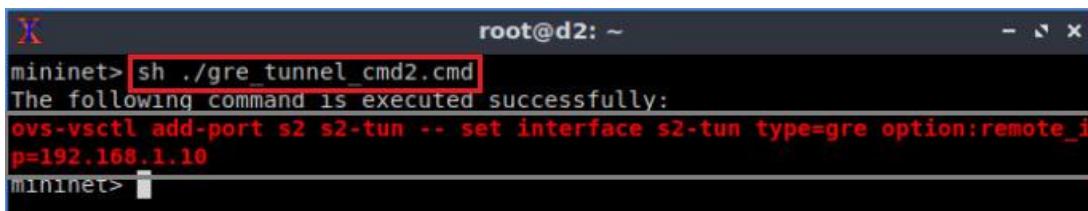
The screenshot shows a terminal window titled 'root@d1: ~'. The command 'sh ovs-vsctl show' is entered, displaying the Open vSwitch configuration. A specific section for the bridge 's1' is highlighted with a blue box, showing the port 's1-tun' with the 'Interface "s1-tun"' entry under it. This entry includes 'type: gre' and 'options: {remote\_ip="192.168.2.10"}'. The entire output of the 'ovs-vsctl show' command is shown, with the highlighted section being the key part of interest.

Figure 39. Verifying *s1-tun* port in switch s1.

The figure shows the port *s1-tun*, including the *remote\_ip=192.168.2.10*.

**Step 5.** In this step, you will configure a GRE tunnel endpoint that will enable outgoing traffic from switch s2 to the outer network. A script is written to facilitate this process. To execute the script, type the following command.

```
sh ./gre_tunnel_cmd2.cmd
```



```
root@d2: ~
mininet> sh ./gre_tunnel_cmd2.cmd
The following command is executed successfully:
ovs-vsctl add-port s2 s2-tun -- set interface s2-tun type=gre option:remote_ip=192.168.1.10
mininet>
```

Figure 40. Configuring GRE tunnel in switch s2.

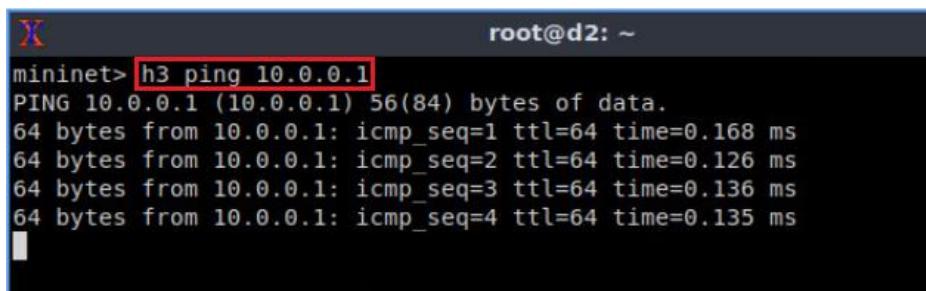
Consider the figure above. The figure shows the command executed for creating the tunnel. Switch s2 will create a virtual port s2-tun. The interface is set to GRE type so that the port s2-tun will perform as an end of the GRE tunnel. The remote IP for s2 is 192.168.1.10, which is the IP address of container d1.

## 6 Verifying GRE configuration

In this section, you will verify the GRE tunnel configuration.

**Step 1.** In container d2's terminal, issue the following command to verify the connectivity between host h1 and host h3.

```
h3 ping 10.0.0.1
```



```
root@d2: ~
mininet> h3 ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.168 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.126 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.136 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.135 ms
```

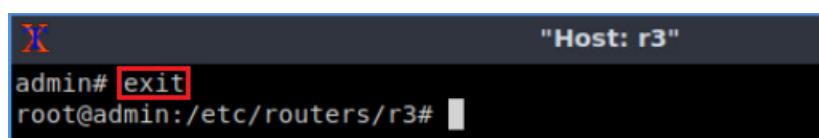
Figure 41. Performing a connectivity test between host h1 and host h3.

The result shows a successful connectivity test. Do not stop the test.

You will notice there is no physical connectivity between two servers, but they can communicate via GRE tunnel. Physically the traffic has been passed through the ISP network, but logically the servers are connected directly and maintain a secure tunneling process.

**Step 2.** In router r3's terminal, issue the following command.

```
exit
```



```
"Host: r3"
admin# exit
root@admin:/etc/routers/r3#
```

Figure 42. Exiting vtysh.

**Step 3.** Start Wireshark dissector by issuing the following command in router r3. A new window will emerge.

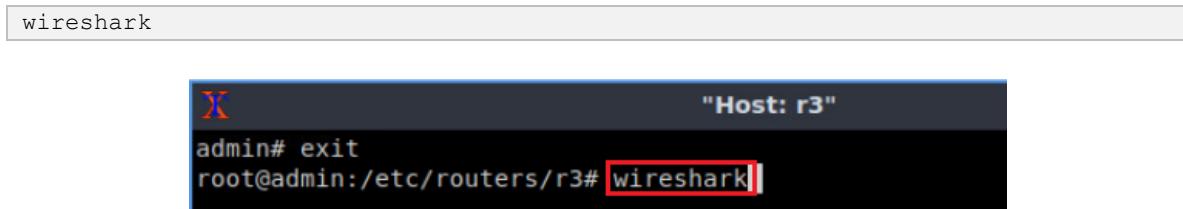


Figure 43. Starting Wireshark dissector.

**Step 4.** Click on the icon located on the upper left-hand side to start capturing packets on the interface *r3-eth0*.

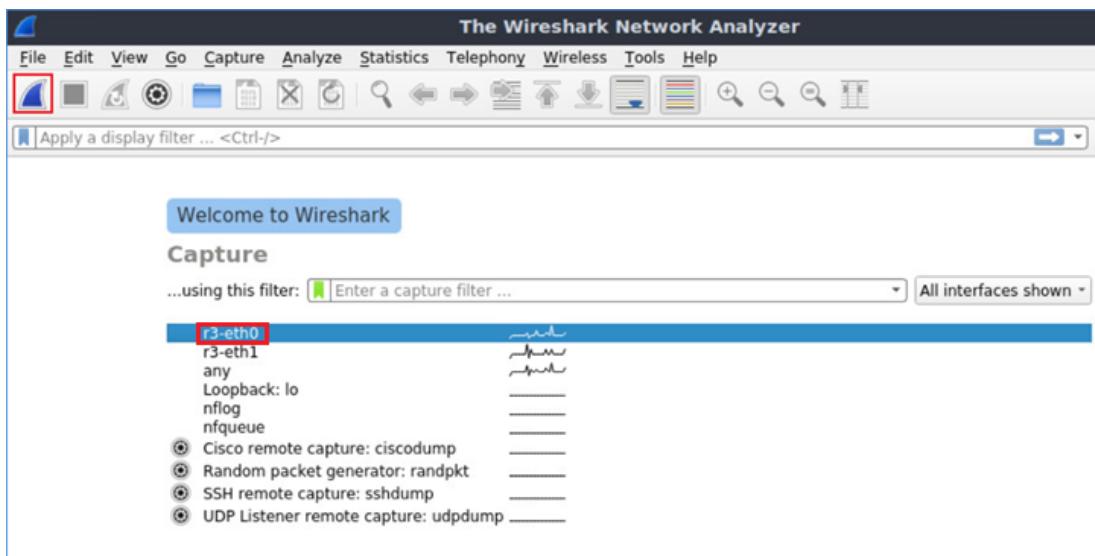


Figure 44. Starting packet capture.

**Step 5.** Click on the arrow located on the leftmost of the field called *Generic Routing Encapsulation*.

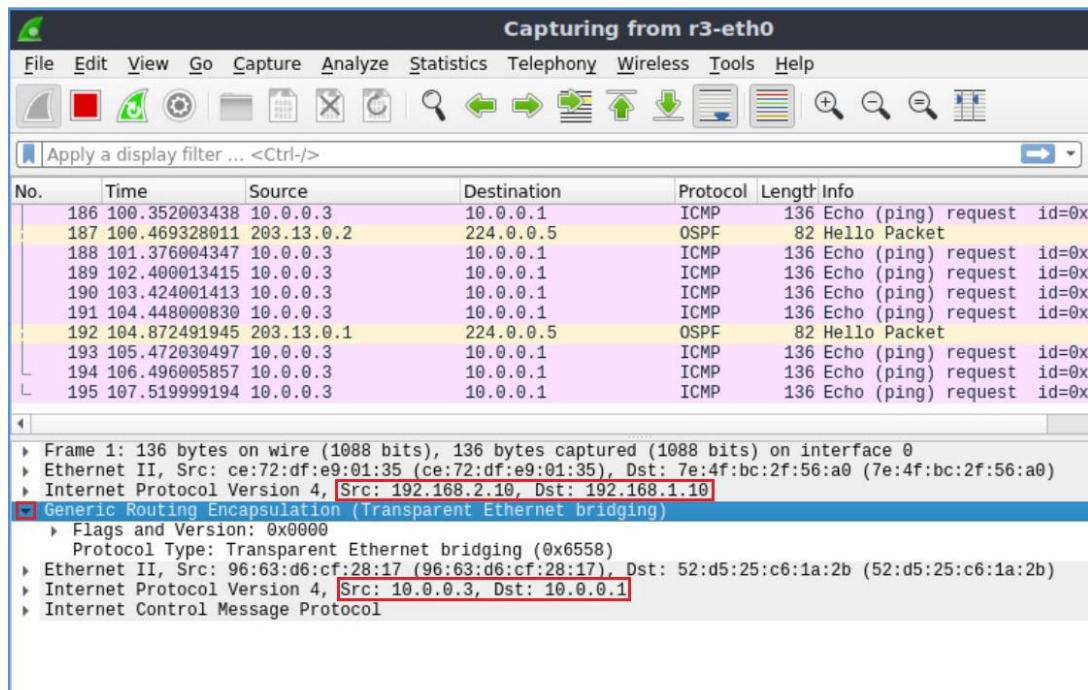


Figure 45. Filtering network traffic.

Consider the figure above. The source IP (10.0.0.3) and destination IP (10.0.0.1) belong to the network 10.0.0.0/8, which is unknown to router r3. GRE includes an outer IP header where it encapsulates the actual source and destination IPs. The outer IP header contains the new source IP (192.168.2.10) and destination IP (192.168.1.10) known by router r3. Router r3 will pass the traffic to router r2. Router r2 will decapsulate the packet, and the traffic will be forwarded to the actual destination (10.0.0.1).

