UNIVERSITY OF
SOUTH CAROLINA

# SOFTWARE DEFINED NETWORKING LAB SERIES

**Book Version: 07-08-2021**

Principal Investigator: Jorge Crichigno

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

# Contents

# SOFTWARE DEFINED NETWORKING

# Lab 1: Introduction to Mininet

**Document Version: 03-30-2021**

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

## Contents

## 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 how to build and emulate topologies using a graphical user interface (GUI) application. In this lab we will use Containernet, a Mininet network emulator fork that allows the use of 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 the 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[1]. 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.

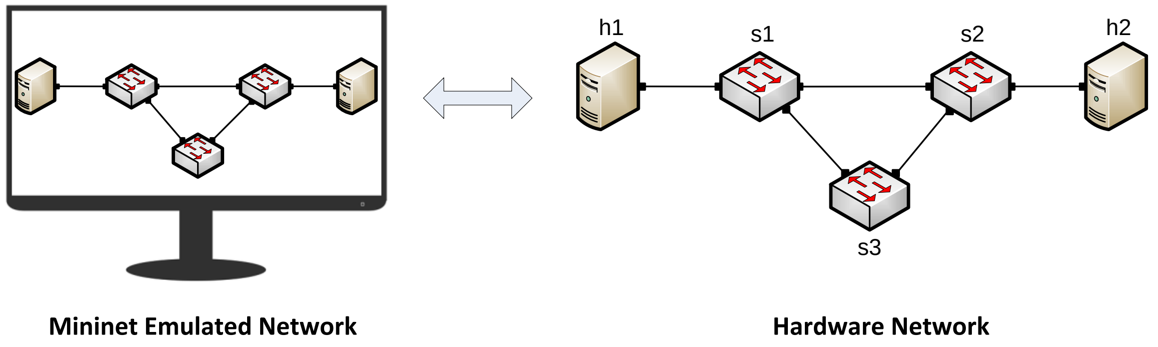


Figure 1. Hardware network vs. Mininet emulated network.

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*[2] and *Software-Defined Networking (SDN)*[3]. This lab, however, only focuses on emulating a simple network environment without SDN-based 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[4]. 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.

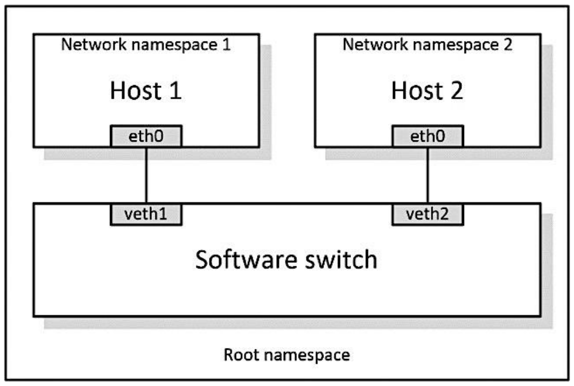


Figure 2. Network namespaces and virtual Ethernet links.

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.

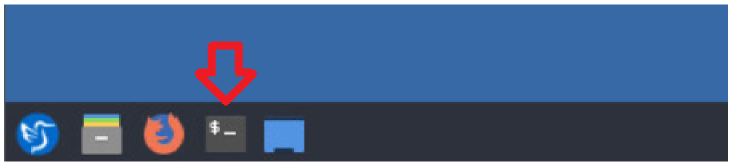
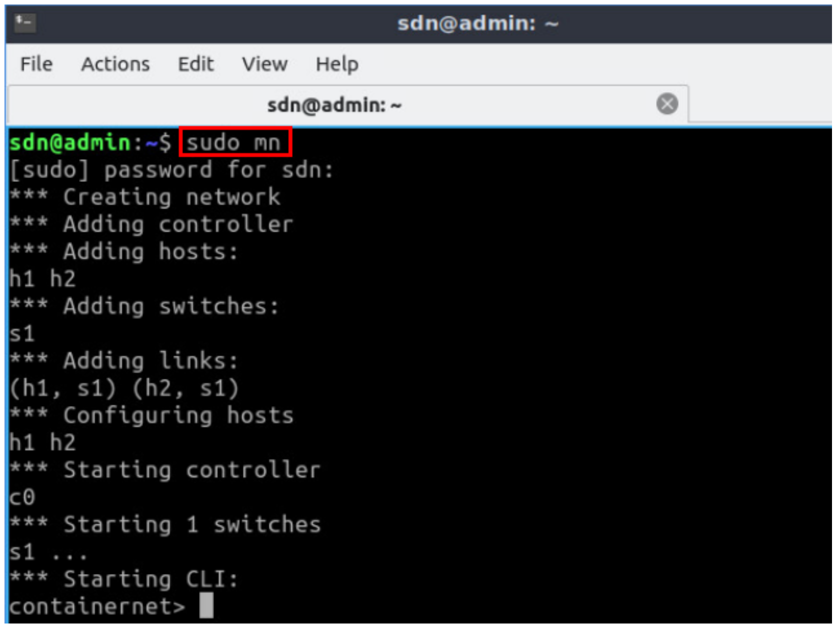


Figure 3. Shortcut to open a Linux terminal.

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
```



```
sdn@admin:~$ sudo mn
[sudo] password for sdn:
*** 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. The loaded topology matches the figure below.

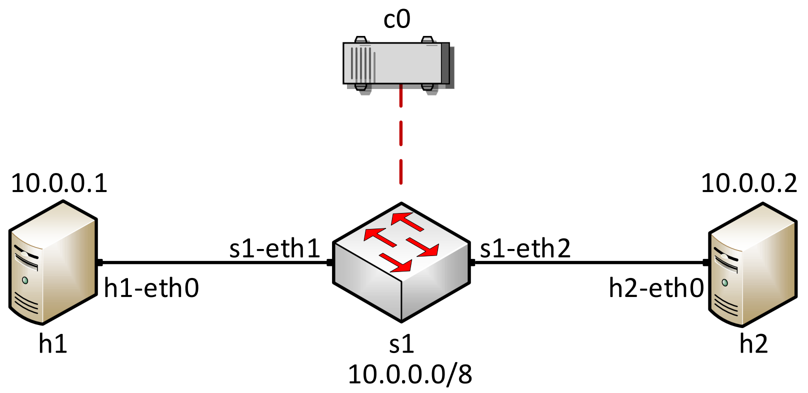


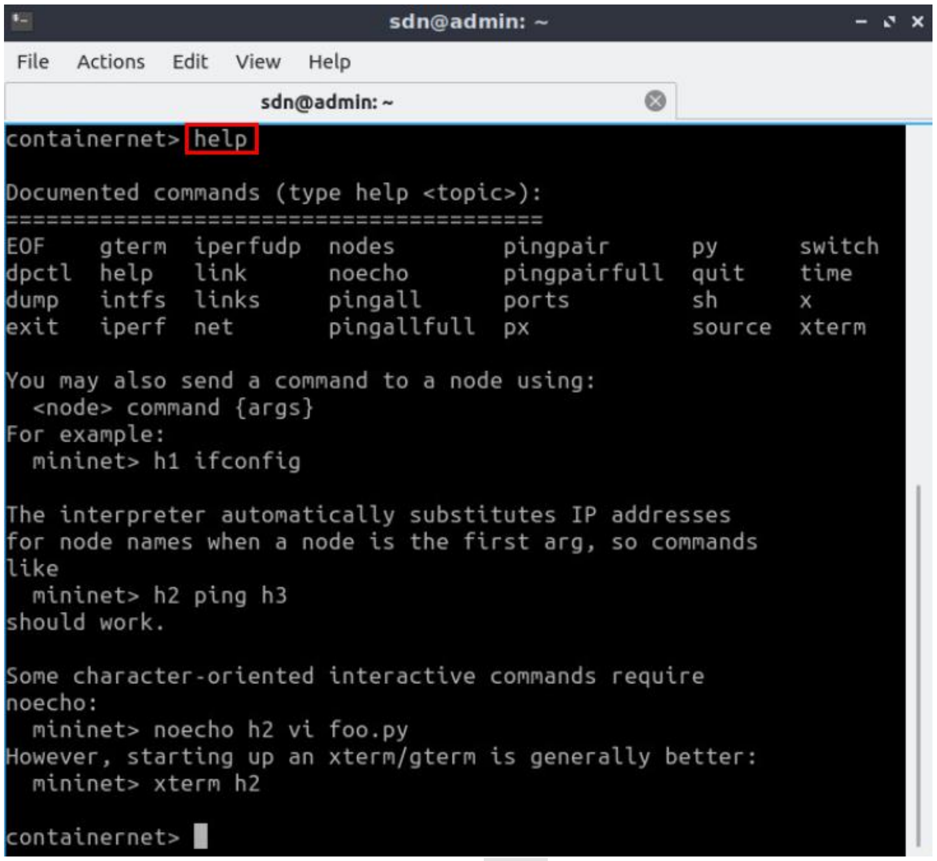
Figure 5. Mininet's default minimal topology.

When issuing the `sudo mn` command, Mininet initializes the topology and launches the containernet 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
```

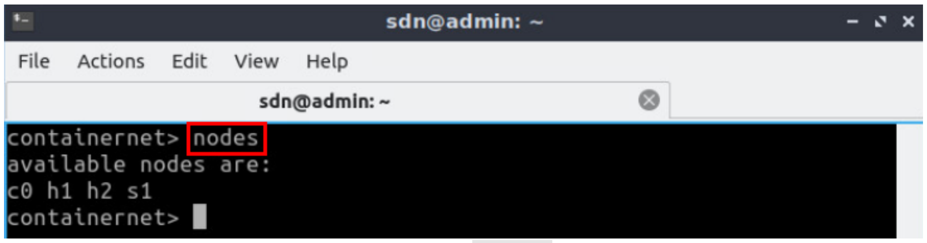


```
containernet> help

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.

```
nodes
```



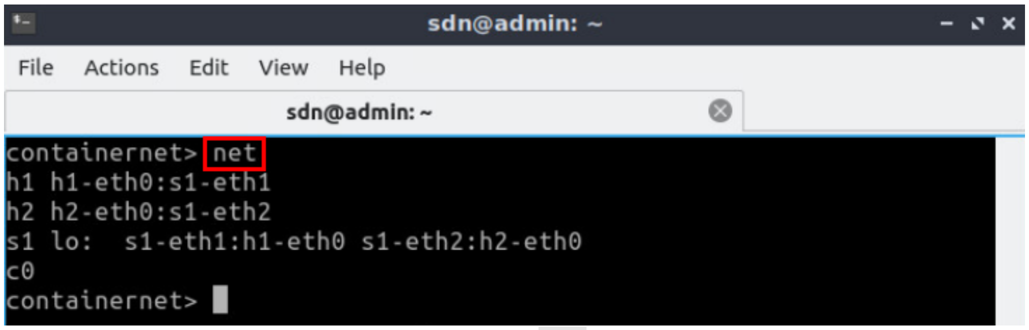
```
containernet> nodes
available nodes are:
c0 h1 h2 s1
containernet>
```

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.

```
net
```



```
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.

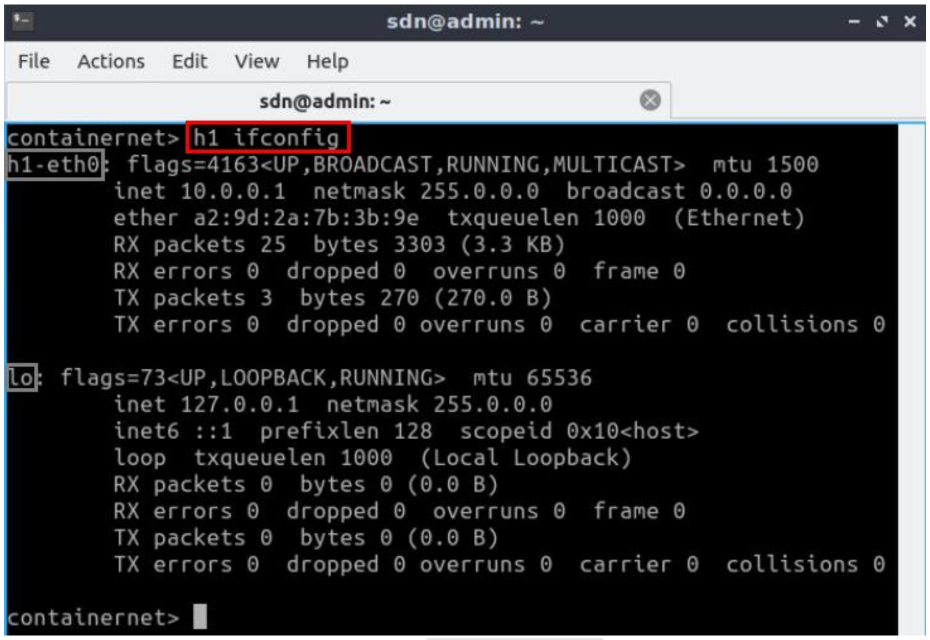
The output of this command shows that:

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 following command.

```
h1 ifconfig
```



```
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 a2:9d:2a:7b:3b:9e  txqueuelen 1000  (Ethernet)
        RX packets 25  bytes 3303 (3.3 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

containernet>
```

Figure 9. Output of `h1 ifconfig` command.

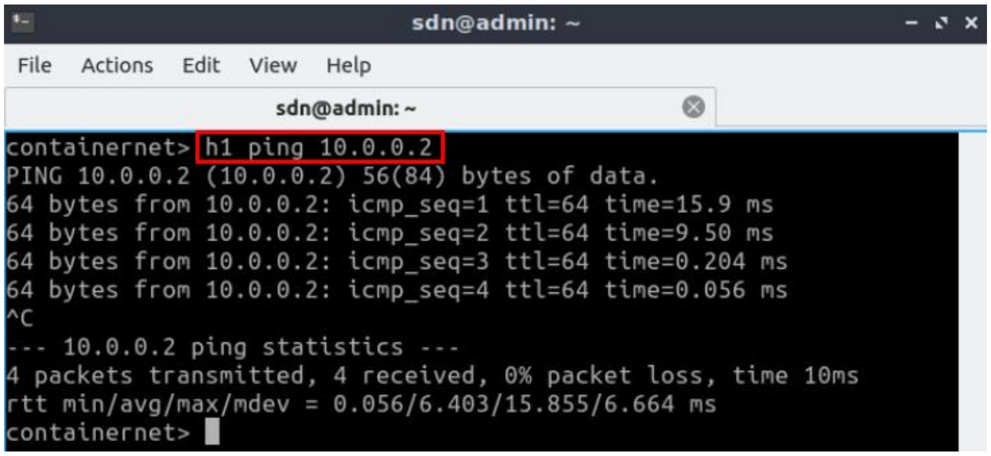
This command executes the `ifconfig` Linux command on host h1. The command shows host h1's interfaces. The display indicates that host h1 has an interface *h1-eth0* configured with IP address 10.0.0.1, and another interface lo, configured with IP address 127.0.0.1 (loopback interface).

## 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 or reply. 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.

```
h1 ping 10.0.0.2
```



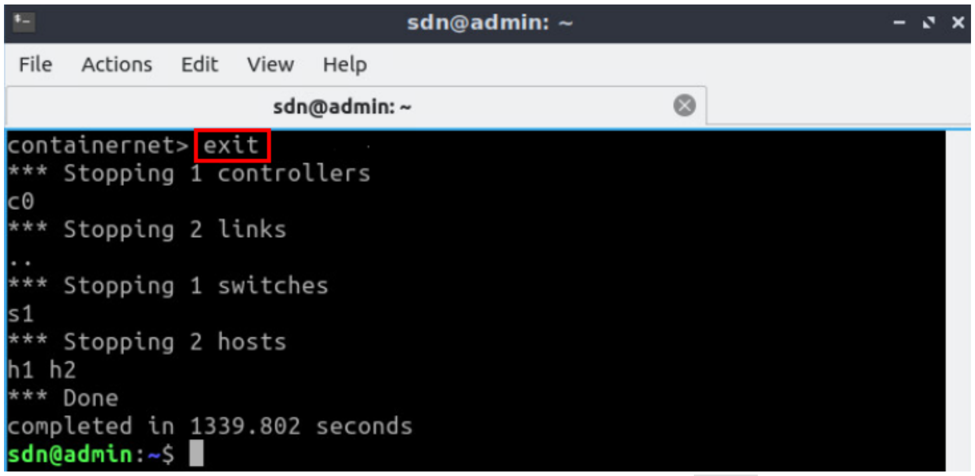
```
containernet> h1 ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=15.9 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=9.50 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.204 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.056 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 10ms
rtt min/avg/max/mdev = 0.056/6.403/15.855/6.664 ms
containernet>
```

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

To stop the test, press `Ctrl+c`. The figure above 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.

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

```
exit
```



```
containernet> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 1339.802 seconds
sdn@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[5] to deploy the topology illustrated below. MiniEdit is a simple GUI network editor for Mininet.

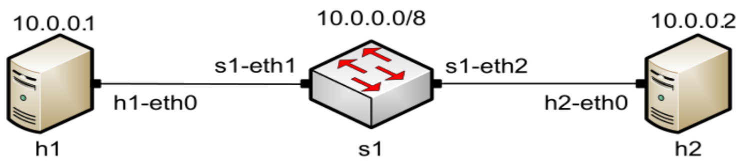


Figure 12. Lab topology.

### 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`.

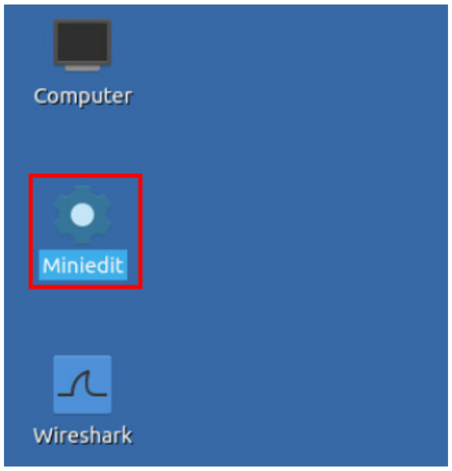


Figure 13. MiniEdit Desktop shortcut.

MiniEdit will start, as illustrated below.

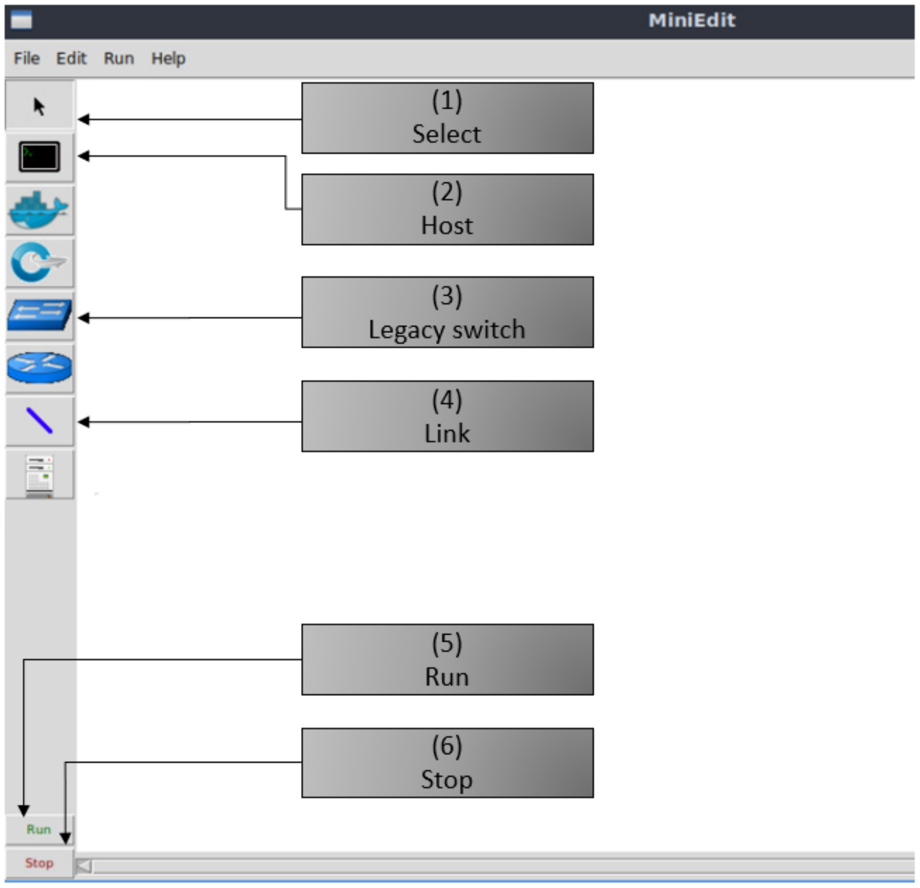


Figure 14. MiniEdit Graphical User Interface (GUI).

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.

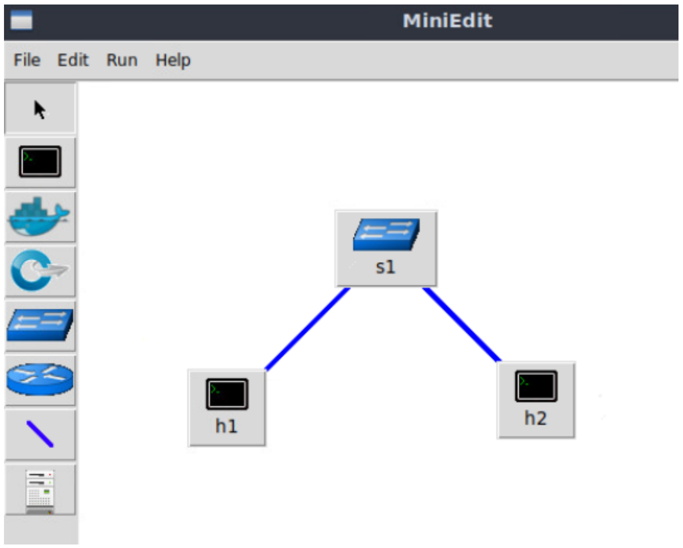
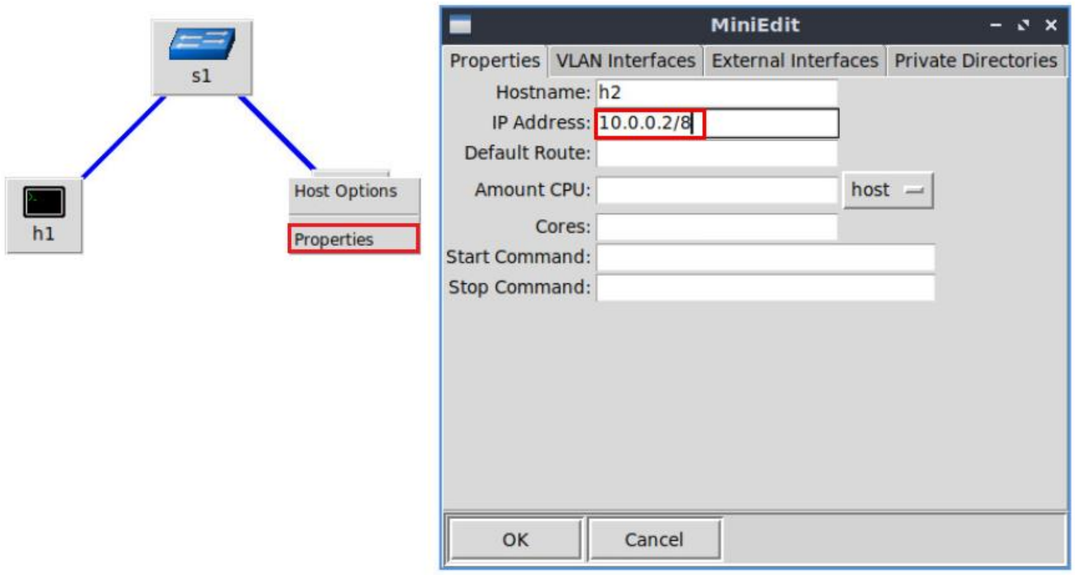


Figure 15. MiniEdit's topology.

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. Click *OK* for the settings to be applied.



Hostname: h2
IP Address: 10.0.0.2/8
Default Route:
Amount CPU: host
Cores:
Start Command:
Stop Command:

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.

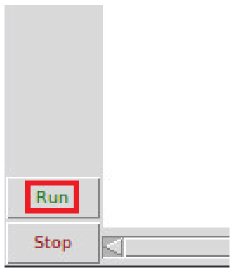


Figure 17. Starting the emulation.

**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.

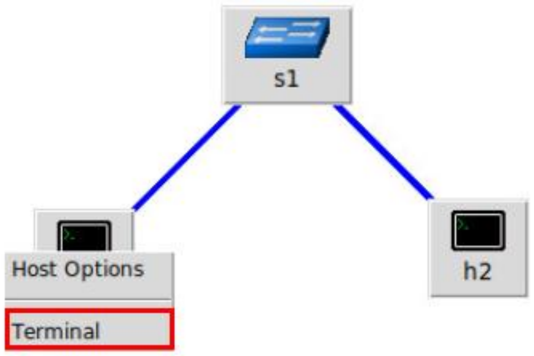
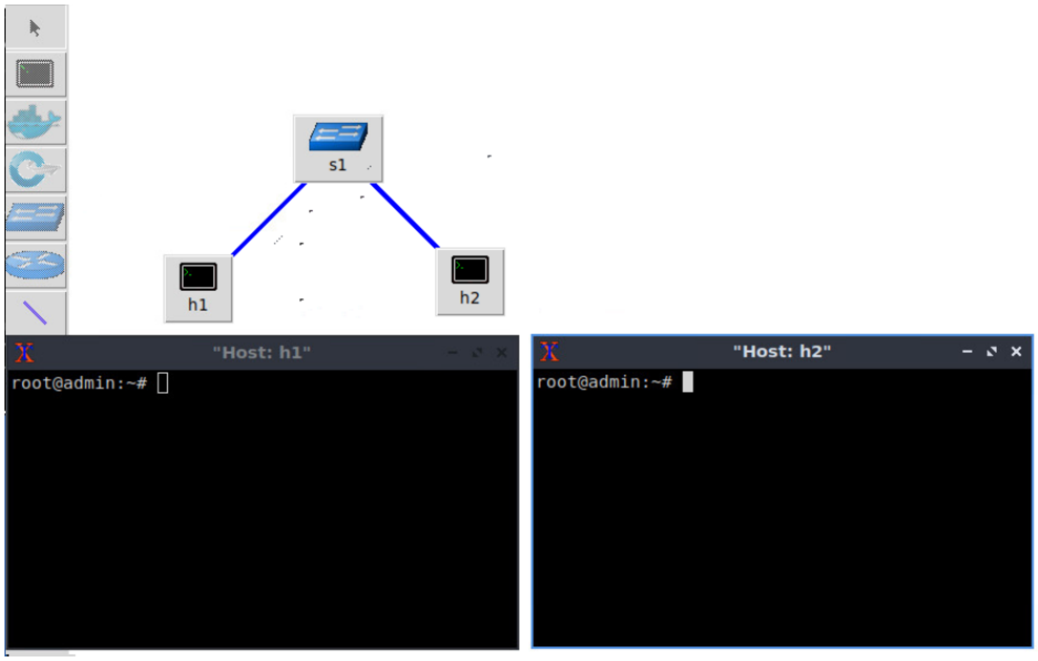


Figure 18. Opening a terminal on host h1.

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



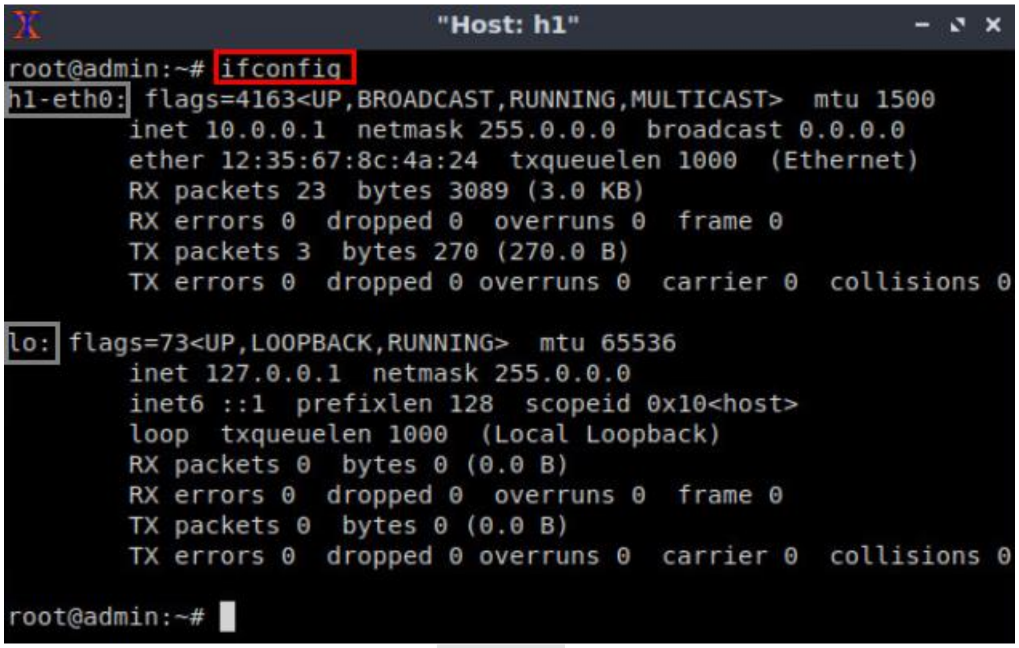
```
root@admin:~#
```

```
root@admin:~#
```

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
```



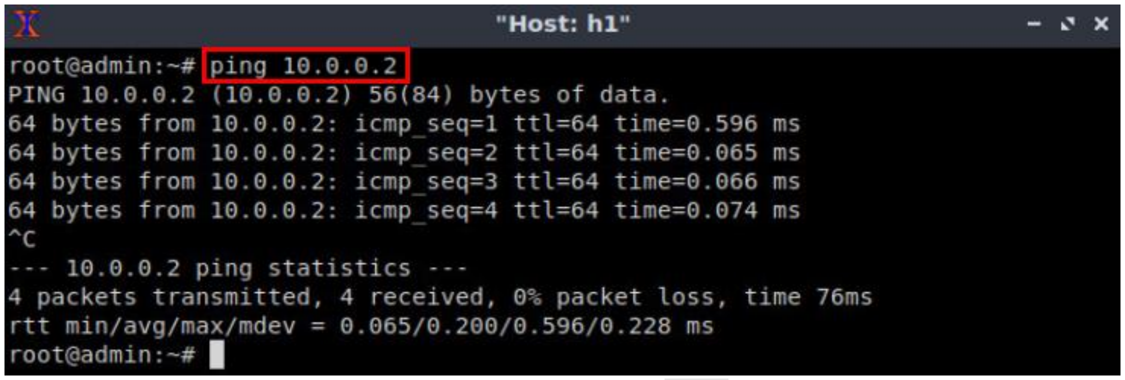
```
root@admin:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 0.0.0.0
        ether 12:35:67:8c:4a:24  txqueuelen 1000  (Ethernet)
        RX packets 23  bytes 3089 (3.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 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:~#
```

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.

```
ping 10.0.0.2
```



```
root@admin:~# 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=0.596 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.066 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.074 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 76ms
rtt min/avg/max/mdev = 0.065/0.200/0.596/0.228 ms
root@admin:~#
```

Figure 21. Connectivity test using `ping` command.

To stop the test, press `Ctrl+c`. The figure above 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.

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

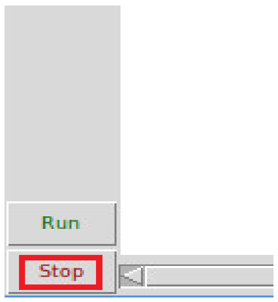
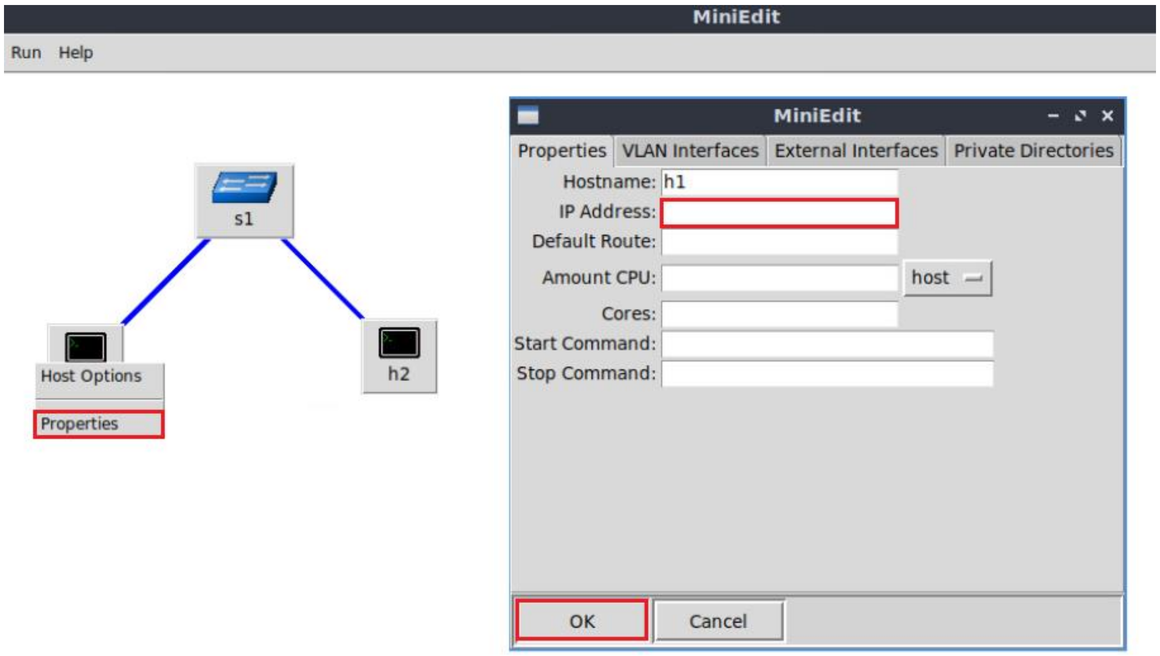


Figure 22. Stopping the emulation.

## 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.

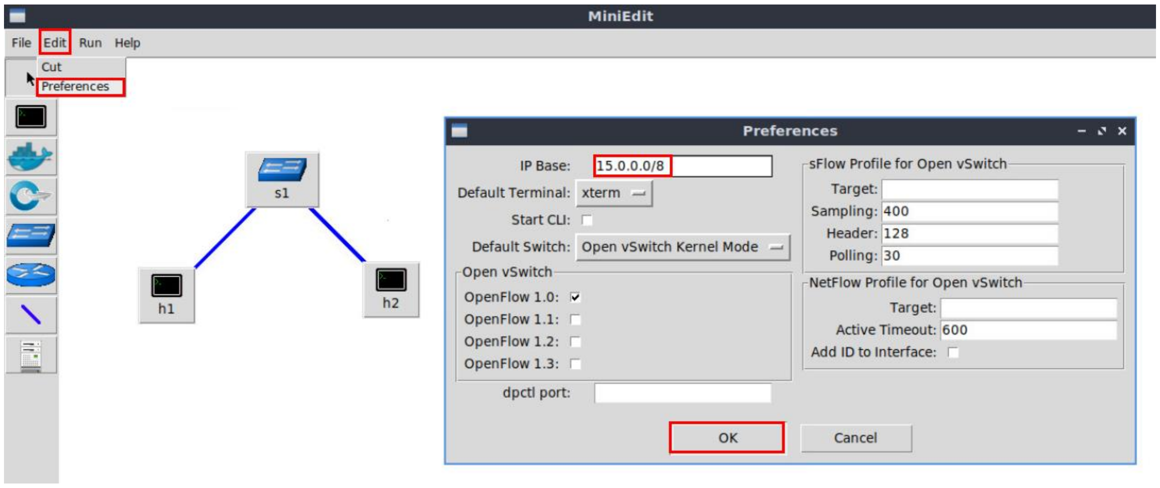
**Step 1.** Remove the manually assigned IP address from host h1. Hold right-click on host h1, *Properties*. Delete the IP address, leaving it unassigned, and press the *OK* button as shown below. Repeat the procedure on host h2.



Hostname: h1
IP Address:
Default Route:
Amount CPU: host
Cores:
Start Command:
Stop Command:

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.



IP Base: 15.0.0.0/8
Default Terminal: xterm
Start CLI: ☐
Default Switch: Open vSwitch Kernel Mode
Open vSwitch
OpenFlow 1.0: ✓
OpenFlow 1.1: ☐
OpenFlow 1.2: ☐
OpenFlow 1.3: ☐
dpctl port:
sFlow Profile for Open vSwitch
Target:
Sampling: 400
Header: 128
Polling: 30
NetFlow Profile for Open vSwitch
Target:
Active Timeout: 600
Add ID to Interface: ☐

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.

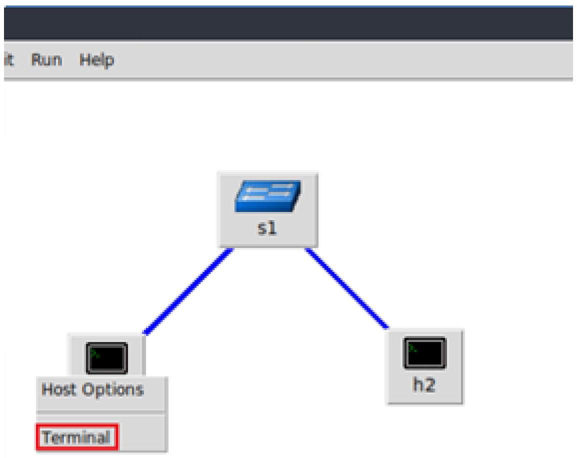
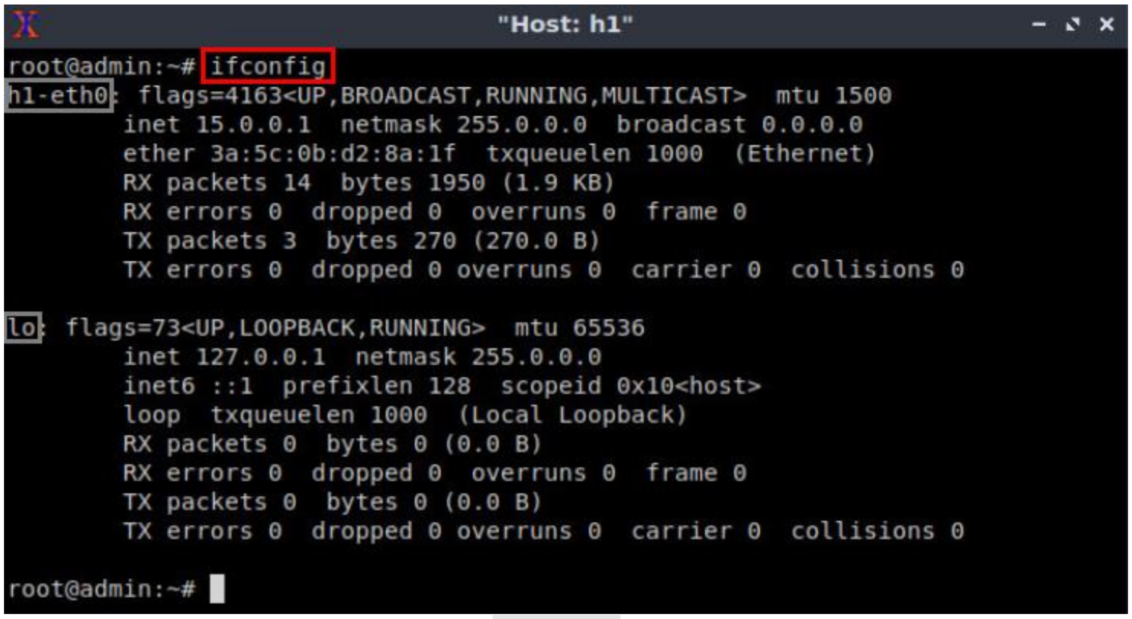


Figure 25. Opening a terminal on host h1.

**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
```



```
root@admin:~# 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 3a:5c:0b:d2:8a:1f  txqueuelen 1000  (Ethernet)
        RX packets 14  bytes 1950 (1.9 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:~# 
```

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.

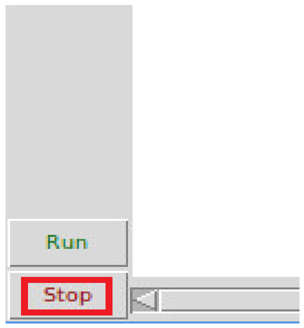
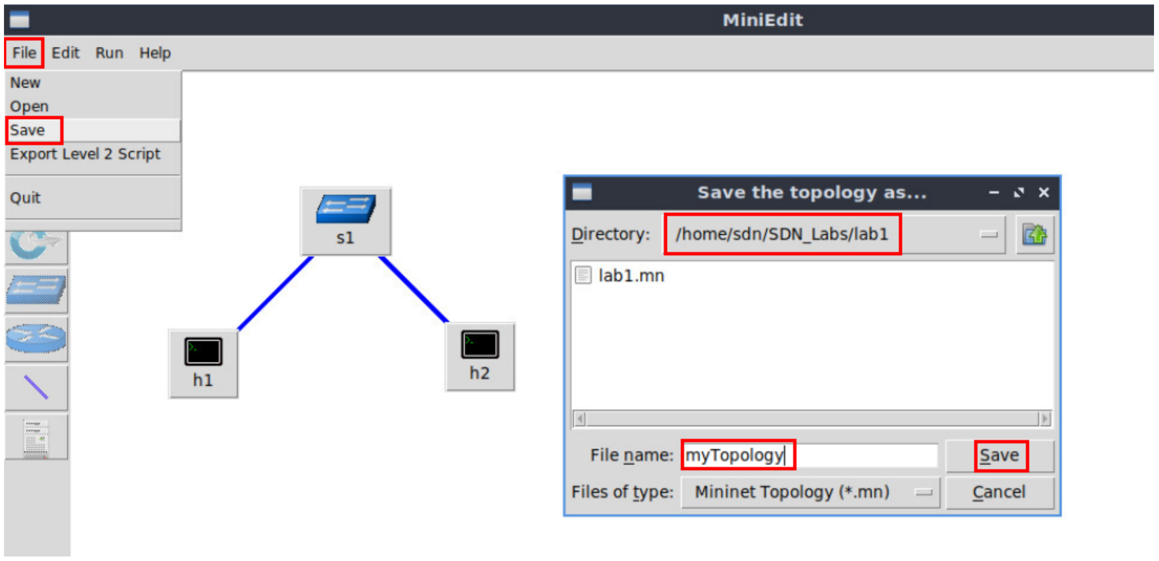


Figure 27. Stopping the emulation.

## 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.



Directory: /home/sdn/SDN_Labs/lab1
File name: myTopology
Files of type: Mininet Topology (*.mn)

Figure 28. Saving the topology.

**Step 2.** Load the topology by clicking on *File* then *Open*. Search for the topology file called *lab1.mn* and click on *Open*. A new topology will be loaded to MiniEdit.

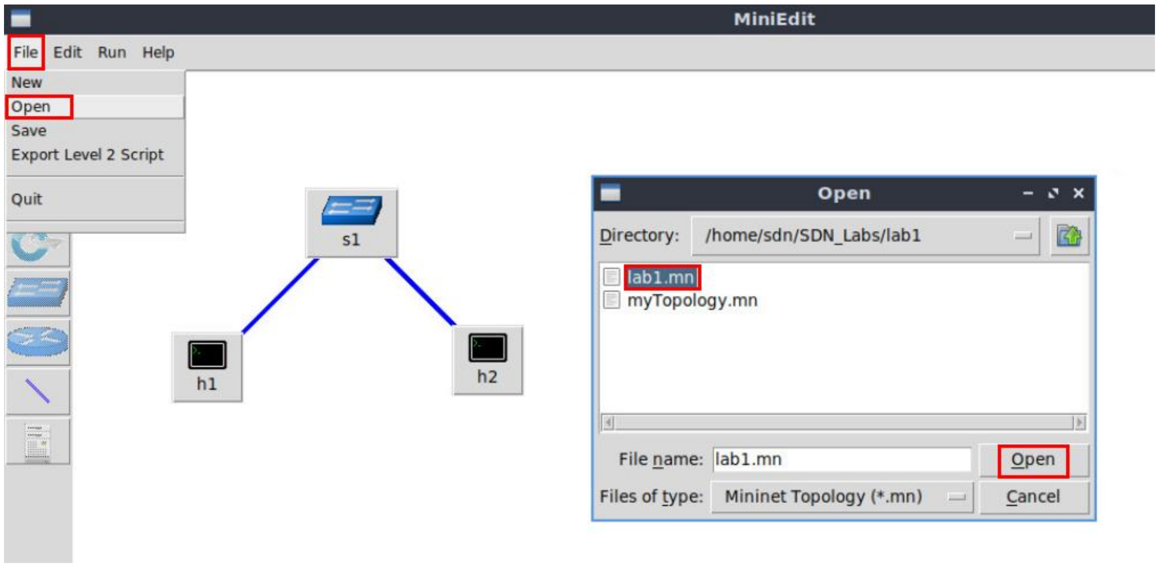


Figure 29. Opening a topology.

## 4 Configure router r1

In the previous step, 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.

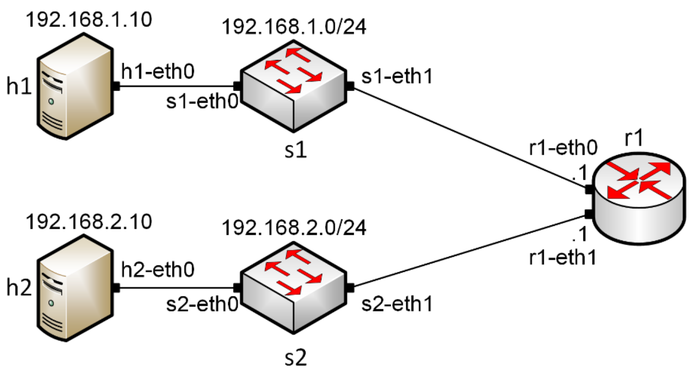


Figure 30. Topology.

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.

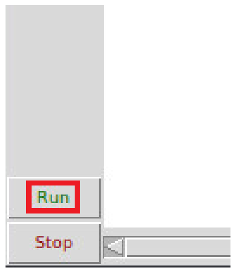


Figure 31. Starting the emulation.

## 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.

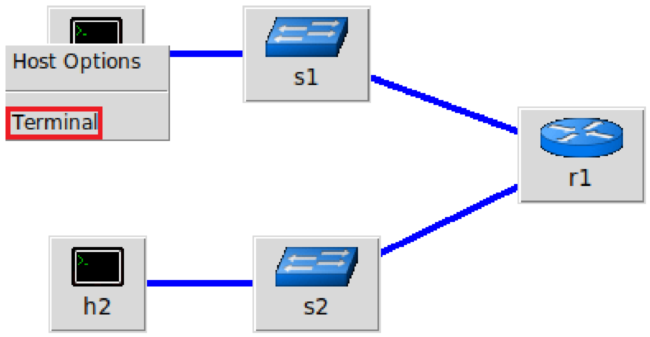
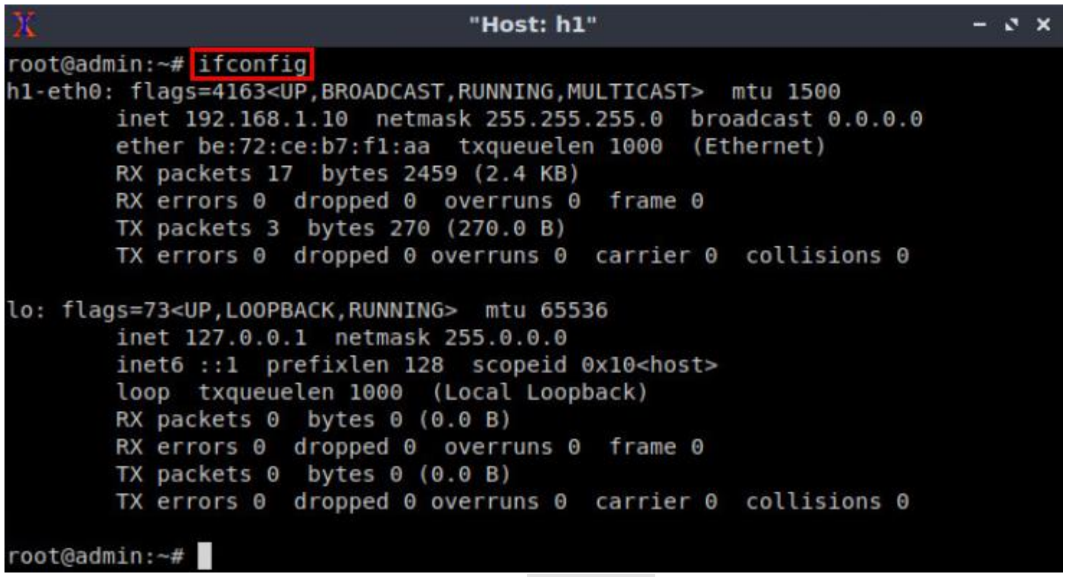


Figure 32. Opening a terminal on host h1.

**Step 2.** On 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
```

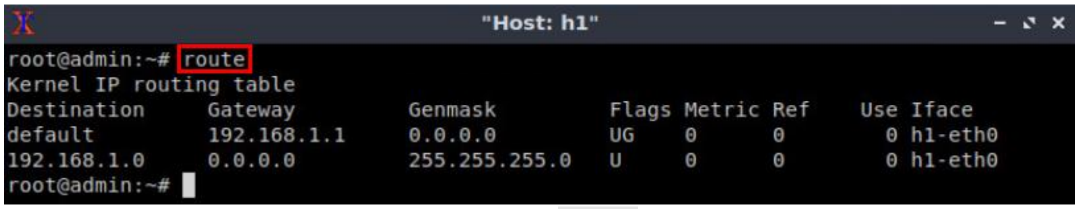


```
"Host: h1"
root@admin:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.10  netmask 255.255.255.0  broadcast 0.0.0.0
        ether be:72:ce:b7:f1:aa  txqueuelen 1000  (Ethernet)
        RX packets 17  bytes 2459 (2.4 KB)
        RX errors 0  dropped 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:~# 
```

Figure 33. Output of `ifconfig` command.

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

```
route
```



```
"Host: h1"
root@admin:~# 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:~# 
```

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 on 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*.

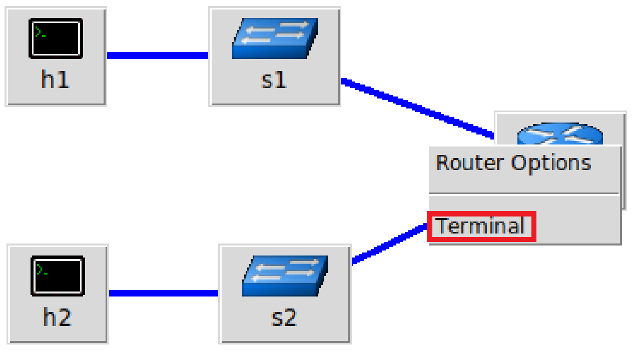
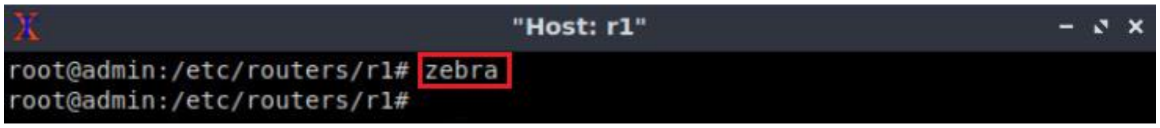


Figure 35. Opening a terminal on router r1.

**Step 2**. In this step, you will start the zebra daemon, 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. To start zebra, type the following command.

```
zebra
```

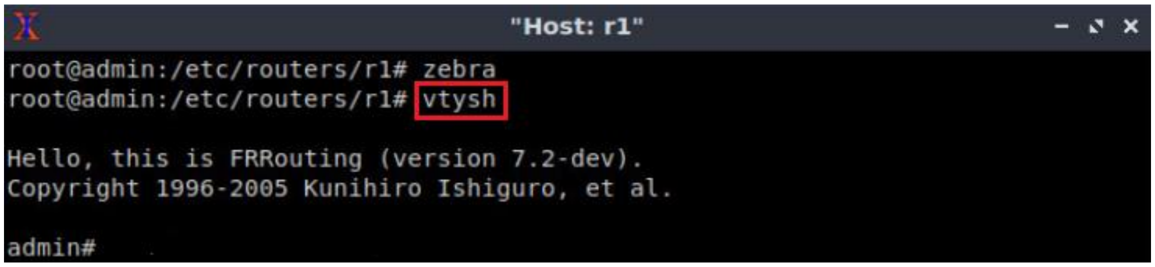


```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1#
```

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
```

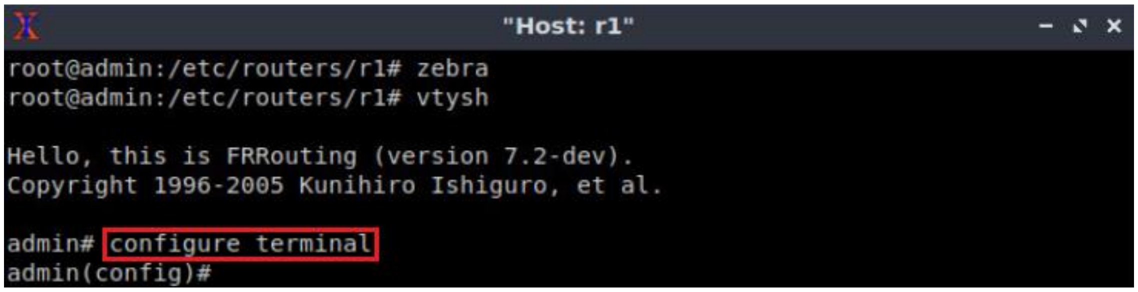


```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

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
```

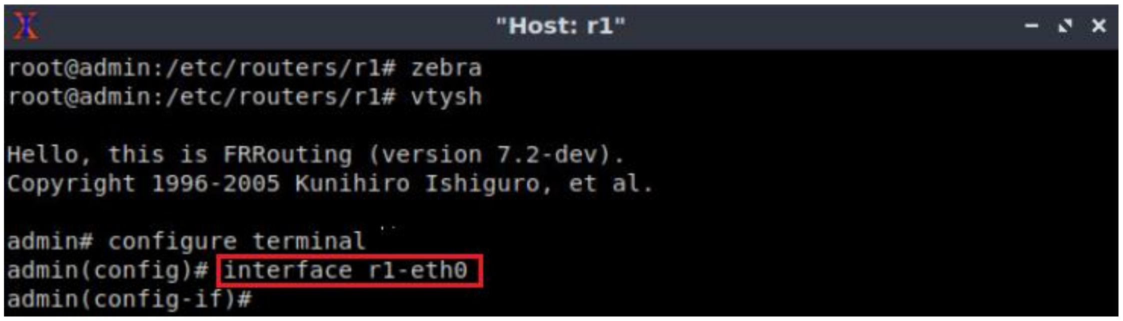


```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

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
```

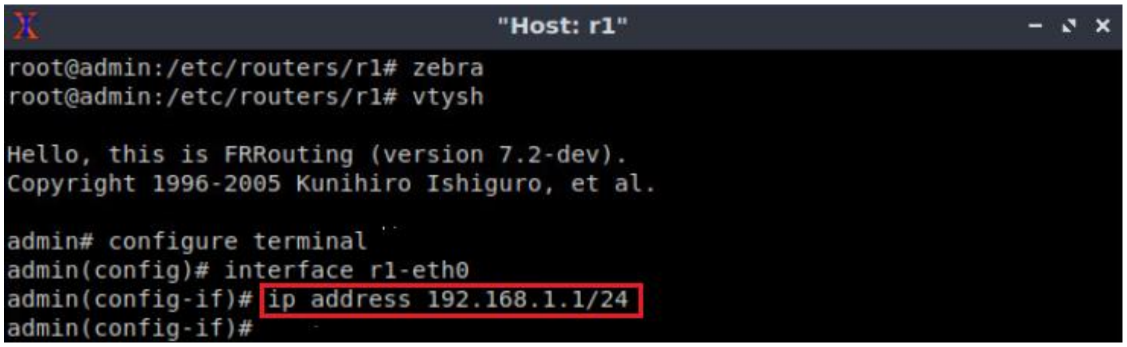


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

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
```

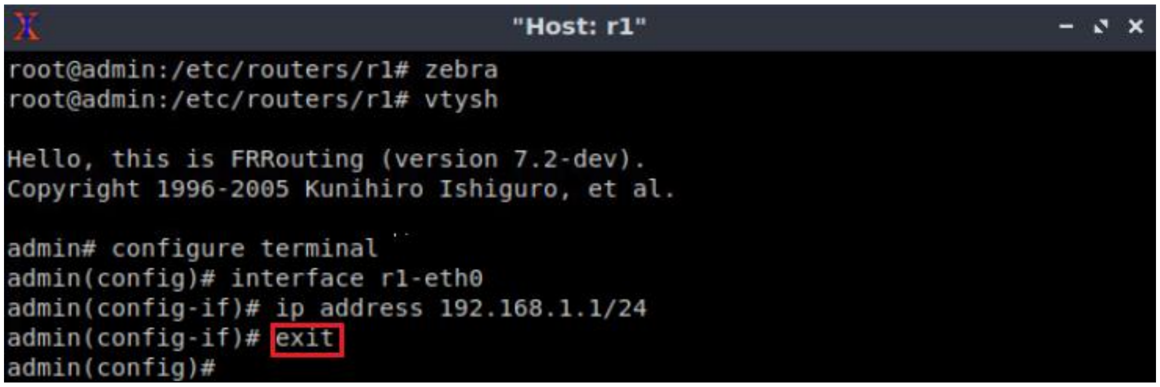


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

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
```

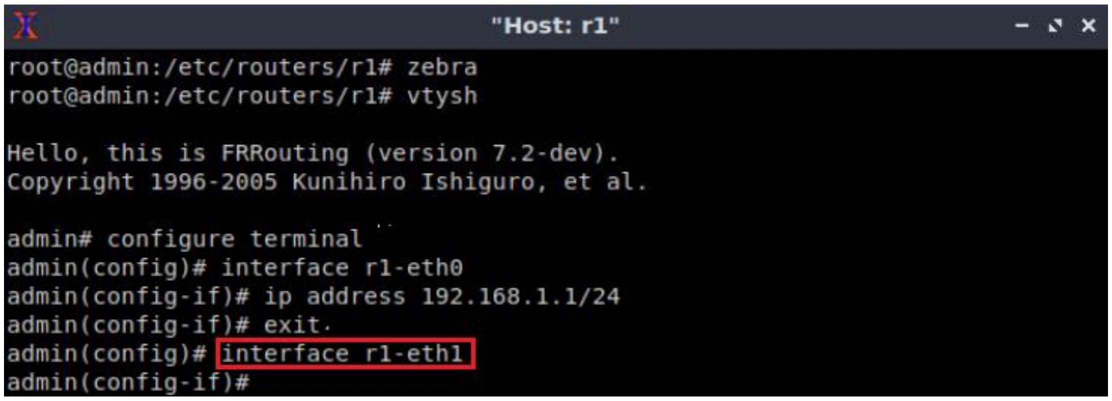


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

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
```

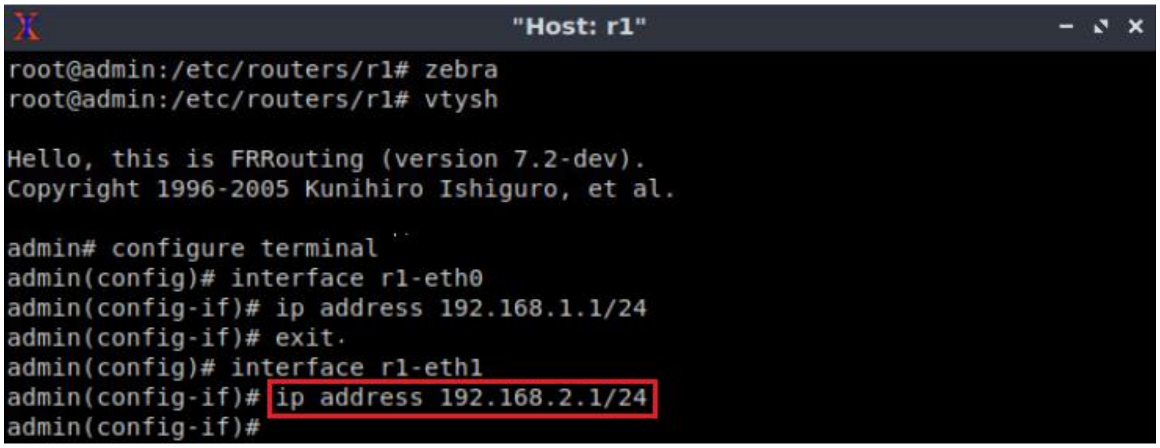


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

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
```

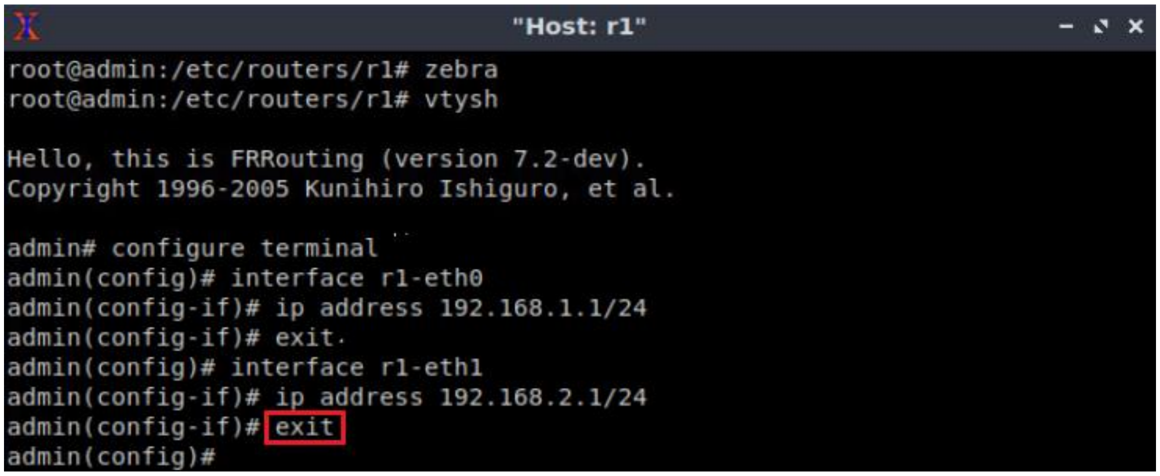


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

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
```



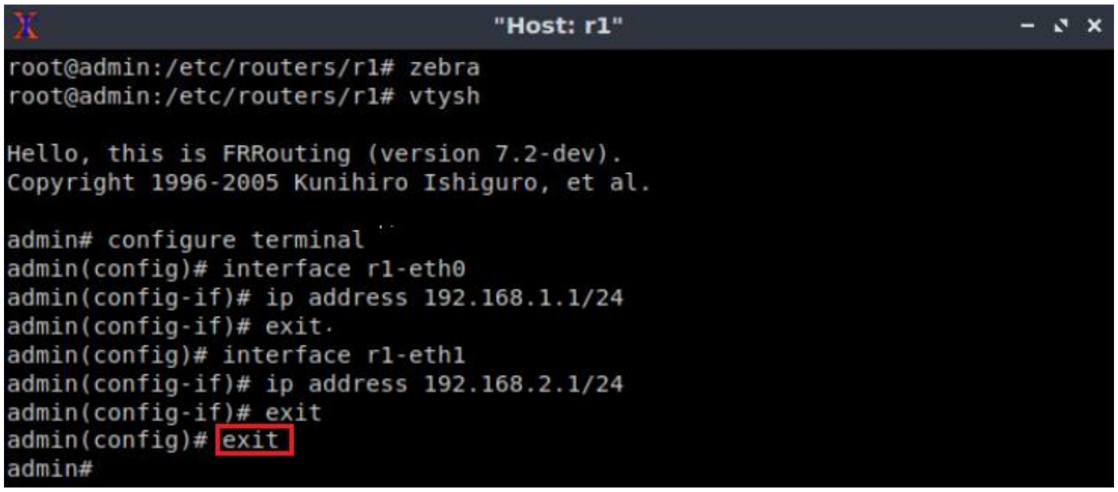
```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

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
```

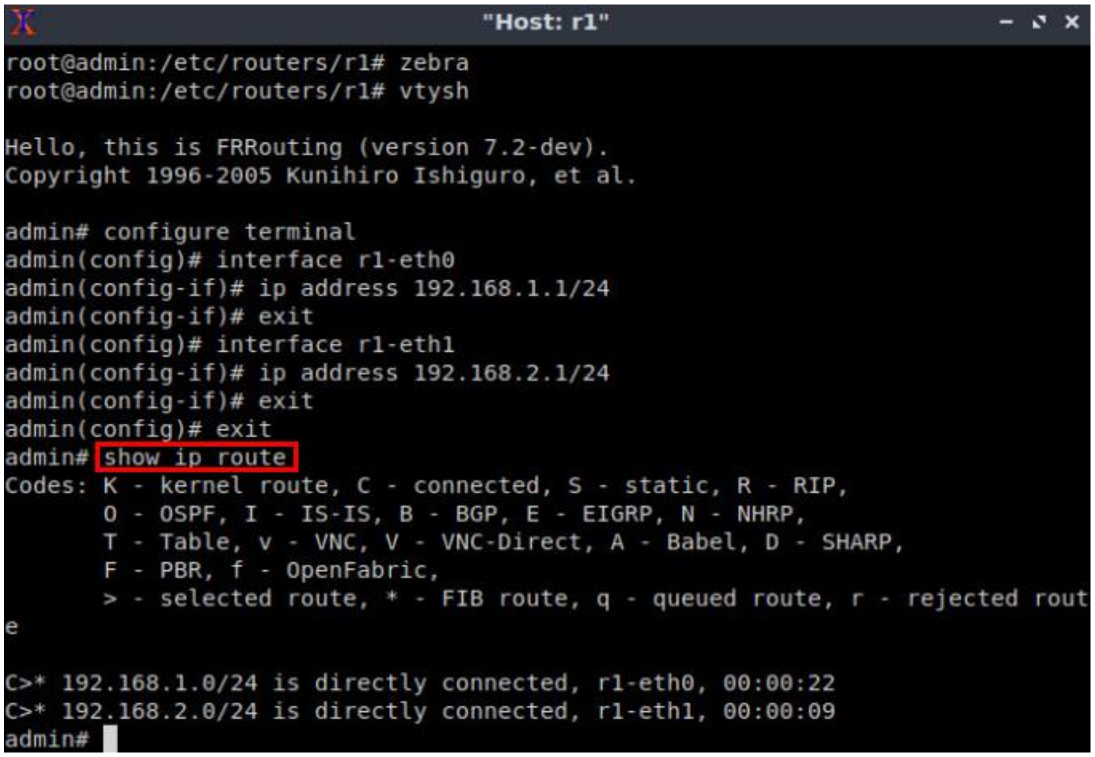


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

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
```



```
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# 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 rout
e

C>* 192.168.1.0/24 is directly connected, r1-eth0, 00:00:22
C>* 192.168.2.0/24 is directly connected, r1-eth1, 00:00:09
admin#
```

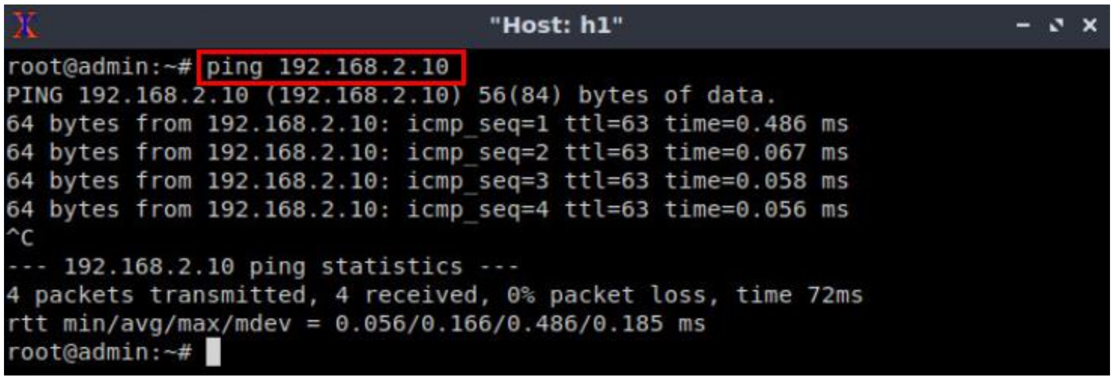
Figure 46. Displaying routing information of router r1.

## 4.4 Test connectivity between end-hosts

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

**Step 1.** On 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
```



```
root@admin:~# 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=0.486 ms
64 bytes from 192.168.2.10: icmp_seq=2 ttl=63 time=0.067 ms
64 bytes from 192.168.2.10: icmp_seq=3 ttl=63 time=0.058 ms
64 bytes from 192.168.2.10: icmp_seq=4 ttl=63 time=0.056 ms
^C
--- 192.168.2.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 72ms
rtt min/avg/max/mdev = 0.056/0.166/0.486/0.185 ms
root@admin:~# 
```

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

This concludes Lab 1. 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. J. Esch, *"Prolog to, software-defined networking: a comprehensive survey,"* Proceedings of the IEEE, vol. 103, no. 1, pp. 10–13, 2015.
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.

UNIVERSITY OF
SOUTH CAROLINA

# SOFTWARE DEFINED NETWORKING

# Lab 2: Legacy Networks: BGP Example as a Distributed System and Autonomous Forwarding Decisions

**Document Version: 03-30-2021**

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

## Contents

## Overview

This lab is an introduction to legacy networks using Free Range Routing (FRR), which is a routing software suite that provides Transmission Control Protocol (TCP)/Internet Protocol (IP) based routing services with routing protocols support. In this lab, you will understand the main difference between legacy and Software Defined Networking (SDN) networks. Furthermore, you will explore FRR architecture, and load its basic configuration. Furthermore, this lab emulates a simple legacy network that runs Border Gateway Protocol (BGP) between two Autonomous Systems (ASes).

## Objectives

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

1. Understand the difference between legacy and SDN networks.
2. Understand the architecture of FRR.
3. Navigate through FRR terminal.
4. Explain the concept of BGP.
5. Configure and verify BGP between two ASes.
6. Perform a connectivity test between end hosts.

## Lab settings

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

Table 1**.** Credentials to access the Client 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: Configure BGP routing protocol.
4. Section 4: Verify connections.

## 1 Introduction

## 1.1 Traditional switch architecture

In a traditional switch architecture, the switching functionalities are segregated into three separate categories, also called layers/planes. These layers can communicate horizontally, i.e., communicate with the same layer in a different switch. Additionally, the layers can communicate vertically, i.e., from one layer to another within the same switch[1].

Consider Figure 1. The vast majority of packets handled by the switch are only managed by the data plane. The latter is composed of ports used to receive and transmit the packets, as well as a forwarding table which instructs the switch on how to deal with incoming packets. The data plane is responsible for packet buffering, packet scheduling, header modification and forwarding[11].

Some packets cannot be processed by the data plane directly, for example, their information is not yet inserted in the forwarding table. Such packets are forwarded to the control plane which lies on top of the data plane. The control plane involves in many activities, mainly, to maintain the forwarding table of the data plane. Essentially, the control plane is responsible for processing different control protocols that may affect the forwarding table[11].

The management plane lies on top of the control plane and it is used by network administrators to configure and monitor the switch. Thus, allowing them to extract information or modify data in the underlying planes (control and data planes) as appropriate[11].

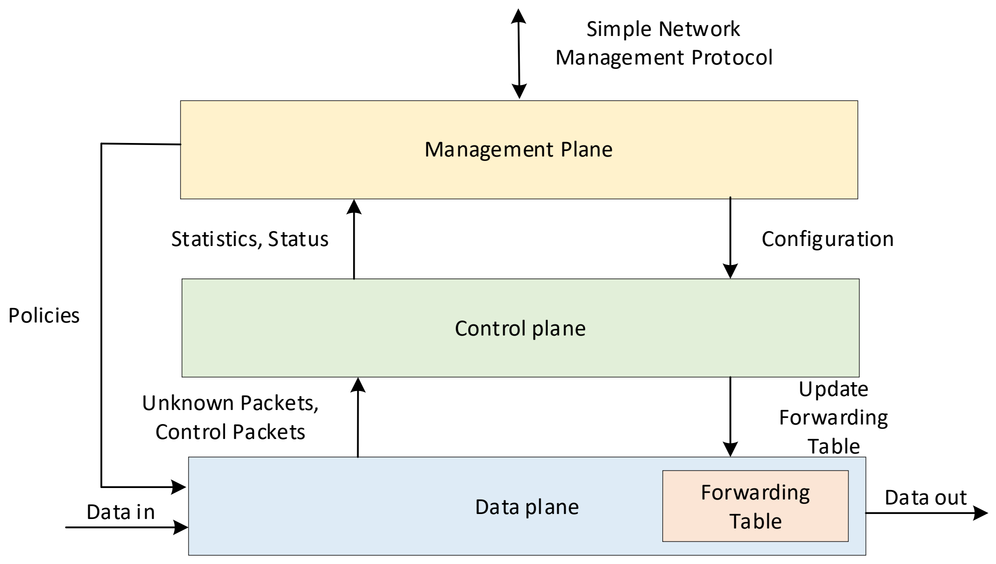


Figure 1. Roles of the control, data and, management planes[3].

## 1.2 Legacy and SDN networks

In a legacy network, the data, control, and management layers are aggregated into the same device, usually referred to as a router. When a packet arrives to a router, it checks to see if the packet should be forwarded out one of its interfaces or if the packet needs further processing. The decision is made based on the routing table of the router, which consists of multiple entries, each maps a network/IP prefix to a next hop. The routing table is built primarily through the use routing protocols. They specify how routers communicate with each other to distribute information that enables them to select routes between any two nodes on a computer network. Routing protocols include Routing Information Protocol[2] (RIP), Open Shortest Path First[3] (OSPF), BGP[4] and Intermediate System to Intermediate System (IS-IS)[5].

Consider Figure 2. A legacy network consists of several connected devices. Each device runs a local algorithm in the control plane and inserts forwarding rules in the routing table of the data plane.

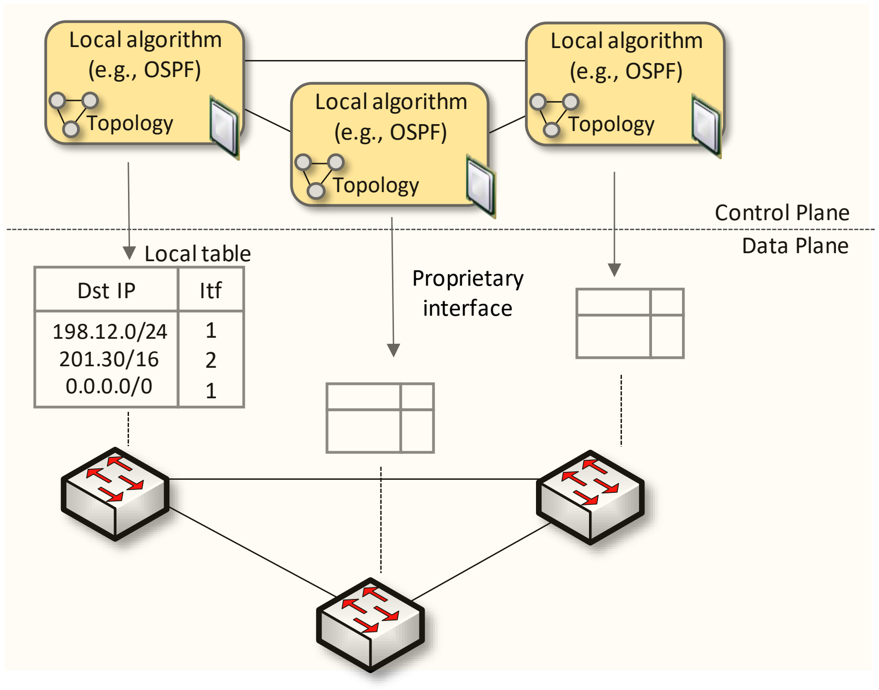


Figure 2. The control plane and the data plane are coupled in the same legacy device.

Routing protocols were essential to respond to rapidly changing network conditions. However, these conditions no longer exist in modern data centers. Typically, legacy routing protocols work as a distributed system transmitting the connection status information over the link and, each router performs the routing computation. A drawback of this scheme is that the routing information relies on flooding the link state to update information among routers. Therefore, this incurs in longer convergence time where the link delay affects the convergence time[8].

SDN is a new paradigm that solves the aforementioned problem by creating a centralized approach, rather than a distributed one. The main concept of SDN is to separate the control plane from the data plane in order to maximize the efficiency of the data plane devices. Moving the control software off the device into a centralized server makes it capable of seeing the entire network and making decisions that are optimal given a complete understanding of the situation[11].

Consider Figure 3. The control plane is decoupled from the data plane. The former is moved into a centrally located computer resource and it controls data plane devices, mainly OpenFlow switches, by pushing rules into their tables[11].

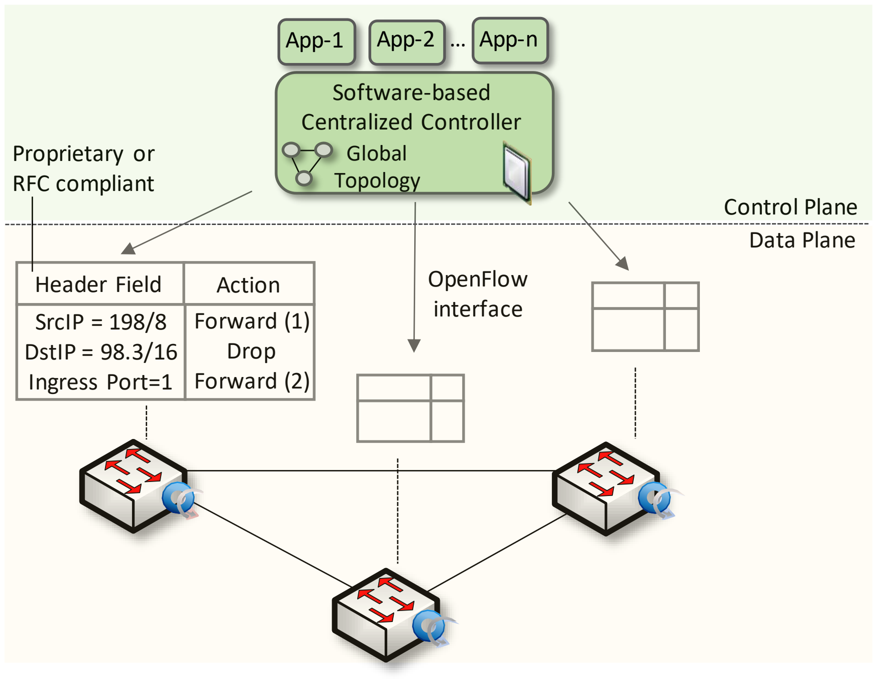


Figure 3. The control plane is embedded in a centralized server and it is decoupled from data plane devices.

This lab solely focuses on understanding how legacy networks work. You will configure BGP on legacy routers and inspect the inserted rules on each router’s forwarding table. In order to do the configuration in an emulated environment, FRR will be used, which is an open-source software that allows to configure the routers with a list of supported routing protocols.

## 1.3 Introduction to FRR

Implementing IP routing usually involves buying expensive and vertically integrated equipment from specific companies. This approach has limitations such as the cost of the hardware, closed source software, and the training required to operate and configure the devices. Networking professionals, operators, and researchers sometimes are limited by the capabilities of such routing products. Moreover, combining routing functionalities with existing open-source software packages is usually constrained by the number of separate devices that can be deployed.

For example, operators could be interested in collecting some information about the behavior of routing devices, process them, and make them available. Therefore, in order to achieve such capabilities, additional storage and scripting capacities are required. Such resources are not available in existing routing products. On the other hand, researchers may be interested in developing routing protocols by extending an existing one without writing a complete implementation from scratch.

FRR suite[1] is a package of Unix/Linux software that implements common network routing protocols, such as RIP[2], OSPF[3], BGP[4] and IS-IS[5]. The package also includes a routing information management process, to act as an intermediary between the various routing protocols and the active routes installed with the kernel. A library provides support for configuration and an interactive command-line interface. The routing protocols supported by FRR, can be extended to enable experimentation, logging, or custom processing. In addition, libraries and kernel daemon provide a framework to facilitate the development of new routing protocol daemons. A wide range of functionalities can be attained by combining other software packages to allow the integration into a single device as well as enabling innovative solutions to networking problems.

## 1.4 FRR architecture

FRR takes a different approach compared to traditional routing software which, consists of a single process program that provides all the routing protocol functionalities. FRR is composed of a suite of daemons that work together to build a routing table. Each routing protocol is implemented in its own daemon. These daemons exchange information through another daemon called zebra, which is responsible for encompassing routing decisions and managing the data plane.

Since all the protocols are running independently, this architecture provides high resiliency, that means that an error, crash or exploit in one protocol daemon will generally not affect the other protocols. It is also flexible and extensible since the modularity makes it easy to implement new protocols and append them to the suite[1]. Additionally, each daemon implements a plugin system allowing new functionality to be loaded at runtime.

Figure 4 illustrates the FRR architecture. It consists of a set of processes communicating via Inter-process Communication (IPC) protocol. This protocol refers to the mechanism provided by an operating system (OS) to manage shared data between different processes. Network routing protocols such as BGP, OSPF and IS-IS are implemented in processes such as *bgpd*, *ripd*, *ospfd*, *ldpd, etc*. These processes are daemons that implement routing protocols e.g., the BGP daemon is implemented by the *bgpd* process, the RIP daemon is implemented by the *ripd* process and so on. Another daemon, called *zebra*, acts as an intermediary between the kernel's forwarding plane and the routing protocol processes. Additionally, an interactive command-line tool called *vtysh* allows these processes to be monitored and configured. The *vtysh* command-line tool communicates with other processes via a simple string passing protocol, where the strings are essentially identical to the commands entered.

The *zebra* process is a fundamental part of FRR architecture. Its purpose is to maintain a backup of packet forwarding states, such as the network interfaces and the table of currently active routes. The currently active routes are also referred to as the Forwarding Information Base (FIB) [6]. Usually, the kernel manages packet forwarding therefore, the kernel maintains these. The *zebra* process also collects routing information from the routing protocol processes and stores these, together with its shadow copy of the FIB, in its own Routing Information Base (RIB)[6] whereas, static routes are also configured. The *zebra* process then is responsible for selecting the best route from all those available for a destination and updating the FIB[7]. Additionally, the information about the current best routes may be distributed to the protocol daemons. The *zebra* process maintains the routing daemons updated if any change occurs in the network interface state.

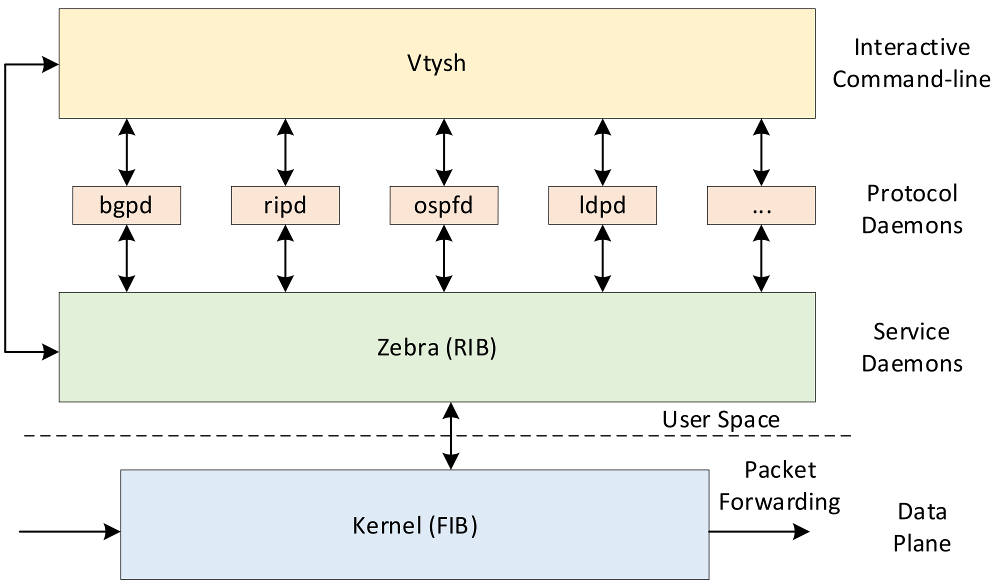


Figure 4. FRR architecture.

## 1.5 FRR and Mininet integration

Mininet is a network emulator which runs a collection of end-hosts, switches, routers and, links on a single Linux kernel[9]. Mininet provides network emulation, 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. Hence, the set of commands provided by FRR are inherited and can be run using Mininet's command-line interface. This feature allows you to run and configure FRR in the emulated routers. FRR is production-ready, but we are using it in an emulated environment.

## 1.6 Introduction to BGP

The Internet can be viewed as a collection of networks or ASes that are interconnected. An AS refers to a group of connected networks under the control of a single administrative entity or domain[8].

BGP is an exterior gateway protocol designed to exchange routing and reachability information among ASes on the Internet. BGP is relevant to network administrators of large organizations which connect to one or more Internet Service Providers (ISPs), as well as to ISPs who connect to other network providers. In terms of BGP, an AS is referred to as a routing domain, where all networked systems operate common routing protocols and are under the control of a single administration[8].

Two routers that establish a BGP connection are referred to as BGP peers or neighbors. BGP sessions run over TCP. If a BGP session is established between two neighbors in different ASes, the session is referred to as an External BGP (EBGP) session. If the session is established between two neighbors in the same AS, the session is referred to as Internal (IBGP)[1]. Figure 5 shows a network running the BGP protocol. Routers that exchange information within the same AS use Internal BGP (IBGP), while routers that exchange information between different ASes use EBGP.

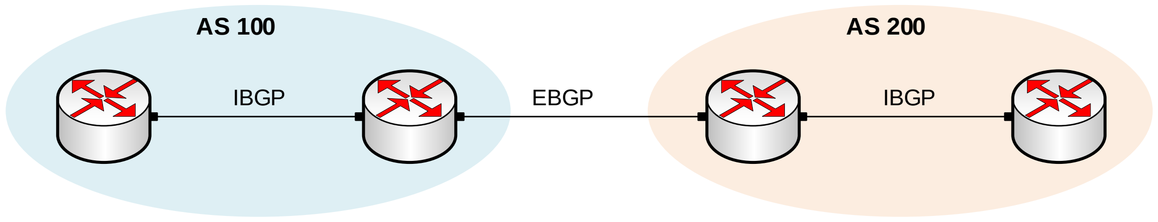


Figure 5. Routers that exchange information within the same AS use IBGP, while routers that exchange information between different ASes use EBGP.

## 2 Lab topology

Consider Figure 6. The topology consists of two networks, Network 1, and Network 2, each in an AS. Both networks have the following elements: a router to connect the networks together, a switch that defines a Local Area Network (LAN) and lastly, a host aimed to test end-to-end connectivity. The Autonomous System Numbers (ASNs) assigned to routers r1 and r2 are 100 and 200, respectively. Routers r1 and r2 exchange routing information via EBGP.

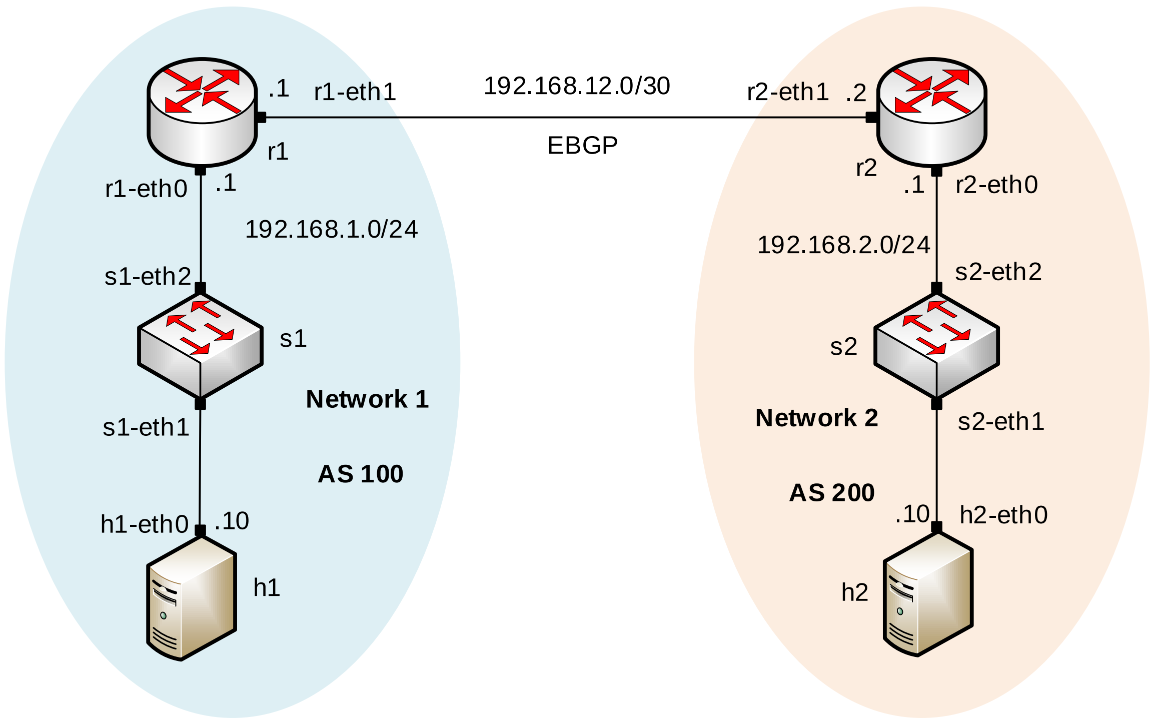


Figure 6. Lab topology.

## 2.1 Lab settings

Routers and hosts are already configured according to the IP addresses shown in Table 2.

Table 2. Topology information.

| Device | Interface | IP Address | Subnet | Default gateway |
|---|---|---|---|---|
| Router r1 | r1-eth0 | 192.168.1.1 | /24 | N/A |
| | r1-eth1 | 192.168.12.1 | /30 | N/A |
| Router r2 | r2-eth0 | 192.168.2.1 | /24 | N/A |
| | r2-eth1 | 192.168.12.2 | /30 | N/A |
| Host h1 | h1-eth0 | 192.168.1.10 | /24 | 192.168.1.1 |
| Host h2 | h2-eth0 | 192.168.2.10 | /24 | 192.168.2.1 |

## 2.2 Loading the topology

In this section, you will open MiniEdit[10] 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 (IP addresses, default gateway), save the topology and export a layer 2 model.

**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`.

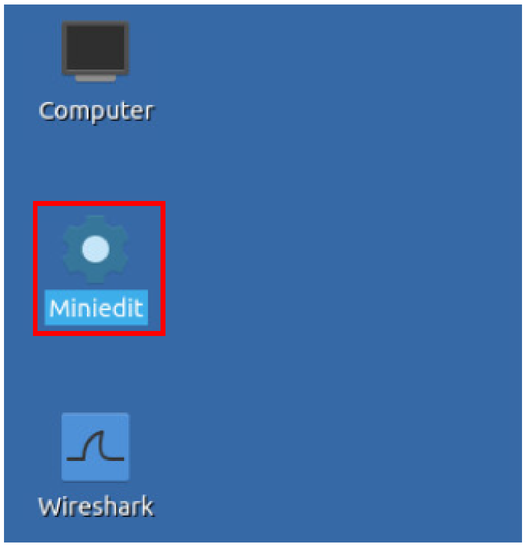


Figure 7. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then *open* to load the lab's topology. Open the *Lab2.mn* topology file stored in the default directory, */home/sdn/SDN_Labs /lab2* and click on *Open*.

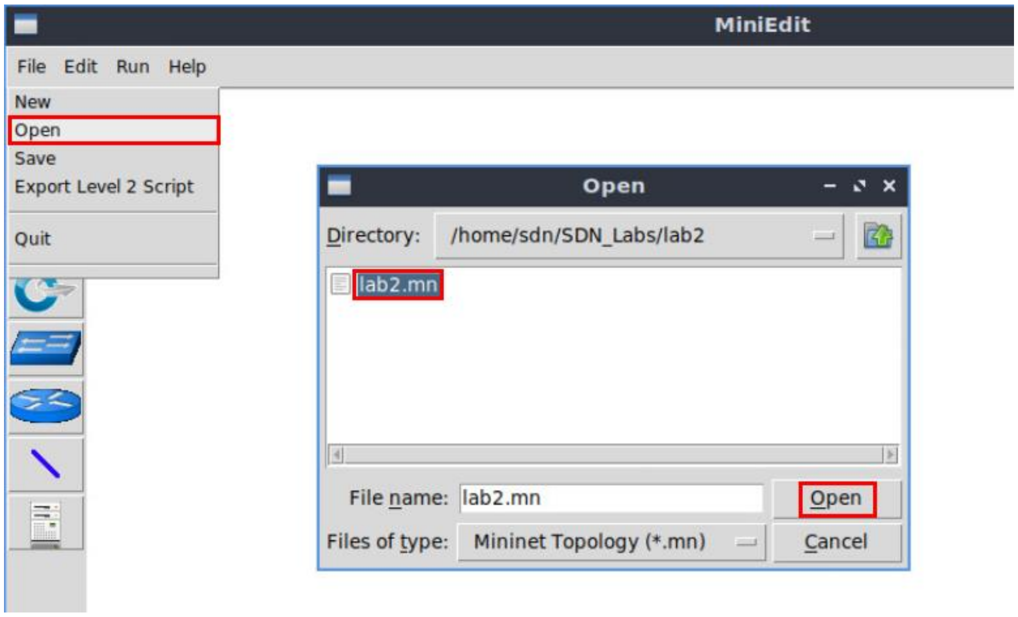


Figure 8. MiniEdit's open dialog.

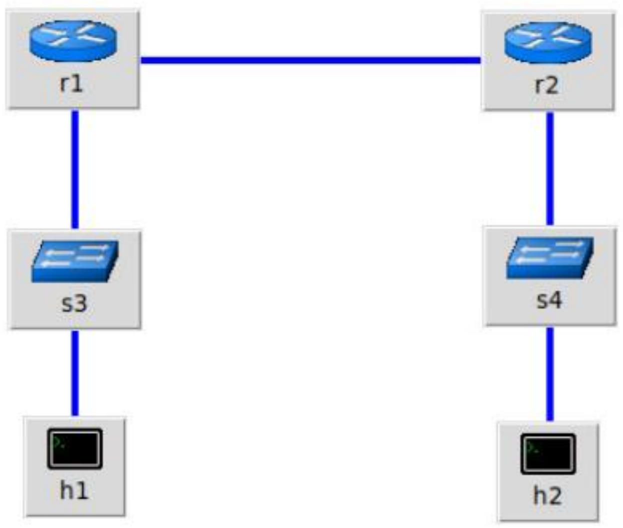


Figure 9. Mininet's topology.

## 2.3 Loading the configuration file

At this point the topology is loaded. However, the interfaces are not configured. In order to assign IP addresses to the interfaces of the devices, you will execute a script that loads the configuration to the routers.

**Step 1.** Click on the icon below to open the Linux terminal.

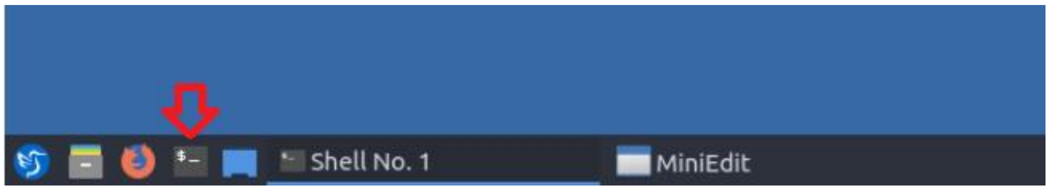
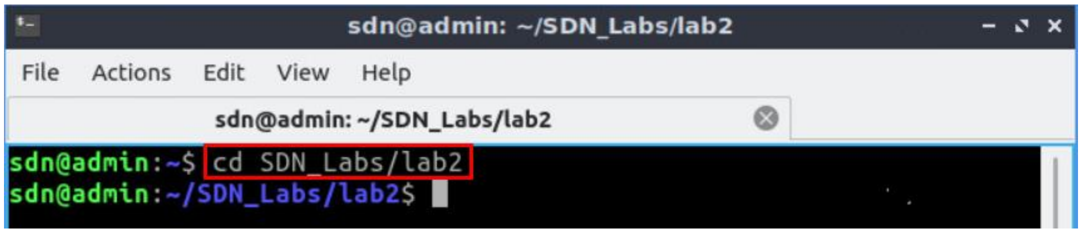


Figure 10. Opening Linux terminal.

**Step 2.** Navigate into *SDN_Labs/lab2* 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 IP addresses to the interfaces of the router. The `cd` command is short for change directory followed by an argument that specifies the destination directory.