**Step 6.** Click on the red button located on the upper left-hand side to stop packet capturing and close Wireshark.

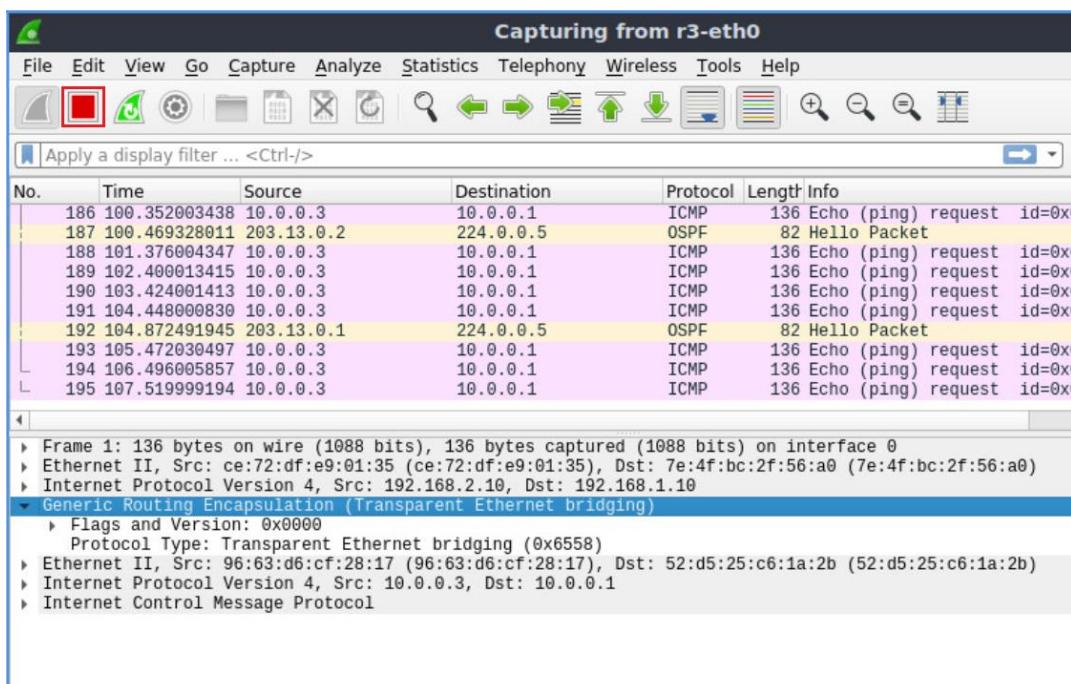


Figure 46. Stopping packet capture.

**Step 7.** In container d2, press **[ctrl+c]** to stop the test.

This concludes Lab 14. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. Mininet walkthrough, [Online]. Available: <http://mininet.org>.
2. Cisco, "Tunneling", [Online]. Available:  
<https://www.cisco.com/c/en/us/products/ios-nx-os-software/tunneling/index.html#:~:text=Tunneling%20is%20a%20technique%20that,through%20a%20public%20data%20network>.
3. Cisco, "How to configure a GRE tunnel", Mar 2019.
4. SearchNetworking, "Generic Routing Encapsulation (GRE)", Dec 2011.
5. Cloudflare, "What is GRE tunneling? | How GRE protocol works", 2020.
6. D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment." Linux journal 2014.239 (2014): 2.
7. Linux foundation, "Open vSwitch", [Online]. Available: <http://openvSwitch.org>.
8. Juniper Networks, "Generic Routing Encapsulation", [Online]. Available:  
<https://www.juniper.net/documentation/us/en/software/junos/interfaces-ethernet-switches/topics/topic-map/switches-interface-gre.html>



## OPEN VIRTUAL SWITCH

### Lab 15: Configuring IPsec Tunnel

Document Version: **09-16-2021**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput  
Networks for Big Science Data Transfers”

## Contents

Overview .....	3
Objectives.....	3
Lab settings .....	3
Lab roadmap .....	3
1    Introduction .....	3
1.1    Introduction to IPsec .....	4
1.2    IPsec in Open vSwitch .....	7
1.3    IPsec header .....	8
2    Lab topology.....	8
2.1    Lab settings.....	9
2.2    Loading a topology .....	10
2.3    Load the configuration file .....	11
2.4    Run the emulation.....	12
2.5    Verifying router configuration .....	13
3    Configuring OSPF in routers.....	16
4    Run Mininet instances within the containers.....	19
5    Configuring IPsec tunnel .....	21
5.1    Starting IPsec daemon in the containers .....	21
5.2    Configuring IPsec tunnel .....	22
6    Verifying tunnel configuration.....	23
References .....	28

## Overview

This lab presents Internet Protocol security (IPsec), a secure network protocol suite that ensures integrity, confidentiality, and authentication of data communication. It encrypts the data so that no one except the sender and the receiver can read the information. Authentication is needed to ensure that the receiver is connected to the actual sender. IPsec also provides integrity to make sure that no one changes the data in the packet. This lab aims to configure an IPsec tunnel to ensure security between two hosts.

## Objectives

By the end of this lab, you should be able to:

1. Understand the concept of IPsec.
2. Understand how IPsec works over GRE.
3. Emulate servers by using docker containers.
4. Configure IPsec tunnel between two servers.
5. Inspect data packets using Wireshark.

## Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

## Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Configuring OSPF in routers.
4. Section 4: Run Mininet instances within the containers.
5. Section 5: Configuring IPsec tunnel.
6. Section 6: Verifying tunnel configuration.

## 1      Introduction

A network security system helps to provide protection against crimes such as theft, fraud, and sabotage. Encryption is a method that is used regularly to protect and securely pass information online. This method relies on secret keys to encrypt and decrypt information as it passes between two parties<sup>1</sup>. A key is just a series of numbers shared between the sender and the receiver.

## 1.1 Introduction to IPsec

IPsec is a network protocol suite that authenticates and encrypts data to provide secure communication between networks. There are two principal protocols in IPsec: Authentication Header (AH) and Encapsulating Security Payload (ESP). AH provides source authentication and data integrity. ESP is much more widely used since it provides confidentiality in addition to authentication and data integrity<sup>2</sup>. The sender and receiver can verify if any changes have been made to the packet using a hash algorithm such as Message-digest v5 (MD5) and Secure Hash Algorithm (SHA).

Before sending data to the destination, the source and destination create a logical connection called Security Association (SA). They determine the IPsec protocols used for securing the packets and the keys<sup>3</sup>. The management of SA can be manual or through a key management protocol called Internet Key Exchange (IKE). IKE offers different types of authentications, such as the Pre-Shared-Key (PSK) and Rivest–Shamir–Adleman (RSA) algorithm. The PSK is a secret key value used in both hosts, which is combined with other information to create the authentication key. In RSA, the authentication key and identity information are used to create a hash that is encrypted with a private key. The encrypted hash is attached to the message and forwarded to the destination host. The encrypted hash can be decrypted using the public key of the sender. Diffie-Hellman (DH) is a public encryption method that establishes a shared secret key between two IPsec peers.

Consider Figure 1. The figure shows the IPsec framework. Rectangles highlighted in light blue are the protocols supported by Open vSwitch.

VPN features	Protocols			
IPsec Protocol	AH	ESP	ESP + AH	
Confidentiality	DES	3DES	AES	SEAL
Integrity	MD5	SHA		
Authentication	PSK	RSA		
Diffie-Hellman	DH1	DH2	...	DH14
				...

Figure 1. Suite of protocols used by Open vSwitch (light blue rectangles).

The connection between two peers is created in two phases. IKE phase 1 is used to create a secure tunnel that can be used in IKE phase 2. Two peers will negotiate about the encryption, authentication, hashing, and other protocols that they want to use. The Internet Security Association and Key Management Protocol (ISAKMP) is the negotiation protocol that let two hosts agree on building an IPsec SA<sup>4</sup>. In Phase 2, the participants negotiate the IPsec SAs for encrypting and authenticating the ensuing exchanges of user data.

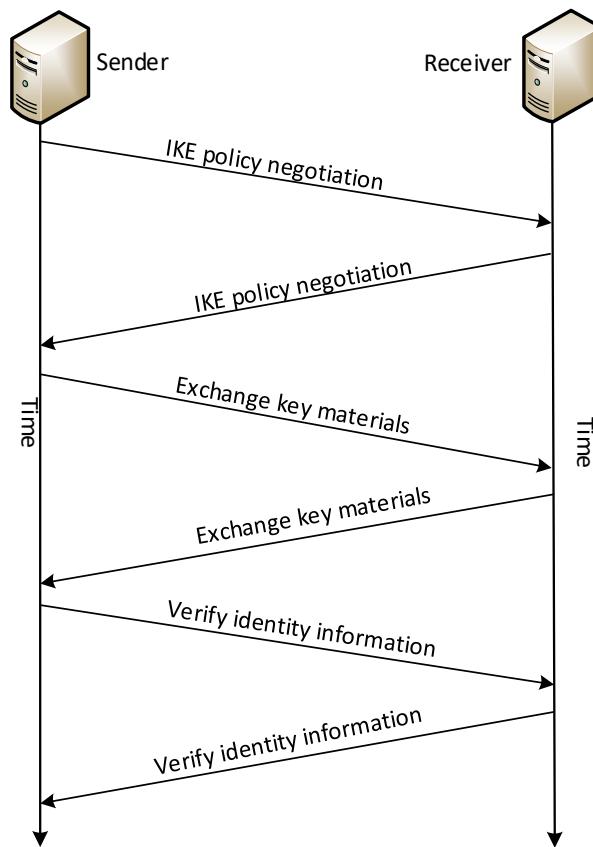


Figure 2. IKE phase 1.

Consider Figure 2. IKE phase 1 negotiates SA between two peers. Initially, the sender and the receiver negotiate parameters for setting up IKE SA. Then, they establish a secret key using DH key exchange. Finally, they exchange identity information and authenticate peer's identities. In this phase, both IKE peers ensure to use the same encryption algorithm, authentication algorithm, identity authentication method, and DH group ID. The key negotiated in phase 1 allows IKE peers to communicate securely in phase 2.

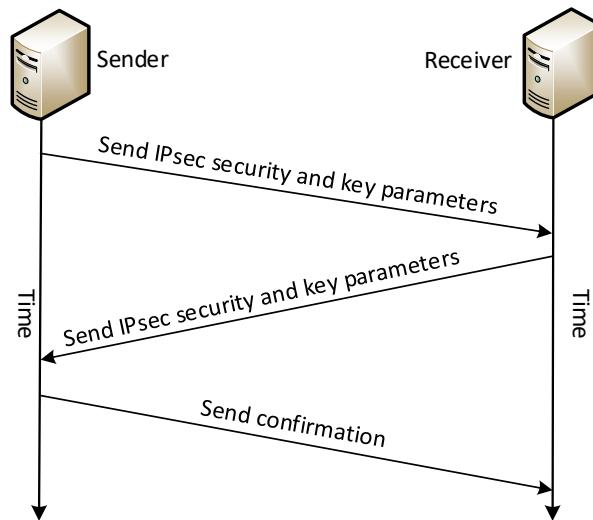


Figure 3. IKE phase 2.