```
cd SDN_Labs/lab2
```

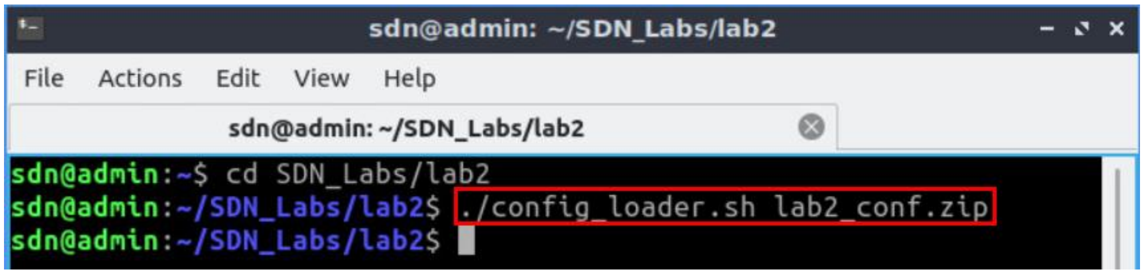


```
sdn@admin:~$ cd SDN_Labs/lab2
sdn@admin:~/SDN_Labs/lab2$
```

Figure 11. Entering the *SDN_Labs/lab2* 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 lab2_conf.zip
```

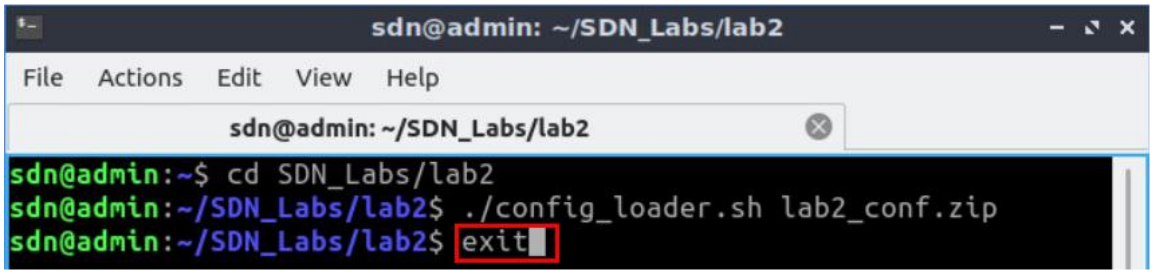


```
sdn@admin:~$ cd SDN_Labs/lab2
sdn@admin:~/SDN_Labs/lab2$ ./config_loader.sh lab2_conf.zip
sdn@admin:~/SDN_Labs/lab2$ 
```

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

**Step 4.** Type the following command to exit the Linux terminal.

```
exit
```



```
sdn@admin:~$ cd SDN_Labs/lab2
sdn@admin:~/SDN_Labs/lab2$ ./config_loader.sh lab2_conf.zip
sdn@admin:~/SDN_Labs/lab2$ exit
```

Figure 13. Exiting from the terminal.

## 2.4 Running the emulation

In this section, you will run the emulation and check the links and interfaces that connect the devices in the given topology.

**Step 1.** At this point host h1 and host h2 interfaces are configured. To proceed with the emulation, click on the *Run* button located in lower left-hand side.

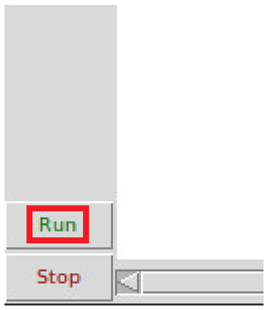
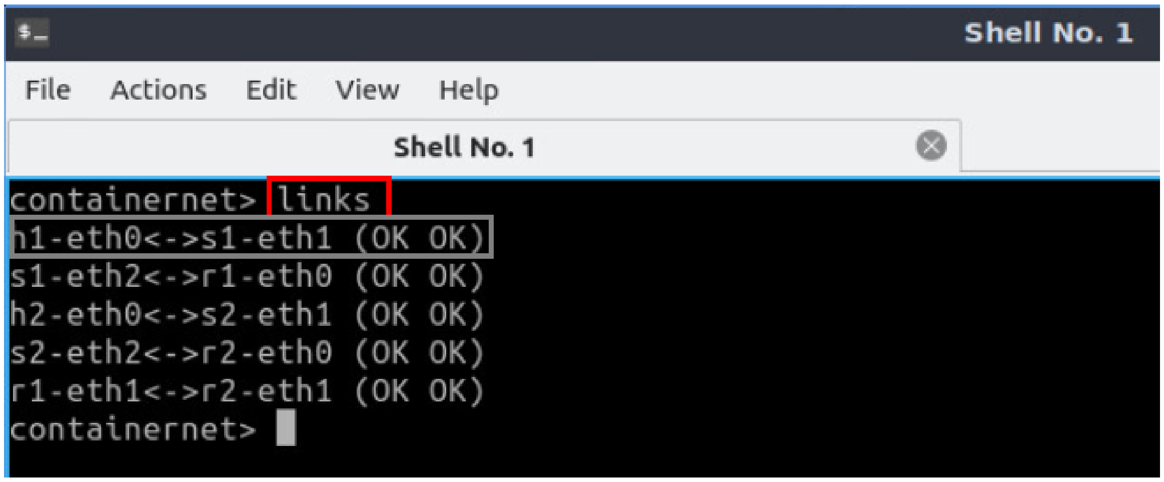


Figure 14. Starting the emulation.

**Step 2.** Issue the following command to display the interface names and connections.

```
links
```



```
containernet> links
h1-eth0<->s1-eth1 (OK OK)
s1-eth2<->r1-eth0 (OK OK)
h2-eth0<->s2-eth1 (OK OK)
s2-eth2<->r2-eth0 (OK OK)
r1-eth1<->r2-eth1 (OK OK)
containernet>
```

Figure 15. Displaying network interfaces.

In Figure 15, 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*).

## 2.5 Verify the configuration

You will verify the IP addresses listed in Table 2 and inspect the routing table of routers r1 and r2.

**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.

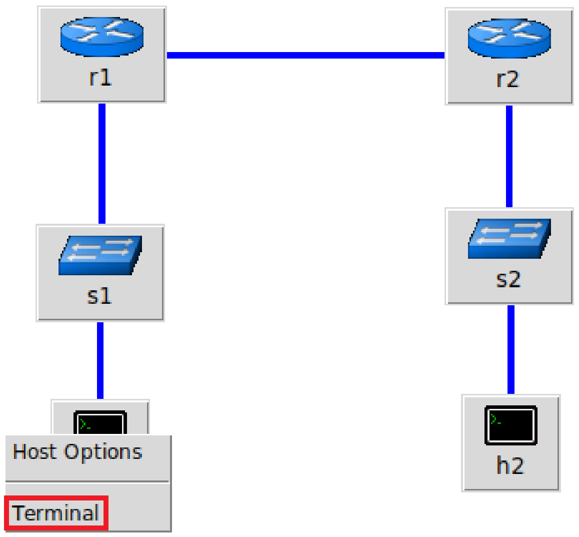
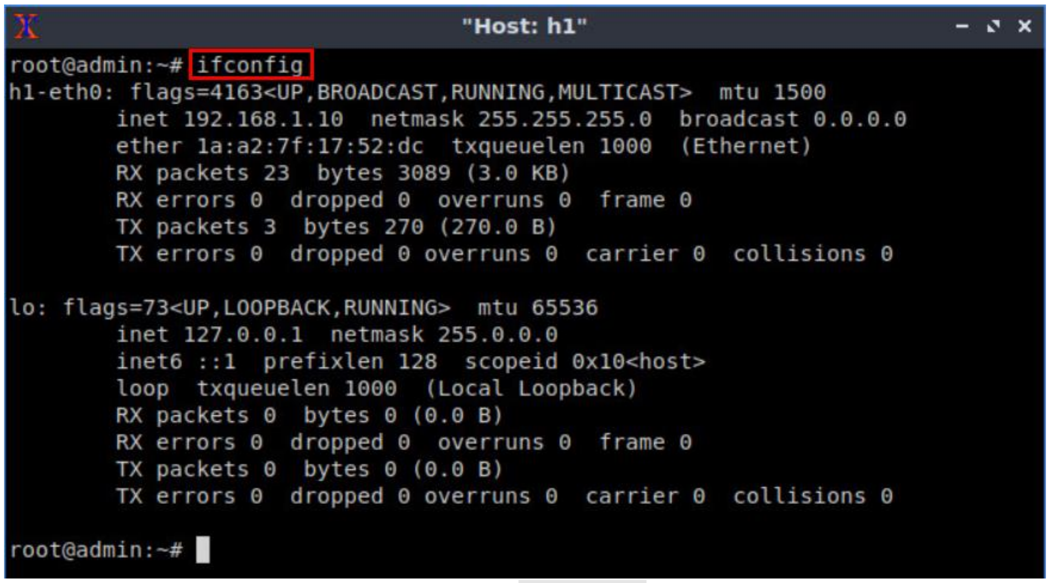


Figure 16. Opening a terminal on host h1.

**Step 2**. On host h1 terminal, type the command shown below to verify that the IP address was assigned successfully. You will corroborate that host h1 has two interfaces. Interface *h1-eth0* is configured with the IP address of 192.168.1.10 and the subnet mask

255.255.255.0. Interface *lo* is configured with the IP address of 127.0.0.1 with the subnet mask of 255.0.0.0.

```
ifconfig
```

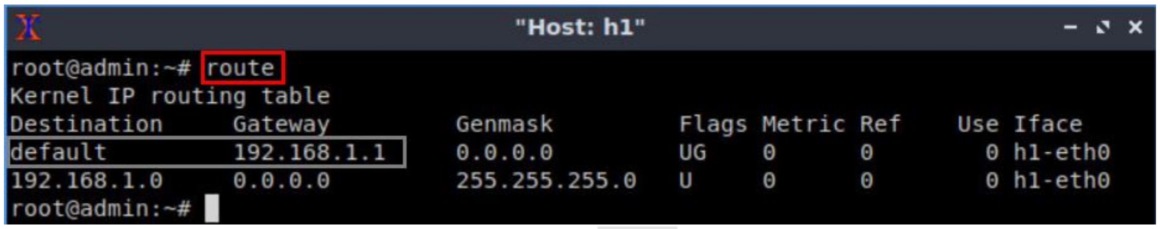


```
root@admin:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.10  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 1a:a2:7f:17:52:dc  txqueuelen 1000  (Ethernet)
        RX packets 23  bytes 3089 (3.0 KB)
        RX errors 0  dropped 0  overruns 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:~#
```

Figure 17. Output of `ifconfig` command.

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

```
route
```



```
root@admin:~# 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:~#
```

Figure 18. Output of `route` command.

**Step 4**. In order to verify host h2 IP address default gateway, proceed similarly by repeating step 1 to step 3 on host h2 terminal. Similar results should be observed.

**Step 5**. In order to verify router r1, hold right-click on router r1 and select *Terminal*.

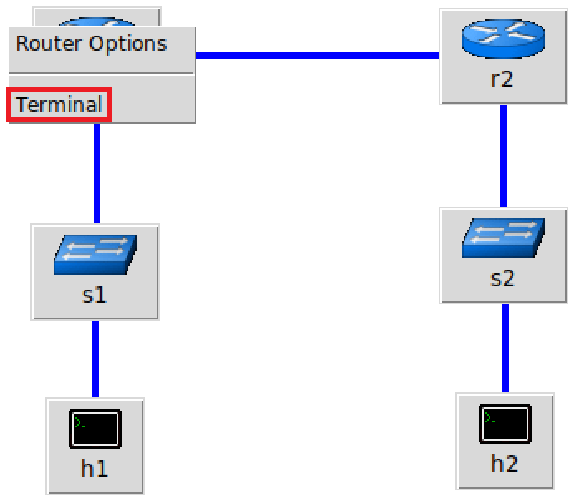
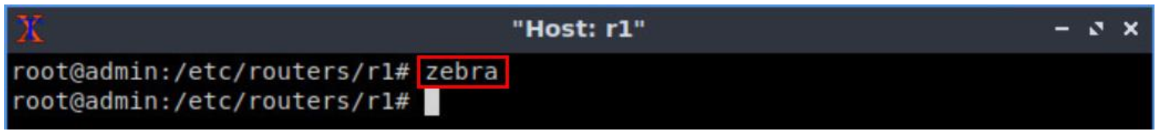


Figure 19. Opening a terminal on router r1.

**Step 6**. In this step, you will start the zebra daemon, 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 zebra, type the following command.

```
zebra
```

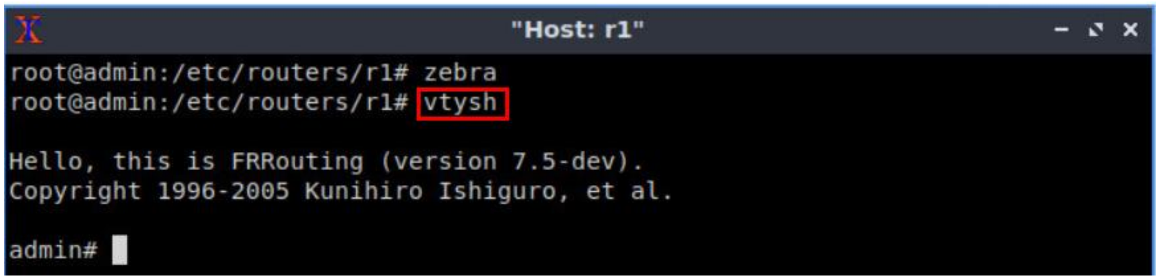


```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# 
```

Figure 20. Starting zebra daemon.

**Step 7**. 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
```

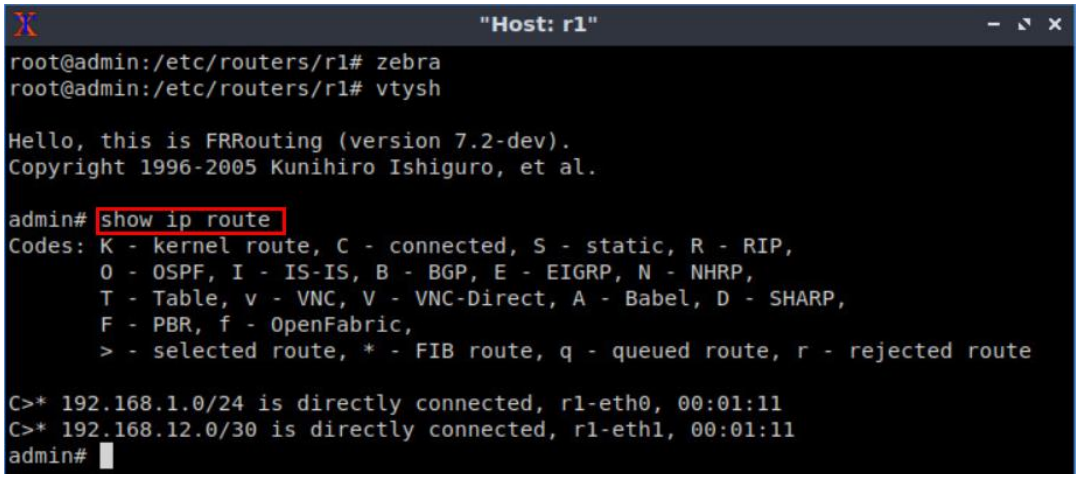


```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.5-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# 
```

Figure 21. Starting vtysh on router r1.

**Step 8.** Type the following command on router r1 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 of router r2 (192.168.2.0/24) as there is no routing protocol configured yet.

```
show ip route
```

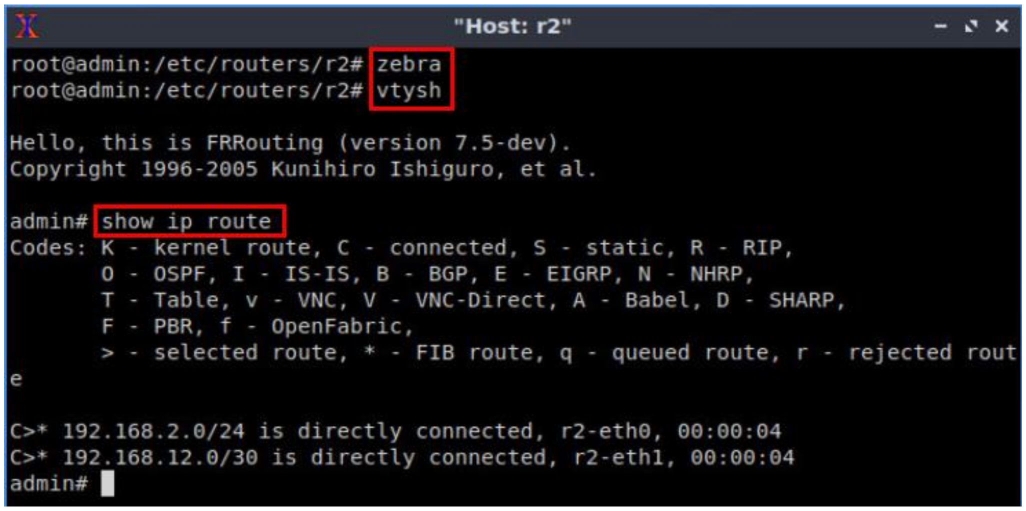


```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

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

admin# 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

C>* 192.168.1.0/24 is directly connected, r1-eth0, 00:01:11
C>* 192.168.12.0/30 is directly connected, r1-eth1, 00:01:11
admin# 
```

Figure 22. Displaying the routing table of router r1.

The output in the figure above shows that the network 192.168.1.0/24 is directly connected through the interface *r1-eth0*. The network 192.168.12.0/30 is connected via the interface *r1-eth1*.

**Step 9.** 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 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 7.5-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

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 rout
e

C>* 192.168.2.0/24 is directly connected, r2-eth0, 00:00:04
C>* 192.168.12.0/30 is directly connected, r2-eth1, 00:00:04
admin# 
```

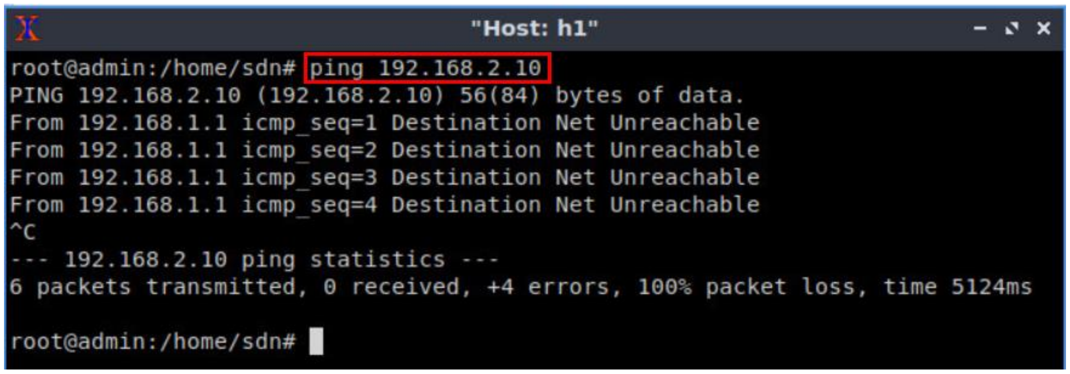
Figure 23. Displaying the routing table of router r2.

## 2.6 Test connectivity between end-hosts

In this section you will run a connectivity test between host 1 and host 2. You will notice that there is no connectivity because there is no routing protocol configured in the routers.

**Step 1.** On host h1 terminal, type the command shown below. Notice that according to Table 1, the IP address 192.168.2.10 is assigned to host h2.

```
ping 192.168.2.10
```



```
root@admin:/home/sdn# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
From 192.168.1.1 icmp_seq=1 Destination Net Unreachable
From 192.168.1.1 icmp_seq=2 Destination Net Unreachable
From 192.168.1.1 icmp_seq=3 Destination Net Unreachable
From 192.168.1.1 icmp_seq=4 Destination Net Unreachable
^C
--- 192.168.2.10 ping statistics ---
6 packets transmitted, 0 received, +4 errors, 100% packet loss, time 5124ms

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

Figure 24. Connectivity test between host h1 and host h2.

To stop the test press `Ctrl+c`. The result in the figure above shows an unsuccessful connectivity test.

## 3 Configure BGP routing protocol

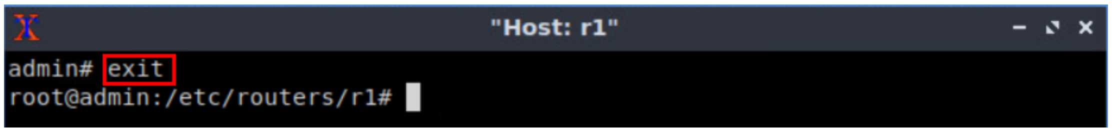
In the previous section you used a script to assign the IP addresses to all the interfaces of the devices, then you performed an unsuccessful connectivity test. In this section you will configure a routing protocol in order to establish a connection between the two networks. You will configure BGP in order to establish a connection between AS 100 and AS 200. First, you will initialize the daemon that enables BGP configuration then. Then, you need to assign BGP neighbors to allow BGP peering to the remote neighbor. Additionally, you will advertise the local networks of each router.

### 3.1 BGP neighbors on the routers

In this section, you will add the neighbor IP address to allow BGP peering to the remote neighbor.

**Step 1.** To configure BGP routing protocol, you need to enable the BGP daemon first. In router r1, type the following command to exit the vtysh session.

```
exit
```

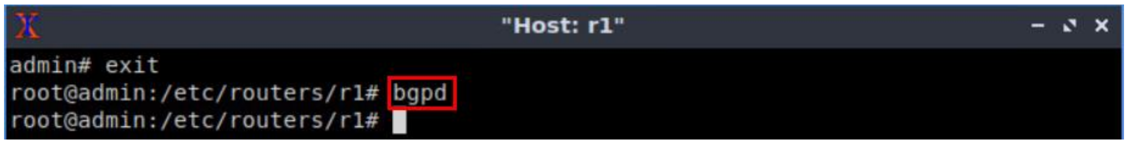


```
admin# exit
root@admin:/etc/routers/r1# 
```

Figure 25. Exiting the vtysh session.

**Step 2.** Type the following command on router r1 terminal to start BGP routing protocol.

```
bgpd
```

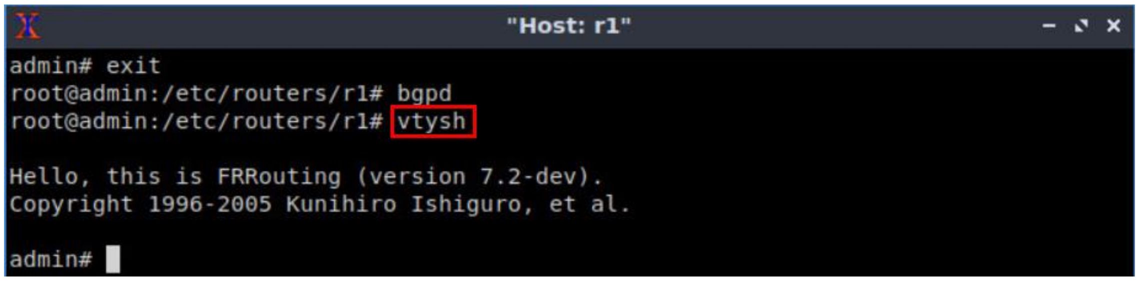


```
admin# exit
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# 
```

Figure 26. Starting BGP daemon.

**Step 3.** In order to enter to router r1 terminal, type the following command.

```
vtysh
```

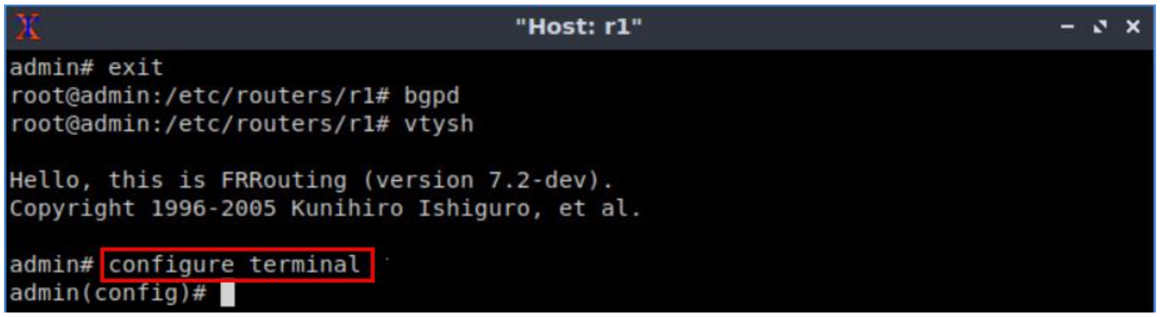


```
admin# exit
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin#
```

Figure 27. Starting vtysh on router r1.

**Step 4.** To enable router r1 configuration mode, issue the following command.

```
configure terminal
```

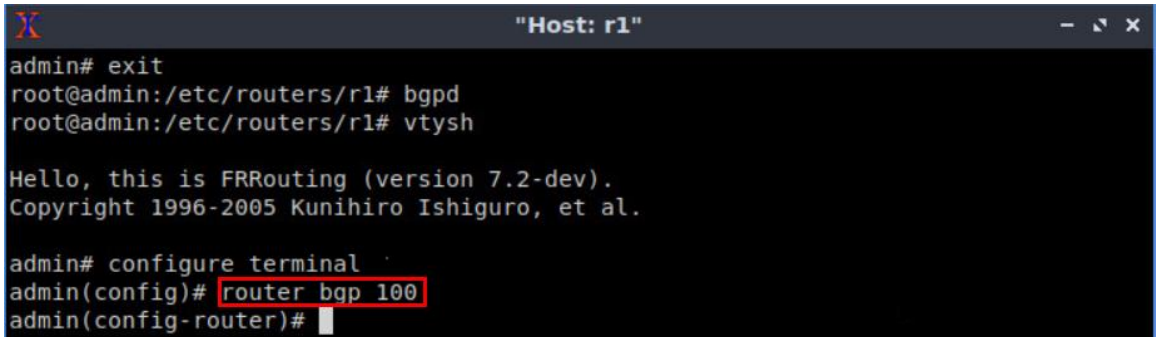


```
admin# exit
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

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

Figure 28. Enabling configuration mode on router r1.

**Step 5.** The ASN assigned for router r1 is 100. In order to configure BGP, type the following command.

```
router bgp 100
```

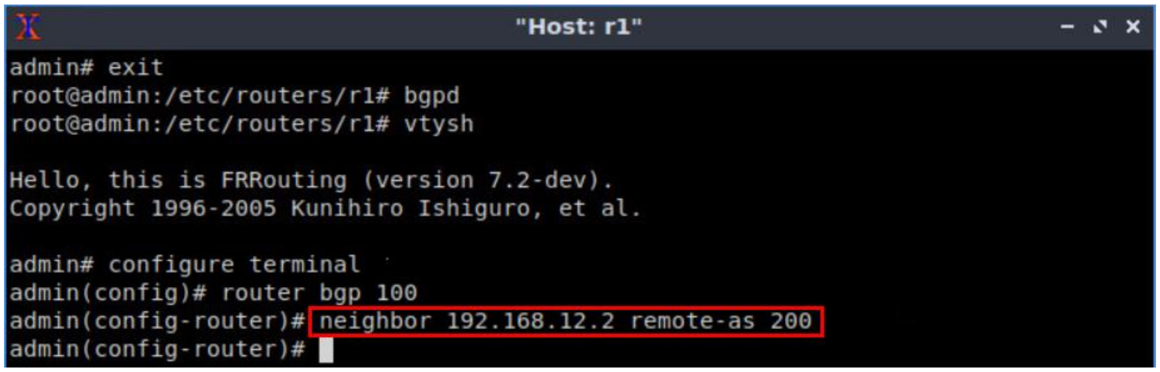


```
admin# exit
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)#
```

Figure 29. Configuring BGP on router r1.

**Step 6.** To configure a BGP neighbor to router r1 (AS 100), type the command shown below. This command specifies the neighbor IP address (192.168.12.2) and ASN of the remote BGP peer (AS 200).

```
neighbor 192.168.12.2 remote-as 200
```

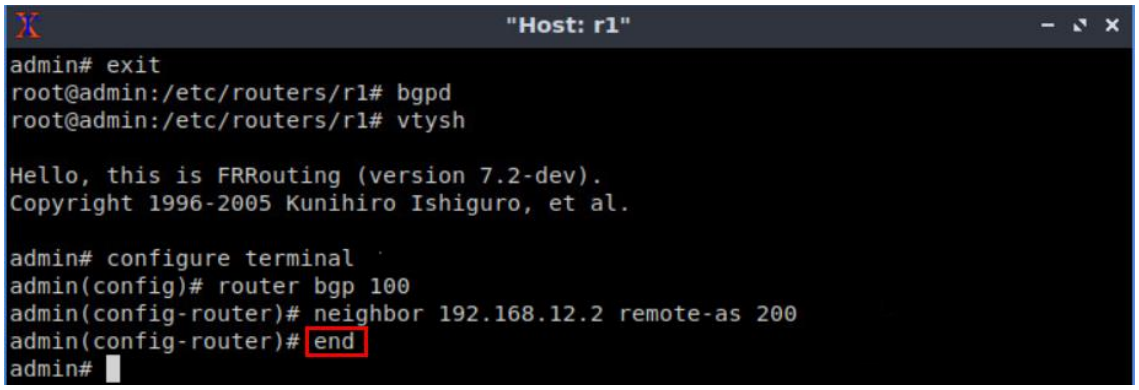


```
admin# exit
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# neighbor 192.168.12.2 remote-as 200
admin(config-router)#
```

Figure 30. Assigning BGP neighbor to router r1.

**Step 7.** Type the following command to exit from the configuration mode.

```
end
```

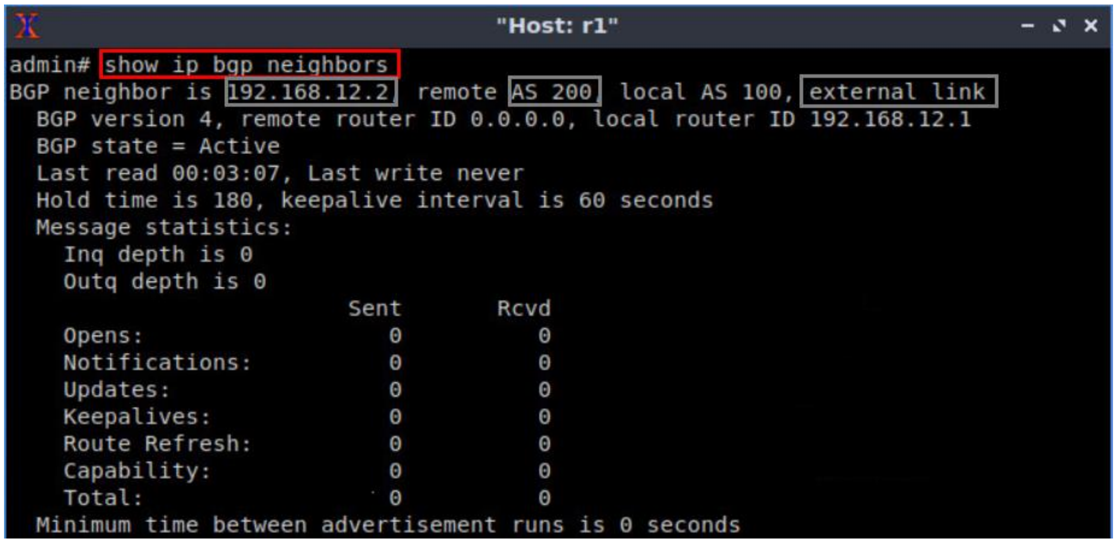


```
admin# exit
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# neighbor 192.168.12.2 remote-as 200
admin(config-router)# end
admin#
```

Figure 31. Exiting from configuration mode.

**Step 8.** Type the following command to verify BGP neighbors. You will verify that the neighbor IP address is 192.168.12.2. The corresponding ASN is 200.

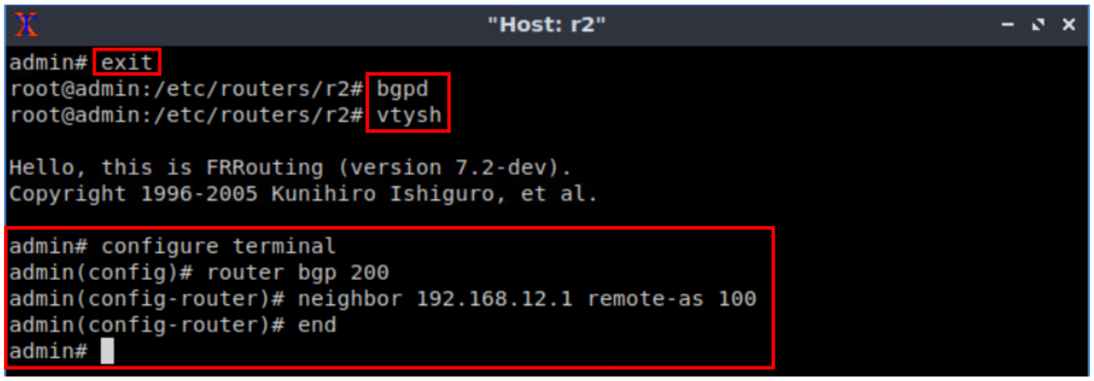
```
show ip bgp neighbors
```



```
admin# show ip bgp neighbors
BGP neighbor is 192.168.12.2, remote AS 200, local AS 100, external link
  BGP version 4, remote router ID 0.0.0.0, local router ID 192.168.12.1
  BGP state = Active
  Last read 00:03:07, Last write never
  Hold time is 180, keepalive interval is 60 seconds
  Message statistics:
    Inq depth is 0
    Outq depth is 0
                          Sent       Rcvd
    Opens:                   0          0
    Notifications:           0          0
    Updates:                 0          0
    Keepalives:              0          0
    Route Refresh:           0          0
    Capability:              0          0
    Total:                   0          0
  Minimum time between advertisement runs is 0 seconds
```

Figure 32. Verifying BGP neighbors on router r1.

**Step 9.** 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 terminal, issue the commands depicted below. At the end, you will verify all the directly connected networks of router r2.

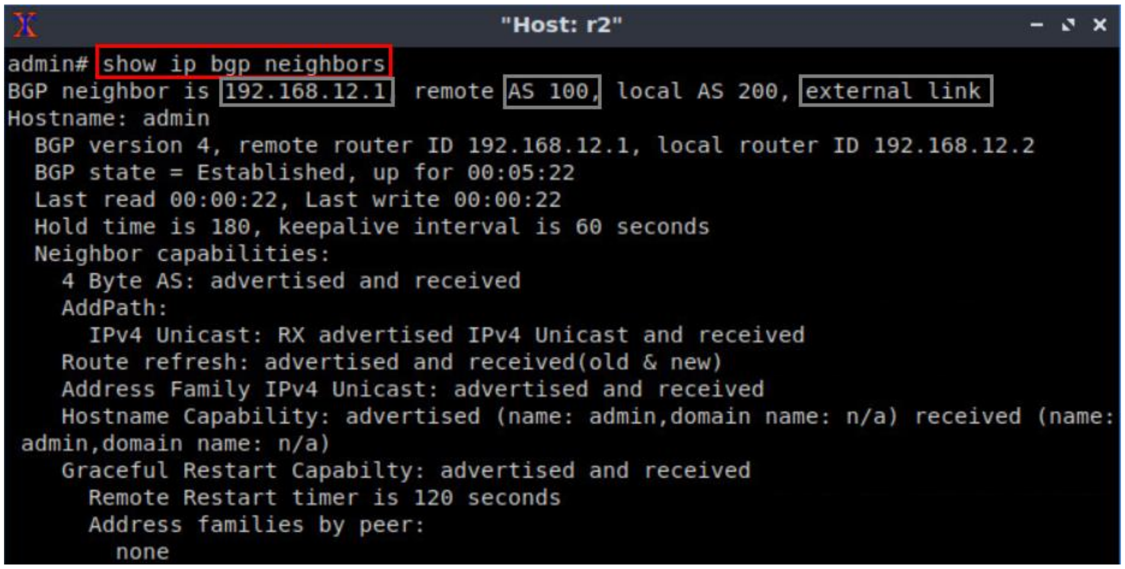


```
admin# exit
root@admin:/etc/routers/r2# bgpd
root@admin:/etc/routers/r2# vtysh

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

admin# configure terminal
admin(config)# router bgp 200
admin(config-router)# neighbor 192.168.12.1 remote-as 100
admin(config-router)# end
admin#
```

Figure 33. Assigning BGP neighbor to router r2.

**Step 10.** Type the following command to verify BGP neighbors. You will verify that the neighbor IP address is 192.168.12.1. The corresponding ASN is 100.

```
show ip bgp neighbors
```

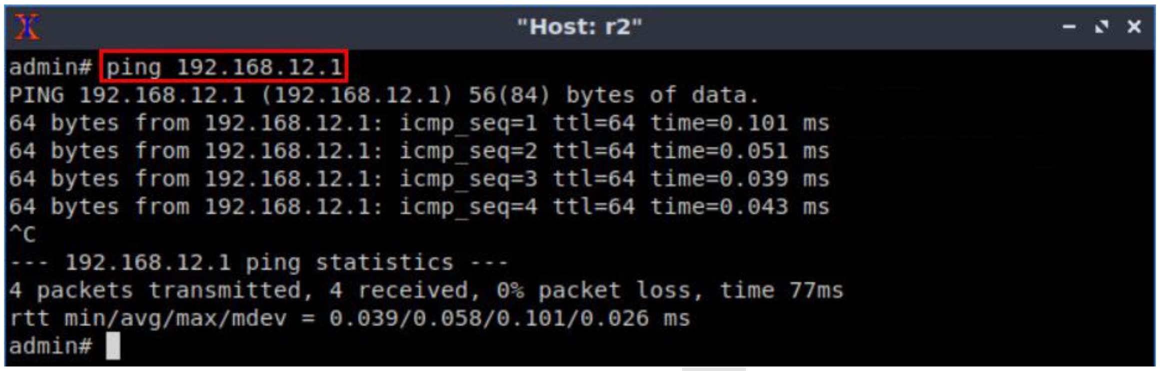


```
admin# show ip bgp neighbors
BGP neighbor is 192.168.12.1, remote AS 100, local AS 200, external link
Hostname: admin
  BGP version 4, remote router ID 192.168.12.1, local router ID 192.168.12.2
  BGP state = Established, up for 00:05:22
  Last read 00:00:22, Last write 00:00:22
  Hold time is 180, keepalive interval is 60 seconds
  Neighbor capabilities:
    4 Byte AS: advertised and received
    AddPath:
      IPv4 Unicast: RX advertised IPv4 Unicast and received
    Route refresh: advertised and received(old & new)
    Address Family IPv4 Unicast: advertised and received
    Hostname Capability: advertised (name: admin,domain name: n/a) received (name:
 admin,domain name: n/a)
    Graceful Restart Capabilty: advertised and received
      Remote Restart timer is 120 seconds
      Address families by peer:
        none
```

Figure 34. Verifying BGP neighbors on router r2.

**Step 11.** In router r2 terminal, perform a connectivity test by running the command shown below.

```
ping 192.168.12.1
```



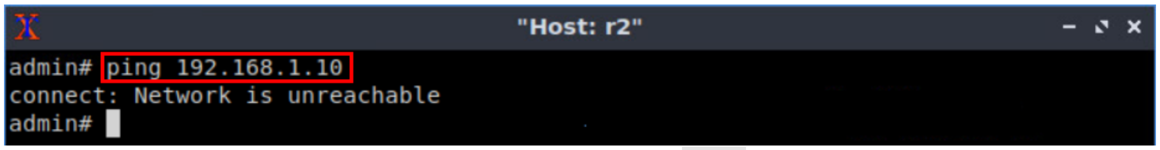
```
admin# ping 192.168.12.1
PING 192.168.12.1 (192.168.12.1) 56(84) bytes of data.
64 bytes from 192.168.12.1: icmp_seq=1 ttl=64 time=0.101 ms
64 bytes from 192.168.12.1: icmp_seq=2 ttl=64 time=0.051 ms
64 bytes from 192.168.12.1: icmp_seq=3 ttl=64 time=0.039 ms
64 bytes from 192.168.12.1: icmp_seq=4 ttl=64 time=0.043 ms
^C
--- 192.168.12.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 77ms
rtt min/avg/max/mdev = 0.039/0.058/0.101/0.026 ms
admin#
```

Figure 35. Connectivity test using `ping` command.

To stop the test, press `Ctrl+c`. The result in the figure above shows a successful connectivity test between router r1 and router r2.

**Step 12.** In router r2 terminal, perform a connectivity test between router r2 and host h1 by issuing the command shown below.

```
ping 192.168.1.10
```



```
admin# ping 192.168.1.10
connect: Network is unreachable
admin#
```

Figure 36. Connectivity test using `ping` command.

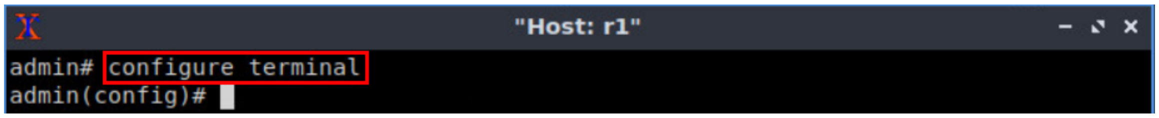
To stop the test, press `Ctrl+c`. As shown in the figure above, router r2 cannot reach host h1 at this point as the routing table of router r2 does not contain the network address of host h1.

## 3.2 Advertise local networks on the routers

In this section, you will advertise the LANs so that the neighbor can receive the network address through EBGP.

**Step 1.** In router r1 terminal, issue the following command.

```
configure terminal
```

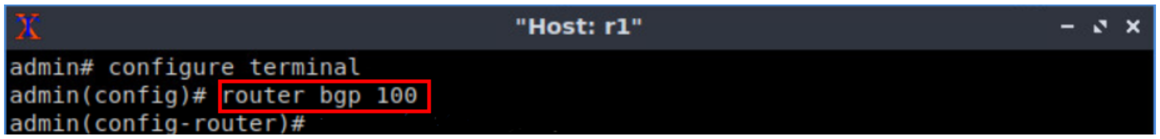


```
admin# configure terminal
admin(config)#
```

Figure 37. Enabling configuration mode on router r1.

**Step 2.** You will advertise the LAN connected to router r1 via BGP. Type the following command to enable BGP configuration mode.

```
router bgp 100
```

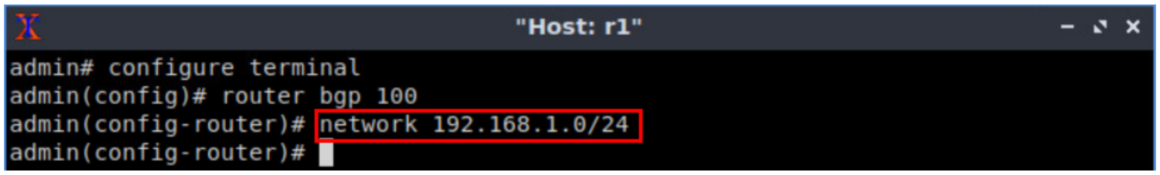


```
admin# configure terminal
admin(config)# router bgp 100
admin(config-router)#
```

Figure 38. Entering to BGP configuration mode.

**Step 3.** Issue the following command so that router r1 advertises the network 192.168.1.0/24.

```
network 192.168.1.0/24
```

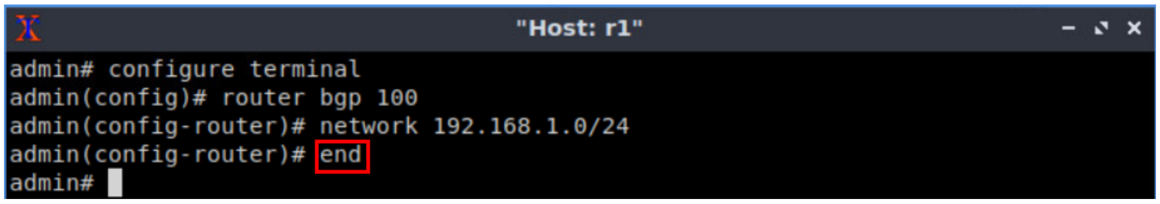


```
admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# network 192.168.1.0/24
admin(config-router)#
```

Figure 39. Advertising the network connected to router r1.

**Step 4.** Type the following command to exit from the configuration mode.

```
end
```

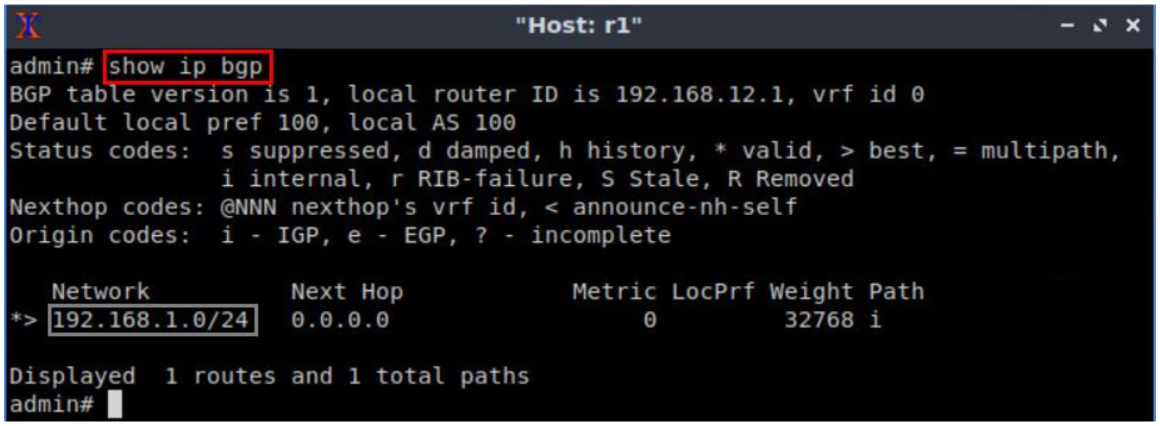


```
admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# network 192.168.1.0/24
admin(config-router)# end
admin#
```

Figure 40. Exiting from configuration mode.

**Step 5.** Type the following command to verify BGP networks.

```
show ip bgp
```

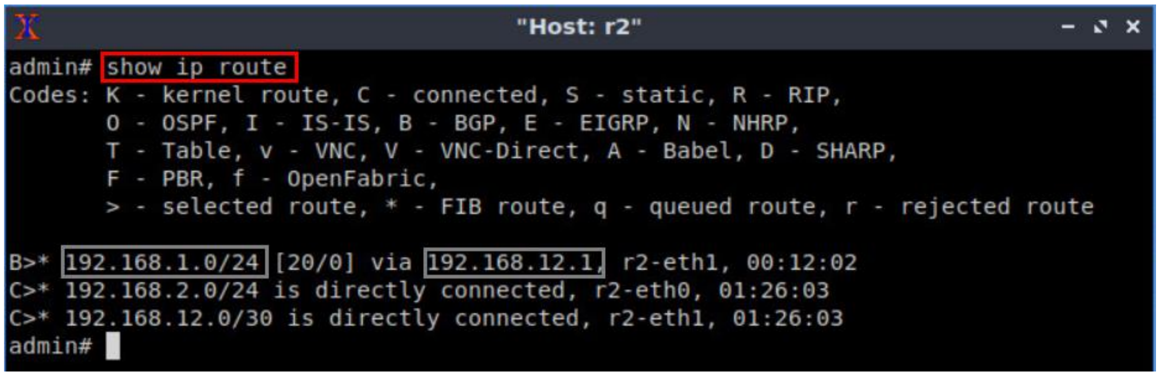


```
admin# show ip bgp
BGP table version is 1, local router ID is 192.168.12.1, vrf id 0
Default local pref 100, local AS 100
Status codes:  s suppressed, d damped, h history, * valid, > best, = multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 192.168.1.0/24   0.0.0.0                  0         32768 i

Displayed  1 routes and 1 total paths
admin#
```

Figure 41. Verifying BGP networks on router r1.

**Step 6.** Type the following command to verify the routing table of router r2. You will observe the route to network 192.168.1.0/24, which is advertised by router r1. It also shows that router r2 will use the neighbor IP 192.168.12.1 to reach the network 192.168.1.0/24.

```
show ip route
```

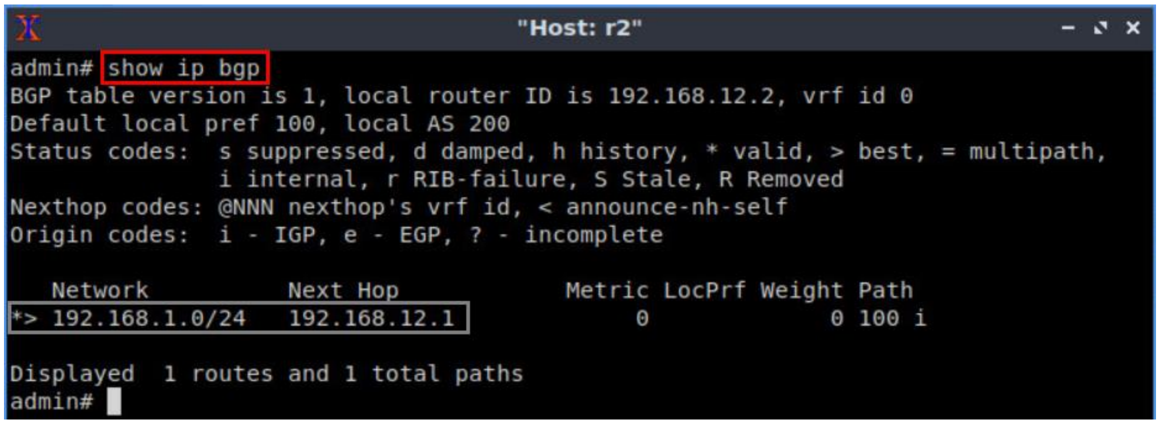


```
"Host: r2"
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected route

B>* 192.168.1.0/24 [20/0] via 192.168.12.1, r2-eth1, 00:12:02
C>* 192.168.2.0/24 is directly connected, r2-eth0, 01:26:03
C>* 192.168.12.0/30 is directly connected, r2-eth1, 01:26:03
admin# 
```

Figure 42. Verifying routing table of router r2.

**Step 7.** In order to verify the BGP table of router r2, issue the command shown below. The output indicates that the network connected to router r1 is listed in the BGP table of router r2. Additionally, it displays the next hop IP address (192.168.12.1) which corresponds to router r2's neighbor IP address (router r1).

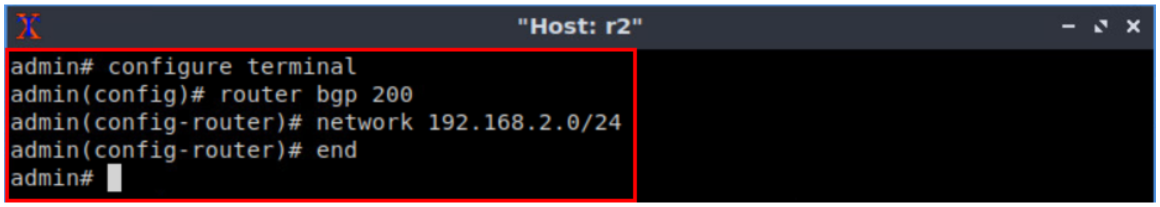
```
show ip bgp
```



```
"Host: r2"
admin# show ip bgp
BGP table version is 1, local router ID is 192.168.12.2, vrf id 0
Default local pref 100, local AS 200
Status codes:  s suppressed, d damped, h history, * valid, > best, = multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 192.168.1.0/24   192.168.12.1             0             0 100 i

Displayed  1 routes and 1 total paths
admin# 
```

Figure 43. Verifying BGP table of router r2.

**Step 8.** Follow from step 1 to step 4 but with different metrics in order to advertise the LAN connected to router r2. All these steps are summarized in the following figure.

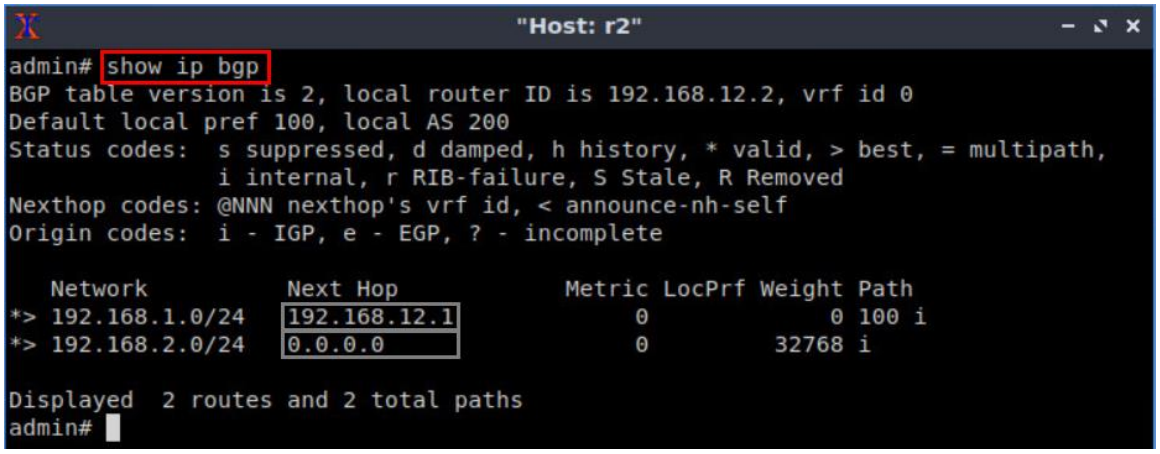


```
"Host: r2"
admin# configure terminal
admin(config)# router bgp 200
admin(config-router)# network 192.168.2.0/24
admin(config-router)# end
admin# 
```

Figure 44. Advertising the network connected to router r2.

**Step 9.** In router r2 terminal, issue the following command to verify the BGP table of router r2. The output will list all the available BGP networks. In particular, the routing table contains its own network (192.168.2.0/24) and the remote network (192.168.1.0/24) which was advertised via EBGP.

```
show ip bgp
```

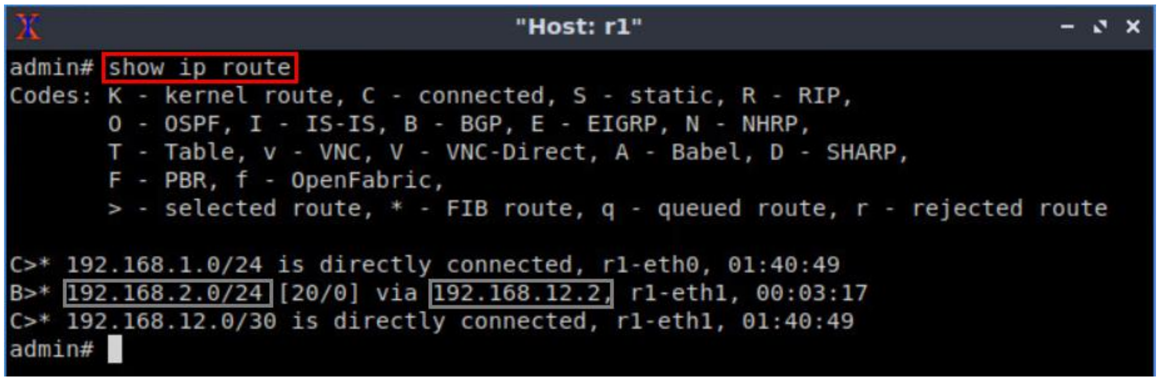


```
"Host: r2"
admin# show ip bgp
BGP table version is 2, local router ID is 192.168.12.2, vrf id 0
Default local pref 100, local AS 200
Status codes:  s suppressed, d damped, h history, * valid, > best, = multipath,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 192.168.1.0/24   192.168.12.1             0             0 100 i
*> 192.168.2.0/24   0.0.0.0                  0         32768 i

Displayed  2 routes and 2 total paths
admin#
```

Figure 45. Verifying BGP table of router r2.

**Step 10.** In router r1 terminal, verify the routing table by typing the following command. The output lists that router r1 contains a route to the network 192.168.2.0/24. Notice that, this route was advertised by router r2.

```
show ip route
```



```
"Host: r1"
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

C>* 192.168.1.0/24 is directly connected, r1-eth0, 01:40:49
B>* 192.168.2.0/24 [20/0] via 192.168.12.2, r1-eth1, 00:03:17
C>* 192.168.12.0/30 is directly connected, r1-eth1, 01:40:49
admin#
```

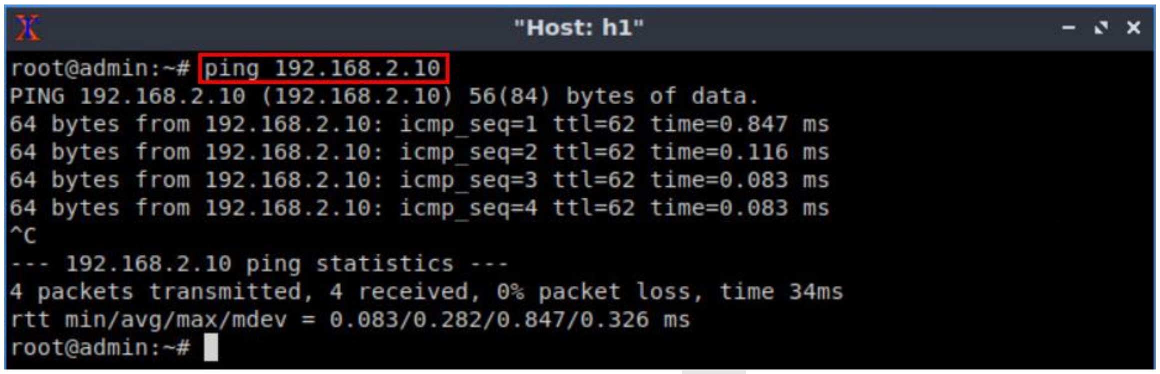
Figure 46. Verifying routing table of router r1.

## 4 Verify connections

In this section, you will verify that the applied configuration is working correctly by running a connectivity test between host h1 and host h2.

**Step 1.** On host h1 terminal, perform a connectivity test between host h1 and host h2 by issuing the command shown below.

```
ping 192.168.2.10
```



```
"Host: h1"
root@admin:~# 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=62 time=0.847 ms
64 bytes from 192.168.2.10: icmp_seq=2 ttl=62 time=0.116 ms
64 bytes from 192.168.2.10: icmp_seq=3 ttl=62 time=0.083 ms
64 bytes from 192.168.2.10: icmp_seq=4 ttl=62 time=0.083 ms
^C
--- 192.168.2.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 34ms
rtt min/avg/max/mdev = 0.083/0.282/0.847/0.326 ms
root@admin:~#
```

Figure 47. Connectivity test using `ping` command.

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

**Step 2.** Hold right-click on host h2 and select *Terminal*. This opens the terminal of host h2.

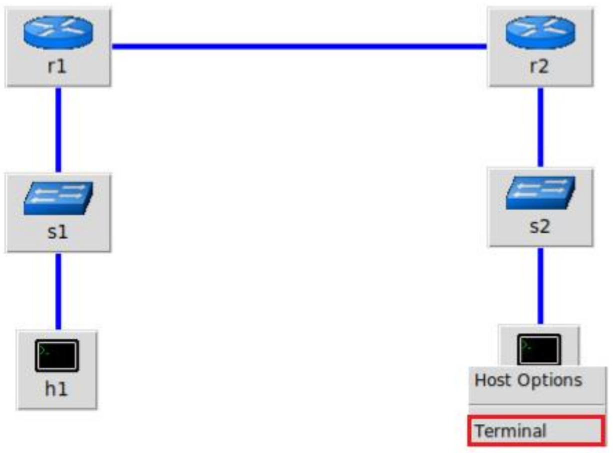
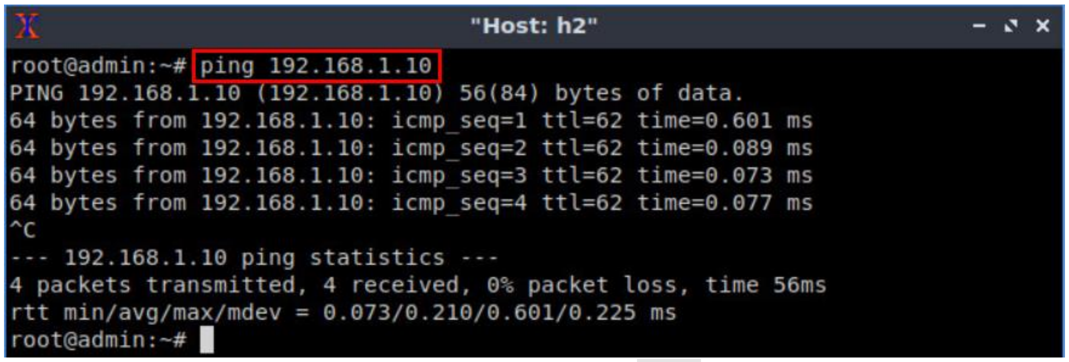


Figure 48. Opening host h2 terminal.

**Step 3.** Similarly, on host h2 terminal, perform a connectivity test between host h2 and host h1 by issuing the command shown below.

```
ping 192.168.1.10
```



```
"Host: h2"
root@admin:~# ping 192.168.1.10
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=62 time=0.601 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=62 time=0.089 ms
64 bytes from 192.168.1.10: icmp_seq=3 ttl=62 time=0.073 ms
64 bytes from 192.168.1.10: icmp_seq=4 ttl=62 time=0.077 ms
^C
--- 192.168.1.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 56ms
rtt min/avg/max/mdev = 0.073/0.210/0.601/0.225 ms
root@admin:~#
```

Figure 49. Connectivity test using `ping` command.

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

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

## References

1. Linux foundation collaborative projects, "*FRR routing documentation*", 2017. [Online]. Available: http://docs.frrouting.org/en/latest/
2. G. Malkin, "*RIP Version 2*," RFC 2453 updated by RFC 4822, 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2453.txt.
3. J. Moy, "*OSPF version 2*", 1998. [Online]. Available: https://www.hjp.at/doc/rfc/rfc2178.html
4. Y. Rekhter, T. Li, S. Hares, "*A border gateway protocol 4 (BGP-4),*" RFC 4271 updated by RFCs 6286, 6608, 6793, 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4271.txt.
5. D. Oran, "*OSI IS-IS intra-domain routing protocol,*" RFC 1142, 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1142.txt.
6. P. Jakma, D. Lamparter. "*Introduction to the quagga routing suite*," 2014, *IEEE Network* 28.
7. K. Ishiguro, "*Gnu zebra,*". [Online]. Available: *http://www. zebra. org* (2002).
8. A. Tanenbaum, D. Wetherall, "*Computer networks*", 5th Edition, Pearson, 2012.
9. Mininet walkthrough. [Online]. Available: http://Mininet.org.
10. B. Lantz, G. Gee, "*MiniEdit: a simple network editor for Mininet,*" 2013. [Online]. Available: https://github.com/Mininet/Mininet/blob/master/examples.
11. P. Goransson, C. Black, T. Culver. "*Software defined networks: a comprehensive approach*". Morgan Kaufmann, 2016

UNIVERSITY OF
SOUTH CAROLINA

# SOFTWARE DEFINED NETWORKING

# Lab 3: Early Efforts of SDN: MPLS Example of a Control Plane that Establishes Semi-static Forwarding Paths

**Document Version: 07-04-2021**

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

## Contents

## Overview

This lab presents an introduction to Multiprotocol Label Switching (MPLS). This protocol allows routers to forward packets based on fixed-length labels rather than the destination IP addresses. In this lab, static MPLS will be configured and verified between two hosts that are required to exchange routes.

## Objectives

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

1. Understand the concept of MPLS.
2. Explore MPLS label distribution protocols.
3. Configure static MPLS in routers.
4. Verify MPLS labels by capturing the traffic between routers.

## Lab settings

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

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.
2. Section 2: Lab topology.
3. Section 3: Configuring static MPLS from router r1 to router r2.
4. Section 4: Configuring static MPLS from router r2 to router r1.
5. Section 5: Verifying the configuration.

## 1 Introduction

### 1.1 Introduction to MPLS

In traditional IP routing, each router must look up the destination IP address of the packet to make the forwarding decision. MPLS eliminates the look up process and ensures transmission between end nodes with short path labels. The predetermined paths that make MPLS work are called label-switched paths (LSPs). A router that operates at the edge of an MPLS network and acts as the entry and exit point for the network is called Label Edge Router (LER). The packet enters the edge of the MPLS backbone is examined and forwarded to the next hop in the pre-set Label Switched Path (LSP). As the packet travels that path, each router on the path uses the label – not other information, such as the IP. An MPLS router that performs routing based only on the label is called a label switch router (LSR). Each LSR has the ability to do three main things: push, pop, or swap labels from a packet. To push a label simply means to add a label to a packet, to pop is to remove a label from a packet, and to swap is to remove and add an alternative label to the packet[1]. A packet can have multiple labels attached which are arranged in a stack and are considered in the order from the most recent label to the least recent label.

However, within each router, the incoming label is examined, and its next hop is matched with a new label. The old label is replaced with the new label for the packet's next destination, and then the freshly labeled packet is sent to the next router. Each router repeats the process until the packet reaches the exit router. The label information is removed at either the last hop or the exit router so that the packet goes back to being identified by an IP header instead of an MPLS label.

The ingress LER is the first to insert an MPLS header and label on a packet. The egress LER is the last point before leaving the network and removes all the MPLS labels and header. Both the ingress and egress LER's are considered as Provider Edge (PE) routers. LSR's are considered as Provider (P) routers.

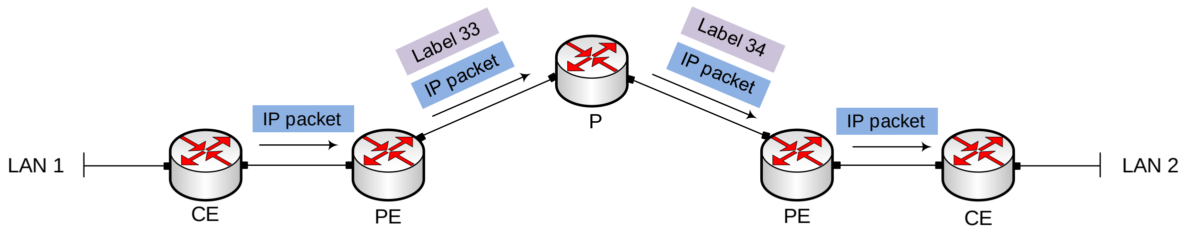


Figure 1. MPLS label forwarding.

Consider Figure 1. Customer Edge (CE) of Local Area Network (LAN) 1 will forward an IP packet to the provider Edge (PE). PE will push label 33 along with the IP packet to the provider (P). P will replace the label with label 34 which will be forwarded to the Provider Edge (PE). PE will pop the label and the IP packet will be delivered to Customer Edge (CE) of LAN 2 which is the destination router.

## 1.2 Label distribution protocols

Label distribution protocol is a set of procedures that provides the information MPLS uses to create the forwarding tables in each LSR in the MPLS domain[2]. Followings are the most widely used label distribution protocols in MPLS.

**Static MPLS:** MPLS entries can be configured statically, handling MPLS consists of pushing, swapping, or popping labels to IP packets.

**LDP:** LDP is a protocol that automatically generates and exchanges labels between routers. LDP enables LSRs to discover potential peers and to establish LDP sessions. It depends on the network's Interior Gateway Protocol (IGP) to determine the path an LSP must take.

**Resource Reservation Protocol-Traffic Engineering (RSVP-TE):** RSVP-TE is used to establish MPLS transport LSPs when there are traffic engineering requirements. RSVP is a signaling protocol that handles bandwidth allocation and true traffic engineering across an MPLS network. When RSVP and MPLS are combined, a flow or session can be defined with greater flexibility and generality.

## 1.3 MPLS header architecture

MPLS operates at a layer that is generally considered to lie between OSI Layer 2 (data link layer) and Layer 3 (network layer), often referred to as a layer 2.5 protocol.

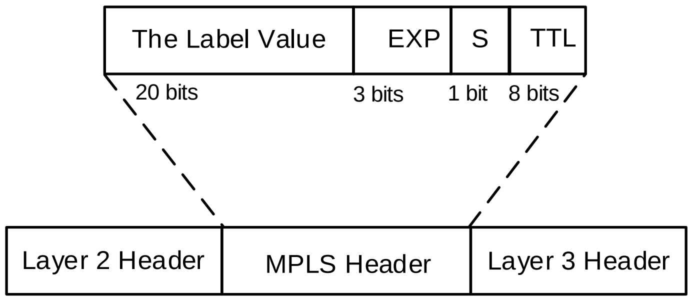


Figure 2. MPLS header architecture.

Consider Figure 2. MPLS works by prefixing packets with an MPLS header, containing one or more labels. This is called a label stack. Each entry in the label stack contains four fields.

1. **The Label Value:** The first 20 bits are the label value. This value can be between 0 and 1,048,575. The first 16 values are exempted from normal use.

2. **EXP:** Three bits are the experimental (EXP) bits. These bits are used solely for quality of service (QoS) purposes, which represents the set of techniques necessary to manage network bandwidth, delay, jitter, and packet loss.

3. **S:** 1 bit is the Bottom of Stack (BoS) bit. The stack is the collection of labels that are found on top of the packet. The BoS bit is set to 1 if this is the bottom label in the stack. Otherwise, the BoS bit remains 0.

4. **TTL:** Last 8 bits are used for Time-to-Live (TTL). This TTL has the same function as the TTL found in the IP header. It is simply decreased by 1 at each hop, and its main function is to avoid a packet being stuck in a routing loop.

## 2 Lab topology

Consider Figure 3. The topology consists of three routers, two switches and two end hosts. Customer 1 (h1) and Customer 2 (h2) are allowed to exchange routes using static MPLS.

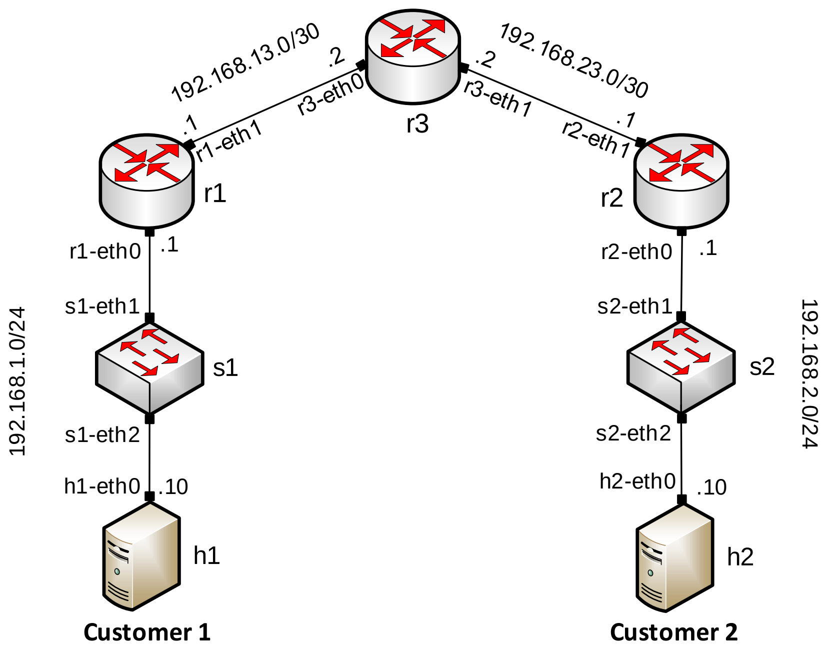


Figure 3. Lab topology.

### 2.1 Lab settings

Routers and hosts are already configured according to Table 2.

Table 2. Topology information.

| Device | Interface | IPV4 Address | Subnet | Default gateway |
|---|---|---|---|---|
| r1 | r1-eth0 | 192.168.1.1 | /24 | N/A |
| | r1-eth1 | 192.168.13.1 | /30 | N/A |
| r2 | r2-eth0 | 192.168.2.1 | /24 | N/A |
| | r2-eth1 | 192.168.23.1 | /30 | N/A |
| r3 | r3-eth0 | 192.168.13.2 | /30 | N/A |
| | r3-eth1 | 192.168.23.2 | /30 | 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 |

## 2.2 Open the topology and load the configuration

In this section, you will open MiniEdit[7] 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 (IP addresses, default gateway), saving topologies, and exporting layer 2 models.

**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`.

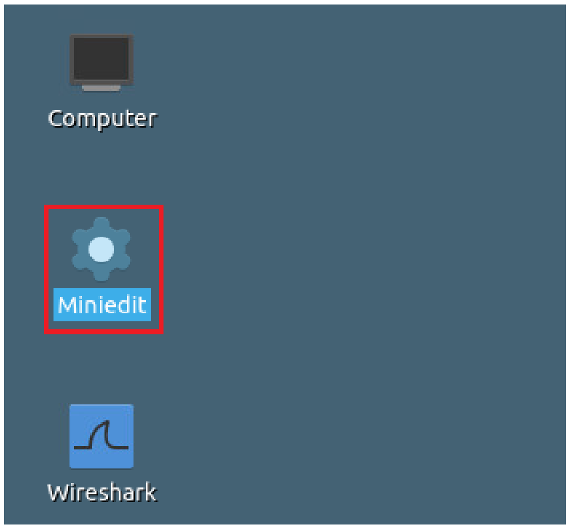
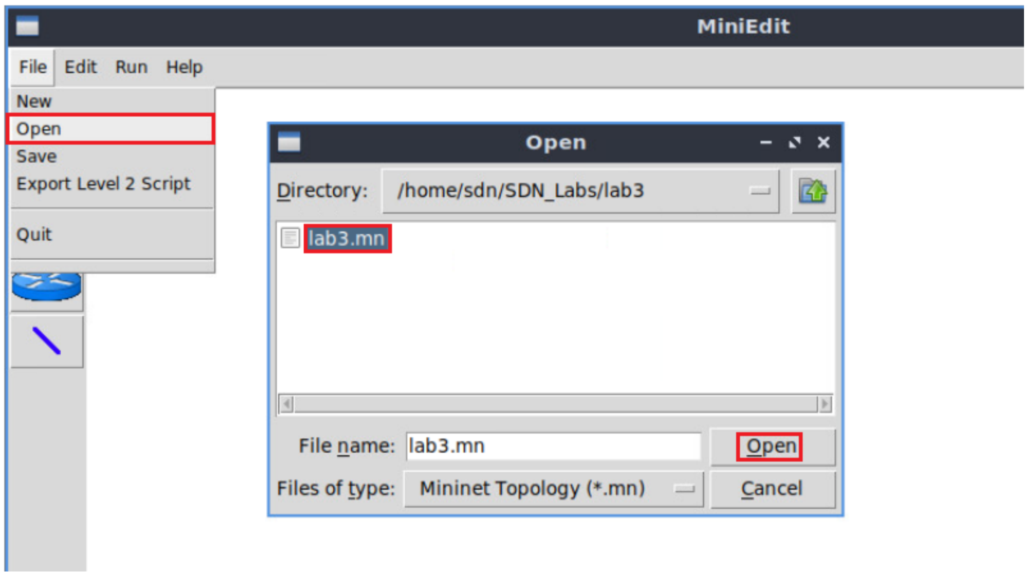


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then *open* to load the lab topology. Open the *Lab3.mn* topology file stored in the default directory, */home/sdn/SDN_Labs/lab3* and click on *Open*.



Directory: /home/sdn/SDN_Labs/lab3

lab3.mn

File name: lab3.mn

Files of type: Mininet Topology (*.mn)

Figure 5. MiniEdit's open dialog.

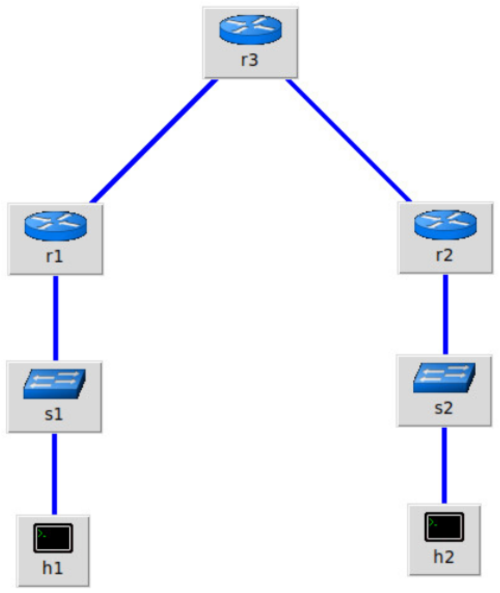


Figure 6. Mininet topology.

At this point, the topology is loaded. However, the interfaces are not configured. In order to assign IP addresses to the interfaces of the devices, you will execute a script that loads the configuration to the routers.

**Step 3.** Click on the icon below to open the Linux terminal.

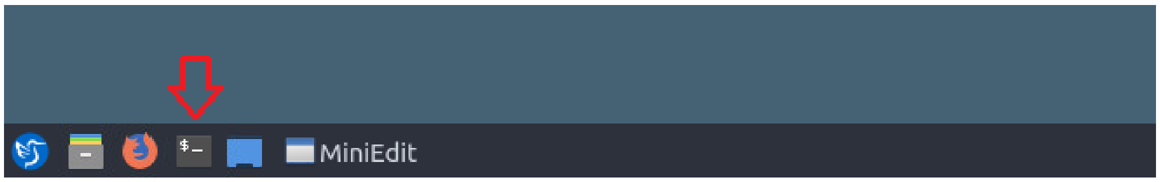
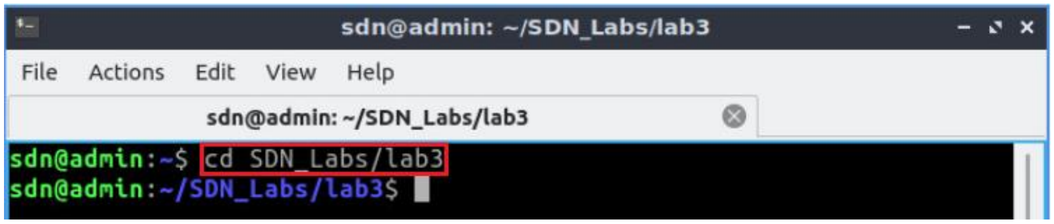


Figure 7. Opening the Linux terminal.

**Step 4.** Click on the Linux terminal and navigate into the *SDN_Labs/lab3* 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 IP addresses to the interfaces of the routers. The `cd` command is short for change directory followed by an argument that specifies the destination directory.

```
cd SDN_Labs/lab3
```

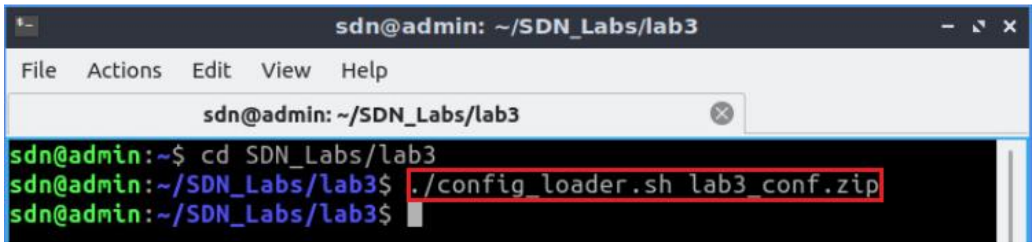

sdn@admin: ~/SDN_Labs/lab3
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab3

```
sdn@admin:~$ cd SDN_Labs/lab3
sdn@admin:~/SDN_Labs/lab3$ 
```

Figure 8. Entering the *SDN_Labs/lab3* directory.

**Step 5.** 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 lab3_conf.zip
```



```
sdn@admin:~$ cd SDN_Labs/lab3
sdn@admin:~/SDN_Labs/lab3$ ./config_loader.sh lab3_conf.zip
sdn@admin:~/SDN_Labs/lab3$
```

Figure 9. Executing the shell script to load the configuration.

**Step 6.** At this point the interfaces of hosts h1 and h2 are configured. To proceed with the emulation, click on the *Run* button located on lower left-hand side.

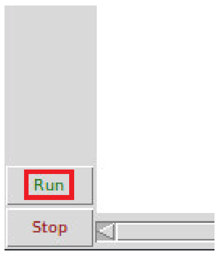


Figure 10. Starting the emulation.

**Step 7.** Click on Mininet's terminal, i.e., the one that launched when MiniEdit was started.

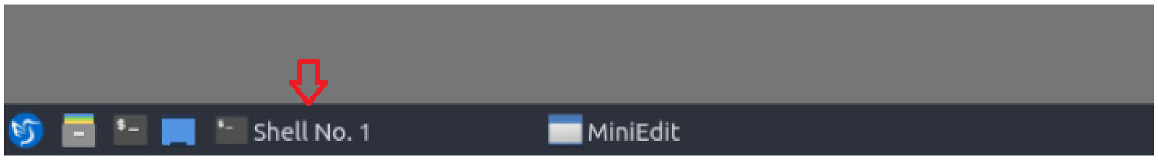
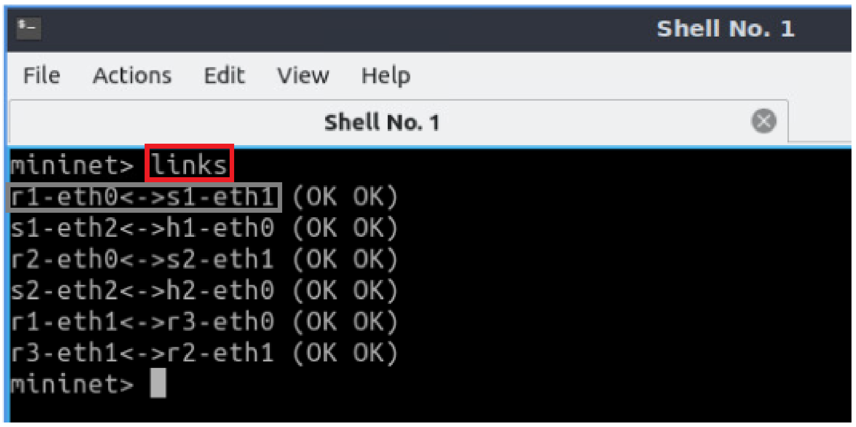


Figure 11. Opening Mininet's terminal.

**Step 8.** Issue the following command to display the interface names and connections.

```
links
```



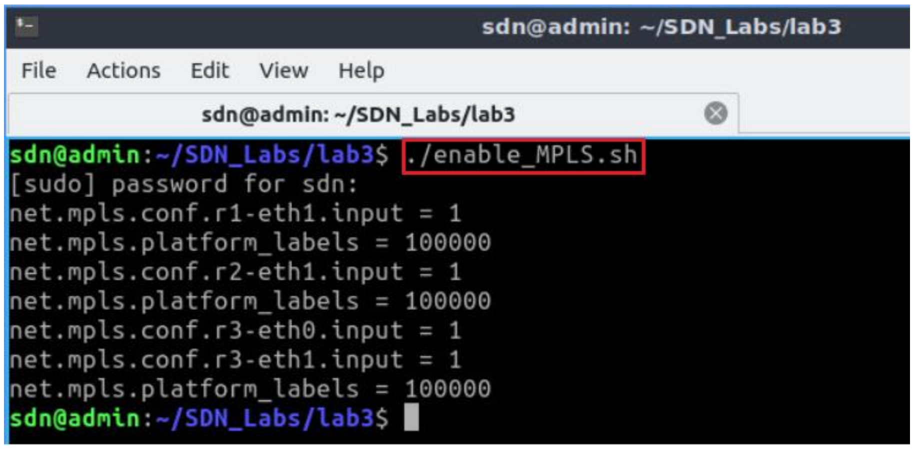
```
mininet> links
r1-eth0<->s1-eth1 (OK OK)
s1-eth2<->h1-eth0 (OK OK)
r2-eth0<->s2-eth1 (OK OK)
s2-eth2<->h2-eth0 (OK OK)
r1-eth1<->r3-eth0 (OK OK)
r3-eth1<->r2-eth1 (OK OK)
mininet>
```

Figure 12. Displaying network interfaces.

In the figure above, the link displayed within the gray box indicates that interface eth0 of router r1 connects to interface eth1 of switch s1 (i.e., *r1-eth0<->s1-eth1*).

**Step 9.** In order to enable MPLS forwarding in all the router's interfaces, type the following command in the opened Linux terminal. If a password is required, type `password`.

```
./enable_MPLS.sh
```


sdn@admin: ~/SDN_Labs/lab3
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab3

```
sdn@admin:~/SDN_Labs/lab3$ ./enable_MPLS.sh
[sudo] password for sdn:
net.mpls.conf.r1-eth1.input = 1
net.mpls.platform_labels = 100000
net.mpls.conf.r2-eth1.input = 1
net.mpls.platform_labels = 100000
net.mpls.conf.r3-eth0.input = 1
net.mpls.conf.r3-eth1.input = 1
net.mpls.platform_labels = 100000
sdn@admin:~/SDN_Labs/lab3$
```

Figure 13. Enabling MPLS forwarding in all the routers.

Consider the figure above. The command executes a shell script that enables MPLS forwarding in all routers. All the router interfaces assign labels that are used to forward packets. Value 1 is assigned to all the router interfaces so that they participate in the label processing. Router interfaces connected to the host do not perform label processing. *Platform_labels* is the table that recognizes all the assigned labels and participates in label forwarding. Value 100000 (maximum value for label forwarding) is assigned to *platform_labels* in order to enable label forwarding.

## 2.3 Load zebra daemon and verify the configuration

You will verify that IP addresses listed in Table 2 and inspect the routing table of the routers.

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

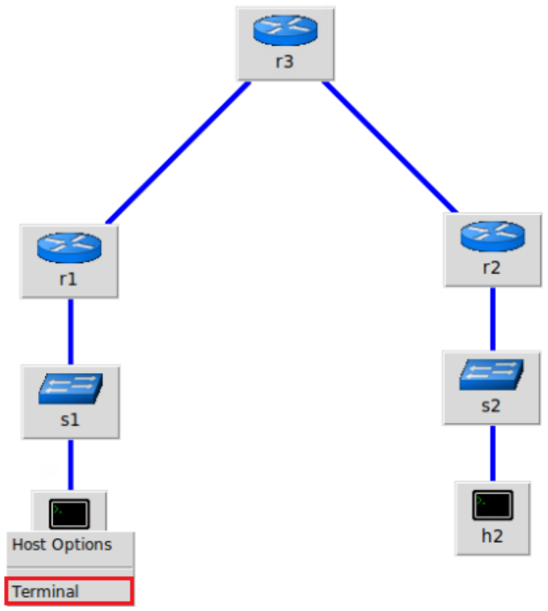
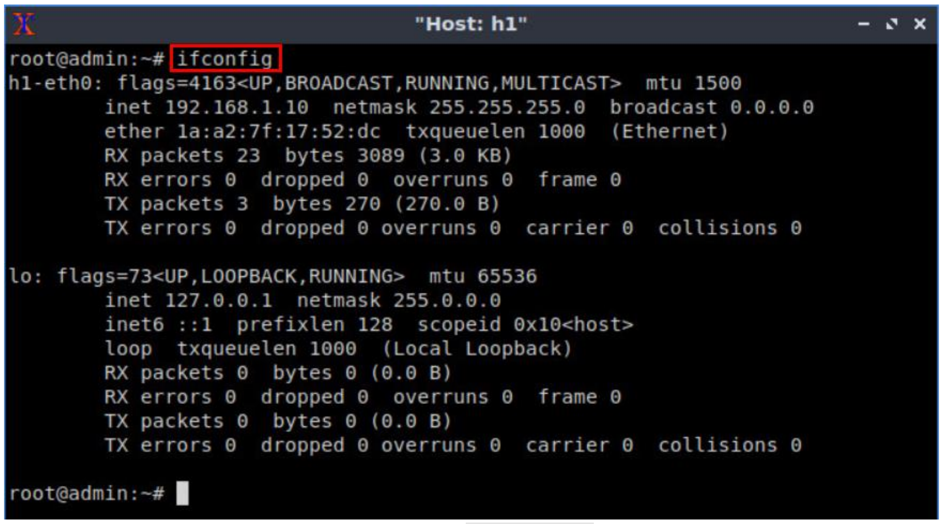


Figure 14. Opening host h1's terminal.

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

```
ifconfig
```

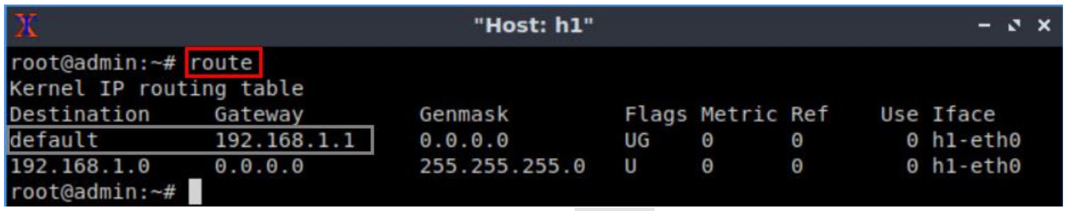


```
root@admin:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.10  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 1a:a2:7f:17:52:dc  txqueuelen 1000  (Ethernet)
        RX packets 23  bytes 3089 (3.0 KB)
        RX errors 0  dropped 0  overruns 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:~# 
```

Figure 15. Output of `ifconfig` command.

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

```
route
```



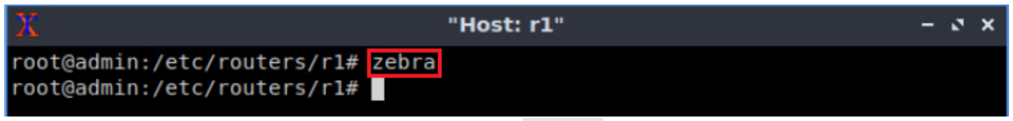
```
root@admin:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.1.1     0.0.0.0         UG    0      0        0 h1-eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 h1-eth0
root@admin:~#
```

Figure 16. Output of the `route` command.

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

**Step 5.** 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. In order to start the zebra, type the following command:

```
zebra
```

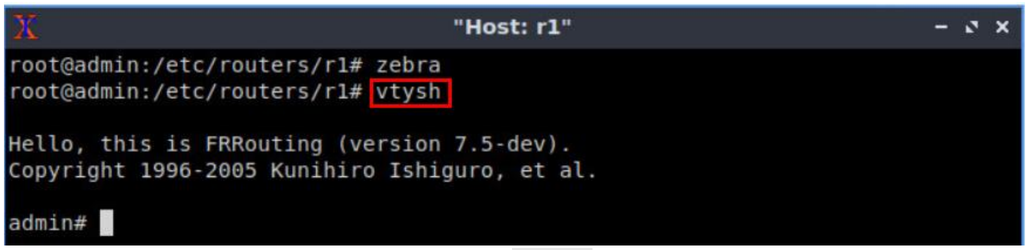


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1#
```

Figure 17. Starting `zebra` daemon.

**Step 6**. 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
```

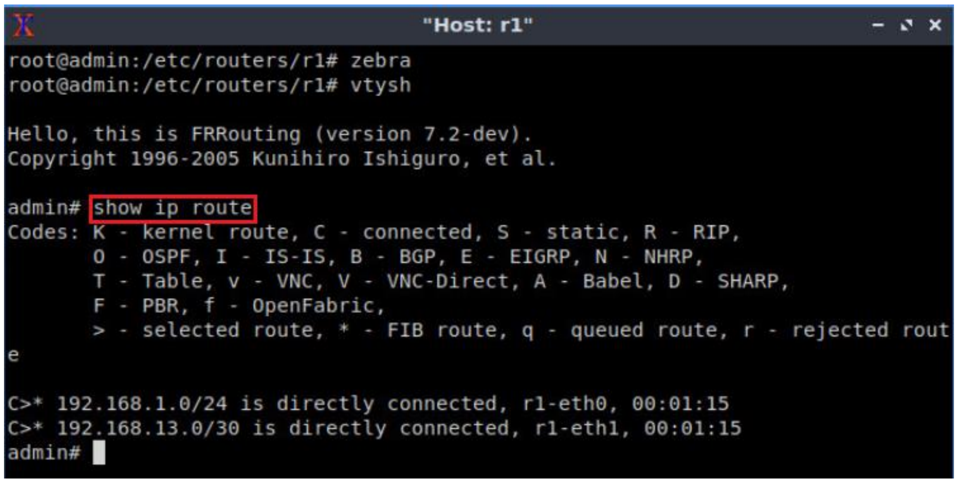


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.5-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin#
```

Figure 18. Starting `vtysh` in router r1.

**Step 7.** 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 (192.168.2.0/24) as there is no routing protocol configured yet.

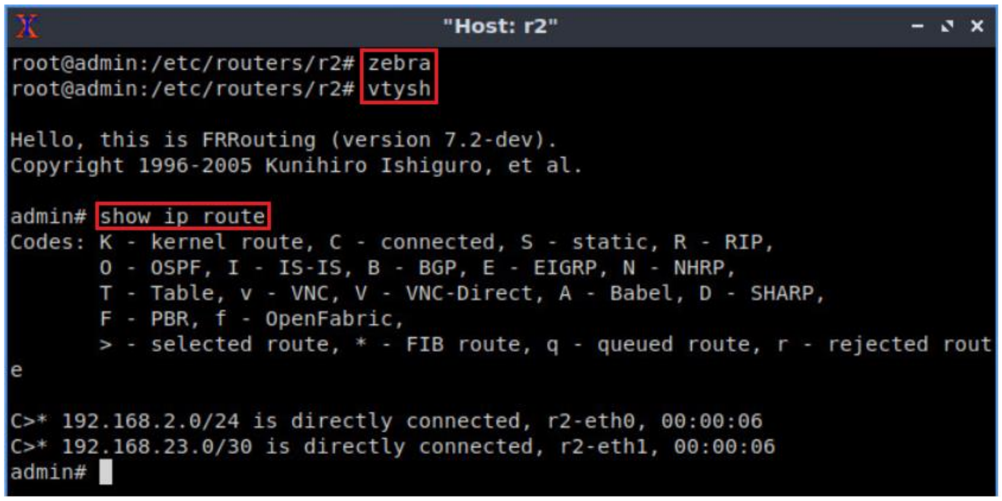
```
show ip route
```



```
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# 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 rout
e

C>* 192.168.1.0/24 is directly connected, r1-eth0, 00:01:15
C>* 192.168.13.0/30 is directly connected, r1-eth1, 00:01:15
admin#
```

Figure 19. Displaying routing table of router r1.