Consider Figure 3. In IKE phase 2, the two IKE peers will continue to exchange key materials. Each peer will perform key computing and generate keys for IPsec SA encryption and authentication. In this way, each IPsec SA is guaranteed to use an absolutely unique key for the subsequent encryption and authentication of data transfers. In this phase, the message is encrypted by an encryption algorithm negotiated in IKE phase 1.

## 1.2 IPsec in Open vSwitch

In Open vSwitch, IPsec aims to provide encryption to Open vSwitch tunnels such as GRE. GRE is a protocol used for the encapsulation of a network layer protocol within another network layer protocol. It encapsulates data packets and redirects them to a device that de-encapsulates them and routes them to their final destination. This allows the source and destination hosts to operate as if they are connected via a virtual point-to-point connection (tunnel)<sup>7</sup>. By using IPsec, the entire GRE encapsulated packet is encrypted with an IPsec header. IPsec daemon and IKE daemon are responsible for creating the IPsec tunnel. IKE daemon uses the SHA algorithm and ESP protocol to provide authentication and encryption.

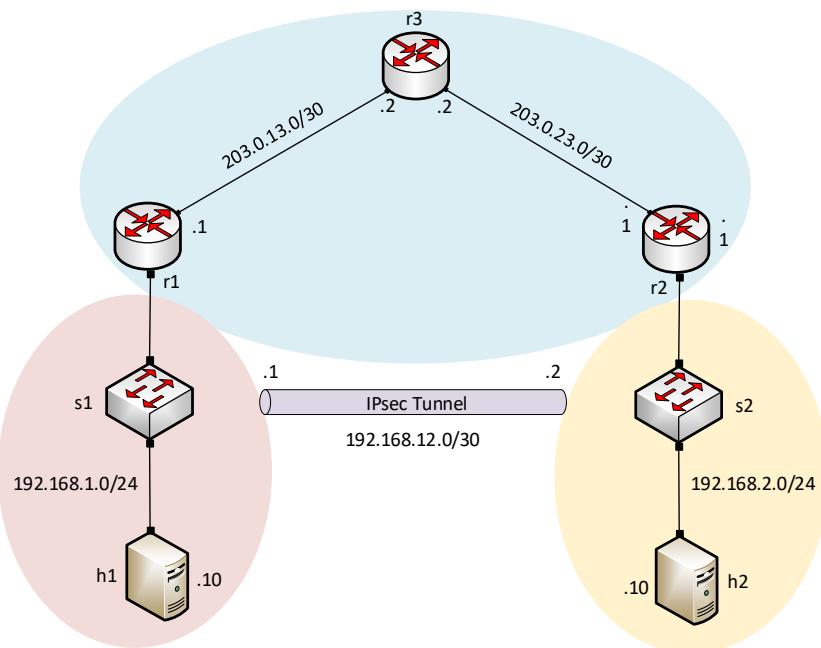


Figure 4. IPsec tunnel.

Consider Figure 4. Three routers can communicate with each other. Router r3 does not contain any route to the networks 192.168.1.0/24 and 192.168.2.0/24. GRE tunnel is configured between routers r1 and r2 (network 192.168.12.0/30). Whenever host h1 (192.168.1.10) wants to communicate with host h2 (192.168.2.10), router r1 will encapsulate the actual source and destination IP address and a GRE header will be added. The IPsec header will be placed on top of the GRE header to encrypt all the GRE information to secure the tunnel.

### 1.3 IPsec header

ESP header is placed over the GRE header to conceal all the information regarding GRE. ESP header includes Security Parameter Index (SPI), an identification tag to establish IPsec Security Association<sup>2</sup>. It allows the destination to identify which SA to use to check the security of the received packets. An attacker could try to capture packets even if the packets are encrypted. The ESP header uses a sequence number that is an increasing integer used to match up requests and responses and identify retransmissions of messages to mitigate the issue.

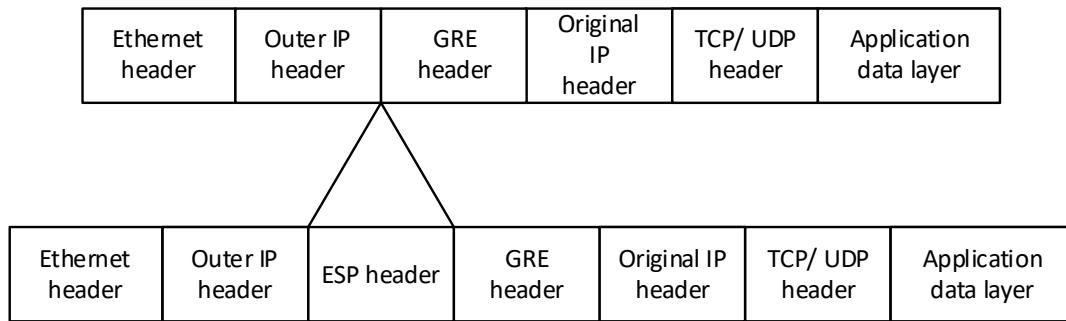


Figure 5. IPsec header.

## 2 Lab topology

Consider Figure 6. The topology consists of four hosts, two switches, and three routers. The end hosts and switches are running inside Server 1 and Server 2. Those servers are implemented by Docker<sup>6</sup> containers, which run Mininet instances. Docker is a platform that uses OS-level virtualization to deliver software packages called container. Routers are supported by the Free-range Routing (FRR) engine. Servers are connected to ISP routers. GRE tunnel is configured between servers so that the hosts within the servers assume they are directly connected through the GRE tunnel. Additionally, IPsec is running to ensure the security of the tunnel traffic.

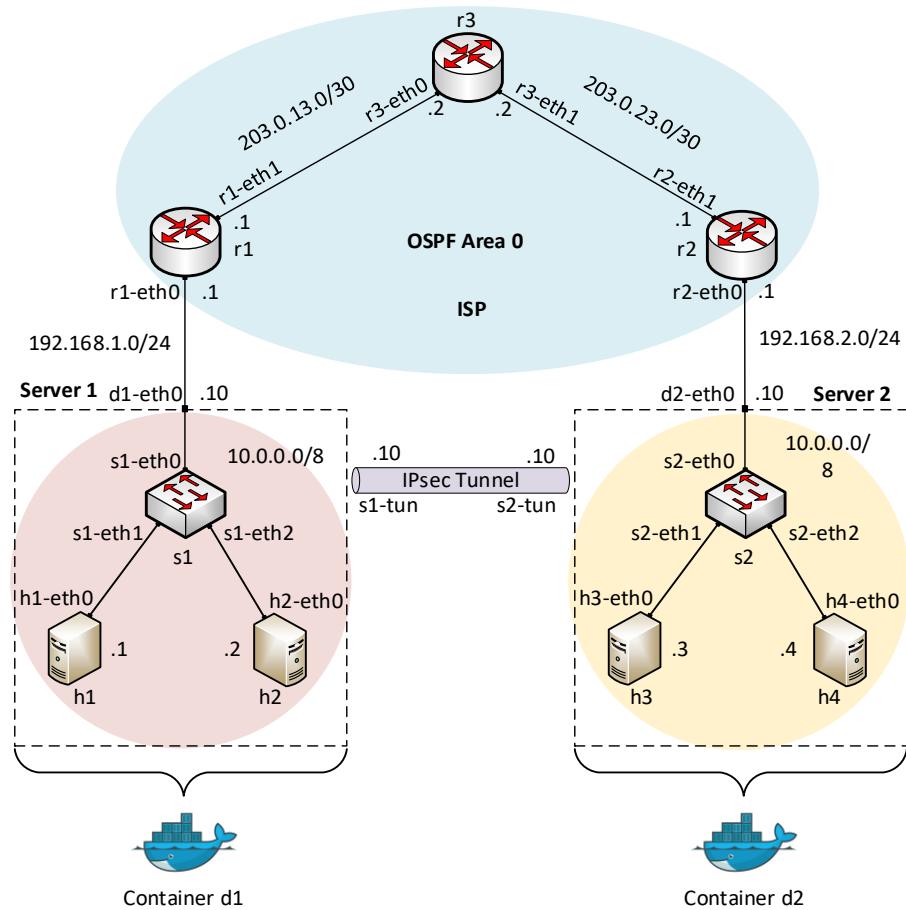


Figure 6. Lab topology.

## 2.1 Lab settings

The devices are configured according to Table 2.

Table 2. Topology information.

Device	Interface	IP Address	Subnet
<b>r1</b>	<b>r1-eth0</b>	192.168.1.1	/24
	<b>r1-eth1</b>	203.0.13.1	/30
<b>r2</b>	<b>r2-eth0</b>	192.168.2.1	/24
	<b>r2-eth1</b>	203.0.23.1	/30
<b>r3</b>	<b>r3-eth0</b>	203.0.13.2	/30
	<b>r3-eth1</b>	203.0.23.2	/30
<b>h1</b>	<b>h1-eth0</b>	10.0.0.1	/8

h2	h2-eth0	10.0.0.2	/8
h3	h3-eth0	10.0.0.3	/8
h4	h4-eth0	10.0.0.4	/8

## 2.2 Loading a topology

In this section, the user will open MiniEdit and load the lab topology. MiniEdit provides a Graphical User Interface (GUI) that facilitates the creation and emulation of network topologies in Mininet. This tool has additional capabilities such as configuring network elements (i.e., IP addresses, default gateway), saving topologies and exporting layer 2 models.

**Step 1.** A shortcut to MiniEdit is located on the desktop. Start MiniEdit by clicking on the MiniEdit shortcut. When prompted for a password, type `password`.

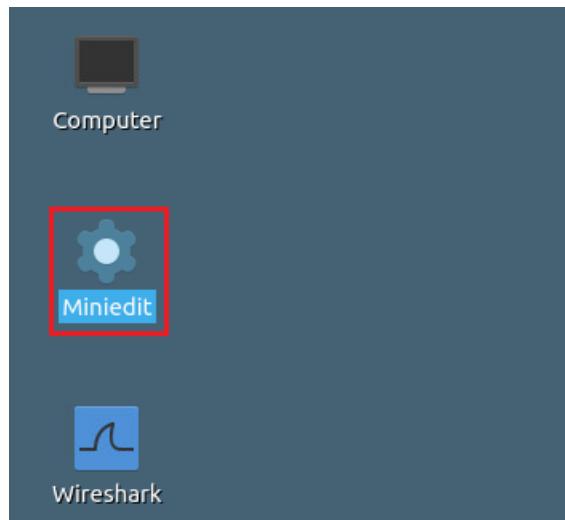


Figure 7. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File*, then *open* to load the lab's topology. Locate the *lab15.mn* topology file in the default directory, */home/ovs/OVS\_Labs/lab15* and click on *Open*.

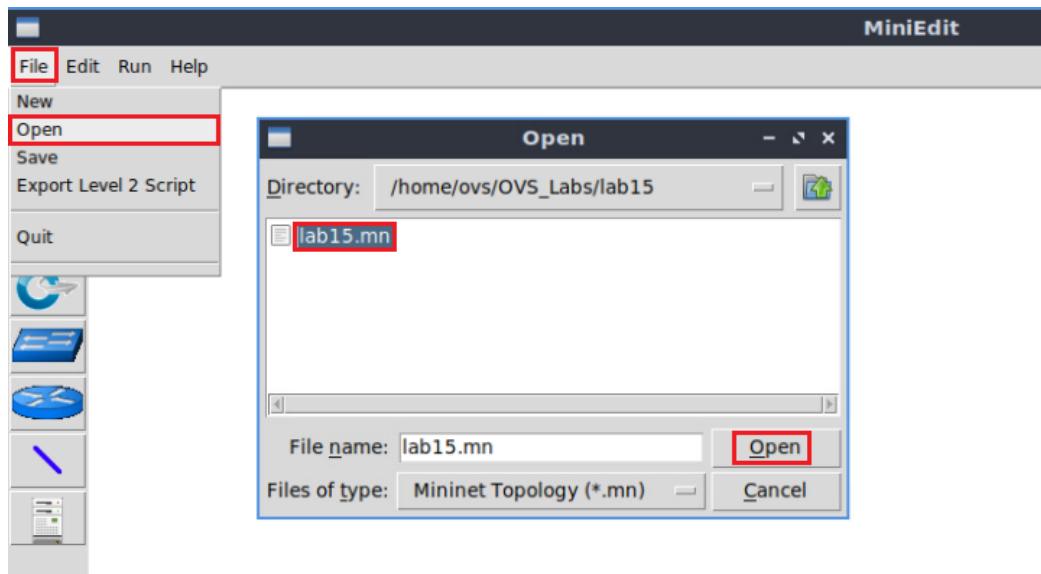


Figure 8. MiniEdit's Open dialog.

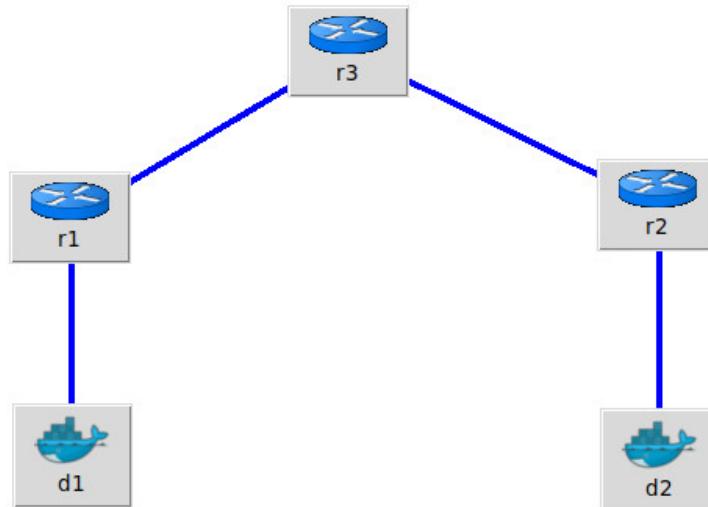


Figure 9. MiniEdit's topology.

## 2.3 Load the configuration file

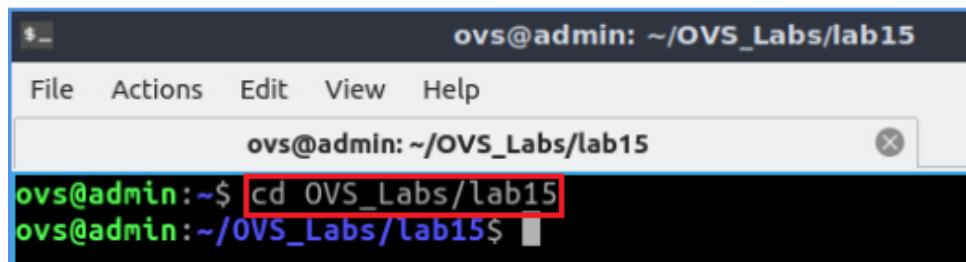
**Step 1.** Open the Linux terminal.



Figure 10. Opening Linux terminal.

**Step 2.** Navigate into *OVS\_Labs/lab15* directory by issuing the following command. This folder contains a configuration file, and the script is responsible for loading the configuration. The configuration file will assign the IP addresses to the routers' interfaces. The `cd` command is short for change directory, followed by an argument that specifies the destination directory.

```
cd OVS_Labs/lab15
```

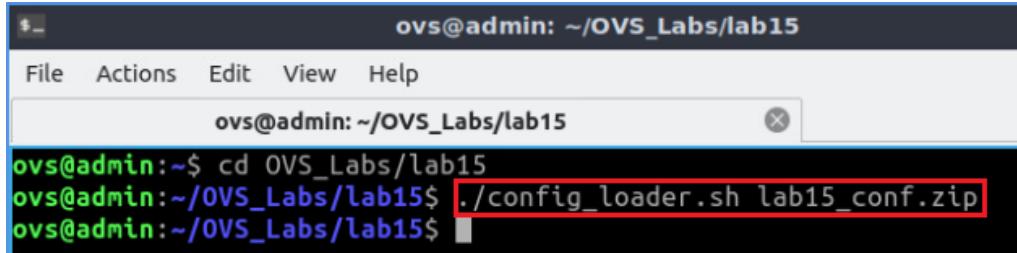


The screenshot shows a terminal window titled 'ovs@admin: ~/OVS\_Labs/lab15'. The user has typed 'cd OVS\_Labs/lab15' at the prompt. The command is highlighted with a red box.

Figure 11. Entering to the *OVS\_Labs/lab15* directory.

**Step 3.** To execute the shell script, type the following command. The argument of the program corresponds to the configuration zip file that will be loaded in all the routers in the topology.

```
./config_loader.sh lab15_conf.zip
```

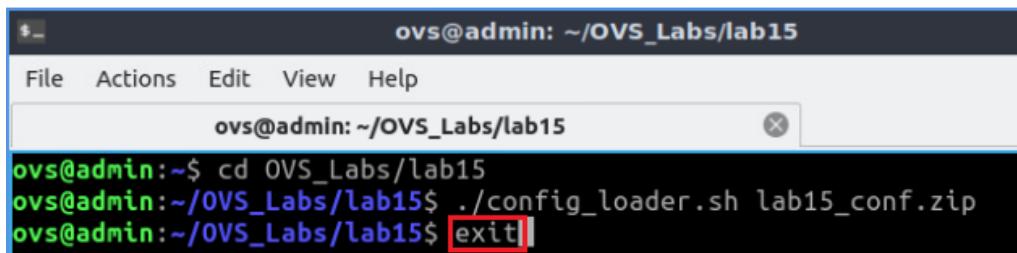


The screenshot shows a terminal window titled 'ovs@admin: ~/OVS\_Labs/lab15'. The user has typed './config\_loader.sh lab15\_conf.zip' at the prompt. The command is highlighted with a red box.

Figure 12. Executing the shell script to load the configuration.

**Step 4.** Type the following command to exit the Linux terminal.

```
exit
```



The screenshot shows a terminal window titled 'ovs@admin: ~/OVS\_Labs/lab15'. The user has typed 'exit' at the prompt. The command is highlighted with a red box.

Figure 13. Exiting from the terminal.

## 2.4 Run the emulation

**Step 1.** Click on the *Run* button to start the emulation. The emulation will start, and the MiniEdit panel buttons will gray out, indicating that they are currently disabled.



Figure 14. Starting the emulation.

**Step 2.** Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

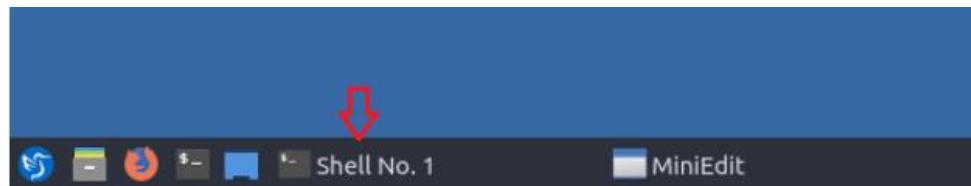


Figure 15. Opening Mininet's terminal.

**Step 3.** Issue the following command to display the interface names and connections.

```
links
```

A screenshot of the 'Shell No. 1' terminal window. The window has a dark blue header bar with the title 'Shell No. 1' and a dark blue footer bar. The main area of the window shows the output of the 'links' command. The output is as follows:  
`containernet> links  
r1-eth0<->d1-eth0 (OK OK)  
r2-eth0<->d2-eth0 (OK OK)  
r1-eth1<->r3-eth0 (OK OK)  
r3-eth1<->r2-eth1 (OK OK)  
containernet>`The first line 'containernet> links' is in a light gray box. The line 'r1-eth0<->d1-eth0 (OK OK)' is also in a light gray box.

Figure 16. Displaying network interfaces.

In Figure 16, the link displayed within the gray box indicates that interface *eth0* of router r1 connects to interface *eth0* of docker container d1 (i.e., *r1-eth0<->d1-eth0*).

## 2.5 Verifying router configuration

**Step 1.** In order to open router r1 terminal, hold right-click on router r1 and select Terminal.

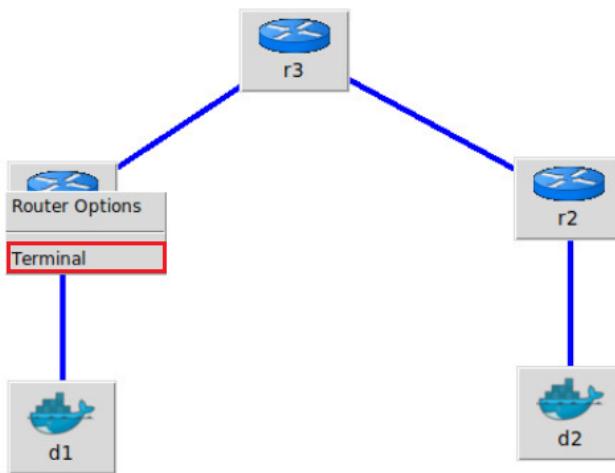


Figure 17. Opening a terminal on router r1.

**Step 2.** In router r1's terminal, you will start zebra daemon, which is a multi-server routing software that provides TCP/IP-based routing protocols. The configuration will not be working if you do not enable the zebra daemon initially. In order to start the zebra, type the following command.

```
zebra
```

```
X "Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1#
```

Figure 18. Starting zebra daemon.