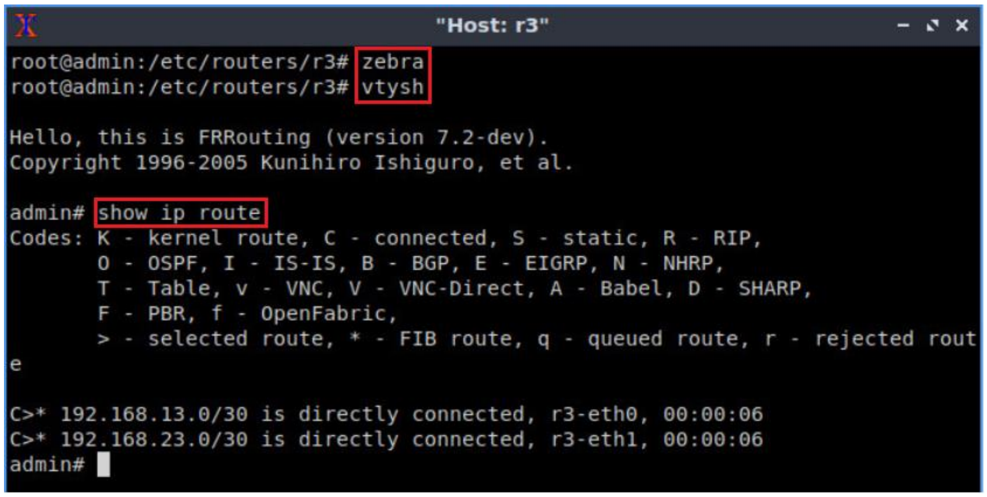
**Step 8.** Router r2 is configured similarly to router r1 but, with different IP addresses (see Table 2). Those steps are summarized in the following figure. To proceed, in router r2's terminal, issue the commands depicted below. At the end, you will verify all the directly connected networks of router r2.



```
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,
       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 rout
e

C>* 192.168.2.0/24 is directly connected, r2-eth0, 00:00:06
C>* 192.168.23.0/30 is directly connected, r2-eth1, 00:00:06
admin#
```

Figure 20. Displaying routing table of router r2.

**Step 9.** 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.



```
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,
       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 rout
e

C>* 192.168.13.0/30 is directly connected, r3-eth0, 00:00:06
C>* 192.168.23.0/30 is directly connected, r3-eth1, 00:00:06
admin# 
```

Figure 21. Displaying routing table of router r3.

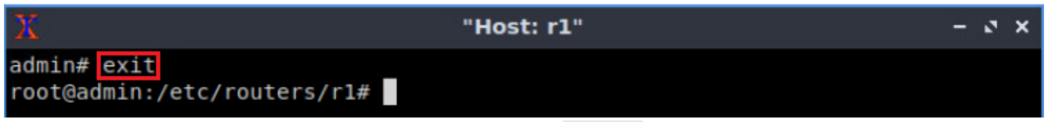
## 3 Configuring static MPLS from router r1 to router r2

In this section, you will configure static MPLS in routers. Router r1 will push a label to router r3. Routers r3 will swap label and the data packet will reach to router r2. Router r2 will pop the label and the data packet will be delivered to the destination host h2.

### 3.1 Push labels

**Step 1.** At this point, router r1 can reach the directly connected network 192.168.13.0/30. To communicate with other networks, you will configure static routing in router r1. To configure static routing, you need to enable the static daemon first. In router r1, type the following command to exit the `vtysh` session:

```
exit
```

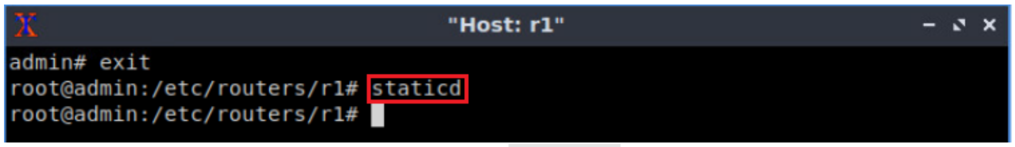


```
admin# exit
root@admin:/etc/routers/r1# 
```

Figure 22. Exiting the `vtysh` session.

**Step 2.** Type the following command to enable the static routing daemon in router r1.

```
staticd
```

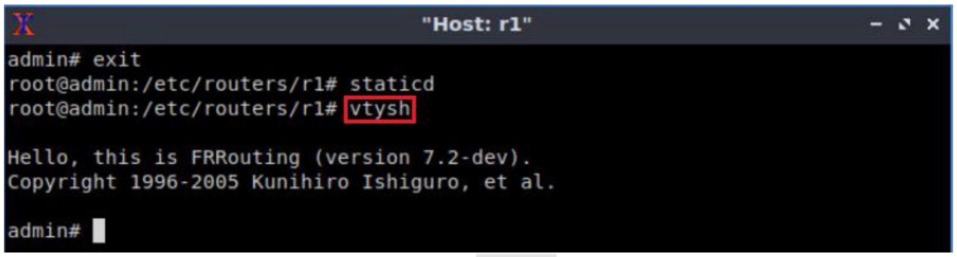


```
admin# exit
root@admin:/etc/routers/r1# staticd
root@admin:/etc/routers/r1# 
```

Figure 23. Starting `staticd` daemon.

**Step 3.** In order to enter router r1's terminal, issue the following command:

```
vtysh
```

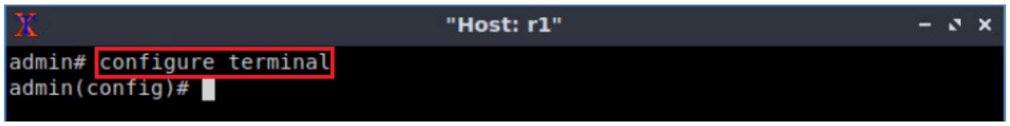


```
admin# exit
root@admin:/etc/routers/r1# staticd
root@admin:/etc/routers/r1# vtysh

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

admin#
```

Figure 24. Starting `vtysh` in router r1.

**Step 4.** To enable router r1's configuration mode, issue the following command:

```
configure terminal
```

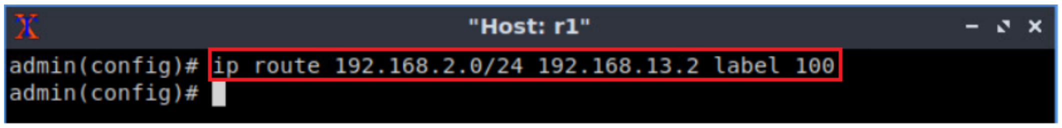


```
admin# configure terminal
admin(config)#
```

Figure 25. Enabling configuration mode in router r1.

**Step 5.** Type the following command to configure a static route to the network 192.168.2.0/24. Router r1 will assign label 100 to forward traffic for the destination network 192.168.2.0/24 via 192.168.13.2.

```
ip route 192.168.2.0/24 192.168.13.2 label 100
```

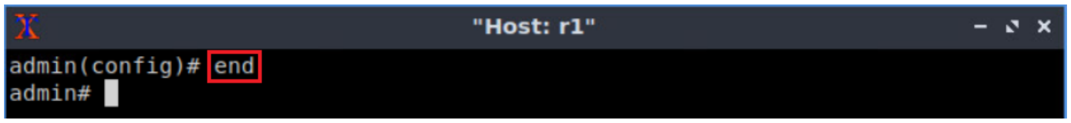


```
admin(config)# ip route 192.168.2.0/24 192.168.13.2 label 100
admin(config)#
```

Figure 26. Pushing label for network 192.168.2.0/24.

**Step 6.** Type the following command to exit from the configuration mode.

```
end
```

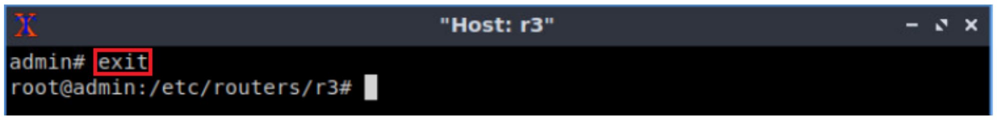


```
admin(config)# end
admin#
```

Figure 27. Exiting from configuration mode.

**Step 7.** In router r3 terminal, type the following command to exit from `vtysh`.

```
exit
```

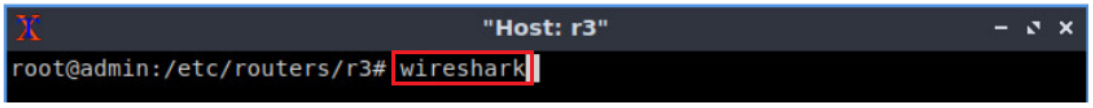


```
admin# exit
root@admin:/etc/routers/r3#
```

Figure 28. Exiting from vtysh.

**Step 8.** Type the following command to start Wireshark packet analyzer. A new window will emerge.

```
wireshark
```



```
root@admin:/etc/routers/r3# wireshark
```

Figure 29. Starting Wireshark packet analyzer.

**Step 9.** Select interface *r3-eth0* and click on the icon located on the upper left-hand side to start capturing packets.

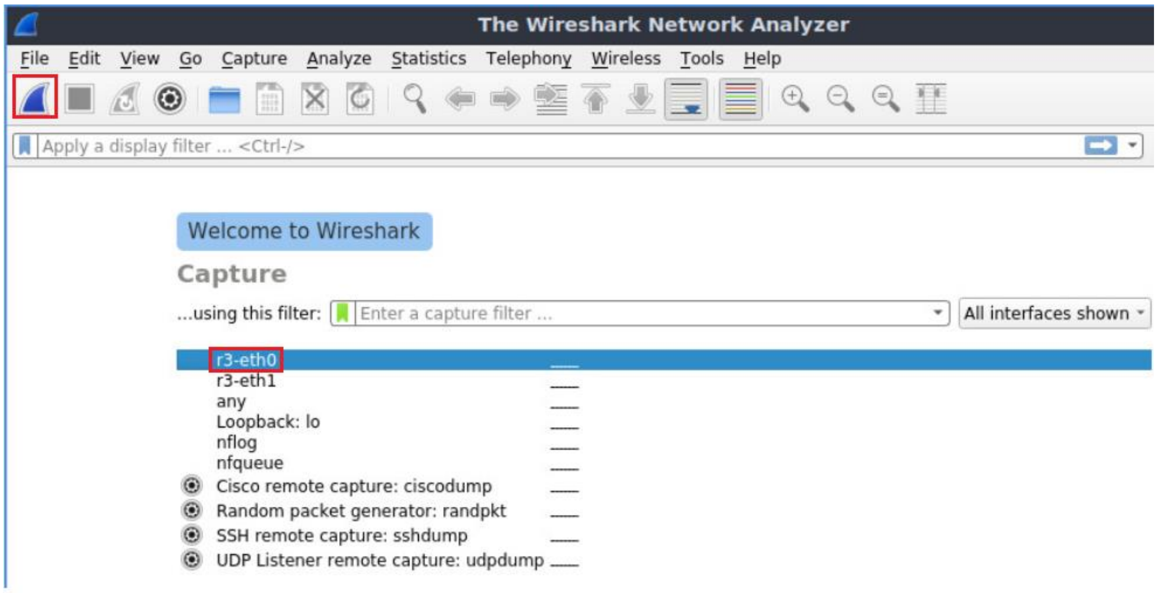
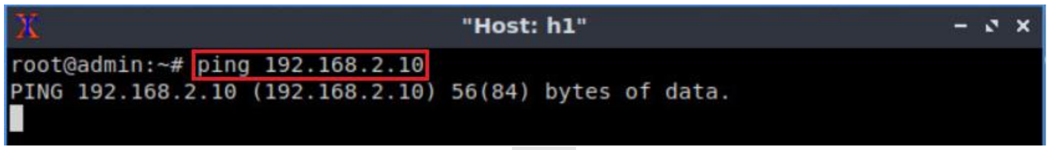

The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
All interfaces shown
r3-eth0
r3-eth1
any
Loopback: lo
nflog
nfqueue
Cisco remote capture: ciscodump
Random packet generator: randpkt
SSH remote capture: sshdump
UDP Listener remote capture: udpdump

Figure 30. Starting packet capture.

**Step 10.** Test the connectivity between hosts h1 and h2 using the `ping` command. In host h1, type the command specified below.

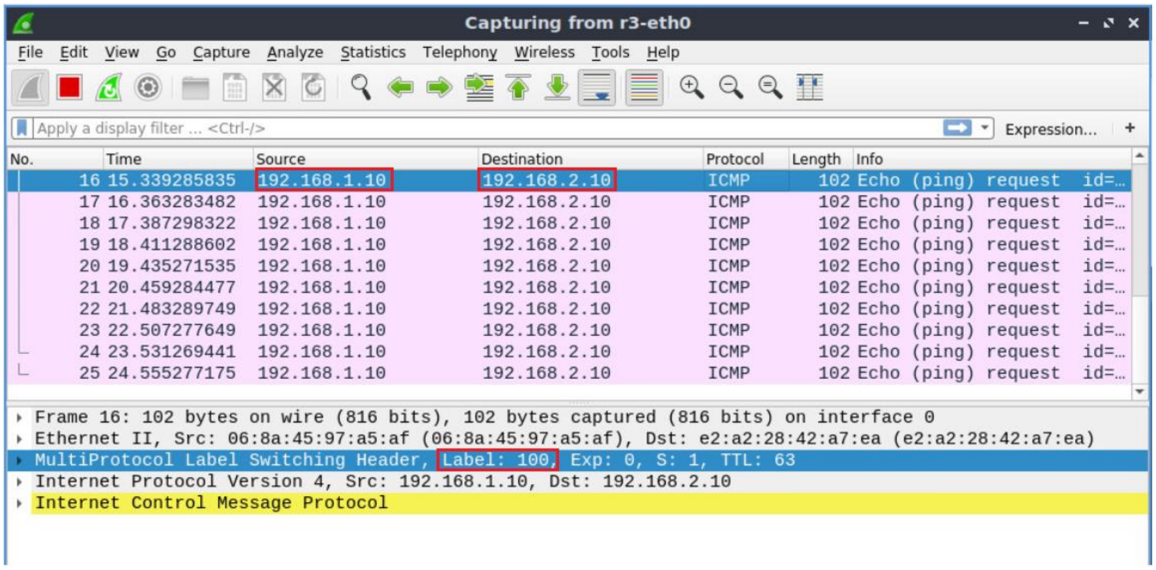
```
ping 192.168.2.10
```



```
root@admin:~# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
```

Figure 31. Output of the `ping` command in host h1.

**Step 11.** In Wireshark, click on any ICMP packet to see the MPLS label.


Capturing from r3-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.  Time            Source        Destination   Protocol Length Info
  16 15.339285835    192.168.1.10  192.168.2.10  ICMP     102 Echo (ping) request  id=…
  17 16.363283482    192.168.1.10  192.168.2.10  ICMP     102 Echo (ping) request  id=…
  18 17.387298322    192.168.1.10  192.168.2.10  ICMP     102 Echo (ping) request  id=…
  19 18.411288602    192.168.1.10  192.168.2.10  ICMP     102 Echo (ping) request  id=…
  20 19.435271535    192.168.1.10  192.168.2.10  ICMP     102 Echo (ping) request  id=…
  21 20.459284477    192.168.1.10  192.168.2.10  ICMP     102 Echo (ping) request  id=…
  22 21.483289749    192.168.1.10  192.168.2.10  ICMP     102 Echo (ping) request  id=…
  23 22.507277649    192.168.1.10  192.168.2.10  ICMP     102 Echo (ping) request  id=…
  24 23.531269441    192.168.1.10  192.168.2.10  ICMP     102 Echo (ping) request  id=…
  25 24.555277175    192.168.1.10  192.168.2.10  ICMP     102 Echo (ping) request  id=…

▸ Frame 16: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface 0
▸ Ethernet II, Src: 06:8a:45:97:a5:af (06:8a:45:97:a5:af), Dst: e2:a2:28:42:a7:ea (e2:a2:28:42:a7:ea)
▸ MultiProtocol Label Switching Header, Label: 100, Exp: 0, S: 1, TTL: 63
▸ Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.2.10
▸ Internet Control Message Protocol
```

Figure 32. Verifying MPLS label.

Consider the figure above. You will notice that MPLS label 100 has been assigned. Router r1 uses label 100 to forward traffic to interface r3-eth0 (192.168.13.2).

Close Wireshark after verifying the label. Select '*Stop and Quit without saving*' option for unsaved packets.

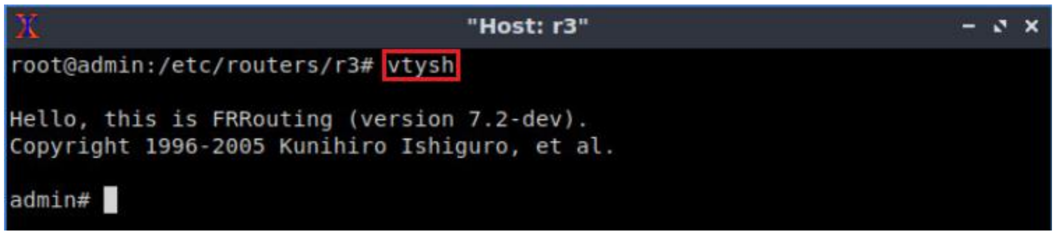
**Step 12.** In host h1's terminal, press `Ctrl+c` to stop the test.

## 3.2 Swap labels

At this point, router r1 will use label 100 to reach router r3. Router r3 will swap the label with a new label to forward the packet.

**Step 1.** In router r3 terminal, type the following command to enable vtysh:

```
vtysh
```

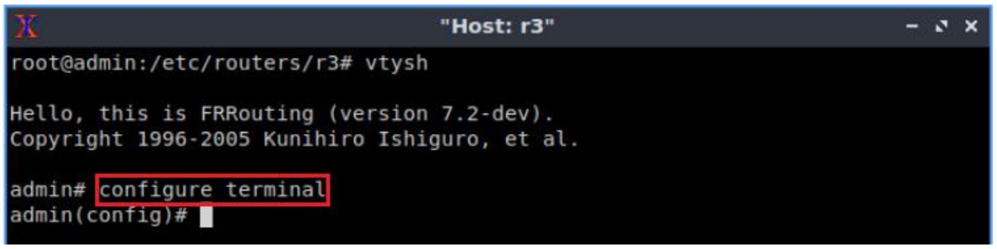


```
root@admin:/etc/routers/r3# vtysh

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

admin#
```

Figure 33. Enabling vtysh in router r3.

**Step 2.** To enable configuration mode in router r3, issue the following command:

```
configure terminal
```

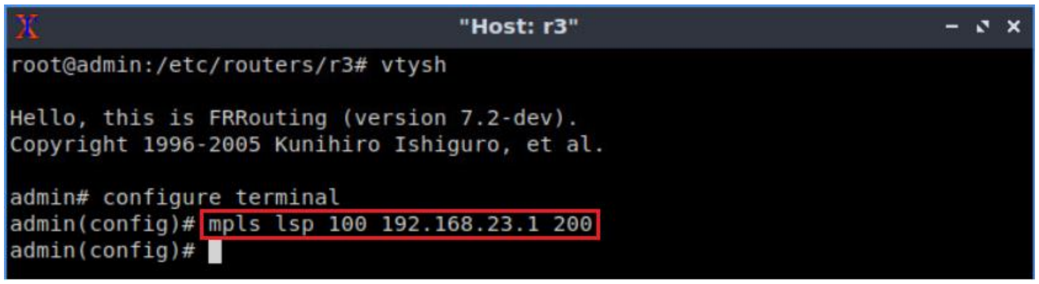


```
root@admin:/etc/routers/r3# vtysh

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

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

Figure 34. Enabling configuration mode in router r3.

**Step 3.** In this step, you will configure label swapping in router r3. Router r3 will receive label 100 from router r1 and swap the label with label 200 to forward the traffic to the interface *r2-eth1* (192.168.23.1). Type the following command to enable label swapping in router r3.

```
mpls lsp 100 192.168.23.1 200
```

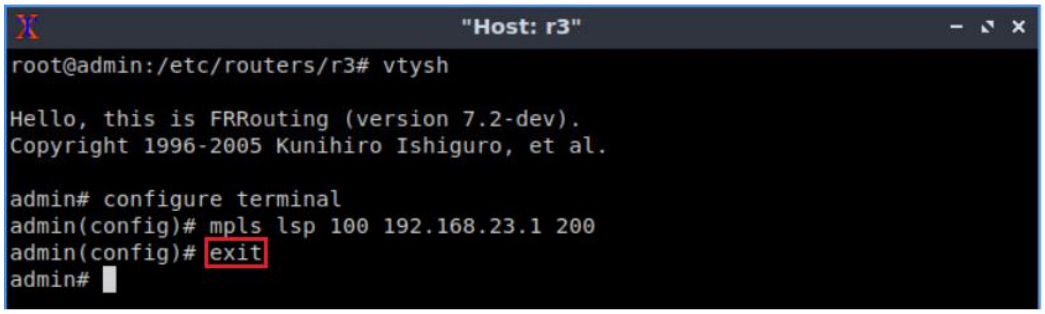


```
root@admin:/etc/routers/r3# vtysh

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

admin# configure terminal
admin(config)# mpls lsp 100 192.168.23.1 200
admin(config)#
```

Figure 35. Enabling label swapping in router r3.

**Step 4.** Type the following command to exit from the configuration mode.

```
exit
```



```
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)# mpls lsp 100 192.168.23.1 200
admin(config)# exit
admin#
```

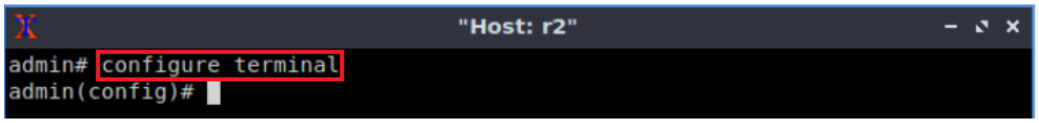
Figure 36. Exiting from configuration mode.

## 3.3 Pop labels

At this point, router r3 will use label 200 to reach router r2. Router r2 will pop the label and the IP packet will be delivered to the destination host h2.

**Step 1.** To enable configuration mode in router r2, issue the following command:

```
configure terminal
```

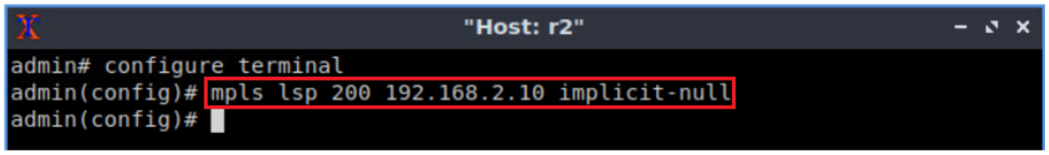


```
"Host: r2"
admin# configure terminal
admin(config)#
```

Figure 37. Enabling configuration mode in router r2.

**Step 2.** Issue the following command in router r2. Router r2 will pop the label (200) and the IP packet will be delivered to the destination host, h2 (192.168.2.10).

```
mpls lsp 200 192.168.2.10 implicit-null
```



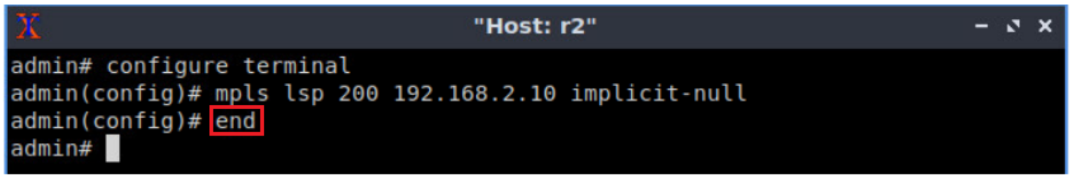
```
"Host: r2"
admin# configure terminal
admin(config)# mpls lsp 200 192.168.2.10 implicit-null
admin(config)#
```

Figure 38. Enabling label popping in router r2.

The keyword `implicit-null` indicates that the router will perform a pop and the IP packet will be delivered to the destination.

**Step 3.** Type the following command to exit from the configuration mode.

```
end
```

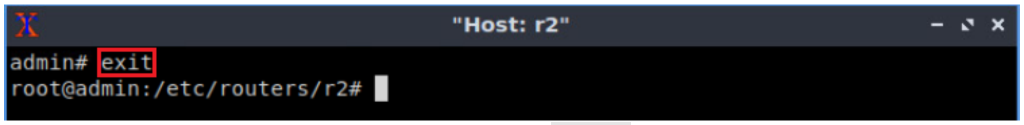


```
"Host: r2"
admin# configure terminal
admin(config)# mpls lsp 200 192.168.2.10 implicit-null
admin(config)# end
admin#
```

Figure 39. Exiting configuration mode.

**Step 4.** Type the following command to exit the `vtysh` session:

```
exit
```

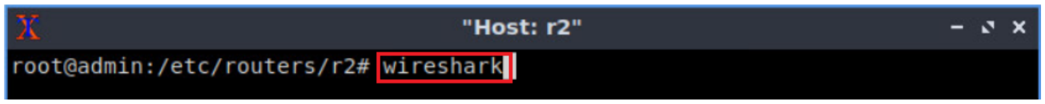


```
"Host: r2"
admin# exit
root@admin:/etc/routers/r2#
```

Figure 40. Exiting the `vtysh` session.

**Step 5.** Type the following command to start Wireshark packet analyzer. A new window will emerge.

```
wireshark
```



```
"Host: r2"
root@admin:/etc/routers/r2# wireshark
```

Figure 41. Starting Wireshark packet analyzer.

**Step 6.** Select interface *r2-eth0* and click on the icon located on the upper left-hand side to start capturing packets.

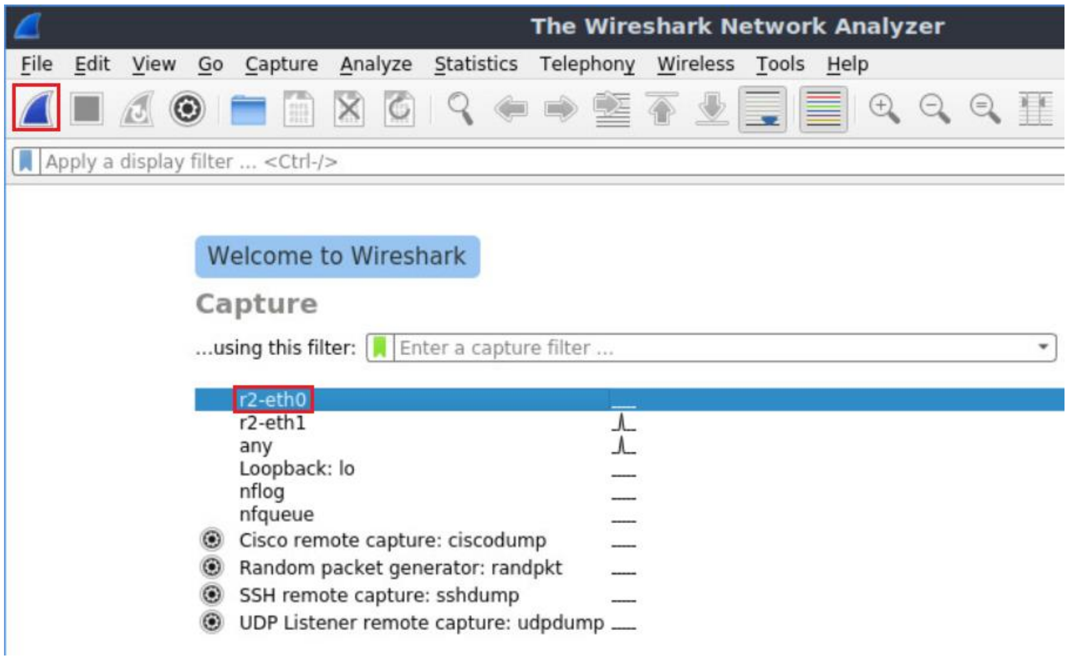
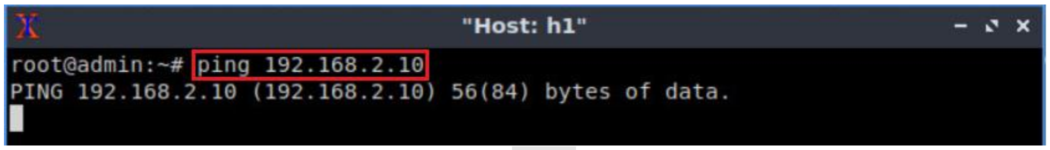

The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
r2-eth0
r2-eth1
any
Loopback: lo
nflog
nfqueue
Cisco remote capture: ciscodump
Random packet generator: randpkt
SSH remote capture: sshdump
UDP Listener remote capture: udpdump

Figure 42. Starting packet capture.

**Step 7.** Test the connectivity between hosts h1 and h2 using the `ping` command. In host h1, type the command specified below.

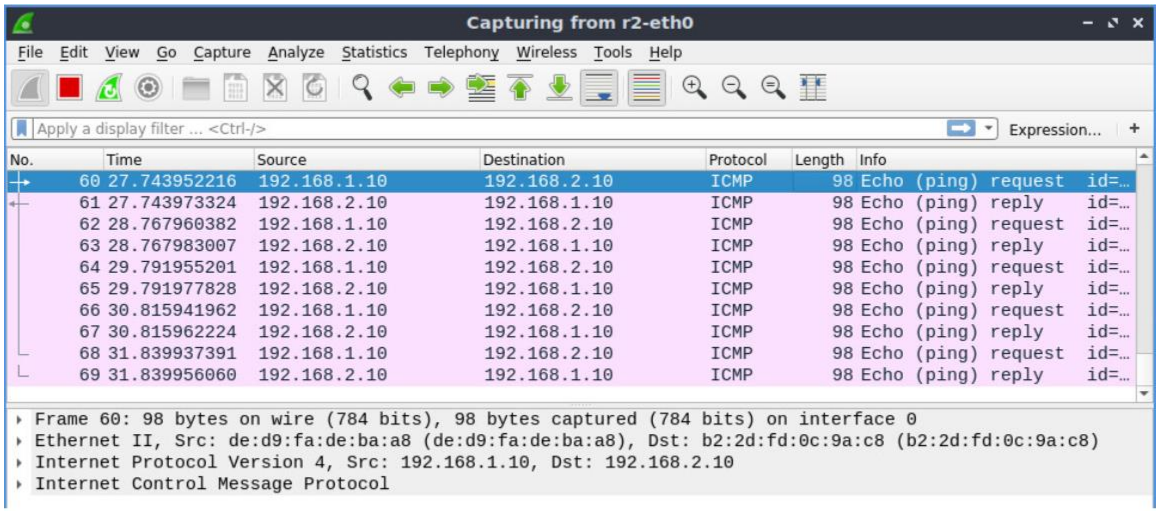
```
ping 192.168.2.10
```



```
root@admin:~# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
```

Figure 43. Output of the `ping` command in host h1.

**Step 8.** Click on any of the ICMP packets and verify MPLS label in Wireshark.


Capturing from r2-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.   Time          Source         Destination    Protocol Length Info
   60 27.743952216  192.168.1.10   192.168.2.10   ICMP        98 Echo (ping) request  id=…
   61 27.743973324  192.168.2.10   192.168.1.10   ICMP        98 Echo (ping) reply    id=…
   62 28.767960382  192.168.1.10   192.168.2.10   ICMP        98 Echo (ping) request  id=…
   63 28.767983007  192.168.2.10   192.168.1.10   ICMP        98 Echo (ping) reply    id=…
   64 29.791955201  192.168.1.10   192.168.2.10   ICMP        98 Echo (ping) request  id=…
   65 29.791977828  192.168.2.10   192.168.1.10   ICMP        98 Echo (ping) reply    id=…
   66 30.815941962  192.168.1.10   192.168.2.10   ICMP        98 Echo (ping) request  id=…
   67 30.815962224  192.168.2.10   192.168.1.10   ICMP        98 Echo (ping) reply    id=…
   68 31.839937391  192.168.1.10   192.168.2.10   ICMP        98 Echo (ping) request  id=…
   69 31.839956060  192.168.2.10   192.168.1.10   ICMP        98 Echo (ping) reply    id=…

▸ Frame 60: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▸ Ethernet II, Src: de:d9:fa:de:ba:a8 (de:d9:fa:de:ba:a8), Dst: b2:2d:fd:0c:9a:c8 (b2:2d:fd:0c:9a:c8)
▸ Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.2.10
▸ Internet Control Message Protocol
```

Figure 44. Verifying MPLS label.

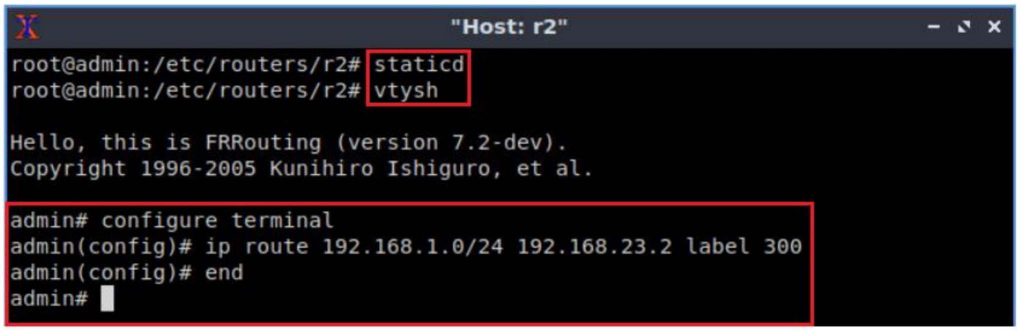
Consider the figure above. You will notice there is no MPLS label attached to the IP packet as router r2 popped the label and delivered the IP packet to its destination. You will notice that host h2 (192.168.2.10) is generating a reply for the destination host h1 (192.168.1.10) but router r2 does not have a route back to router r1.

**Step 9.** In host h1's terminal, press `Ctrl+c` to stop the test and close Wireshark.

## 4 Configuring static MPLS from router r2 to router r1

In this section, you will configure static MPLS from router r2 to router r1 so that both the hosts, h1 and h2 can ping each other.

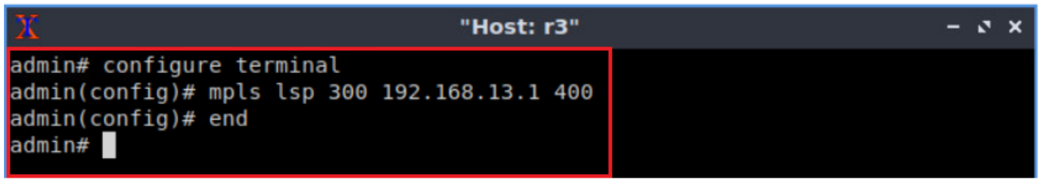
**Step 1.** Type the following command to create a static route for the network 192.168.1.0/24. Assign label 300 for the next hop 192.168.23.2. The necessary steps are summarized in the following figure.



```
root@admin:/etc/routers/r2# staticd
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)# ip route 192.168.1.0/24 192.168.23.2 label 300
admin(config)# end
admin#
```

Figure 45. Pushing labels in router r2.

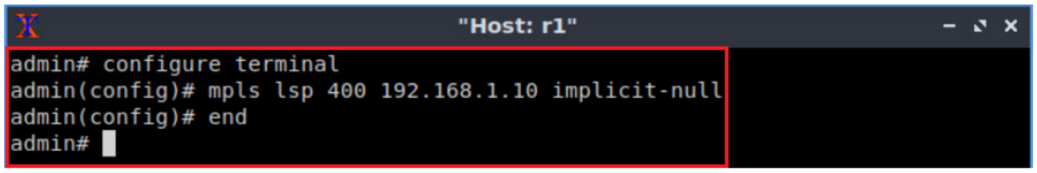
**Step 2.** Type the following commands in router r3. Router r3 will receive label 300, swap the label with 400 and the packet will be delivered to router r1 (192.168.13.1). All the commands are summarized in the following figure.



```
admin# configure terminal
admin(config)# mpls lsp 300 192.168.13.1 400
admin(config)# end
admin#
```

Figure 46. Enabling label swapping in router r3.

**Step 3.** Type the following commands to pop the label in router r1 which was received from router r3. Router r1 will pop the label (400) and the IP packet will be delivered to the destination host, h1 (192.168.1.10).



```
admin# configure terminal
admin(config)# mpls lsp 400 192.168.1.10 implicit-null
admin(config)# end
admin#
```

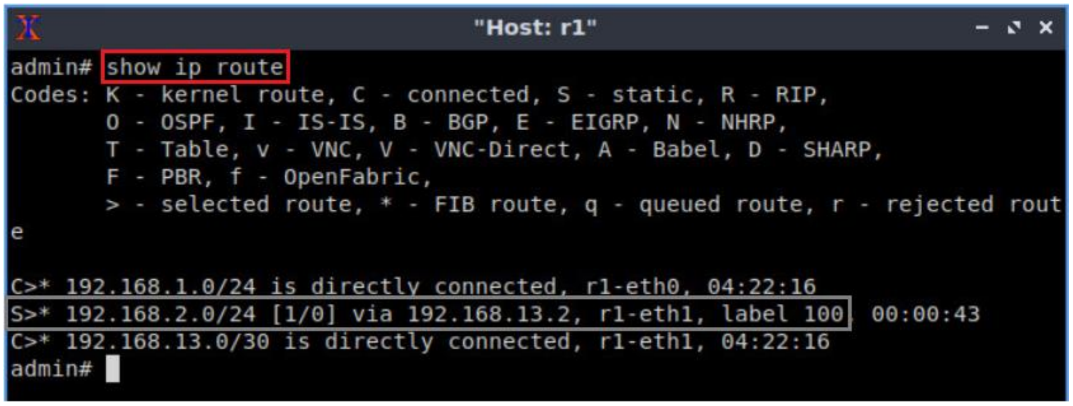
Figure 47. Popping label in router r1.

## 5 Verifying the configuration

In this section, you will verify the MPLS table and the connectivity between the two hosts.

**Step 1.** Type the following command to verify the routing table in router r1. You will notice static routes and the labels assigned to router r1.

```
show ip route
```

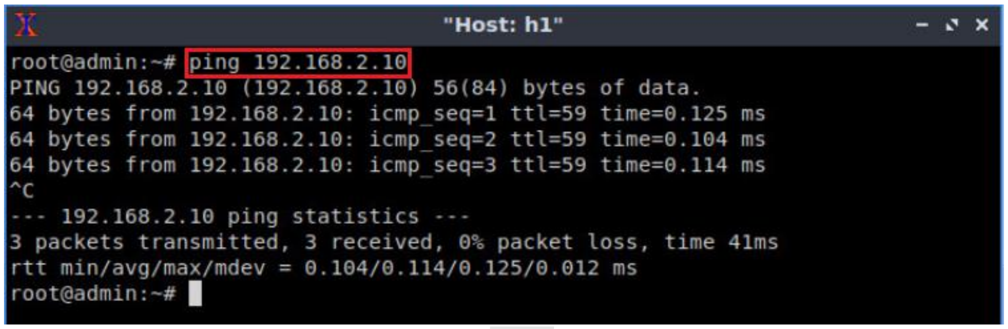


```
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected rout
e

C>* 192.168.1.0/24 is directly connected, r1-eth0, 04:22:16
S>* 192.168.2.0/24 [1/0] via 192.168.13.2, r1-eth1, label 100, 00:00:43
C>* 192.168.13.0/30 is directly connected, r1-eth1, 04:22:16
admin#
```

Figure 48. Verifying routing table in router r1.

**Step 2**. In host h1, type the following command to test the connectivity between host h1 and host h2 using the `ping` command. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test.

```
ping 192.168.2.10
```

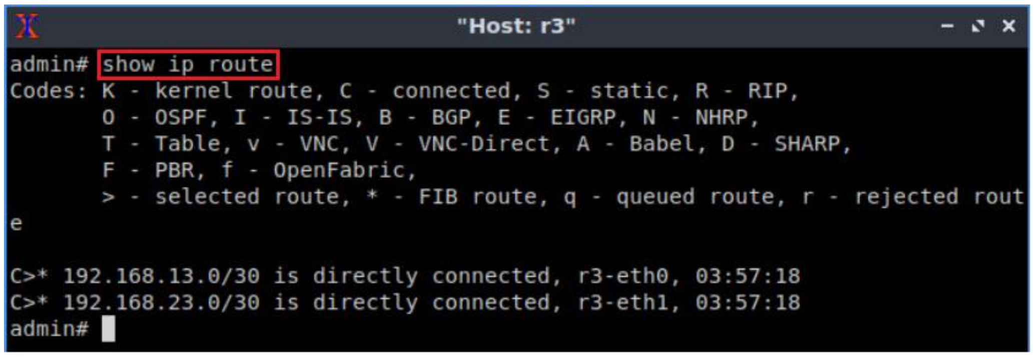


```
root@admin:~# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
64 bytes from 192.168.2.10: icmp_seq=1 ttl=59 time=0.125 ms
64 bytes from 192.168.2.10: icmp_seq=2 ttl=59 time=0.104 ms
64 bytes from 192.168.2.10: icmp_seq=3 ttl=59 time=0.114 ms
^C
--- 192.168.2.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 41ms
rtt min/avg/max/mdev = 0.104/0.114/0.125/0.012 ms
root@admin:~#
```

Figure 49. Output of the `ping` command in host h2.

**Step 3.** Type the following command to verify routing table in router r3.

```
show ip route
```



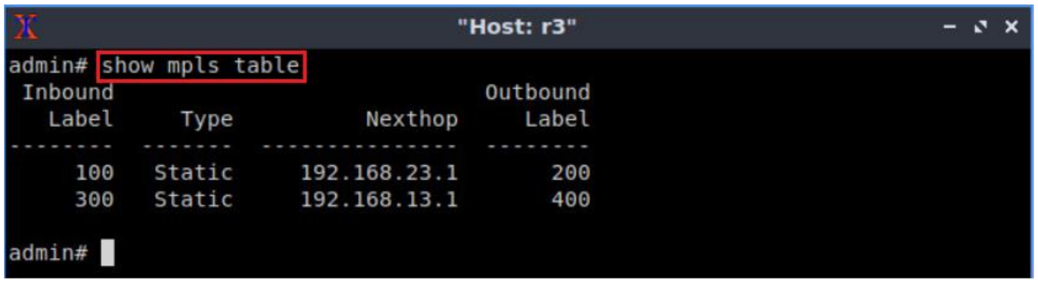
```
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected rout
e

C>* 192.168.13.0/30 is directly connected, r3-eth0, 03:57:18
C>* 192.168.23.0/30 is directly connected, r3-eth1, 03:57:18
admin#
```

Figure 50. Verifying routing table in router r3.

Consider the figure above. Router r3 only knows about directly connected networks. Hosts h1 and h2 can communicate with each other since routers r3 uses MPLS labels to perform packet forwarding.

**Step 4.** Type the following command to verify the MPLS table in router r3.

```
show mpls table
```



```
admin# show mpls table
 Inbound                            Outbound
   Label     Type          Nexthop     Label
 --------  -------  ---------------  --------
     100   Static     192.168.23.1       200
     300   Static     192.168.13.1       400

admin#
```

Figure 51. Verifying MPLS table in router r3.

Consider the figure above. Router r3 forwards packets based on the labels only. If the router receives packet with label 100, it will forward the traffic to router r2 (192.168.23.1), if the receiving label is 300, the nexthop is 192.168.13.1.

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

## References

1. Luc De Ghein, "*MPLS fundamentals*", 1st Edition, Pearson. 2006.
2. Juniper, "*MPLS label distribution protocols overview*", 2018. [Online]. Available: https://www.juniper.net/documentation/en_US/junose15.1/topics/concept/mpls-label-distribution-protocols.html
3. Greek University, "*Static route*", [Online]. Available: https://geek-university.com/ccna/static-routes/#:~:text=Static%20routes%20are%20manually%20added,to%20one%20of%20its%20interfaces.

4. Lydia Parziale, David T. Britt, Chuck Davis, Jason Forrester, Wei Liu, Carolyn Matthews, Nicolas Rosselot, "*TCP/IP tutorial and technical overview*", 8th Edition, Pearson, 2006.
5. Linux foundation collaborative projects, "*FRR routing documentation*", 2017. [Online]. Available: http://docs.frrouting.org/en/latest/zebra.html#mpls-commands
6. Juniper, "*Understanding MPLS label operations*", 2020. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/concept/mpls-label-operations-qfx-series.html

UNIVERSITY OF
SOUTH CAROLINA

# SOFTWARE DEFINED NETWORKING

# Lab 4: Introduction to SDN

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

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

## Contents

## Overview

This lab is an introduction to Software Defined Networking (SDN), a new networking paradigm that overcomes several limitations of the current network infrastructure. In this lab, we will introduce the components of SDN and describe how they operate. The focus in this lab is to gain in-depth knowledge about the role of the control plane and the data plane.

## Objectives

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

1. Understand the concept of SDN.
2. Enable the ONOS controller.
3. Navigate through the ONOS environment.
4. Activate basic ONOS applications and understand their effects.
5. Understand flow tables in SDN devices.
6. Perform a connectivity test.
7. Visualize topology information in the GUI dashboard.

## Lab settings

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

Table 1**.** Credentials to access the 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: Starting ONOS
4. Section 4: Navigating the ONOS GUI.

## 1 Introduction

In just a few years, SDN has created enormous interest in academia and industry. An open, vendor-neutral, control-data plane interface such as OpenFlow[1] allows network hardware and software to evolve independently. Furthermore, it facilitates the replacement of expensive, proprietary hardware and firmware with commodity hardware and a free, open-source Network Operating System (NOS). By managing network resources and providing high-level abstractions and APIs for interacting with, managing, monitoring, and programming network switches, the NOS provides an open platform that simplifies the creation of innovative and beneficial network applications and services that work across a wide range of hardware[2].

## 1.1 Introduction to SDN

Traditional IP networks depend on distributed routing protocols running inside the routers to exchange routing information. Despite their widespread adoption, traditional IP networks are complicated by their nature, and they are not trivial to be managed. For example, network administrators need to configure each network separately using low-level, vendor-specific commands. A traditional networking device bundles the control plane (that decides how to handle network traffic) and the data plane (that forward network traffic). Thus, reducing the flexibility and hindering innovation and evolution of the networking infrastructure[3].

SDN is an emerging networking paradigm that gives hope to change the limitations of current network infrastructures[3]. It breaks the control plane from the data plane and implements each separately. The disaggregation of the control plane and the data plane allows network switches to become simple forwarding devices and the control logic to be implemented in a logically centralized controller. Thus, amplifying the flexibility in the network, breaking the network control problem into tractable pieces, introducing new abstractions, and facilitating network evolution and innovation[3].

Consider Figure 1. the vast majority of packets handled by the switch are only managed by the data plane. The latter is composed of ports used to receive and transmit the packets, as well as a forwarding table that instructs the switch on how to deal with incoming packets. The data plane is responsible for packet buffering, packet scheduling, header modification, and forwarding[4].

Some packets cannot be processed by the data plane directly, for example, their information is not yet inserted in the forwarding table. Such packets are forwarded to the control plane, which lies on top of the data plane. The control plane involves many activities, mainly to maintain the forwarding table of the data plane. Essentially, the control plane is responsible for processing different control protocols that may affect the forwarding table.

SDN applications run on top of the controller. They are ultimately responsible for managing the flow table on the network devices (data plane). For example, route packets through the best path between two endpoints, or balance traffic loads across multiple paths[4].

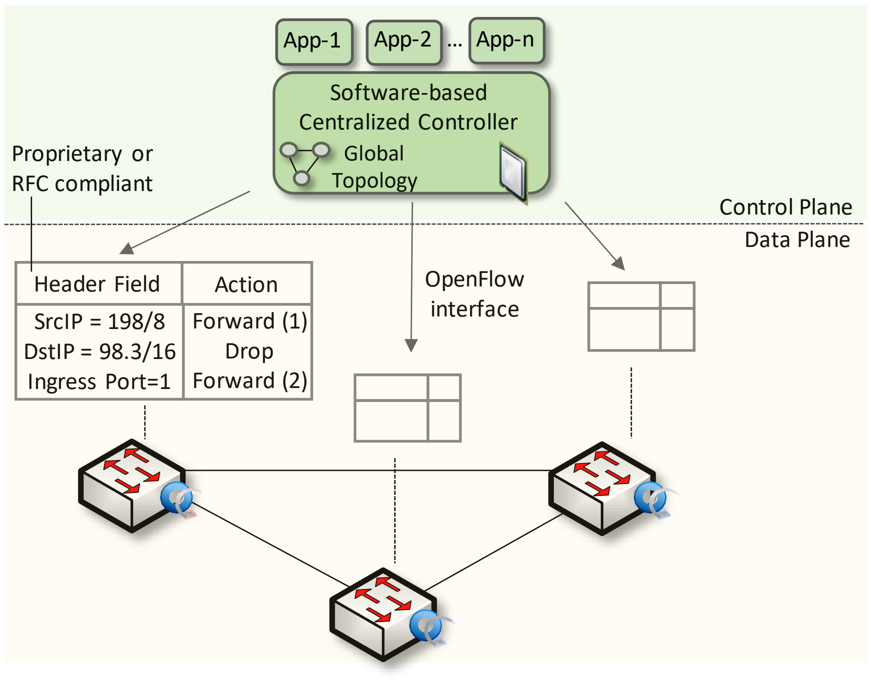


Figure 1. The control plane is embedded in a centralized server, and it is decoupled from data plane devices in SDN networks.

## 1.2 SDN data plane architecture

Consider Figure 2. An SDN device is composed of an Application Programming Interface (API) for communication with the controller, an abstraction layer, and a packet-processing function.

The standard API used to communicate with the controller is OpenFlow[1], or it could be some proprietary alternative in certain SDN solutions[4].

The abstraction layer embodies one or more flow tables, which are the fundamental data structures in an SDN device. These flow tables allow the device to evaluate incoming packets and take the appropriate action. Flow tables consist of a number of flow entries, each of which typically consists of two components: match fields and actions. Match fields are used to compare against incoming packets, for example, matching against the Media Access Control (MAC) address, User Datagram Protocol (UDP), or Transmission Control Protocol (TCP) port. Actions are the instructions that the forwarding device should perform if an incoming packet matches a flow entry. These actions may include forwarding the packet to a specific port, dropping the packet, or flooding the packet on all ports, among others[4].

The packet-processing logic consists of the mechanisms used to take actions based on the results of evaluating incoming packets. In a hardware switch, these mechanisms are implemented by specialized hardware, such as Ternary Content Addressable Memory (TCAM) and Content Addressable Memory (CAM)[4].

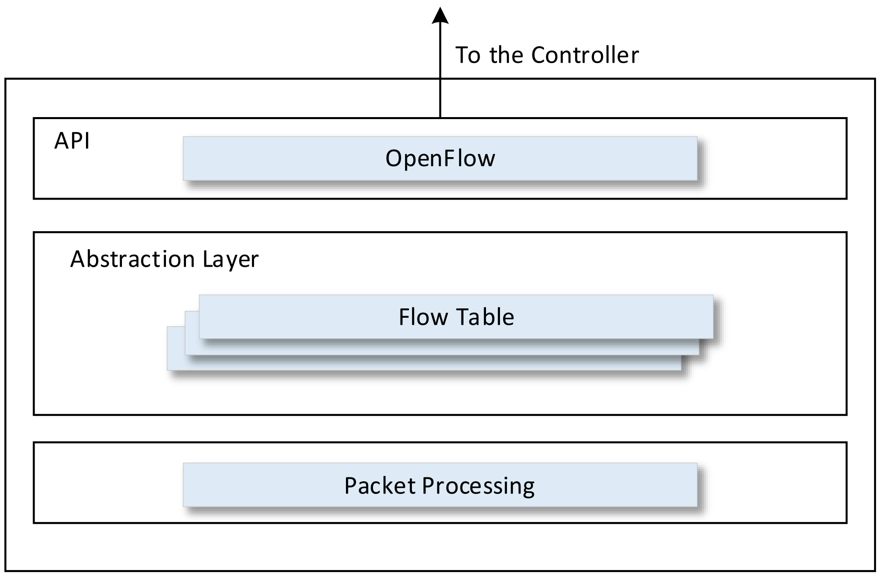


Figure 2. SDN switch architecture.

## 1.3 SDN controller architecture

A controller keeps a view of the entire network, implements policy decisions, controls all the SDN devices that comprise the network infrastructure, and provides a northbound API for the application. Also, controllers implement policy decisions about routing, forwarding, redirecting, and load balancing. Controllers often come with their own set of common application modules, such as a learning switch, a router, a basic firewall, and a simple load balancer. Such features are considered SDN applications, and they are often bundled with the controller[4].

Figure 3 depicts the modules that provide the controller's core functionality, both a northbound and southbound API, and a few sample applications. The southbound API is used to interface with the SDN device. Commonly, this API is OpenFlow. The controller abstracts the details of the SDN controller-to-device protocol so that the applications such as the GUI, learning switch, router, and others can transparently communicate with the SDN devices. Every controller provides core functionality between these raw interfaces. Core features in the controller include[4]:

- End-user device discovery: discovery of end-user devices, such as laptops, desktops, printers, mobile devices, etc.
- Network device discovery: discovery of network devices that comprise the infrastructure of the network, such as switches, routers, and wireless access points.
- Network device topology management: maintain information about the interconnection details of the network device to each other, and to the end-user devices to which they are directly attached.
- Flow management: maintain a database of the flows being managed by the controller and perform all necessary coordination with the devices to ensure synchronization of the device flow entries with that database.

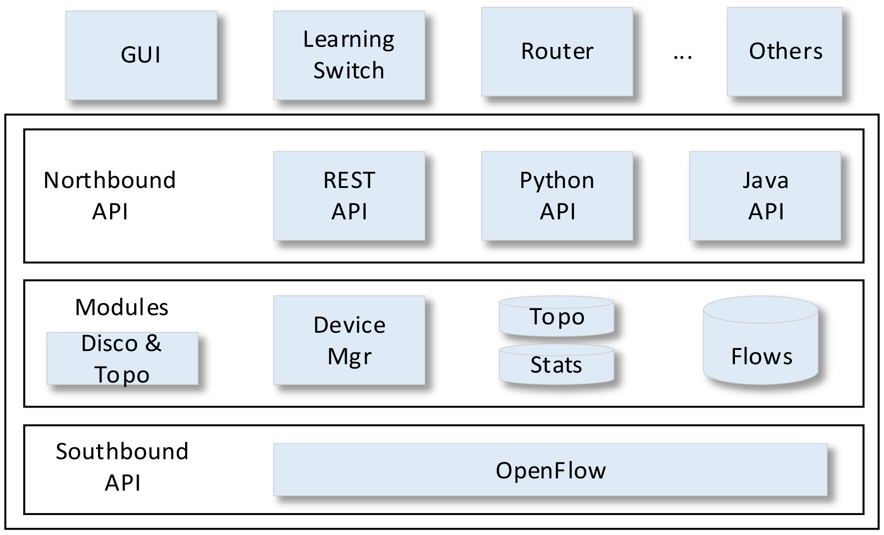


Figure 3. SDN controller architecture.

## 2 Lab topology

Consider Figure 4. The topology consists of four end-hosts, three switches, and a controller. The devices with the blue circle represent OpenFlow switches. All switches are connected to the controller c0.

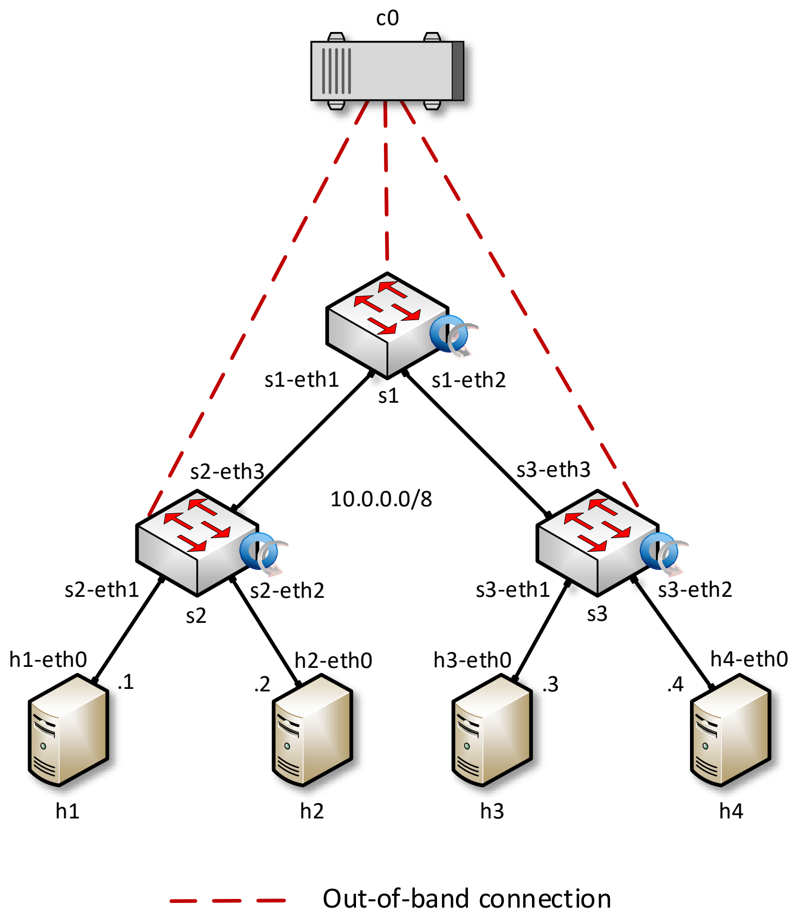


Figure 4. Lab topology.

## 2.1 Lab settings

The devices are already configured according to Table 2.

Table 2. Topology information.

| Device | Interface | MAC Address | IP Address | Subnet |
|---|---|---|---|---|
| h1 | h1-eth0 | 00:00:00:00:00:01 | 10.0.0.1 | /8 |
| h2 | h2-eth0 | 00:00:00:00:00:02 | 10.0.0.2 | /8 |
| h3 | h3-eth0 | 00:00:00:00:00:03 | 10.0.0.3 | /8 |
| h4 | h4-eth0 | 00:00:00:00:00:04 | 10.0.0.4 | /8 |
| c0 | N/A | N/A | 172.17.0.2 | /16 |

## 2.2 Loading the 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[5]. This tool has additional capabilities such as: configuring interface parameters (IP addresses, default gateway), save the topology and export a layer 2 model.

**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`.

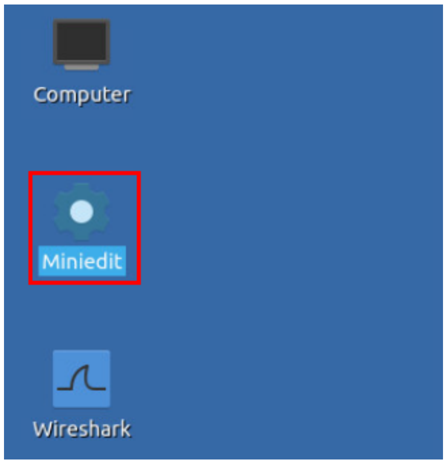


Figure 5. MiniEdit shortcut.

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

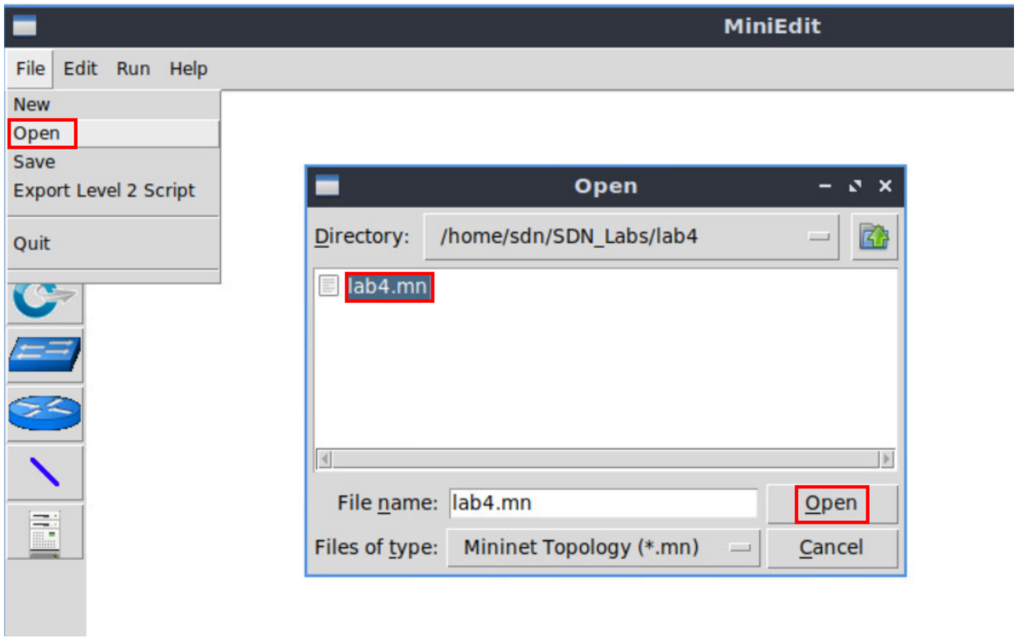


Figure 6. Opening topology.

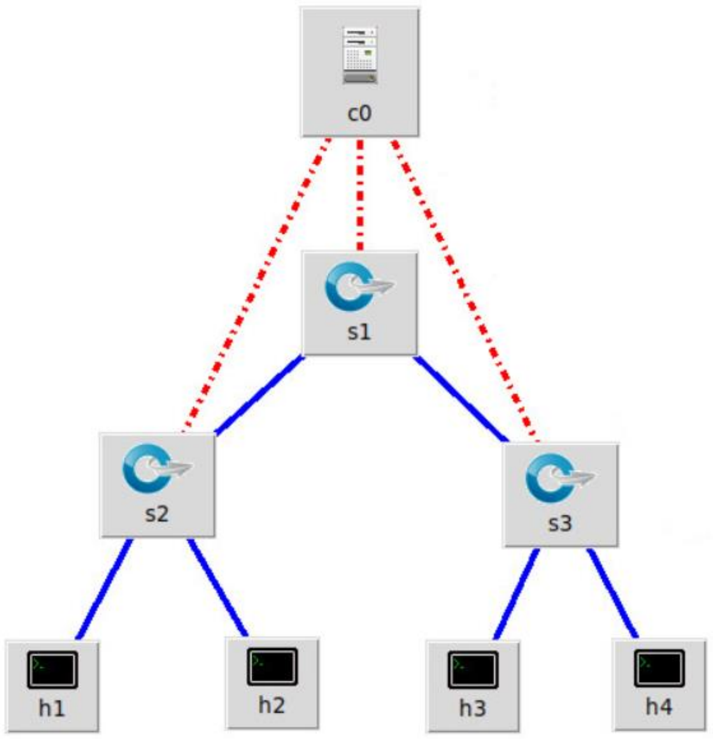


Figure 7. MiniEdit's topology.

**Step 3.** 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.

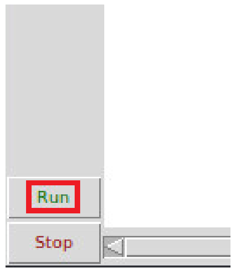


Figure 8. Starting the emulation.

## 3 Starting ONOS

**Step 1.** Open the Linux terminal by clicking on the shortcut depicted below.

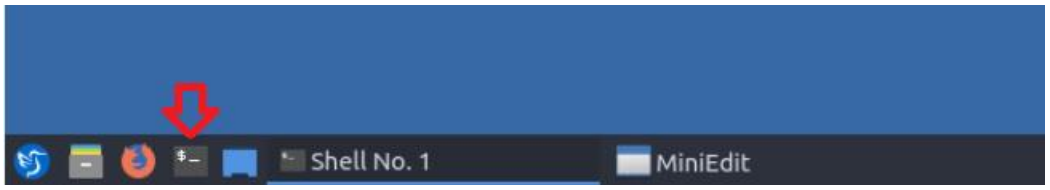
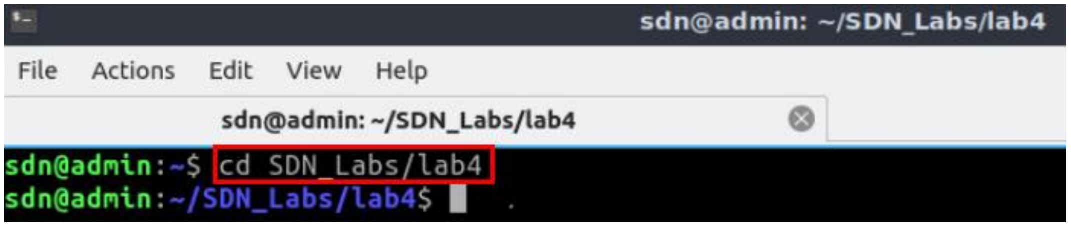


Figure 9. Opening Linux terminal.

**Step 2.** Navigate into *SDN_Labs/lab4* directory by issuing the following command. This folder contains the script responsible for starting ONOS. The `cd` command is short for change directory followed by an argument that specifies the destination directory.

```
cd SDN_Labs/lab4
```

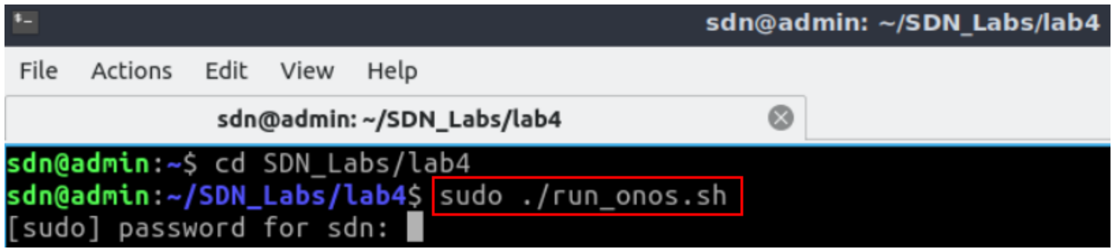

sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
sdn@admin:~$ cd SDN_Labs/lab4
sdn@admin:~/SDN_Labs/lab4$
```

Figure 10. Entering the *SDN_Labs/lab4* directory.

**Step 3.** A script was written to run ONOS and enter its Command Line Interface (CLI). In order to run the script in superuser (root) mode, issue the following command. When prompted for a password, type `password`. In addition to running ONOS, the script will modify the MAC addresses of the hosts so that they conform with the topology.

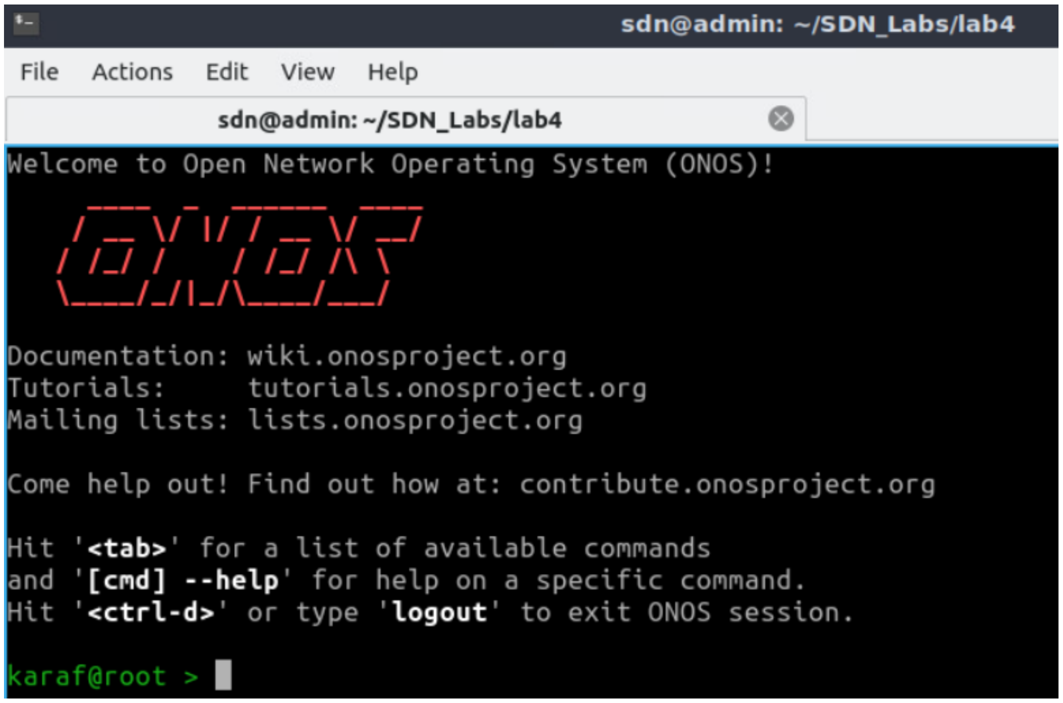
```
sudo ./run_onos.sh
```


sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
sdn@admin:~$ cd SDN_Labs/lab4
sdn@admin:~/SDN_Labs/lab4$ sudo ./run_onos.sh
[sudo] password for sdn:
```

Figure 11. Starting the ONOS controller.

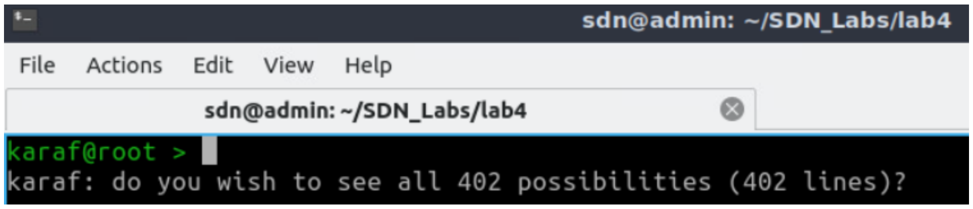
Once the script finishes executing and ONOS is ready, you will be able to execute commands on the ONOS CLI as shown in the figure below. Note that this script may take few seconds.


sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
Welcome to Open Network Operating System (ONOS)!
     ____  _  ______  ____
    / __ \/ |/ / __ \/ __/
   / /_/ /    / /_/ /\ \
   \____/_/|_/\____/___/

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

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

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

karaf@root >
```

Figure 12. ONOS CLI.

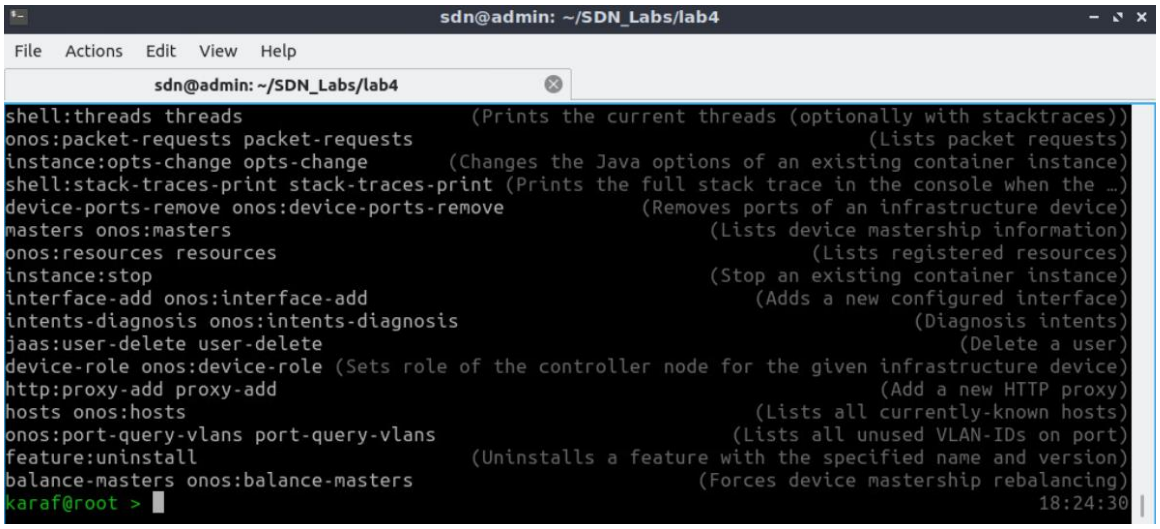
**Step 4.** ONOS supplies a set of its own commands, in order to list all available commands, press the TAB key.


sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
karaf@root >
karaf: do you wish to see all 402 possibilities (402 lines)?
```

Figure 13. Displaying a list of ONOS commands.

**Step 5.** To confirm displaying all possibilities of ONOS commands, press the `TAB` key one more time. This will display all ONOS commands on the left-hand side of the CLI, as well as their explanation on the right-hand side of the CLI.

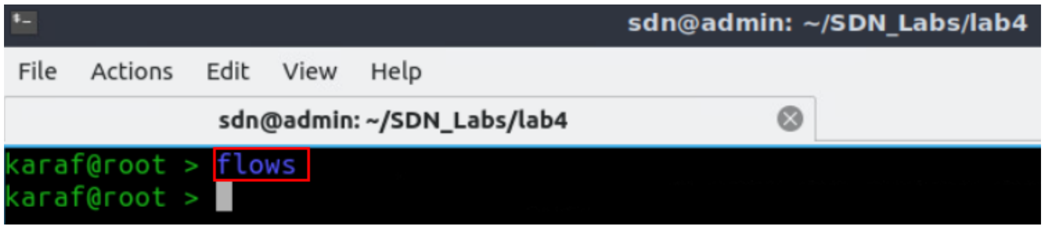

sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
shell:threads threads                    (Prints the current threads (optionally with stacktraces))
onos:packet-requests packet-requests                                    (Lists packet requests)
instance:opts-change opts-change       (Changes the Java options of an existing container instance)
shell:stack-traces-print stack-traces-print (Prints the full stack trace in the console when the …)
device-ports-remove onos:device-ports-remove           (Removes ports of an infrastructure device)
masters onos:masters                                        (Lists device mastership information)
onos:resources resources                                          (Lists registered resources)
instance:stop                                               (Stop an existing container instance)
interface-add onos:interface-add                                 (Adds a new configured interface)
intents-diagnosis onos:intents-diagnosis                                      (Diagnosis intents)
jaas:user-delete user-delete                                                      (Delete a user)
device-role onos:device-role (Sets role of the controller node for the given infrastructure device)
http:proxy-add proxy-add                                                     (Add a new HTTP proxy)
hosts onos:hosts                                                 (Lists all currently-known hosts)
onos:port-query-vlans port-query-vlans                         (Lists all unused VLAN-IDs on port)
feature:uninstall                        (Uninstalls a feature with the specified name and version)
balance-masters onos:balance-masters                         (Forces device mastership rebalancing)
karaf@root >                                                                              18:24:30
```

Figure 14. Displaying a list of ONOS commands.

**Step 6.** To display the list of all currently known flows for the ONOS controller, type the following command.

```
flows
```

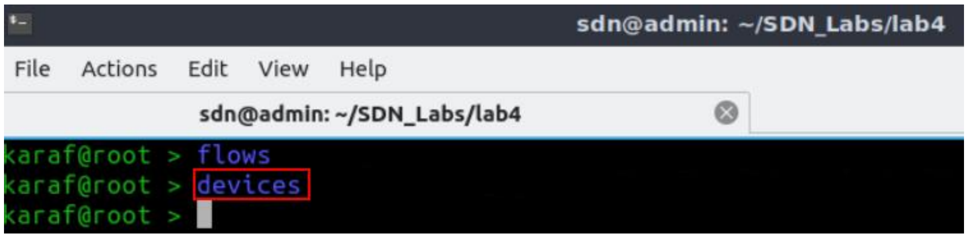

sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
karaf@root > flows
karaf@root >
```

Figure 15. Displaying the currently known flows.

**Step 7.** To display the list of all currently known devices (OVS switches), type the following command.

```
devices
```

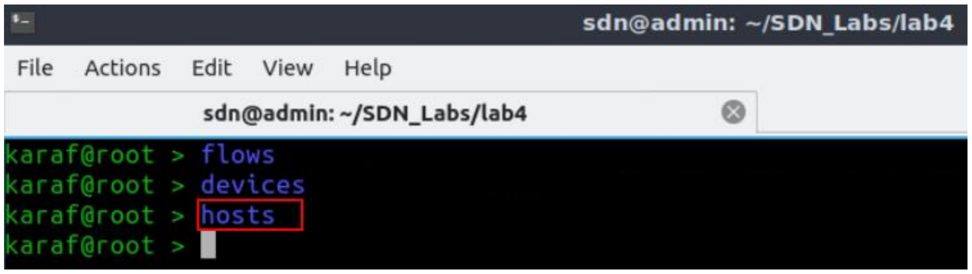

sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
karaf@root > flows
karaf@root > devices
karaf@root >
```

Figure 16. Displaying the currently known devices (switches).

**Step 8.** To display the list of all currently known hosts, type the following command.

```
hosts
```

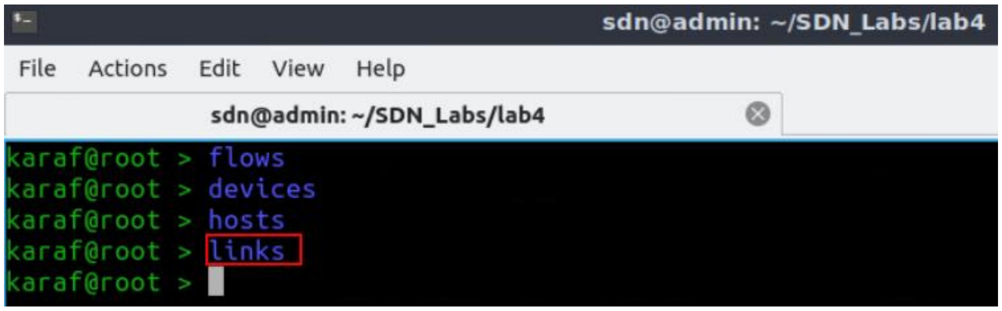


```
karaf@root > flows
karaf@root > devices
karaf@root > hosts
karaf@root > 
```

Figure 17. Displaying the currently known hosts.

**Step 9.** To display the list of all currently known links, type the following command.

```
links
```



```
karaf@root > flows
karaf@root > devices
karaf@root > hosts
karaf@root > links
karaf@root > 
```

Figure 18. Displaying the currently known link.

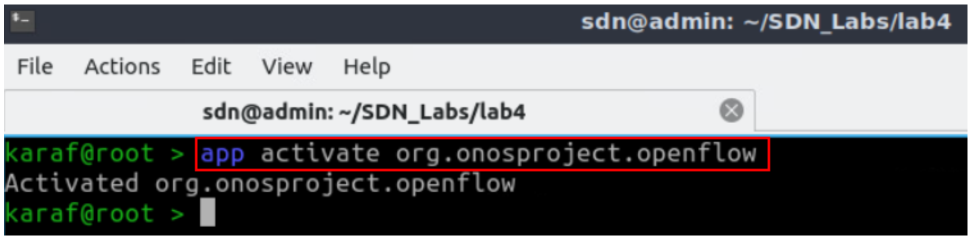
Consider Figures 15, 16, 17, and 18. No flows, devices, hosts, or link are displayed since we have not activated the necessary applications that allow the controller to discover them.

## 3.1 Activating the ONOS OpenFlow application

In this section, you will activate OpenFlow application that comes with ONOS and allows to speak OpenFlow protocol with the devices. You will notice how the Mininet topology becomes visible, i.e., the devices (switches), and hosts are now recognized by ONOS.

**Step 1.** In the ONOS terminal, issue the following command to activate the OpenFlow application.

```
app activate org.onosproject.openflow
```



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

Figure 19. Activating OpenFlow application.

**Step 2.** Open Mininet terminal.

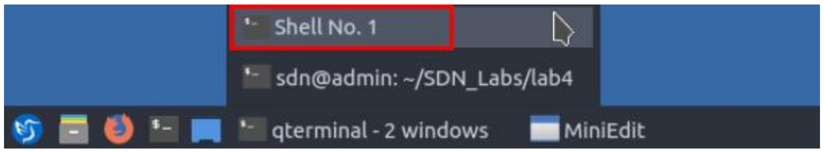
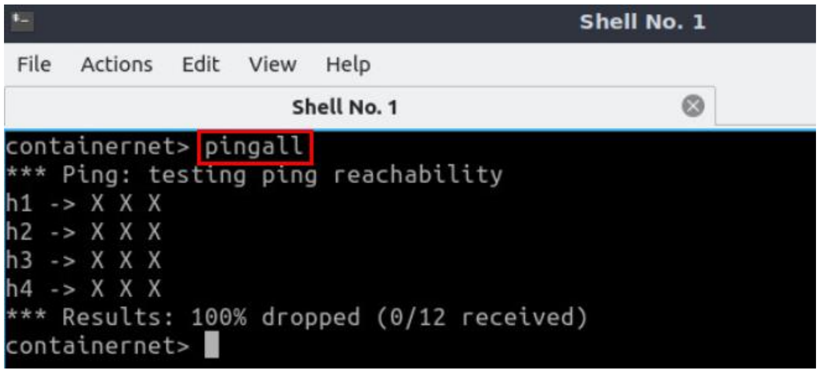


Figure 20. Opening Mininet terminal.

**Step 3.** Once the OpenFlow application is activated, you might not get accurate results immediately, for example, not all the hosts are discovered yet. In order to stimulate ONOS and the activated application, perform a `pingall` command. This command pings from every host in the network to all other hosts. To do so, write the following command.

```
pingall
```



```
containernet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
*** Results: 100% dropped (0/12 received)
containernet>
```

Figure 21. Pinging all hosts to stimulate host discovery in the Controller.

Consider Figure 21. The connectivity test resulted in 100% dropped. The pings sent from a host to a destination are IP packets received by a switch. For example, pinging host h2 from h1 is received by switch s1. Since there are no flows inserted that deal with IP packets, then the packets will be dropped immediately.

**Step 4.** Go to the ONOS terminal.

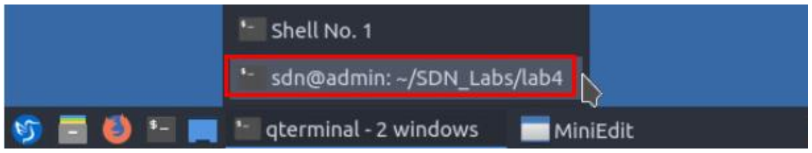
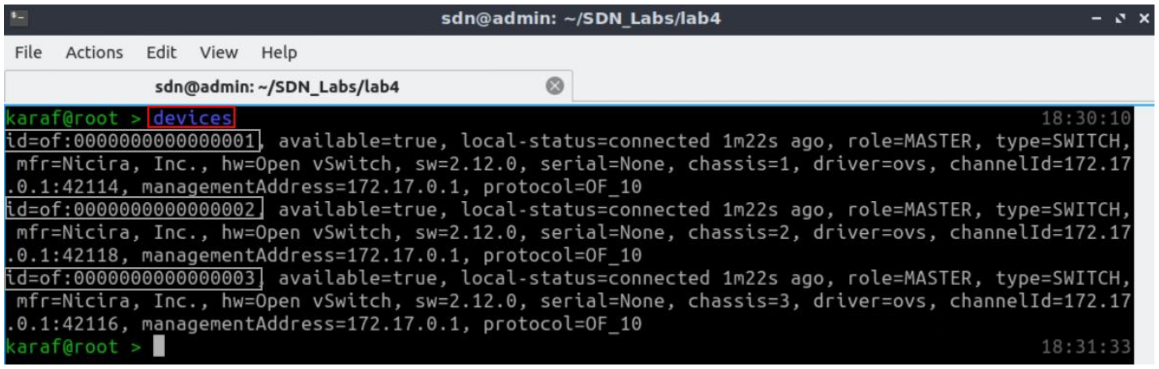


Figure 22. Opening ONOS terminal.

**Step 5.** To display the list of all currently known devices (OVS switches), type the following command.

```
devices
```


sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

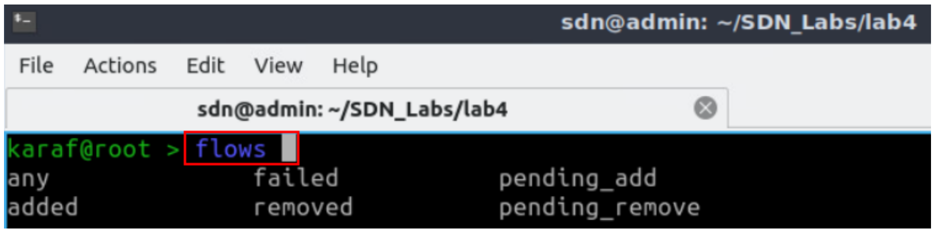
```
karaf@root > devices                                                    18:30:10
id=of:0000000000000001, available=true, local-status=connected 1m22s ago, role=MASTER, type=SWITCH,
 mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=1, driver=ovs, channelId=172.17
.0.1:42114, managementAddress=172.17.0.1, protocol=OF_10
id=of:0000000000000002, available=true, local-status=connected 1m22s ago, role=MASTER, type=SWITCH,
 mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=2, driver=ovs, channelId=172.17
.0.1:42118, managementAddress=172.17.0.1, protocol=OF_10
id=of:0000000000000003, available=true, local-status=connected 1m22s ago, role=MASTER, type=SWITCH,
 mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=3, driver=ovs, channelId=172.17
.0.1:42116, managementAddress=172.17.0.1, protocol=OF_10
karaf@root >                                                            18:31:33
```

Figure 23. Displaying the currently known devices (switches).

Consider Figure 23. The three switches are displayed with their corresponding ids, as well as additional attributes, such as the status (`available=true`), the type of the switch (`hw=Open vSwitch`), and which OpenFlow version the switch is running (`protocol=OF_10`).

**Step 6.** ONOS commands might have multiple options. To display the list of options for the flows command, type the command `flows` , then press the `TAB` button twice.

```
flows
```


sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
karaf@root > flows
any                  failed               pending_add
added                removed              pending_remove
```

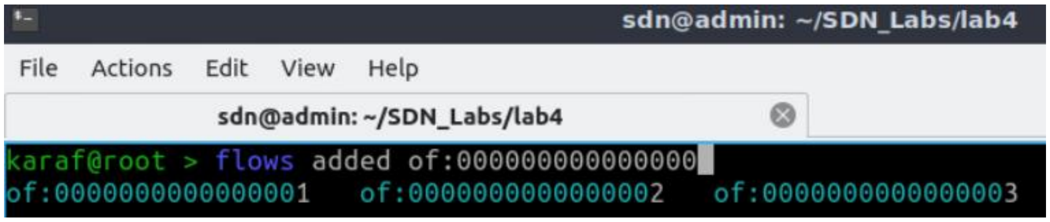
Figure 24. Displaying the currently known flows.

Consider Figure 24. The `flows` command has multiple options according to the state of the flow, which could be:

- `added`: the flow has been added to the switch.
- `failed`: the flow failed to be added.
- `pending_add`: the flow has been submitted and forwarded to the switch.
- `pending_remove`: the request to remove the flow has been submitted and forwarded to the switch.
- `removed`: the rule has been removed.
- `any`: all flows.

**Step 7.** To display the list of all added switches, complete the above command by typing `added`, then press the `TAB` button twice.

```
flows added
```


sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

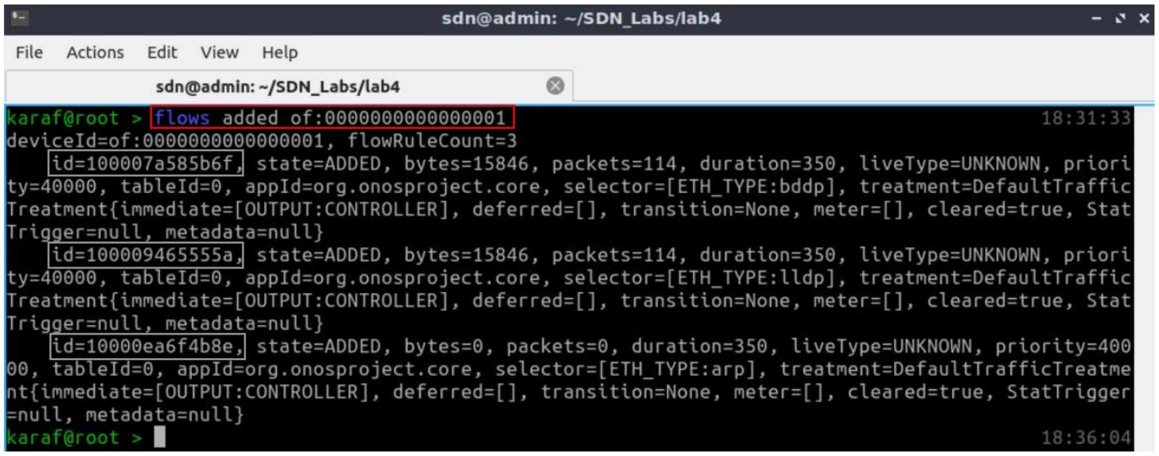
```
karaf@root > flows added of:000000000000000
of:0000000000000001   of:0000000000000002   of:0000000000000003
```

Figure 25. Displaying the currently known added flows for switch 1.

Consider Figure 25. Once you press the `TAB` button, the ONOS CLI automatically fills the common prefix of all the switches' ids (000000000000000). You can also alternate between the displayed switches' ids using the `TAB` button.

**Step 8.** To display the list of all added flows for switch 1 (id: 00000000000001), complete the id of the switch (you only have to enter the number `1` for switch s1) and hit enter.

```
flows added of:00000000000001
```


sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
karaf@root > flows added of:0000000000000001                                        18:31:33
deviceId=of:0000000000000001, flowRuleCount=3
    id=100007a585b6f, state=ADDED, bytes=15846, packets=114, duration=350, liveType=UNKNOWN, priori
ty=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:bddp], treatment=DefaultTraffic
Treatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, Stat
Trigger=null, metadata=null}
    id=100009465555a, state=ADDED, bytes=15846, packets=114, duration=350, liveType=UNKNOWN, priori
ty=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:lldp], treatment=DefaultTraffic
Treatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, Stat
Trigger=null, metadata=null}
    id=10000ea6f4b8e, state=ADDED, bytes=0, packets=0, duration=350, liveType=UNKNOWN, priority=400
00, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatment=DefaultTrafficTreatme
nt{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger
=null, metadata=null}
karaf@root >                                                                        18:36:04
```

Figure 26. Displaying the currently known added flows for switch 1.

Consider Figure 26. Three flows for switch s1 were added. ONOS provides many details about the flows inserted in the switches. For example, each flow entry defines a `selector` and `treatment` which are the set of traffic matched by the flow entry and how this traffic should be handled, respectively.

The controller has installed three initial flows which are:

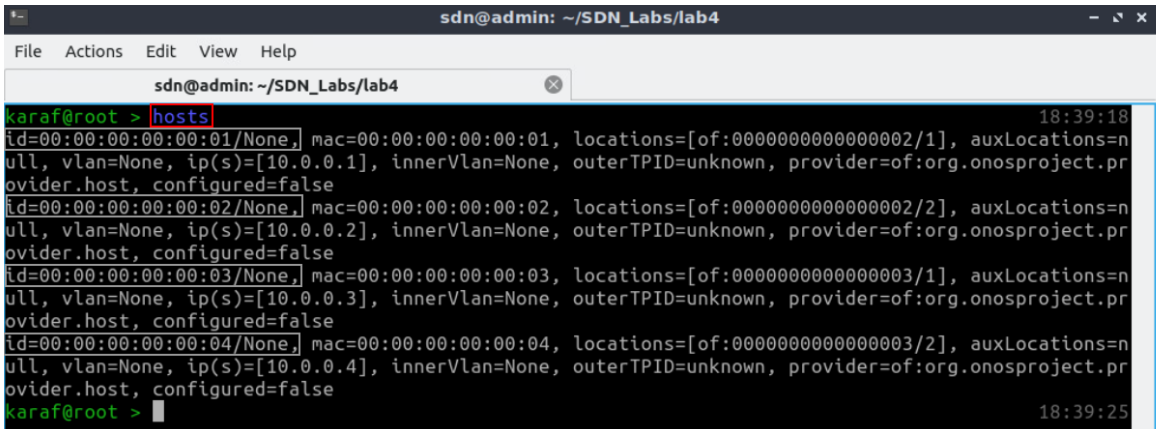
- Flow 1 (`ETH_TYPE=bddp`): forwards Broadcast Domain Discovery Protocol (BDDP) to the controller (`[OUTPUT:CONTROLLER]`).
- Flow 2 (`ETH_TYPE=lldp`): forwards Link Layer Discovery Protocol (LLDP) packets to the controller (`[OUTPUT:CONTROLLER]`).
- Flow 3 (`ETH_TYPE=arp`): forwards Address Resolution Protocol (ARP) packets to the controller (`[OUTPUT:CONTROLLER]`).

The above flows are used for link and host discovery. Notice as well that each flow entry is tagged by an `appId` (application id), this `appId` identifies which application installed the corresponding flow entry. Other important details include:

- `packets`: number of packets forwarded or dropped for that flow.
- `bytes`: byte count of the matched packets.
- `tableId`: id of the table in which these flows are installed.
- `duration`: time elapsed in seconds since the flow was installed.

**Step 9.** To display the list of all currently known hosts, type the following command.

```
hosts
```


sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
karaf@root > hosts                                                          18:39:18
id=00:00:00:00:00:01/None, mac=00:00:00:00:00:01, locations=[of:0000000000000002/1], auxLocations=n
ull, vlan=None, ip(s)=[10.0.0.1], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.pr
ovider.host, configured=false
id=00:00:00:00:00:02/None, mac=00:00:00:00:00:02, locations=[of:0000000000000002/2], auxLocations=n
ull, vlan=None, ip(s)=[10.0.0.2], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.pr
ovider.host, configured=false
id=00:00:00:00:00:03/None, mac=00:00:00:00:00:03, locations=[of:0000000000000003/1], auxLocations=n
ull, vlan=None, ip(s)=[10.0.0.3], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.pr
ovider.host, configured=false
id=00:00:00:00:00:04/None, mac=00:00:00:00:00:04, locations=[of:0000000000000003/2], auxLocations=n
ull, vlan=None, ip(s)=[10.0.0.4], innerVlan=None, outerTPID=unknown, provider=of:org.onosproject.pr
ovider.host, configured=false
karaf@root >                                                                18:39:25
```

Figure 27. Displaying the currently known hosts.

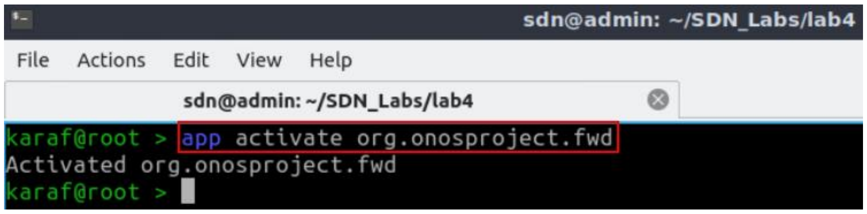
Consider Figure 27. Each discovered host is displayed along with some details, such as the identification of the host, the location where the host is connected to (with the identification of the switch), and the IP address.

## 3.2 Activating the ONOS forwarding application

In the previous section, when performing a connectivity test, 100% of the packets were dropped. The pings sent from a host to a destination are IP packets received by a switch, which does not have flows that match IP packets. In this section, you will activate a simple forwarding application that comes with ONOS. This application installs flows in response to every *miss* IP packet that arrives at the controller.

**Step 1.** To enable the forwarding application, type the command shown below.