**Step 3.** After initializing zebra, vtysh should be started to provide all the CLI commands defined by the daemons. To proceed, issue the following command.

```
vtysh
```

```
X "Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

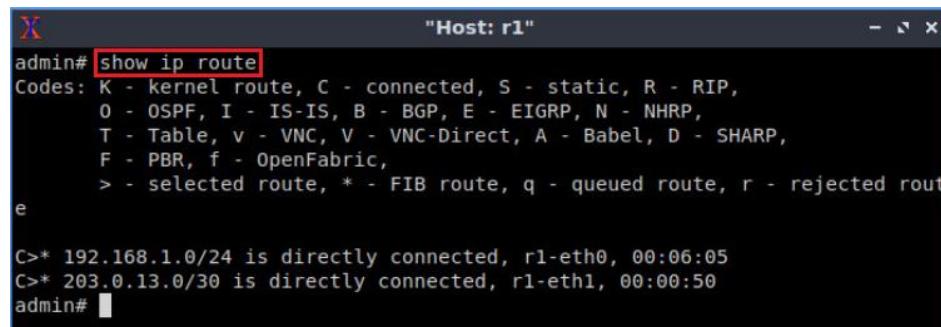
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin#
```

Figure 19. Starting `vtysh` in router r1.

**Step 4.** Type the following command in router r1's terminal to verify the routing table of router r1. It will list all the directly connected networks. The routing table of router r1 does not contain any route to the network attached to router r2 (203.0.23.0/30, 192.168.2.0/24). There is no routing protocol configured yet.

```
show ip route
```

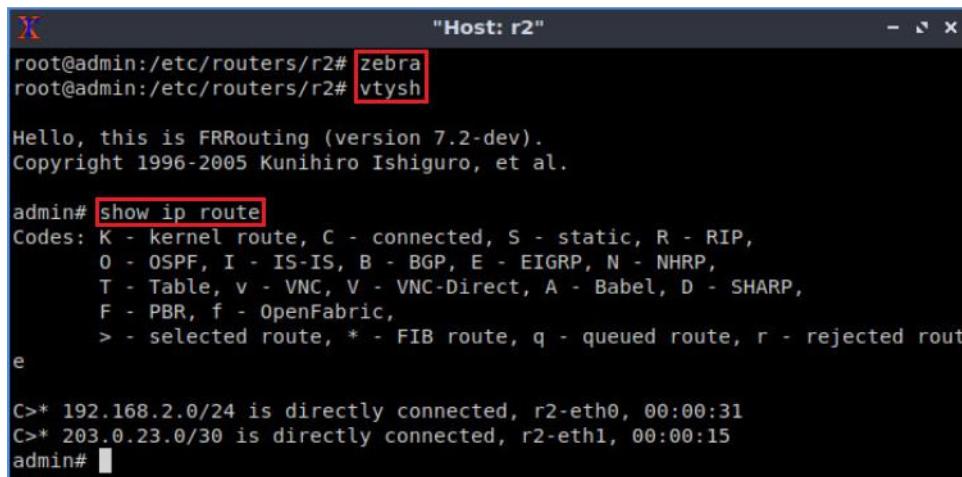


```
"Host: r1"
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       0 - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected route
e

C>* 192.168.1.0/24 is directly connected, r1-eth0, 00:06:05
C>* 203.0.13.0/30 is directly connected, r1-eth1, 00:00:50
admin#
```

Figure 20. Displaying routing table of router r1.

**Step 5.** Router r2 is configured similarly to router r1 but with different IP addresses (see Table 2). Those steps are summarized in the following figure. To proceed, in router r2's terminal, issue the commands depicted below. At the end, you will verify all the directly connected networks of router r2.



```
"Host: r2"
root@admin:/etc/routers/r2# zebra
root@admin:/etc/routers/r2# vtysh

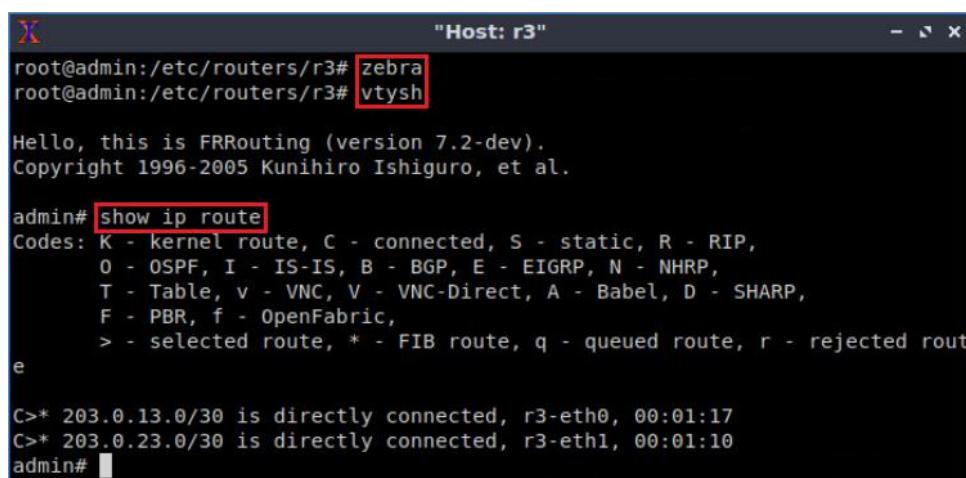
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       0 - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected route
e

C>* 192.168.2.0/24 is directly connected, r2-eth0, 00:00:31
C>* 203.0.23.0/30 is directly connected, r2-eth1, 00:00:15
admin#
```

Figure 21. Displaying routing table of router r2.

**Step 6.** Router r3 is configured similarly to router r1 but, with different IP addresses (see Table 2). Those steps are summarized in the following figure. To proceed, in router r3's terminal, issue the commands depicted below. At the end, you will verify all the directly connected networks of router r3.



```
"Host: r3"
root@admin:/etc/routers/r3# zebra
root@admin:/etc/routers/r3# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       0 - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected route
e

C>* 203.0.13.0/30 is directly connected, r3-eth0, 00:01:17
C>* 203.0.23.0/30 is directly connected, r3-eth1, 00:01:10
admin#
```

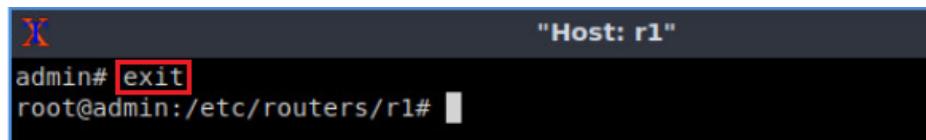
Figure 22. Displaying routing table of router r3.

### 3 Configuring OSPF in routers

At this point, routers are configured with their corresponding IP addresses (see Table 2). However, to provide end-to-end connectivity, it is necessary to enable and configure a routing protocol. In this section, you will configure OSPF as the routing protocol in routers r1, r2, and r3.

**Step 1.** Type the command shown below to close `vtysh`.

```
exit
```



A terminal window titled "Host: r1". The command "admin# exit" is being typed, with the word "exit" highlighted in red. The prompt "root@admin:/etc/routers/r1#" is visible at the bottom.

Figure 23. Exiting `vtysh`.

**Step 2.** To enable the OSPF daemon, issue the following command:

```
ospfd
```

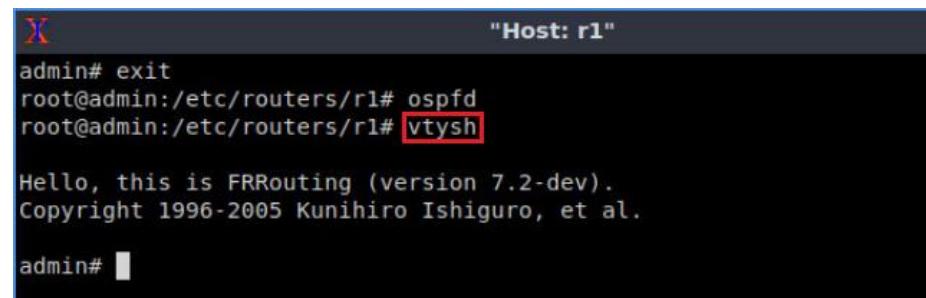


A terminal window titled "Host: r1". The command "root@admin:/etc/routers/r1# ospfd" is being typed, with "ospfd" highlighted in red. The prompt "root@admin:/etc/routers/r1#" is visible at the bottom.

Figure 24. Starting the `OSPF` daemon.

**Step 3.** To enable `vtysh` again, issue the following command:

```
vtysh
```

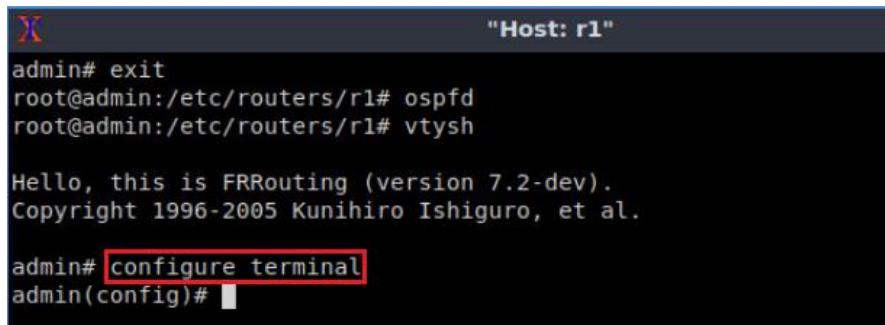


A terminal window titled "Host: r1". The command "root@admin:/etc/routers/r1# vtysh" is being typed, with "vtysh" highlighted in red. Below the command, the FRRouting copyright notice is displayed. The prompt "admin#" is visible at the bottom.

Figure 25. Starting `vtysh`.

**Step 4.** To enable the configuration mode in router r1, type the following command:

```
configure terminal
```



```
"Host: r1"
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

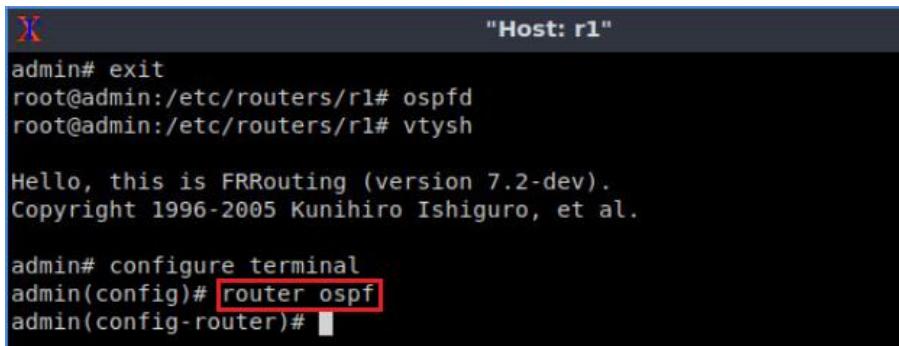
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# 
```

Figure 26. Starting router r1 in configuration mode.

**Step 5.** Issue the following command to configure OSPF in router r1.

```
router ospf
```



```
"Host: r1"
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

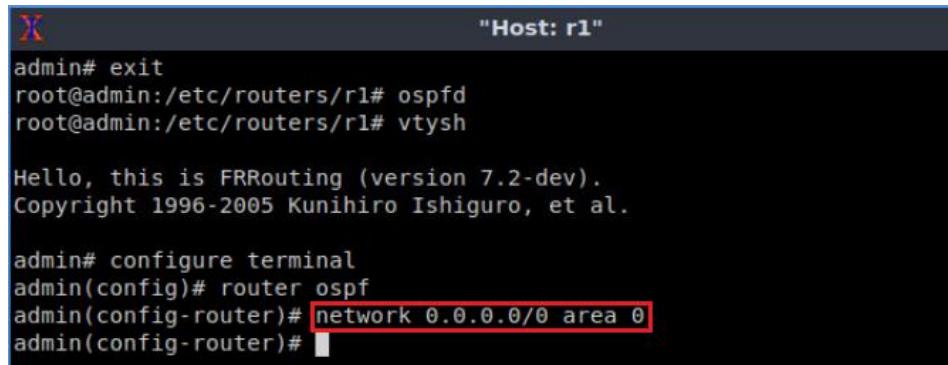
Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# router ospf
admin(config-router)# 
```

Figure 27. Configuring OSPF in router r1.

**Step 6.** Type the following command to assign a network and an area to router r1.

```
network 0.0.0.0/0 area 0
```



```
"Host: r1"
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# router ospf
admin(config-router)# network 0.0.0.0/0 area 0
admin(config-router)# 
```

Figure 28. Configuring OSPF network and area settings.

Consider the command above. OSPF will advertise all the networks connected to router r1.

**Step 7.** Issue the following command to end the configuration in router r1.

```
end
```

```
"Host: r1"
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# router ospf
admin(config-router)# network 0.0.0.0/0 area 0
admin(config-router)# end
admin# "
```

Figure 29. Finishing router r1 configuration.

**Step 8.** At this point, router r1 has configured OSPF. Proceed similarly on router r2 by following from step 1 to step 7. All those steps are summarized in the figure below.

```
"Host: r2"
admin# exit
root@admin:/etc/routers/r2# ospfd
root@admin:/etc/routers/r2# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# router ospf
admin(config-router)# network 0.0.0.0/0 area 0
admin(config-router)# end
admin# "
```

Figure 30. Summary of OSPF configuration in router r2.

**Step 9.** Proceed similarly on router r3 by following from step 1 to step 7. All those steps are summarized in the figure below.

```
"Host: r3"
admin# exit
root@admin:/etc/routers/r3# ospfd
root@admin:/etc/routers/r3# vtysh

Hello, this is FRRouting (version 7.2-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# configure terminal
admin(config)# router ospf
admin(config-router)# network 0.0.0.0/0 area 0
admin(config-router)# end
admin# "
```

Figure 31. Summary of OSPF configuration in router r3.

**Step 10.** Type the following command on router r3 terminal to verify the routing table of router r3. You will notice that the routing table of router r3 is not aware of the network 10.0.0.0/8.

```
show ip route
```

```
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
      O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
      T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
      F - PBR, f - OpenFabric,
      > - selected route, * - FIB route, q - queued route, r - rejected route
      e

O>* 192.168.1.0/24 [110/20] via 203.0.13.1, r3-eth0, 00:05:47
O>* 192.168.2.0/24 [110/20] via 203.0.23.1, r3-eth1, 00:05:48
O  203.0.13.0/30 [110/10] is directly connected, r3-eth0, 00:05:57
C>* 203.0.13.0/30 is directly connected, r3-eth0, 00:20:52
O  203.0.23.0/30 [110/10] is directly connected, r3-eth1, 00:05:58
C>* 203.0.23.0/30 is directly connected, r3-eth1, 00:20:43
admin#
```

Figure 32. Displaying routing table of router r3.

#### 4 Run Mininet instances within the containers

The section shows the steps to start Mininet in the containers and display on the CLI prompt the configuration files.

**Step 1.** In container d1, type the following command to start Mininet. A topology that consists of two hosts connected to a switch is started.

```
python3 start_server1.py
```

```
root@d1:/# python3 start_server1.py
ovsdb-server is already running.
ovs-vswitchd is already running.
Enabling remote OVSDB managers.
*** Error setting resource limits. Mininet's performance may be affected.
SIOCADDRT: File exists
*** Adding Controller
*** Adding hosts
*** Adding Switch
*** Creating Links
*** Starting network
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
Host ', h1.name, 'has IP address = 10.0.0.1
Host ', h2.name, 'has IP address = 10.0.0.2
*** Running CLI
*** Starting CLI:
mininet>
```

Figure 33. Starting a Mininet instance within container d1.

The figure above shows a Mininet topology running within container d1. The information about the hosts is summarized after starting switch s1.

**Step 2.** Run the following command to display the devices contained in the topology:

```
links
```

```
X root@d1: ~
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
mininet> 
```

Figure 34. Displaying the links between the devices in container d1.

The figure above shows that host h1 and switch s1 are connected via the interface pair  $h1\text{-}eth0 <-> s1\text{-}eth1$ . Similarly, host h2 is connected to the switch s1 ( $h2\text{-}eth0 <-> s1\text{-}eth2$ ).

**Step 3.** In container d2, type the following command to start a Mininet topology within the container. A topology that consists of two hosts connected to a switch is started.

```
python3 start_server2.py
```

```
X "Host: d2"
root@d2://# python3 start_server2.py
Starting ovsdb-server.
no default system version, please use --system-version ... failed!
Configuring Open vswitch system IDs.
Starting ovs-vswitchd.
Enabling remote OVSDB managers.
*** Error setting resource limits. Mininet's performance may be affected.
*** Adding Controller
*** Adding hosts
*** Adding Switch
*** Creating Links
*** Starting network
*** Configuring hosts
h3 h4
*** Starting controller

*** Starting 1 switches
s2 ...
Host ', h3.name, 'has IP address = 10.0.0.3
Host ', h4.name, 'has IP address = 10.0.0.4
*** Running CLI
*** Starting CLI:
mininet> 
```

Figure 35. Starting a Mininet instance within container d2.

The figure above shows a Mininet topology running within container d2. The information about the hosts is summarized after starting switch s2.

**Step 4.** Run the following command to display the devices contained in the topology:

```
links
```

```
X
mininet> links
h3-eth0<->s2-eth1 (OK OK)
h4-eth0<->s2-eth2 (OK OK)
mininet> 
```

Figure 36. Displaying the links between the devices in container d2.

The figure shows that the host h3 and switch s2 are connected via the interface pair  $h3\text{-}eth0 <\rightarrow s2\text{-}eth1$ . Similarly, host h4 is connected to the switch s2 ( $h4\text{-}eth0 <\rightarrow s2\text{-}eth2$ ).

**Step 5.** In container d2's terminal, issue the following command to verify the connectivity between hosts h3 and h4.

```
h3 ping 10.0.0.4
```

```
root@d2: ~
mininet> h3 ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.396 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.062 ms
^C
--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2031ms
rtt min/avg/max/mdev = 0.062/0.173/0.396/0.157 ms
mininet>
```

Figure 37. Performing a connectivity test between host h3 and host h4.

The result shows a successful connectivity test since they belong to the same server. To stop the test, press **ctrl+c**.

At this point, there is no connectivity between hosts h1 and h3 since they belong to different servers.

## 5 Configuring IPsec tunnel

In this section, you will configure a GRE tunnel so that hosts from different servers can communicate via a virtual tunnel. Additionally, you will configure IPsec in order to encrypt the data.

### 5.1 Starting IPsec daemon in the containers

**Step 1.** Type the following command to start Open vSwitch in container d1.

```
sh service openvswitch-switch start
```

```
"Host: d1"
mininet> sh service openvswitch-switch start
ovsdb-server is already running.
ovs-vswitchd is already running.
Enabling remote OVSDB managers.
mininet>
```

Figure 38. Starting Open vSwitch within the container d1.

**Step 2.** Type the following command to start the IPsec daemon in container d1. It will execute a shell script to start the daemon.

```
sh ./start_ipsec_daemon.cmd
```

```
"Host: d1"
mininet> sh ./start_ipsec_daemon.cmd
IPsec daemon started successfully!
mininet> 
```

Figure 39. Starting IPsec daemon within the container d1.

The following command is required to start the IPsec daemon, which is running in the script.

```
python3 test-monitor.py --pidfile=/var/run/openvswitch/ovs-monitor-ipsec.pid --ike-daemon=strongswan --log-file --detach --monitor unix:/var/run/openvswitch/db.sock
```

**Step 3.** Starting the Open vSwitch and IPsec daemon in container d2 following the same steps as container d1. All the steps are summarized in the following figure.

```
"Host: d2"
mininet> sh service openvswitch-switch start
ovsdb-server is already running.
ovs-vswitchd is already running.
Enabling remote OVSDB managers.
mininet> sh ./start_ipsec_daemon.cmd
IPsec daemon started successfully!
mininet> 
```

Figure 40. Starting the switch and the daemon within the container d2.

## 5.2 Configuring IPsec tunnel

**Step 1.** This step will configure a tunnel endpoint that will enable outgoing traffic from switch s2 to the external network. A script is written to facilitate this process. To execute the script, type the following command in docker d1.

```
sh ./ipsec_tunnel_cmd1.cmd
```

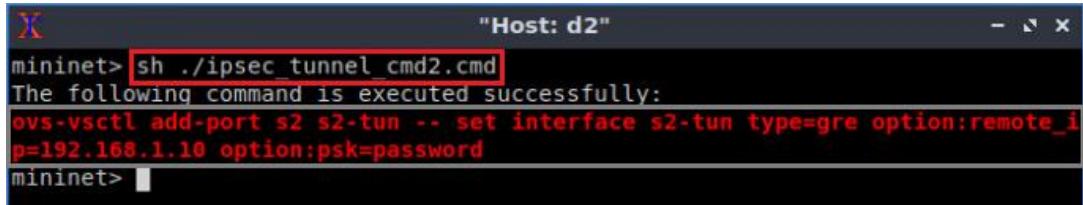
```
"Host: d1"
mininet> sh ./ipsec_tunnel_cmd1.cmd
The following command is executed successfully:
ovs-vsctl add-port s1 s1-tun -- set interface s1-tun type=gre option:remote_ip=192.168.2.10 option:psk=password
mininet> 
```

Figure 41. Configuring IPsec tunnel in switch s1.

Consider the figure above. The figure shows the command executed for creating the tunnel. Switch s1 will create a virtual port, s1-tun. The interface is set to GRE type so that the port s1-tun will perform as an end of the GRE tunnel. The remote IP for switch s1 is 192.168.2.10, which is the IP address of container d2. The tunnel includes PSK to ensure authentication between two peers.