```
app activate org.onosproject.fwd
```

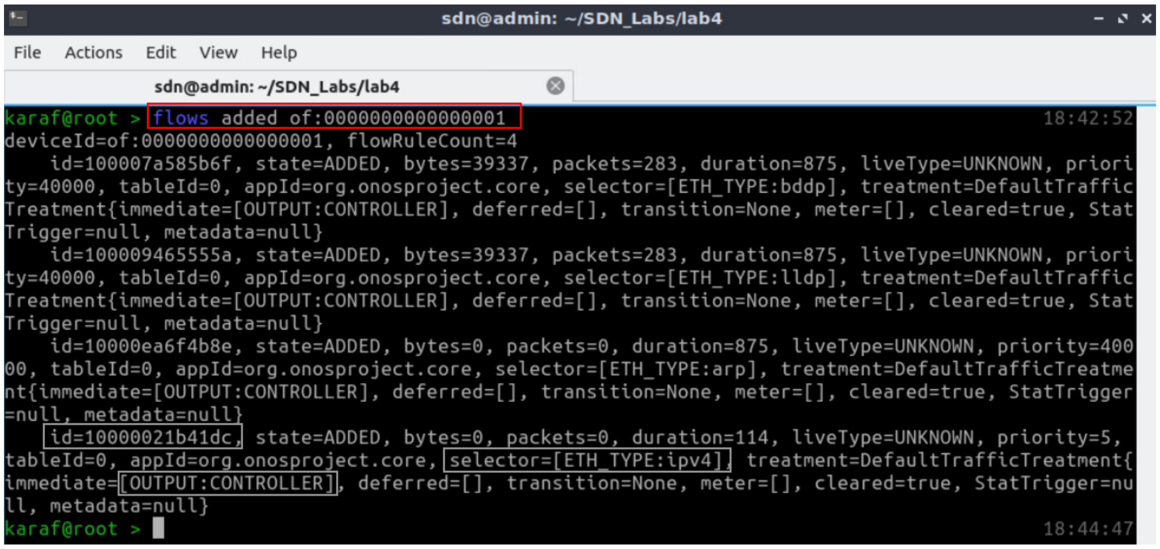

sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
karaf@root > app activate org.onosproject.fwd
Activated org.onosproject.fwd
karaf@root >
```

Figure 28. Activating the forwarding application.

**Step 2.** To display the flows of switch s1, type the following command.

```
flows added of:0000000000000001
```



```
karaf@root > flows added of:0000000000000001                                        18:42:52
deviceId=of:0000000000000001, flowRuleCount=4
    id=100007a585b6f, state=ADDED, bytes=39337, packets=283, duration=875, liveType=UNKNOWN, priori
ty=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:bddp], treatment=DefaultTraffic
Treatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, Stat
Trigger=null, metadata=null}
    id=100009465555a, state=ADDED, bytes=39337, packets=283, duration=875, liveType=UNKNOWN, priori
ty=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:lldp], treatment=DefaultTraffic
Treatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, Stat
Trigger=null, metadata=null}
    id=10000ea6f4b8e, state=ADDED, bytes=0, packets=0, duration=875, liveType=UNKNOWN, priority=400
00, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatment=DefaultTrafficTreatme
nt{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger
=null, metadata=null}
    id=10000021b41dc, state=ADDED, bytes=0, packets=0, duration=114, liveType=UNKNOWN, priority=5,
tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:ipv4], treatment=DefaultTrafficTreatment{
immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=true, StatTrigger=nu
ll, metadata=null}
karaf@root >                                                                        18:44:47
```

Figure 29. Displaying the currently known devices (switches).

Consider Figure 29. A new flow is added with `ETH_TYPE:ipv4` that deals with IPv4 packet by forwarding them to the controller (`OUTPUT:CONTROLLER`), which in turn decides the action on the corresponding packet.

**Step 3.** Hold right-click on host h1 and select *Terminal.*

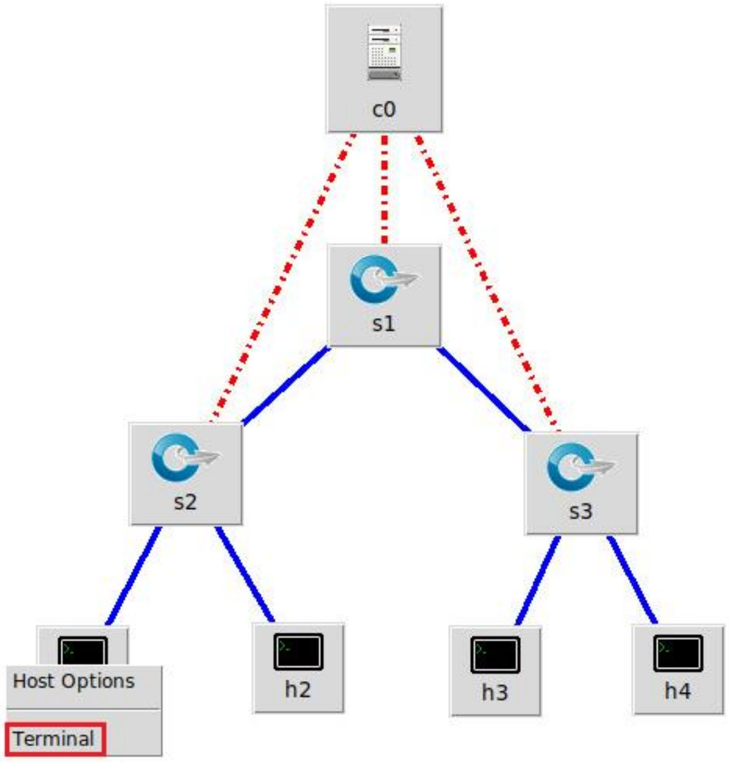
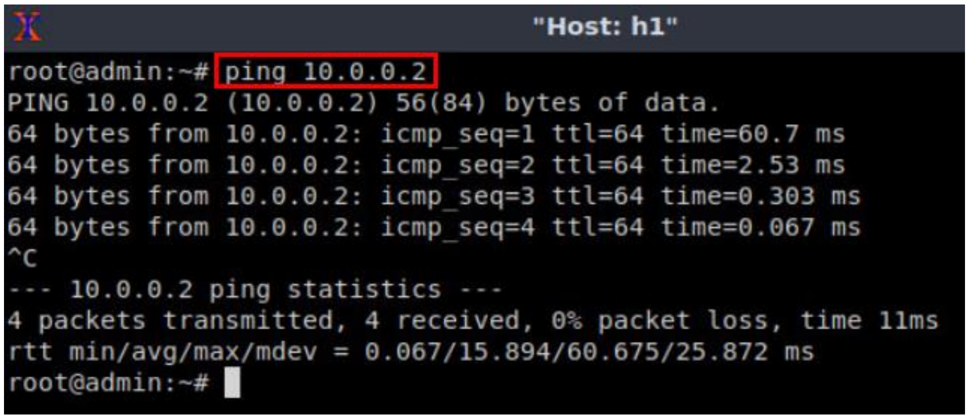


Figure 30. Displaying the currently known devices (switches).

**Step 4.** On host h1 terminal, run a connectivity test by issuing the command shown below.

```
ping 10.0.0.2
```



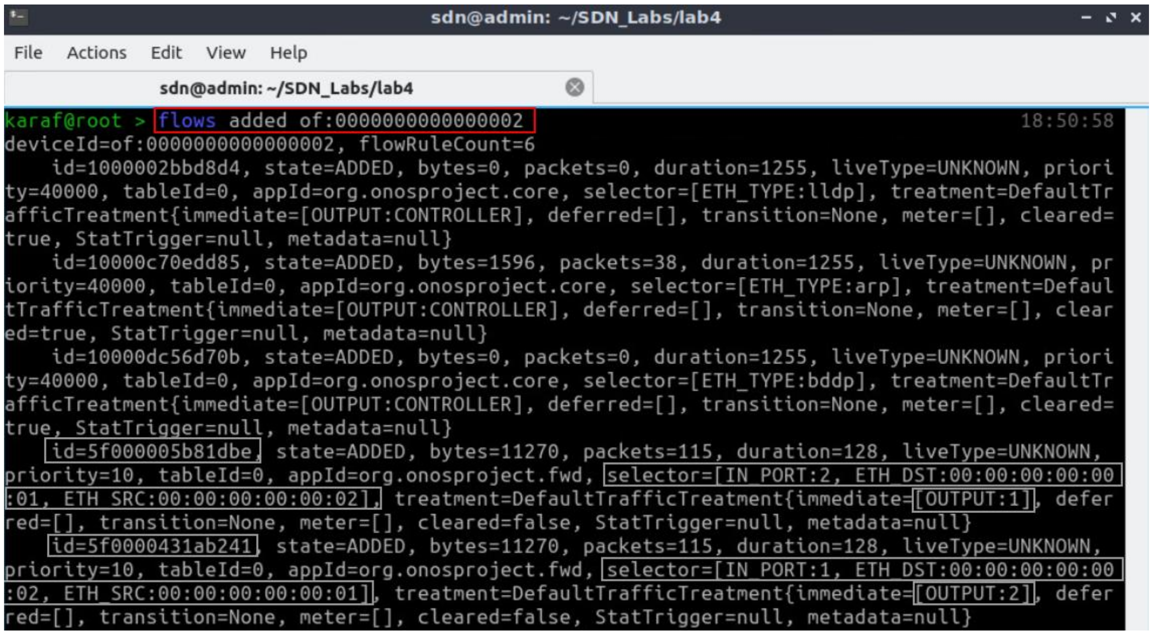
```
"Host: h1"
root@admin:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=60.7 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2.53 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.303 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.067 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 11ms
rtt min/avg/max/mdev = 0.067/15.894/60.675/25.872 ms
root@admin:~#
```

Figure 31. Pinging host h2 from host h1.

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

**Step 5.** To display the flows of switch s2, type the following command on the ONOS terminal.

```
flows added of:0000000000000002
```



sdn@admin: ~/SDN_Labs/lab4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab4

```
karaf@root > flows added of:0000000000000002                                   18:50:58
deviceId=of:0000000000000002, flowRuleCount=6
    id=1000002bbd8d4, state=ADDED, bytes=0, packets=0, duration=1255, liveType=UNKNOWN, priori
ty=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:lldp], treatment=DefaultTr
afficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=
true, StatTrigger=null, metadata=null}
    id=10000c70edd85, state=ADDED, bytes=1596, packets=38, duration=1255, liveType=UNKNOWN, pr
iority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatment=Defaul
tTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], clear
ed=true, StatTrigger=null, metadata=null}
    id=10000dc56d70b, state=ADDED, bytes=0, packets=0, duration=1255, liveType=UNKNOWN, priori
ty=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:bddp], treatment=DefaultTr
afficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, meter=[], cleared=
true, StatTrigger=null, metadata=null}
    id=5f000005b81dbe, state=ADDED, bytes=11270, packets=115, duration=128, liveType=UNKNOWN,
priority=10, tableId=0, appId=org.onosproject.fwd, selector=[IN_PORT:2, ETH_DST:00:00:00:00:00
:01, ETH_SRC:00:00:00:00:00:02], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:1], defer
red=[], transition=None, meter=[], cleared=false, StatTrigger=null, metadata=null}
    id=5f0000431ab241, state=ADDED, bytes=11270, packets=115, duration=128, liveType=UNKNOWN,
priority=10, tableId=0, appId=org.onosproject.fwd, selector=[IN_PORT:1, ETH_DST:00:00:00:00:00
:02, ETH_SRC:00:00:00:00:00:01], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:2], defer
red=[], transition=None, meter=[], cleared=false, StatTrigger=null, metadata=null}
```

Figure 32. Displaying the added flows on switch s2.

Consider Figure 32. After pinging host h2 from h1, two flows are installed on switch s2. The first flow (`id=5f000005b81dbe`) instructs the switch to forward incoming packets at port 2 (`IN_PORT:2`) out of port 1 (`OUTPUT:1`). Similarly, the second flow instructs the switch to forward the packets from port 1 to port 2. The inserted flows allow switch s1 to exchange packets between hosts h1 and h2 without relaying them to the controller.

Note that these two flows expire after a certain duration. This is because the forwarding application sets an expiry time for the inserted flows to avoid overflowing the flow table of the switches.

## 4 Navigating the ONOS GUI

The ONOS GUI is a *single-page web-application*, providing a visual interface to the ONOS controller. In this section, you will navigate through the ONOS GUI and discover a number of its features.

### 4.1 Accessing the web user interface

**Step 1.** Open the web browser by clicking on the shortcut located on the lower left-hand side.

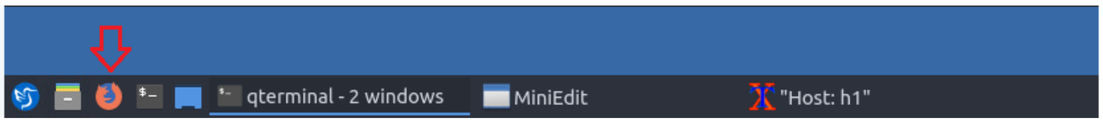


Figure 33. Opening the web browser

**Step 2.** Navigate to the following URL to access the ONOS web user interface.

```
localhost:8181/onos/ui
```

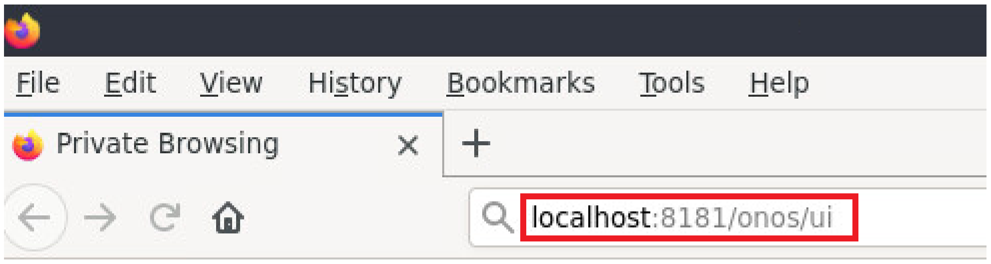


Figure 34. Opening the ONOS web user interface.

**Step 3.** Provide the following credentials to access the web user interface.

- User: `onos`
- Password: `rocks`

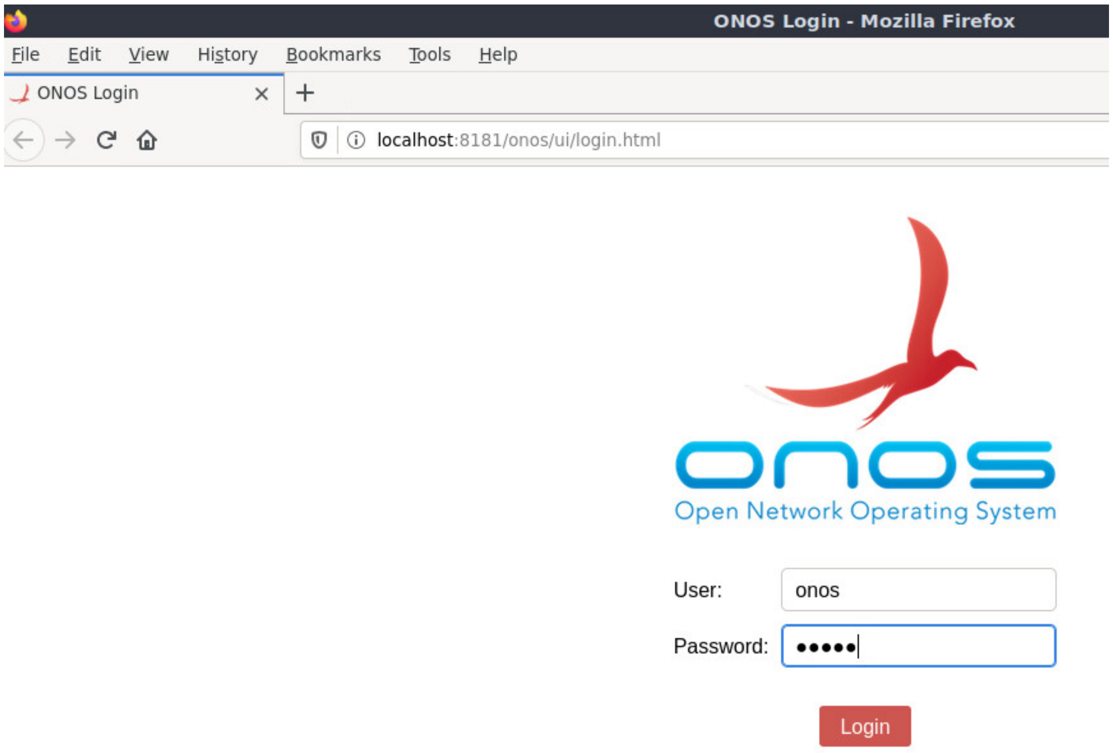
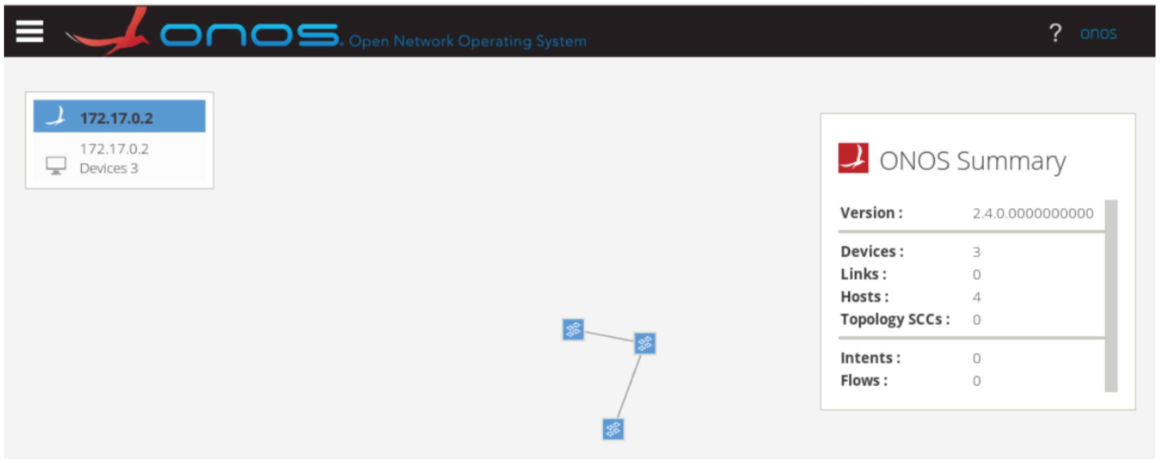

ONOS Login - Mozilla Firefox
File Edit View History Bookmarks Tools Help
ONOS Login
localhost:8181/onos/ui/login.html
onos
Open Network Operating System
User: onos
Password: •••••
Login

Figure 35. ONOS authentication window.

A topology consisting of three switches will be displayed. Such topology corresponds to the one created on Mininet.



| | |
|---|---|
| Version : | 2.4.0.0000000000 |
| Devices : | 3 |
| Links : | 0 |
| Hosts : | 4 |
| Topology SCCs : | 0 |
| Intents : | 0 |
| Flows : | 0 |

Figure 36. Displaying the topology in ONOS GUI.

Notice that the network components will show up in a random arrangement. You can click on the components and drag them to their preferred location.

## 4.2 Exploring network components

The ONOS web user interface provides the tools to monitor the specification of each device that in the network.

**Step 1.** To open the ONOS menu, click on the upper left-hand side icon. A drop-down menu will be displayed. To check the devices, click on devices. A new window will emerge.

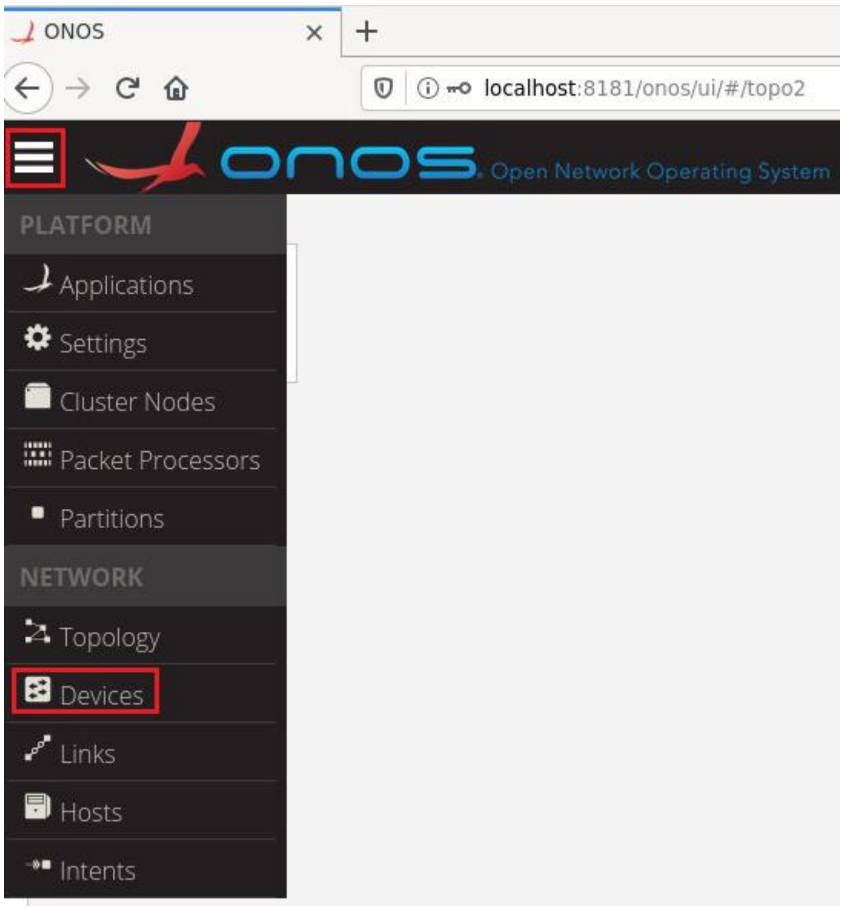
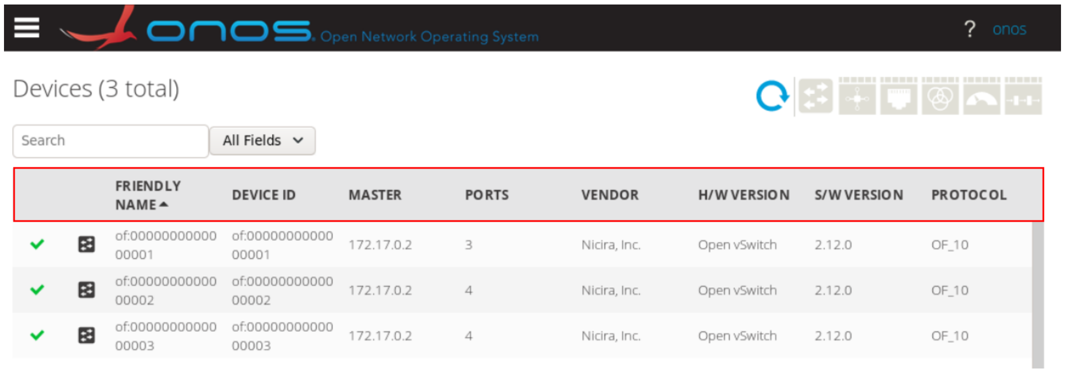


Figure 37. Opening devices.

The emerging window will display three devices. Those three devices correspond to the three switches that compound the topology. The information provided in the GUI includes:

- FRIENDLY NAME: if not specified, it is the name of the device.
- DEVICE ID: name of the switch.
- MASTER: IP address where ONOS is running. ONOS is running locally in a docker container (172.17.0.2).
- PORTS: number of ports of the switch.
- PROTOCOL: OpenFlow version running on the switch.



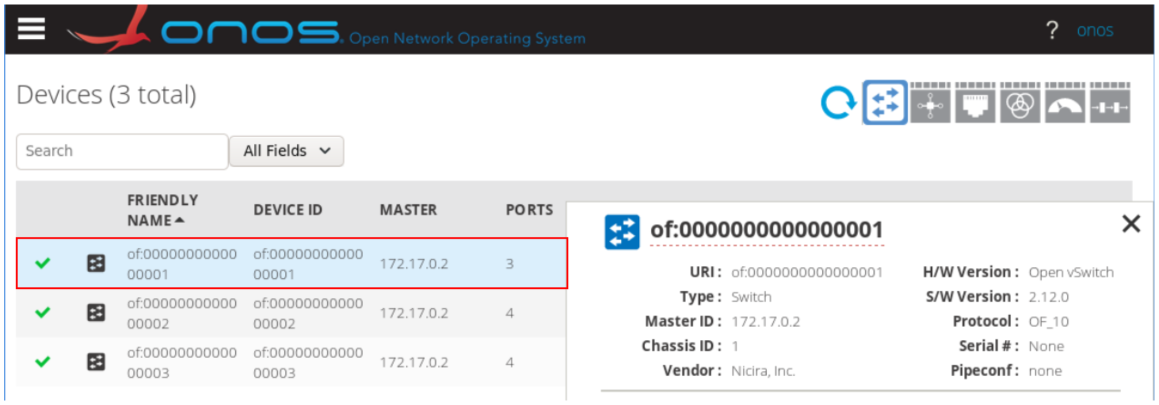
Devices (3 total)

Search | All Fields

| | | FRIENDLY NAME ▲ | DEVICE ID | MASTER | PORTS | VENDOR | H/W VERSION | S/W VERSION | PROTOCOL |
|---|---|---|---|---|---|---|---|---|---|
| ✓ | | of:0000000000000001 | of:0000000000000001 | 172.17.0.2 | 3 | Nicira, Inc. | Open vSwitch | 2.12.0 | OF_10 |
| ✓ | | of:0000000000000002 | of:0000000000000002 | 172.17.0.2 | 4 | Nicira, Inc. | Open vSwitch | 2.12.0 | OF_10 |
| ✓ | | of:0000000000000003 | of:0000000000000003 | 172.17.0.2 | 4 | Nicira, Inc. | Open vSwitch | 2.12.0 | OF_10 |

Figure 38. Information about the devices.

**Step 2.** Click on the entry corresponding to switch s1 (0000000000000001).



Devices (3 total)

Search | All Fields

| | | FRIENDLY NAME ▲ | DEVICE ID | MASTER | PORTS |
|---|---|---|---|---|---|
| ✓ | | of:0000000000000001 | of:0000000000000001 | 172.17.0.2 | 3 |
| ✓ | | of:0000000000000002 | of:0000000000000002 | 172.17.0.2 | 4 |
| ✓ | | of:0000000000000003 | of:0000000000000003 | 172.17.0.2 | 4 |

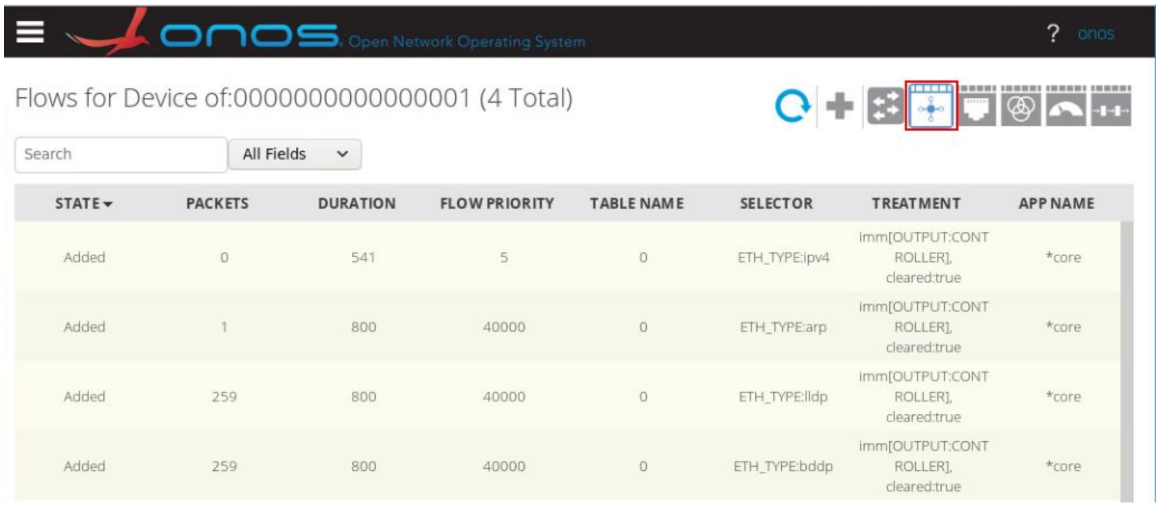
of:0000000000000001

| | | | |
|---|---|---|---|
| URI : | of:0000000000000001 | H/W Version : | Open vSwitch |
| Type : | Switch | S/W Version : | 2.12.0 |
| Master ID : | 172.17.0.2 | Protocol : | OF_10 |
| Chassis ID : | 1 | Serial # : | None |
| Vendor : | Nicira, Inc. | Pipeconf : | none |

Figure 39. Selecting switch s1.

Consider Figure 39. Once you select a device, a side window will appear on the right-hand side of the screen showing a summary of the device.

**Step 3.** Click on the upper right button to show a view of the flow table of switch s1.



Flows for Device of:0000000000000001 (4 Total)

Search | All Fields

| STATE ▾ | PACKETS | DURATION | FLOW PRIORITY | TABLE NAME | SELECTOR | TREATMENT | APP NAME |
|---|---|---|---|---|---|---|---|
| Added | 0 | 541 | 5 | 0 | ETH_TYPE:ipv4 | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 1 | 800 | 40000 | 0 | ETH_TYPE:arp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 259 | 800 | 40000 | 0 | ETH_TYPE:lldp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 259 | 800 | 40000 | 0 | ETH_TYPE:bddp | imm[OUTPUT:CONTROLLER], cleared:true | *core |

Figure 40. Displaying the flow table of switch s1.

Similarly, you can navigate around the buttons displayed on the upper right to display information related to the port of the selected device, as well as other advanced features.

**Step 4.** From the menu list, click on hosts to verify the hosts' information.

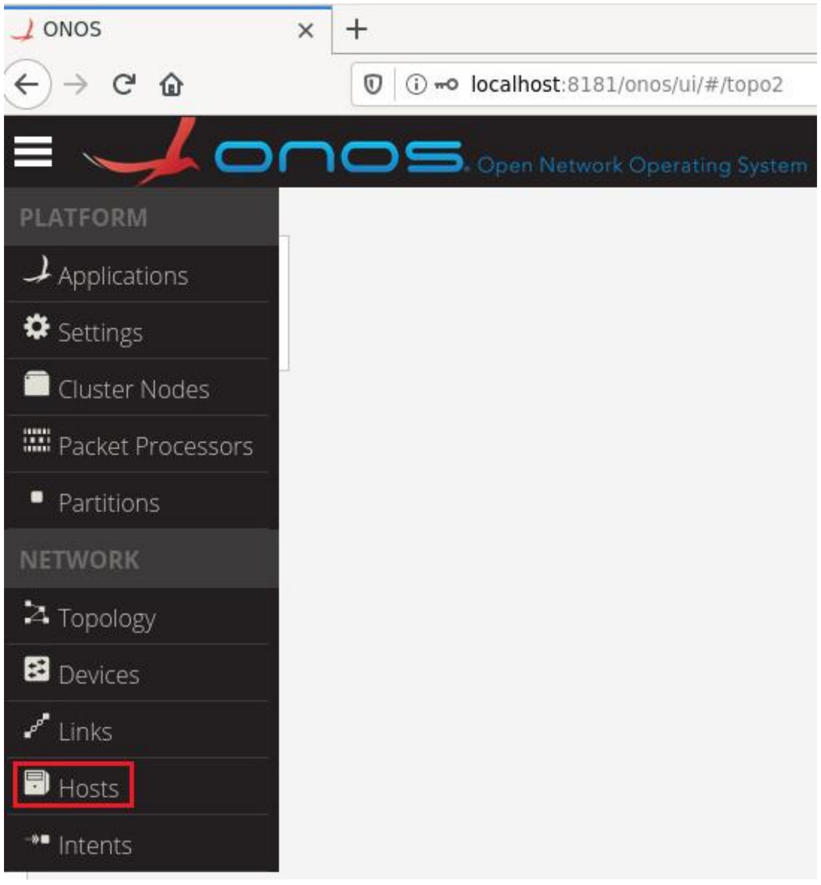
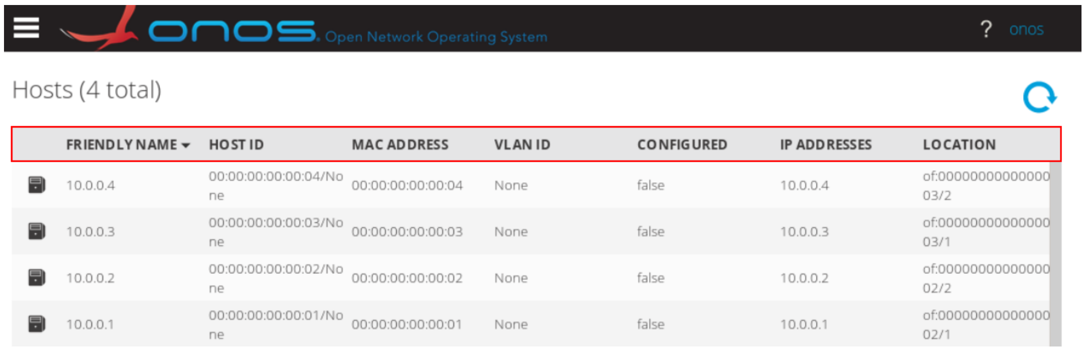


Figure 41. Opening hosts.

Similarly, the new window presents the information regarding the hosts that compound the topology. The information provided in the GUI include:

- Host ID/MAC ADDRESS: mac address of the host.
- IP ADDRESSES: IP address of the host.
- Location: the port of the switch connected to the host.



Hosts (4 total)

| | FRIENDLY NAME | HOST ID | MAC ADDRESS | VLAN ID | CONFIGURED | IP ADDRESSES | LOCATION |
|---|---|---|---|---|---|---|---|
| | 10.0.0.4 | 00:00:00:00:00:04/None | 00:00:00:00:00:04 | None | false | 10.0.0.4 | of:0000000000000003/2 |
| | 10.0.0.3 | 00:00:00:00:00:03/None | 00:00:00:00:00:03 | None | false | 10.0.0.3 | of:0000000000000003/1 |
| | 10.0.0.2 | 00:00:00:00:00:02/None | 00:00:00:00:00:02 | None | false | 10.0.0.2 | of:0000000000000002/2 |
| | 10.0.0.1 | 00:00:00:00:00:01/None | 00:00:00:00:00:01 | None | false | 10.0.0.1 | of:0000000000000002/1 |

Figure 42. Hosts' information.

**Step 5.** Go back to the main window (topology).

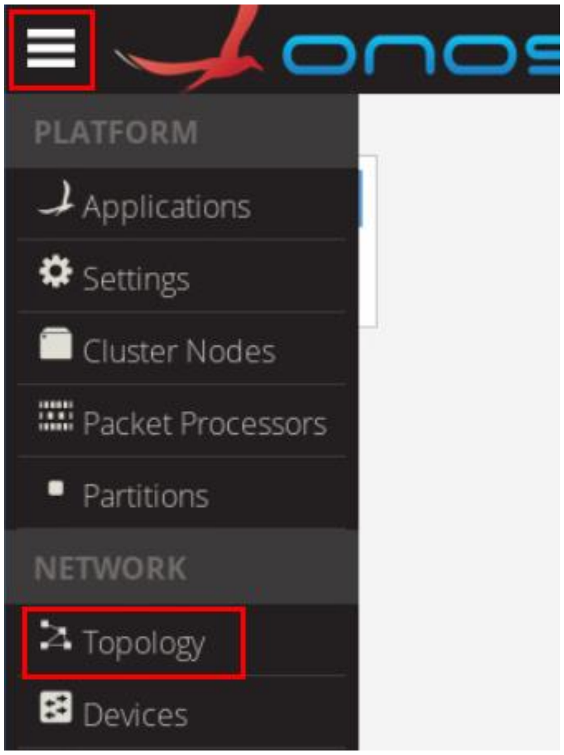


Figure 43. Opening the topology.

**Step 6.** Click on the bar located on the lower left-hand side. A toolbox will show up. Select the host icon to see the hosts connected to the topology.

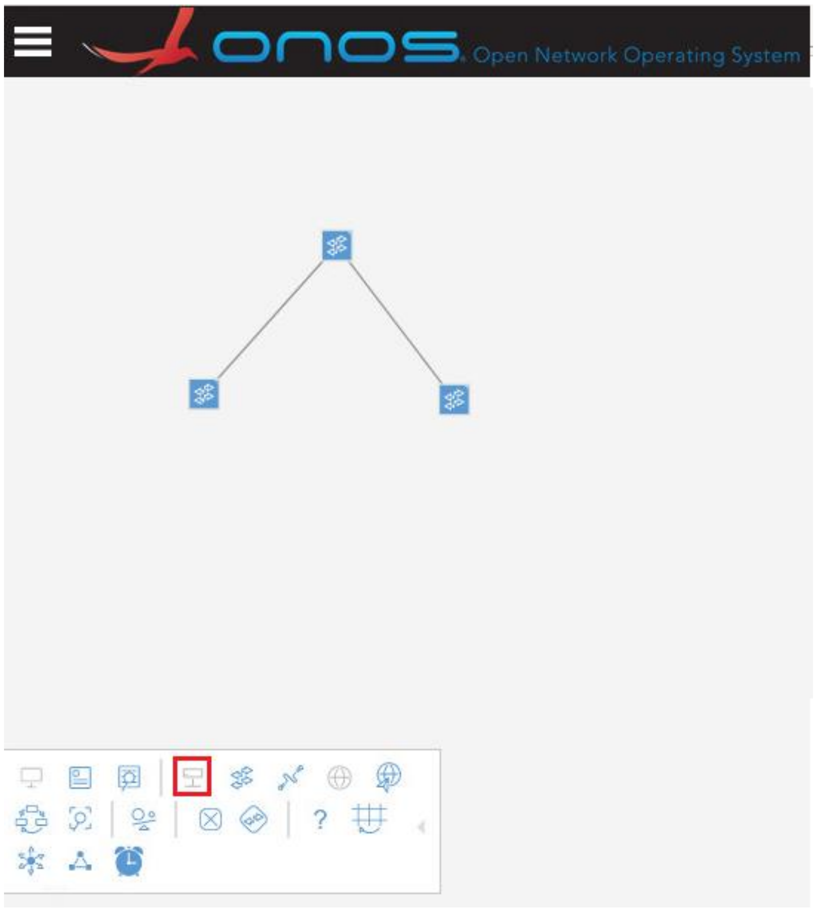


Figure 44. Enabling hosts visualization.

**Step 7.** Go back to the topology to verify the network components.

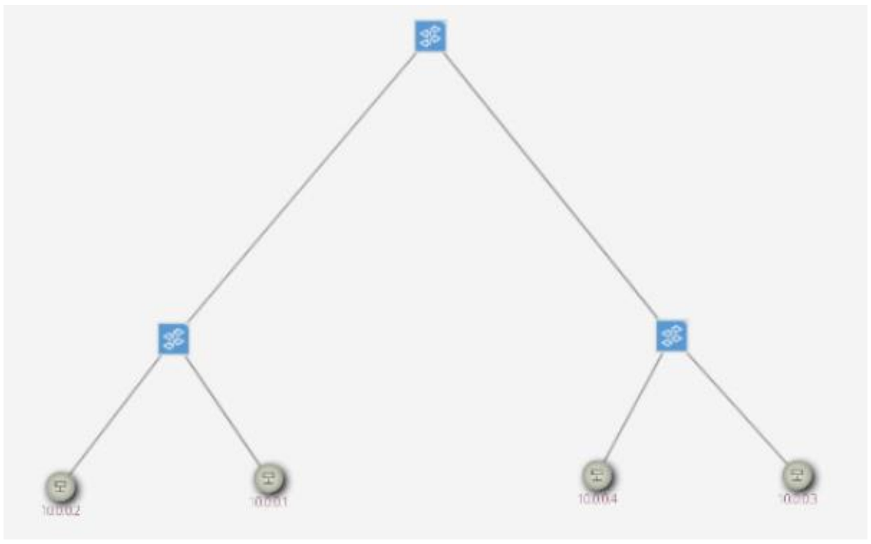
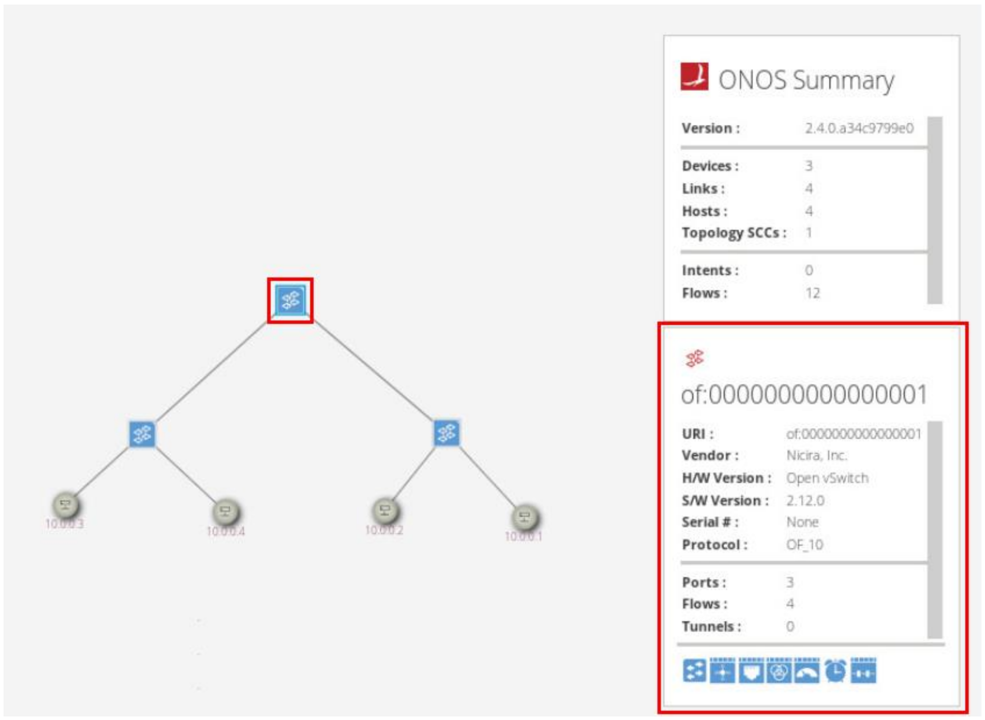


Figure 45. Network components.

Consider Figure 45. The information attached to the hosts includes their IP addresses.

**Step 8.** To view a summary of a specific device, you can click on that device as shown in the figure below.



ONOS Summary

| | |
|---|---|
| Version : | 2.4.0.a34c9799e0 |
| Devices : | 3 |
| Links : | 4 |
| Hosts : | 4 |
| Topology SCCs : | 1 |
| Intents : | 0 |
| Flows : | 12 |

of:0000000000000001

| | |
|---|---|
| URI : | of:0000000000000001 |
| Vendor : | Nicira, Inc. |
| H/W Version : | Open vSwitch |
| S/W Version : | 2.12.0 |
| Serial # : | None |
| Protocol : | OF_10 |
| Ports : | 3 |
| Flows : | 4 |
| Tunnels : | 0 |

Figure 46. Network components.

Consider Figure 46. Upon clicking on switch s1, a panel opens on the right-hand side of the topology. The panel displays a summary of the device. For example, if it is a switch, it would display information such as the name, number of ports, and flows in that switch.

**Step 9.** From the menu list, click on Applications to load the applications available in ONOS.

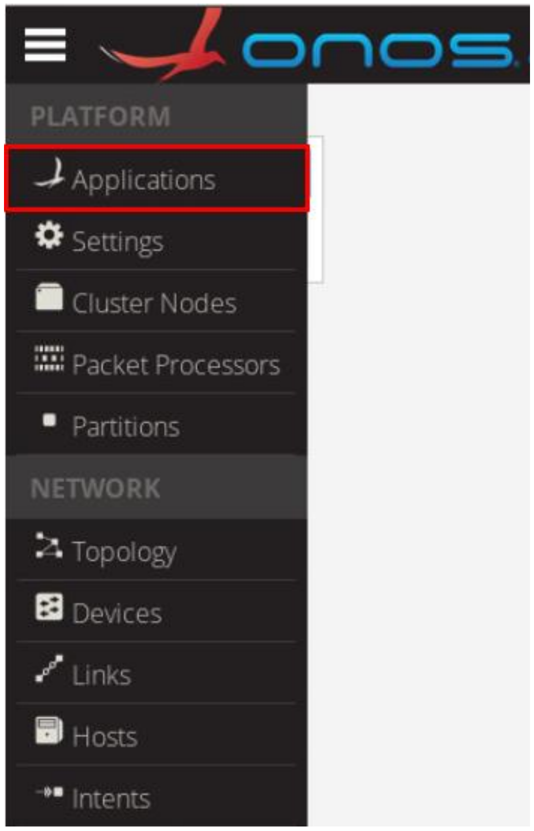
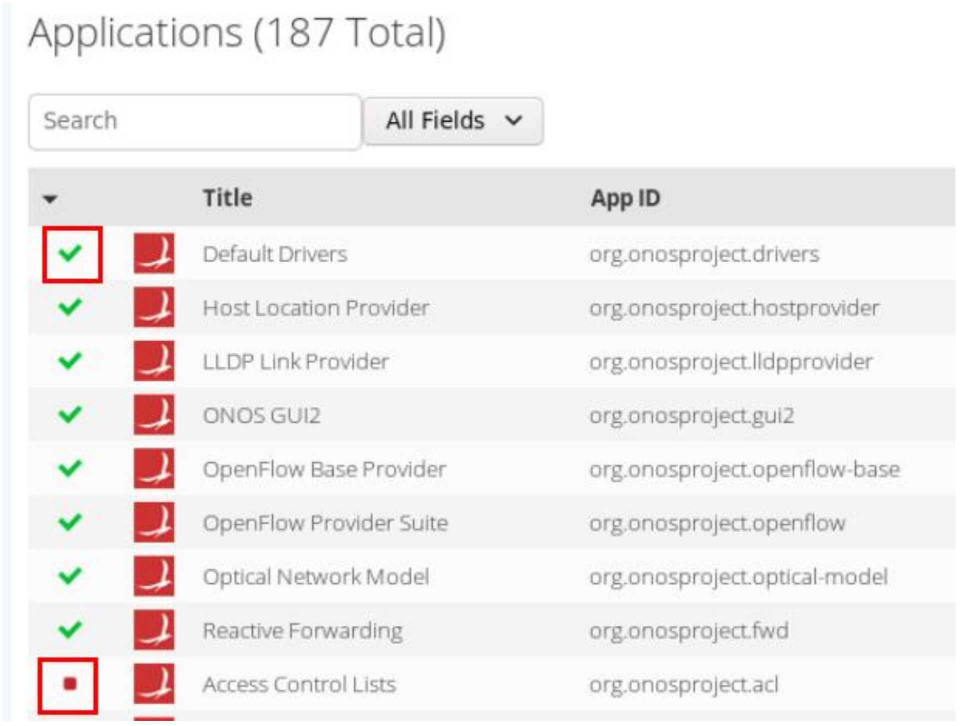


Figure 47. Opening applications.



Applications (187 Total)

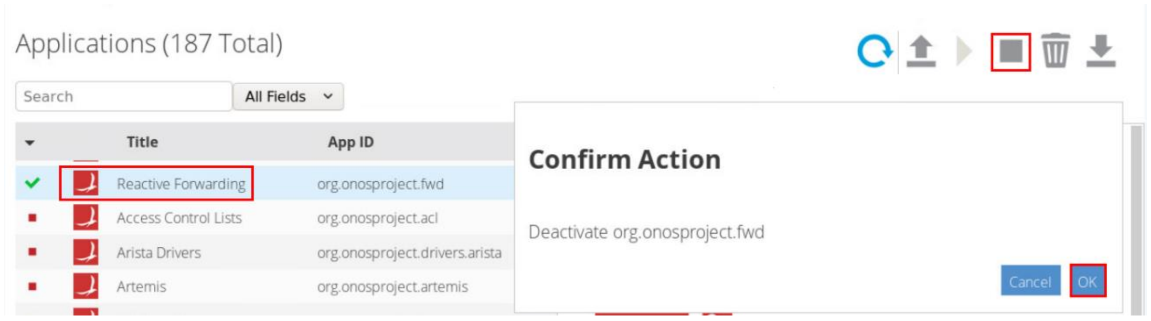
Search | All Fields

| | | Title | App ID |
|---|---|---|---|
| ✓ | | Default Drivers | org.onosproject.drivers |
| ✓ | | Host Location Provider | org.onosproject.hostprovider |
| ✓ | | LLDP Link Provider | org.onosproject.lldpprovider |
| ✓ | | ONOS GUI2 | org.onosproject.gui2 |
| ✓ | | OpenFlow Base Provider | org.onosproject.openflow-base |
| ✓ | | OpenFlow Provider Suite | org.onosproject.openflow |
| ✓ | | Optical Network Model | org.onosproject.optical-model |
| ✓ | | Reactive Forwarding | org.onosproject.fwd |
| 🟥 | | Access Control Lists | org.onosproject.acl |

Figure 48. Displaying ONOS applications.

Consider Figure 48. A list of all the available applications is displayed. The activated applications are checked in green (e.g., Default Drivers application), whereas the deactivated applications are marked with a red square (e.g., Access Control Lists application). Note that the activated applications are a result of activating the OpenFlow and the forwarding application.

**Step 10.** To deactivate an application, you can click on the application, click the deactivation button (gray square on the right-hand side of the window), then confirm the operation. The steps are shown in the figure below, where we deactivate the forwarding application (reactive forwarding).



Applications (187 Total)

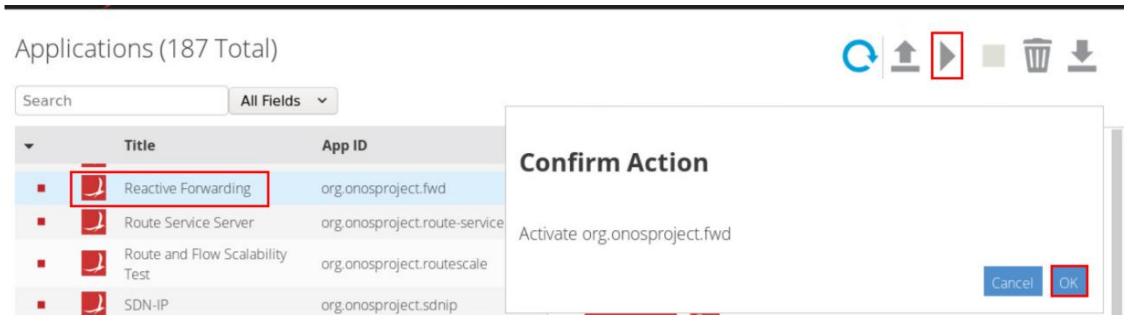
| | Title | App ID |
|---|---|---|
| ✓ | Reactive Forwarding | org.onosproject.fwd |
| ■ | Access Control Lists | org.onosproject.acl |
| ■ | Arista Drivers | org.onosproject.drivers.arista |
| ■ | Artemis | org.onosproject.artemis |

Confirm Action

Deactivate org.onosproject.fwd

Figure 49. Deactivating an application using the GUI.

**Step 11.** To activate an application, you can click on the application, click the activation button (gray arrow on the right-hand side of the window), then confirm the operation. The steps are shown in the figure below, where we activate the forwarding application (reactive forwarding).



Applications (187 Total)

| | Title | App ID |
|---|---|---|
| ■ | Reactive Forwarding | org.onosproject.fwd |
| ■ | Route Service Server | org.onosproject.route-service |
| ■ | Route and Flow Scalability Test | org.onosproject.routescale |
| ■ | SDN-IP | org.onosproject.sdnip |

Confirm Action

Activate org.onosproject.fwd

Figure 50. Activating an application using the GUI.

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

## References

1. P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar. "*ONOS: towards an open, distributed SDN OS*," In Proceedings of the third workshop on Hot topics in software defined networking, pp. 1-6, 2014.
2. 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.
3. D. Kreutz, F.M. Ramos, P.E. Verissimo, C.E. Rothenberg, S. Azodolmolky, and S. Uhlig, *"Software-defined networking: A comprehensive survey*," Proceedings of the IEEE, 103(1), pp.14-76, 2014.

4. P. Goransson, C. Black, T. Culver. "*Software defined networks: a comprehensive approach*". Morgan Kaufmann, 2016.
5. Mininet walkthrough, [Online]. Available: http://mininet.org.

UNIVERSITY OF SOUTH CAROLINA

# SOFTWARE DEFINED NETWORKING

# Exercise 1: SDN Network Configuration

**Document Version: 09-02-2021**

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

## Contents

## 1 Exercise description

Consider Figure 1. The topology consists of two end-hosts, two switches and a controller within the Software Defined Networking (SDN) network. The blue devices represent OpenFlow switches. All switches are connected to the controller c0.

The goal of this exercise is to manage the OpenFlow switches using the ONOS controller so that the two hosts can communicate with each other. Essentially, you should navigate through the ONOS Command Line Interface (CLI) to enable applications, inspect the links, devices, etc., and the flow table of the switches. The topology below is already built and you should use Mininet to emulate it.

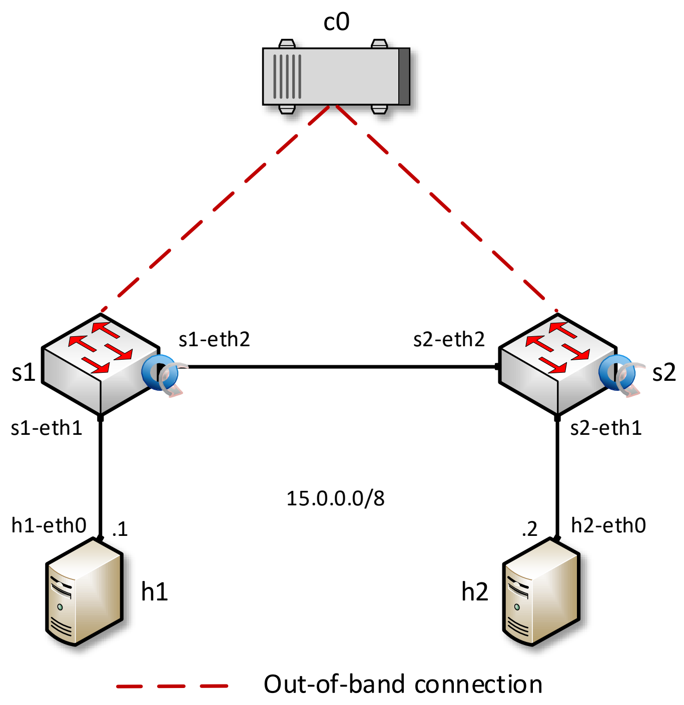


Figure 1. Exercise topology.

### 1.1 Topology settings

The devices are already configured according to Table 1.

Table 1. Topology information.

| Device | Interface | MAC Address | IP Address | Subnet |
|---|---|---|---|---|
| h1 | h1-eth0 | 00:00:00:00:00:01 | 15.0.0.1 | /8 |
| h2 | h2-eth0 | 00:00:00:00:00:02 | 15.0.0.2 | /8 |
| c0 | N/A | N/A | 172.17.0.2 | /16 |

### 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)** Open MiniEdit and load the topology above. The topology file *Exercise1.mn* of this exercise is in the directory *~/SDN_Labs/Exercise1* as shown in the figure below.

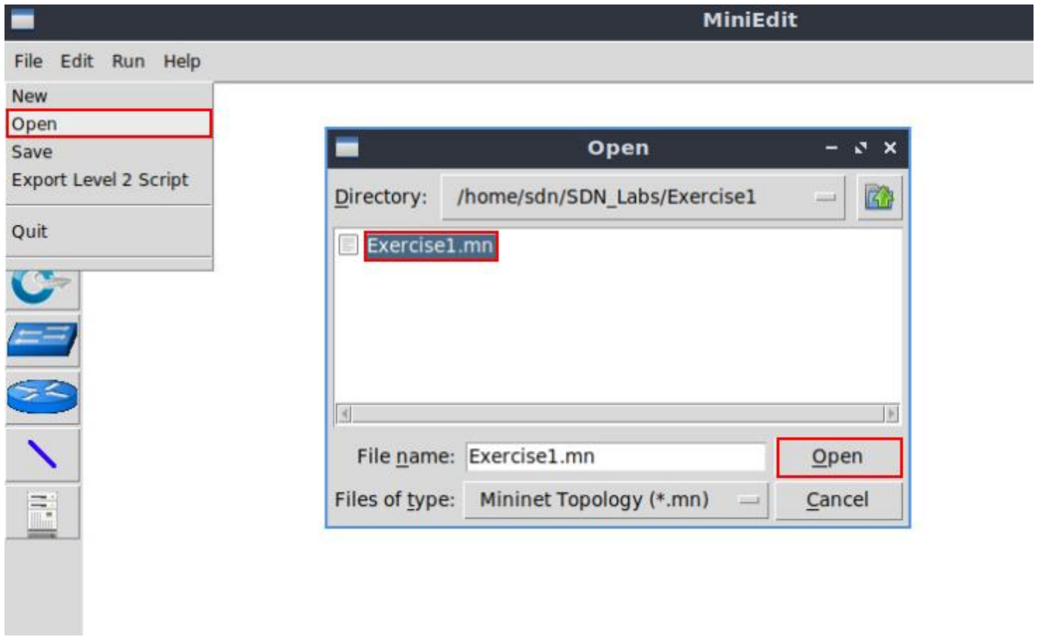
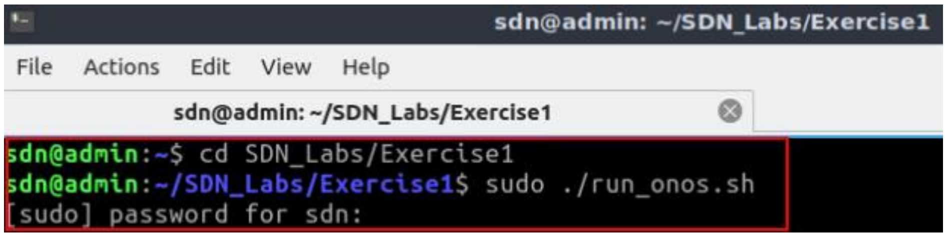


Figure 2. Loading the topology file in Mininet.

**b)** In the Linux terminal, navigate to the directory *~/SDN_Labs/Exercise1* and execute, in superuser mode, the script *run_onos.sh* that runs the ONOS controller. When prompted for a password, type `password`. The steps to run ONOS are depicted in the figure below.


sdn@admin: ~/SDN_Labs/Exercise1
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/Exercise1

```
sdn@admin:~$ cd SDN_Labs/Exercise1
sdn@admin:~/SDN_Labs/Exercise1$ sudo ./run_onos.sh
[sudo] password for sdn:
```

Figure 3. Starting the ONOS controller.

**c)** In the ONOS terminal, inspect and list the flows, devices, hosts, and links observed by the ONOS controller. Explain the results.

**d)** Activate the necessary ONOS application that allows the controller to communicate with the OpenFlow switches. Repeat the previous step and report any changes. Explain the results.

**e)** Test the connectivity between the two hosts by performing a ping test from host h1 to host h2. Explain the result of the connectivity test.

**f)** Activate the necessary ONOS application that enables IP forwarding.

**g)** In the ONOS terminal, inspect the flow table of switches s1 and s2 and report the flow entry (specifically, the match and the action of the entry) that enables IP forwarding.

**h)** Test the connectivity between the two hosts by performing a ping test from host h1 to host h2. While the connectivity test is running, report the added entries in the flow table of switch s1 and explain their functionality.

# SOFTWARE DEFINED NETWORKING

# Lab 5: Configuring VXLAN to Provide Network Traffic Isolation

**Document Version: 03-30-2021**

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

## Contents

## Overview

This lab presents Virtual eXtensible Local Area Network (VXLAN), a network virtualization scheme that provides a solution for the scalability problems associated with data center and large cloud computing deployments. The goal of this lab is to configure VXLAN 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 VXLAN.
2. Emulate servers by using docker containers.
3. Push flow tables to configure VXLAN in a switch.
4. Isolate network traffic by using VXLAN.
5. Visualize VXLAN network identifiers by using Wireshark.

## Lab settings

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

Table 1**.** Credentials to access the 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 router r1 and router r2.
4. Section 4: Configuring VXLAN.
5. Section 5: Verifying configuration.

## 1 Introduction

Data centers operate by hosting services for multiple tenants, such as data servers and cloud computing services. Those services require on-demand provisioning of computing resources for multi-tenant environments. Network virtualization supports such requirements and provides an efficient way to host multiple tenants on the same server.

Additionally, it incorporates traffic isolation to avoid a tenant having access to another tenant's data.

Network traffic isolation can be performed on layer 2 or layer 3 networks. For example, Virtual Local Area Networks (VLANs) isolate layer 2 traffic by tagging and handling network frames. However, VLAN-based network isolation can handle up to 4094 VLANs, which is insufficient considering cloud services' high demand. Additionally, a tenant might require multiple VLANs, which aggravates the issue.

On the other hand, layer 3 networks do not provide a comprehensive solution for multi-tenant networks. Two tenants might use the same set of layers 3 addresses within their networks, which requires the cloud provider to provide traffic isolation in other forms. Further, requiring all tenants to use IP excludes customers relying on direct layer 2 or non-IP layer 3 protocols for inter virtual machine (VM) communication.

## 1.1 VXLAN architecture

Virtual eXtensible Local Area Network[1] (VXLAN) addresses the shortcomings of VLAN, providing a framework for overlaying virtualized layer 2 networks over layer 3 networks. Hypervisor-based overlay networks are a novel use of Software-Defined Network (SDN) capabilities. This concept does not modify the physical network, which means that networking devices and their configurations remain unchanged. In these networks, details about the physical network are not shared or accessible from the end-devices. VXLAN runs over the existing networking infrastructure and supports a higher number of hosts compared to what VLANs can handle. Each overlay is unique within the tenant domain and is known as the VXLAN segment. The communication is restricted just among hosts within the same VXLAN segment.

Figure 1(a) presents a topology that illustrates how VXLAN segments are transported over a layer 3 network (e.g., IP network). The endpoints of the tunnels are known as the VXLAN Tunnel End Point (VTEP). The VTEP is responsible for encapsulating the layer 2 frames in a VXLAN header and forward that on the IP Network. It also handles the reversal process that consists of de-encapsulating an incoming VXLAN segment and forward the original frame to its corresponding Local Area Network (LAN). Figure 1(b) shows that the VXLAN framework is represented as a tunneling scheme that transports layer 2 frames on top of a layer 3 network. The tunnels are stateless, meaning that each frame is encapsulated according to a set of predefined rules. The end-hosts do not store session information.

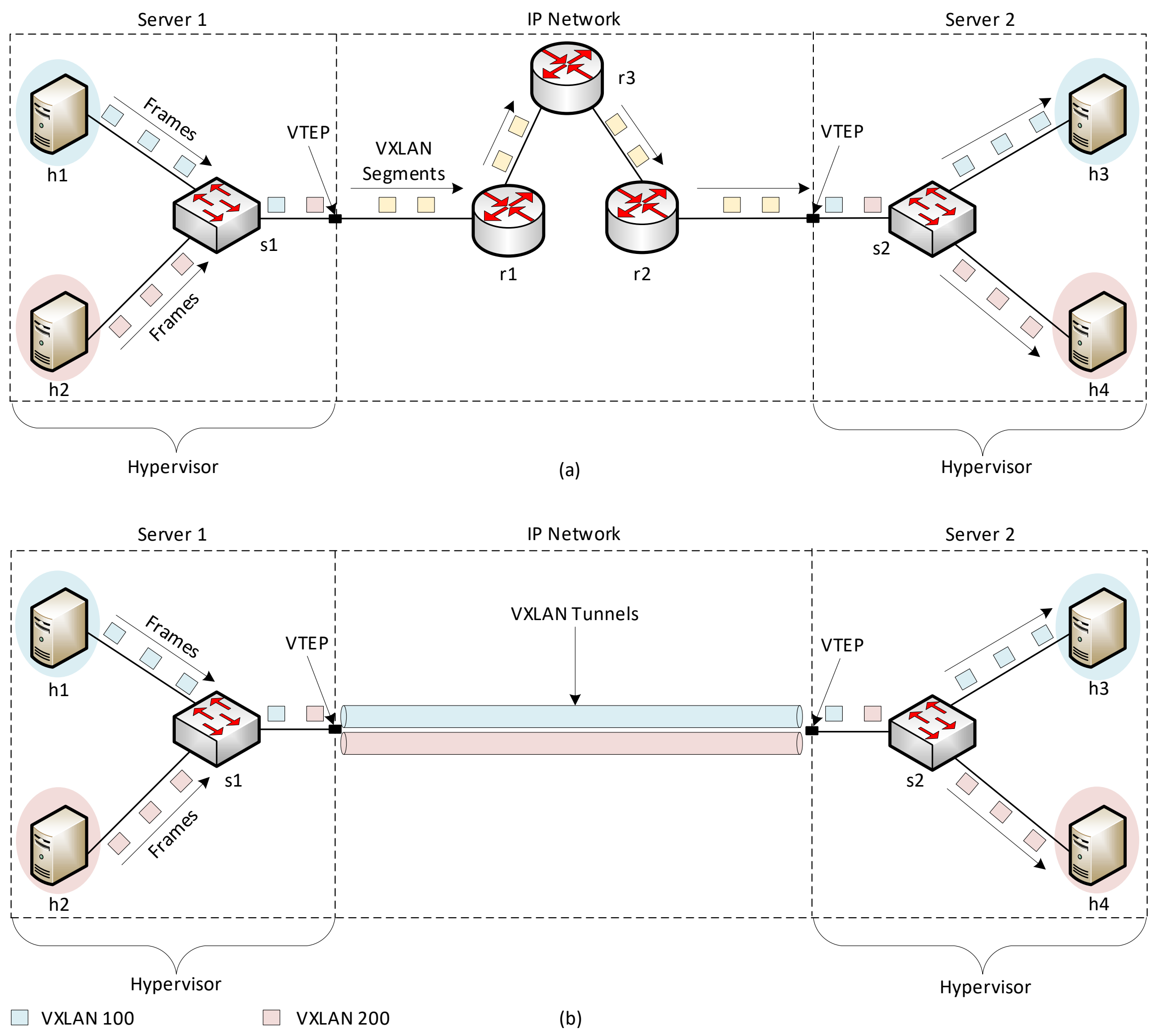


Figure 1. VXLAN overview. (a) VXLAN segments are transported over a layer 3 network (e.g., IP network). (b) The VXLAN framework is represented as a tunneling scheme that transports layer 2 frames on top of layer 3 networks.

## 1.2 VXLAN segment format

Figure 2 illustrates the format of a VXLAN segment[2]. The original layer 2 frame contains a VXLAN header encapsulated in a UDP-IP packet. VXLAN encapsulation adds an 8-bytes header to the original layer 2 frame. The VXLAN header and the original layer 2 frame are in the UDP payload. The outer header contains the MAC and IP addresses appropriate for sending a unicast packet to the destination switch, acting as a virtual tunnel endpoint. A VXLAN segment consists of a 24-bit segment tag called the VXLAN Network Identifier (VNI). Therefore, VXLANs can handle around 16 million (i.e., $2^{24}$) network segments that coexist within the same administrative domain.

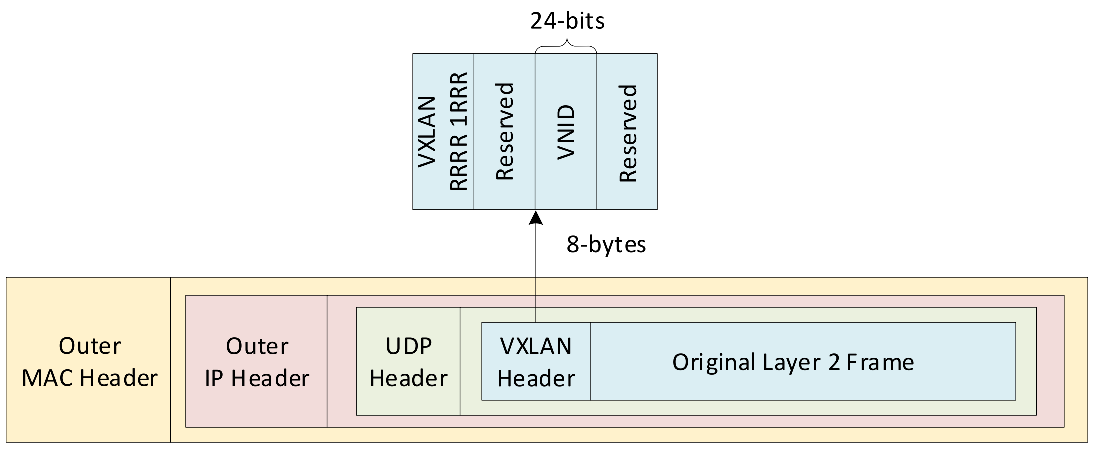


Figure 2. VXLAN segment format.

## 2 Lab topology

Consider Figure 3. The topology consists of four end-hosts, two switches, and three routers. The end-hosts and switches are running inside Server 1 and Server 2. Those servers are implemented in Docker[10] containers (i.e., Container d1 and Container d2), which run Mininet instances. Routers r1, r2, and r3 run Free-Range Routing (FRR) engine representing an IP network.

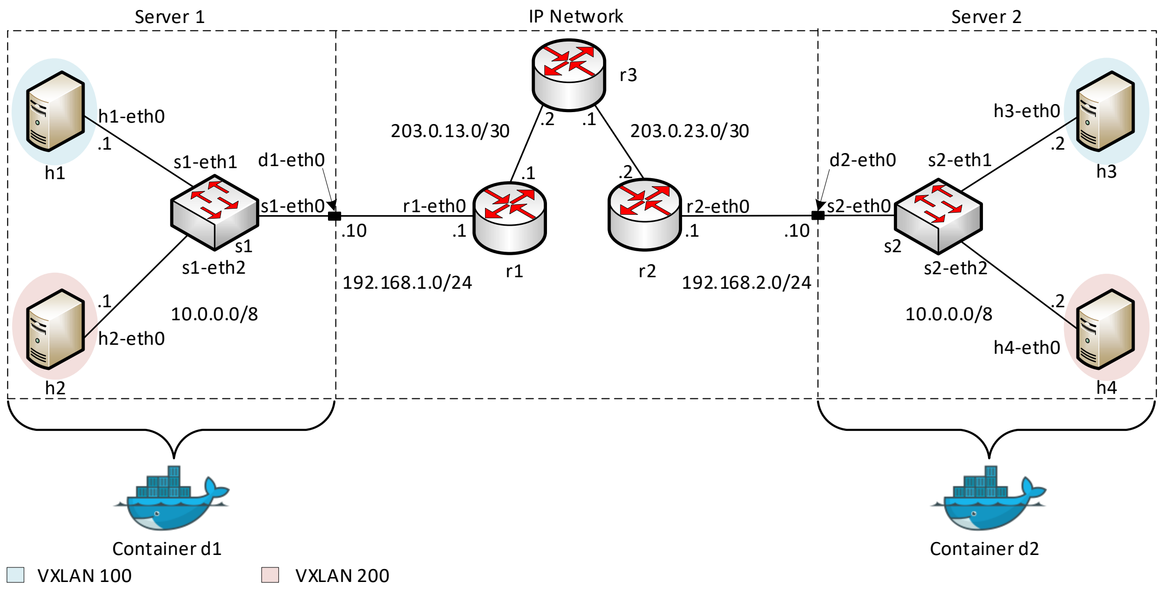


Figure 3. Lab topology.

### 2.1 Lab settings

The devices are already 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 | 203.0.23.2 | /30 |
| r3 | r3-eth0 | 203.0.13.2 | /30 |
| | r3-eth1 | 203.0.23. | /30 |
| h1 | h1-eth0 | 10.0.0.1 | /8 |
| h2 | h2-eth0 | 10.0.0.1 | /8 |
| h3 | h2-eth0 | 10.0.0.2 | /8 |
| h4 | h4-eth0 | 10.0.0.2 | /8 |
| d1 | d1-eth0 | 192.168.1.10 | /24 |
| d2 | d2-eth0 | 192.168.2.10 | /24 |

## 2.2 Loading the 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 such as configuring network elements (i.e., IP addresses, default gateway), saving the topology, and exporting a layer 2 model.

**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`.

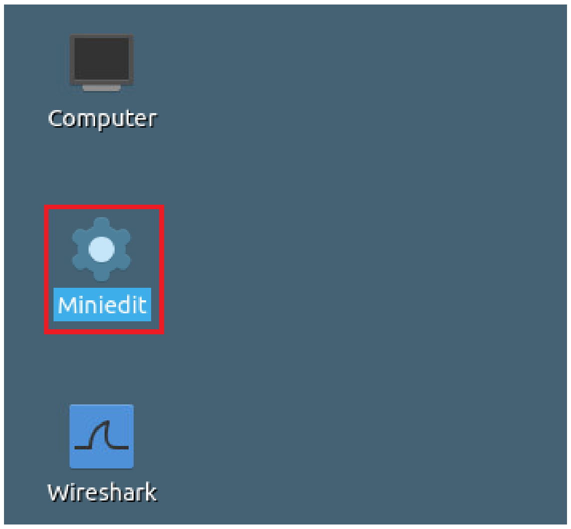
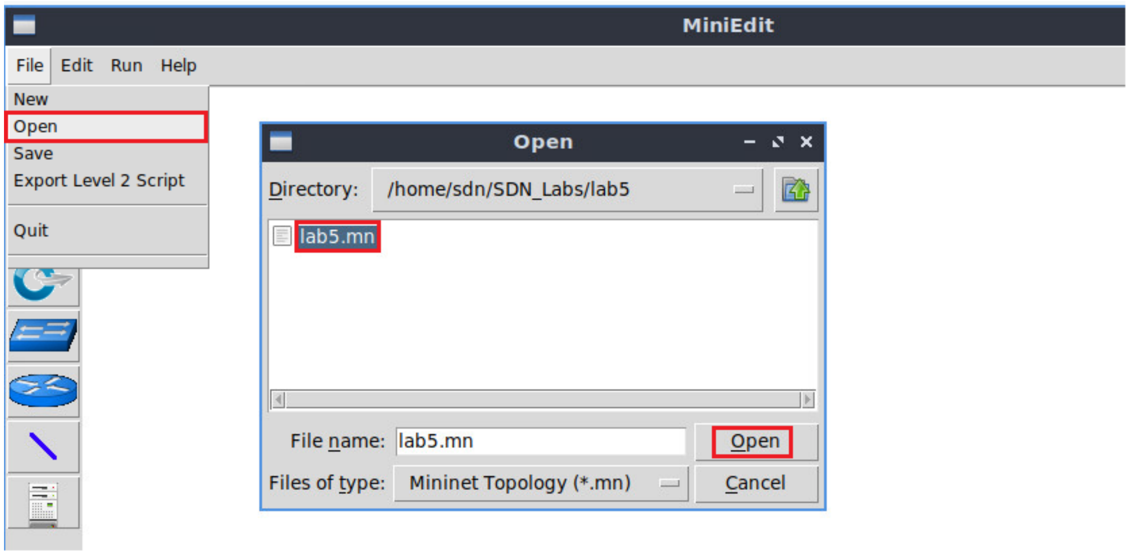


Figure 4. MiniEdit shortcut.

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



Directory: /home/sdn/SDN_Labs/lab5

lab5.mn

File name: lab5.mn

Files of type: Mininet Topology (*.mn)

Figure 5. Opening topology.

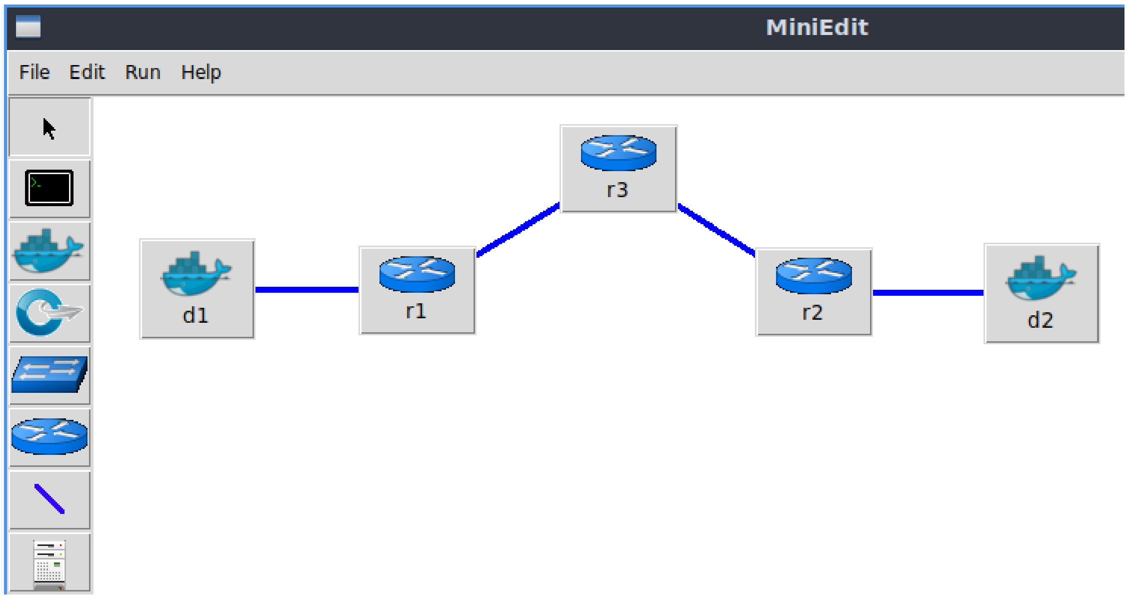


Figure 6. MiniEdit's topology.

## 2.3 Load the configuration file

At this point, the topology is loaded. However, the interfaces are not configured. In order to assign IP addresses to the interfaces of the devices, you will execute a script that loads the configuration to the routers.

**Step 1.** Click on the icon below to open the Linux terminal.

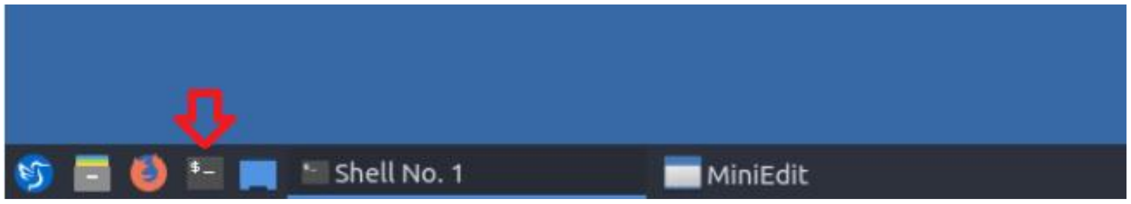
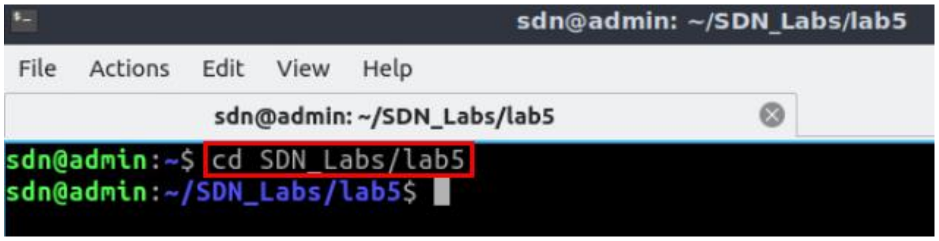


Figure 7. Opening Linux terminal.

**Step 2.** Click on the Linux terminal and navigate into *SDN_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 IP addresses to the interfaces of the routers. The `cd` command is short for the change directory followed by an argument that specifies the destination directory.

```
cd SDN_Labs/lab5
```

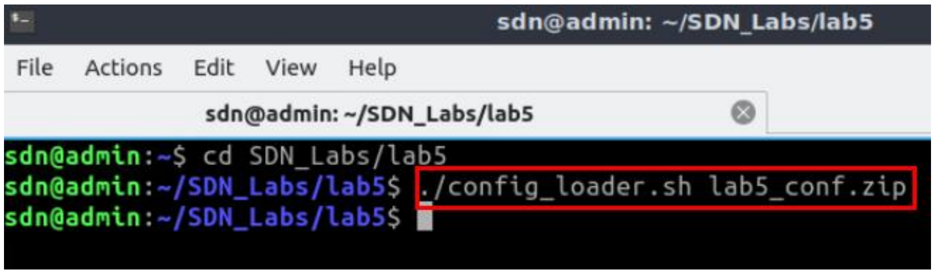

sdn@admin: ~/SDN_Labs/lab5
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab5

```
sdn@admin:~$ cd SDN_Labs/lab5
sdn@admin:~/SDN_Labs/lab5$
```

Figure 8. Entering the *SDN_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 routers in the topology.

```
./config_loader.sh lab5_conf.zip
```

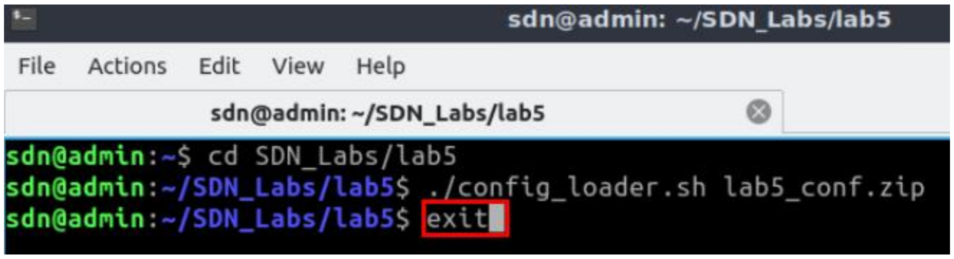

sdn@admin: ~/SDN_Labs/lab5
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab5

```
sdn@admin:~$ cd SDN_Labs/lab5
sdn@admin:~/SDN_Labs/lab5$ ./config_loader.sh lab5_conf.zip
sdn@admin:~/SDN_Labs/lab5$
```

Figure 9. Executing the shell script to load the configuration.

**Step 4.** Type the following command to exit the Linux terminal.

```
exit
```


sdn@admin: ~/SDN_Labs/lab5
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab5

```
sdn@admin:~$ cd SDN_Labs/lab5
sdn@admin:~/SDN_Labs/lab5$ ./config_loader.sh lab5_conf.zip
sdn@admin:~/SDN_Labs/lab5$ exit
```

Figure 10. Exiting from the terminal.

## 2.4 Run the emulation

In this section, you will run the emulation and check the links and interfaces that connect the devices in the given topology.

**Step 1.** To proceed with the emulation, click the *Run* button located on the lower left-hand side.

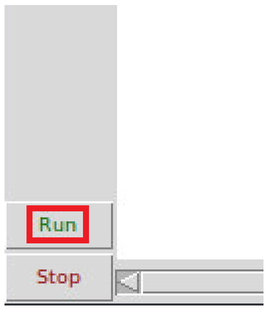
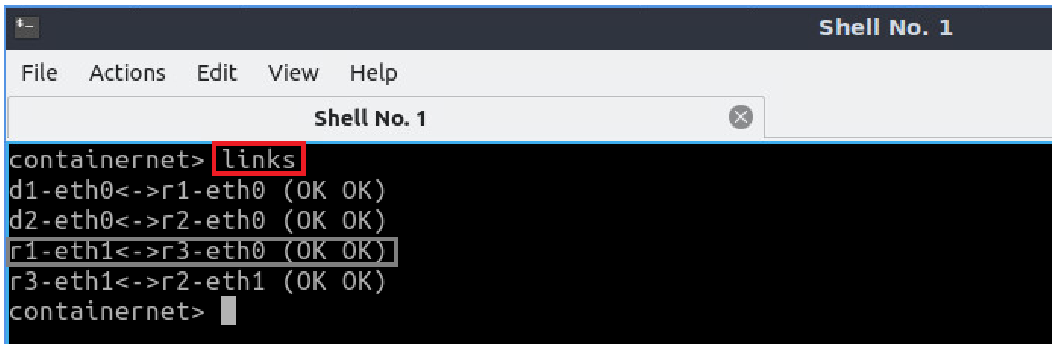


Figure 11. Starting the emulation.

**Step 2.** Issue the following command on the Mininet terminal to display the interface names and connections.

```
links
```



```
containernet> links
d1-eth0<->r1-eth0 (OK OK)
d2-eth0<->r2-eth0 (OK OK)
r1-eth1<->r3-eth0 (OK OK)
r3-eth1<->r2-eth1 (OK OK)
containernet>
```

Figure 12. Displaying network interfaces.

In Figure 12, the link displayed within the gray box indicates that interface *eth1* of router r1 connects to interface *eth0* of router r3 (i.e., *r1-eth1<->r3-eth0*).

## 2.5 Verify the configuration

You will verify the IP addresses listed in Table 2 and inspect the routing table of routers r1, r2, and r3.

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

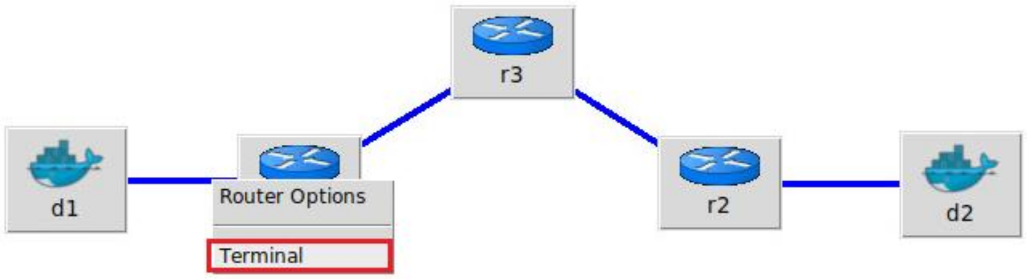
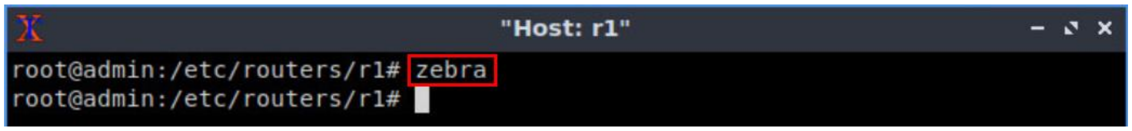


Figure 13. Opening a terminal on router r1.

**Step 2**. In this step, you will start the zebra daemon, 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 zebra, type the following command.

```
zebra
```

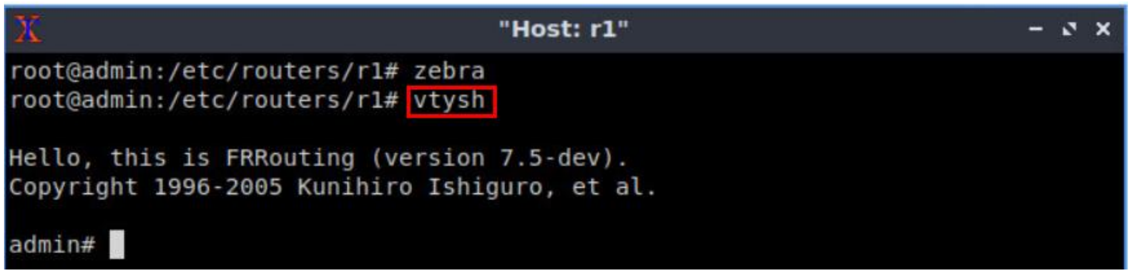


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1#
```

Figure 14. Starting zebra daemon.

**Step 3**. After initializing zebra, vtysh must be started, vtysh is the command-line interface (CLI) used to configure the router. To proceed, issue the following command.

```
vtysh
```

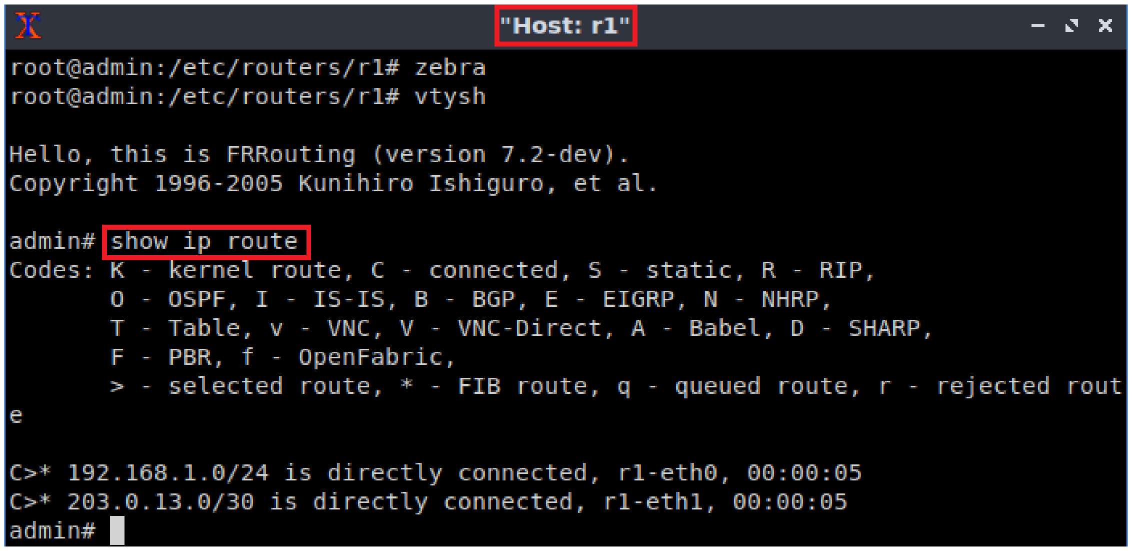


```
"Host: r1"
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.5-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin#
```

Figure 15. Starting vtysh on router r1.

**Step 4.** Type the following command on router r1 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 external networks as there is no routing protocol configured yet.

```
show ip route
```



```
"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# 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 rout
e

C>* 192.168.1.0/24 is directly connected, r1-eth0, 00:00:05
C>* 203.0.13.0/30 is directly connected, r1-eth1, 00:00:05
admin#
```

Figure 16. Displaying routing table of router r1.

The output in the figure above shows that the networks 192.168.1.0/24 and 203.0.13.0/30 are directly connected through the interfaces *r1-eth0* and *r1-eth1*, respectively.

**Step 5.** Hold right-click on router r2 and select *Terminal*.

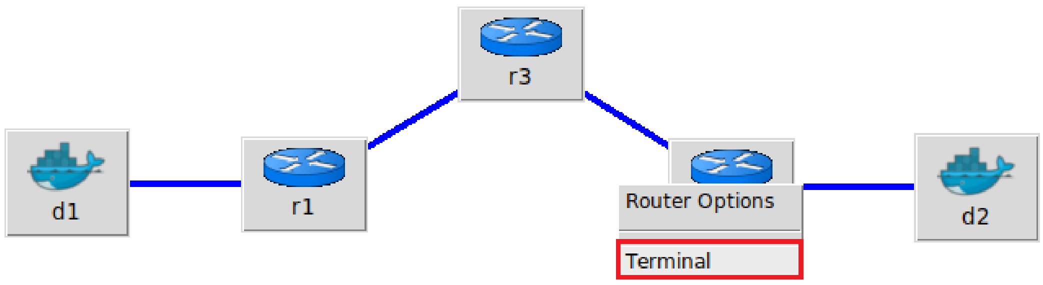
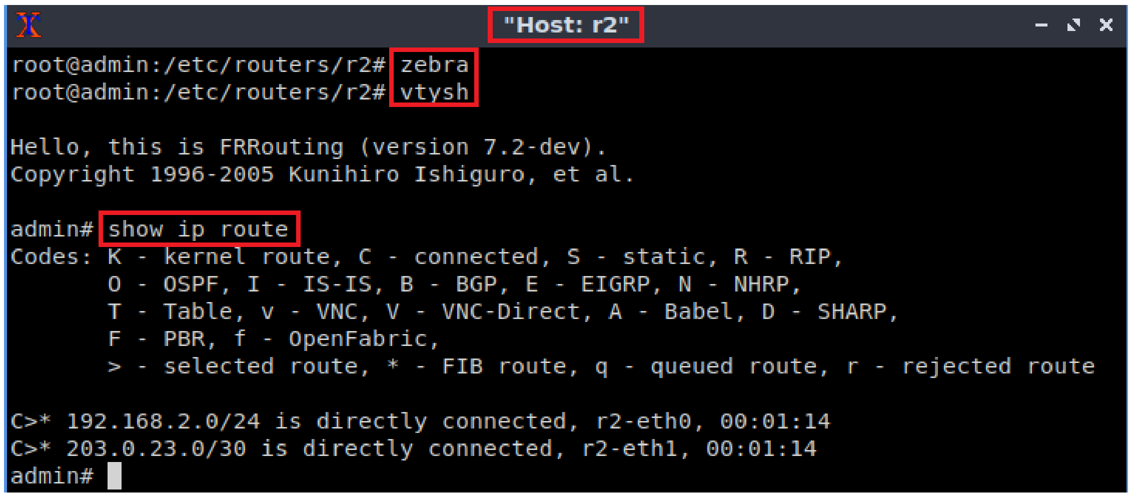


Figure 17. Opening a terminal on router r2.

**Step 6.** 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 terminal issue the commands depicted below. In 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,
       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

C>* 192.168.2.0/24 is directly connected, r2-eth0, 00:01:14
C>* 203.0.23.0/30 is directly connected, r2-eth1, 00:01:14
admin# 
```

Figure 18. Displaying routing table of router r2.

**Step 7.** Hold right-click on router r3 and select *Terminal*.

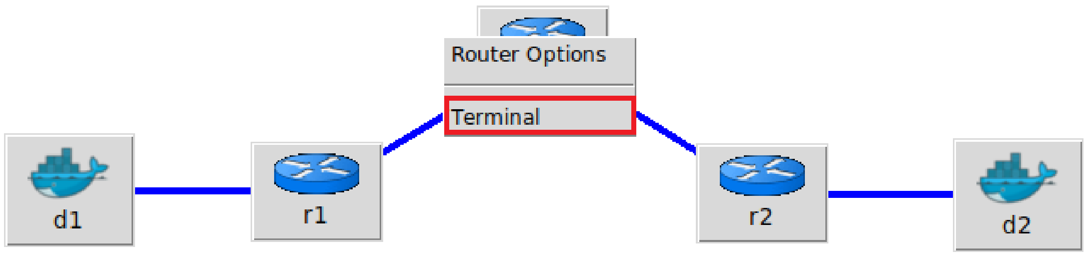
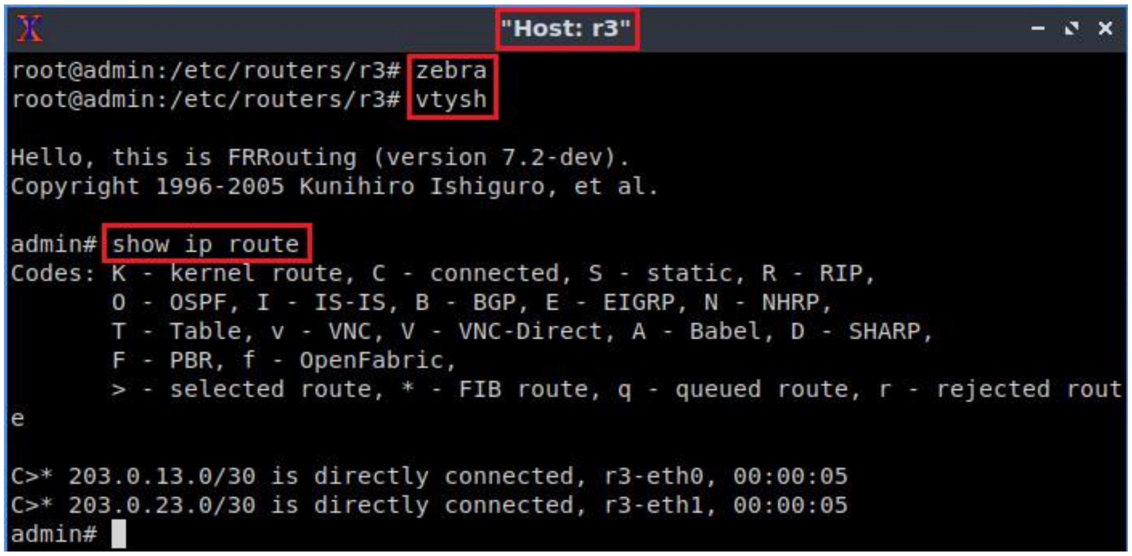


Figure 19. Opening a terminal on router r3.