**Step 2.** This step will configure a tunnel endpoint that will enable outgoing traffic from switch s2 to the external network. A script is written to facilitate this process. To execute the script, type the following command in docker d2.

```
sh ./ipsec_tunnel_cmd2.cmd
```



```
"Host: d2"
mininet> sh ./ipsec_tunnel_cmd2.cmd
The following command is executed successfully:
ovs-vsctl add-port s2 s2-tun -- set interface s2-tun type=gre option:remote_ip=192.168.1.10 option:psk=password
mininet>
```

Figure 42. Configuring IPsec tunnel in switch s2.

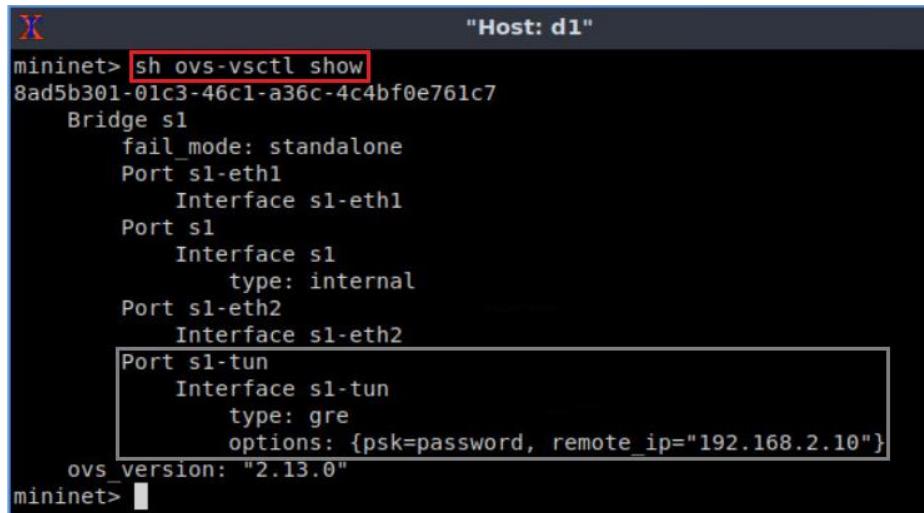
Consider the figure above. The figure shows the command executed for creating the tunnel. Switch s2 will create a virtual port, s2-tun. The interface is set to GRE type so that the port s2-tun will perform as an end of the GRE tunnel. The remote IP for switch s2 is 192.168.1.10, which is the IP address of container d1. The tunnel includes a PSK to ensure authentication between two peers.

## 6 Verifying tunnel configuration

In this section, you will verify the tunnel configuration.

**Step 1.** Type the following command in container d1 to verify the tunnel configuration in switch s1.

```
sh ovs-vsctl show
```



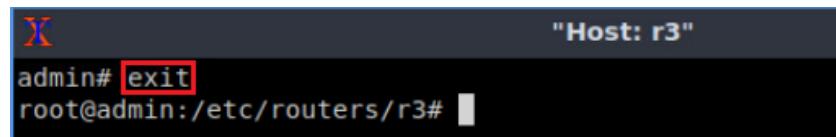
```
"Host: d1"
mininet> sh ovs-vsctl show
8ad5b301-01c3-46c1-a36c-4c4bf0e761c7
    Bridge s1
        fail_mode: standalone
        Port s1-eth1
            Interface s1-eth1
        Port s1
            Interface s1
                type: internal
        Port s1-eth2
            Interface s1-eth2
        Port s1-tun
            Interface s1-tun
                type: gre
                options: {psk=password, remote_ip="192.168.2.10"}
    ovs_version: "2.13.0"
mininet>
```

Figure 43. Verifying s1-tun port in switch s1.

The figure shows the port *s1-tun*, including the *remote\_ip=192.168.2.10* and PSK is password.

**Step 2.** In the router r3's terminal, issue the following command.

```
exit
```

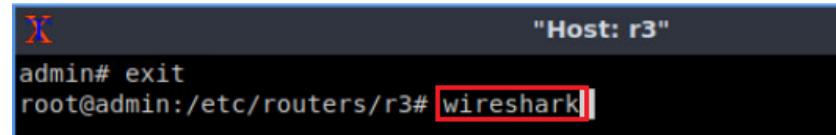


```
"Host: r3"
admin# exit
root@admin:/etc/routers/r3#
```

Figure 44. Exiting vtysh.

**Step 3.** Start Wireshark by issuing the following command in router r3. A new window will emerge.

```
wireshark
```



```
"Host: r3"
admin# exit
root@admin:/etc/routers/r3# wireshark
```

Figure 45. Starting Wireshark dissector.

**Step 4.** Click on the icon located on the upper left-hand side to start capturing packets on the interface *r3-eth1*.

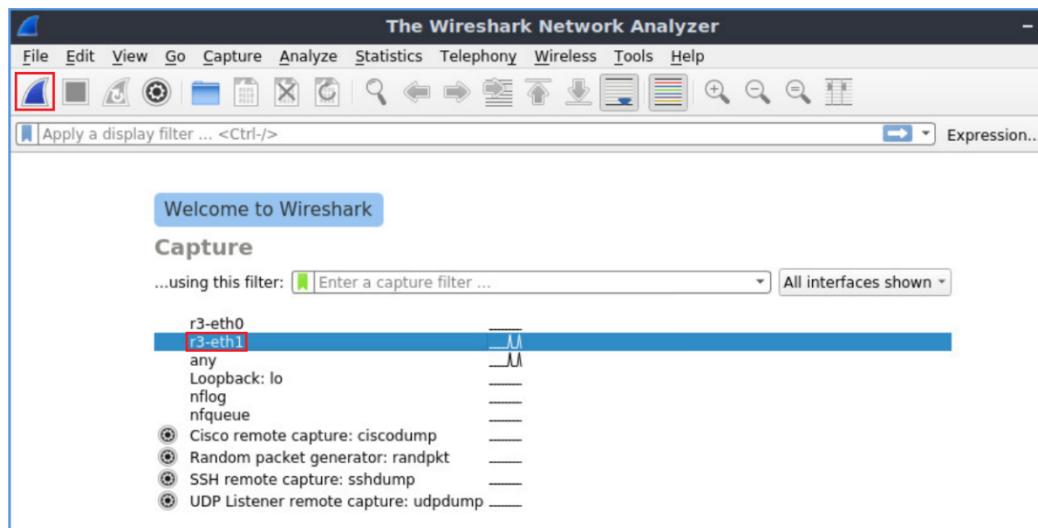
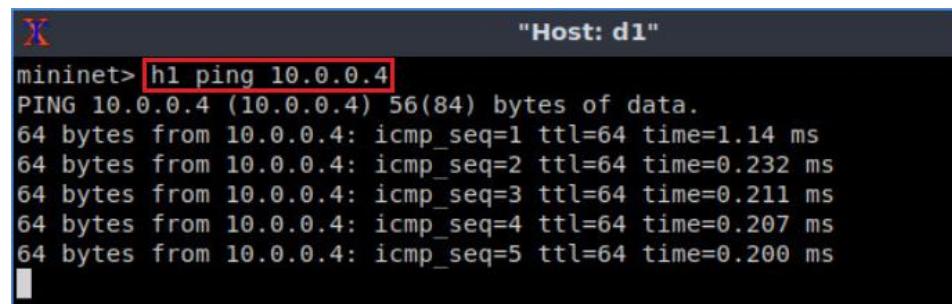


Figure 46. Starting packet capture.

**Step 5.** In container d1's terminal, issue the following command to verify the connectivity between hosts h1 and h4.

```
h1 ping 10.0.0.4
```



```
"Host: d1"
mininet> h1 ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=1.14 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.232 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.211 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.207 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.200 ms
```

Figure 47. Performing a connectivity test between hosts h1 and h4.

The result shows a successful connectivity test. Do not stop the connectivity test.

It may take few seconds to receive a reply from host h4. Meanwhile, do not stop the connectivity test.

**Step 6.** In the filter box located on the upper left-hand side, type *isakmp* to filter Internet Security Association and Key Management Protocol (ISAKMP) packets.

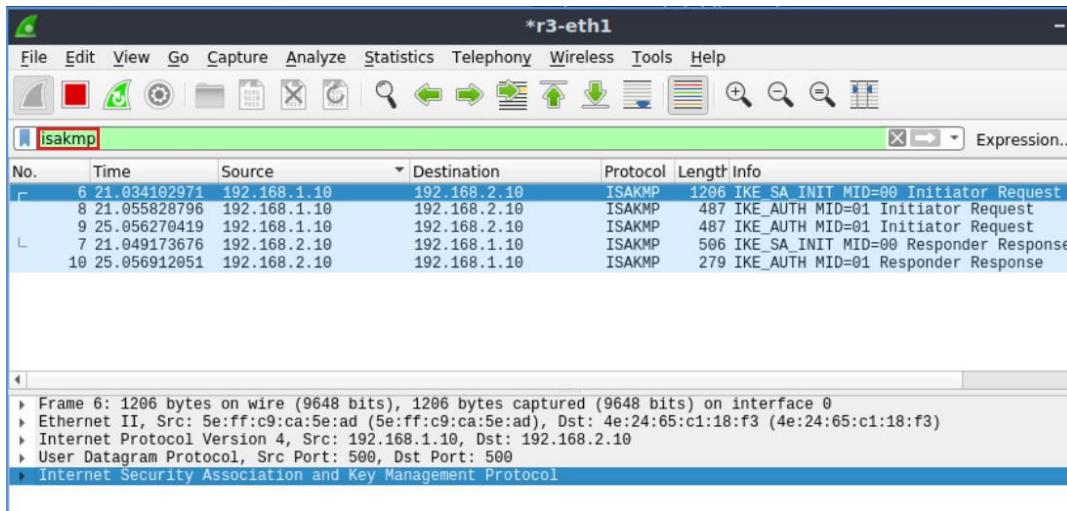


Figure 48. Filtering network traffic.

Consider the figure above. This is the first phase of IKE, where the key is being exchanged between the peers to ensure authentication. The hashing algorithm will be processed to make sure the peers are communicating in a secured manner.

You will see the ISAKMP packets only when the peers are generating a new connection. If you capture packets for the second time, you will only see the ESP packets.

**Step 7.** Click on any of the packets. A new window will be prompted, which includes all the information regarding the ISAKMP protocol.

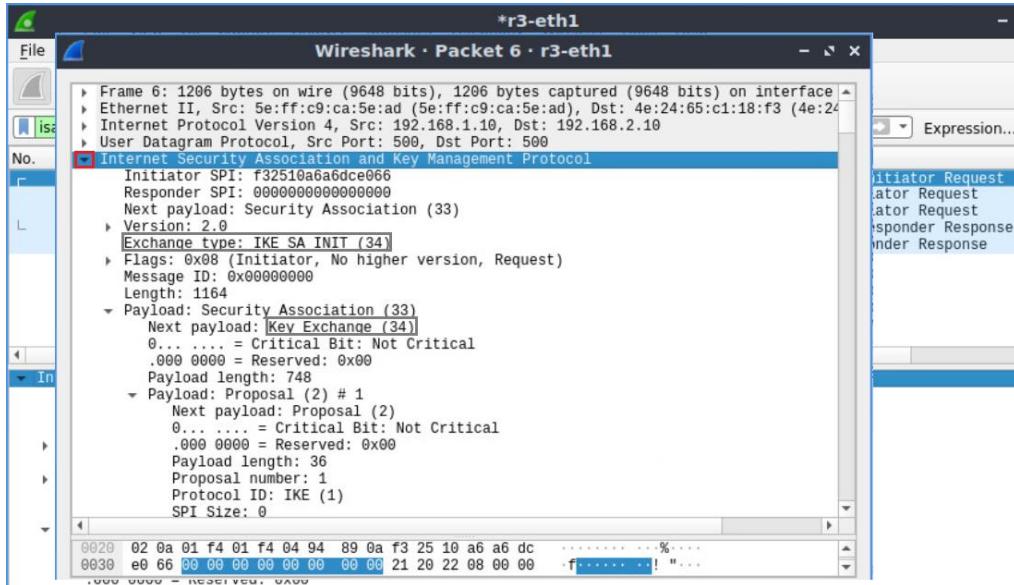


Figure 49. Displaying ISAKMP information.

The figure shows the ISAKMP key exchange information.

#### Step 8. Close the new window.

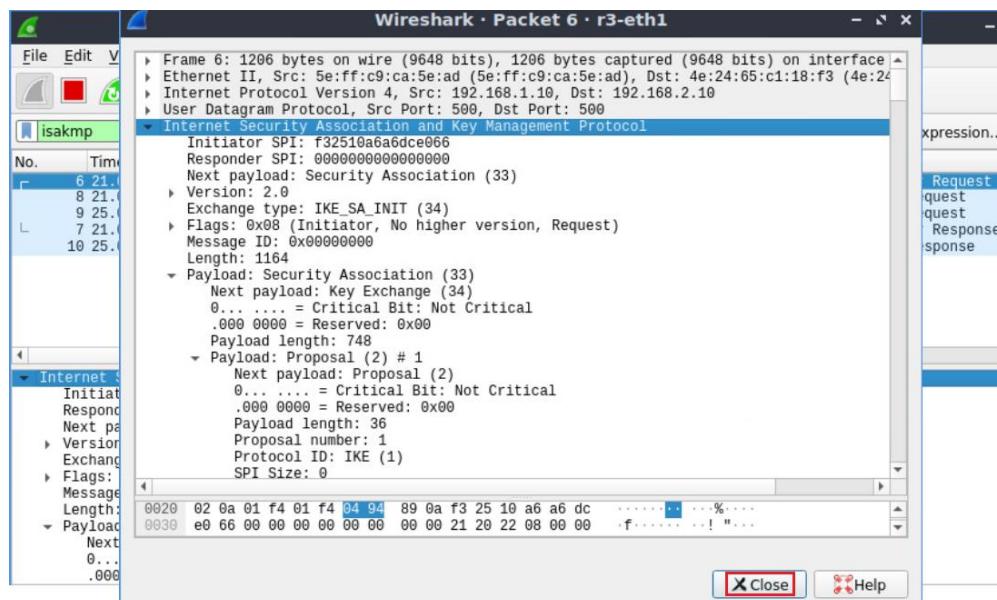


Figure 50. Closing the new window.

#### Step 9. In the filter box located on the upper left-hand side, type `esp` to filter ESP packets.

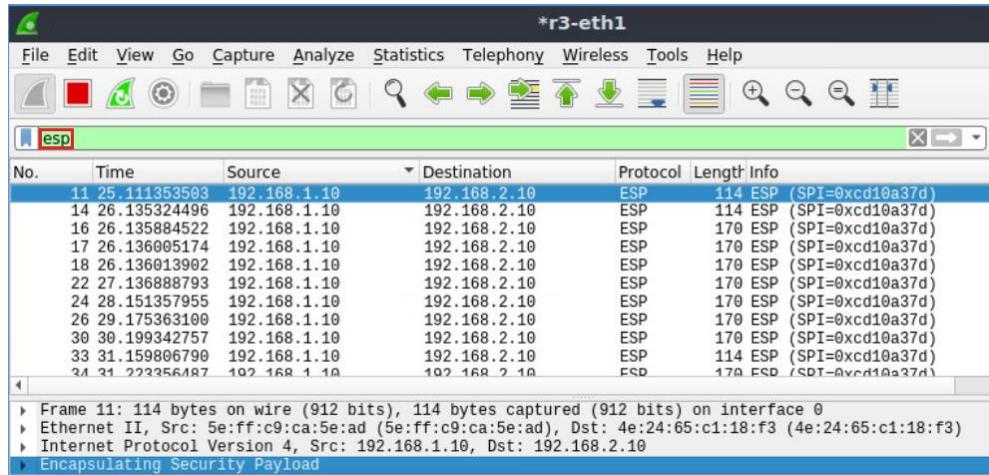


Figure 51. Filtering network traffic.

**Step 10.** Click on the arrow located on the left most of the field called *Encapsulating Security Payload*.

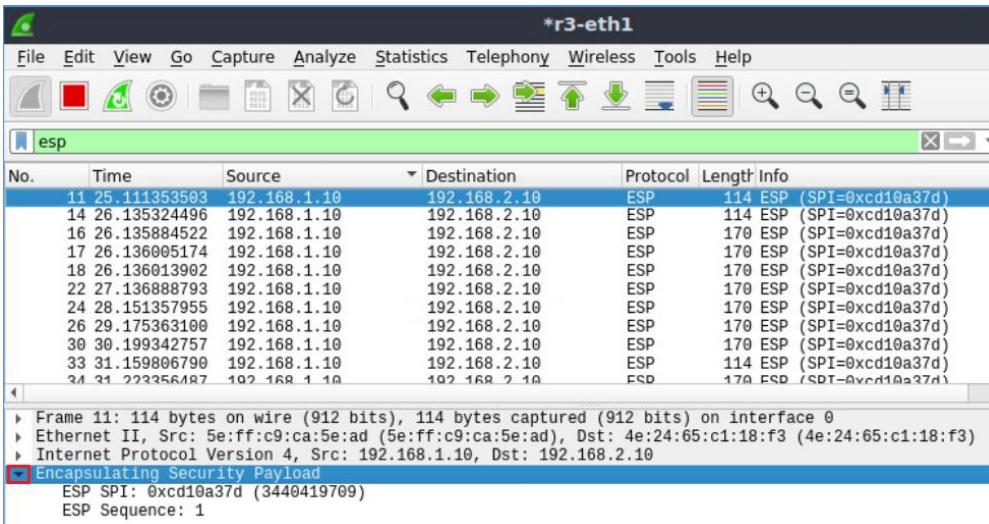


Figure 52. Verifying data encryption.

Consider the figure above. This is the second phase of the IKE protocol. In the figure, you can only see the ESP SPI and the ESP sequence number. You will not see the actual source (10.0.0.1) and destination IP address (10.0.0.4) since all the data are encrypted.

**Step 11.** Click on the red button located on the upper left-hand side to stop packet capturing and close Wireshark.

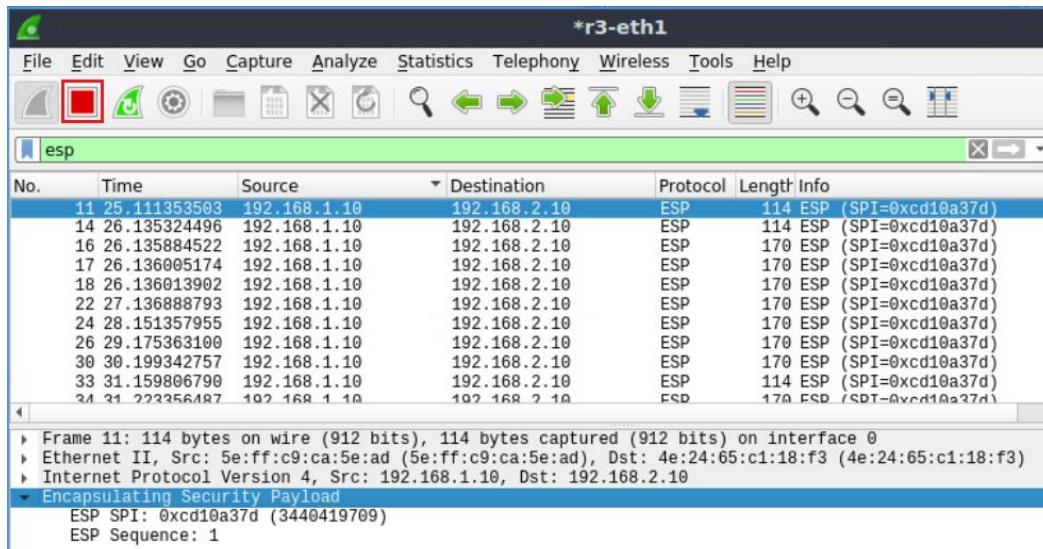


Figure 53. Stopping packet capture.

**Step 12.** In container d1, press **ctrl+c** to stop the test.

This concludes Lab 15. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. L. Shinder, M.Cross, “*Scene of cybercrime*”, 2<sup>nd</sup> edition, 2008.
2. J. Kurose, K. Ross, “*Computer Networking A top-down-approach*”, 6<sup>th</sup> Edition, 2017.
3. N. Doraswamy, D. Harkins, “*IPsec the new security standard for the Internet, Intranets, and Virtual Private Networks*”, 2<sup>nd</sup> Edition, 2003.
4. Cisco, “*IPsec and ISAKMP*” [Online]. Available:  
<https://www.cisco.com/c/en/us/td/docs/security/asa/asa94/config-guides/cli/vpn/asa-94-vpn-config/vpn-ike.pdf>
5. NetworkLessons.com, “*IPsec (Internet Protocol Security)*”, [Online], Available: IPsec  
<https://networklessons.com/cisco/ccie-routing-switching/ipsec-internet-protocol-security>
6. Cisco, “*How to configure a GRE tunnel*”, Mar 2019.
7. SearchNetworking, “*Generic Routing Encapsulation (GRE)*”, Dec 2011.
8. D. Merkel, “*Docker: lightweight Linux containers for consistent development and deployment*” *Linux journal* 2014.239 (2014): 2.
9. Linux Foundation, “*Encrypt Open vSwitch tunnels with IPsec*”, [Online]. Available:  
<https://docs.openvswitch.org/en/latest/howto/ipsec/>
10. Linux Foundation, “*OVS IPsec tutorial*”, [Online], Available:  
<https://docs.openvswitch.org/en/latest/tutorials/ipsec/>
11. Mininet walkthrough, [Online]. Available: <http://mininet.org>.