**Step 8.** Router r3 is configured according to Table 2. Those steps are summarized in the following figure. To proceed, in router r3 terminal issue the commands depicted below. In the end, you will verify all the directly connected networks of router 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,
       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 rout
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#
```

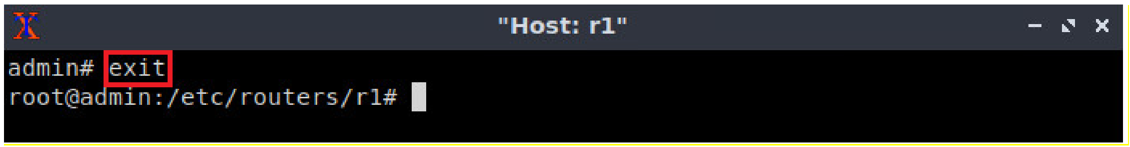
Figure 20. Displaying routing table of router r3.

## 3 Configuring OSPF on routers r1, r2, and r3

In this section, you will configure the OSPF routing protocol in routers r1, r2, and r3. First, you will enable the OSPF daemon on the routers. Second, you will establish a single-area OSPF, which is classified as area 0 or backbone area. Finally, you will advertise all the connected networks.

**Step 1.** To configure the OSPF routing protocol, you need to enable the OSPF daemon first. In router r1, type the following command to exit the vtysh session.

```
exit
```

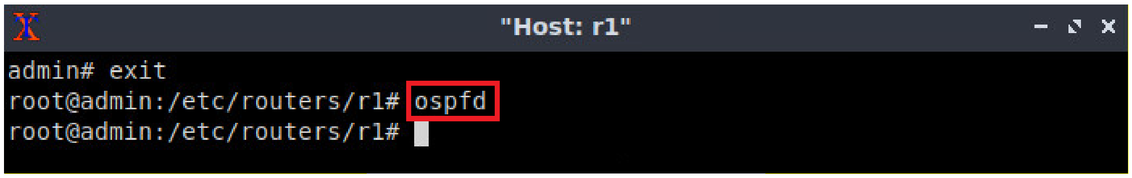


```
admin# exit
root@admin:/etc/routers/r1#
```

Figure 21. Exiting the vtysh session.

**Step 2.** Type the following command on the router r1 terminal to enable the OSPF daemon.

```
ospfd
```

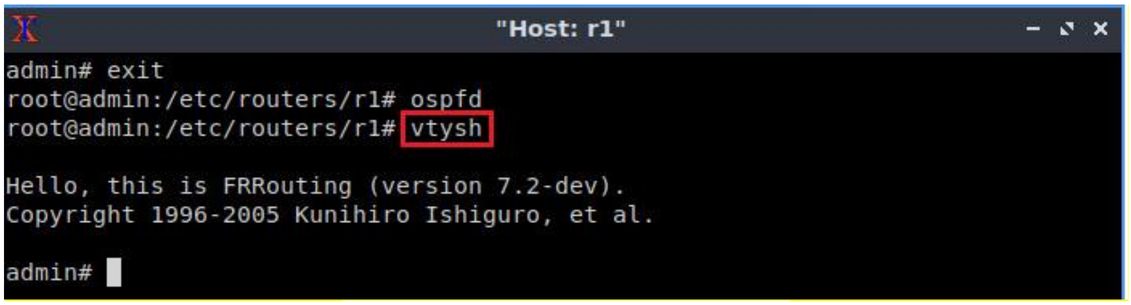


```
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1#
```

Figure 22. Starting OSPF daemon.

**Step 3.** In order to enter to router r1 terminal, issue the following command.

```
vtysh
```

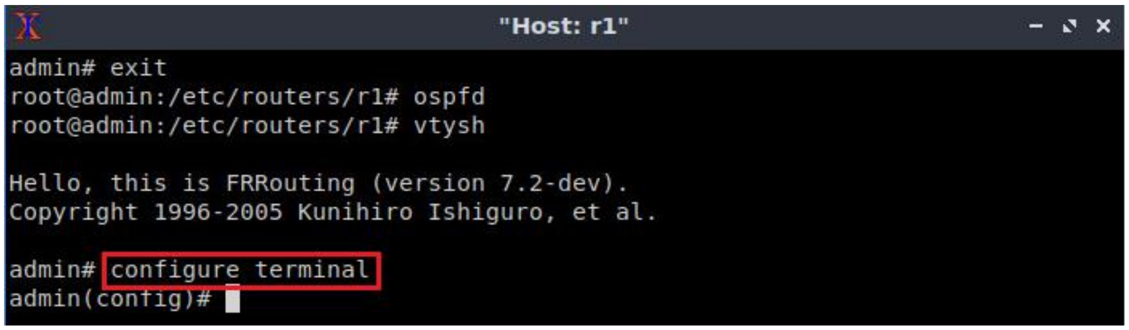


```
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

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

admin#
```

Figure 23. Starting vtysh on router r1.

**Step 4.** To enable router r1 global configuration mode, issue the following command.

```
configure terminal
```

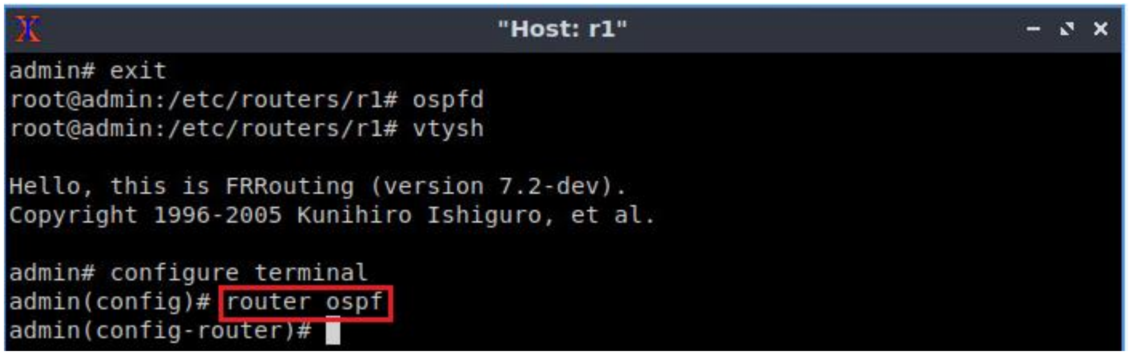


```
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

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

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

Figure 24. Enabling configuration mode on router r1.

**Step 5.** In order to configure the OSPF routing protocol, type the command shown below. This command enables OSPF configuration mode where you advertise the networks directly connected to router r1.

```
router ospf
```

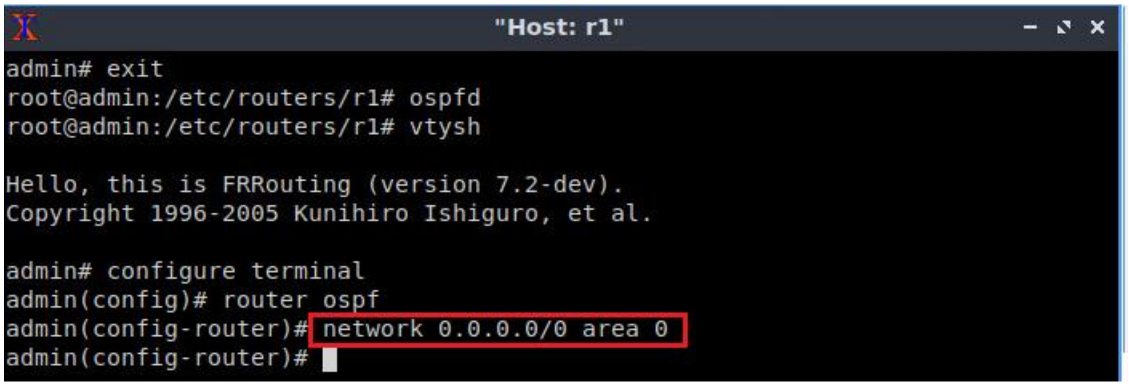


```
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

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 25. Configuring OSPF on router r1.

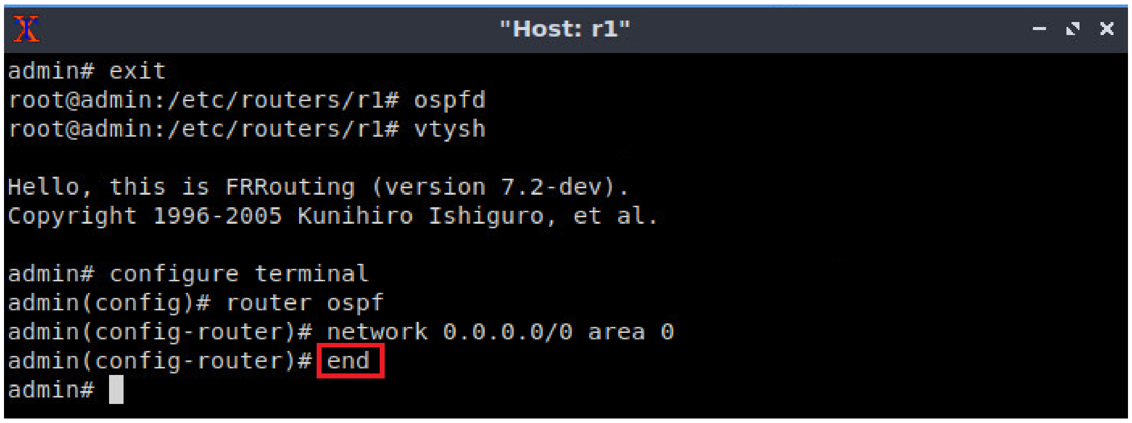
**Step 6.** In this step, you will enable all the interfaces of the router r1 to participate in the OSPF routing process, i.e., all the attached networks will be advertised to OSPF neighbors. The advertised networks have area 0. To advertise all connected networks, issue the following command.

```
network 0.0.0.0/0 area 0
```

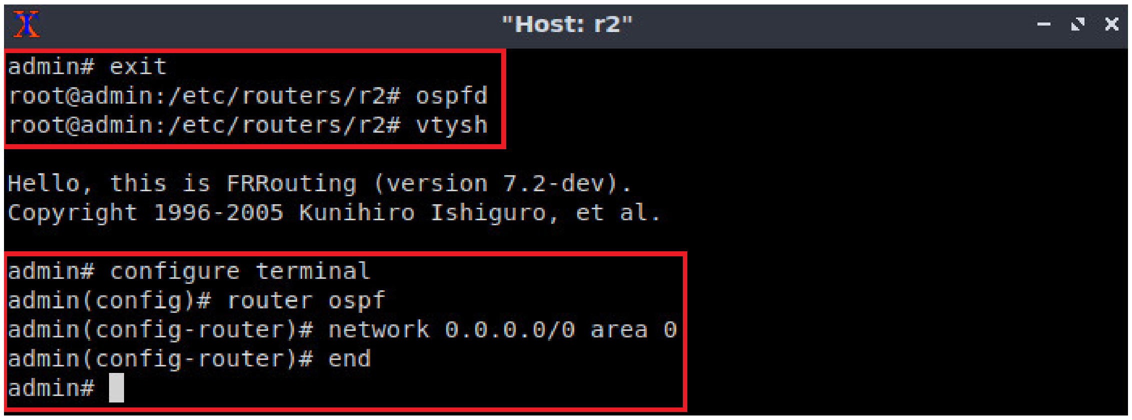

```
"Host: r1"
admin# exit
root@admin:/etc/routers/r1# ospfd
root@admin:/etc/routers/r1# vtysh

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 26. Enabling all the interfaces of router r1 to participate in the OSPF routing process.

**Step 7.** Type the following command to exit from the configuration mode.

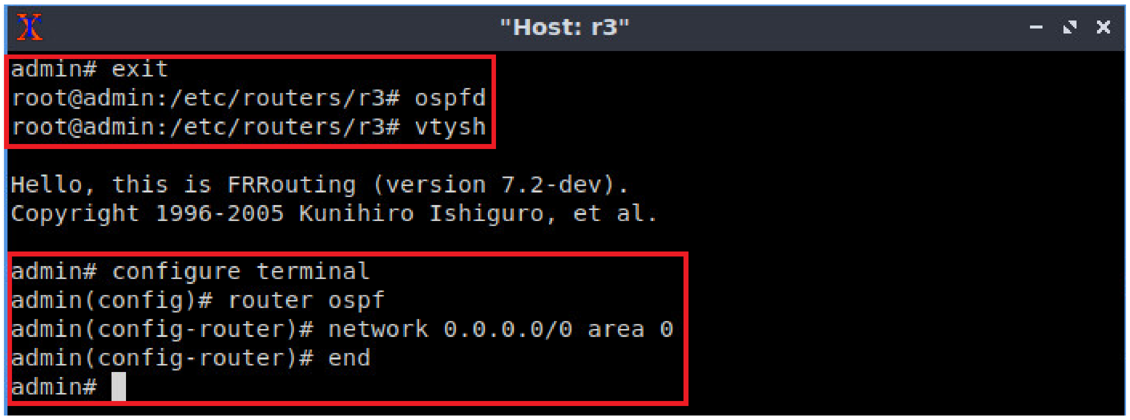
```
end
```


```
"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 27. Exiting from the configuration mode.

**Step 8.** Router r2 is configured similarly to router r1. Those steps are summarized in the following figure. To proceed, on route r2 terminal, issue the commands depicted 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 28. Summary of router r2 configuration.

**Step 9.** Router r3 is configured similarly to router r2. Those steps are summarized in the following figure. To proceed, on route r3 terminal, issue the commands depicted below.

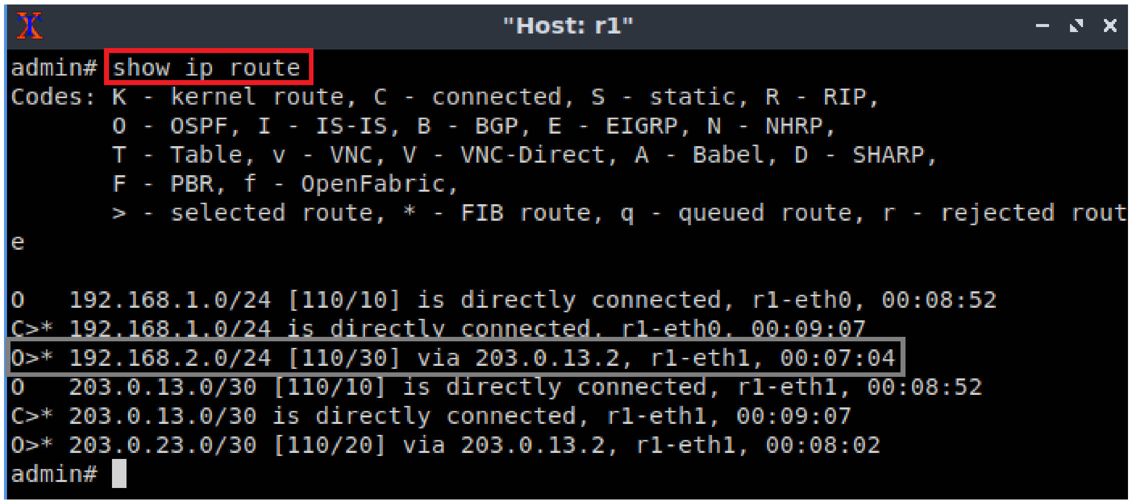


```
admin# exit
root@admin:/etc/routers/r3# ospfd
root@admin:/etc/routers/r3# vtysh

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 29. Summary of router r3 configuration.

**Step 10.** Navigate into router r1 terminal and, type the following command to verify the routing table of router r1.

```
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 rout
e

O   192.168.1.0/24 [110/10] is directly connected, r1-eth0, 00:08:52
C>* 192.168.1.0/24 is directly connected, r1-eth0, 00:09:07
O>* 192.168.2.0/24 [110/30] via 203.0.13.2, r1-eth1, 00:07:04
O   203.0.13.0/30 [110/10] is directly connected, r1-eth1, 00:08:52
C>* 203.0.13.0/30 is directly connected, r1-eth1, 00:09:07
O>* 203.0.23.0/30 [110/20] via 203.0.13.2, r1-eth1, 00:08:02
admin#
```

Figure 30. Verifying the routing table of router r1.

Consider the figure above. The network 192.168.2.0/24 is learned via OSPF (`O>*`) and it is reachable via the next hop 203.0.13.2 (router r3).

## 4 Configuring VXLAN

In this section, you will start the networks within containers d1 and d2. Both containers run a Mininet topology, as depicted in Figure 3. In container d1, the topology consists of two end-hosts (h1 and h2) connected to a switch (s1). Similarly, container d2 runs a topology with two end-hosts (h3 and h4) connected to a switch (s2). The end-hosts within the containers will be isolated by using VXLAN.

Note that the containers d1 and d2 emulate a multi-tenant environment. Multi-tenancy is a mode of operation where multiple independent instances such as end-hosts (see Figure 3) of a tenant operate in a shared environment while ensuring logical segmentation between the instances. A tenant could be a business entity, user group, applications, or cloud services. The tenant instances such as h1, h2, h3, and h4 are logically isolated but physically operate on the same fabric.

## 4.1 Run Mininet instances within the containers

The following section shows the steps to run a Mininet topology within the containers and how to navigate through the configuration files.

**Step 1.** Hold right-click on container d1 and select *Terminal* as shown below. A window will appear.

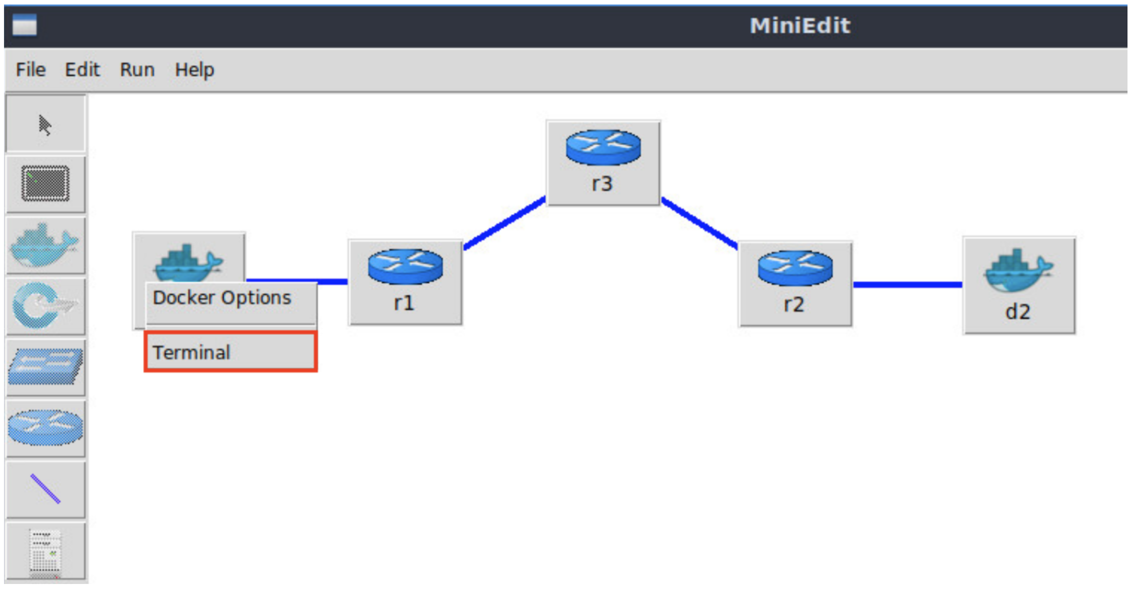
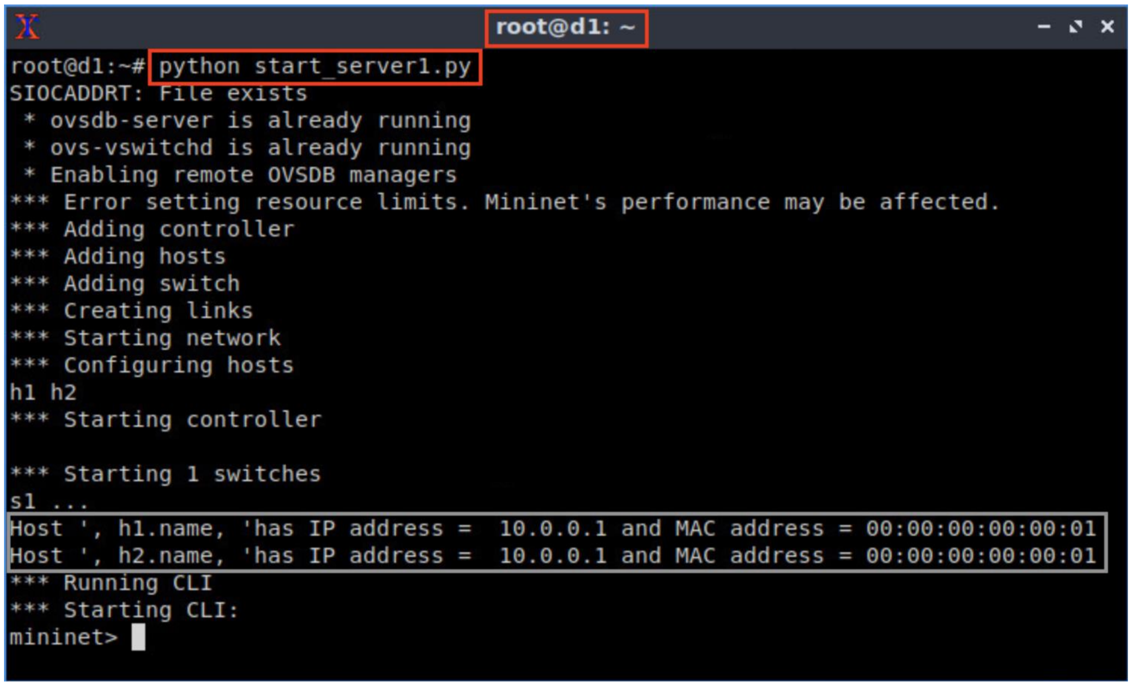


Figure 31. Opening container's d1 Terminal.

**Step 2.** In container d1 terminal, execute the following python script to start a Mininet instance that consists of two end-hosts connected to a switch.

```
python start_server1.py
```



```
root@d1: ~
root@d1:~# python start_server1.py
SIOCADDRT: File exists
 * ovsdb-server is already running
 * ovs-vswitchd is already running
 * 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 and MAC address = 00:00:00:00:00:01
Host ', h2.name, 'has IP address =  10.0.0.1 and MAC address = 00:00:00:00:00:01
*** Running CLI
*** Starting CLI:
mininet> 
```

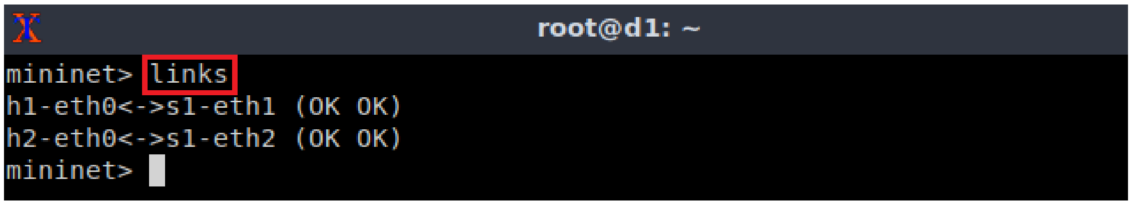
Figure 32. Starting a Mininet instance within container d1.

The figure above starts a Mininet instance in container d1. Also, the information about the end-hosts is summarized after starting switch s1.

Notice that host h1 and host h2 have the same IP addresses and MAC addresses. They will be isolated using VXLAN.

**Step 3.** In container d1, run the following command to verify the devices in the topology.

```
links
```



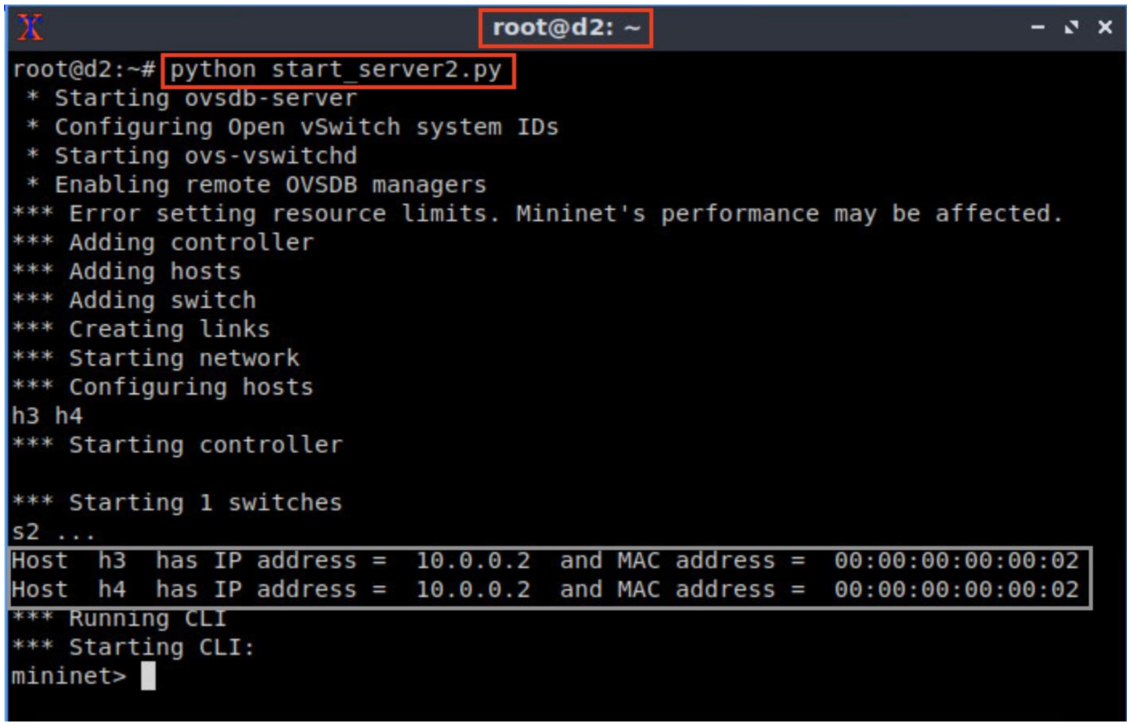
```
root@d1: ~
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
mininet> 
```

Figure 33. Verifying the links between the devices in container d1.

The figure above shows that the host h1 and switch s1 are connected via the interface pair *h1-eth0<->s1-eth1*. Similarly, host h2 is connected to the switch s1 (*h2-eth0<->s1-eth2*).

**Step 4.** Similarly, in container d2 terminal, execute the following python script to start a Mininet instance that consists of two end-hosts connected to a switch as well.

```
python start_server2.py
```



```
root@d2: ~
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 =  10.0.0.2  and MAC address =  00:00:00:00:00:02
Host  h4  has IP address =  10.0.0.2  and MAC address =  00:00:00:00:00:02
*** Running CLI
*** Starting CLI:
mininet> 
```

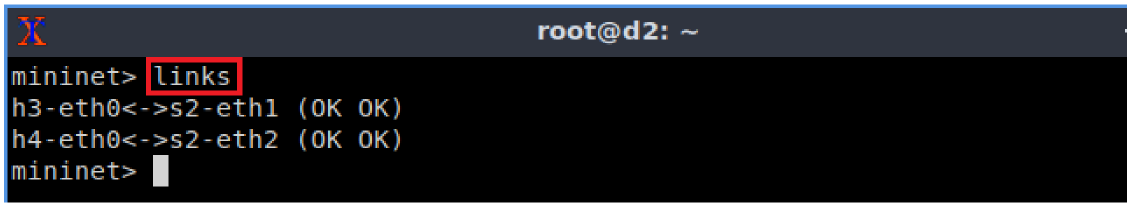
Figure 34. Starting a Mininet instance within container d2.

The figure above starts a Mininet instance in container d2. Also, the information about the hosts is summarized after starting switch s2.

Notice that host h3 and host h4 have the same IP addresses and MAC addresses. These hosts will be isolated b using VXLAN.

**Step 5.** In container d2 terminal, run the following command to verify the devices in the topology.

```
links
```



```
root@d2: ~
mininet> links
h3-eth0<->s2-eth1 (OK OK)
h4-eth0<->s2-eth2 (OK OK)
mininet> 
```

Figure 35. Verifying the links between the devices in container d2.

The figure above shows that the host h3 and switch s2 are connected via the interface pair *h3-eth0<->s2-eth1*. Similarly, host h4 is connected to the switch s2 (*h4-eth0<->s2-eth2*).

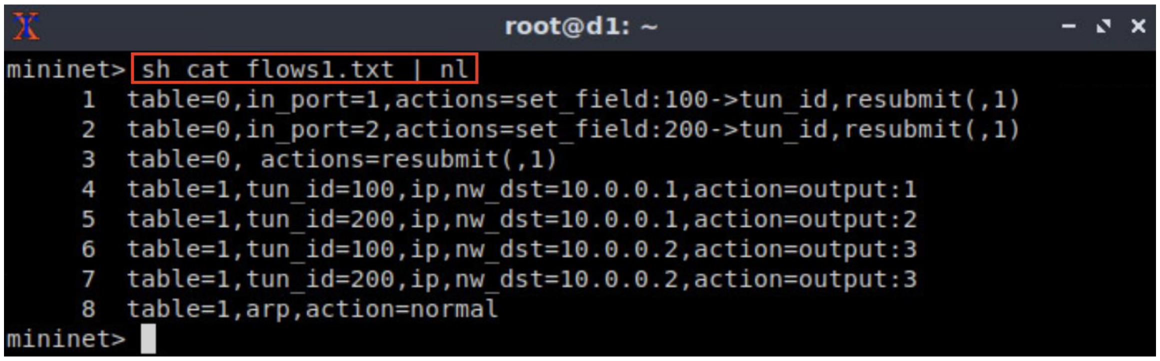
## 4.2 Adding entries to the flow tables of the switches

In this section, you will add entries to the flow tables of switch s1 and switch s2. These entries are added to a table that is responsible for traffic processing. In this lab, the flow tables specify the VXLAN tags and the actions to forward the packets to their right destination.

The main purpose of configuring VXLAN in this lab is to isolate the traffic from h1 to h3 and from h2 to h4.

**Step 1.** in container d1, type the following to visualize the entries of the flow table to be added to switch s1.

```
sh cat flows1.txt | nl
```



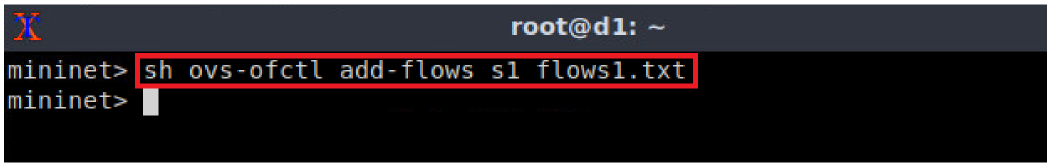
```
root@d1: ~
mininet> sh cat flows1.txt | nl
     1  table=0,in_port=1,actions=set_field:100->tun_id,resubmit(,1)
     2  table=0,in_port=2,actions=set_field:200->tun_id,resubmit(,1)
     3  table=0, actions=resubmit(,1)
     4  table=1,tun_id=100,ip,nw_dst=10.0.0.1,action=output:1
     5  table=1,tun_id=200,ip,nw_dst=10.0.0.1,action=output:2
     6  table=1,tun_id=100,ip,nw_dst=10.0.0.2,action=output:3
     7  table=1,tun_id=200,ip,nw_dst=10.0.0.2,action=output:3
     8  table=1,arp,action=normal
mininet> 
```

Figure 36. Flow table in container d1.

The file contains entries for two tables (i.e., table 0 and table 1). The first three lines correspond to table 0. In line 1 and 2, the switch is instructed to add the VNID as a function of the ingress ports. Line 3 specifies that all packets from table 0 must be resubmitted to table 1. Table 1 (i.e., from line 4 to line 8) specifies the output of a packet depending on its destination IP address and its VNIC. Line 8 enables ARP.

**Step 2.** In container d1**,** type the following command to add entries to the flow table of switch s1.

```
sh ovs-ofctl add-flows s1 flows1.txt
```

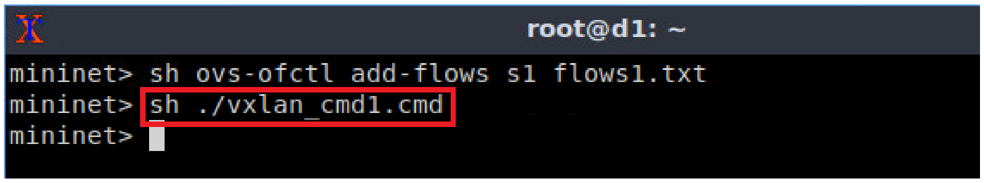


```
root@d1: ~
mininet> sh ovs-ofctl add-flows s1 flows1.txt
mininet> 
```

Figure 37. Adding flow entries to switch s1.

**Step 3.** In this step, you will configure a VTEP 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 ./vxlan_cmd1.cmd
```



```
root@d1: ~
mininet> sh ovs-ofctl add-flows s1 flows1.txt
mininet> sh ./vxlan_cmd1.cmd
mininet> 
```

Figure 38. Enabling outgoing traffic in switch s1.

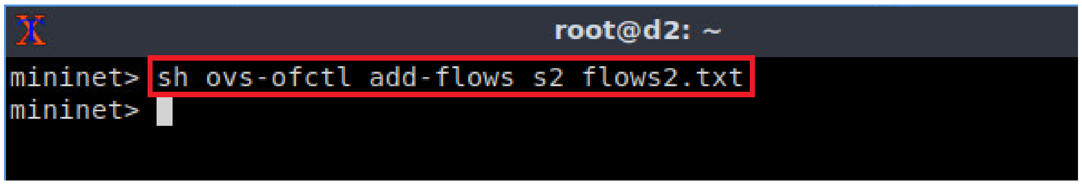
The script above executes the following command:

ovs-vsctl add-port s1 vtep -- set interface vtep type=vxlan option:
remote_ip=192.168.2.10 option:key=flow ofport_request=10

VTEP is the entity responsible for encapsulating and de-encapsulating layer 2 traffic. It handles the connection between the overlay and the underlay network. In this case, the VTEP is configured to transfer packets between the switches and the container's interface.

**Step 4.** In container d2, issue the following command to add entries to the flow table of switch s2.

```
sh ovs-ofctl add-flows s2 flows2.txt
```

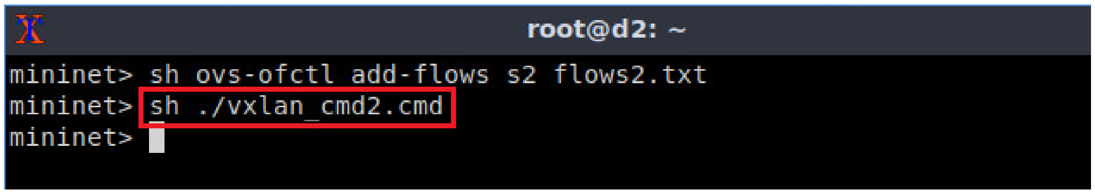


```
root@d2: ~
mininet> sh ovs-ofctl add-flows s2 flows2.txt
mininet> 
```

Figure 39. Adding flow entries to switch s2.

**Step 5.** Similarly, in container d2, type the command below to configure a VTEP in order that enables outgoing traffic from switch s2 to the outer network.

```
sh ./vxlan_cmd2.cmd
```



```
root@d2: ~
mininet> sh ovs-ofctl add-flows s2 flows2.txt
mininet> sh ./vxlan_cmd2.cmd
mininet> 
```

Figure 40. Enabling outgoing traffic in switch s2.

The script above executes the following command:

ovs-vsctl add-port s2 vtep -- set interface vtep type=vxlan option:
remote_ip=192.168.1.10 option:key=flow ofport_request=10

## 5 Verifying configuration

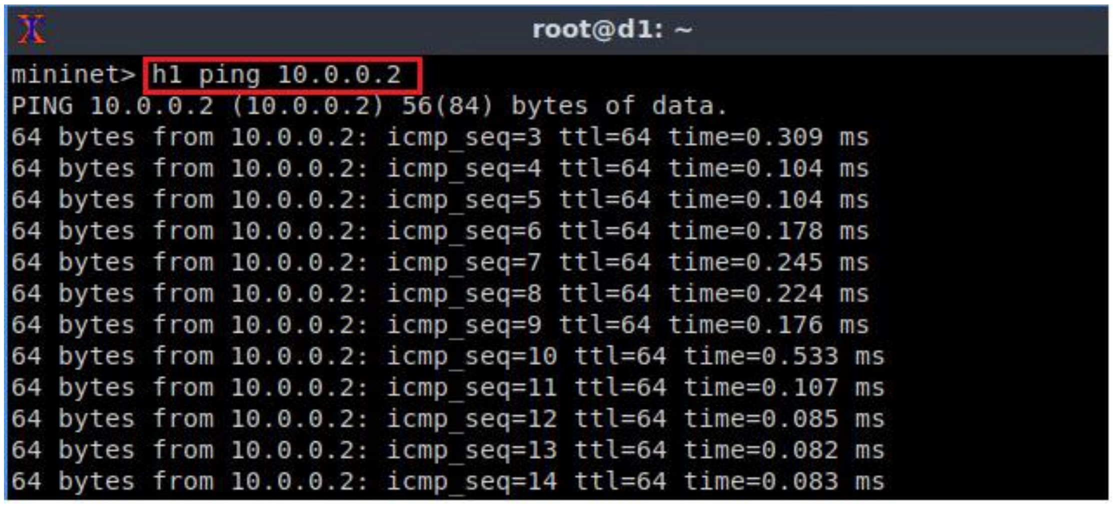
In this section, you will verify that the VXLAN tags were applied correctly. You will also notice that the traffic between h1 and h3 has the VXLAN tag 100 and the traffic between h2 and h4 has the VXLAN tag 200. This tag is known as the VNI, which is used to identify VXLAN traffic.

## 5.1 Performing connectivity test between end-hosts

The following steps aim to verify the connectivity between end-hosts. This means that there should be connectivity between h1 and h3, as well as between hosts h2 and h4.

**Step 1.** In container d1 terminal, issue the following command to verify the connectivity between host h1 and host h3. Notice that `h1` specifies host 1 as the source.

```
h1 ping 10.0.0.2
```



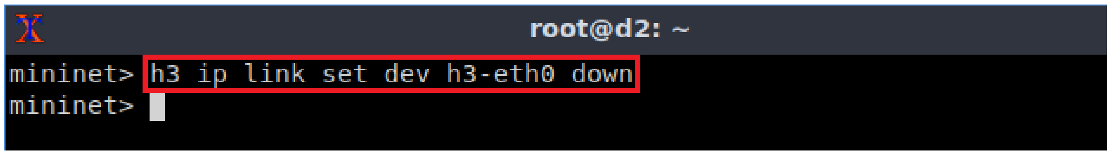
```
root@d1: ~
mininet> h1 ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.309 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.104 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.104 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.178 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.245 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.224 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.176 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.533 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.107 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.085 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.082 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.083 ms
```

Figure 41. Performing a connectivity test between host h1 and host h3.

Consider the figure above. The results show a successful connectivity test.

**Step 2.** In container d2 terminal, issue the command shown below to disable the network interface of host h3.

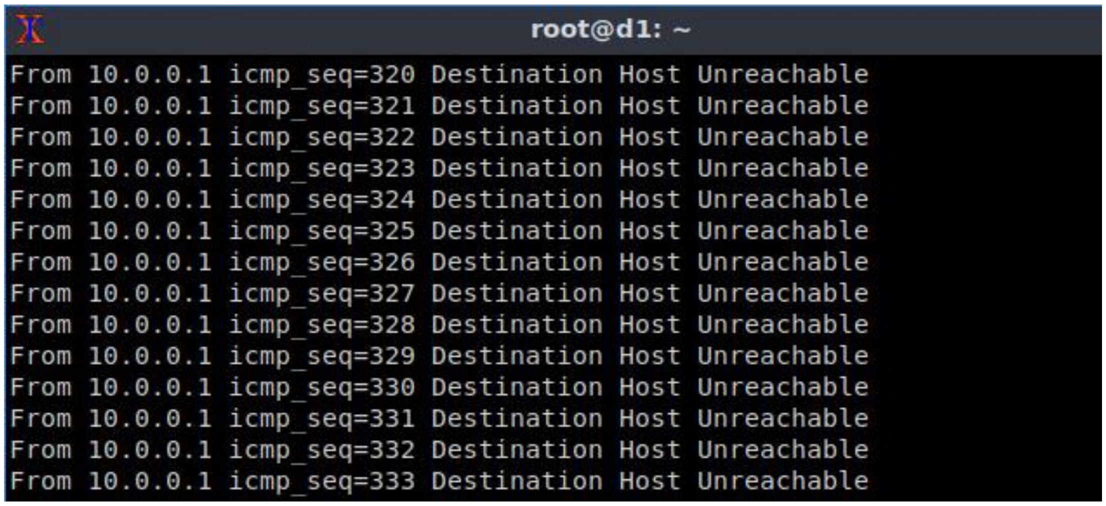
```
h3 ip link set dev h3-eth0 down
```



```
root@d2: ~
mininet> h3 ip link set dev h3-eth0 down
mininet>
```

Figure 42. Disabling h3 network interface.

**Step 3.** Click on container d1 terminal. You will verify that the connectivity is lost. Press `Ctrl+c` to stop the test.

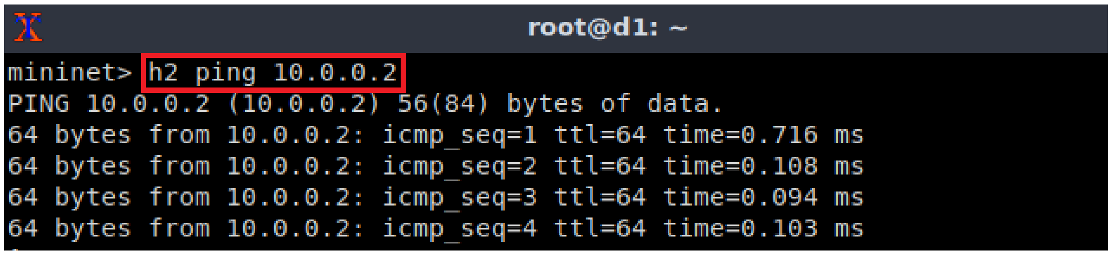


```
root@d1: ~
From 10.0.0.1 icmp_seq=320 Destination Host Unreachable
From 10.0.0.1 icmp_seq=321 Destination Host Unreachable
From 10.0.0.1 icmp_seq=322 Destination Host Unreachable
From 10.0.0.1 icmp_seq=323 Destination Host Unreachable
From 10.0.0.1 icmp_seq=324 Destination Host Unreachable
From 10.0.0.1 icmp_seq=325 Destination Host Unreachable
From 10.0.0.1 icmp_seq=326 Destination Host Unreachable
From 10.0.0.1 icmp_seq=327 Destination Host Unreachable
From 10.0.0.1 icmp_seq=328 Destination Host Unreachable
From 10.0.0.1 icmp_seq=329 Destination Host Unreachable
From 10.0.0.1 icmp_seq=330 Destination Host Unreachable
From 10.0.0.1 icmp_seq=331 Destination Host Unreachable
From 10.0.0.1 icmp_seq=332 Destination Host Unreachable
From 10.0.0.1 icmp_seq=333 Destination Host Unreachable
```

Figure 43. Verifying connectivity between host h1 and host h3.

**Step 4.** In container d1 terminal, issue the following command to test the connectivity between host h2 and host h4. Notice that `h2` specifies host h2 as the source.

```
h2 ping 10.0.0.2
```



```
root@d1: ~
mininet> h2 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=0.716 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.094 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.103 ms
```

Figure 44. Performing a connectivity test between host h2 and host h4.

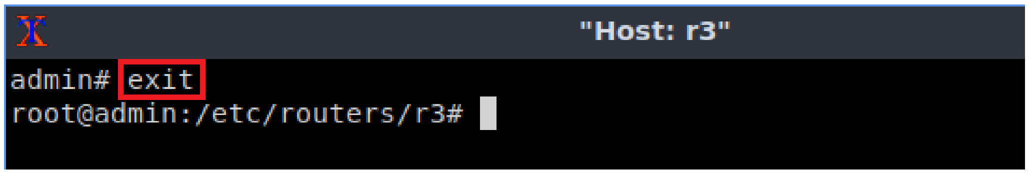
The results will display a successful connectivity test. Do not stop the connectivity test.

## 5.2 Verifying VXLAN network identifiers using Wireshark

The following steps show how to verify VXLAN network identifiers using the Wireshark network analyzer. The identifiers are used by the switch to isolate network traffic.

**Step 1.** Click on router r3 terminal and issue the following command to exit the vtysh session.

```
exit
```

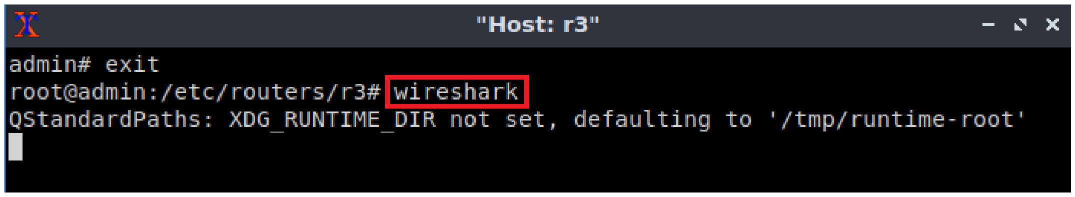


```
"Host: r3"
admin# exit
root@admin:/etc/routers/r3#
```

Figure 45. Exiting from vtysh.

**Step 2.** In router r3 terminal, start Wireshark by issuing the following command. A new window will emerge.

```
wireshark
```

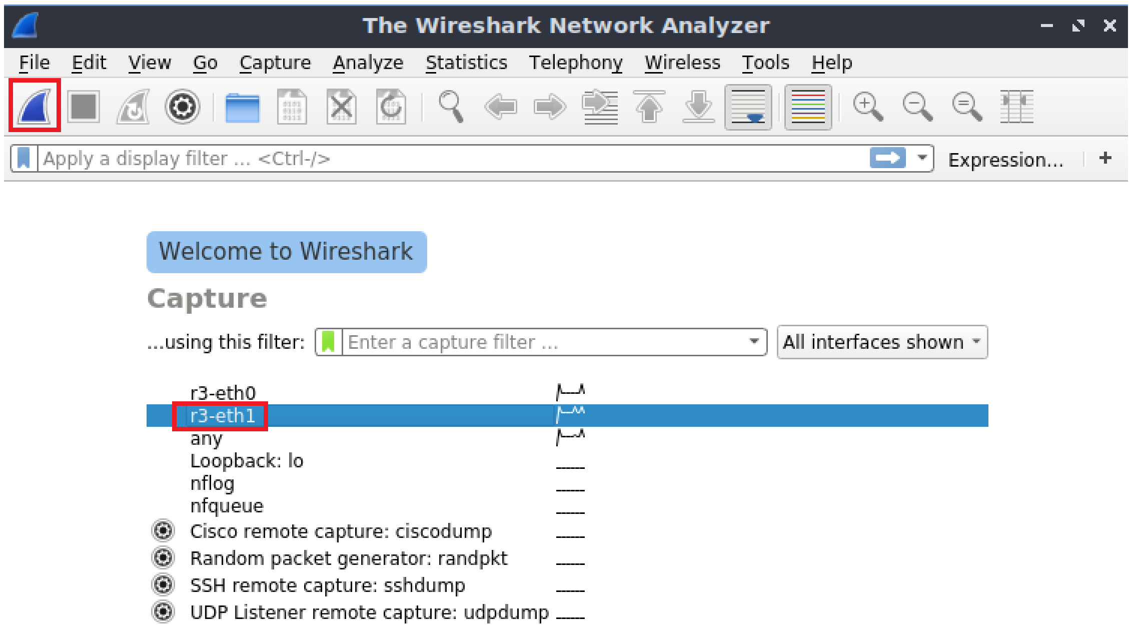


```
admin# exit
root@admin:/etc/routers/r3# wireshark
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

Figure 46. Starting Wireshark network analyzer.

After executing the above command on the router r3 terminal, the Wireshark window will open, where you monitor different interfaces related to router r3.

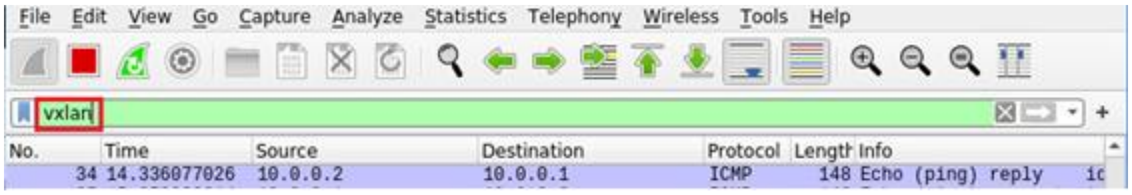
**Step 3**. Click on interface *r3-eth1* then, click the blue 'shark fin' icon located on the upper left-hand side to start capturing packets on this interface.


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
All interfaces shown

r3-eth0
r3-eth1
any
Loopback: lo
nflog
nfqueue
Cisco remote capture: ciscodump
Random packet generator: randpkt
SSH remote capture: sshdump
UDP Listener remote capture: udpdump

Figure 47. Starting packet capturing on interface *r3-eth0*.

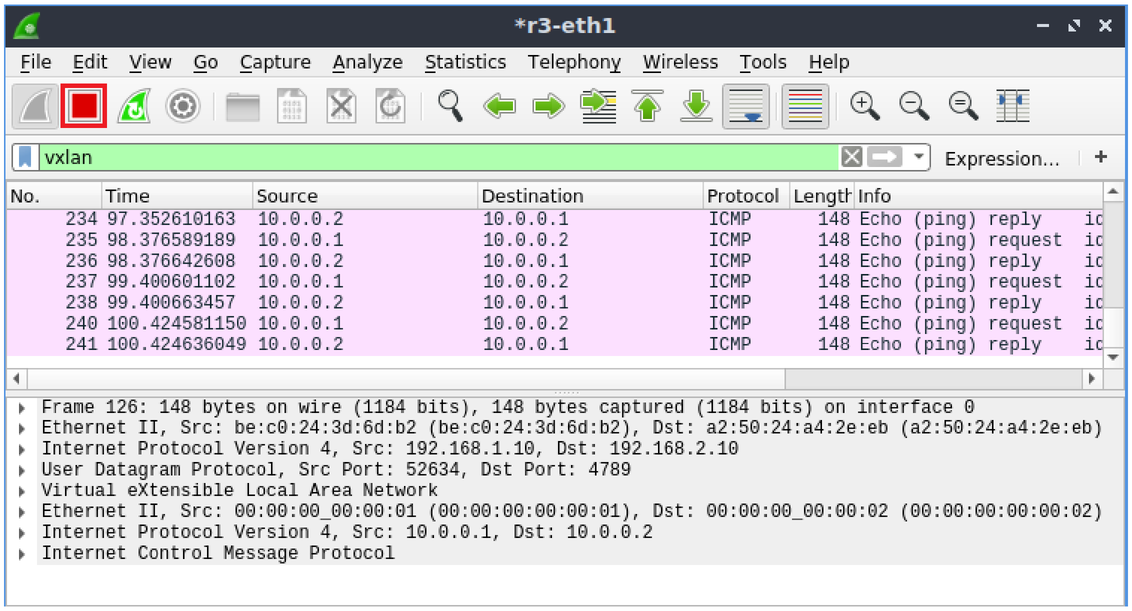
**Step 4.** In the filter box located on the upper left-hand side, type *vxlan* to filter the packets containing VXLAN tags.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
vxlan

```
No.   Time          Source     Destination   Protocol  Length Info
   34 14.336077026  10.0.0.2   10.0.0.1      ICMP        148 Echo (ping) reply    ic
```

Figure 48. Filtering network traffic.

**Step 5.** Stop the packet capturing by clicking the red stop button.

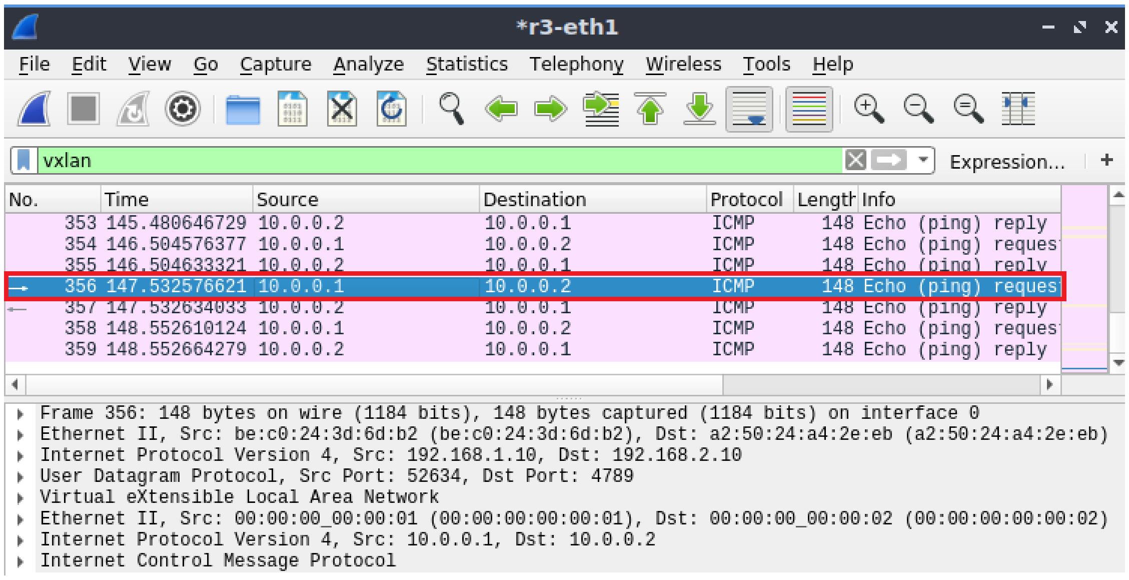

*r3-eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
vxlan
Expression...

```
No.  Time           Source     Destination  Protocol Length Info
 234 97.352610163   10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply    id
 235 98.376589189   10.0.0.1   10.0.0.2     ICMP     148 Echo (ping) request  id
 236 98.376642608   10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply    id
 237 99.400601102   10.0.0.1   10.0.0.2     ICMP     148 Echo (ping) request  id
 238 99.400663457   10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply    id
 240 100.424581150  10.0.0.1   10.0.0.2     ICMP     148 Echo (ping) request  id
 241 100.424636049  10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply    id
```

```
Frame 126: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
Ethernet II, Src: be:c0:24:3d:6d:b2 (be:c0:24:3d:6d:b2), Dst: a2:50:24:a4:2e:eb (a2:50:24:a4:2e:eb)
Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.2.10
User Datagram Protocol, Src Port: 52634, Dst Port: 4789
Virtual eXtensible Local Area Network
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
Internet Control Message Protocol
```

Figure 49. Stopping the packet capturing.

**Step 6.** Select a packet which source IP is 10.0.0.1 and destination IP is 10.0.0.2.

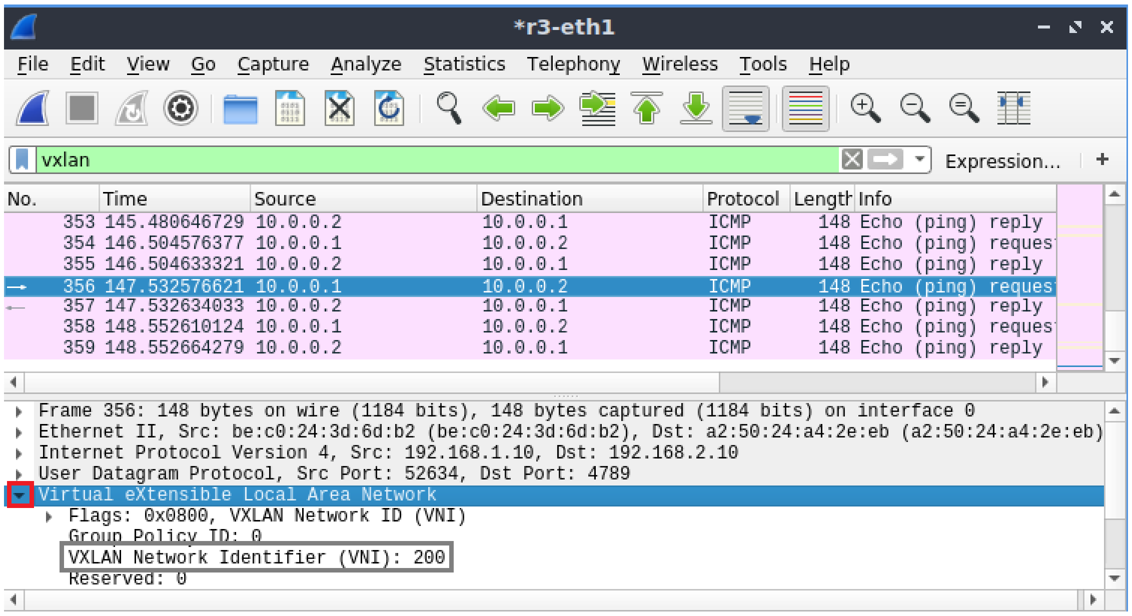

*r3-eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
vxlan
Expression...

```
No.  Time           Source     Destination  Protocol Length Info
 353 145.480646729  10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply
 354 146.504576377  10.0.0.1   10.0.0.2     ICMP     148 Echo (ping) reques
 355 146.504633321  10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply
 356 147.532576621  10.0.0.1   10.0.0.2     ICMP     148 Echo (ping) reques
 357 147.532634033  10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply
 358 148.552610124  10.0.0.1   10.0.0.2     ICMP     148 Echo (ping) reques
 359 148.552664279  10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply
```

```
Frame 356: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
Ethernet II, Src: be:c0:24:3d:6d:b2 (be:c0:24:3d:6d:b2), Dst: a2:50:24:a4:2e:eb (a2:50:24:a4:2e:eb)
Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.2.10
User Datagram Protocol, Src Port: 52634, Dst Port: 4789
Virtual eXtensible Local Area Network
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
Internet Control Message Protocol
```

Figure 50. Selecting a packet for further inspection.

**Step 7.** Click on the arrow located on the left most of the field called *Virtual eXtensible Local Area Network.* A list will be displayed. Verify that the *VXLAN Network Identifier* is 200. Notice that such tag corresponds to the traffic from h2 to h4.


*r3-eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
vxlan
Expression...

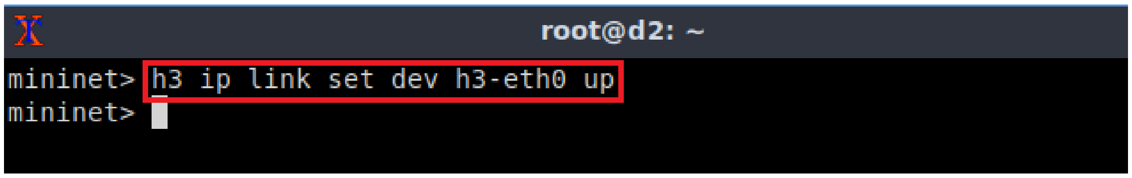
```
No.  Time           Source     Destination  Protocol Length Info
353 145.480646729 10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply
354 146.504576377 10.0.0.1   10.0.0.2     ICMP     148 Echo (ping) reques
355 146.504633321 10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply
356 147.532576621 10.0.0.1   10.0.0.2     ICMP     148 Echo (ping) reques
357 147.532634033 10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply
358 148.552610124 10.0.0.1   10.0.0.2     ICMP     148 Echo (ping) reques
359 148.552664279 10.0.0.2   10.0.0.1     ICMP     148 Echo (ping) reply

Frame 356: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
Ethernet II, Src: be:c0:24:3d:6d:b2 (be:c0:24:3d:6d:b2), Dst: a2:50:24:a4:2e:eb (a2:50:24:a4:2e:eb)
Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.2.10
User Datagram Protocol, Src Port: 52634, Dst Port: 4789
Virtual eXtensible Local Area Network
  Flags: 0x0800, VXLAN Network ID (VNI)
  Group Policy ID: 0
  VXLAN Network Identifier (VNI): 200
  Reserved: 0
```

Figure 51. Verifying VXLAN network identifier.

**Step 8.** In container d1, press `Ctrl+c` to stop the test.

**Step 9.** In the container d2 terminal, re-enable the network interface in host h3 by issuing the following command.

```
h3 ip link set dev h3-eth0 up
```

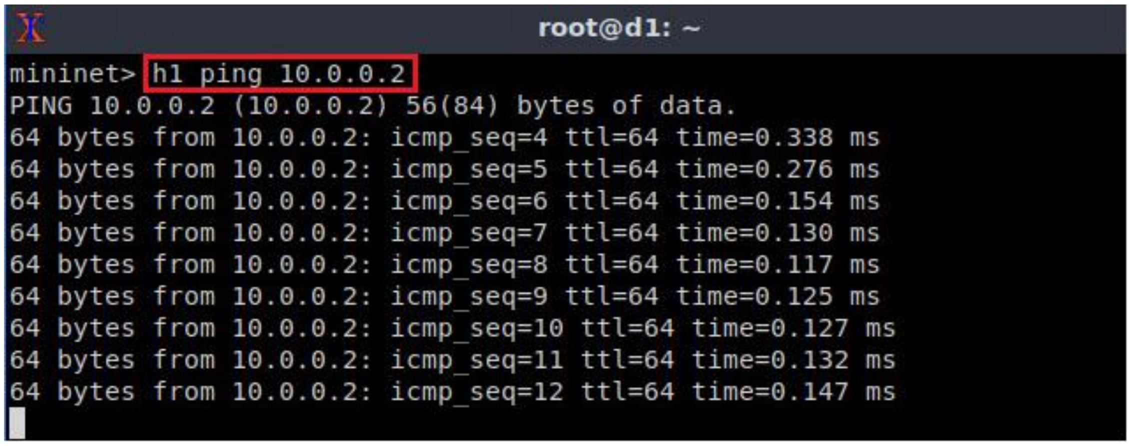


```
root@d2: ~
mininet> h3 ip link set dev h3-eth0 up
mininet>
```

Figure 52. Re-enabling interface *h2-eth0*.

**Step 10.** Perform a connectivity test between h1 and h3 by issuing the following command.

```
h1 ping 10.0.0.2
```



```
root@d1: ~
mininet> 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=4 ttl=64 time=0.338 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.276 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.154 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.130 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.117 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.125 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.127 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.132 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.147 ms
```

Figure 53. Performing a connectivity test between host h1 and host h3.

Consider the figure above. The results show a successful connectivity test.

**Step 11.** The Wireshark window starts the packet capturing by clicking on the button located on the upper left-hand side.

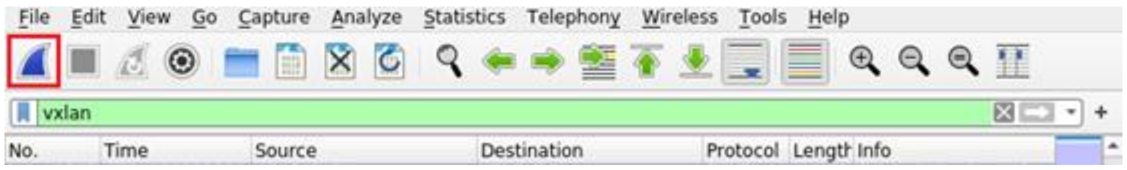
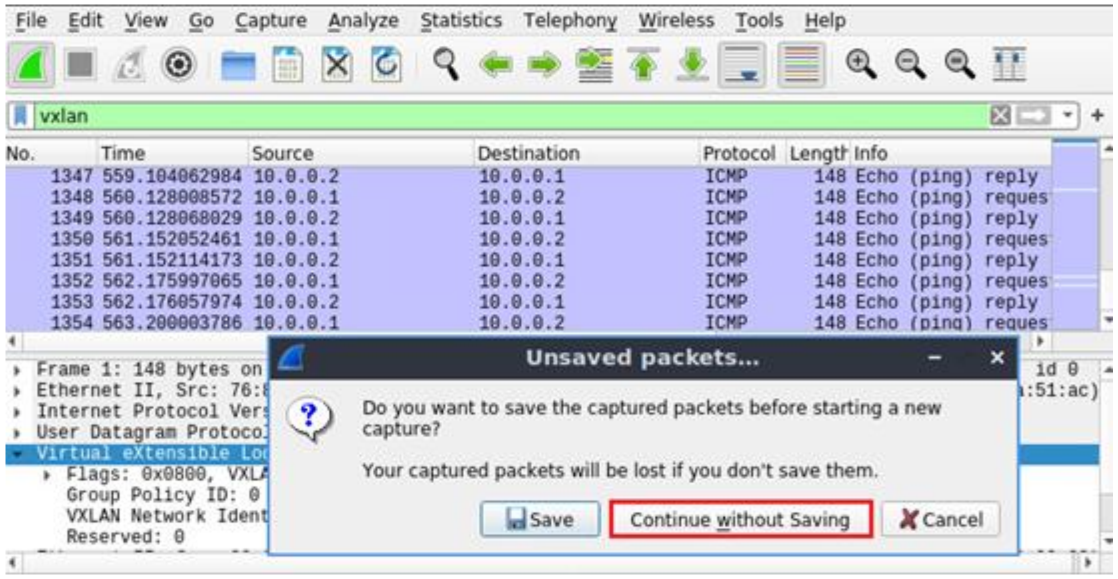


Figure 54. Starting packet capturing.

**Step 12.** A notification window will be prompted. Click on *Continue without Saving* to proceed.



```
No.  Time            Source    Destination Protocol Length Info
1347 559.104062984 10.0.0.2   10.0.0.1    ICMP    148 Echo (ping) reply
1348 560.128008572 10.0.0.1   10.0.0.2    ICMP    148 Echo (ping) reques
1349 560.128068029 10.0.0.2   10.0.0.1    ICMP    148 Echo (ping) reply
1350 561.152052461 10.0.0.1   10.0.0.2    ICMP    148 Echo (ping) reques
1351 561.152114173 10.0.0.2   10.0.0.1    ICMP    148 Echo (ping) reply
1352 562.175997065 10.0.0.1   10.0.0.2    ICMP    148 Echo (ping) reques
1353 562.176057974 10.0.0.2   10.0.0.1    ICMP    148 Echo (ping) reply
1354 563.200003786 10.0.0.1   10.0.0.2    ICMP    148 Echo (ping) reques
```

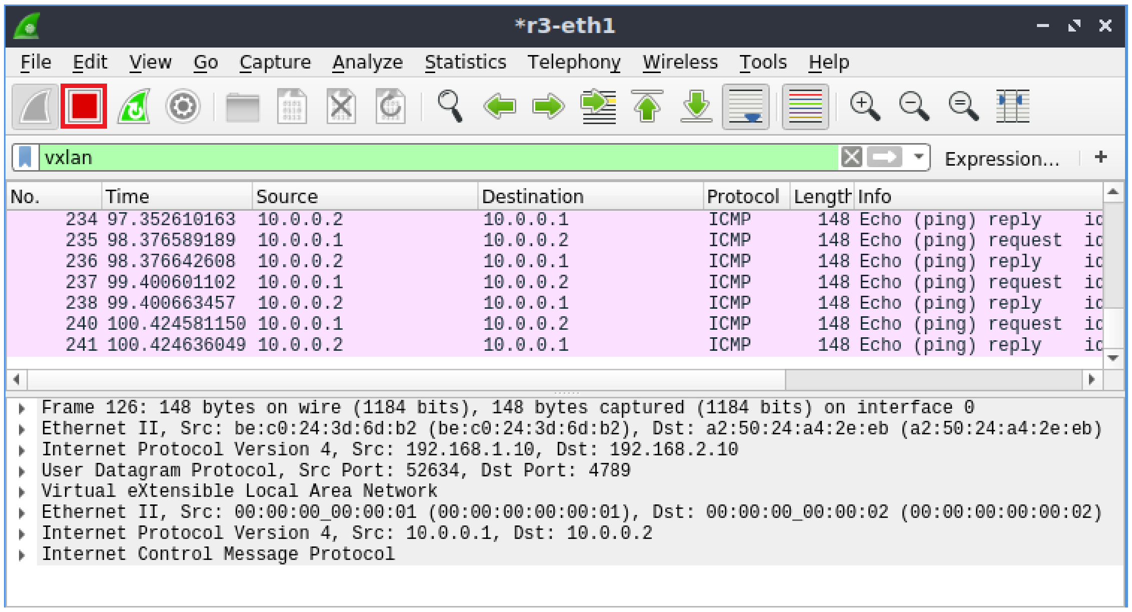
Unsaved packets...

Do you want to save the captured packets before starting a new capture?

Your captured packets will be lost if you don't save them.

Figure 55. Closing without saving previous packet capture.

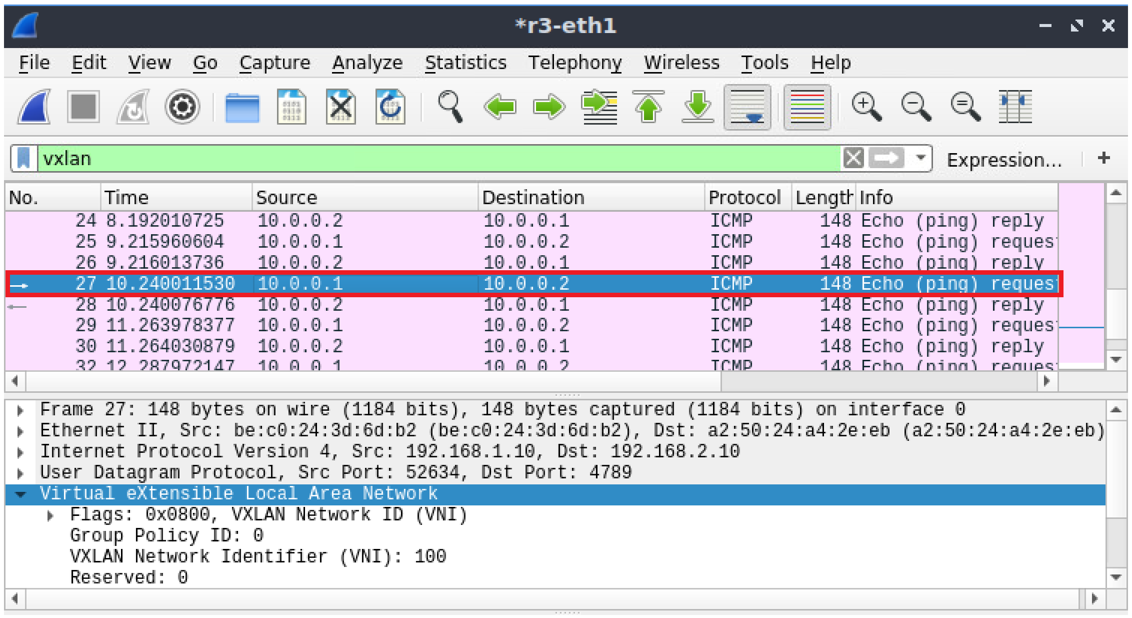
**Step 13.** Stop the packet capturing by clicking the red stop button.



```
No.  Time           Source    Destination Protocol Length Info
 234 97.352610163  10.0.0.2  10.0.0.1    ICMP    148 Echo (ping) reply   id
 235 98.376589189  10.0.0.1  10.0.0.2    ICMP    148 Echo (ping) request id
 236 98.376642608  10.0.0.2  10.0.0.1    ICMP    148 Echo (ping) reply   id
 237 99.400601102  10.0.0.1  10.0.0.2    ICMP    148 Echo (ping) request id
 238 99.400663457  10.0.0.2  10.0.0.1    ICMP    148 Echo (ping) reply   id
 240 100.424581150 10.0.0.1  10.0.0.2    ICMP    148 Echo (ping) request id
 241 100.424636049 10.0.0.2  10.0.0.1    ICMP    148 Echo (ping) reply   id

Frame 126: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
Ethernet II, Src: be:c0:24:3d:6d:b2 (be:c0:24:3d:6d:b2), Dst: a2:50:24:a4:2e:eb (a2:50:24:a4:2e:eb)
Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.2.10
User Datagram Protocol, Src Port: 52634, Dst Port: 4789
Virtual eXtensible Local Area Network
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.2
Internet Control Message Protocol
```

Figure 56. Stopping the packet capturing.

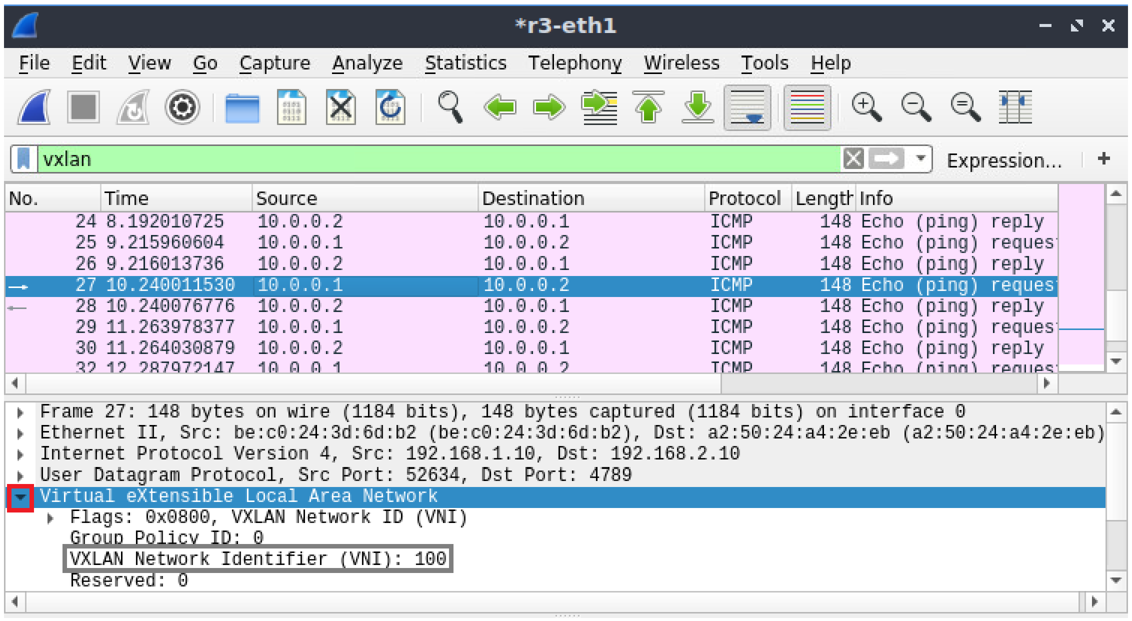
**Step 14.** Select a packet which source IP is 10.0.0.1 and destination IP is 10.0.0.2.


*r3-eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
vxlan
Expression... +

```
No.  Time          Source     Destination  Protocol Length Info
  24 8.192010725   10.0.0.2   10.0.0.1     ICMP       148 Echo (ping) reply
  25 9.215960604   10.0.0.1   10.0.0.2     ICMP       148 Echo (ping) reques
  26 9.216013736   10.0.0.2   10.0.0.1     ICMP       148 Echo (ping) reply
  27 10.240011530  10.0.0.1   10.0.0.2     ICMP       148 Echo (ping) reques
  28 10.240076776  10.0.0.2   10.0.0.1     ICMP       148 Echo (ping) reply
  29 11.263978377  10.0.0.1   10.0.0.2     ICMP       148 Echo (ping) reques
  30 11.264030879  10.0.0.2   10.0.0.1     ICMP       148 Echo (ping) reply
  32 12.287972147  10.0.0.1   10.0.0.2     ICMP       148 Echo (ping) reques

Frame 27: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
Ethernet II, Src: be:c0:24:3d:6d:b2 (be:c0:24:3d:6d:b2), Dst: a2:50:24:a4:2e:eb (a2:50:24:a4:2e:eb)
Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.2.10
User Datagram Protocol, Src Port: 52634, Dst Port: 4789
Virtual eXtensible Local Area Network
    Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 100
    Reserved: 0
```

Figure 57. Selecting a packet for further inspection.

**Step 15.** Select a packet which source IP is 10.0.0.1 and verify that the VXLAN Network Identifier is 100. Notice that this tag corresponds to the traffic from h1 to h3.


*r3-eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
vxlan
Expression... +

```
No.  Time          Source     Destination  Protocol Length Info
  24 8.192010725   10.0.0.2   10.0.0.1     ICMP       148 Echo (ping) reply
  25 9.215960604   10.0.0.1   10.0.0.2     ICMP       148 Echo (ping) reques
  26 9.216013736   10.0.0.2   10.0.0.1     ICMP       148 Echo (ping) reply
  27 10.240011530  10.0.0.1   10.0.0.2     ICMP       148 Echo (ping) reques
  28 10.240076776  10.0.0.2   10.0.0.1     ICMP       148 Echo (ping) reply
  29 11.263978377  10.0.0.1   10.0.0.2     ICMP       148 Echo (ping) reques
  30 11.264030879  10.0.0.2   10.0.0.1     ICMP       148 Echo (ping) reply
  32 12.287972147  10.0.0.1   10.0.0.2     ICMP       148 Echo (ping) reques

Frame 27: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
Ethernet II, Src: be:c0:24:3d:6d:b2 (be:c0:24:3d:6d:b2), Dst: a2:50:24:a4:2e:eb (a2:50:24:a4:2e:eb)
Internet Protocol Version 4, Src: 192.168.1.10, Dst: 192.168.2.10
User Datagram Protocol, Src Port: 52634, Dst Port: 4789
Virtual eXtensible Local Area Network
    Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 100
    Reserved: 0
```

Figure 58. Verifying VXLAN network identifier.

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

## References

1. M. Mallik, D. Dut, K.Duda, P. Agarwal, L. Kreeger, TSridhar, M. Bursell and C. Wright. "*RFC* 7348: *Virtual eXtensible Local Area Network (VXLAN): a framework for overlaying virtualized layer 2 networks over layer 3 Networks.*", (2014).
2. Cisco press., "*Implementing Data Center Overlay Protocols.*", [Online]: Available: https://www.ciscopress.com/articles/article.asp?p=2999385&seqNum=3.
3. Mininet walkthrough, [Online]. Available: http://mininet.org.
4. M. Peuster, J. Kampmeyer, and H. Karl. "*Containernet 2.0: A rapid prototyping platform for hybrid service function chains.*" 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft). (2018).
5. 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).
6. P. Goransson, C. Black, T. Culver. "*Software defined networks: a comprehensive approach*". Morgan Kaufmann, (2016).
7. P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, "*ONOS: towards an open, distributed SDN OS*," In Proceedings of the third workshop on Hot topics in software defined networking, (2014).
8. Juniper Networks, "*Understanding EVPN with VXLAN Data Plane Encapsulation*", [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics /concept/evpn-vxlan-data-plane-encapsulation.html.
9. Q. Xiaorong, W. Hao, and Y. Yin. "*L3 gateway for VXLAN.*" U.S. Patent No. 8,923,155. (2014).
10. D. Merkel. "*Docker: lightweight linux containers for consistent development and deployment*." Linux journal, (2014).
11. Linux foundation collaborative projects, "*FRRouting: what's in your router*", [Online]. https://frrouting.org/

# SOFTWARE DEFINED NETWORKING

# Exercise 2: Configuring VXLAN

**Document Version: 09-02-2021**

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

## Contents

## 1 Exercise description

Consider Figure 1. The topology comprises two servers using Docker containers, both connected to the IP network. Each server contains three hosts that run in Mininet. In each server, every host has a unique IP address and a VXLAN Network Identifier (VNID). Within the two servers, hosts that have the same VNID, also share the same IP address.

The goal of this exercise is to isolate the traffic in each server and provide end-to-end connectivity between the hosts with the same VNID. To do that, you should connect two remote servers over an IP network using VXLAN. Essentially, you should configure a single area OSPF within the IP network. Then, you need to configure VTEP in each server to isolate the traffic, and load the flow tables to switches s1 and s2.

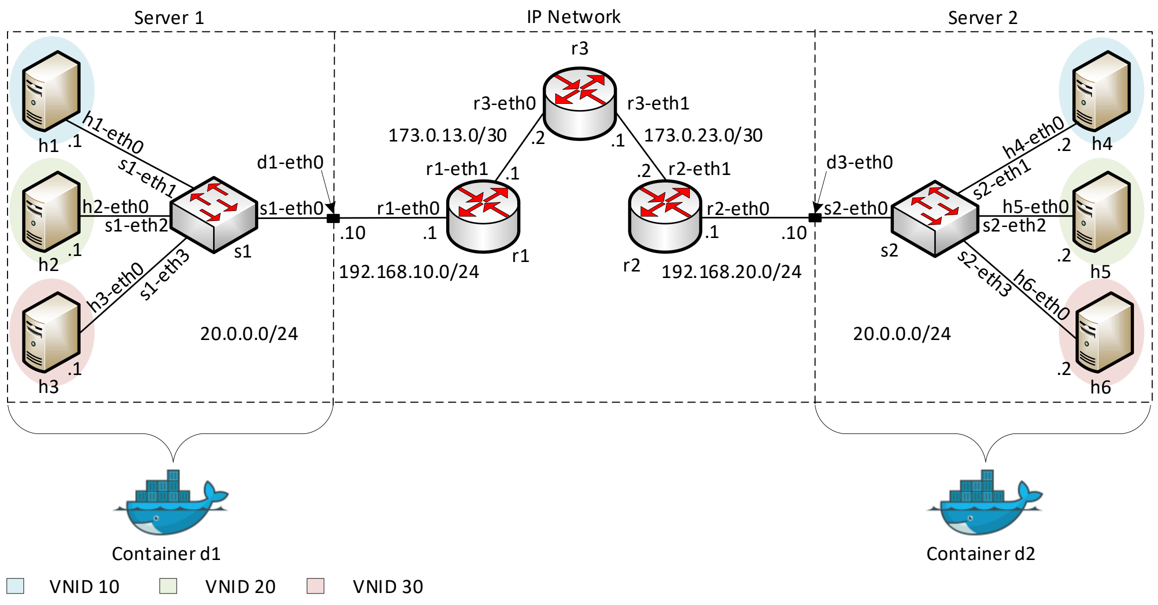


Figure 1. Exercise topology.

### 1.1 Topology settings

The devices are already configured according to Table 1.

Table 1. Topology information.

| Device | Interface | IP Address | Subnet |
|---|---|---|---|
| Router r1 | r1-eth0 | 192.168.10.1 | /24 |
| | r1-eth1 | 173.0.13.1 | /30 |
| Router r2 | r2-eth0 | 192.168.20.1 | /24 |
| | r2-eth1 | 173.0.23.2 | /30 |
| Router r3 | r3-eth0 | 173.0.13.2 | /30 |
| | r3-eth1 | 173.0.23.1 | /30 |

| | | | |
|---|---|---|---|
| Host h1 | h1-eth0 | 20.0.0.1 | /24 |
| Host h2 | h2-eth0 | 20.0.0.1 | /8 |
| Host h3 | h3-eth0 | 20.0.0.1 | /8 |
| Host h4 | h4-eth0 | 20.0.0.2 | /8 |
| Host h5 | h5-eth0 | 20.0.0.2 | /8 |
| Host h6 | h6-eth0 | 20.0.0.2 | /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)** Open MiniEdit and load the topology above. The topology file *Exercise2.mn* of this exercise is in the directory */home/sdn/SDN_Labs/Exercise2* as shown in the figure below. Do not run MiniEdit.

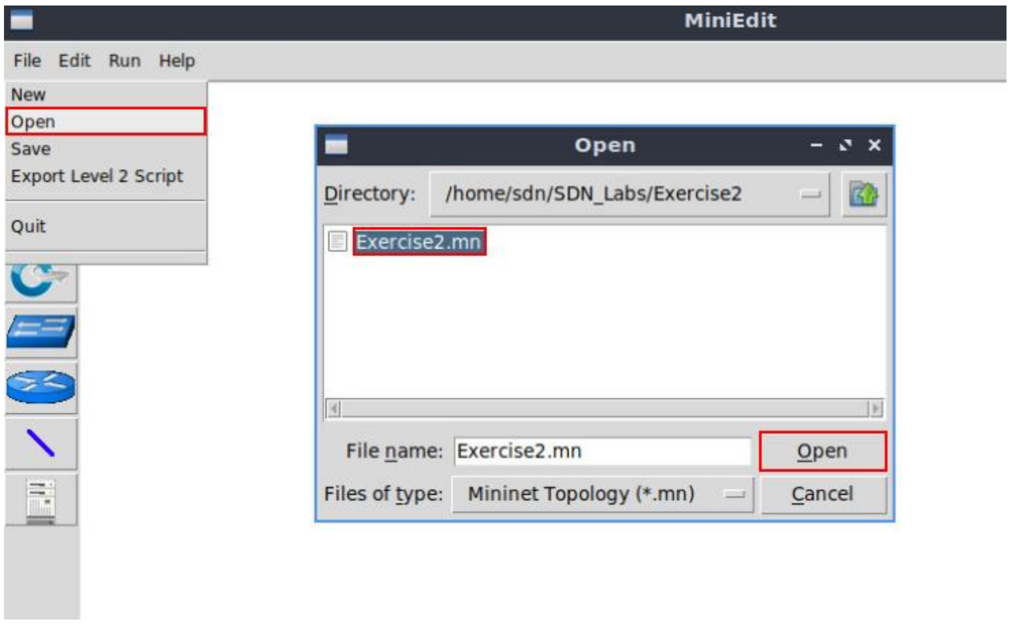
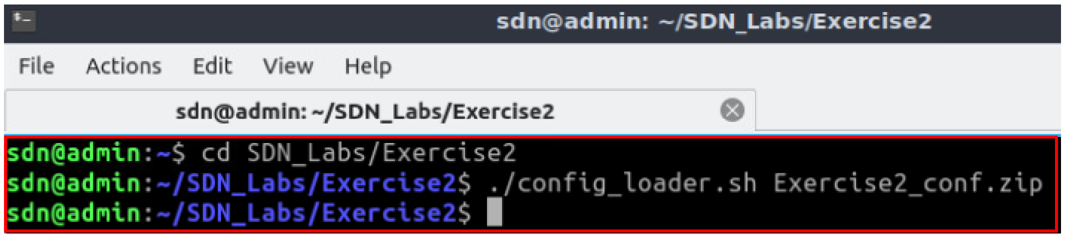


Figure 2. Loading the topology file in Mininet.

**b)** In Linux terminal, run the script responsible for loading the IP addresses to the interfaces of the routers as shown in the figure below. The script *config_loader.sh* takes the IP addresses file *Exercise2_conf.zip* as an argument, and both files are in the directory *SDN_Labs/Exercise2.*


sdn@admin: ~/SDN_Labs/Exercise2
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/Exercise2

```
sdn@admin:~$ cd SDN_Labs/Exercise2
sdn@admin:~/SDN_Labs/Exercise2$ ./config_loader.sh Exercise2_conf.zip
sdn@admin:~/SDN_Labs/Exercise2$
```

Figure 3. Executing the shell script to load the configuration.

**c)** Run MiniEdit and verify in Mininet terminal that the links conform to the topology figure and settings table above.

**d)** Configure a single area OSPF in each router. Essentially, the routers should advertise via OSPF all their directly connected networks.

**e)** In the container d1, run the following python script to start the Mininet topology:

```
python start_server1.py
```

**f)** In the container d2, run the following python script to start the Mininet topology:

```
python start_server2.py
```

**g)** In the container d1, add entries to the flow tables of switch s1 by issuing the following command:

```
ovs-ofctl add-flows s1 flows1.txth
```

**h)** In the container d2, add entries to the flow tables of switch s2 by issuing the following command:

```
ovs-ofctl add-flows s2 flows2.txt
```

**i)** In the container d1, run the following command to configure the VTEP:

```
sh ./vxlan_cmd1.cmd
```

**j)** In the container d2, run the following command to configure the VTEPs:

```
sh ./vxlan_cmd2.cmd
```

**k)** Verify the configuration by testing the connectivity between hosts with the same VNID.

**l)** Capture packets on the interface *r2-eth0* using Wireshark and verify that the VNIDs are configured according to the topology in Figure 1.

UNIVERSITY OF SOUTH CAROLINA

# SOFTWARE DEFINED NETWORKING

# Lab 6: Introduction to OpenFlow

**Document Version: 07-03-2021**

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

## Contents

## Overview

This lab is an introduction to OpenFlow, which defines both the communications protocol between the Software Defined Networking (SDN) data plane and the SDN control plane, and part of the behavior of the data plane. In this lab, you will use the *ovs-ofctl* command line utility to administer OpenFlow switches, such as inserting/deleting flows. The focus in this lab is to understand and inspect the OpenFlow messages exchanged between the control plane and the data plane.

## Objectives

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

1. Understand SDN and its components.
2. Understand OpenFlow.
3. Configure OpenFlow switches using ovs-ofctl.
4. Configure the ONOS controller.
5. Use the Wireshark network analyzer to capture OpenFlow packets.

## Lab settings

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

Table 1**.** Credentials to access the 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: Monitoring and administering OpenFlow switches.
4. Section 4: Capturing OpenFlow packets.

## 1 Introduction

### 1.1 OpenFlow Overview

OpenFlow defines both the communication protocol between the SDN data plane and the SDN control plane, as well as part of the behavior of the data plane. It does not describe the behavior of the controller itself. There are other approaches to SDN, but today OpenFlow is the only nonproprietary, general-purpose protocol for programming the forwarding plane of SDN switches[3].

Consider Figure 1. In a basic component of the OpenFlow system, there is always an OpenFlow controller that communicates to one or more OpenFlow switches. The OpenFlow protocol defines the specific messages and message formats exchanged between the controller (control plane) and the device (data plane). The OpenFlow behavior specifies how the device should react in various situations and how it should respond to commands from the controller.

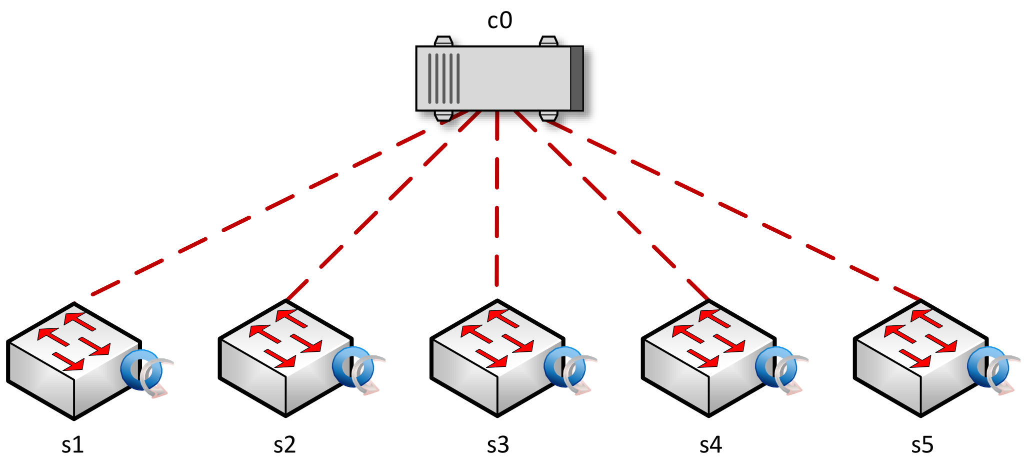


Figure 1. OpenFlow components.

## 1.2 OpenFlow components

In a packet switch, the core function is to take packets that arrive on one port and forward them through another port. OpenFlow switches perform this operation using the packet-matching function with the flow table. Thus, once a packet arrives to the switch, the packet matching function 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. The action could be:

- Forward the packet out a local port.
- Drop the packet.
- Pass the packet to the controller.

The basic functions of an OpenFlow switch and its relationship to a controller are depicted in Figure 2. When the data plane does not have a match to 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 what the forwarding tables and logic in the data plane should be. Consequently, when the controller has a data packet to forward out through the switch, it uses the OpenFlow PACKET_OUT message. All the communication between the OpenFlow controller and data plane is defined by the OpenFlow protocol.

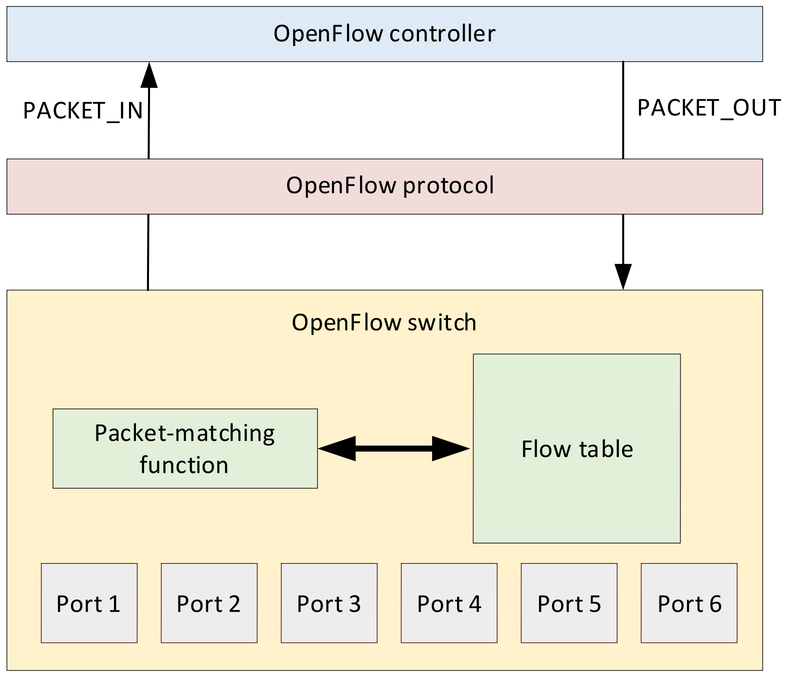


Figure 2. OpenFlow switch.

## 2 Lab topology

Consider Figure 3. The topology consists of two end-hosts, a switch, and a controller. The blue device is an OpenFlow switch, and it is directly connected to the controller c0.

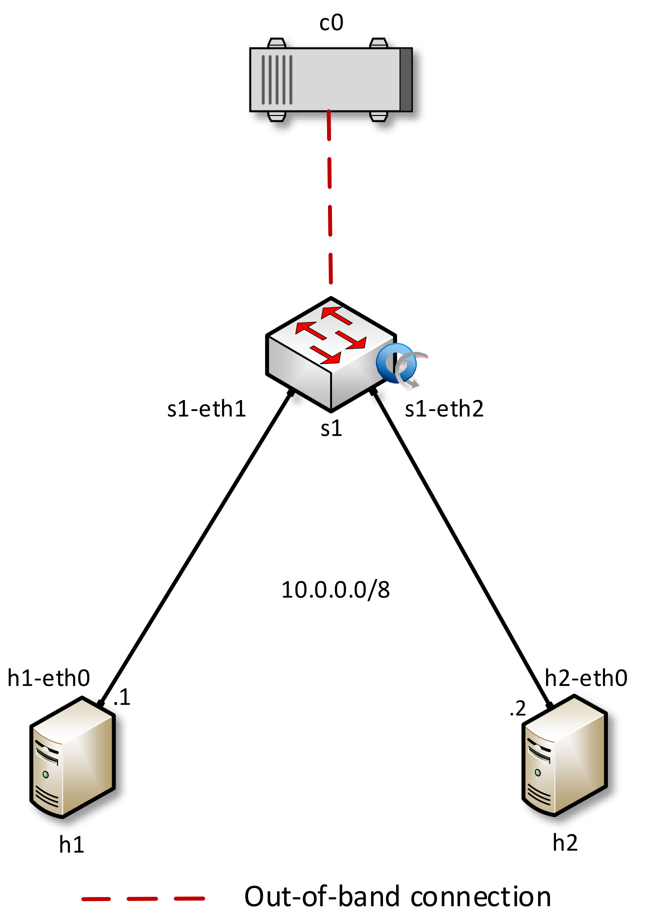


Figure 3. Lab topology.

## 2.1 Lab settings

The devices are already configured according to Table 2.

Table 2. Topology information.

| Device | Interface | MAC Address | IP Address | Subnet |
|---|---|---|---|---|
| h1 | h1-eth0 | 00:00:00:00:00:01 | 10.0.0.1 | /8 |
| h2 | h2-eth0 | 00:00:00:00:00:02 | 10.0.0.2 | /8 |
| c0 | N/A | N/A | 172.17.0.2 | /16 |

## 2.2 Loading the 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 such as: configuring network elements (i.e IP addresses, default gateway), saving the topology, and exporting a layer 2 model.

**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`.

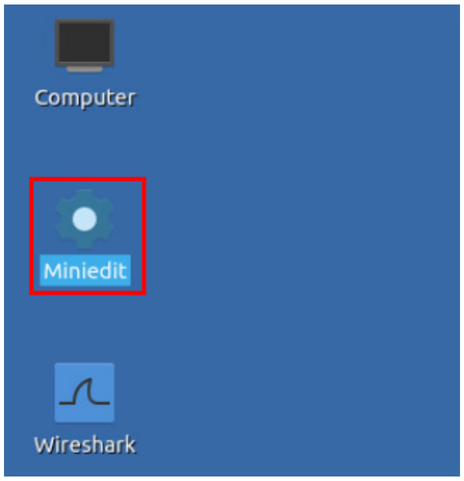


Figure 4. MiniEdit shortcut.

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

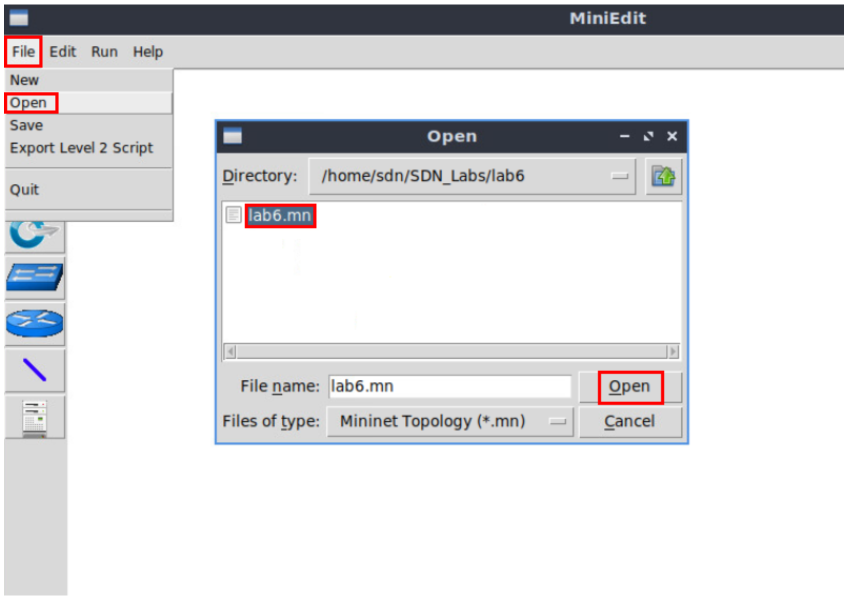


Figure 5. Opening topology.

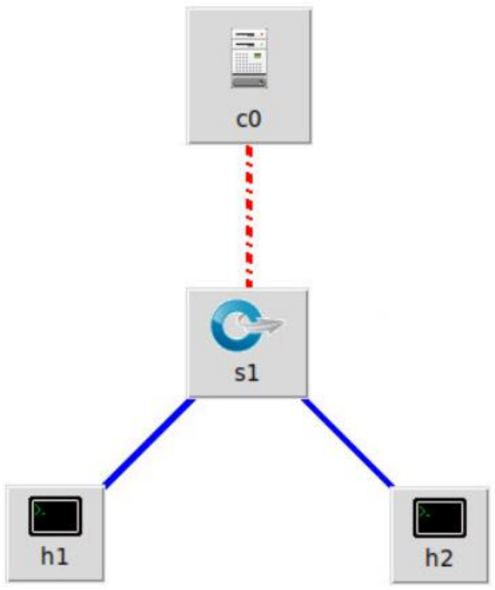


Figure 6. MiniEdit's topology.

**Step 3.** 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.

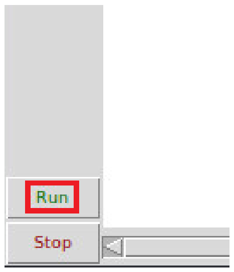


Figure 7. Starting the emulation.

## 3 Monitoring and administering OpenFlow switches

In this section, you will use *ovs-ofctl* command line tool to monitor and administer OpenFlow switches. This tool can show the current state of an OpenFlow switch, including its features, configuration, and table entries.

**Step 1.** Open the Linux terminal by clicking on the shortcut depicted below.

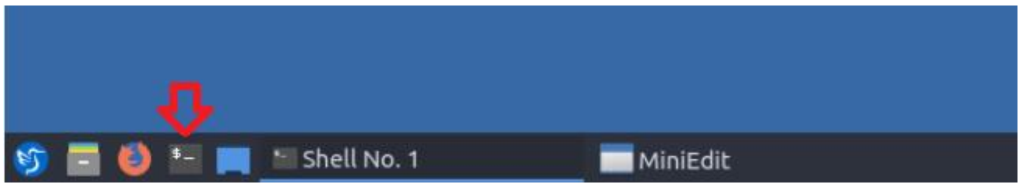
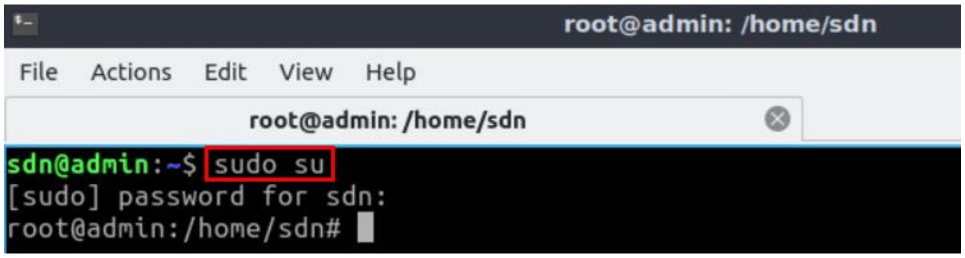


Figure 8. Opening Linux terminal.

**Step 2.** Issue the command below to execute programs with the security privileges of the superuser (root). When prompted for a password, type `password`.

```
sudo su
```

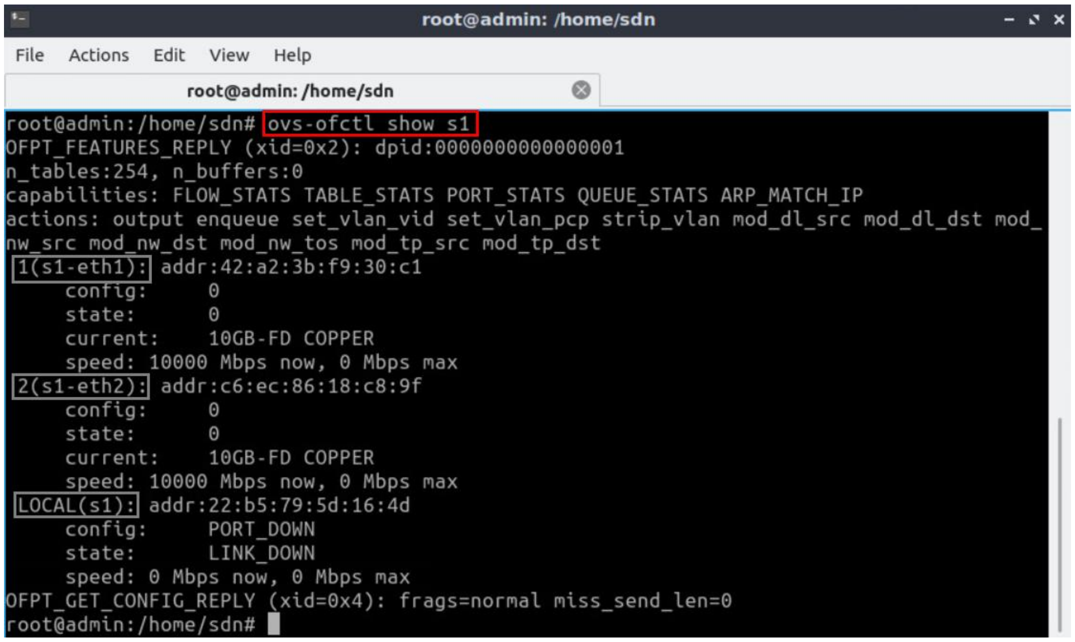

File Actions Edit View Help

```
sdn@admin:~$ sudo su
[sudo] password for sdn:
root@admin:/home/sdn#
```

Figure 9. Switching to root mode.

**Step 3.** Issue the command below to connect to switch s1 and show its information.

```
ovs-ofctl show s1
```


File Actions Edit View Help

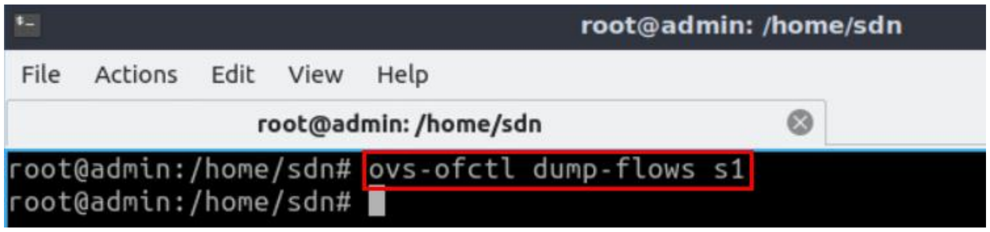
```
root@admin:/home/sdn# ovs-ofctl show s1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst mod_
nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
 1(s1-eth1): addr:42:a2:3b:f9:30:c1
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:c6:ec:86:18:c8:9f
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:22:b5:79:5d:16:4d
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
root@admin:/home/sdn#
```

Figure 10. Showing switch s1 information.

Consider Figure 10. Switch s1 has three interfaces. Each interface displays the Media Access Control (MAC) address (`addr`) along with other information, such as the current state of the switch. Note that the generated MAC addresses might be different since they are randomly generated by Mininet.

**Step 4.** Issue the command below to print the flow entries of switch s1.

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


File Actions Edit View Help

```
root@admin:/home/sdn# ovs-ofctl dump-flows s1
root@admin:/home/sdn#
```

Figure 11. Showing the flow entries of switch s1.

Consider Figure 11. No output was shown in response to the command above. This is because initially, the switch has no flow entries.

**Step 5.** Hold right-click on host h1 and select Terminal.

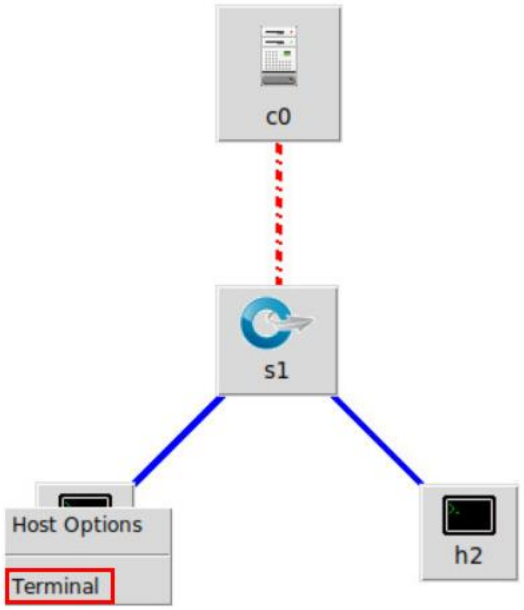
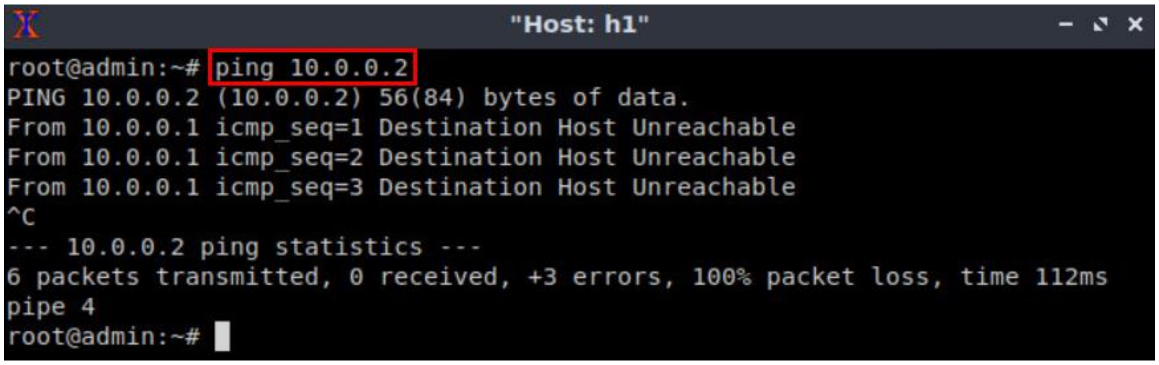


Figure 12. Opening host h1 terminal.

**Step 6.** Run a connectivity test by issuing the command shown below. The `ping` command is used to verify the connectivity between two ends. It must be followed by the IP address of the destination host, which is 10.0.0.2 (host h2) in this case. Then, you can stop the test, press `Ctrl+c`.

```
ping 10.0.0.2
```



```
root@admin:~# 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 ---
6 packets transmitted, 0 received, +3 errors, 100% packet loss, time 112ms
pipe 4
root@admin:~#
```

Figure 13. Pinging host h2 from host h1.

In the figure above, the connectivity test is unsuccessful since switch s1 flow table is empty. Incoming traffic to the switch will not match any rule, and hence no action will be taken. Therefore, switch s1 does not know what to do with incoming traffic, leading to ping failure.

**Step 7.** Open the Linux terminal.

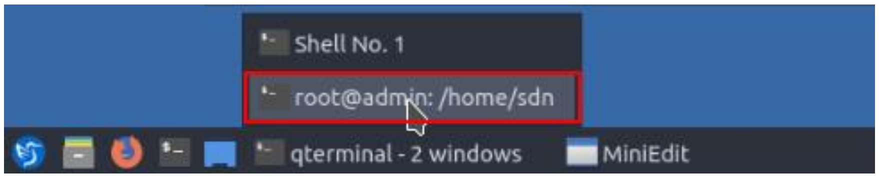
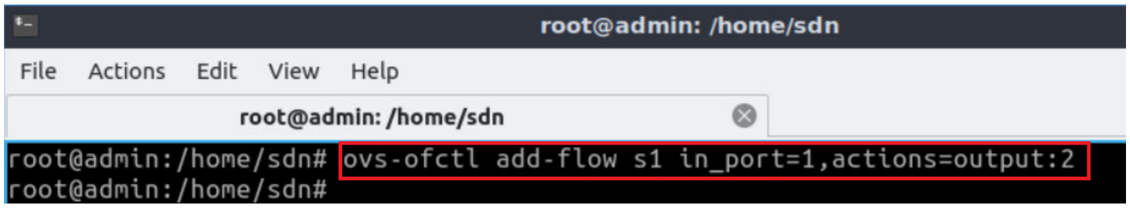


Figure 14. Opening Linux terminal.

**Step 8.** Issue the below command to manually install a flow into switch s1. The inserted flow forwards incoming packets at port 1 (`in_port=1`) to port 2 (`actions=output:2`).

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

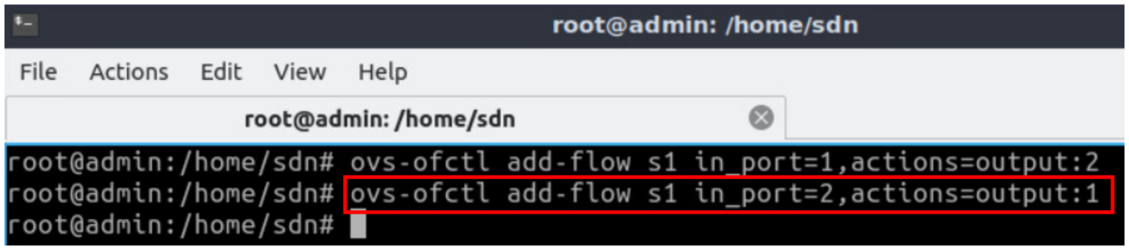


```
root@admin:/home/sdn# ovs-ofctl add-flow s1 in_port=1,actions=output:2
root@admin:/home/sdn#
```

Figure 15. Adding a flow entry to switch s1.

**Step 9.** Issue the below command to manually install a flow into switch s1. The inserted flow forwards incoming packets at port 2 (`in_port=2`) to port 1 (`actions=output:1`).

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

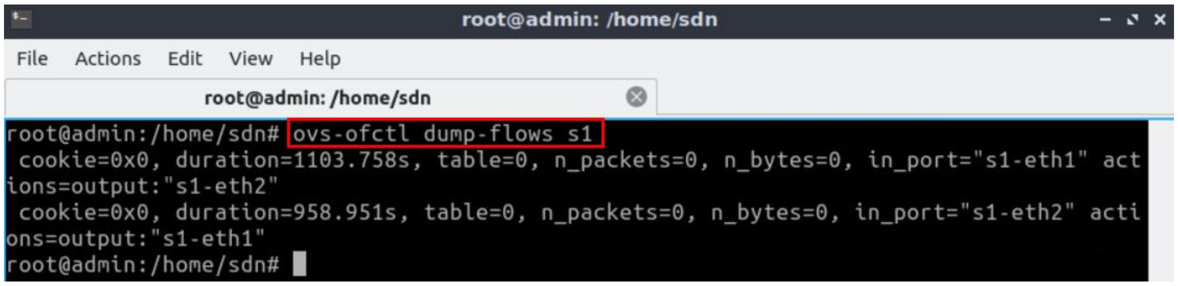


```
root@admin:/home/sdn# ovs-ofctl add-flow s1 in_port=1,actions=output:2
root@admin:/home/sdn# ovs-ofctl add-flow s1 in_port=2,actions=output:1
root@admin:/home/sdn#
```

Figure 16. Adding a flow entry to switch s1.

**Step 10.** Issue the command below to print the flow entries of switch s1.

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

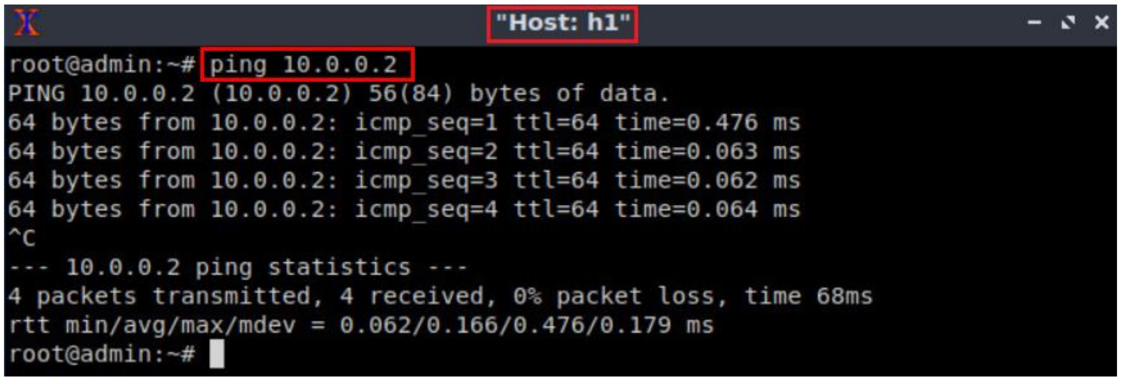


```
root@admin:/home/sdn# ovs-ofctl dump-flows s1
 cookie=0x0, duration=1103.758s, table=0, n_packets=0, n_bytes=0, in_port="s1-eth1" act
ions=output:"s1-eth2"
 cookie=0x0, duration=958.951s, table=0, n_packets=0, n_bytes=0, in_port="s1-eth2" acti
ons=output:"s1-eth1"
root@admin:/home/sdn#
```

Figure 17. Showing the flow entries of switch s1.

**Step 11.** On host h1 terminal, run a connectivity test with host h2 by issuing the following command. Then, you can press `Ctrl+c` to stop the test.

```
ping 10.0.0.2
```

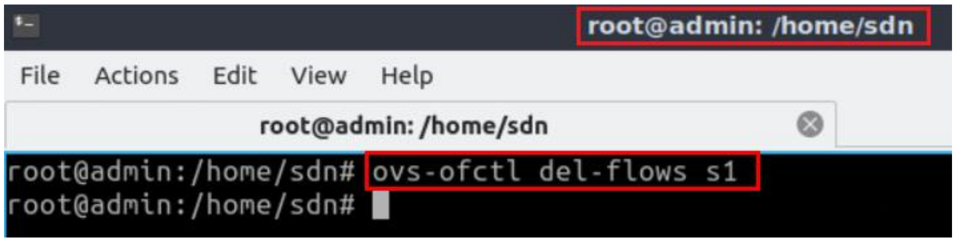


```
root@admin:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.476 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.062 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.064 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 68ms
rtt min/avg/max/mdev = 0.062/0.166/0.476/0.179 ms
root@admin:~#
```

Figure 18. Pinging host h2 from host h1.

**Step 12.** In addition to adding flow entries to the switches using ovs-ofctl, you can also delete entries as well as deleting the whole flow table. Issue the following command on the Linux terminal to delete the flow table of switch s1.

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



```
root@admin:/home/sdn# ovs-ofctl del-flows s1
root@admin:/home/sdn#
```

Figure 19. Deleting the flow table of switch s1.

## 4 Capturing OpenFlow packets

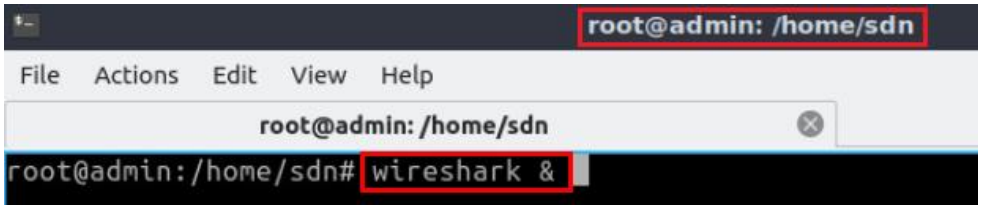
In this section, you will start Wireshark, navigate through some of its features, and learn how to monitor network traffic. Additionally, you will enable the ONOS controller and capture OpenFlow packets.

### 4.1 Starting Wireshark

In this section, you will use Wireshark, the defacto network protocol analyzer, to monitor the network and inspect OpenFlow packets that are being transmitted between the controller and the data plane (switch).

**Step 1.** In the Linux terminal, issue the following command to launch Wireshark.

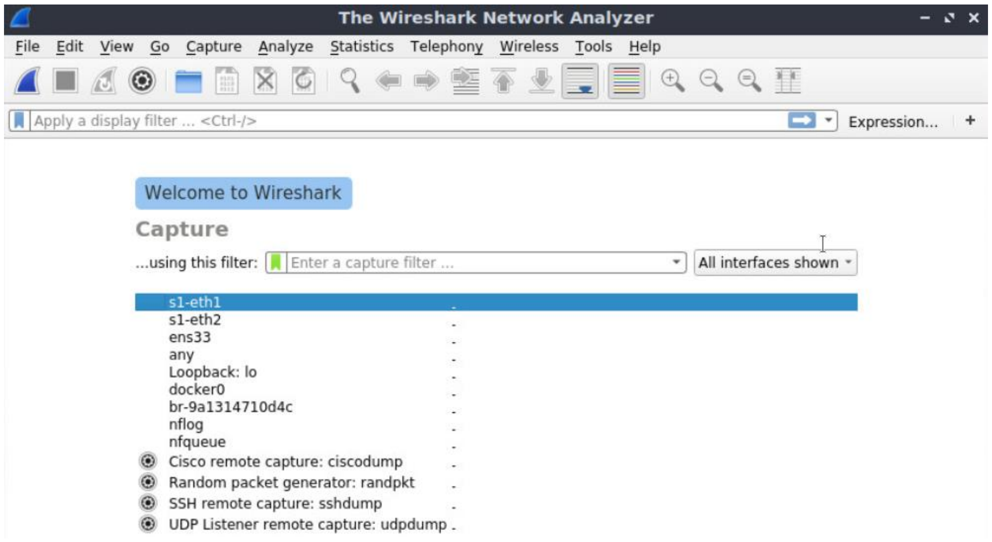
```
wireshark &
```



```
root@admin:/home/sdn# wireshark &
```

Figure 20. Starting Wireshark.

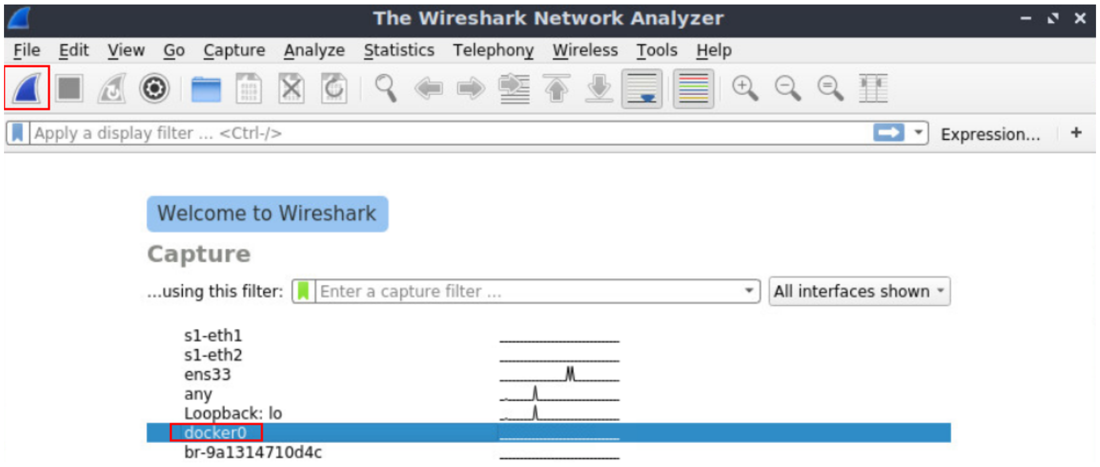
Wireshark window depicted in Figure 21 will appear after executing the command above.


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
All interfaces shown

```
s1-eth1
s1-eth2
ens33
any
Loopback: lo
docker0
br-9a1314710d4c
nflog
nfqueue
Cisco remote capture: ciscodump
Random packet generator: randpkt
SSH remote capture: sshdump
UDP Listener remote capture: udpdump
```

Figure 21. Wireshark window.

**Step 2.** In the opened Wireshark window, you will see a list of interfaces that Wireshark can capture network traffic on, such as *s1-eth1, s1-eth2*. Click on *docker0* then start capturing the packets by clicking the blue *shark fin* icon on the top left of the Window. By clicking on this interface, you will capture the traffic on the ONOS controller that is running in a docker container.


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
All interfaces shown

```
s1-eth1
s1-eth2
ens33
any
Loopback: lo
docker0
br-9a1314710d4c
```

Figure 22. Start capturing packets in Wireshark.

**Step 3.** When you start capturing packets, you will notice that Wireshark is divided into three sections. The first section displays the captured packets including their number, time they were captured, source and destination IP addresses, protocol, length, and information about the packet. The second section contains detailed information about every captured packet (each selected packet will have its own information). The third section contains the real data that was captured in the packet. Currently, no packets are captured on the docker container since ONOS is not started ONOS yet.

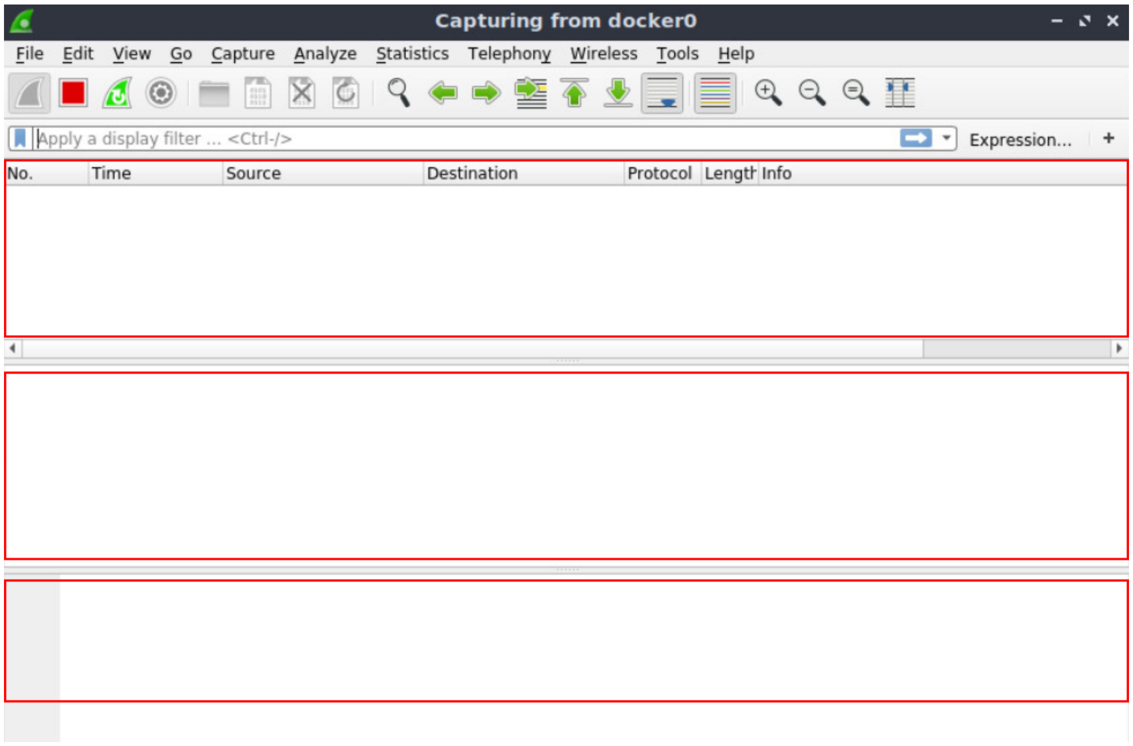

Capturing from docker0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination Protocol Length Info

Figure 23. Capturing from Loopback: lo interface.

**Step 4.** Wireshark supports filters, i.e., you can apply filters to display a specific set of capture packets. To show OpenFlow packets only (protocol: OpenFlow), write the following expression in the Wireshark filter text box, then press Enter.

```
openflow_v1
```

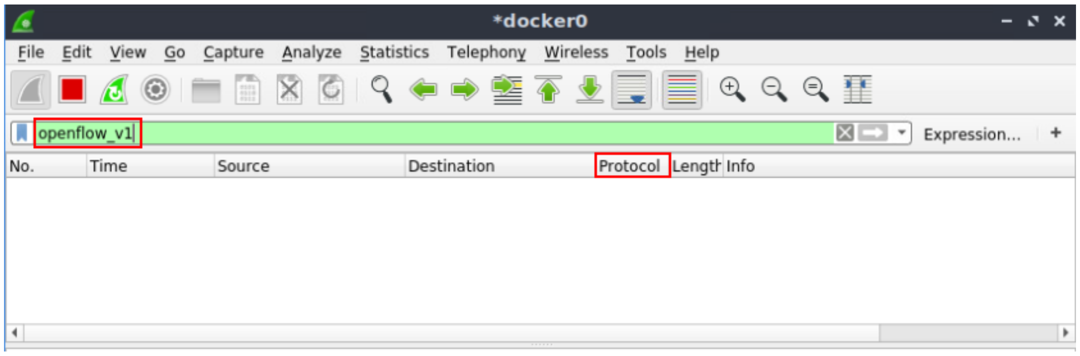

*docker0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
openflow_v1
Expression...
No. Time Source Destination Protocol Length Info

Figure 24. Capturing only OpenFlow packets in Wireshark.

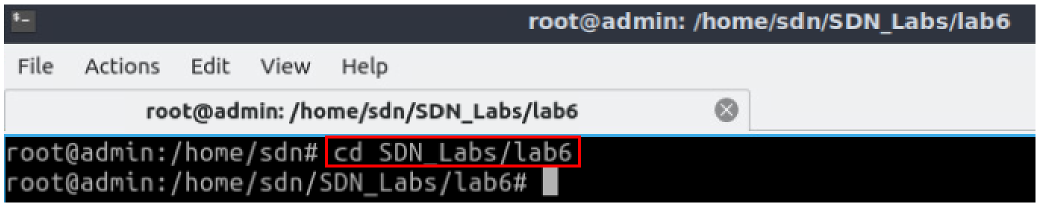
Consider Figure 24. The applied filter in Wireshark displays packets with protocol type OpenFlow only, specifically. You will use Wireshark in the next section to capture OpenFlow packets once you run ONOS and activate the OpenFlow application.

## 4.2 Starting the ONOS controller

In this section, you will start the ONOS controller and activate basic ONOS applications, such as the OpenFlow application. The latter triggers the exchange of OpenFlow packets between the data plane (switch s1) and the control plane (c0). Thus, allowing the controller to discover the topology and insert flow entries into switch s1. Using Wireshark, you will capture the exchanged OpenFlow packets and understand their main types.

**Step 1.** In the opened Linux terminal, press `Enter` then navigate into *SDN_Labs/lab6* directory by issuing the following command. This folder contains the script responsible for starting ONOS. The `cd` command is short for change directory followed by an argument that specifies the destination directory.

```
cd SDN_Labs/lab6
```

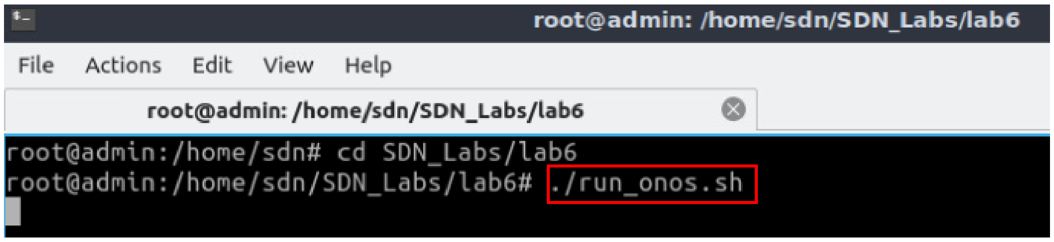

root@admin: /home/sdn/SDN_Labs/lab6
File
Actions
Edit
View
Help
root@admin: /home/sdn/SDN_Labs/lab6

```
root@admin:/home/sdn# cd SDN_Labs/lab6
root@admin:/home/sdn/SDN_Labs/lab6# 
```

Figure 25. Entering the *SDN_Labs/lab6* directory.

**Step 2.** A script was written to run ONOS and enter its Command Line Interface (CLI). In order to run the script, issue the following command. In addition to running ONOS, the script will modify the MAC addresses of the hosts so that they conform with the topology.

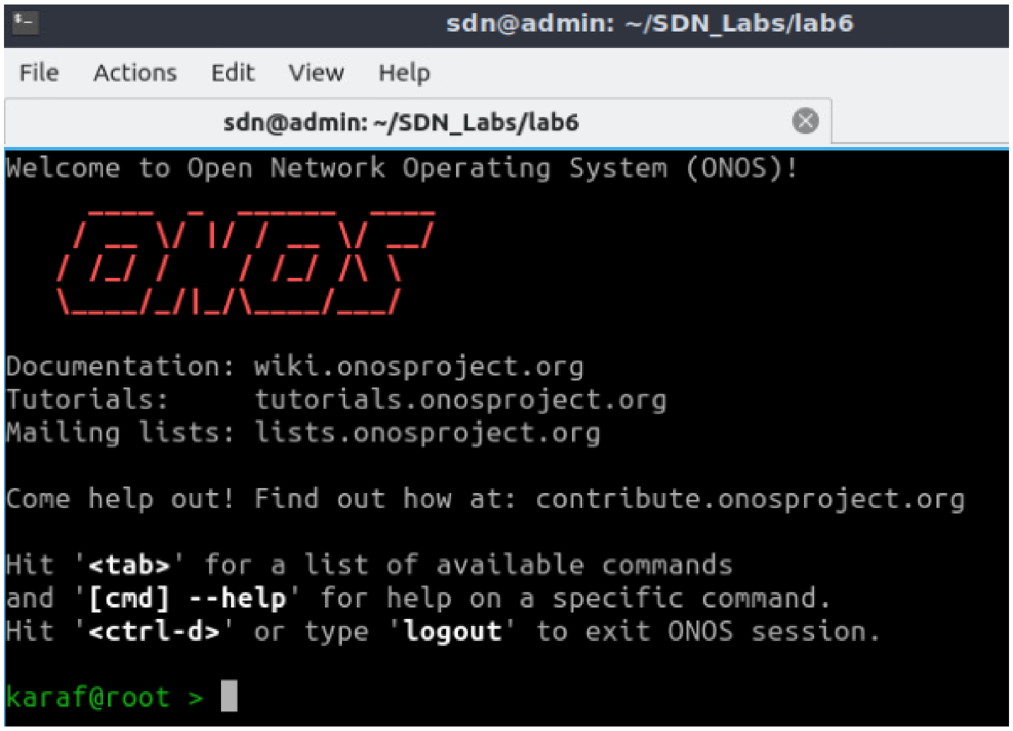
```
./run_onos.sh
```


root@admin: /home/sdn/SDN_Labs/lab6
File
Actions
Edit
View
Help
root@admin: /home/sdn/SDN_Labs/lab6

```
root@admin:/home/sdn# cd SDN_Labs/lab6
root@admin:/home/sdn/SDN_Labs/lab6# ./run_onos.sh
```

Figure 26. Starting the ONOS controller.

Once the script finishes executing and ONOS is ready, you will be able to execute commands on the ONOS CLI as shown in the figure below. Note that this script may take few seconds.

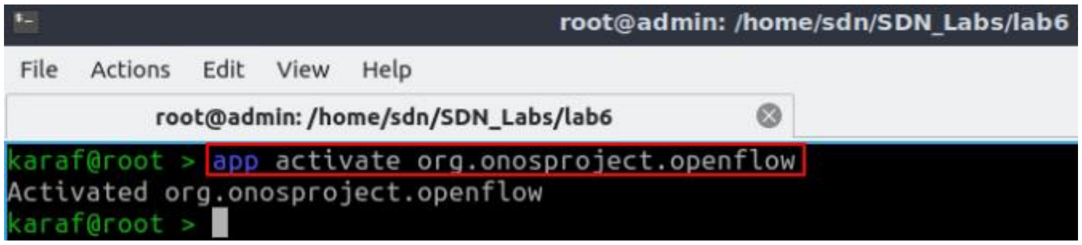

sdn@admin: ~/SDN_Labs/lab6
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab6

```
Welcome to Open Network Operating System (ONOS)!
     ____  _  ______  ____
    / __ \/ |/ / __ \/ __/
   / /_/ /    / /_/ /\ \
   \____/_/|_/\____/___/

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

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

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

karaf@root >
```

Figure 27. ONOS CLI.

**Step 3.** In the ONOS terminal, issue the following command to activate the OpenFlow application. This application allows the ONOS controller to discover the hosts, devices, and links in the current topology.

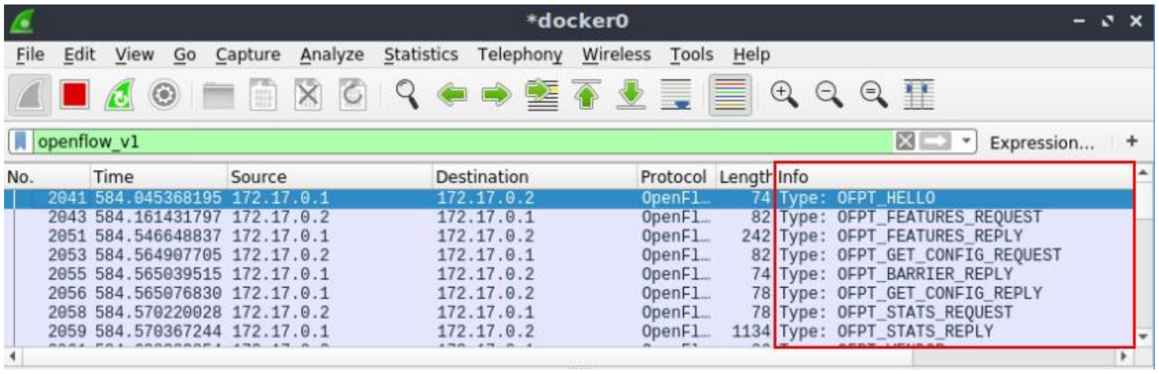
```
app activate org.onosproject.openflow
```


root@admin: /home/sdn/SDN_Labs/lab6
File Actions Edit View Help
root@admin: /home/sdn/SDN_Labs/lab6

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

Figure 28. Activating the OpenFlow application.

**Step 4.** After activating the OpenFlow application, you should see a number of OpenFlow messages displayed in Wireshark as shown in the below figure.


*docker0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
openflow_v1
Expression...

```
No.   Time          Source      Destination  Protocol Length Info
 2041 584.045368195 172.17.0.1  172.17.0.2   OpenFl…      74 Type: OFPT_HELLO
 2043 584.161431797 172.17.0.2  172.17.0.1   OpenFl…      82 Type: OFPT_FEATURES_REQUEST
 2051 584.546648837 172.17.0.1  172.17.0.2   OpenFl…     242 Type: OFPT_FEATURES_REPLY
 2053 584.564907705 172.17.0.2  172.17.0.1   OpenFl…      82 Type: OFPT_GET_CONFIG_REQUEST
 2055 584.565039515 172.17.0.1  172.17.0.2   OpenFl…      74 Type: OFPT_BARRIER_REPLY
 2056 584.565076830 172.17.0.1  172.17.0.2   OpenFl…      78 Type: OFPT_GET_CONFIG_REPLY
 2058 584.570220028 172.17.0.2  172.17.0.1   OpenFl…      78 Type: OFPT_STATS_REQUEST
 2059 584.570367244 172.17.0.1  172.17.0.2   OpenFl…    1134 Type: OFPT_STATS_REPLY
```

Figure 29. Capturing OpenFlow packets using Wireshark.

The figure above shows the first captured packets in Wireshark. The exchanged OpenFlow messages include:

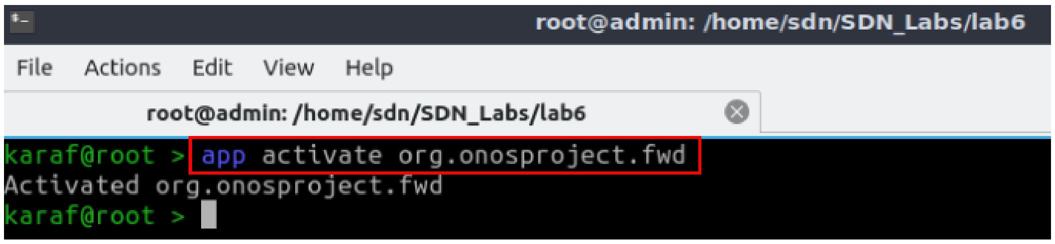
- **Hello message** (from the controller to the switch): the controller sends its version number to the switch.
- **Hello message** (from the switch to the controller): the switch replies with its supported version number.
- **Features request** (from the controller to the switch): the controller asks to see which ports are available.
- **Features reply** (from the switch to the controller): the switch replies with a list of ports, port speeds, and supported tables and actions.
- **Set Config** (from the controller to the switch): the controller asks the switch to send flow expirations.
- **Port status** (from the switch to the controller): the switch informs the controller if any ports are added, modified, or removed from the datapath.

## 4.3 Capturing PACKET_IN and PACKET_OUT messages

In this section, you will capture more OpenFlow messages exchanged between the controller and the switch after activating the ONOS forwarding application. The latter inserts flow entries into the flow table of the switches allowing them to handle IP packets.

**Step 1.** To enable the forwarding application, type the command shown below in the ONOS terminal. This command activates the forwarding application.

```
app activate org.onosproject.fwd
```


root@admin: /home/sdn/SDN_Labs/lab6
File
Actions
Edit
View
Help
root@admin: /home/sdn/SDN_Labs/lab6

```
karaf@root > app activate org.onosproject.fwd
Activated org.onosproject.fwd
karaf@root >
```

Figure 30. Activating the OpenFlow application.

**Step 2.** On the Linux terminal, click on *File>New Tab* to open an additional tab in the Linux terminal.

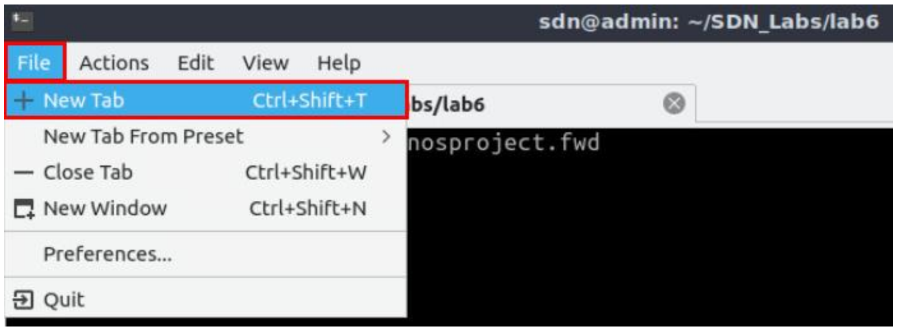
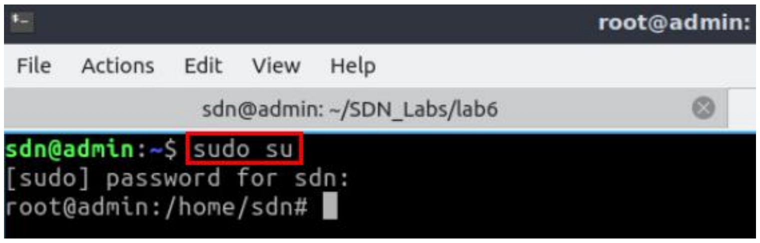


Figure 31. Opening an additional tab.

**Step 3.** Issue the command below to execute programs with the security privileges of the superuser (root). When prompted for a password, type `password`.

```
sudo su
```

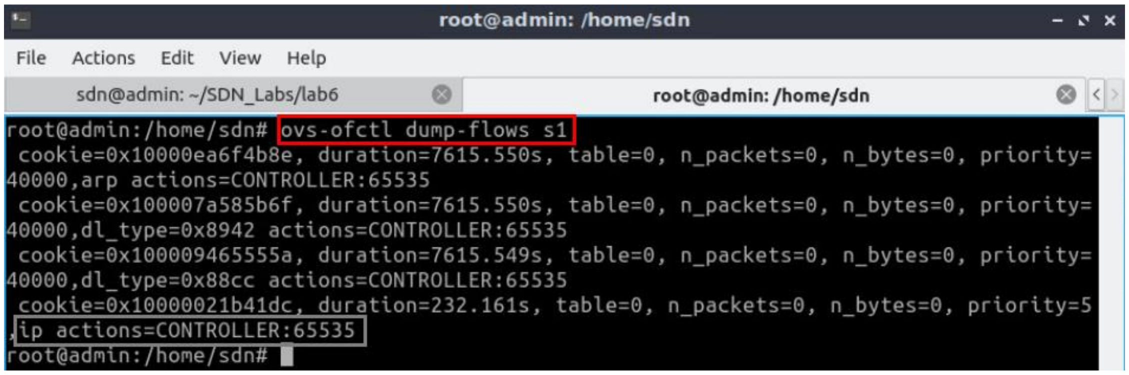


```
sdn@admin:~$ sudo su
[sudo] password for sdn:
root@admin:/home/sdn#
```

Figure 32. Switching to root mode.

**Step 4.** Issue the command below to print the flow entries of switch s1.

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



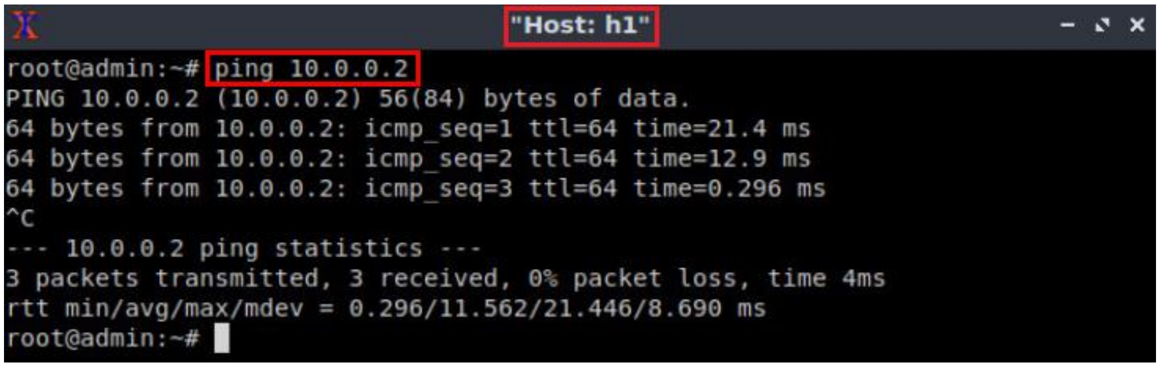
```
root@admin:/home/sdn# ovs-ofctl dump-flows s1
 cookie=0x10000ea6f4b8e, duration=7615.550s, table=0, n_packets=0, n_bytes=0, priority=
40000,arp actions=CONTROLLER:65535
 cookie=0x100007a585b6f, duration=7615.550s, table=0, n_packets=0, n_bytes=0, priority=
40000,dl_type=0x8942 actions=CONTROLLER:65535
 cookie=0x100009465555a, duration=7615.549s, table=0, n_packets=0, n_bytes=0, priority=
40000,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x10000021b41dc, duration=232.161s, table=0, n_packets=0, n_bytes=0, priority=5
,ip actions=CONTROLLER:65535
root@admin:/home/sdn#
```

Figure 33. Showing the flow entries of switch s1.

Consider Figure 33. Instead of manually adding entries in switch s1 flow table, the ONOS controller inserted the rules above to discover the topology, as well as to manage incoming IP packets by forwarding them to the controller (`ip actions=CONTROLLER:65535`).

**Step 5.** On host h1 terminal, ping host h2 and observe the captured packets in Wireshark. To do this, write the following command. Then, you can stop the ping test pressing `Ctrl+c`.

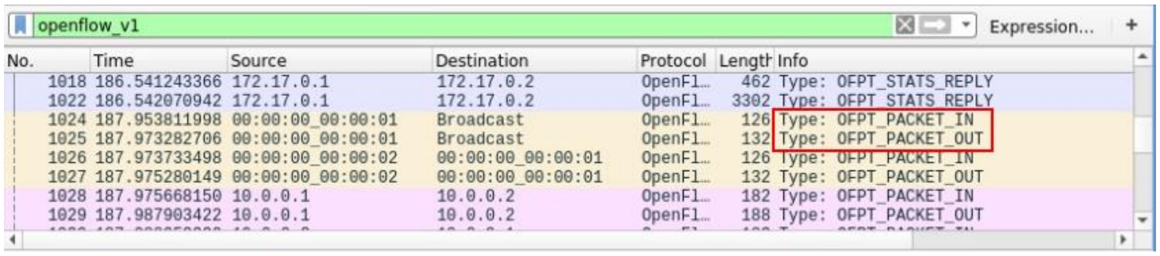
```
ping 10.0.0.2
```



```
root@admin:~# 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=21.4 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=12.9 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.296 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 4ms
rtt min/avg/max/mdev = 0.296/11.562/21.446/8.690 ms
root@admin:~#
```

Figure 34. Pinging host h2 from host h1.

**Step 6.** Go to the Wireshark window and inspect the exchanged OpenFlow packets.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1018 | 186.541243366 | 172.17.0.1 | 172.17.0.2 | OpenFl… | 462 | Type: OFPT_STATS_REPLY |
| 1022 | 186.542070942 | 172.17.0.1 | 172.17.0.2 | OpenFl… | 3302 | Type: OFPT_STATS_REPLY |
| 1024 | 187.953811998 | 00:00:00_00:00:01 | Broadcast | OpenFl… | 126 | Type: OFPT_PACKET_IN |
| 1025 | 187.973282706 | 00:00:00_00:00:01 | Broadcast | OpenFl… | 132 | Type: OFPT_PACKET_OUT |
| 1026 | 187.973733498 | 00:00:00_00:00:02 | 00:00:00_00:00:01 | OpenFl… | 126 | Type: OFPT_PACKET_IN |
| 1027 | 187.975280149 | 00:00:00_00:00:02 | 00:00:00_00:00:01 | OpenFl… | 132 | Type: OFPT_PACKET_OUT |
| 1028 | 187.975668150 | 10.0.0.1 | 10.0.0.2 | OpenFl… | 182 | Type: OFPT_PACKET_IN |
| 1029 | 187.987903422 | 10.0.0.1 | 10.0.0.2 | OpenFl… | 188 | Type: OFPT_PACKET_OUT |

Figure 35. Capturing OpenFlow packets using Wireshark.

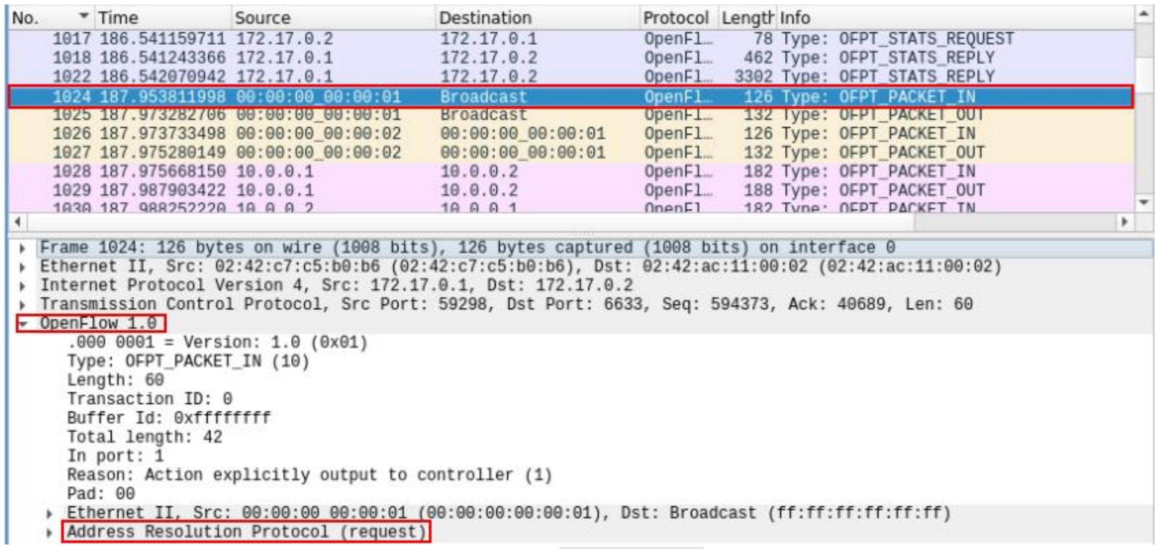
Consider Figure 35. During the pinging process from host h1 to host h2, you will notice a number of OpenFlow packets of the following types:

- `PACKET_IN`: the switch sends this message to the controller when a packet is received and did not match any entry in the flow table of the switch.
- `PACKET_OUT`: the controller sends a packet out of one or more switch ports.

Other OpenFlow packet types include:

- `OFPT_STATS_REQUEST`: the controller sends this message type to query the current state of the datapath.
- `OFPT_STATS_REPLY`: the switch responds to the request sent by the controller (OFPT_STATS_REQUEST).

**Step 7.** Click on the first `PACKET_IN` captured packet and expand the header field `OpenFlow 1.0` as shown in the below figure.



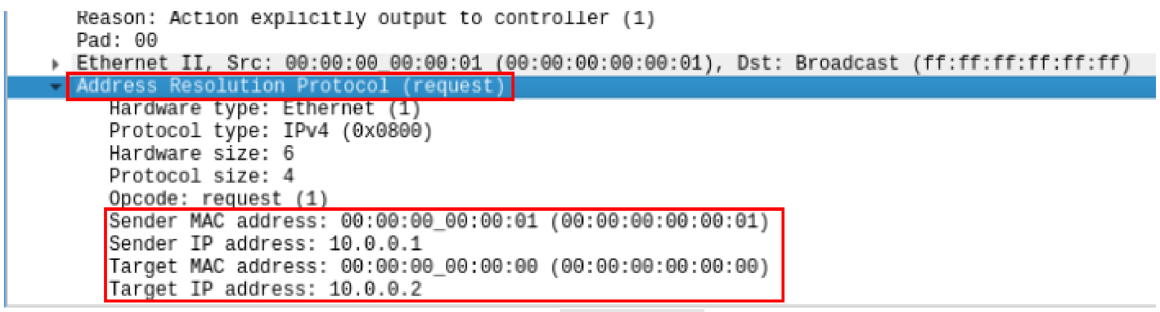
```
No.    Time          Source             Destination        Protocol Length Info
  1017 186.541159711 172.17.0.2         172.17.0.1         OpenFl…     78 Type: OFPT_STATS_REQUEST
  1018 186.541243366 172.17.0.1         172.17.0.2         OpenFl…    462 Type: OFPT_STATS_REPLY
  1022 186.542070942 172.17.0.1         172.17.0.2         OpenFl…   3302 Type: OFPT_STATS_REPLY
  1024 187.953811998 00:00:00_00:00:01  Broadcast          OpenFl…    126 Type: OFPT_PACKET_IN
  1025 187.973282706 00:00:00_00:00:01  Broadcast          OpenFl…    132 Type: OFPT_PACKET_OUT
  1026 187.973733498 00:00:00_00:00:02  00:00:00_00:00:01  OpenFl…    126 Type: OFPT_PACKET_IN
  1027 187.975280149 00:00:00_00:00:02  00:00:00_00:00:01  OpenFl…    132 Type: OFPT_PACKET_OUT
  1028 187.975668150 10.0.0.1           10.0.0.2           OpenFl…    182 Type: OFPT_PACKET_IN
  1029 187.987903422 10.0.0.1           10.0.0.2           OpenFl…    188 Type: OFPT_PACKET_OUT
  1030 187.988252220 10.0.0.2           10.0.0.1           OpenFl…    182 Type: OFPT_PACKET_IN
```

```
▸ Frame 1024: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface 0
▸ Ethernet II, Src: 02:42:c7:c5:b0:b6 (02:42:c7:c5:b0:b6), Dst: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
▸ Internet Protocol Version 4, Src: 172.17.0.1, Dst: 172.17.0.2
▸ Transmission Control Protocol, Src Port: 59298, Dst Port: 6633, Seq: 594373, Ack: 40689, Len: 60
▾ OpenFlow 1.0
    .000 0001 = Version: 1.0 (0x01)
    Type: OFPT_PACKET_IN (10)
    Length: 60
    Transaction ID: 0
    Buffer Id: 0xffffffff
    Total length: 42
    In port: 1
    Reason: Action explicitly output to controller (1)
    Pad: 00
  ▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Address Resolution Protocol (request)
```

Figure 36. Inspecting the first PACKET_IN packet in OpenFlow.

Consider Figure 36. The first PACKET_IN packet is a broadcast message that has the source MAC address of host h1 (00:00:00:00:00:01). Initially, when host h1 pings host h2, it will request the MAC address of host h2 using an Address Resolution Protocol (ARP) request. Switch s2 will receive the request, matches it against the flow entry that deals with ARP packets (i.e., the flow installed by the activated OpenFlow application), and forwards it to the controller.

**Step 8.** In the same inspected packet, expand the Address Resolution Protocol (request) field as shown in the below figure.

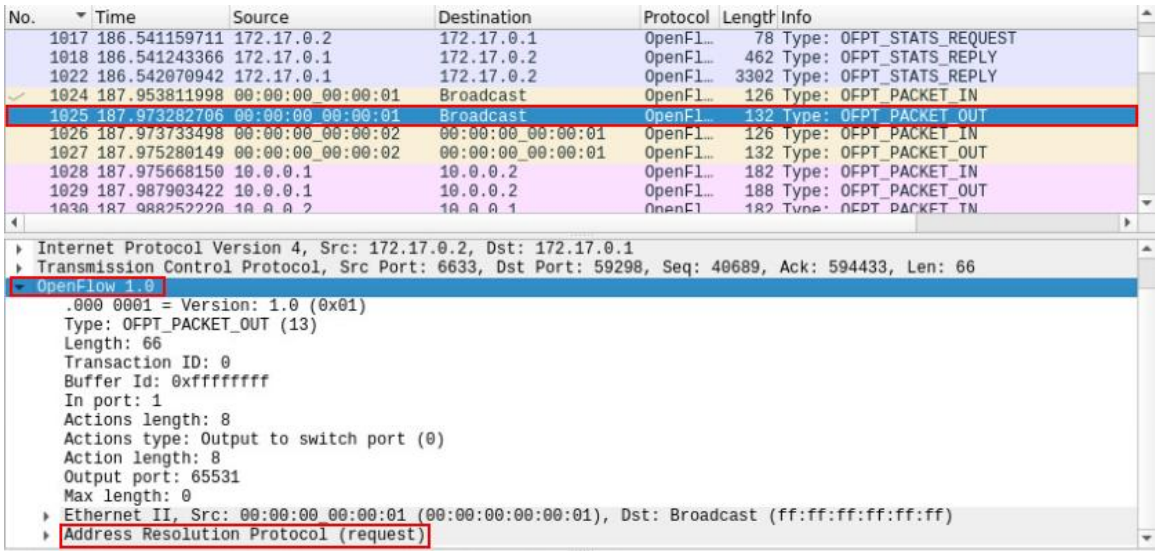


```
    Reason: Action explicitly output to controller (1)
    Pad: 00
  ▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▾ Address Resolution Protocol (request)
      Hardware type: Ethernet (1)
      Protocol type: IPv4 (0x0800)
      Hardware size: 6
      Protocol size: 4
      Opcode: request (1)
      Sender MAC address: 00:00:00_00:00:01 (00:00:00:00:00:01)
      Sender IP address: 10.0.0.1
      Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
      Target IP address: 10.0.0.2
```

Figure 37. Inspecting the first PACKET_IN packet in OpenFlow.

Consider Figure 37. In the ARP request, the sender's MAC (00:00:00:00:00:01) and IP (10.0.0.1) addresses belong to host h1, whereas the target IP address (10.0.0.2) belongs to host h2. The target MAC address is filled with zeros since host h1 does not know it yet.

**Step 9.** Click on the first PACKET_OUT captured packet and expand the header field OpenFlow 1.0 as shown in the below figure.



```
No.    Time            Source             Destination        Protocol Length Info
  1017 186.541159711 172.17.0.2         172.17.0.1         OpenFl...     78 Type: OFPT_STATS_REQUEST
  1018 186.541243366 172.17.0.1         172.17.0.2         OpenFl...    462 Type: OFPT_STATS_REPLY
  1022 186.542070942 172.17.0.1         172.17.0.2         OpenFl...   3302 Type: OFPT_STATS_REPLY
  1024 187.953811998 00:00:00_00:00:01  Broadcast          OpenFl...    126 Type: OFPT_PACKET_IN
  1025 187.973282706 00:00:00_00:00:01  Broadcast          OpenFl...    132 Type: OFPT_PACKET_OUT
  1026 187.973733498 00:00:00_00:00:02  00:00:00_00:00:01  OpenFl...    126 Type: OFPT_PACKET_IN
  1027 187.975280149 00:00:00_00:00:02  00:00:00_00:00:01  OpenFl...    132 Type: OFPT_PACKET_OUT
  1028 187.975668150 10.0.0.1           10.0.0.2           OpenFl...    182 Type: OFPT_PACKET_IN
  1029 187.987903422 10.0.0.1           10.0.0.2           OpenFl...    188 Type: OFPT_PACKET_OUT
  1030 187.988252220 10.0.0.2           10.0.0.1           OpenFl...    182 Type: OFPT_PACKET_IN

▸ Internet Protocol Version 4, Src: 172.17.0.2, Dst: 172.17.0.1
▸ Transmission Control Protocol, Src Port: 6633, Dst Port: 59298, Seq: 40689, Ack: 594433, Len: 66
▾ OpenFlow 1.0
    .000 0001 = Version: 1.0 (0x01)
    Type: OFPT_PACKET_OUT (13)
    Length: 66
    Transaction ID: 0
    Buffer Id: 0xffffffff
    In port: 1
    Actions length: 8
    Actions type: Output to switch port (0)
    Action length: 8
    Output port: 65531
    Max length: 0
  ▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Address Resolution Protocol (request)
```

Figure 38. Inspecting the first `PACKET_OUT` packet in OpenFlow.

Consider Figure 38. The first `PACKET_OUT` packet is a broadcast message that has the source MAC address of host h1 (00:00:00:00:00:01). This message (`PACKET_OUT`) is an ARP request that is sent from the controller to discover the MAC address of host h2. The subsequent `PACKET_IN` packet will include an ARP reply from host h2 with its MAC address to the controller. Consequently, the controller will send this ARP reply message to host h1 so that it knows the MAC address of host h2.

**Step 10.** Stop capturing packets in Wireshark by clicking the red icon.

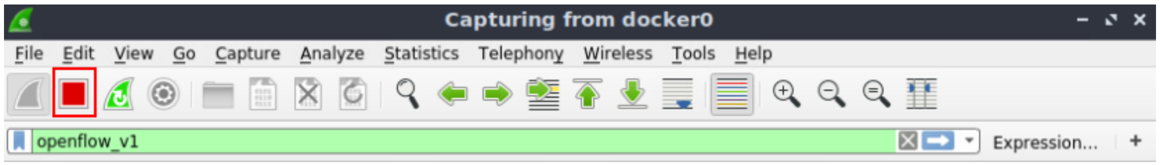


Figure 39. Stop Capturing Wireshark packets.

This concludes Lab 6. 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: enabling innovation in campus networks*." ACM SIGCOMM Computer Communication Review 38, no. 2 (2008): 69-74.
3. P. Goransson, C. Black, T. Culver. *"Software defined networks: a comprehensive approach"*. Morgan Kaufmann, 2016.
4. P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, "*ONOS: towards an open, distributed SDN OS*," In Proceedings of the third workshop on Hot topics in software defined networking, pp. 1-6, 2014.

# SOFTWARE DEFINED NETWORKING

# Exercise 3: OpenFlow Protocol Management

**Document Version: 09-02-2021**

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

## Contents

## 1 Exercise description

Consider Figure 1. The topology consists of two end-hosts, two switches and a controller within a Software Defined Networking (SDN) network. The blue devices represent OpenFlow switches. All switches are connected to the controller c0.

The goal of this exercise is to connect the two hosts by managing the switches via the OpenFlow protocol. Initially, you should administer the OpenFlow switches without running the ONOS controller, where you will install flow entries to forward the traffic between host h1 and host h2. Later, you should run ONOS, activate some of its applications, and inspect the OpenFlow messages exchanged between the control plane and data plane. The topology below is already built and you should use Mininet to emulate it.

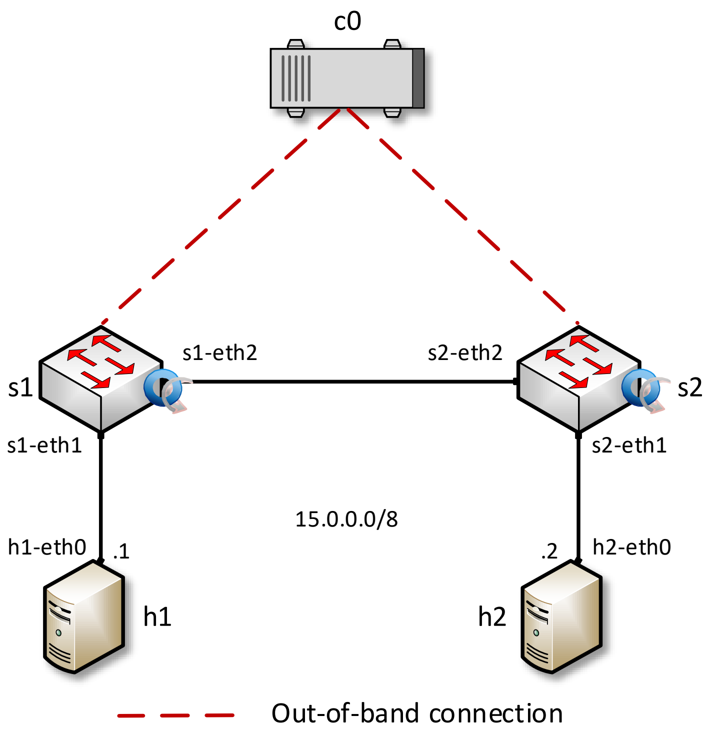


Figure 1. Exercise topology.

### 1.1 Topology settings

The devices are already configured according to Table 1.

Table 1. Topology information.

| Device | Interface | MAC Address | IP Address | Subnet |
|---|---|---|---|---|
| h1 | h1-eth0 | 00:00:00:00:00:01 | 15.0.0.1 | /8 |
| h2 | h2-eth0 | 00:00:00:00:00:02 | 15.0.0.2 | /8 |
| c0 | N/A | N/A | 172.17.0.2 | /16 |

## 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)** Open MiniEdit and load the topology above. The topology file *Exercise3.mn* of this exercise is in the directory *~/SDN_Labs/Exercise3* as shown in the figure below.

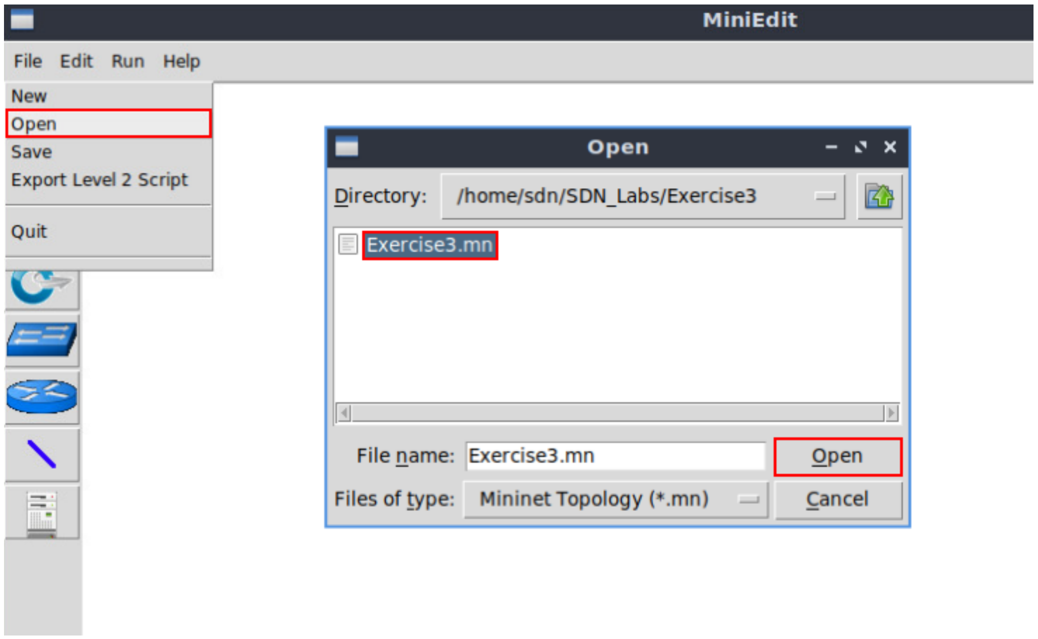


Figure 2. Loading the topology file in Mininet.

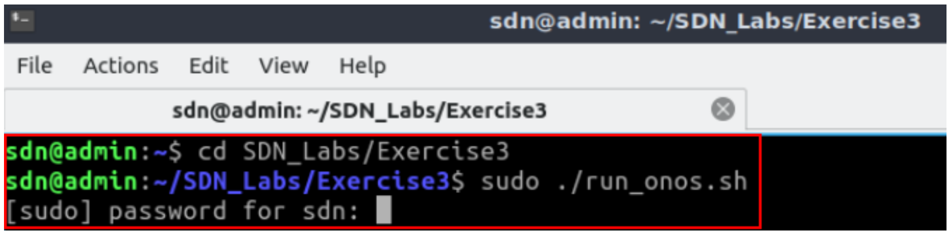
**b)** Using the command line tool that monitors and administers OpenFlow switches, inspect the flow tables of switches s1 and s2. Validate that there are no entries inserted in the switches.

**c)** Insert flow entries on switches s1 and s2 to connect hosts h1 and h2. The flow entries should match incoming traffic based on the port number.

**d)** Inspect the flow tables of the switches and validate the added entries. Additionally, test the connectivity between the hosts by performing a ping test from host h1 to host h2.

**e)** Delete all the added flow entries on the switches and run the ONOS controller.

**f)** In the Linux terminal, navigate to the directory *~/SDN_Labs/Exercise3* and execute, in superuser mode, the script *run_onos.sh* that runs the ONOS controller. When prompted for a password, type `password`. The steps to run ONOS are depicted in the figure below.


sdn@admin: ~/SDN_Labs/Exercise3
File
Actions
Edit
View
Help
sdn@admin: ~/SDN_Labs/Exercise3

```
sdn@admin:~$ cd SDN_Labs/Exercise3
sdn@admin:~/SDN_Labs/Exercise3$ sudo ./run_onos.sh
[sudo] password for sdn:
```

Figure 3. Starting the ONOS controller.

**g)** Activate the necessary ONOS application that allows the controller to communicate with the OpenFlow switches. Using Wireshark, capture on *docker0* interface (i.e., where the controller is running) the first OpenFlow messages (e.g., the first four OpenFlow messages) exchanged when activating the application and explain their functionality.

**h)** Activate the necessary ONOS application that enables IP forwarding. Inspect the newly added entries in the flow tables of the switches.

**i)** Test the connectivity between the hosts by performing a ping test from host h1 to host h2. While the connectivity test is running, you should capture the necessary OpenFlow packets exchanged between the controller and the switches that allow the hosts to communicate. Explain briefly how the two hosts can communicate starting from the first message sent by host h1.

# SOFTWARE DEFINED NETWORKING

# Lab 7: Routing within an SDN network

**Document Version: 07-08-2021**

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

## Contents

## Overview

The focus in this lab is to perform Internet Protocol (IP) routing within the Software Defined Networking (SDN) network via the reactive-routing application. The reactive-routing application is dependent on the SDN-IP application, which allows SDN networks to connect to legacy networks using the standard Border Gateway Protocol (BGP).

## Objectives

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

1. Understand the concept of SDN network.
2. Understand how the ONOS controller interacts with the legacy router.
3. Configure BGP on a legacy router.
4. Activate the SDN-IP application.
5. Activate the reactive-routing application.
6. Perform IP routing within the SDN network.

## Lab settings

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

Table 1**.** Credentials to access the 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: Starting the ONOS controller.
4. Section 4: Integrating SDN and BGP.
5. Section 5: Activating reactive-routing application and verifying the connectivity between the networks.

## 1 Introduction

SDN provides a set of abstractions delivered by the control plane, such as a global network view, network applications, the flexibility of testing new protocols, and centralized management. Integrating the widely deployed Internet infrastructure with SDN represents a significant challenge that requires innovative approaches to increase the deployment of SDN networks. During the transition, SDN networks need to coexist with traditional IP networks. Any SDN deployment must be able to exchange reachability information.

## 1.1 Legacy routing vs SDN routing

Routing protocols were essential to respond to rapidly changing network conditions. However, these conditions no longer exist in modern data centers. Typically, legacy routing protocols work as a distributed system transmitting the connection status information over the link and, each router performs the routing computation. A drawback of this scheme is that the routing information relies on flooding the link state to update information among routers. Therefore, this incurs in longer convergence time where the link delay affects the convergence time[1].

SDN is a new paradigm that solves the problem mentioned above by creating a centralized approach rather than a distributed one. The central concept of SDN is to separate the control plane from the data plane to maximize the efficiency of data plane devices. Moving the control software outside the device simplifies the network management and enables optimal routing. Also, SDN routing provides robustness, scalability, and self-healing[2].

## 1.2 Integrating BGP with an SDN network

SDN networks operate differently from legacy networks. One of the obstacles to deploying an SDN network is integrating it with the existing IP networks[1]. Usually, peering between Autonomous Systems (ASes) on the Internet is traditionally done using BGP. Therefore, a precise mechanism is needed for an SDN Autonomous System (AS) to communicate with other ASes via BGP[1].

In this lab, the SDN switches use the Open Network Operating System (ONOS) controller, which provides applications with the capability to enable several functionalities such as connecting to external networks. The ONOS application that performs the function mentioned above is called SDN-IP. SDN-IP allows an SDN network to connect to external networks on the Internet using the standard BGP[3].

## 1.3 Establishing a virtual gateway via the reactive-routing application

In a legacy IP network, hosts use gateway as the default router to access the Internet. However, in SDN networks, OpenFlow switches are used instead of routers to connect the network by inserting flow entries into these switches. Thus, there is no physical gateway router in the SDN network. As a result, hosts within the SDN network will not reach other networks without a default gateway as the next hop to the sent packets. Additionally, the Media Access Control (MAC) address of the next-hop is unknown to the host, leading to insufficient information to compose the packet and send it out.

To mitigate this issue, the ONOS controller includes the reactive-routing application that establishes a virtual gateway for SDN networks. Once the host is assigned a default gateway address, it will send out an Address Resolution Protocol (ARP) packet to look for the MAC address. Since there is no physical gateway in SDN, the virtual gateway module in ONOS will handle all ARP requests. It will compose the ARP reply packet and send it out as packet-out to the host.

Every SDN network only needs one virtual gateway. This virtual gateway has one MAC address only and may have several gateway IP addresses. The MAC address of the virtual gateway corresponds to the BGP speaker (the legacy router that uses BGP) within the SDN network. Essentially, the existence of a BGP speaker is required for establishing a virtual gateway within the SDN network.

Consider Figure 1. Typically, an SDN network is composed of various OpenFlow switches (switches s1, s2, and s3) and a BGP router (router r1) connected to the controller (c0). The SDN-IP application runs on top of the ONOS controller and, it is connected to the BGP speaker (router r1) within the SDN network through an Internal BGP (IBGP) session[4].

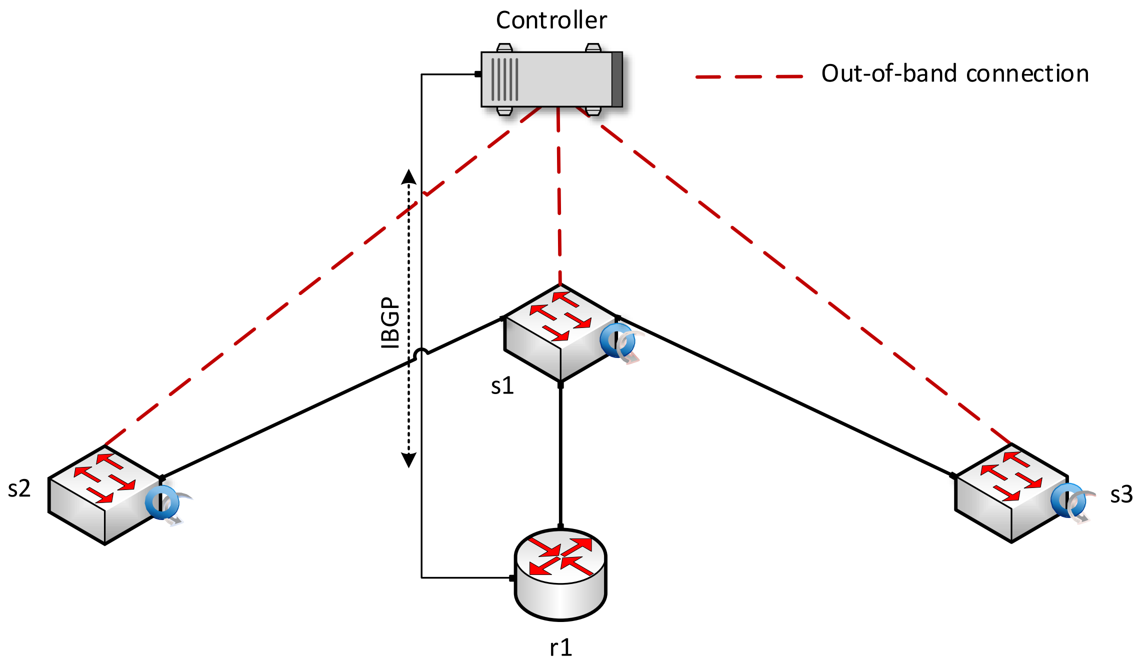


Figure 1. Integrating SDN and IP networks.

## 2 Lab topology

Consider Figure 2. The topology consists of one SDN network (AS 100). Router r1 runs BGP and acts as a BGP speaker. Router r1 is connected to the controller in order to propagate the BGP advertisements to the SDN-IP application running on top of the ONOS controller. Router r1 and the controller are connected via the network 10.0.0.0/24.

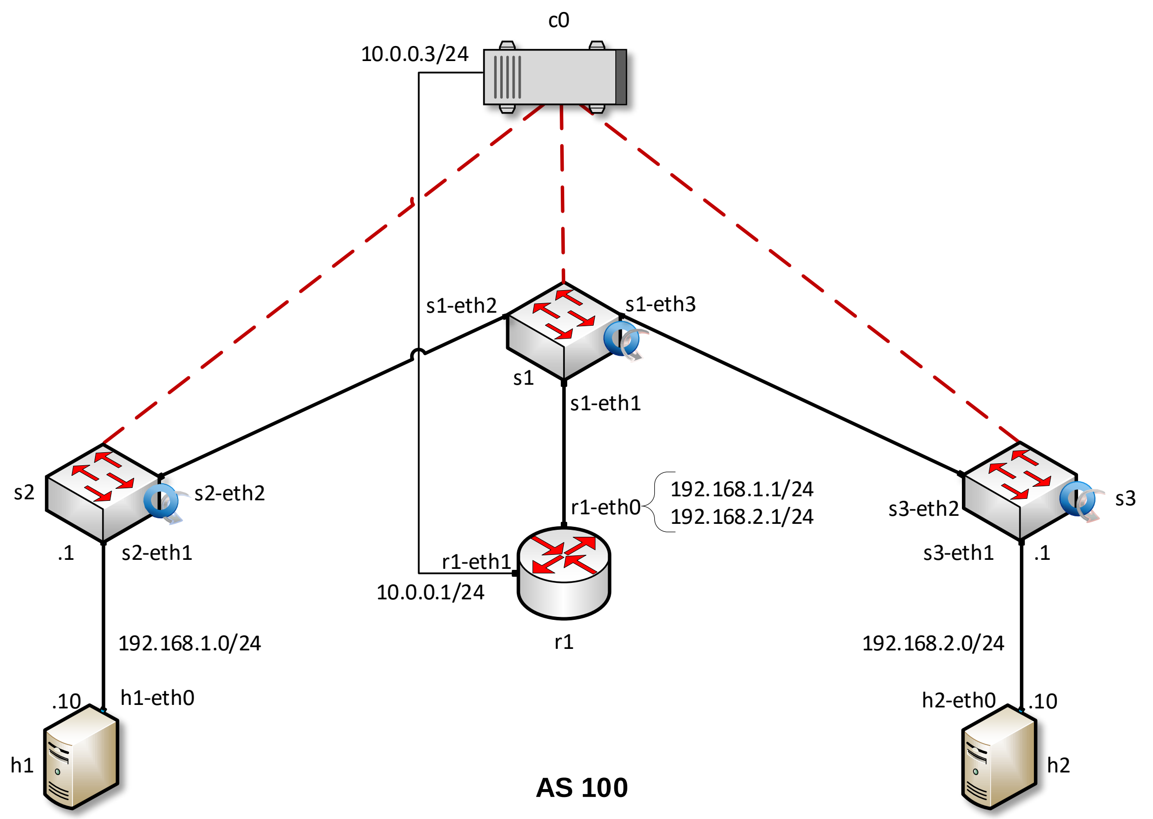


Figure 2. Lab topology.

## 2.1 Lab settings

The devices are already configured according to Table 2.

Table 2. Topology information.

| Device | Interface | MAC address | IP Address | Subnet | Default gateway |
|---|---|---|---|---|---|
| Router r1 | r1-eth0 | 00:00:00:00:01:01 | 192.168.1.1 | /24 | N/A |
| | | 00:00:00:00:01:01 | 192.168.2.1 | /24 | N/A |
| | r1-eth1 | 00:00:00:00:01:02 | 10.0.0.1 | /24 | N/A |
| Switch s2 | s2-eth1 | N/A | 192.168.1.1 | /24 | N/A |
| Switch s3 | s3-eth1 | N/A | 192.168.2.1 | /24 | N/A |
| Host h1 | h1-eth0 | 00:00:00:00:00:01 | 192.168.1.10 | /24 | 192.168.1.1 |
| Host h2 | h2-eth0 | 00:00:00:00:00:02 | 192.168.2.10 | /24 | 192.168.2.1 |
| c0 | N/A | N/A | 172.17.0.2 | /16 | N/A |
| | N/A | N/A | 10.0.0.3 | /24 | N/A |

## 2.2 Loading the 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 such as: configuring network elements (i.e., IP addresses, default gateway), saving the topologies, and exporting layer 2 models.

**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`.

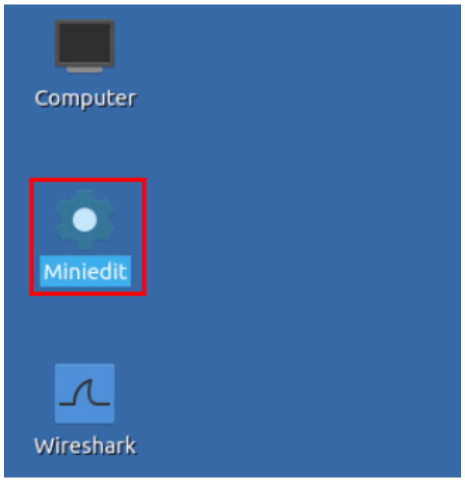
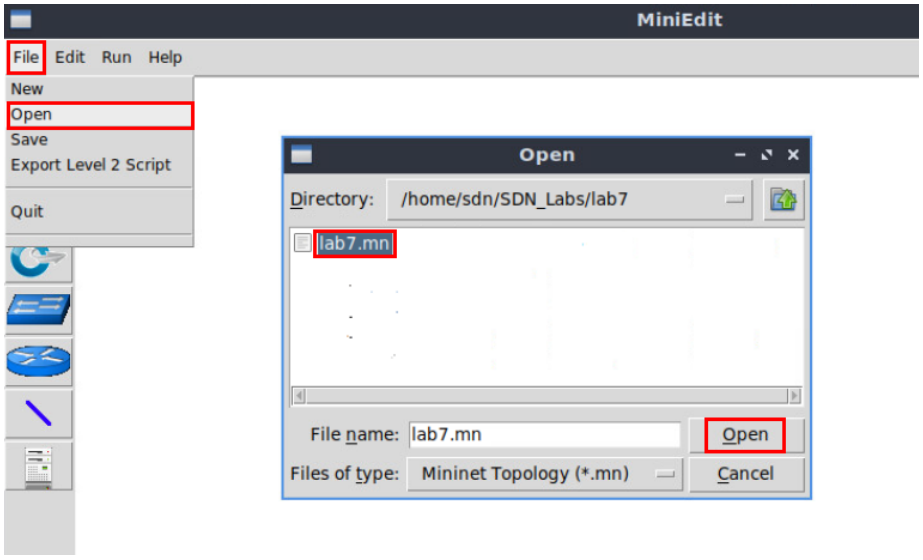


Figure 3. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then *open* to load the lab's topology. Open the *Lab7.mn* topology file stored in the default directory, */home/sdn/SDN_Labs /lab7* and click on *Open*.



Directory: /home/sdn/SDN_Labs/lab7

lab7.mn

File name: lab7.mn

Files of type: Mininet Topology (*.mn)

Figure 4. Opening topology.

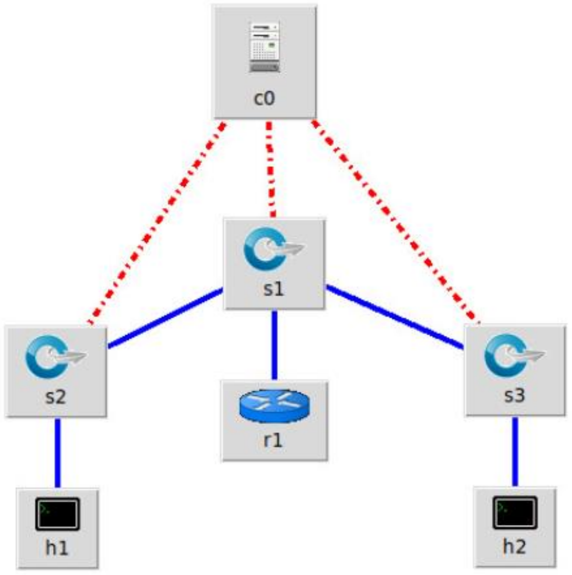


Figure 5. MiniEdit's topology.

Consider Figure 5. The direct link between router r1 and controller c0 is not yet established since MiniEdit does not allow that. In section 4, you will execute a script to connect the controller and the router via the network 10.0.0.0/24.

## 2.3 Loading the configuration file

At this point, the topology is loaded. However, the interfaces are not configured. In order to assign IP addresses to the interfaces of the device, you will execute a script that loads the configuration to the routers.

**Step 1.** Click on the icon below to open the Linux terminal.

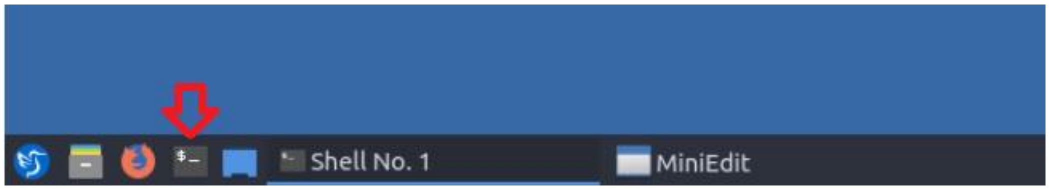
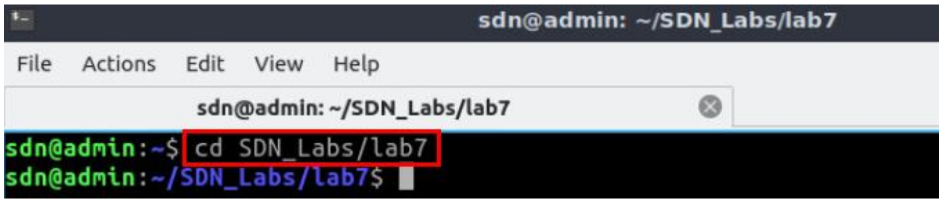


Figure 6. Opening Linux terminal.

**Step 2.** Click on the Linux terminal and navigate into *SDN_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 IP addresses to the interfaces of the router. The `cd` command is short for change directory followed by an argument that specifies the destination directory.

```
cd SDN_Labs/lab7
```

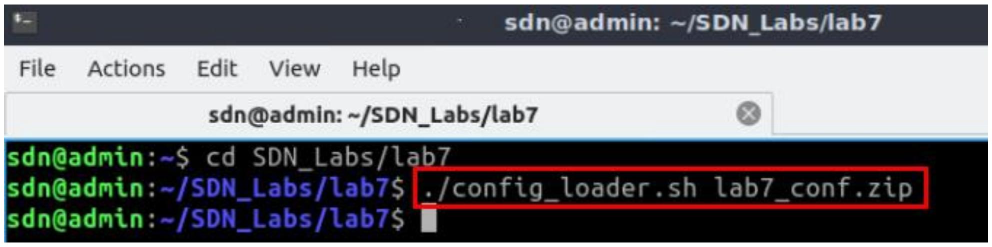

sdn@admin: ~/SDN_Labs/lab7
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab7

```
sdn@admin:~$ cd SDN_Labs/lab7
sdn@admin:~/SDN_Labs/lab7$
```

Figure 7. Entering the SDN_Labs/lab7 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 lab7_conf.zip
```

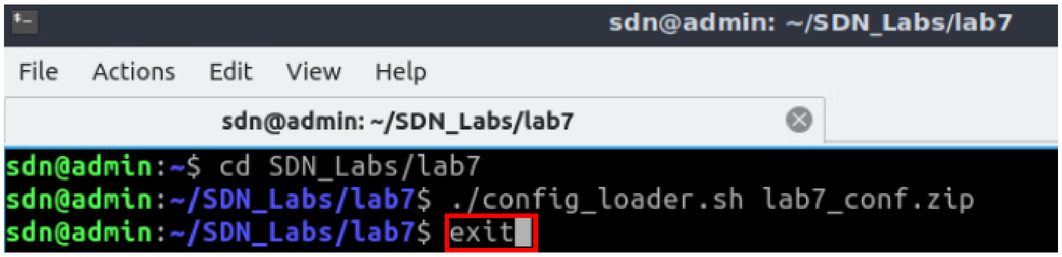

sdn@admin: ~/SDN_Labs/lab7
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab7

```
sdn@admin:~$ cd SDN_Labs/lab7
sdn@admin:~/SDN_Labs/lab7$ ./config_loader.sh lab7_conf.zip
sdn@admin:~/SDN_Labs/lab7$
```

Figure 8. Executing the shell script to load the configuration.

**Step 4.** Type the following command to exit the Linux terminal.

```
exit
```


sdn@admin: ~/SDN_Labs/lab7
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab7

```
sdn@admin:~$ cd SDN_Labs/lab7
sdn@admin:~/SDN_Labs/lab7$ ./config_loader.sh lab7_conf.zip
sdn@admin:~/SDN_Labs/lab7$ exit
```

Figure 9. Exiting from the terminal.

## 2.4 Running the emulation

In this section, you will run the emulation and check the links and interfaces that connect the devices in the given topology.

**Step 1.** At this point, host h1 and host h2 interfaces are configured. To proceed with the emulation, click on the *Run* button located on the lower left-hand side.

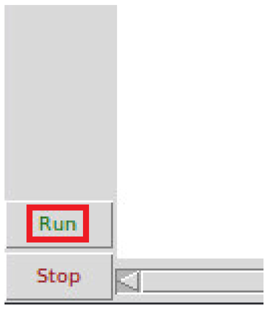
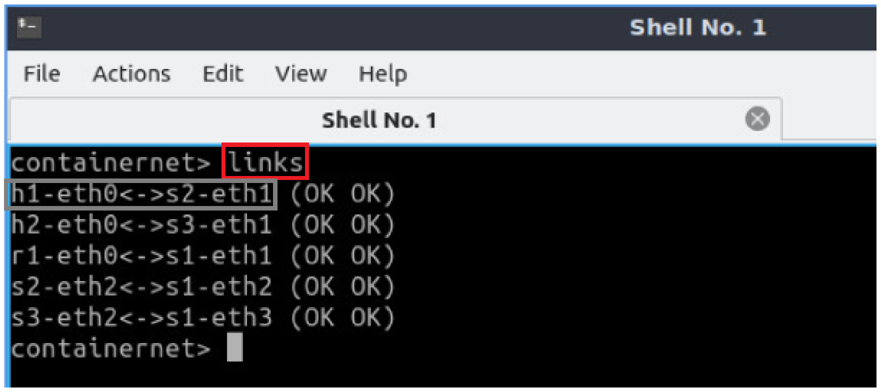


Figure 10. Starting the emulation.

**Step 2.** Issue the following command on Mininet terminal to display the interface names and connections.

```
links
```



```
containernet> links
h1-eth0<->s2-eth1 (OK OK)
h2-eth0<->s3-eth1 (OK OK)
r1-eth0<->s1-eth1 (OK OK)
s2-eth2<->s1-eth2 (OK OK)
s3-eth2<->s1-eth3 (OK OK)
containernet> 
```

Figure 11. Displaying network interfaces.

In Figure 11, the link displayed within the gray box indicates that interface *eth0* of host h1 connects to interface *eth1* of switch s2 (i.e., *h1-eth0<->s2-eth1*).

## 2.5 Verifying the configuration

You will verify the IP addresses listed in Table 2 and inspect the routing table of router r1.

**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.

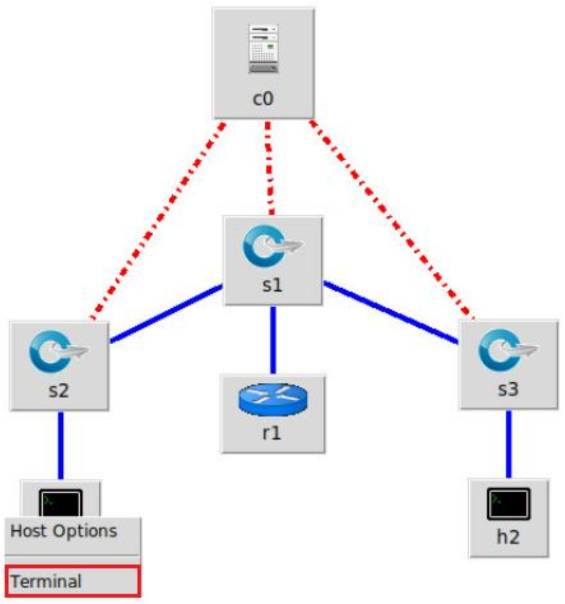
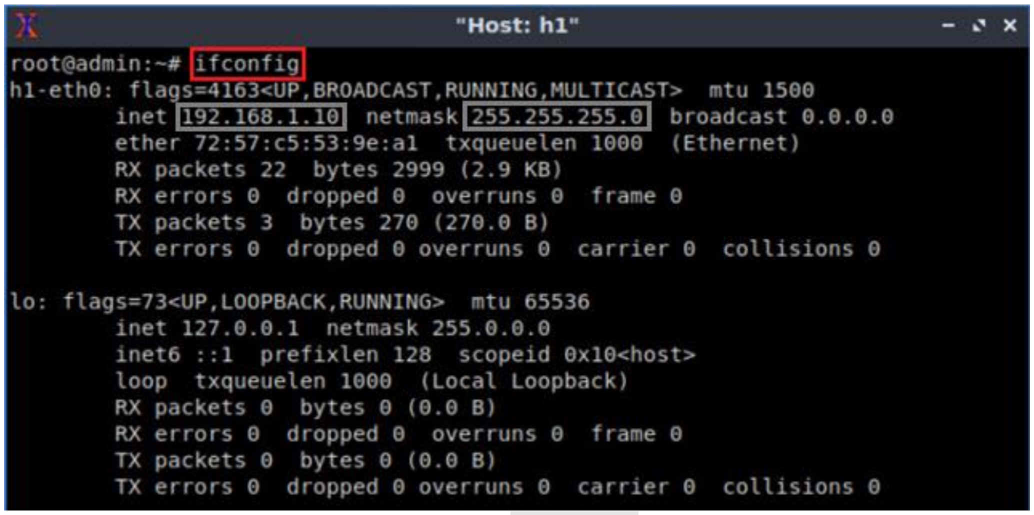


Figure 12. Opening a terminal on host h1.

**Step 2**. On host h1 terminal, type the command shown below to verify that the IP address was assigned successfully. You will corroborate that host h1 has two interfaces. Interface *h1-eth0* is configured with the IP address 192.168.1.10 and the subnet mask 255.255.255.0. Interface *lo* is configured with the IP address 127.0.0.1 and the subnet mask 255.0.0.0.

```
ifconfig
```

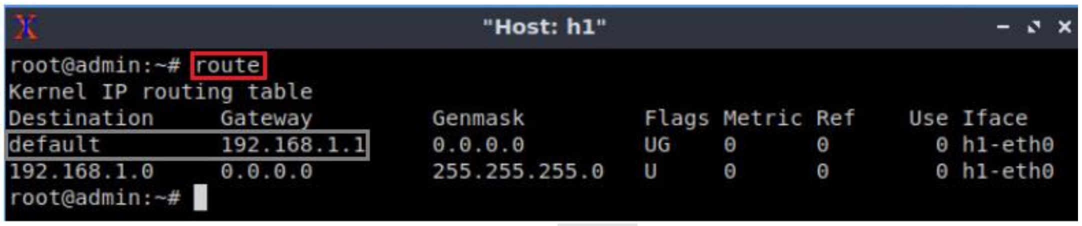


```
"Host: h1"
root@admin:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.10  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 72:57:c5:53:9e:a1  txqueuelen 1000  (Ethernet)
        RX packets 22  bytes 2999 (2.9 KB)
        RX errors 0  dropped 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
```

Figure 13. Output of `ifconfig` command.

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

```
route
```



```
root@admin:~# 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:~#
```

Figure 14. Output of `route` command.

**Step 4**. In order to verify host h2 IP address and default gateway, proceed similarly by repeating step 1 to step 3 on host h2 terminal. Similar results should be observed.

**Step 5**. In order to verify router r1, hold right-click on router r1 and select *Terminal*.

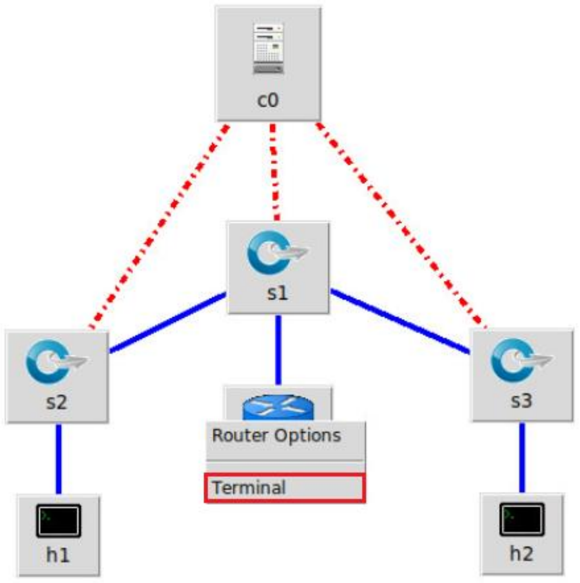
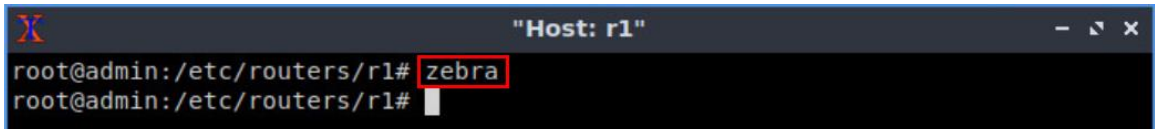


Figure 15. Opening a terminal on router r1.

**Step 6**. In this step, you will start the zebra daemon, 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 zebra, type the following command.

```
zebra
```

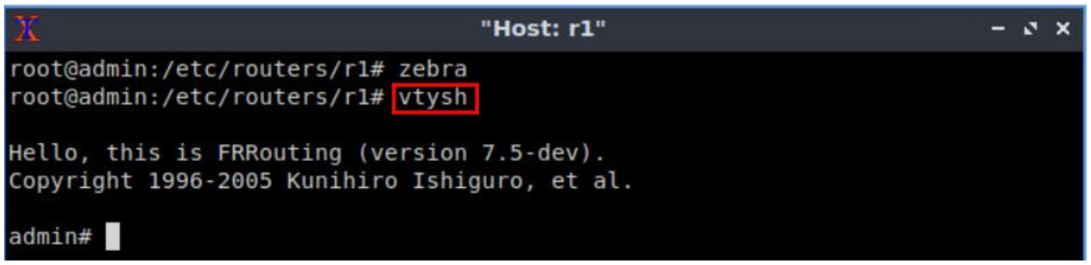


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1#
```

Figure 16. Starting zebra daemon.

**Step 7**. After initializing zebra, vtysh should be started in order to provide all the Command Line Interface (CLI) commands defined by the daemons. To proceed, issue the following command.

```
vtysh
```

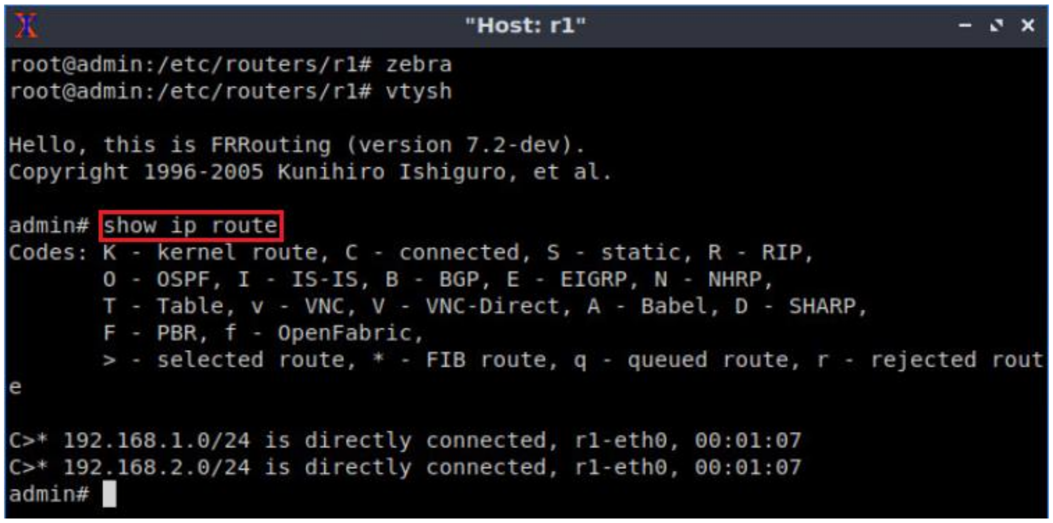


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.5-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin#
```

Figure 17. Starting vtysh on router r1.

**Step 8.** Type the following command on router r1 terminal to verify the routing table of router r1. It will list all the directly connected networks.

```
show ip route
```



```
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# 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 rout
e

C>* 192.168.1.0/24 is directly connected, r1-eth0, 00:01:07
C>* 192.168.2.0/24 is directly connected, r1-eth0, 00:01:07
admin#
```

Figure 18. Displaying the routing table of router r1.

The output in the figure above shows that the networks 192.168.1.0/24 and 192.168.2.0/24 are directly connected through the interface *r1-eth0*.

## 3 Starting the ONOS controller

In this section, you will start the ONOS controller and activate OpenFlow application so that the controller discovers the devices, hosts, and links in the topology.

**Step 1.** Go to the Linux terminal, Shell No.1.

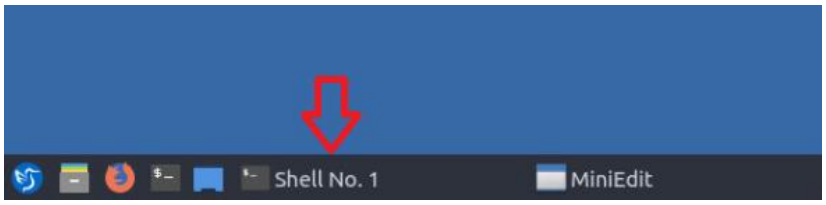


Figure 19. Opening Linux terminal.

**Step 2.** Click on *File>New Tab* to open an additional tab in the Linux terminal. Alternatively, you may press `Ctrl+Shift+T`.

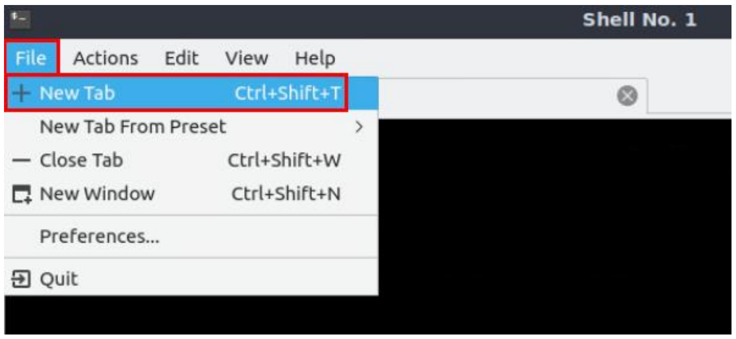
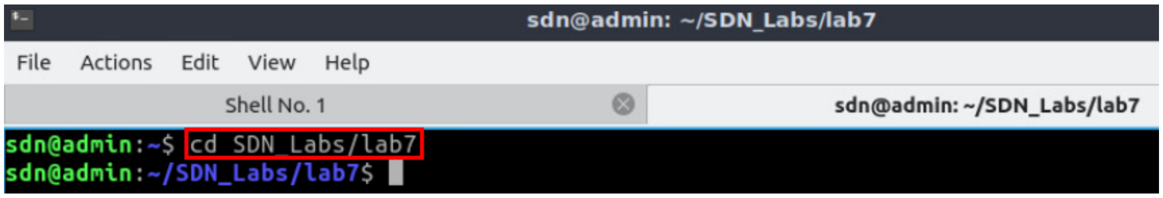


Figure 20. Opening an additional tab.

**Step 3.** Navigate into *SDN_Labs/lab7* directory by issuing the following command.

```
cd SDN_Labs/lab7
```

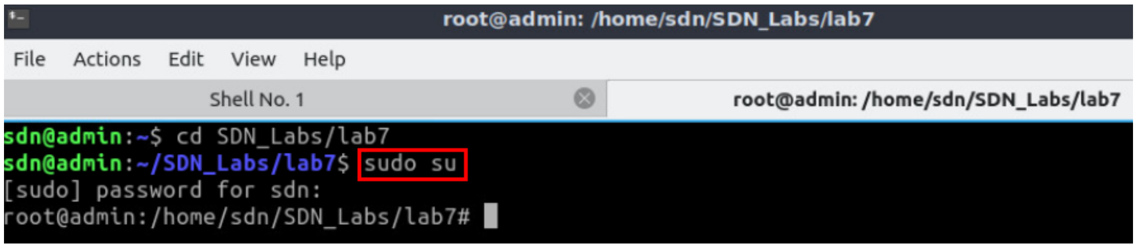


```
sdn@admin:~$ cd SDN_Labs/lab7
sdn@admin:~/SDN_Labs/lab7$
```

Figure 21. Entering the SDN_Labs/lab7 directory.

**Step 4.** Issue the command below to execute programs with the security privileges of the superuser (root). When prompted for a password, type `password`.

```
sudo su
```

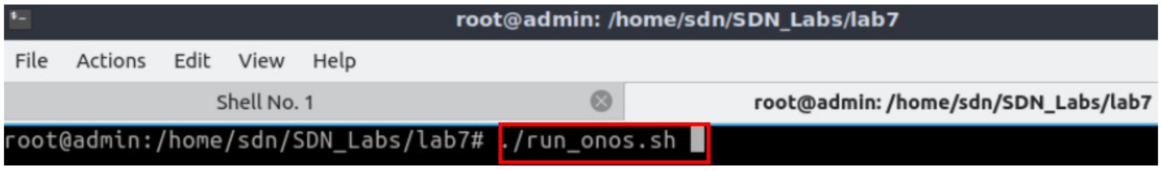

root@admin: /home/sdn/SDN_Labs/lab7
File
Actions
Edit
View
Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

```
sdn@admin:~$ cd SDN_Labs/lab7
sdn@admin:~/SDN_Labs/lab7$ sudo su
[sudo] password for sdn:
root@admin:/home/sdn/SDN_Labs/lab7#
```

Figure 22. Switching to root mode.

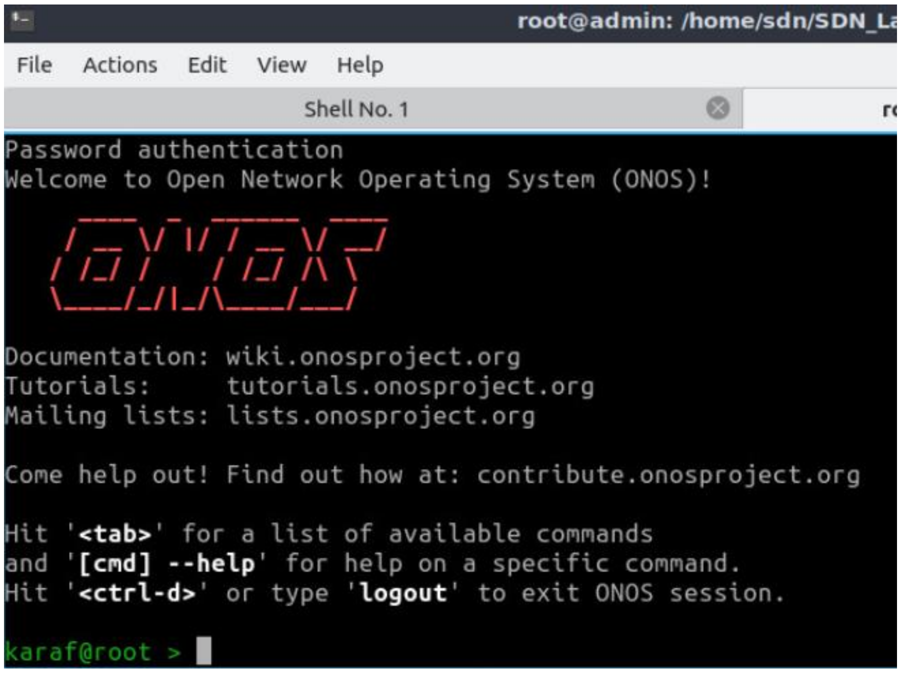
**Step 5.** A script was written to run ONOS and enter its CLI. In order to run the script, issue the following command. In addition to running ONOS, the script will modify the MAC addresses of the hosts and the router so that they conform with the topology.

```
./run_onos.sh
```


root@admin: /home/sdn/SDN_Labs/lab7
File
Actions
Edit
View
Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

```
root@admin:/home/sdn/SDN_Labs/lab7# ./run_onos.sh
```

Figure 23. Starting the ONOS controller.

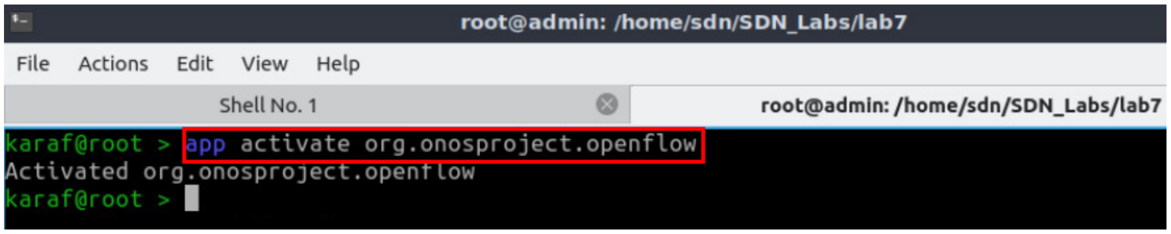


```
Password authentication
Welcome to Open Network Operating System (ONOS)!

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

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

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

karaf@root >
```

Figure 24. ONOS CLI.

**Step 6.** On the ONOS terminal, issue the following command to activate the OpenFlow application.

```
app activate org.onosproject.openflow
```


root@admin: /home/sdn/SDN_Labs/lab7
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

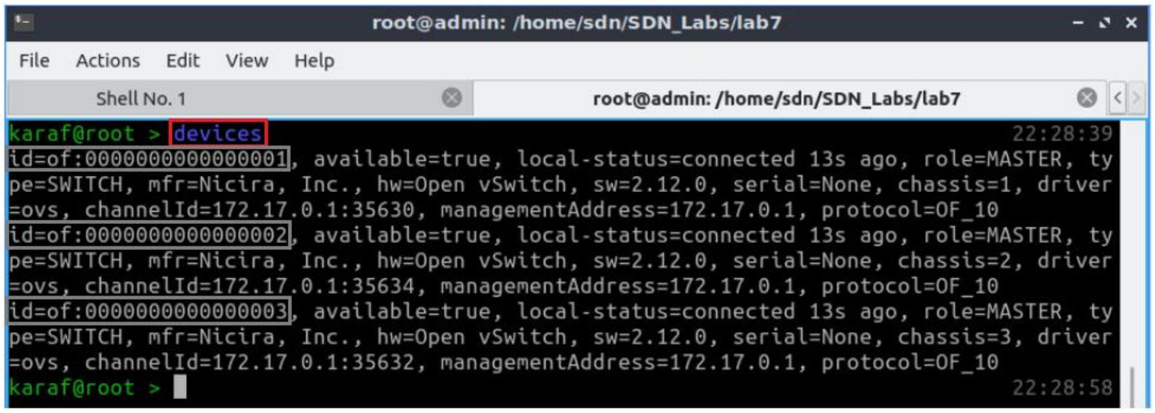
```
karaf@root > app activate org.onosproject.openflow
Activated org.onosproject.openflow
karaf@root >
```

Figure 25. Activating OpenFlow application.

Note that when you activate any ONOS application, you may have to wait few seconds so that the application gives the correct output.

**Step 7.** To display the list of all currently known devices (OVS switches), type the following command.

```
devices
```

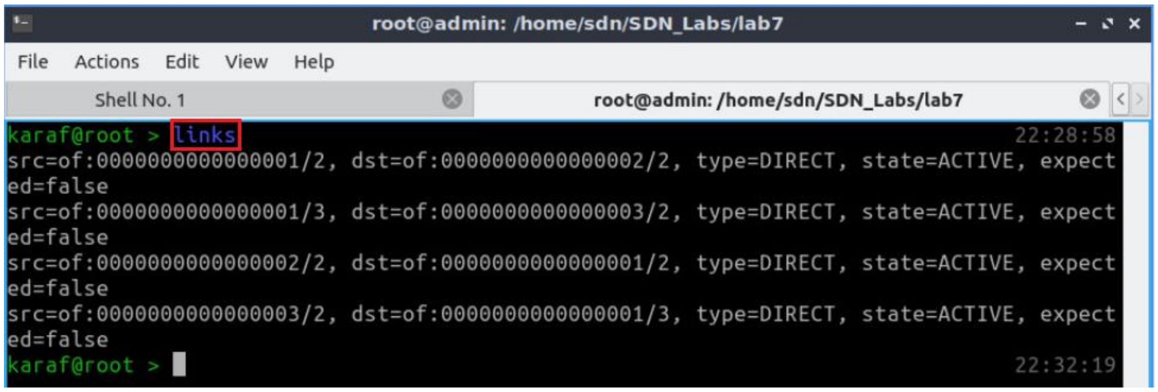

root@admin: /home/sdn/SDN_Labs/lab7
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

```
karaf@root > devices                                                   22:28:39
id=of:0000000000000001, available=true, local-status=connected 13s ago, role=MASTER, ty
pe=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=1, driver
=ovs, channelId=172.17.0.1:35630, managementAddress=172.17.0.1, protocol=OF_10
id=of:0000000000000002, available=true, local-status=connected 13s ago, role=MASTER, ty
pe=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=2, driver
=ovs, channelId=172.17.0.1:35634, managementAddress=172.17.0.1, protocol=OF_10
id=of:0000000000000003, available=true, local-status=connected 13s ago, role=MASTER, ty
pe=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=3, driver
=ovs, channelId=172.17.0.1:35632, managementAddress=172.17.0.1, protocol=OF_10
karaf@root >                                                           22:28:58
```

Figure 26. Displaying the currently known devices (switches).

**Step 8.** To display the list of all currently known links, type the following command.

```
links
```

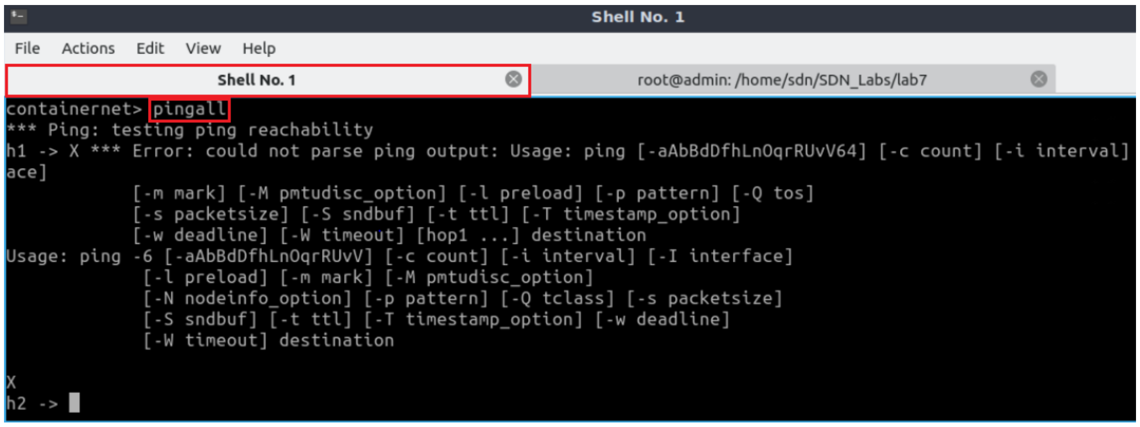

root@admin: /home/sdn/SDN_Labs/lab7
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

```
karaf@root > links                                                     22:28:58
src=of:0000000000000001/2, dst=of:0000000000000002/2, type=DIRECT, state=ACTIVE, expect
ed=false
src=of:0000000000000001/3, dst=of:0000000000000003/2, type=DIRECT, state=ACTIVE, expect
ed=false
src=of:0000000000000002/2, dst=of:0000000000000001/2, type=DIRECT, state=ACTIVE, expect
ed=false
src=of:0000000000000003/2, dst=of:0000000000000001/3, type=DIRECT, state=ACTIVE, expect
ed=false
karaf@root >                                                           22:32:19
```

Figure 27. Displaying the currently known links.

**Step 9**. Click on the Mininet tab on the left-hand side and type the command shown below to ping all hosts in the topology.

```
pingall
```


Shell No. 1
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

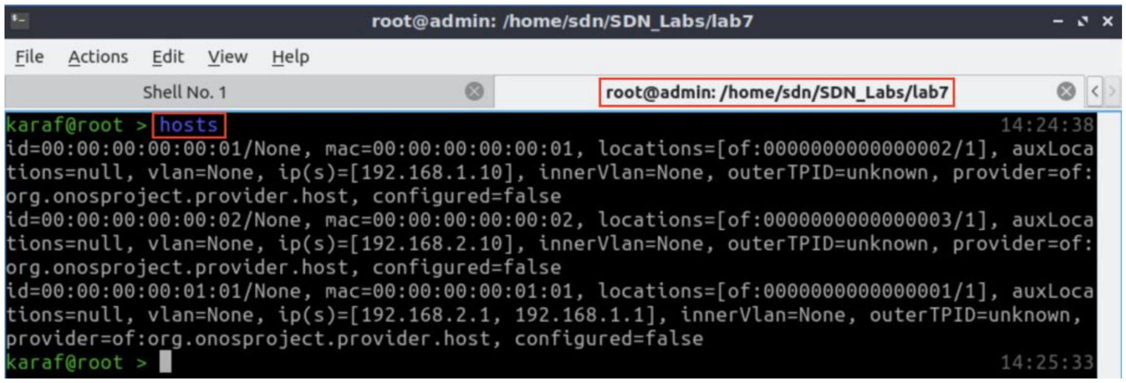
```
containernet> pingall
*** Ping: testing ping reachability
h1 -> X *** Error: could not parse ping output: Usage: ping [-aAbBdDfhLnOqrRUvV64] [-c count] [-i interval]
ace]
            [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
            [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
            [-w deadline] [-W timeout] [hop1 ...] destination
Usage: ping -6 [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
             [-l preload] [-m mark] [-M pmtudisc_option]
             [-N nodeinfo_option] [-p pattern] [-Q tclass] [-s packetsize]
             [-S sndbuf] [-t ttl] [-T timestamp_option] [-w deadline]
             [-W timeout] destination

X
h2 ->
```

Figure 28. Pinging all the hosts.

In the figure above, the hosts and the router will ping each other. The packets will reach the OpenFlow switches. The OpenFlow switches will forward them to the controller as they do not have the necessary rules installed to route the packets. Once the controller receives the packets, it will store the information related to these devices.

**Step 10.** Switch back to the ONOS CLI, the right tab. To display the list of all currently known hosts, type the following command.

```
hosts
```


root@admin: /home/sdn/SDN_Labs/lab7
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

```
karaf@root > hosts                                                             14:24:38
id=00:00:00:00:00:01/None, mac=00:00:00:00:00:01, locations=[of:0000000000000002/1], auxLoca
tions=null, vlan=None, ip(s)=[192.168.1.10], innerVlan=None, outerTPID=unknown, provider=of:
org.onosproject.provider.host, configured=false
id=00:00:00:00:00:02/None, mac=00:00:00:00:00:02, locations=[of:0000000000000003/1], auxLoca
tions=null, vlan=None, ip(s)=[192.168.2.10], innerVlan=None, outerTPID=unknown, provider=of:
org.onosproject.provider.host, configured=false
id=00:00:00:00:01:01/None, mac=00:00:00:00:01:01, locations=[of:0000000000000001/1], auxLoca
tions=null, vlan=None, ip(s)=[192.168.2.1, 192.168.1.1], innerVlan=None, outerTPID=unknown,
provider=of:org.onosproject.provider.host, configured=false
karaf@root >                                                                   14:25:33
```

Figure 29. Displaying the currently known hosts.

Consider Figure 29. ONOS recognizes host h1 (192.168.1.10), host h2 (192.168.2.10), and the interface of router r1 (192.168.1.1 and 192.168.2.1). Furthermore, ONOS displays the interfaces of the OpenFlow switches connected to the hosts. Note that you might have to wait until ONOS discovers the devices in case they do not appear immediately.

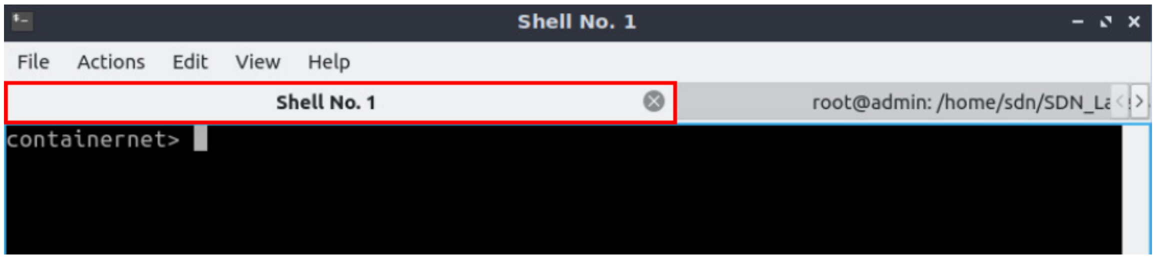
## 4 Integrating SDN and BGP

In the previous sections, you configured the legacy devices, as well as started ONOS and its OpenFlow application to discover the topology. In this section, you will first execute a script that connects the IBGP speaker (router r1) with the ONOS controller, so that the two entities can communicate. Furthermore, you will activate the ONOS SDN-IP application in order for the controller to access BGP information. This section is a prerequisite for the reactive-routing application to be activated and functional.

### 4.1 Connecting the IBGP speaker (router r1) with the ONOS controller

In this section, you will execute a script that creates a peer-to-peer link connecting router r1 with ONOS.

**Step 1.** Go to Mininet tab in the Linux terminal.

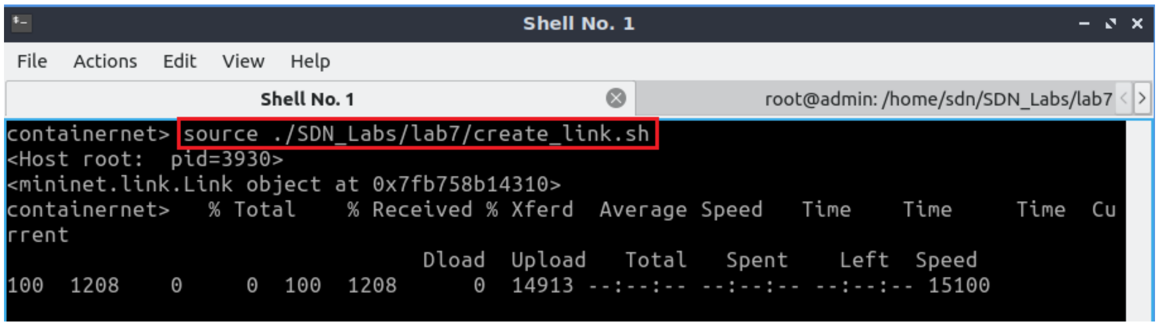


```
containernet>
```

Figure 30. Opening Mininet tab.

**Step 2.** In order to create a point-to-point network between the IBGP speaker (router r1) and ONOS, a script was written to facilitate the process. In order to execute the script, type the following command.

```
source ./SDN_Labs/lab7/create_link.sh
```


Shell No. 1
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

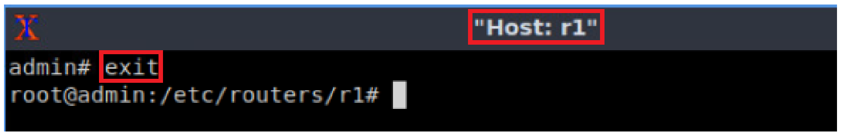
```
containernet> source ./SDN_Labs/lab7/create_link.sh
<Host root:  pid=3930>
<mininet.link.Link object at 0x7fb758b14310>
containernet>   % Total    % Received % Xferd  Average Speed   Time    Time     Time  Cu
rrent
                                 Dload  Upload   Total   Spent    Left  Speed
100  1208    0     0  100  1208      0  14913 --:--:-- --:--:-- --:--:-- 15100
```

Figure 31. Creating a point-to-point network (link) between the IBGP speaker and the ONOS controller.

Consider Figure 31. The script creates a point-to-point network between router r1 and ONOS. The network address of the point-to-point network is 10.0.0.0/24. Router r1 is assigned the IP address 10.0.0.1/24, whereas the ONOS controller is assigned 10.0.0.3/24. Furthermore, the script pushes a configuration file *(network-cfg.json)* to the controller necessary to run ONOS applications, such as SDN-IP and reactive-routing.

**Step 3.** In router r1 terminal, type the following command to exit the vtysh session.

```
exit
```

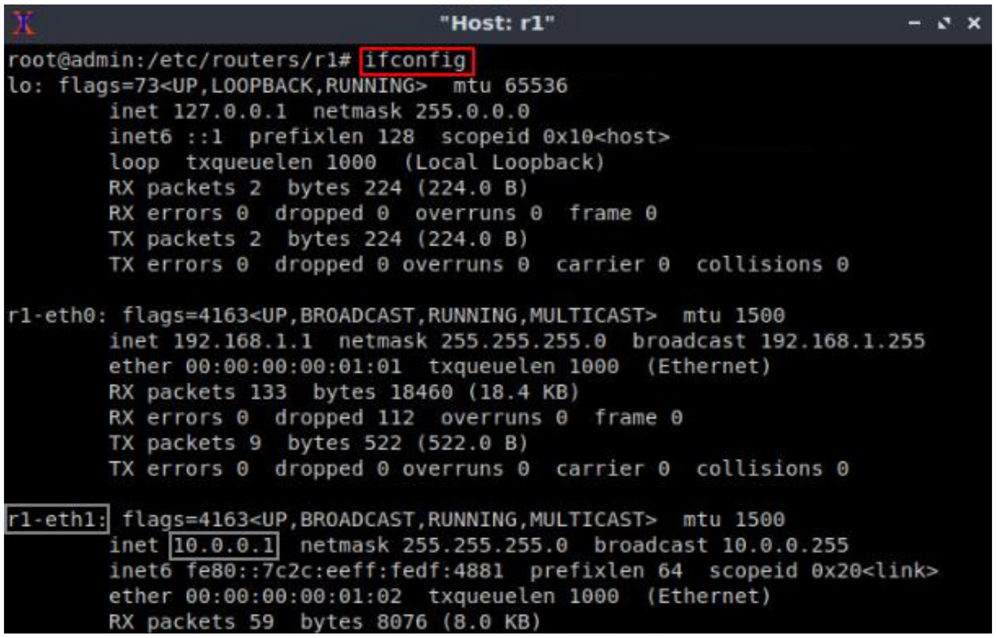


```
admin# exit
root@admin:/etc/routers/r1#
```

Figure 32. Exiting the vtysh session.

**Step 4.** Now that router r1 is connected to the ONOS controller, a new interface must appear. In order to verify the connected interface, type the following command.

```
ifconfig
```


```
"Host: r1"
root@admin:/etc/routers/r1# ifconfig
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 2  bytes 224 (224.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 224 (224.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.1  netmask 255.255.255.0  broadcast 192.168.1.255
        ether 00:00:00:00:01:01  txqueuelen 1000  (Ethernet)
        RX packets 133  bytes 18460 (18.4 KB)
        RX errors 0  dropped 112  overruns 0  frame 0
        TX packets 9  bytes 522 (522.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::7c2c:eeff:fedf:4881  prefixlen 64  scopeid 0x20<link>
        ether 00:00:00:00:01:02  txqueuelen 1000  (Ethernet)
        RX packets 59  bytes 8076 (8.0 KB)
```
Figure 33. Listing the interfaces of router r1.

Consider Figure 33. Interface *r1-eth1* is added after creating a point-to-point network between router r1 and the ONOS controller. Furthermore, the interface has the IP address 10.0.0.1.

**Step 5.** Navigate back to the Client Desktop and double click on the Computer icon.

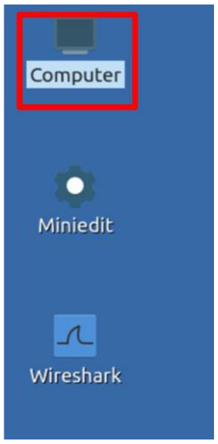


Figure 34. Opening the Computer icon.

**Step 6.** Navigate to the directory */home/sdn/SDN_Labs/lab7* and open the file *network-cfg.json*.

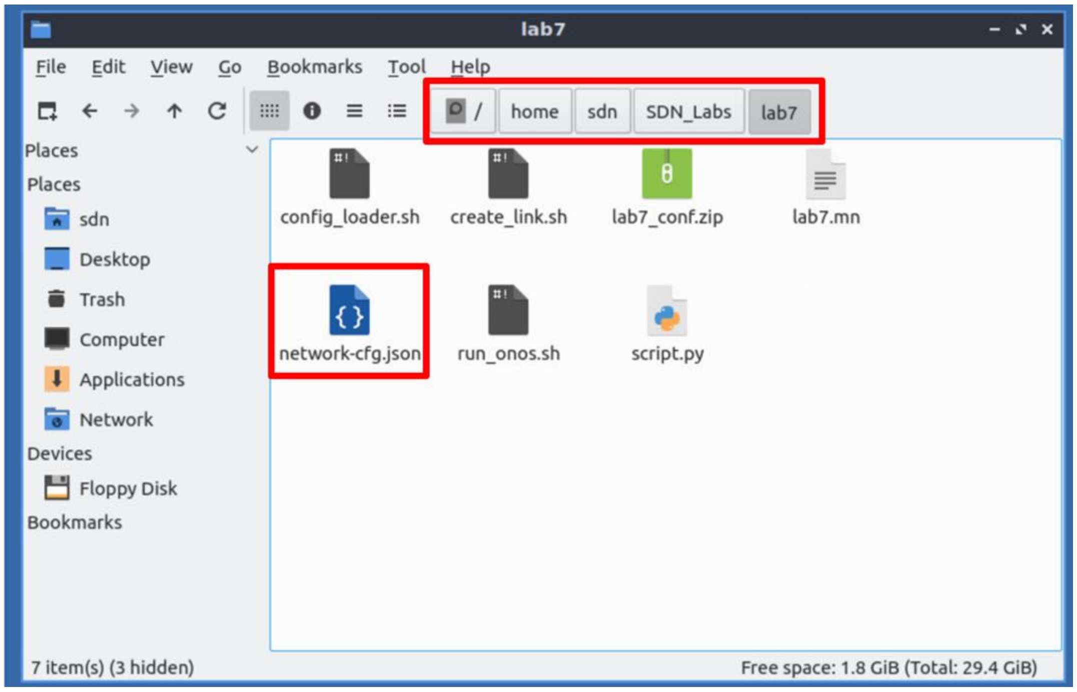
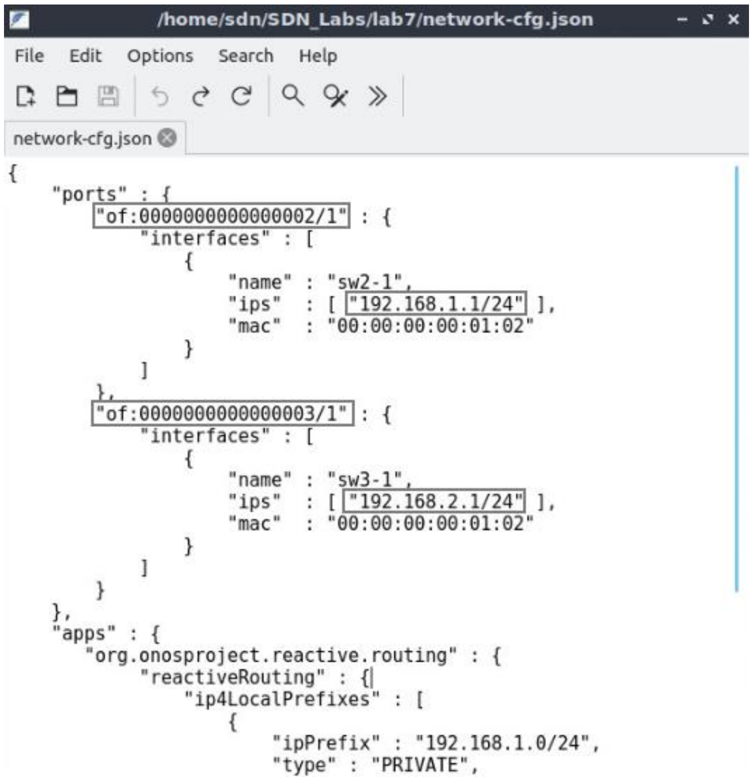
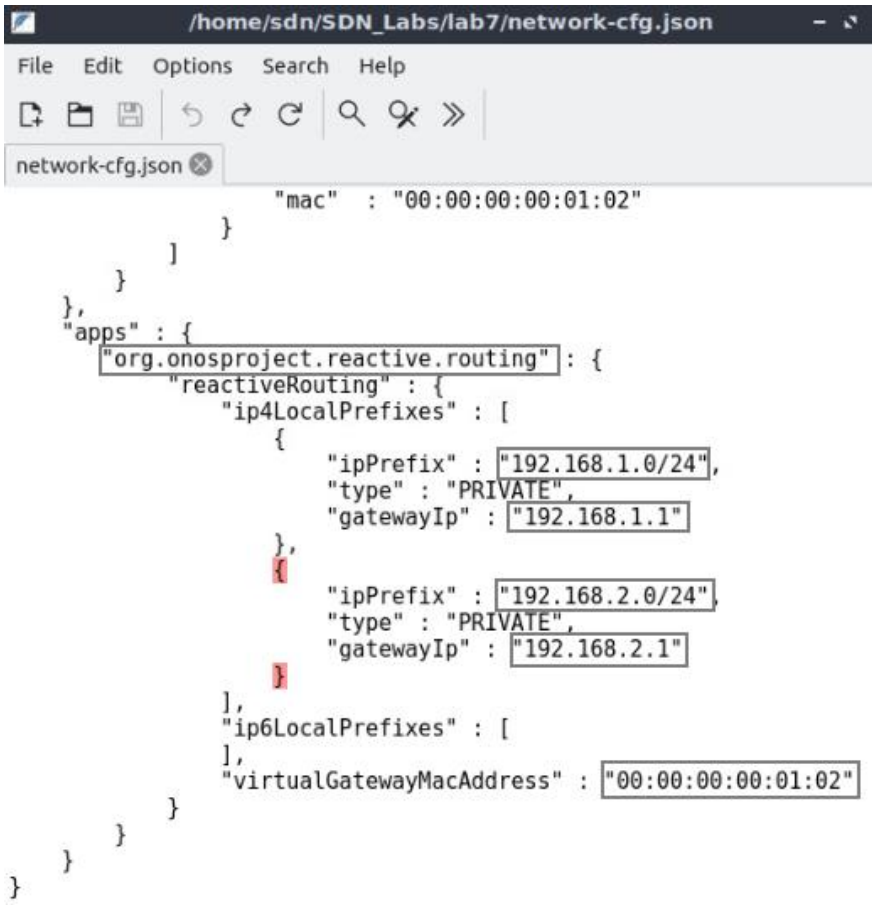


Figure 35. Opening the network configuration file.


/home/sdn/SDN_Labs/lab7/network-cfg.json
File Edit Options Search Help
network-cfg.json

```
{
    "ports" : {
        "of:0000000000000002/1" : {
            "interfaces" : [
                {
                    "name" : "sw2-1",
                    "ips"  : [ "192.168.1.1/24" ],
                    "mac"  : "00:00:00:00:01:02"
                }
            ]
        },
        "of:0000000000000003/1" : {
            "interfaces" : [
                {
                    "name" : "sw3-1",
                    "ips"  : [ "192.168.2.1/24" ],
                    "mac"  : "00:00:00:00:01:02"
                }
            ]
        }
    },
    "apps" : {
       "org.onosproject.reactive.routing" : {
            "reactiveRouting" : {
                "ip4LocalPrefixes" : [
                    {
                        "ipPrefix" : "192.168.1.0/24",
                        "type" : "PRIVATE",
```

Figure 36. Opening network-cfg.json file.


/home/sdn/SDN_Labs/lab7/network-cfg.json
File Edit Options Search Help
network-cfg.json

```
                    "mac"  : "00:00:00:00:01:02"
                }
            ]
        }
    },
    "apps" : {
        "org.onosproject.reactive.routing" : {
            "reactiveRouting" : {
                "ip4LocalPrefixes" : [
                    {
                        "ipPrefix" : "192.168.1.0/24",
                        "type" : "PRIVATE",
                        "gatewayIp" : "192.168.1.1"
                    },
                    {
                        "ipPrefix" : "192.168.2.0/24",
                        "type" : "PRIVATE",
                        "gatewayIp" : "192.168.2.1"
                    }
                ],
                "ip6LocalPrefixes" : [
                ],
                "virtualGatewayMacAddress" : "00:00:00:00:01:02"
            }
        }
    }
}
```

Figure 37. Opening network-cfg.json file.

Consider Figures 36 and 37. The configuration file *network-cfg.jso*n is pushed into the ONOS controller so that the applications work properly. The configuration file consists of two json objects, namely ports and apps. Figure 36 specifies the interfaces of the OpenFlow switches connected to the hosts, along with the associated IP address of these switches. Figure 37 includes the name of the ONOS application (*reactive.routing*) with some attributes needed by this application. For instance, the reactive-routing application requires the IP address of the SDN networks (192.168.1.0/24 and 192.168.2.0/24) and their default gateways. The MAC address *00:00:00:00:01:02* is the virtual gateway address for the ONOS controller.

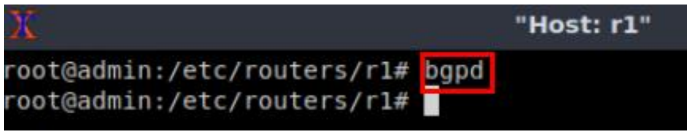
Note that the file *network-cfg.json* is customized based on the running ONOS applications. This file is generated automatically by a script to facilitate the process of configuring the lab.

## 4.2 Configuring BGP on router r1

In this section, you will configure BGP on router r1 to peer with the ONOS controller.

**Step 1.** Type the following command on router r1 terminal to start BGP routing protocol.

```
bgpd
```

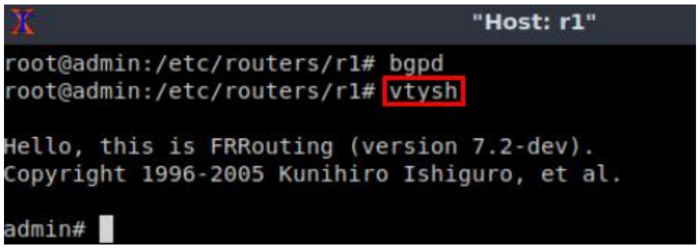


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1#
```

Figure 38. Starting BGP daemon.

**Step 2.** In order to enter to router r1 terminal, type the following command.

```
vtysh
```

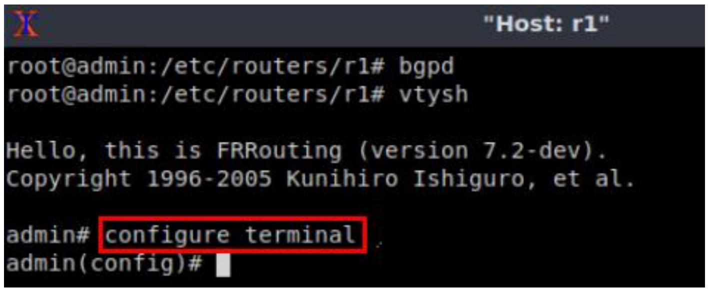


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin#
```

Figure 39. Starting vtysh on router r1.

**Step 3.** To enable router r1 global configuration mode, issue the following command.

```
configure terminal
```

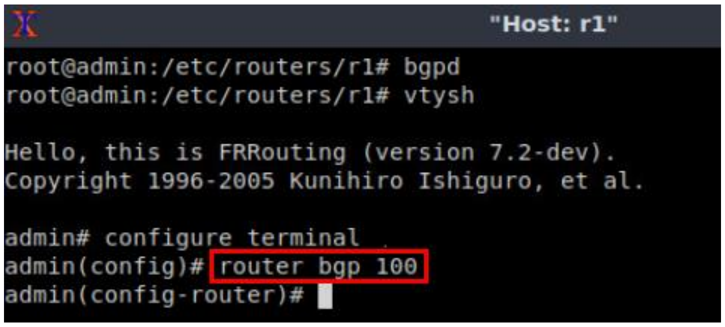


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

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

Figure 40. Enabling configuration mode on router r1.

**Step 4.** The ASN assigned for router r1 is 100. In order to configure BGP, type the following command.

```
router bgp 100
```

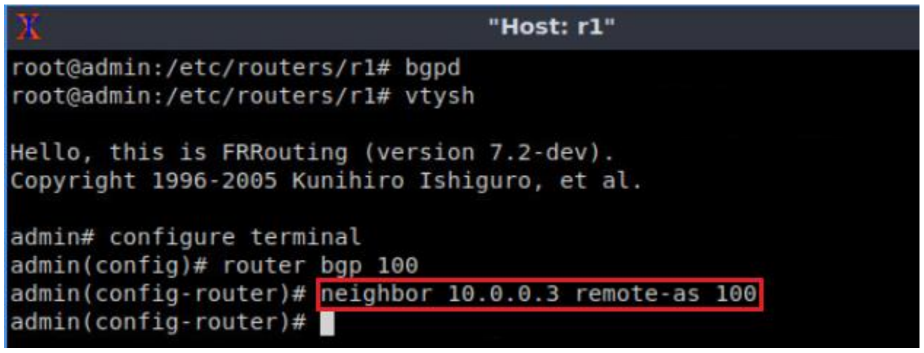


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)#
```

Figure 41. Configuring BGP on router r1.

**Step 5.** Router r1 and the ONOS controller are connected using a point-to-point network (10.0.0.0/24). The IP address assigned to the controller is 10.0.0.3. As router r1 is the IBGP speaker within the SDN network, it must establish a BGP peering relationship with the controller in its network (AS 100). In order to establish BGP peering relationship with the controller, type the following command.

```
neighbor 10.0.0.3 remote-as 100
```

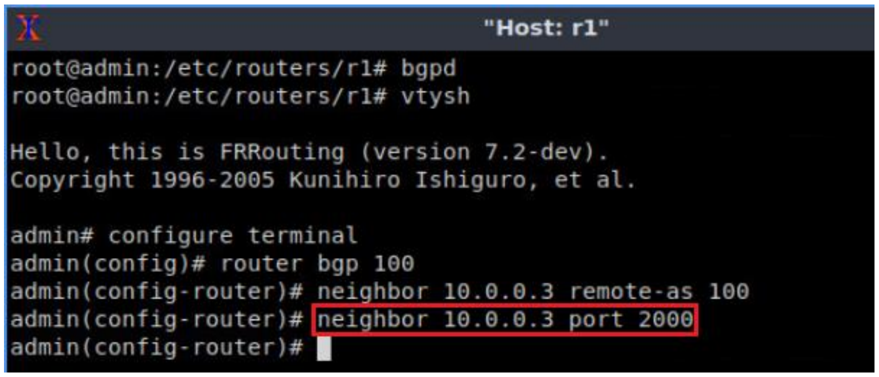


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# neighbor 10.0.0.3 remote-as 100
admin(config-router)#
```

Figure 42. Assigning BGP neighbor to router r1.

**Step 6.** By default, ONOS listens to TCP port number 2000 for incoming BGP connections, which is not the default BGP port number 179. In order to specify the port for incoming BGP messages from ONOS, write the following command.

```
neighbor 10.0.0.3 port 2000
```

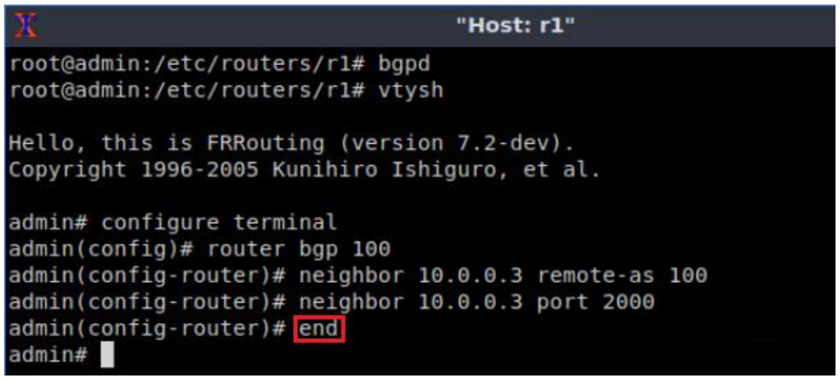


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# neighbor 10.0.0.3 remote-as 100
admin(config-router)# neighbor 10.0.0.3 port 2000
admin(config-router)#
```

Figure 43. Changing the Listening port for BGP connections.

**Step 7.** Type the following command to exit from the configuration mode.

```
end
```

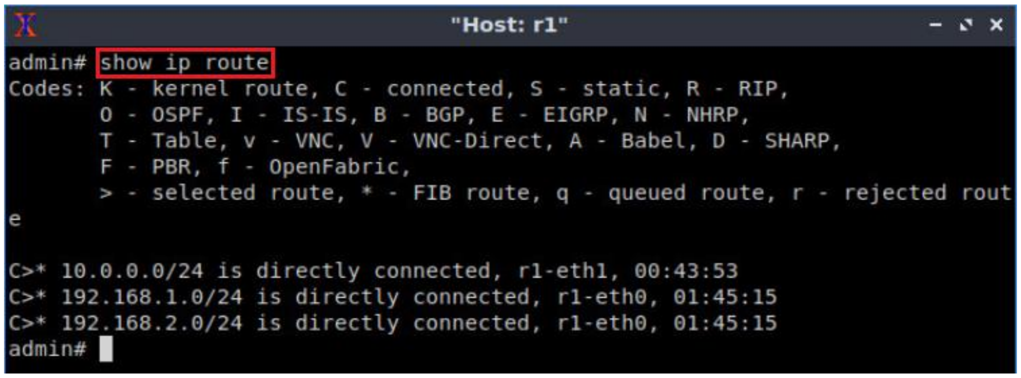


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# neighbor 10.0.0.3 remote-as 100
admin(config-router)# neighbor 10.0.0.3 port 2000
admin(config-router)# end
admin#
```

Figure 44. Exiting from configuration mode.

**Step 8.** Type the following command on router r1 terminal to verify the routing table of router r1. It will list all the directly connected networks. The routing table of router r1 contains a directly connected network, 10.0.0.0/24.

```
show ip route
```



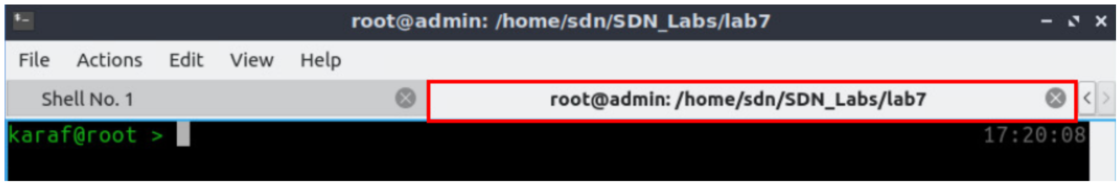
```
"Host: r1"
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected rout
e

C>* 10.0.0.0/24 is directly connected, r1-eth1, 00:43:53
C>* 192.168.1.0/24 is directly connected, r1-eth0, 01:45:15
C>* 192.168.2.0/24 is directly connected, r1-eth0, 01:45:15
admin#
```

Figure 45. Displaying the routing table of router r1.

## 4.3 Activating the SDN-IP application

In this section, you will activate the SDN-IP application and other dependencies (applications) that will interconnect the SDN network with BGP.

**Step 1.** Go to the ONOS terminal.



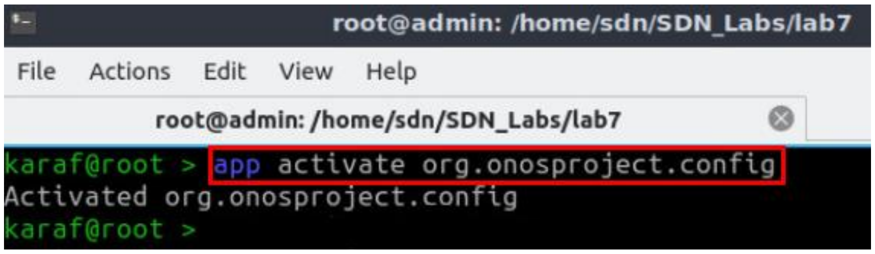
root@admin: /home/sdn/SDN_Labs/lab7
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

```
karaf@root >                                                   17:20:08
```

Figure 46. Opening ONOS terminal.

**Step 2.** Before activating the SDN-IP application you must start the config application. This is an application for the network configuration. In order to activate the config application, type the following command.

```
app activate org.onosproject.config
```



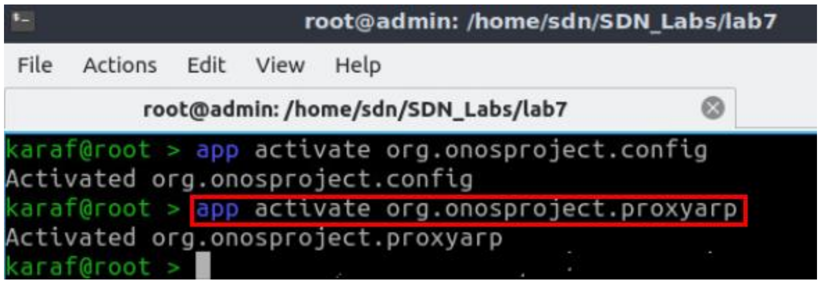
root@admin: /home/sdn/SDN_Labs/lab7
File Actions Edit View Help
root@admin: /home/sdn/SDN_Labs/lab7

```
karaf@root > app activate org.onosproject.config
Activated org.onosproject.config
karaf@root >
```

Figure 47. Activating ONOS config application.

**Step 3.** The SDN-IP application has an additional application dependency, which is used to resolve ARP requests. This is the proxyarp application that responds to ARP requests on behalf of hosts and external routers.

```
app activate org.onosproject.proxyarp
```

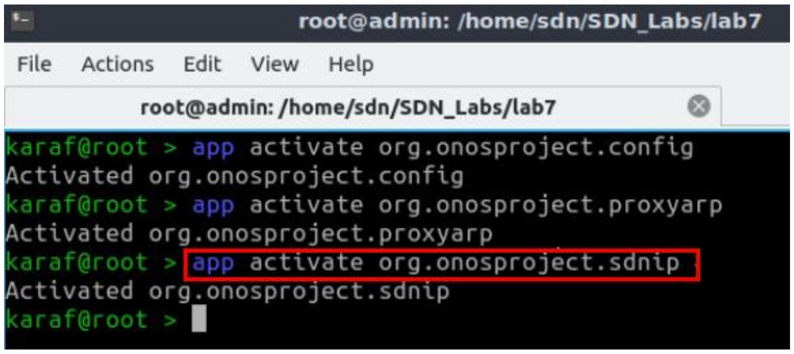

root@admin: /home/sdn/SDN_Labs/lab7
File Actions Edit View Help
root@admin: /home/sdn/SDN_Labs/lab7

```
karaf@root > app activate org.onosproject.config
Activated org.onosproject.config
karaf@root > app activate org.onosproject.proxyarp
Activated org.onosproject.proxyarp
karaf@root >
```

Figure 48. Activating the ONOS proxyarp application.

**Step 4.** Once the dependencies are started, the SDN-IP application can be activated. In order to do that, type the following command.

```
app activate org.onosproject.sdnip
```


root@admin: /home/sdn/SDN_Labs/lab7
File Actions Edit View Help
root@admin: /home/sdn/SDN_Labs/lab7

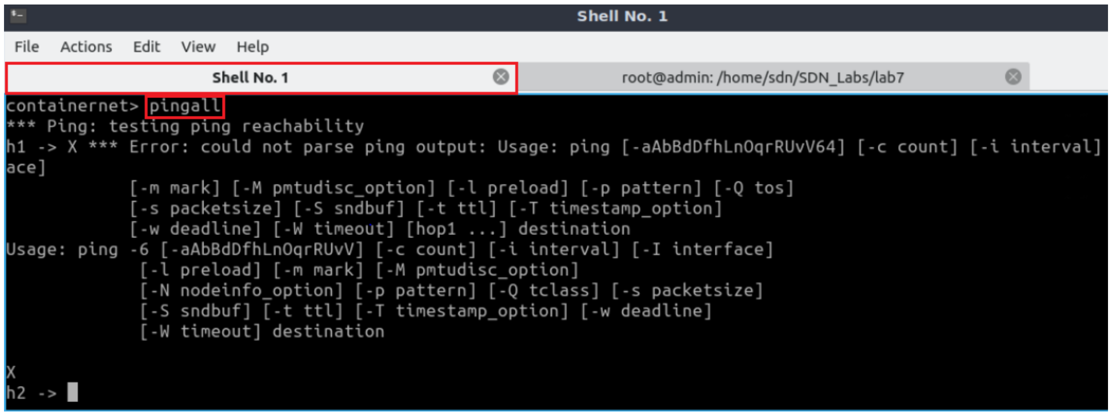
```
karaf@root > app activate org.onosproject.config
Activated org.onosproject.config
karaf@root > app activate org.onosproject.proxyarp
Activated org.onosproject.proxyarp
karaf@root > app activate org.onosproject.sdnip
Activated org.onosproject.sdnip
karaf@root >
```

Figure 49. Activating ONOS SDN-IP application.

After activating the applications above, you might have to wait few minutes until the applications discover the topology and exchange information in order to get correct results.

**Step 5**. Click on the Mininet tab on the left-hand side and type the command shown below to ping all hosts in the topology.

```
pingall
```



Shell No. 1
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

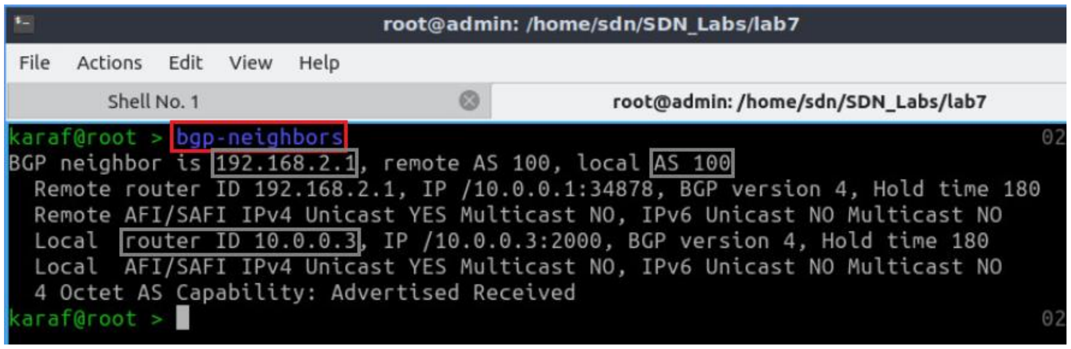
```
containernet> pingall
*** Ping: testing ping reachability
h1 -> X *** Error: could not parse ping output: Usage: ping [-aAbBdDfhLnOqrRUvV64] [-c count] [-i interval]
ace]
            [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
            [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
            [-w deadline] [-W timeout] [hop1 ...] destination
Usage: ping -6 [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
             [-l preload] [-m mark] [-M pmtudisc_option]
             [-N nodeinfo_option] [-p pattern] [-Q tclass] [-s packetsize]
             [-S sndbuf] [-t ttl] [-T timestamp_option] [-w deadline]
             [-W timeout] destination

X
h2 ->
```

Figure 50. Pinging all the hosts.

The test result will be unsuccessful. The purpose of this step is to provide information about the hosts to the controller.

**Step 6.** On the ONOS terminal, type the following command to show the IBGP neighbors that have connected to SDN-IP application.

```
bgp-neighbors
```



root@admin: /home/sdn/SDN_Labs/lab7
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

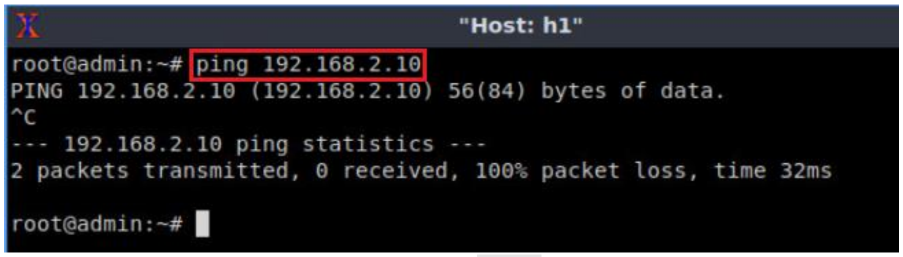
```
karaf@root > bgp-neighbors
BGP neighbor is 192.168.2.1, remote AS 100, local AS 100
  Remote router ID 192.168.2.1, IP /10.0.0.1:34878, BGP version 4, Hold time 180
  Remote AFI/SAFI IPv4 Unicast YES Multicast NO, IPv6 Unicast NO Multicast NO
  Local  router ID 10.0.0.3, IP /10.0.0.3:2000, BGP version 4, Hold time 180
  Local  AFI/SAFI IPv4 Unicast YES Multicast NO, IPv6 Unicast NO Multicast NO
  4 Octet AS Capability: Advertised Received
karaf@root >
```

Figure 51. Viewing IBGP neighbors within the SDN network.

Consider Figure 51. The neighbor 192.168.2.1 corresponds to router r1 in AS 100. This is the internal BGP speaker in the SDN network. The local router ID that the SDN-IP application uses is 10.0.0.3.

**Step 7.** Test the connectivity between host h1 and host h2 using the `ping` command. On host h1, type the command specified below.

```
ping 192.168.2.10
```



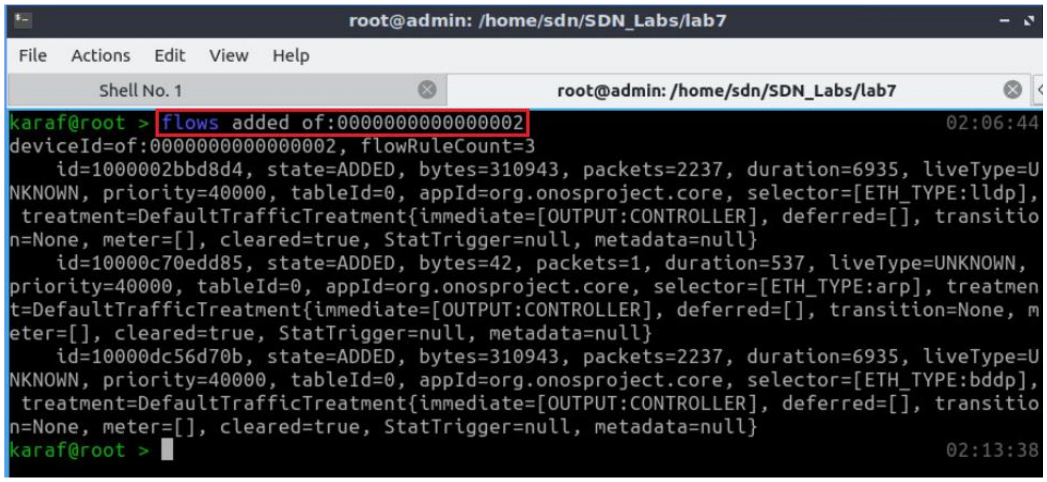
```
root@admin:~# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
^C
--- 192.168.2.10 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 32ms

root@admin:~#
```

Figure 52. Output of `ping` command.

To stop the test, press `Ctrl+c`. In the figure above, the result of the ping test shows an unsuccessful connectivity test as the reactive-routing application is not activated yet.

**Step 8.** On the ONOS terminal, type the following command to verify the flows in switch s2. Use the `tab` key to autocomplete the OpenFlow port.

```
flows added of:0000000000000002
```



root@admin: /home/sdn/SDN_Labs/lab7

```
karaf@root > flows added of:0000000000000002                                  02:06:44
deviceId=of:0000000000000002, flowRuleCount=3
    id=1000002bbd8d4, state=ADDED, bytes=310943, packets=2237, duration=6935, liveType=U
NKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:lldp],
 treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transitio
n=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
    id=10000c70edd85, state=ADDED, bytes=42, packets=1, duration=537, liveType=UNKNOWN,
priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatmen
t=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, m
eter=[], cleared=true, StatTrigger=null, metadata=null}
    id=10000dc56d70b, state=ADDED, bytes=310943, packets=2237, duration=6935, liveType=U
NKNOWN, priority=40000, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:bddp],
 treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transitio
n=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
karaf@root >                                                                  02:13:38
```

Figure 53. Verifying flows on s2.

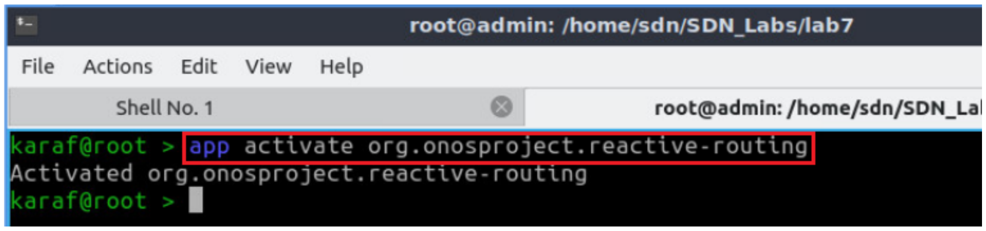
Consider Figure 53. The hosts are unreachable at this point since no flow deals with IPv4 traffic in the flow table of the switches.

## 5 Activating reactive-routing application and verifying the connectivity between the networks

In this section, you will activate the reactive-routing application in order to establish connectivity between the two networks 192.168.1.0/24 and 192.168.2.0/24. Additionally, you will inspect the flow table of the switches, and verify the connectivity between the two hosts.

**Step 1.** In order to activate the reactive-routing application, type the following command.

```
app activate org.onosproject.reactive-routing
```



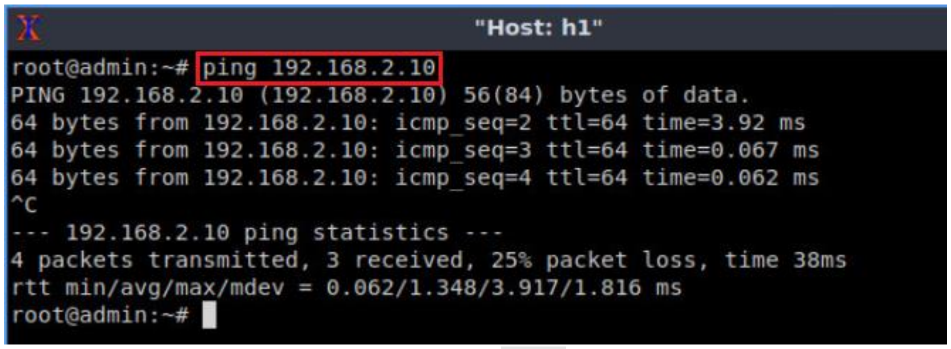
root@admin: /home/sdn/SDN_Labs/lab7

```
karaf@root > app activate org.onosproject.reactive-routing
Activated org.onosproject.reactive-routing
karaf@root >
```

Figure 54. Activating ONOS SDN-IP application.

**Step 2.** Test the connectivity between host h1 and host h2 using the `ping` command. On host h1, type the command specified below.

```
ping 192.168.2.10
```



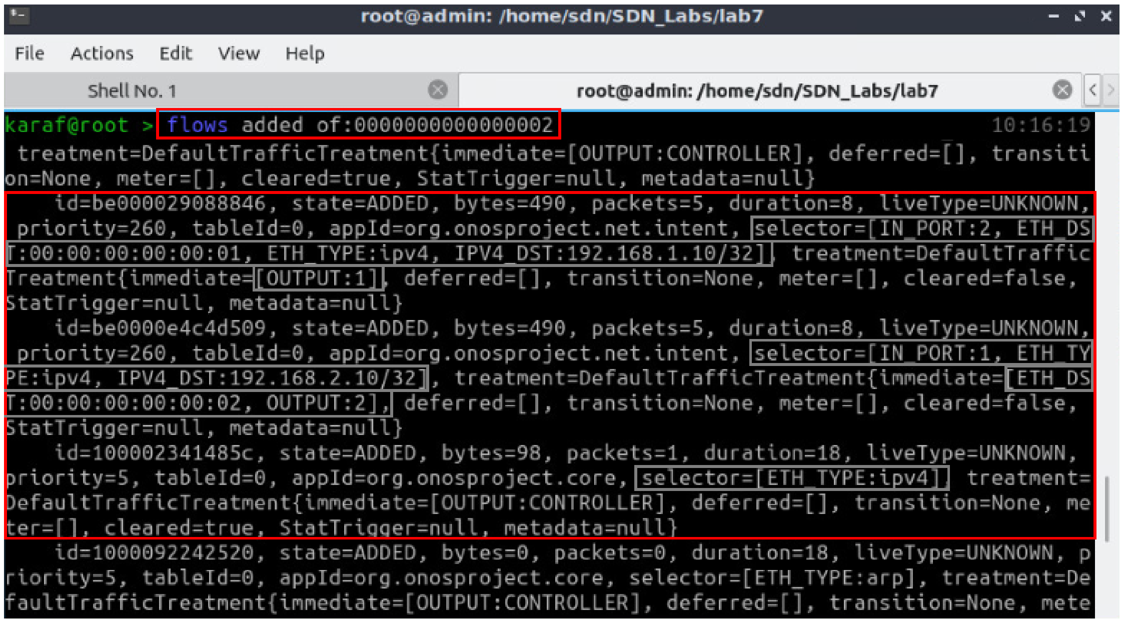
```
root@admin:~# ping 192.168.2.10
PING 192.168.2.10 (192.168.2.10) 56(84) bytes of data.
64 bytes from 192.168.2.10: icmp_seq=2 ttl=64 time=3.92 ms
64 bytes from 192.168.2.10: icmp_seq=3 ttl=64 time=0.067 ms
64 bytes from 192.168.2.10: icmp_seq=4 ttl=64 time=0.062 ms
^C
--- 192.168.2.10 ping statistics ---
4 packets transmitted, 3 received, 25% packet loss, time 38ms
rtt min/avg/max/mdev = 0.062/1.348/3.917/1.816 ms
root@admin:~#
```

Figure 55. Output of `ping` command.

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

**Step 3.** Type the following command to verify the flows on the switch s2.

```
flows added of:0000000000000002
```


root@admin: /home/sdn/SDN_Labs/lab7
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

```
karaf@root > flows added of:0000000000000002                              10:16:19
 treatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transiti
on=None, meter=[], cleared=true, StatTrigger=null, metadata=null}
    id=be000029088846, state=ADDED, bytes=490, packets=5, duration=8, liveType=UNKNOWN,
 priority=260, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:2, ETH_DS
T:00:00:00:00:00:01, ETH_TYPE:ipv4, IPV4_DST:192.168.1.10/32], treatment=DefaultTraffic
Treatment{immediate=[OUTPUT:1], deferred=[], transition=None, meter=[], cleared=false,
StatTrigger=null, metadata=null}
    id=be0000e4c4d509, state=ADDED, bytes=490, packets=5, duration=8, liveType=UNKNOWN,
 priority=260, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:1, ETH_TY
PE:ipv4, IPV4_DST:192.168.2.10/32], treatment=DefaultTrafficTreatment{immediate=[ETH_DS
T:00:00:00:00:00:02, OUTPUT:2], deferred=[], transition=None, meter=[], cleared=false,
StatTrigger=null, metadata=null}
    id=100002341485c, state=ADDED, bytes=98, packets=1, duration=18, liveType=UNKNOWN,
priority=5, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:ipv4] treatment=
DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, me
ter=[], cleared=true, StatTrigger=null, metadata=null}
    id=1000092242520, state=ADDED, bytes=0, packets=0, duration=18, liveType=UNKNOWN, p
riority=5, tableId=0, appId=org.onosproject.core, selector=[ETH_TYPE:arp], treatment=De
faultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=None, mete
```

Figure 56. Verifying flows on s2.

Consider Figure 56. You will notice *ipv4* flow is added in the flow table. Additionally, you can verify the reactively installed routing path to reach a specific destination. For instance, incoming packets on port 2 (`IN_PORT:2`) having target MAC address 00:00:00:00:00:01 (`ETH_DST`) and IPv4 destination 192.168.1.10/32 (`IPV4_DST`) will be forwarded out of port 1 (`OUTPUT:1`). Similarly, incoming IPv4 packets on port 1 (`IN_PORT:1`) having IPv4 destination 192.168.2.10/32 will undergo two actions. First, their MAC destination address will be modified to 00:00:00:00:00:02, then they will be forwarded out of port 2.

Since the routing paths are installed reactively, each traffic appears in the flow table only for a certain period (approx. 60 sec)

This concludes Lab 7. Stop the emulation and then exit out of MiniEdit and Linux terminal.

## References

1. A. Tanenbaum, D. Wetherall, "*Computer networks*", 5th Edition, Pearson, 2012.
2. P. Goransson, C. Black, T. Culver. "*Software defined networks: a comprehensive approach*". Morgan Kaufmann, 2016.
3. P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.C. Wang, J. Bi. "*Seamless interworking of SDN and IP*". In Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, pp. 475-476, 2013.
4. ONOS project, "*SDN-IP*", [Online]. Available: https://wiki.onosproject.org/display/ONOS/SDN-IP.

# SOFTWARE DEFINED NETWORKING

# Lab 8: Interconnection between legacy networks and SDN networks

**Document Version: 07-08-2021**

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

## Contents

## Overview

This lab is an introduction to integrating Software Defined Networking (SDN) networks with legacy networks. The focus of the lab is to understand how to use the SDN-IP application to peer with a legacy network. The SDN-IP application allows the SDN network to communicate with legacy networks using the Border Gateway Protocol (BGP).

## Objectives

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

1. Understand how SDN networks exchange routing information with legacy networks.
2. Configure BGP on legacy routers.
3. Integrate SDN and legacy networks through the SDN-IP application.
4. Verify the connectivity between the SDN and legacy networks.
5. Inspect the flow table of the switches to understand the SDN-IP application.

## Lab settings

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

Table 1**.** Credentials to access the 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 BGP within legacy networks.
4. Section 4: Starting the ONOS controller.
5. Section 5: Integrating SDN and legacy networks.
6. Section 6: Verifying the connectivity between the networks.
7. Section 7: Inspecting the flow table of the OpenFlow switches.

## 1 Introduction

Today's Internet is utterly dependent on routing protocols, such as BGP, so without a clean mechanism to integrate an OpenFlow/SDN and legacy/IP networks, the use of OpenFlow will remain restricted to isolated data center deployments. This lab shows how to integrate SDN and legacy networks and how the SDN controller translates the BGP information into OpenFlow entries[1].

## 1.1 Exchanging routing information within legacy networks

The Internet is a collection of networks or Autonomous Systems (ASes) that are interconnected. An AS refers to a group of connected networks under a single administrative entity or domain. Traditional networks depend on routing protocols to interconnect and share routing information. Such protocols are also referred to as control protocols, and each device runs them in the network[1].

BGP is the standard exterior gateway protocol designed to exchange routing and reachability information among ASes on the Internet. BGP is relevant to network administrators of large organizations that connect to one or more Internet Service Providers (ISPs) and ISPs who connect to other network providers[1].

Two routers that establish a BGP connection are referred to as BGP peers or neighbors. BGP sessions run over Transmission Control Protocol (TCP). Suppose a BGP session is established between two neighbors in different ASes. In that case, the session is referred to as an External BGP (EBGP) session. If the session is established between two neighbors in the same AS, the session is referred to as Internal BGP (IBGP) session[1].

Figure 1 shows two legacy networks, each in an AS. Each router runs its internal local algorithm (routing protocol) to communicate with other peers. The routing protocol used between ASes is BGP.

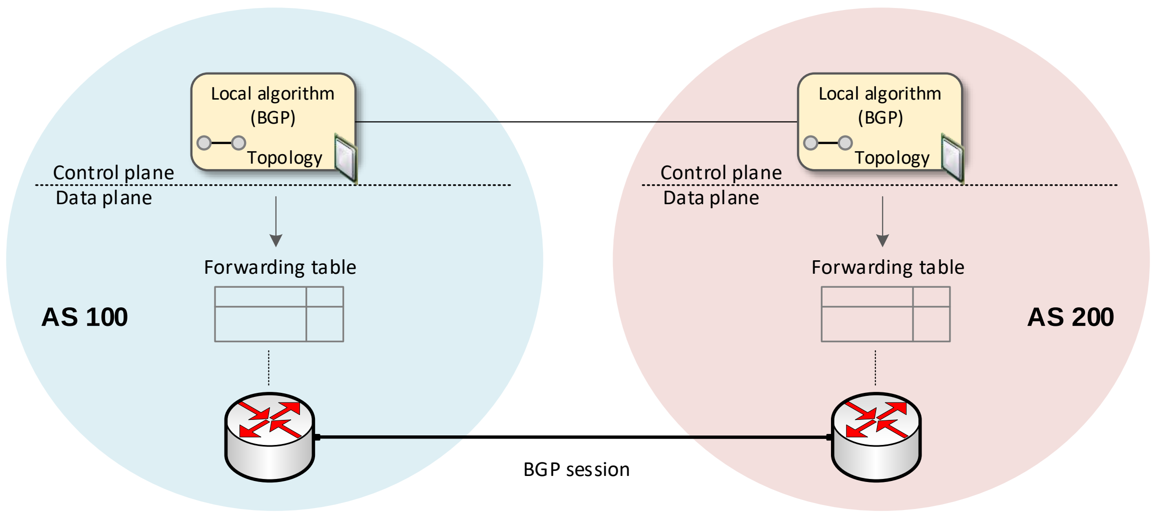


Figure 1. Legacy networks use BGP to share routing information between ASes.

## 1.2 Integrating SDN with legacy networks via SDN-IP application

The Border Gateway Protocol version 4 (BGPv4)[2] is the most widely adopted technique used to peer ASes on the Internet. Therefore, to integrate legacy networks with SDN-driven ASes, a peering mechanism that facilitates the BGP protocol is required. This integration is performed via the SDN-IP application that runs on top of the SDN controller.

Consider Figure 2. Typically, an SDN network is composed of various OpenFlow switches (switches s1 and s2) connected to the controller (c0). The external network (AS 100) can connect to the SDN network (AS 200) by establishing an EBGP session between the BGP router (router r1) and the OpenFlow switch (switch s1). The BGP speaker (router r2), referred to as an IBGP router, must exist within the SDN network and be connected to the data plane to communicate with external IP networks. The SDN-IP application is connected to the IBGP speaker within the SDN network via an IBGP session[3].

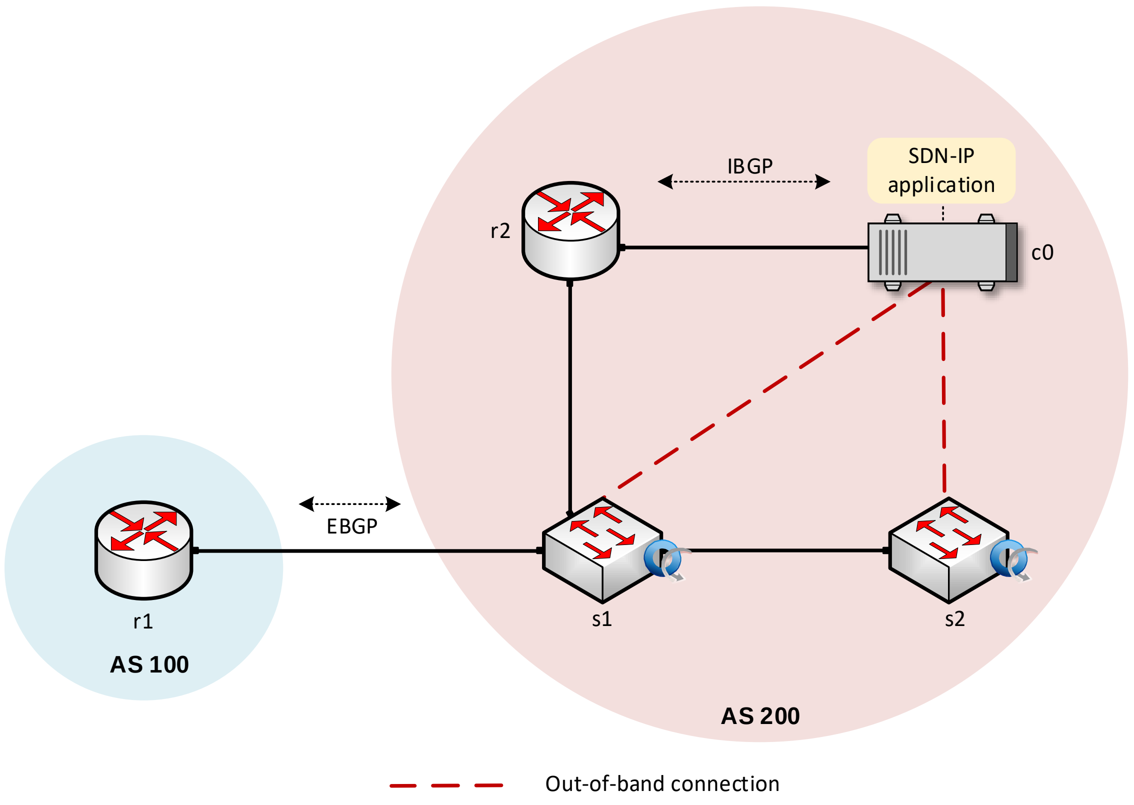


Figure 2. Integrating SDN and IP networks.

## 2 Lab topology

Consider Figure 3. The topology consists of two IP networks (AS 200 and AS 300) and one SDN network (AS 100). The IP networks connect to the SDN network through their BGP routers. Router r1 is a BGP router within the SDN network. It communicates with EBGP routers r2 and r3 via the networks 192.168.12.0/30 and 192.168.13.0/30, respectively. Furthermore, router r1 is connected to the controller in order to propagate the BGP advertisements to the SDN-IP application running on top of the ONOS controller. Router r1 and the controller are connected via the network 10.0.0.0/24.

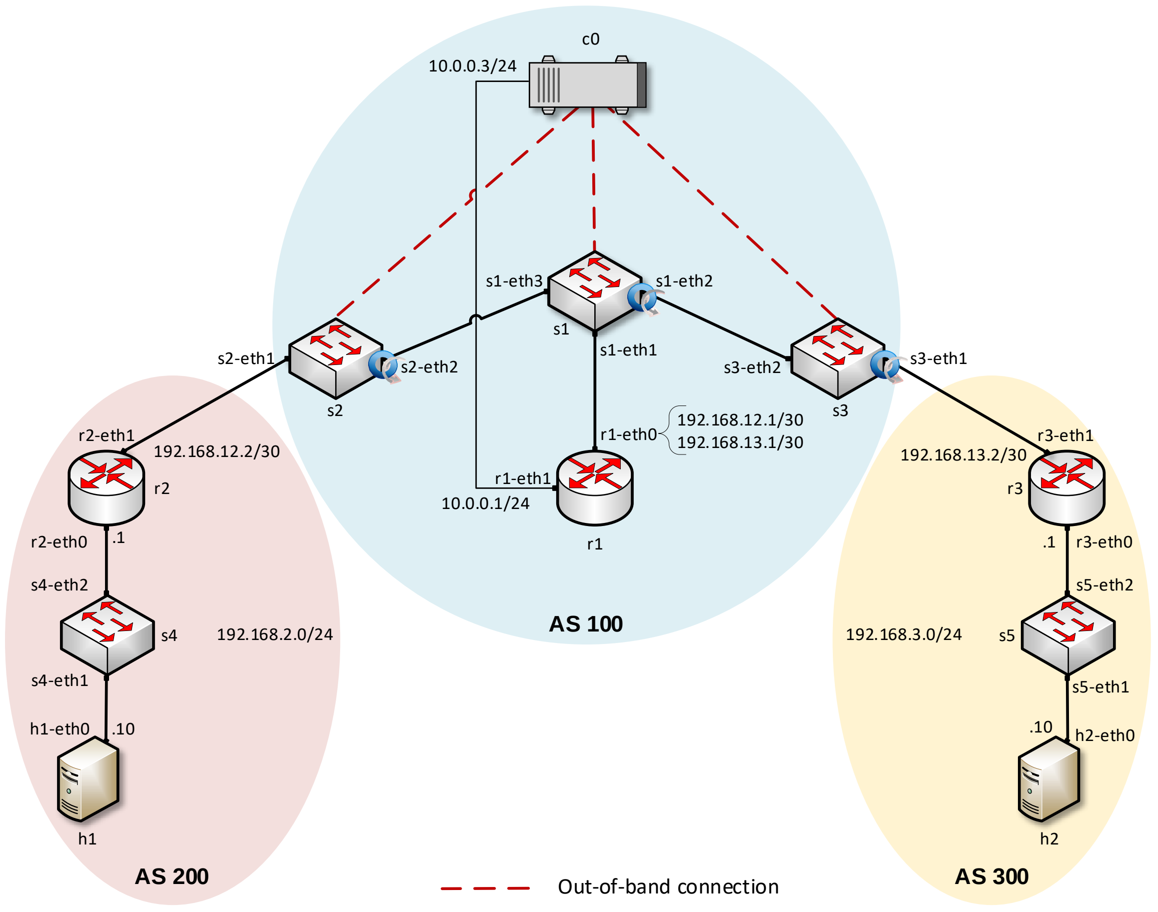


Figure 3. Lab topology.

## 2.1 Lab settings

The devices are already configured according to Table 2.

Table 2. Topology information.

| Device | Interface | MAC Address | IP Address | Subnet | Default gateway |
|---|---|---|---|---|---|
| Router r1 | r1-eth0 | 00:00:00:00:01:01 | 192.168.12.1 | /30 | N/A |
| | | 00:00:00:00:01:01 | 192.168.13.1 | /30 | N/A |
| | r1-eth1 | 00:00:00:00:01:02 | 10.0.0.1 | /24 | N/A |
| Router r2 | r2-eth0 | 00:00:00:00:02:01 | 192.168.2.1 | /24 | N/A |
| | r2-eth1 | 00:00:00:00:02:02 | 192.168.12.2 | /30 | N/A |
| Router r3 | r3-eth0 | 00:00:00:00:03:01 | 192.168.3.1 | /24 | N/A |
| | r3-eth1 | 00:00:00:00:03:02 | 192.168.13.2 | /30 | N/A |
| Host h1 | h1-eth0 | 00:00:00:00:00:01 | 192.168.2.10 | /24 | 192.168.2.1 |

| | | | | | |
|---|---|---|---|---|---|
| Host h2 | h2-eth0 | 00:00:00:00:00:02 | 192.168.3.10 | /24 | 192.168.3.1 |
| c0 | N/A | N/A | 172.17.0.2 | /16 | N/A |
| | N/A | N/A | 10.0.0.3 | /24 | N/A |

## 2.2 Loading the 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 such as: configuring network elements (i.e IP addresses, default gateway), saving the topologies, and exporting layer 2 models.

**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`.

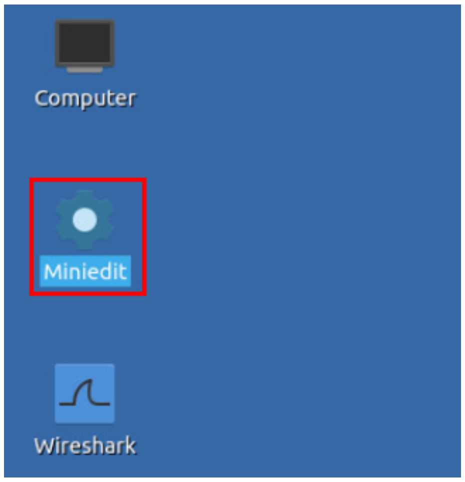


Figure 4. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then *open* to load the lab's topology. Open the *Lab8.mn* topology file stored in the default directory, */home/sdn/SDN_Labs /lab8* and click on *Open*.

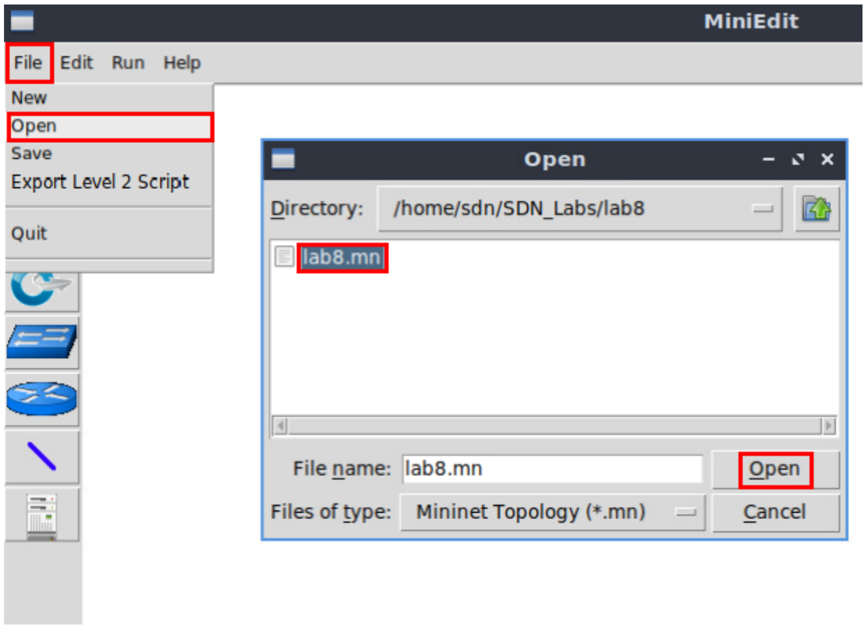


Figure 5. Opening topology.

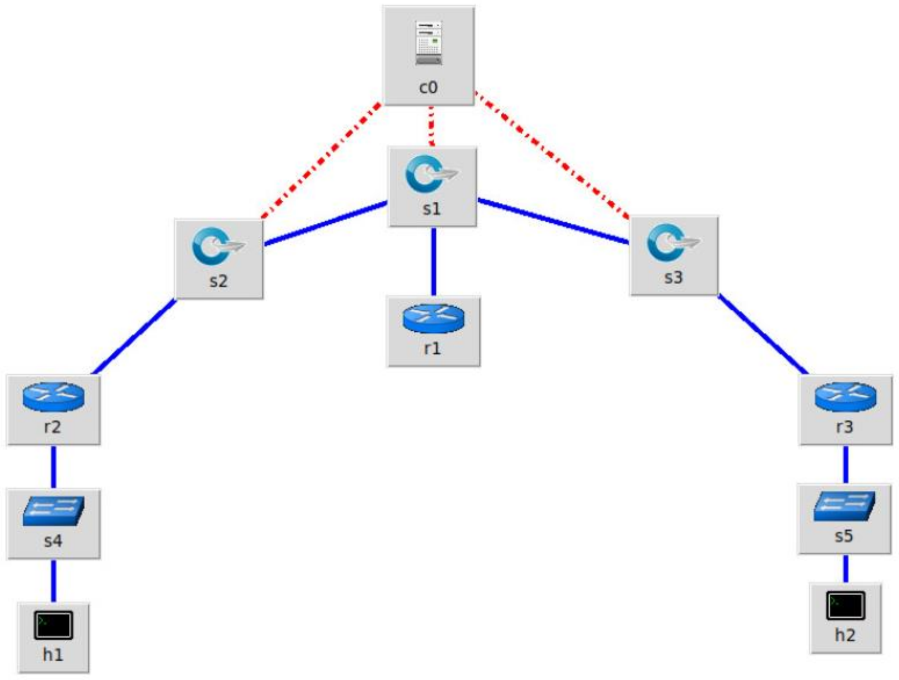


Figure 6. MiniEdit's topology.

## 2.3 Loading the configuration file

At this point, the topology is loaded. However, the interfaces are not configured. In order to assign IP addresses to the interfaces of the devices, you will execute a script that loads the configuration to the routers.

**Step 1.** Click on the icon below to open the Linux terminal.

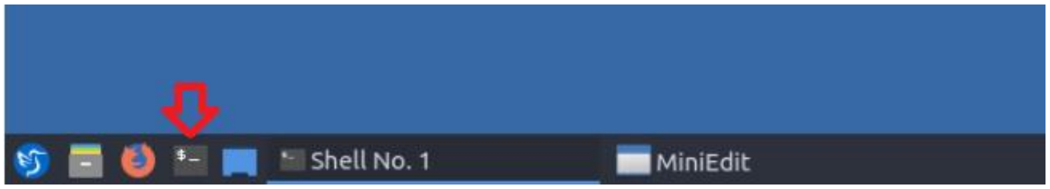
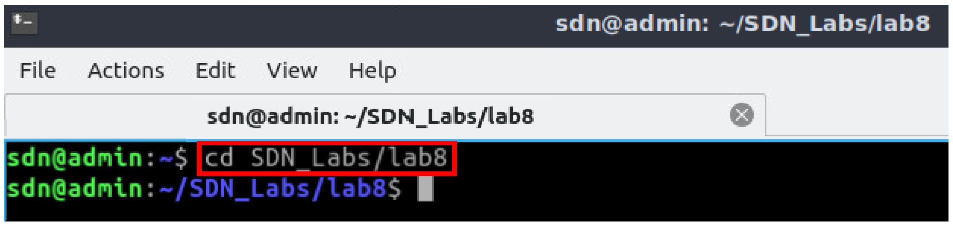


Figure 7. Opening the Linux terminal.

**Step 2.** Click on the Linux terminal and navigate into *SDN_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 IP addresses to the interfaces of the routers. The `cd` command is short for change directory followed by an argument that specifies the destination directory.

```
cd SDN_Labs/lab8
```

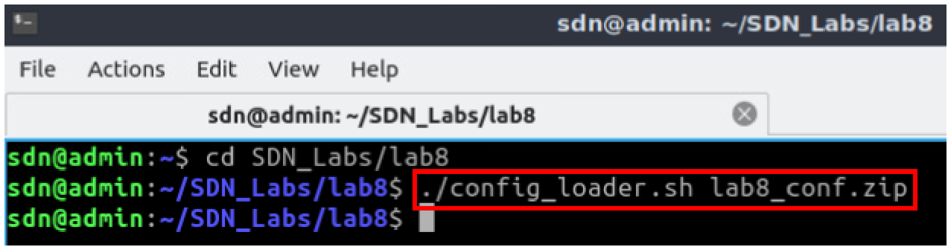


```
sdn@admin:~$ cd SDN_Labs/lab8
sdn@admin:~/SDN_Labs/lab8$ 
```

Figure 8. Entering the *SDN_Labs/lab8* 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 lab8_conf.zip
```

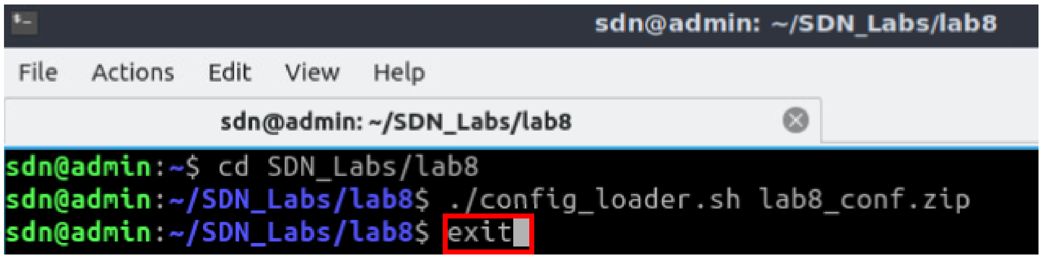


```
sdn@admin:~$ cd SDN_Labs/lab8
sdn@admin:~/SDN_Labs/lab8$ ./config_loader.sh lab8_conf.zip
sdn@admin:~/SDN_Labs/lab8$ 
```

Figure 9. Executing the shell script to load the configuration.

**Step 4.** Type the following command to exit the Linux terminal.

```
exit
```



```
sdn@admin:~$ cd SDN_Labs/lab8
sdn@admin:~/SDN_Labs/lab8$ ./config_loader.sh lab8_conf.zip
sdn@admin:~/SDN_Labs/lab8$ exit
```

Figure 10. Exiting from the terminal.

## 2.4 Running the emulation

In this section, you will run the emulation and check the links and interfaces that connect the devices in the given topology.

**Step 1.** At this point, host h1 and host h2 interfaces are configured. To proceed with the emulation, click on the *Run* button located on the lower left-hand side.

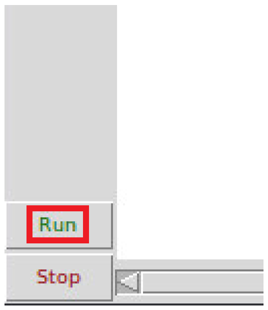
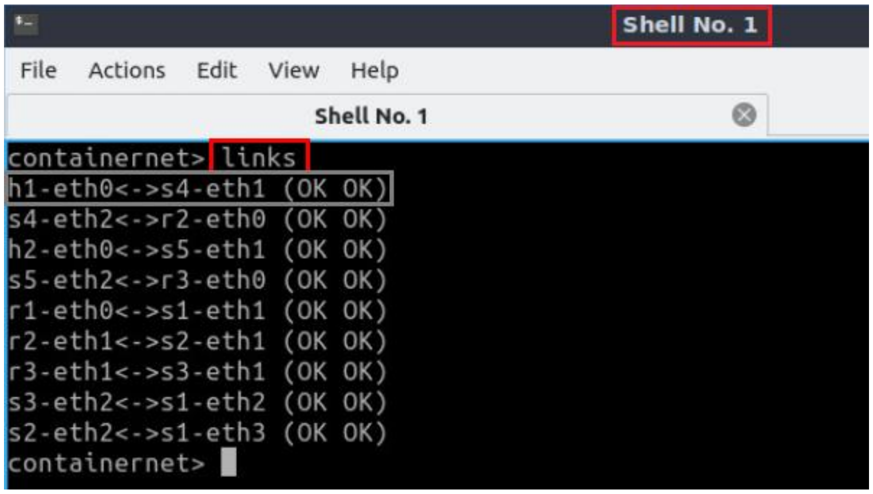


Figure 11. Starting the emulation.

**Step 2.** Issue the following command on Mininet terminal to display the interface names and connections.

```
links
```



```
containernet> links
h1-eth0<->s4-eth1 (OK OK)
s4-eth2<->r2-eth0 (OK OK)
h2-eth0<->s5-eth1 (OK OK)
s5-eth2<->r3-eth0 (OK OK)
r1-eth0<->s1-eth1 (OK OK)
r2-eth1<->s2-eth1 (OK OK)
r3-eth1<->s3-eth1 (OK OK)
s3-eth2<->s1-eth2 (OK OK)
s2-eth2<->s1-eth3 (OK OK)
containernet>
```

Figure 12. Displaying network interfaces.

In Figure 12, the link displayed within the gray box indicates that interface *eth1* of switch s4 connects to interface *eth0* of host h1 (i.e., *s4-eth1<->h1-eth0*).

## 2.5 Verifying the configuration

You will verify the IP addresses listed in Table 2 and inspect the routing table of routers r1, r2, and r3.

**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.

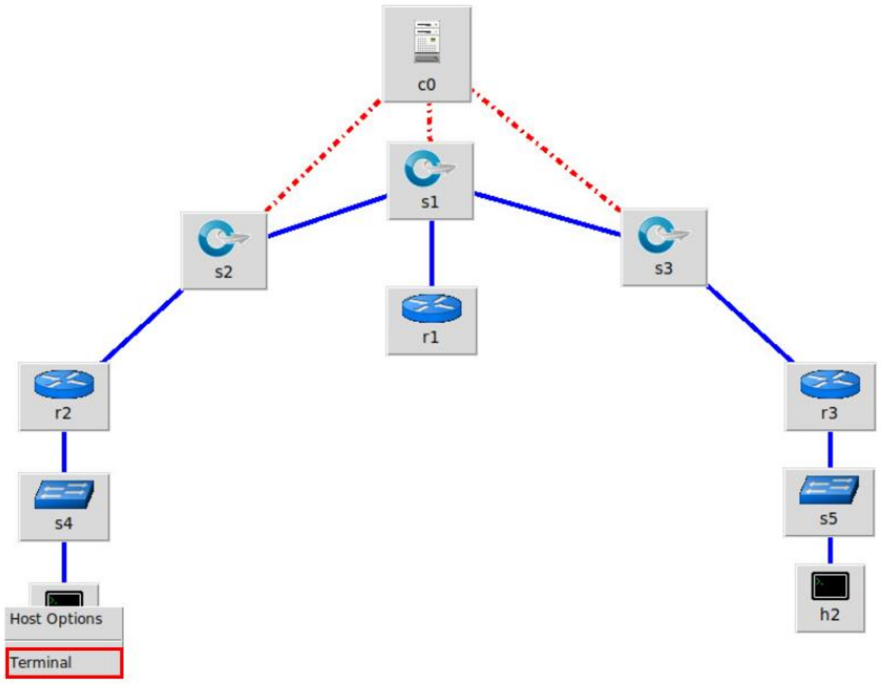
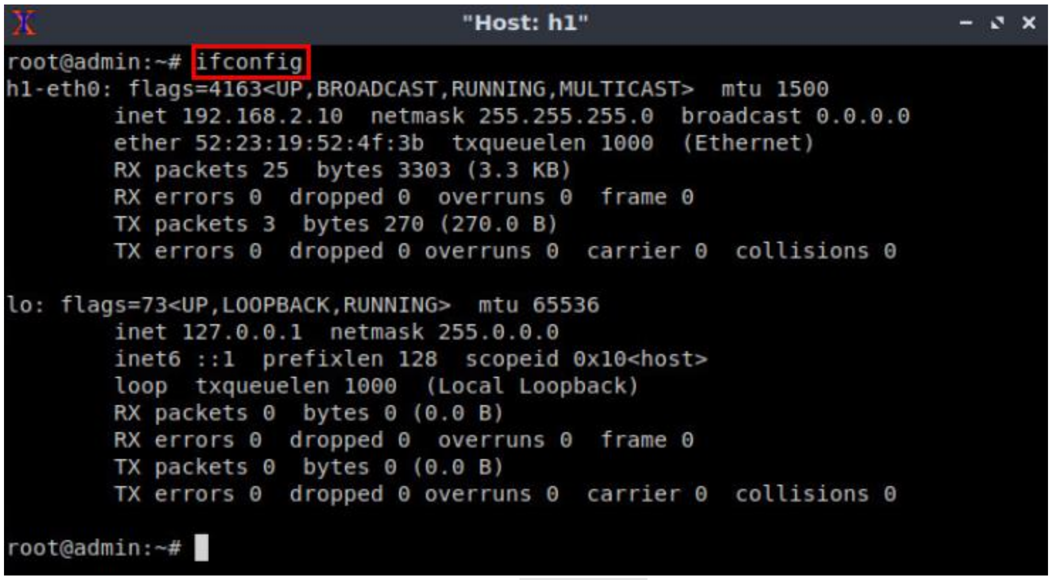


Figure 13. Opening a terminal on host h1.

**Step 2**. On host h1 terminal, type the command shown below to verify that the IP address was assigned successfully. You will corroborate that host h1 has two interfaces. Interface *h1-eth0* is configured with the IP address 192.168.2.10 and the subnet mask 255.255.255.0. Interface *lo* is configured with the IP address 127.0.0.1 and the subnet mask of 255.0.0.0.

```
ifconfig
```

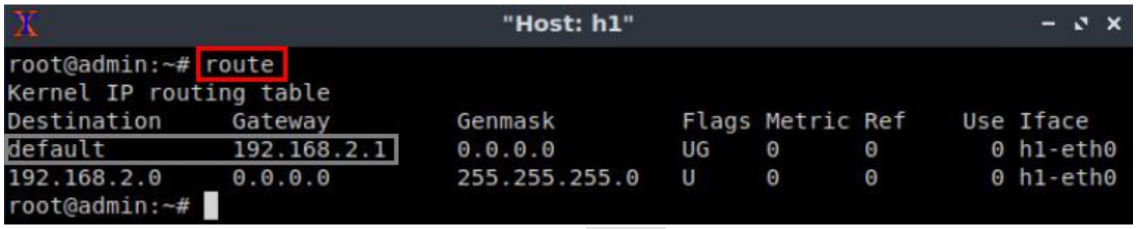


```
root@admin:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.10  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 52:23:19:52:4f:3b  txqueuelen 1000  (Ethernet)
        RX packets 25  bytes 3303 (3.3 KB)
        RX errors 0  dropped 0  overruns 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:~# 
```

Figure 14. Output of `ifconfig` command.

**Step 3**. On host h1 terminal, type the command shown below to verify that the default gateway IP address is 192.168.2.1.

```
route
```



```
root@admin:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.2.1     0.0.0.0         UG    0      0        0 h1-eth0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 h1-eth0
root@admin:~#
```

Figure 15. Output of `route` command.

**Step 4**. In order to verify host h2 IP address and default gateway, proceed similarly by repeating step 1 to step 3 on host h2 terminal. Similar results should be observed.

**Step 5**. In order to verify router r1, hold right-click on router r1 and select *Terminal*.

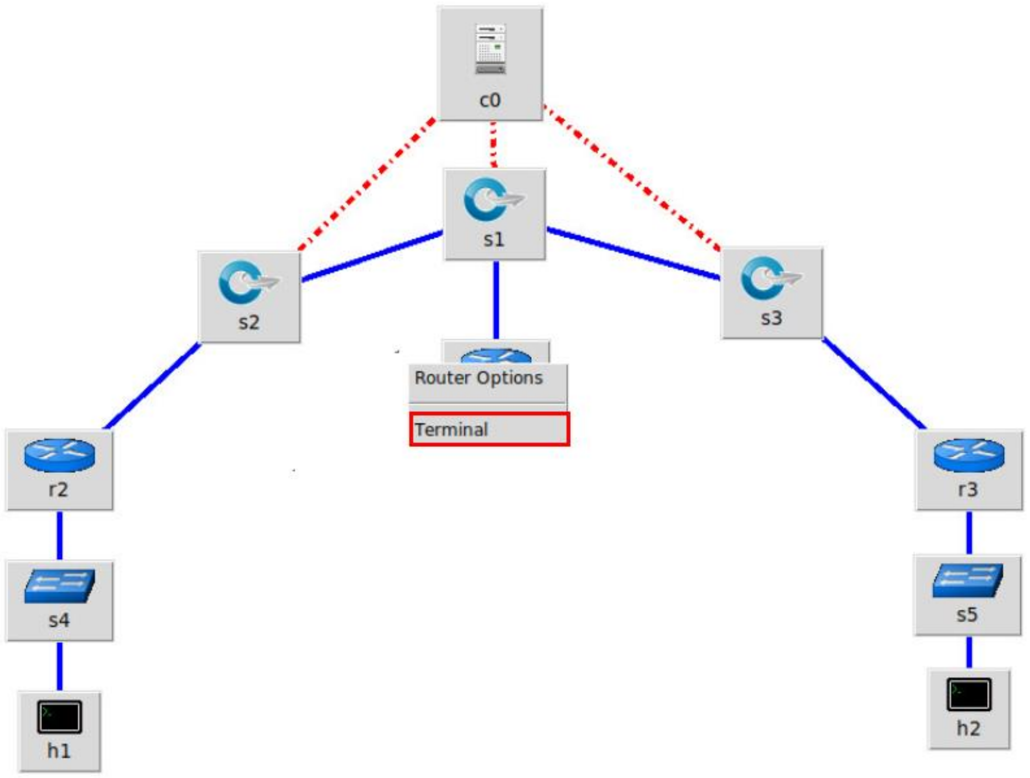
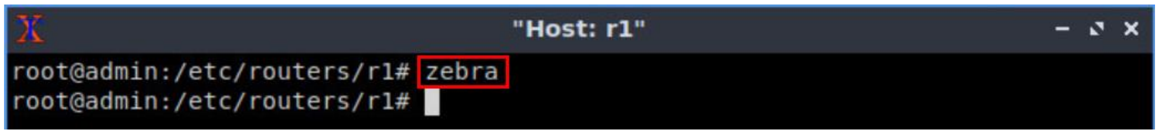


Figure 16. Opening a terminal on router r1.

**Step 6**. In this step, you will start the zebra daemon, 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 zebra, type the following command.

```
zebra
```

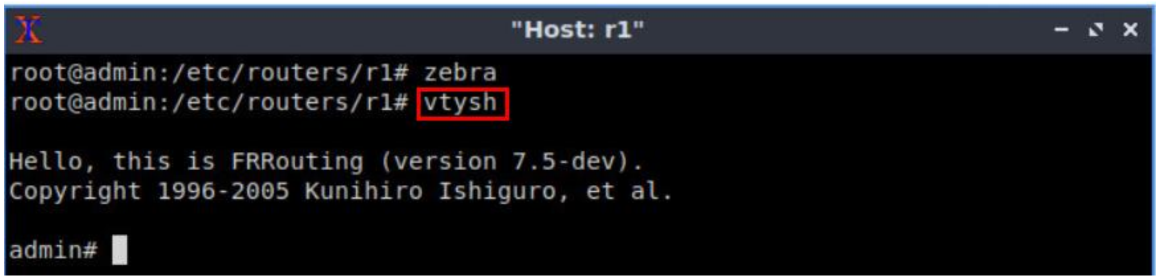


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# 
```

Figure 17. Starting zebra daemon.

**Step 7**. 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
```

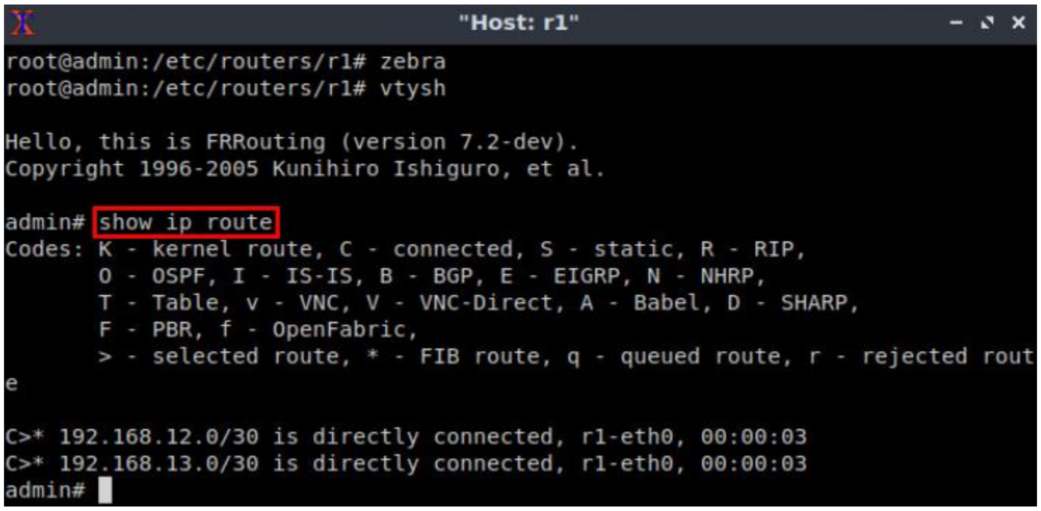


```
root@admin:/etc/routers/r1# zebra
root@admin:/etc/routers/r1# vtysh

Hello, this is FRRouting (version 7.5-dev).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

admin# 
```

Figure 18. Starting vtysh on router r1.

**Step 8.** Type the following command on router r1 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 of router r2 (192.168.2.0/24) or router r3 (192.168.3.0/24) as there is no routing protocol configured yet.

```
show ip route
```



```
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# 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 rout
e

C>* 192.168.12.0/30 is directly connected, r1-eth0, 00:00:03
C>* 192.168.13.0/30 is directly connected, r1-eth0, 00:00:03
admin# 
```

Figure 19. Displaying the routing table of router r1.

The output in the figure above shows that the networks 192.168.12.0/24 and 192.168.13.0/30 are directly connected through the interface *r1-eth0*.

**Step 9.** Hold right-click on router r2 and select *Terminal*.

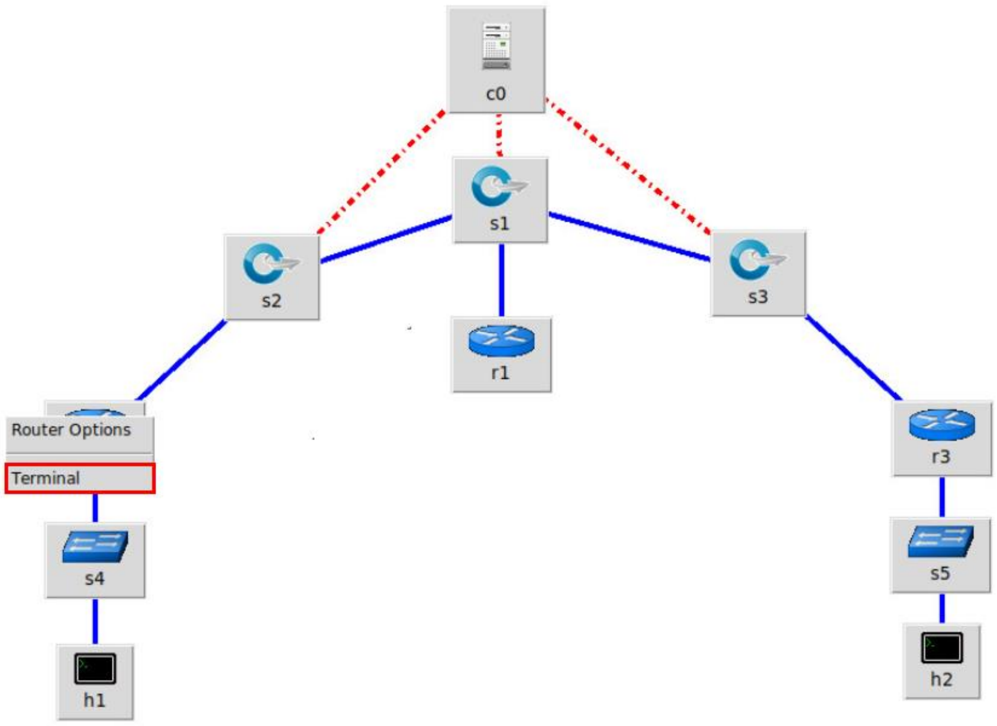
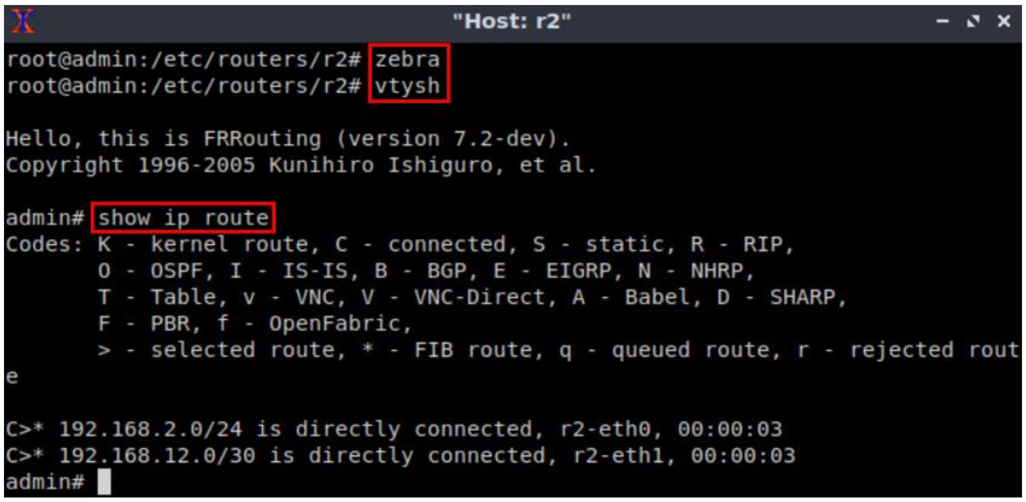


Figure 20. Opening a terminal on router r2.

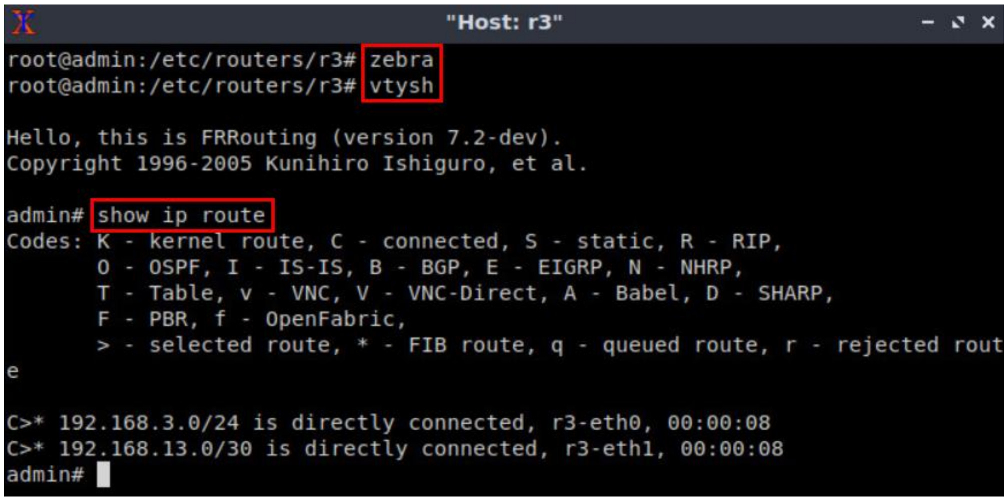
**Step 10.** Router r2 is configured similarly to router r1 but with different IP addresses (see Table 2). Those steps are summarized in the following figure. To proceed, in router r2 terminal issue the commands depicted below. In the end, you will verify all the directly connected networks of router 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,
       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 rout
e

C>* 192.168.2.0/24 is directly connected, r2-eth0, 00:00:03
C>* 192.168.12.0/30 is directly connected, r2-eth1, 00:00:03
admin#
```

Figure 21. Displaying the routing table of router r2.

**Step 11.** 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 terminal issue the commands depicted below. In the end, you will verify all the directly connected networks of router 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,
       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 rout
e

C>* 192.168.3.0/24 is directly connected, r3-eth0, 00:00:08
C>* 192.168.13.0/30 is directly connected, r3-eth1, 00:00:08
admin#
```

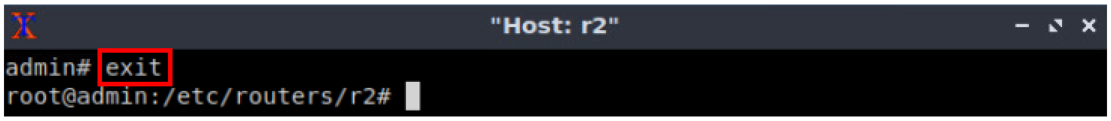
Figure 22. Displaying the routing table of router r3.

## 3 Configuring BGP within legacy networks

In the previous section, you used a script to assign the IP addresses to all device interfaces. In this section, you will configure the BGP routing protocol on the legacy networks (routers r2 and r3). First, you will initialize the daemon that enables BGP configuration. Then, you need to assign BGP neighbors to allow BGP peering to the remote neighbor. Additionally, you will advertise the local networks so that they are advertised to EBGP neighbors.

**Step 1.** To configure the BGP routing protocol, you need to enable the BGP daemon first. In router r2, type the following command to exit the vtysh session.

```
exit
```

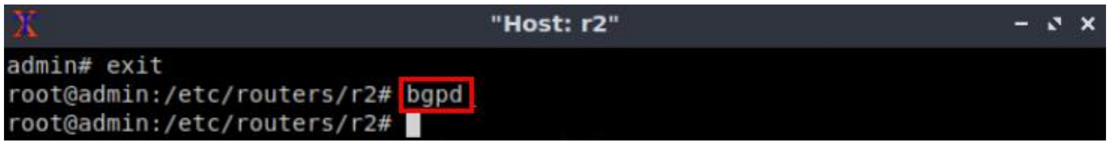


```
admin# exit
root@admin:/etc/routers/r2#
```

Figure 23. Exiting the vtysh session.

**Step 2.** Type the following command on router r2 terminal to start the BGP routing protocol.

```
bgpd
```

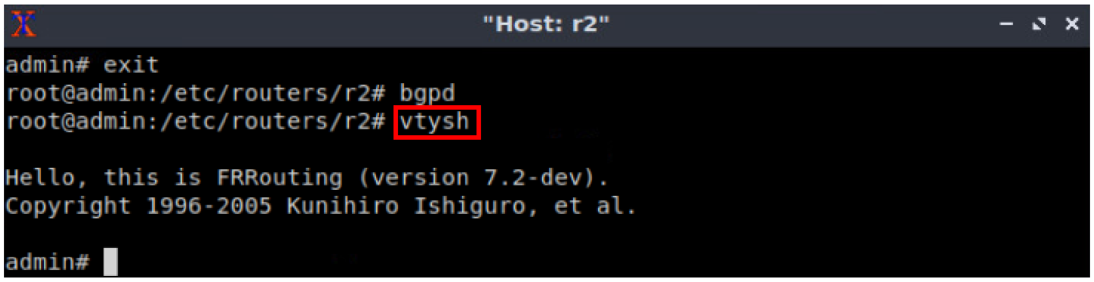


```
admin# exit
root@admin:/etc/routers/r2# bgpd
root@admin:/etc/routers/r2#
```

Figure 24. Starting BGP daemon.

**Step 3.** In order to enter to router r2 terminal, type the following command.

```
vtysh
```

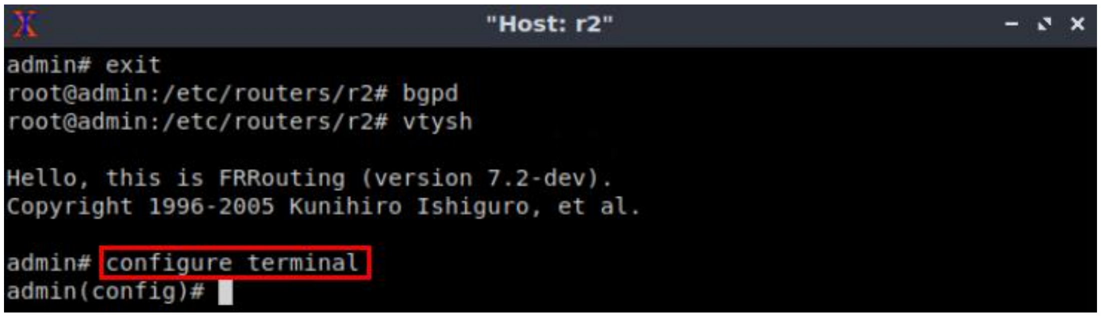


```
admin# exit
root@admin:/etc/routers/r2# bgpd
root@admin:/etc/routers/r2# vtysh

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

admin#
```

Figure 25. Starting vtysh on router r2.

**Step 4.** To enable router r2 global configuration mode, issue the following command.

```
configure terminal
```

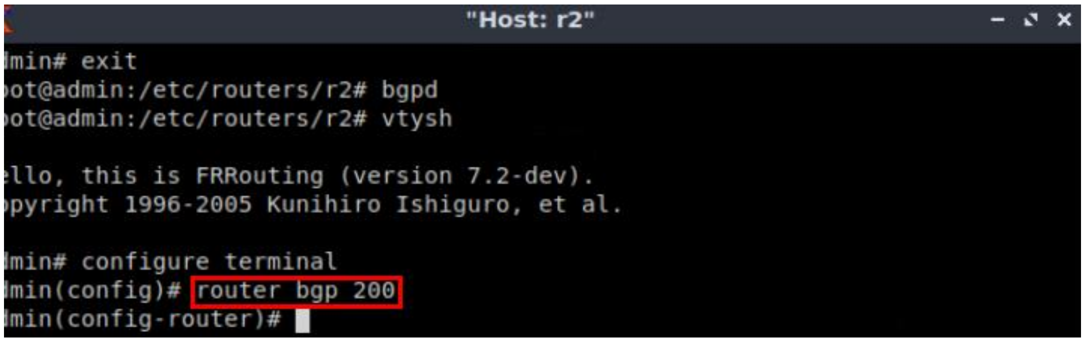


```
admin# exit
root@admin:/etc/routers/r2# bgpd
root@admin:/etc/routers/r2# vtysh

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

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

Figure 26. Enabling configuration mode on router r2.

**Step 5.** The Autonomous System Number (ASN) assigned for router r2 is 200. In order to configure BGP, type the following command.

```
router bgp 200
```

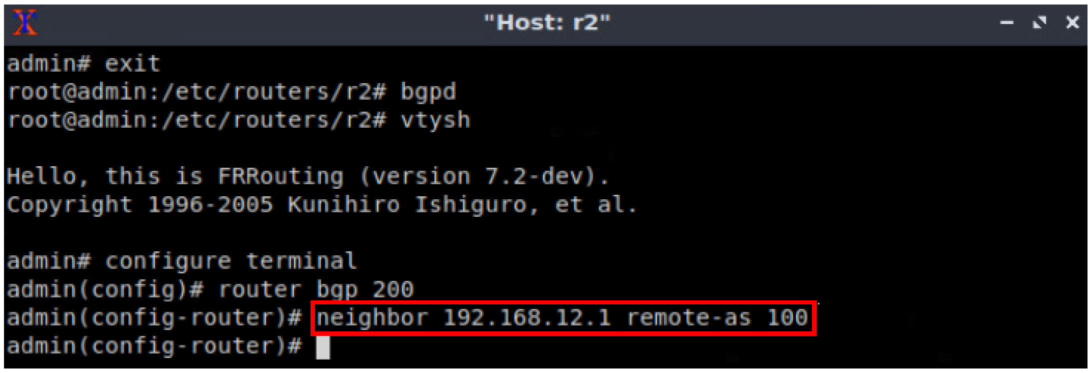


```
dmin# exit
oot@admin:/etc/routers/r2# bgpd
oot@admin:/etc/routers/r2# vtysh

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

dmin# configure terminal
dmin(config)# router bgp 200
dmin(config-router)#
```

Figure 27. Configuring BGP on router r2.

**Step 6.** To configure a BGP neighbor to router r2 (AS 200), type the command shown below. This command specifies the neighbor IP address (192.168.12.1) and ASN of the remote BGP peer (AS 100).

```
neighbor 192.168.12.1 remote-as 100
```

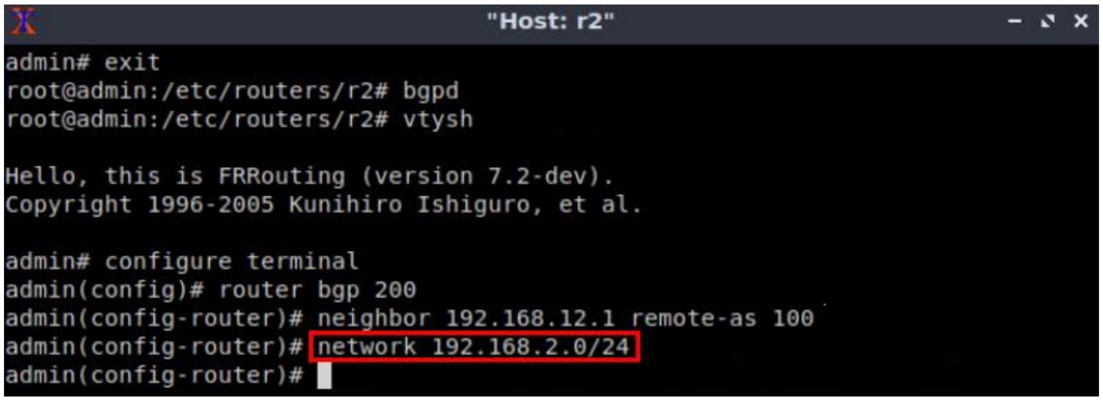


```
admin# exit
root@admin:/etc/routers/r2# bgpd
root@admin:/etc/routers/r2# vtysh

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

admin# configure terminal
admin(config)# router bgp 200
admin(config-router)# neighbor 192.168.12.1 remote-as 100
admin(config-router)#
```

Figure 28. Assigning BGP neighbor to router r2.

**Step 7.** Issue the following command so that router r2 advertises the network 192.168.2.0/24.

```
network 192.168.2.0/24
```

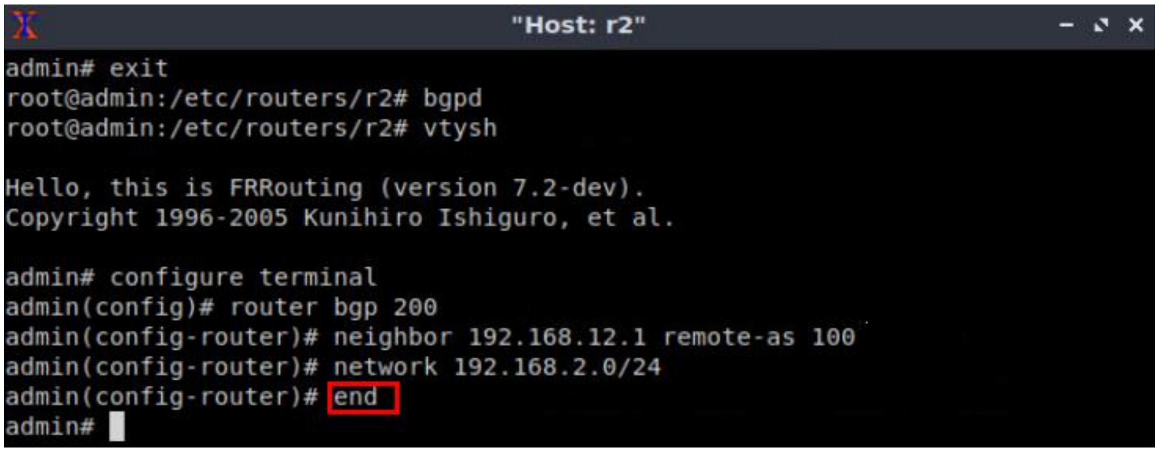


```
admin# exit
root@admin:/etc/routers/r2# bgpd
root@admin:/etc/routers/r2# vtysh

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

admin# configure terminal
admin(config)# router bgp 200
admin(config-router)# neighbor 192.168.12.1 remote-as 100
admin(config-router)# network 192.168.2.0/24
admin(config-router)#
```

Figure 29. Advertising the network connected to router r2.

**Step 8.** Type the following command to exit from the configuration mode.

```
end
```

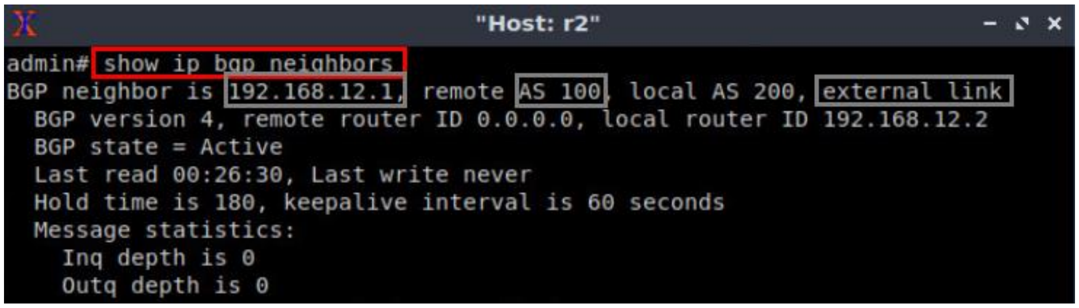


```
admin# exit
root@admin:/etc/routers/r2# bgpd
root@admin:/etc/routers/r2# vtysh

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

admin# configure terminal
admin(config)# router bgp 200
admin(config-router)# neighbor 192.168.12.1 remote-as 100
admin(config-router)# network 192.168.2.0/24
admin(config-router)# end
admin#
```

Figure 30. Exiting from configuration mode.

**Step 9.** Type the following command to show the BGP neighbors. You will verify that the neighbor’s IP address is 192.168.12.1. The corresponding ASN is 100.

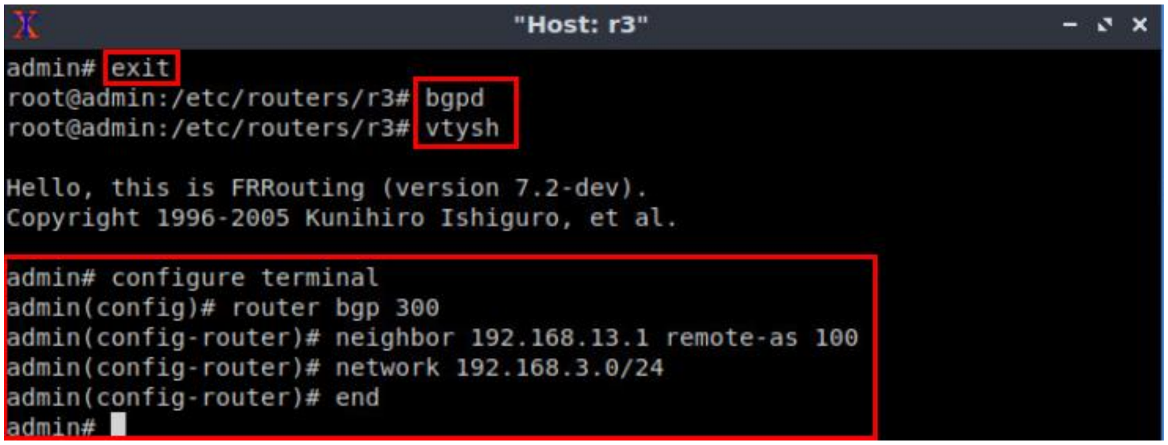
```
show ip bgp neighbors
```



```
admin# show ip bgp neighbors
BGP neighbor is 192.168.12.1, remote AS 100, local AS 200, external link
  BGP version 4, remote router ID 0.0.0.0, local router ID 192.168.12.2
  BGP state = Active
  Last read 00:26:30, Last write never
  Hold time is 180, keepalive interval is 60 seconds
  Message statistics:
    Inq depth is 0
    Outq depth is 0
```

Figure 31. Verifying BGP neighbors on router r2.

**Step 10.** The configuration of BGP on router r3 is similarly configured as router r2. Router r3 lies within AS 300, it establishes BGP neighbor relationship with router r1 (192.168.13.1) in AS 100 and advertises the network 192.168.3.0/24. The configuration of BGP on router r3 is depicted in the figure below.

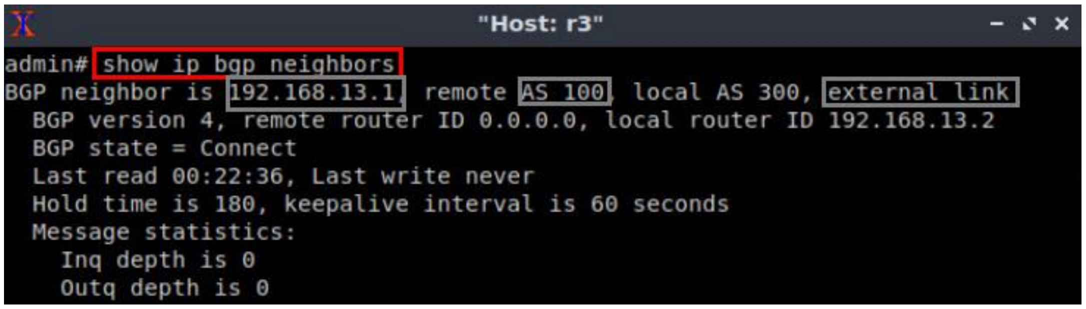


```
admin# exit
root@admin:/etc/routers/r3# bgpd
root@admin:/etc/routers/r3# vtysh

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

admin# configure terminal
admin(config)# router bgp 300
admin(config-router)# neighbor 192.168.13.1 remote-as 100
admin(config-router)# network 192.168.3.0/24
admin(config-router)# end
admin#
```

Figure 32. BGP configuration on router r3.

**Step 11.** To verify BGP neighbors of router r3, type the following command.

```
show ip bgp neighbors
```



```
admin# show ip bgp neighbors
BGP neighbor is 192.168.13.1, remote AS 100, local AS 300, external link
  BGP version 4, remote router ID 0.0.0.0, local router ID 192.168.13.2
  BGP state = Connect
  Last read 00:22:36, Last write never
  Hold time is 180, keepalive interval is 60 seconds
  Message statistics:
    Inq depth is 0
    Outq depth is 0
```

Figure 33. Verifying BGP neighbors on router r3.

Currently no BGP peering session has been established since BGP is not configured yet on the neighboring router r1.

## 4 Starting the ONOS controller

In this section, you will start the ONOS controller and activate OpenFlow application so that the controller discovers the devices, hosts, and links in the topology.

**Step 1.** Go to the Linux terminal, Shell No. 1.

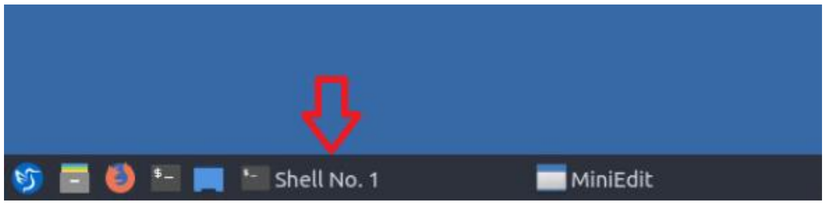


Figure 34. Opening Linux terminal.

**Step 2.** Click on *File>New Tab* to open an additional tab in the Linux terminal. Alternatively, you may press `Ctrl+Shift+T`.

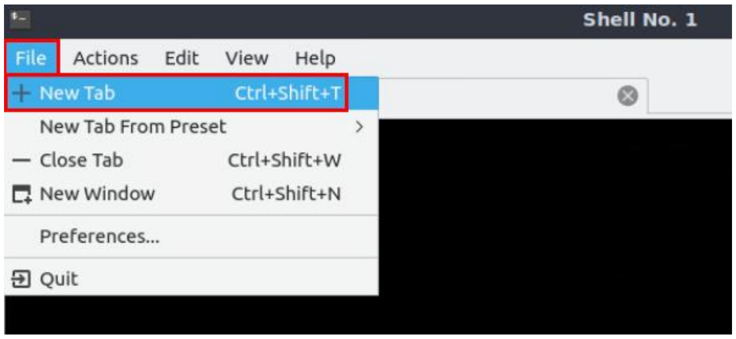
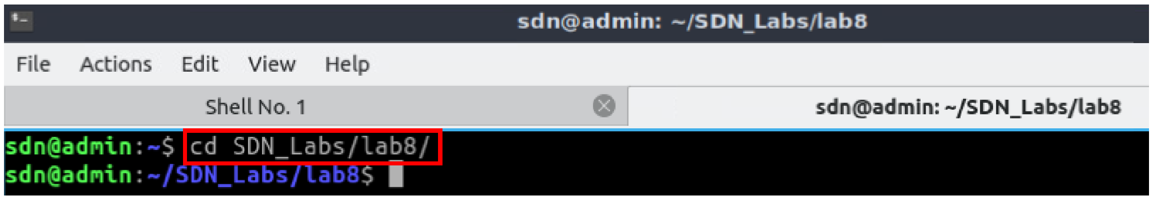


Figure 35. Opening an additional tab.

**Step 3.** Navigate into *SDN_Labs/lab8/* directory by issuing the following command.

```
cd SDN_Labs/lab8/
```

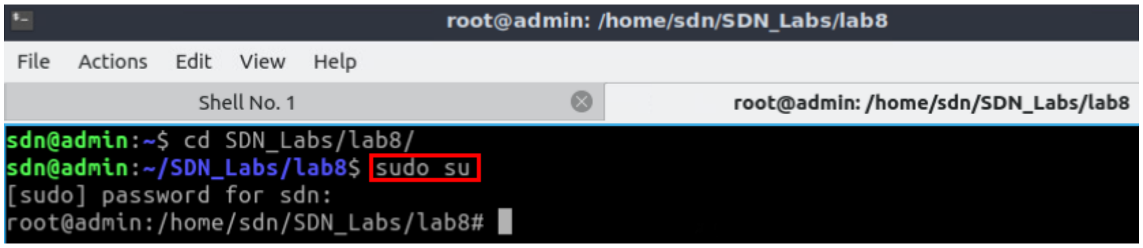


```
sdn@admin:~$ cd SDN_Labs/lab8/
sdn@admin:~/SDN_Labs/lab8$
```

Figure 36. Entering the *SDN_Labs/lab8/* directory.

**Step 4.** Issue the command below to execute programs with the security privileges of the superuser (root). When prompted for a password, type `password`.

```
sudo su
```

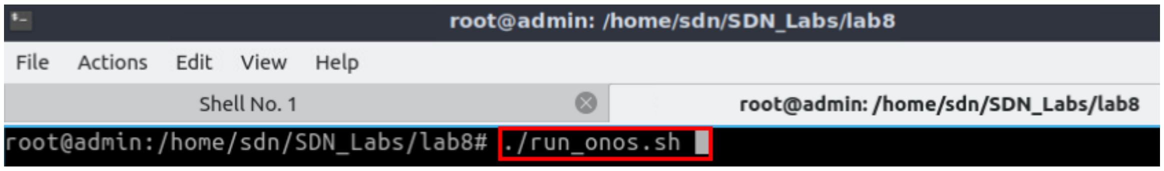

root@admin: /home/sdn/SDN_Labs/lab8
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab8

```
sdn@admin:~$ cd SDN_Labs/lab8/
sdn@admin:~/SDN_Labs/lab8$ sudo su
[sudo] password for sdn:
root@admin:/home/sdn/SDN_Labs/lab8#
```

Figure 37. Switching to root mode.

**Step 5.** A script was written to run ONOS and enter its Command Line Interface (CLI). In order to run the script, issue the following command. In addition to running ONOS, the script will modify the Media Access Control (MAC) addresses of the hosts and routers so that they conform with the topology.

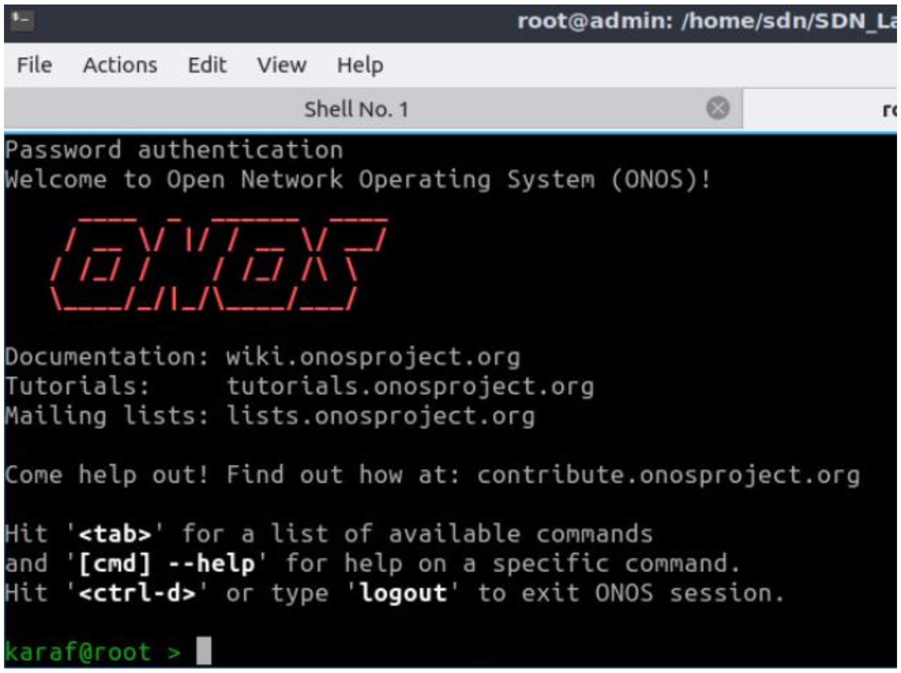
```
./run_onos.sh
```


root@admin: /home/sdn/SDN_Labs/lab8
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab8

```
root@admin:/home/sdn/SDN_Labs/lab8# ./run_onos.sh
```

Figure 38. Starting the ONOS controller.

Once the script finishes executing and ONOS is ready, you will be able to execute commands on the ONOS CLI as shown in the figure below. Note that this script may take few seconds.

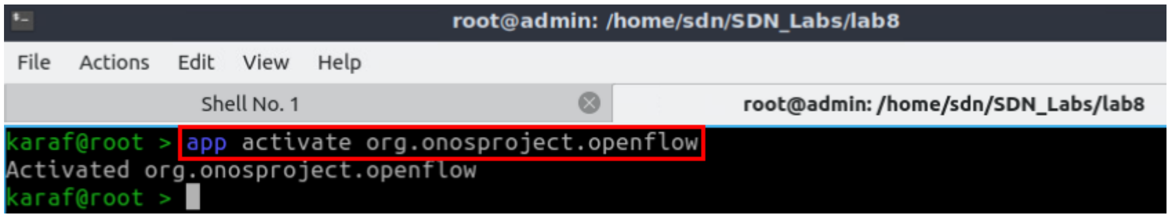


```
Password authentication
Welcome to Open Network Operating System (ONOS)!

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

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

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

karaf@root >
```

Figure 39. ONOS CLI.

**Step 6.** In the ONOS terminal, issue the following command to activate the OpenFlow application.

```
app activate org.onosproject.openflow
```


root@admin: /home/sdn/SDN_Labs/lab8
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab8

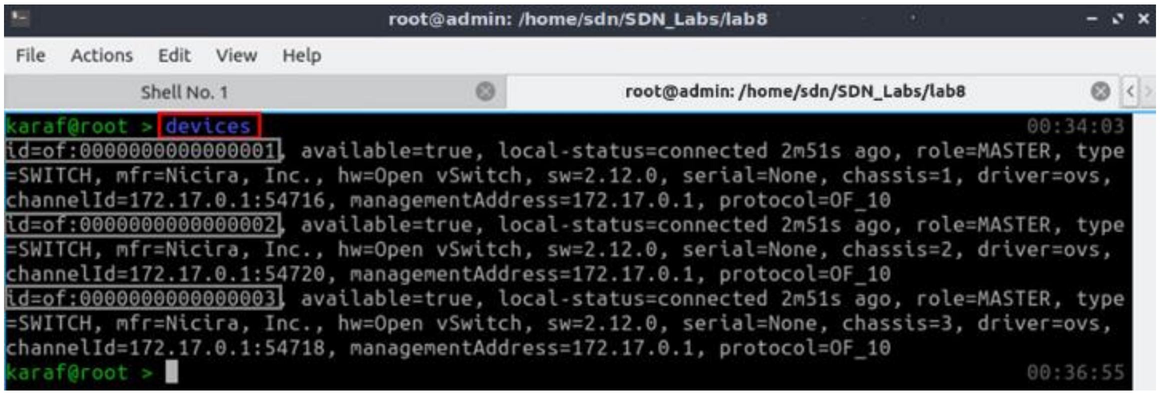
```
karaf@root > app activate org.onosproject.openflow
Activated org.onosproject.openflow
karaf@root >
```

Figure 40. Activating OpenFlow application.

Note that when you activate any ONOS application, you may have to wait few seconds so that the application gives the correct output.

**Step 8.** To display the list of all currently known devices (OVS switches), type the following command.

```
devices
```

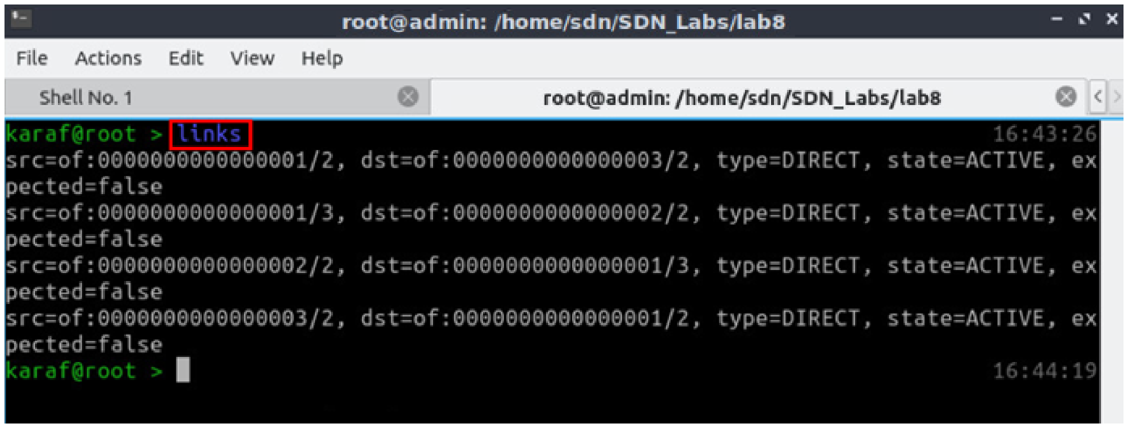

root@admin: /home/sdn/SDN_Labs/lab8
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab8

```
karaf@root > devices                                                    00:34:03
id=of:0000000000000001, available=true, local-status=connected 2m51s ago, role=MASTER, type
=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=1, driver=ovs,
channelId=172.17.0.1:54716, managementAddress=172.17.0.1, protocol=OF_10
id=of:0000000000000002, available=true, local-status=connected 2m51s ago, role=MASTER, type
=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=2, driver=ovs,
channelId=172.17.0.1:54720, managementAddress=172.17.0.1, protocol=OF_10
id=of:0000000000000003, available=true, local-status=connected 2m51s ago, role=MASTER, type
=SWITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=3, driver=ovs,
channelId=172.17.0.1:54718, managementAddress=172.17.0.1, protocol=OF_10
karaf@root >                                                            00:36:55
```

Figure 41. Displaying the currently known devices (switches).

**Step 9.** To display the list of all currently known links, type the following command.

```
links
```

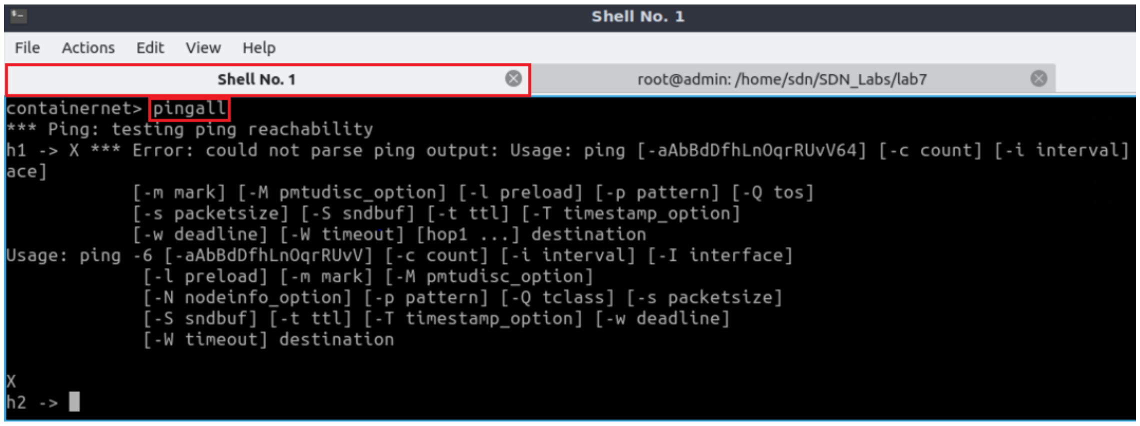

root@admin: /home/sdn/SDN_Labs/lab8
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab8

```
karaf@root > links                                                      16:43:26
src=of:0000000000000001/2, dst=of:0000000000000003/2, type=DIRECT, state=ACTIVE, ex
pected=false
src=of:0000000000000001/3, dst=of:0000000000000002/2, type=DIRECT, state=ACTIVE, ex
pected=false
src=of:0000000000000002/2, dst=of:0000000000000001/3, type=DIRECT, state=ACTIVE, ex
pected=false
src=of:0000000000000003/2, dst=of:0000000000000001/2, type=DIRECT, state=ACTIVE, ex
pected=false
karaf@root >                                                            16:44:19
```

Figure 42. Displaying the currently known links.

**Step 10**. Click on the Mininet tab on the left-hand side and type the command shown below to ping all hosts in the topology.

```
pingall
```


Shell No. 1
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

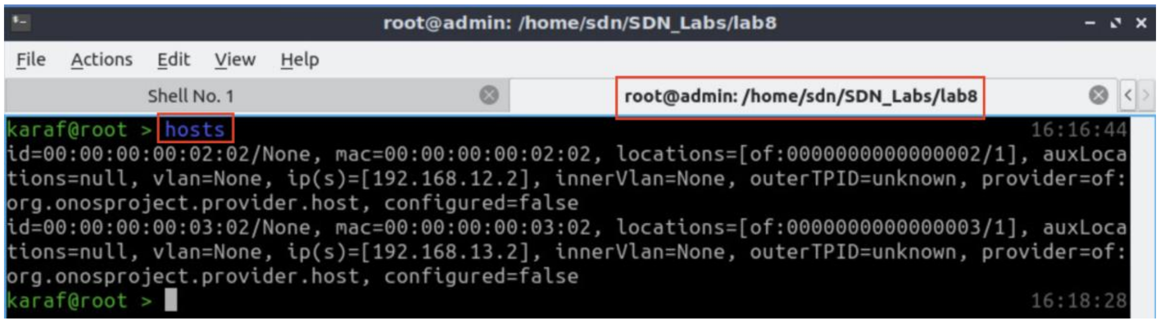
```
containernet> pingall
*** Ping: testing ping reachability
h1 -> X *** Error: could not parse ping output: Usage: ping [-aAbBdDfhLnOqrRUvV64] [-c count] [-i interval]
ace]
            [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
            [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
            [-w deadline] [-W timeout] [hop1 ...] destination
Usage: ping -6 [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
             [-l preload] [-m mark] [-M pmtudisc_option]
             [-N nodeinfo_option] [-p pattern] [-Q tclass] [-s packetsize]
             [-S sndbuf] [-t ttl] [-T timestamp_option] [-w deadline]
             [-W timeout] destination

X
h2 ->
```

Figure 43. Pinging all the hosts.

Consider the figure above. The purpose of this step is to provide hosts information to the controller.

**Step 11.** To display the list of all currently known hosts, type the following command in the ONOS terminal.

```
hosts
```


root@admin: /home/sdn/SDN_Labs/lab8
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab8

```
karaf@root > hosts                                                              16:16:44
id=00:00:00:00:02:02/None, mac=00:00:00:00:02:02, locations=[of:0000000000000002/1], auxLoca
tions=null, vlan=None, ip(s)=[192.168.12.2], innerVlan=None, outerTPID=unknown, provider=of:
org.onosproject.provider.host, configured=false
id=00:00:00:00:03:02/None, mac=00:00:00:00:03:02, locations=[of:0000000000000003/1], auxLoca
tions=null, vlan=None, ip(s)=[192.168.13.2], innerVlan=None, outerTPID=unknown, provider=of:
org.onosproject.provider.host, configured=false
karaf@root >                                                                    16:18:28
```

Figure 44. Displaying the current known links.

Consider Figure 44. ONOS recognizes router r2 (192.168.12.2) and router r3 (192.168.13.2) and displays the interfaces of the OpenFlow switches they are connected to. Note that you might have to wait until ONOS discovers the two hosts in case they do not appear immediately.

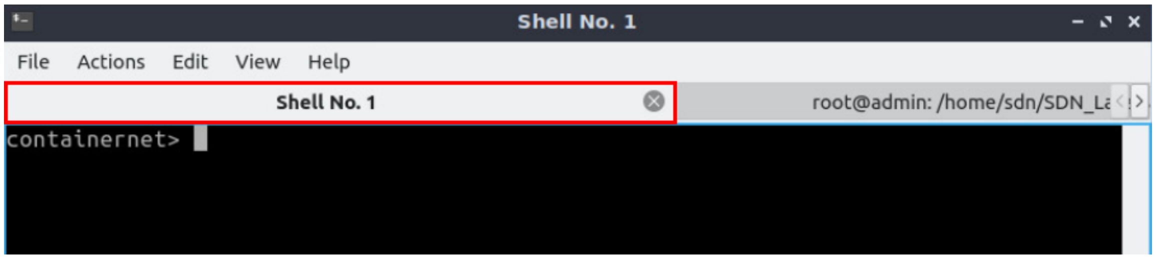
## 5 Integrating SDN and legacy networks

In the previous sections, you configured the legacy devices as well as started ONOS and its OpenFlow application to discover the topology. In this section, you will first execute a script that connects the IBGP speaker (router r1) with the ONOS controller, so that the two entities can communicate. Additionally, you will configure BGP on router r1 so that it peers with routers r2 and r3 in the external networks, as well as with ONOS. Furthermore, you will activate the ONOS SDN-IP application to interconnect the three ASes.

## 5.1 Connecting the IBGP speaker (router r1) with the ONOS controller

In this section, you will execute a script that creates a peer-to-peer link connecting router r1 with ONOS.

**Step 1.** Go to Mininet tab in the Linux terminal.

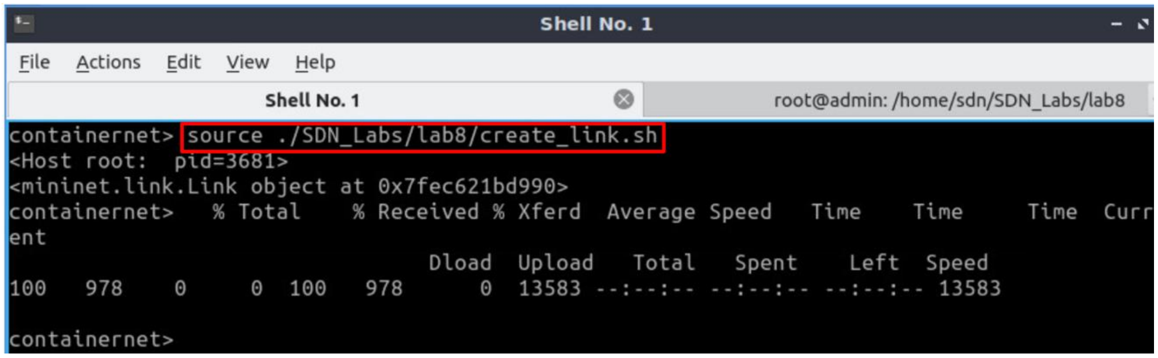


```
containernet>
```

Figure 45. Opening Mininet tab.

**Step 2.** In order to create a point-to-point network between the IBGP speaker (router r1) and ONOS, a script was written to facilitate the process. In order to execute the script, type the following command.

```
source ./SDN_Labs/lab8/create_link.sh
```


Shell No. 1
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab8

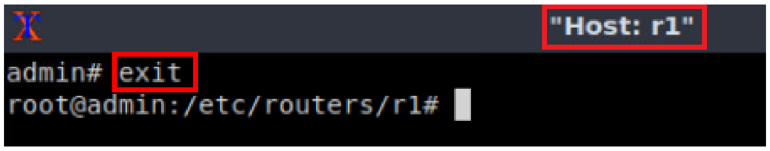
```
containernet> source ./SDN_Labs/lab8/create_link.sh
<Host root:  pid=3681>
<mininet.link.Link object at 0x7fec621bd990>
containernet>   % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curr
ent
                                 Dload  Upload   Total   Spent    Left  Speed
100   978    0     0  100   978      0  13583 --:--:-- --:--:-- --:--:-- 13583

containernet>
```

Figure 46. Creating a point-to-point network (link) between the IBGP speaker and the ONOS controller.

Consider Figure 46. The script creates a point-to-point network between router r1 and ONOS. The network address of the point-to-point network is 10.0.0.0/24. Router r1 is assigned the IP address 10.0.0.1/24, whereas the ONOS controller is assigned 10.0.0.3/24. Furthermore, the script pushes a configuration file *(network-cfg.json*) to the controller necessary to run the SDN-IP application.

**Step 3.** In router r1 terminal, type the following command to exit the vtysh session.

```
exit
```

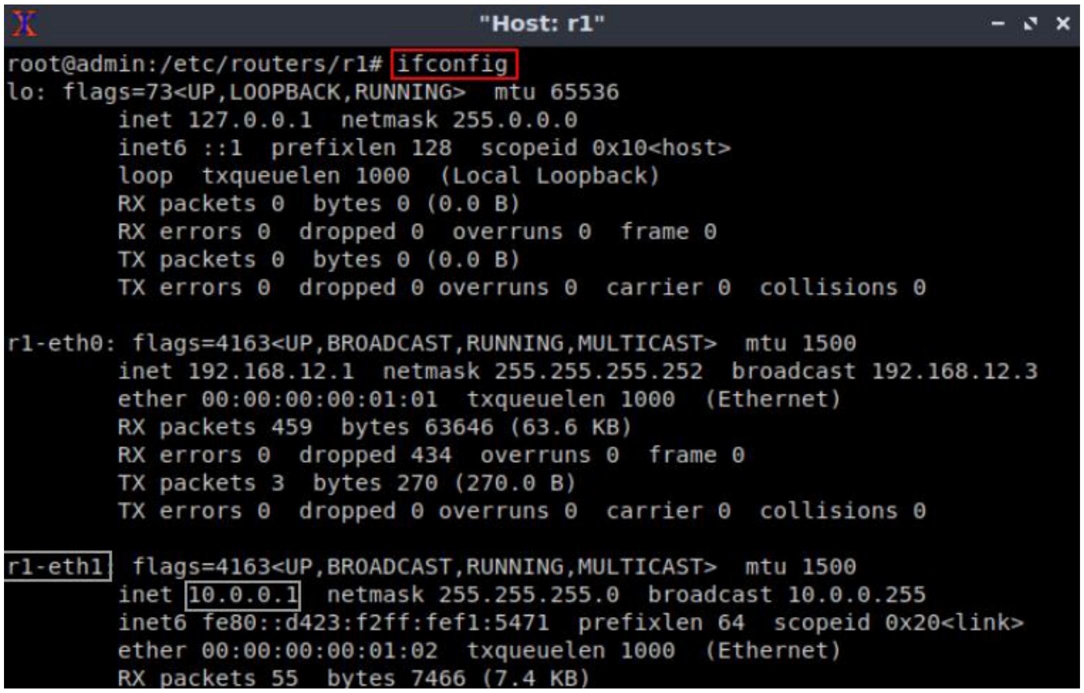


```
"Host: r1"
admin# exit
root@admin:/etc/routers/r1#
```

Figure 47. Creating a network between the IBGP speaker and the ONOS controller.

**Step 4.** Now that router r1 is connected to the ONOS controller, a new interface must appear. In order to verify the connected interface, type the following command.

```
ifconfig
```



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

r1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.12.1  netmask 255.255.255.252  broadcast 192.168.12.3
        ether 00:00:00:00:01:01  txqueuelen 1000  (Ethernet)
        RX packets 459  bytes 63646 (63.6 KB)
        RX errors 0  dropped 434  overruns 0  frame 0
        TX packets 3  bytes 270 (270.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

r1-eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::d423:f2ff:fef1:5471  prefixlen 64  scopeid 0x20<link>
        ether 00:00:00:00:01:02  txqueuelen 1000  (Ethernet)
        RX packets 55  bytes 7466 (7.4 KB)
```

Figure 48. Listing the interfaces of router r1.

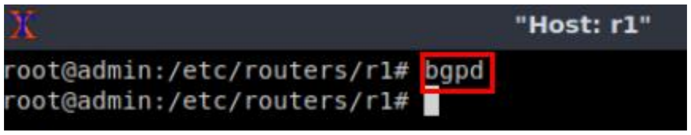
Consider Figure 48. Interface *r1-eth1* is added after creating a point-to-point network between router r1 and the ONOS controller. Furthermore, the interface has the IP address 10.0.0.1.

## 5.2 Configuring BGP on router r1

In this section, you will configure BGP on router r1 to peer with routers r2 and r3, as well as with ONOS.

**Step 1.** Type the following command on router r1 terminal to start the BGP routing protocol.

```
bgpd
```

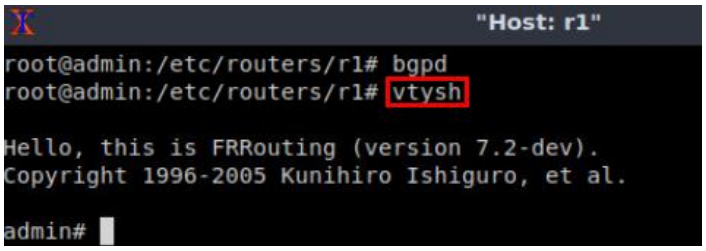


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1#
```

Figure 49. Starting BGP daemon.

**Step 2.** In order to enter to router r1 terminal, type the following command.

```
vtysh
```

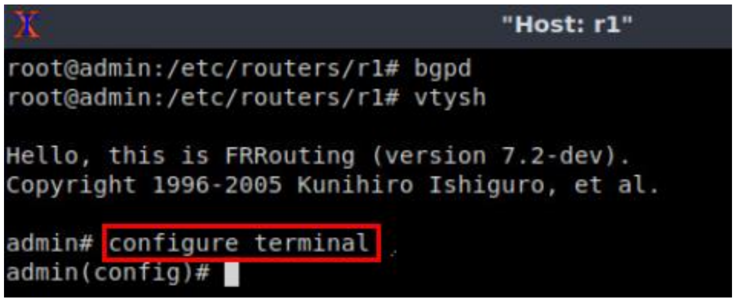


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin#
```

Figure 50. Starting vtysh on router r1.

**Step 3.** To enable router r1 global configuration mode, issue the following command.

```
configure terminal
```

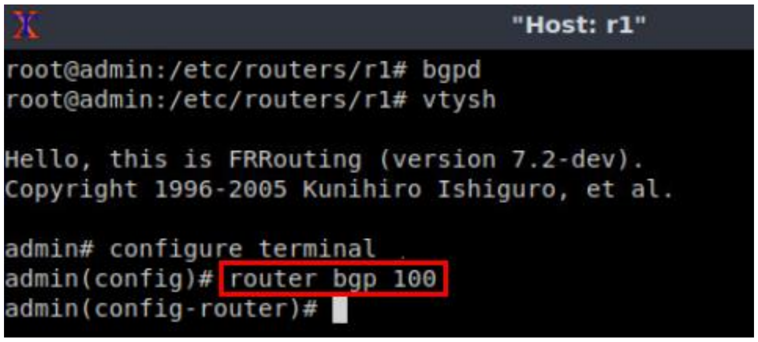


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

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

Figure 51. Enabling configuration mode on router r1.

**Step 4.** The ASN assigned for router r1 is 100. In order to configure BGP, type the following command.

```
router bgp 100
```

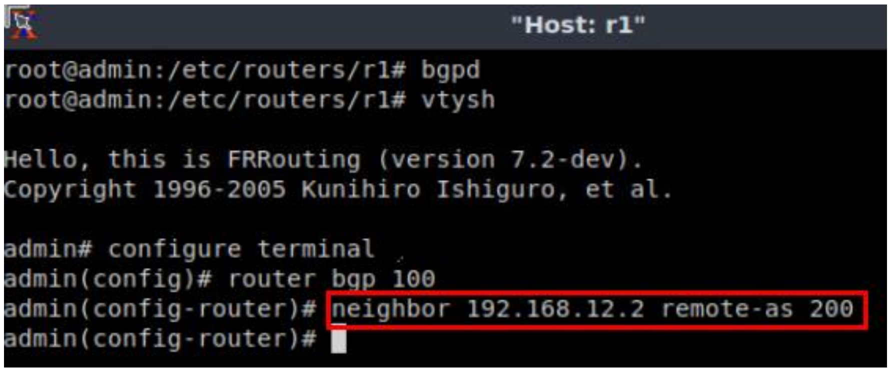


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)#
```

Figure 52. Configuring BGP on router r1.

**Step 5.** To configure a BGP neighbor to router r1 (AS 100), type the command shown below. This command specifies the neighbor IP address (192.168.12.2) and ASN of the remote BGP peer (AS 200).

```
neighbor 192.168.12.2 remote-as 200
```

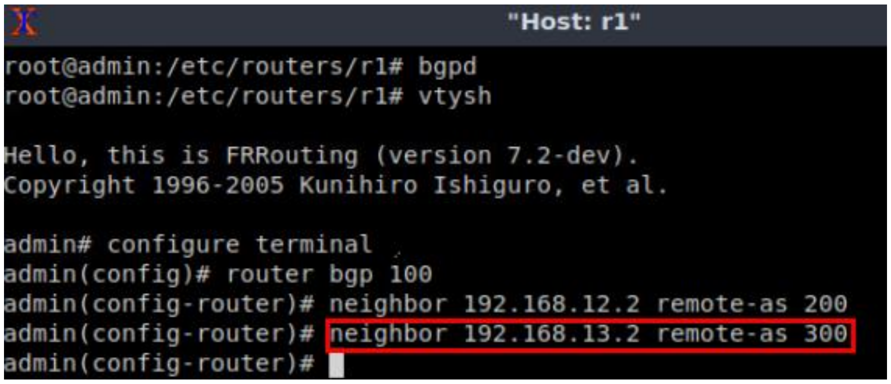


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# neighbor 192.168.12.2 remote-as 200
admin(config-router)#
```

Figure 53. Assigning BGP neighbor to router r1.

**Step 6.** Similarly, add router r3 (192.168.13.2) in AS 300 as a BGP neighbor to router r1.

```
neighbor 192.168.13.2 remote-as 300
```

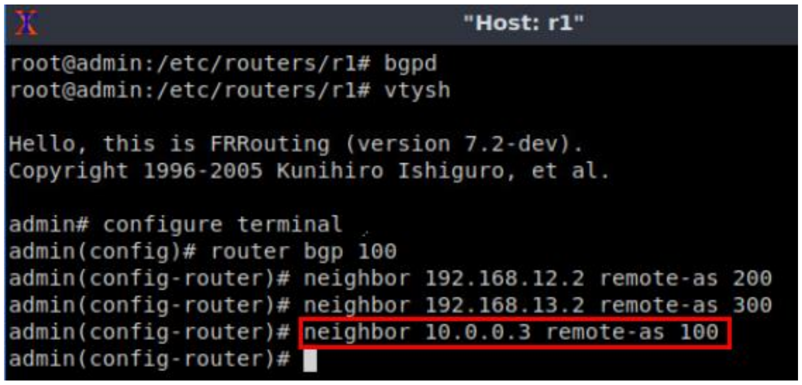


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# neighbor 192.168.12.2 remote-as 200
admin(config-router)# neighbor 192.168.13.2 remote-as 300
admin(config-router)#
```

Figure 54. Assigning BGP neighbor to router r1.

**Step 7.** Router r1 and the ONOS controller are connected using a point-to-point network (10.0.0.0/24). The IP address assigned to the controller is 10.0.0.3. As router r1 is the IBGP speaker within the SDN network, it must establish a BGP peering relationship with the controller in its network (AS 100). In order to establish BGP peering relationship with the controller, type the following command.

```
neighbor 10.0.0.3 remote-as 100
```

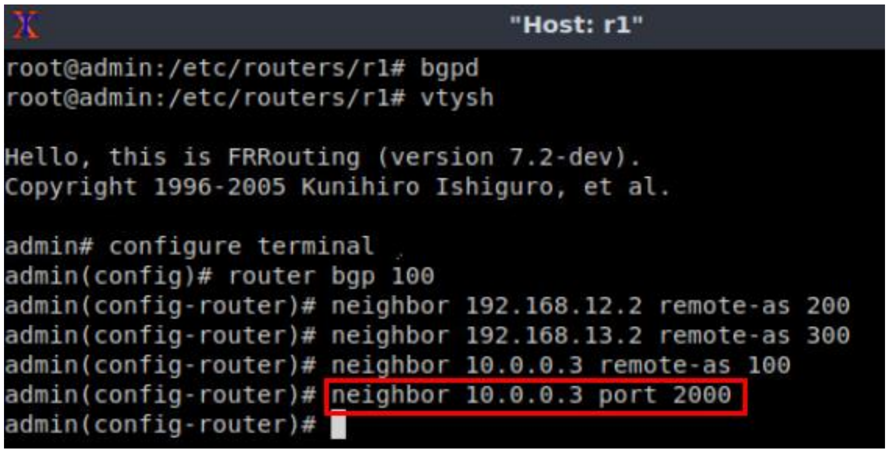


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# neighbor 192.168.12.2 remote-as 200
admin(config-router)# neighbor 192.168.13.2 remote-as 300
admin(config-router)# neighbor 10.0.0.3 remote-as 100
admin(config-router)#
```

Figure 55. Assigning BGP neighbor to router r1.

**Step 8.** By default, ONOS listens on TCP port number 2000 for incoming BGP connections, which is not the default BGP port number 179. In order to specify the port for incoming BGP messages from ONOS, write the following command.

```
neighbor 10.0.0.3 port 2000
```

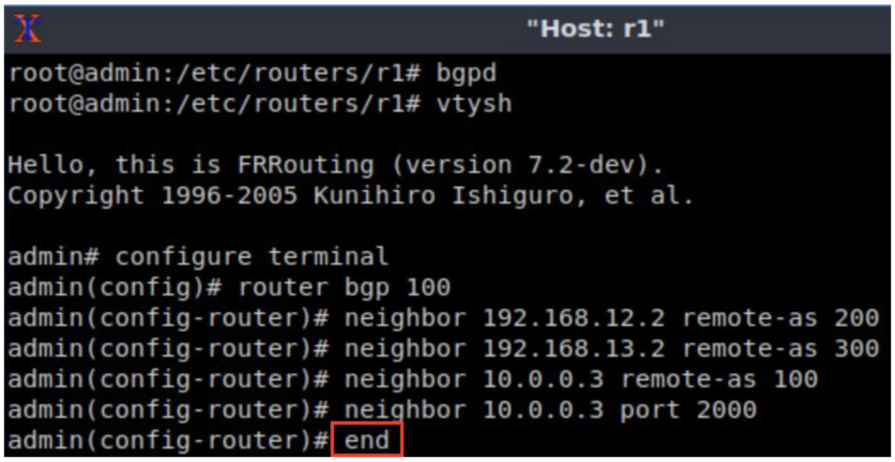


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# neighbor 192.168.12.2 remote-as 200
admin(config-router)# neighbor 192.168.13.2 remote-as 300
admin(config-router)# neighbor 10.0.0.3 remote-as 100
admin(config-router)# neighbor 10.0.0.3 port 2000
admin(config-router)#
```

Figure 56. Changing the Listening port for BGP connections.

**Step 9.** Type the following command to exit from the configuration mode.

```
end
```

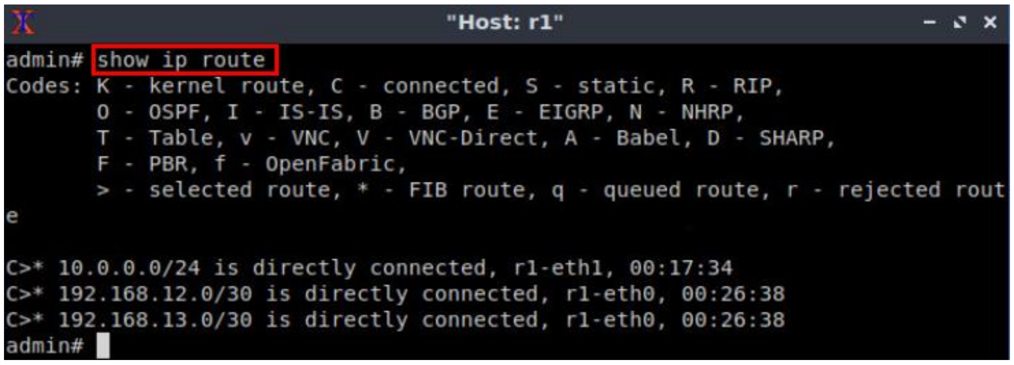


```
"Host: r1"
root@admin:/etc/routers/r1# bgpd
root@admin:/etc/routers/r1# vtysh

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

admin# configure terminal
admin(config)# router bgp 100
admin(config-router)# neighbor 192.168.12.2 remote-as 200
admin(config-router)# neighbor 192.168.13.2 remote-as 300
admin(config-router)# neighbor 10.0.0.3 remote-as 100
admin(config-router)# neighbor 10.0.0.3 port 2000
admin(config-router)# end
```

Figure 57. Exiting from configuration mode.

**Step 10.** Type the following command on router r1 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 of router r2 (192.168.2.0/24) or router r3 (192.168.3.0/24) as there is no enabled ONOS application that deals with BGP routes.

```
show ip route
```



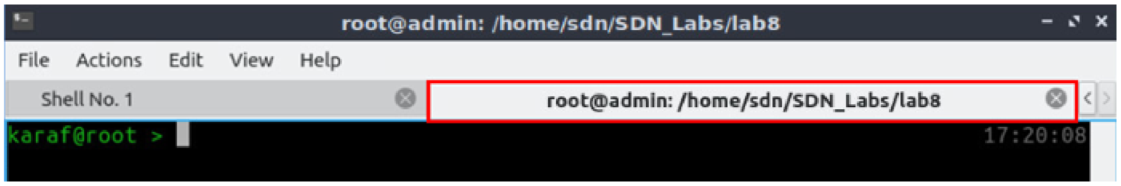
```
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected rout
e

C>* 10.0.0.0/24 is directly connected, r1-eth1, 00:17:34
C>* 192.168.12.0/30 is directly connected, r1-eth0, 00:26:38
C>* 192.168.13.0/30 is directly connected, r1-eth0, 00:26:38
admin#
```

Figure 58. Displaying the routing table of router r1.

## 5.3 Activating the SDN-IP application

In this section, you will activate the SDN-IP application and other dependencies (applications) that will interconnect the SDN network with the legacy network.

**Step 1.** Go to the ONOS terminal.

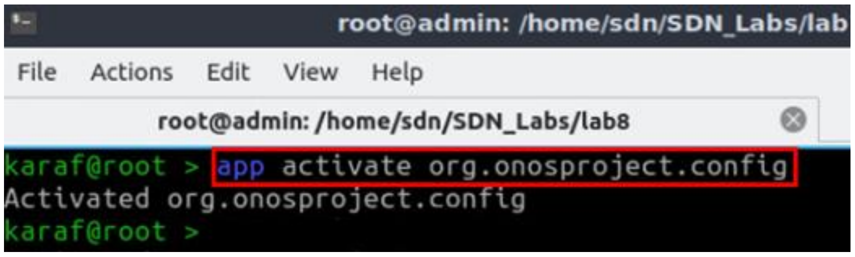

root@admin: /home/sdn/SDN_Labs/lab8
File
Actions
Edit
View
Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab8

```
karaf@root >                                                    17:20:08
```

Figure 59. Opening the ONOS terminal.

**Step 2.** Before activating the SDN-IP application you must start the config application. The latter is an application for the network configuration. In order to activate the config application, type the following command.

```
app activate org.onosproject.config
```

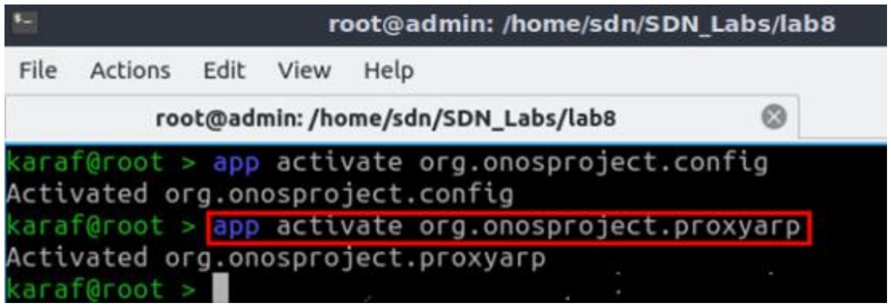

root@admin: /home/sdn/SDN_Labs/lab
File
Actions
Edit
View
Help
root@admin: /home/sdn/SDN_Labs/lab8

```
karaf@root > app activate org.onosproject.config
Activated org.onosproject.config
karaf@root >
```

Figure 60. Activating ONOS config application.

**Step 3.** The SDN-IP application has an additional application dependency, which is used to resolve Address Resolution Protocol (ARP) requests. This is the proxyarp application that responds to ARP requests on behalf of hosts and external routers.

```
app activate org.onosproject.proxyarp
```

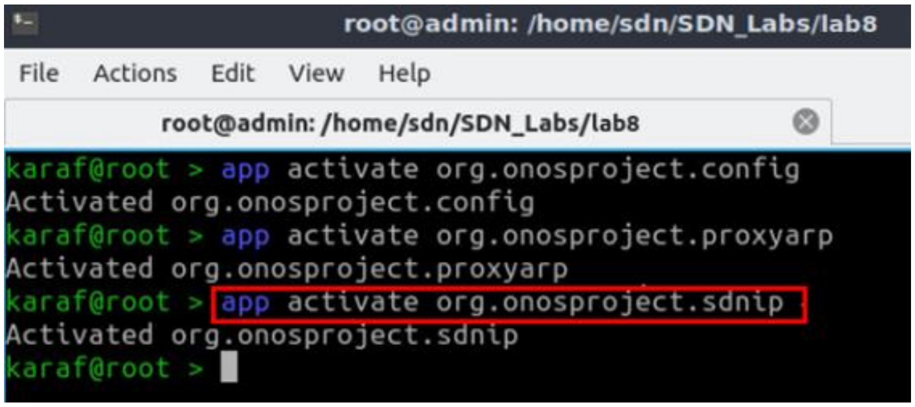

root@admin: /home/sdn/SDN_Labs/lab8
File Actions Edit View Help
root@admin: /home/sdn/SDN_Labs/lab8

```
karaf@root > app activate org.onosproject.config
Activated org.onosproject.config
karaf@root > app activate org.onosproject.proxyarp
Activated org.onosproject.proxyarp
karaf@root >
```

Figure 61. Activating ONOS proxyarp application.

**Step 4.** Once the dependencies are started, the SDN-IP application can be activated. In order to do that, type the following command.

```
app activate org.onosproject.sdnip
```


root@admin: /home/sdn/SDN_Labs/lab8
File Actions Edit View Help
root@admin: /home/sdn/SDN_Labs/lab8

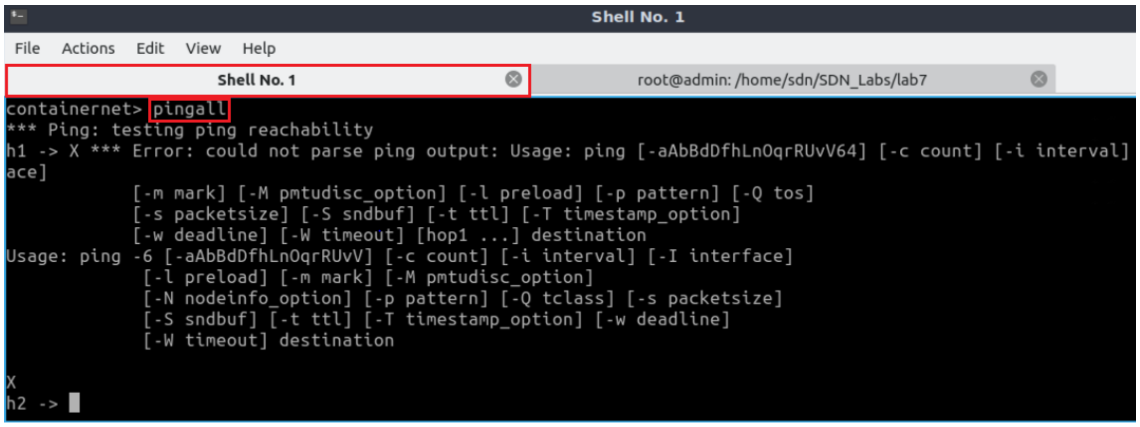
```
karaf@root > app activate org.onosproject.config
Activated org.onosproject.config
karaf@root > app activate org.onosproject.proxyarp
Activated org.onosproject.proxyarp
karaf@root > app activate org.onosproject.sdnip
Activated org.onosproject.sdnip
karaf@root >
```

Figure 62. Activating ONOS SDN-IP application.

After activating the applications above, you might have to wait few minutes until the applications discover the topology and exchange information in order to get correct results.

**Step 5**. Click on the Mininet tab on the left-hand side and type the command shown below to ping all hosts in the topology.

```
pingall
```


Shell No. 1
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab7

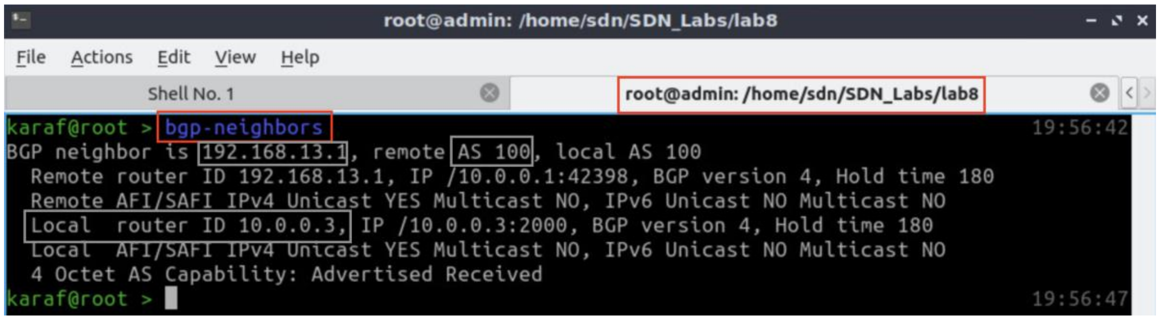
```
containernet> pingall
*** Ping: testing ping reachability
h1 -> X *** Error: could not parse ping output: Usage: ping [-aAbBdDfhLnOqrRUvV64] [-c count] [-i interval]
ace]
            [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
            [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
            [-w deadline] [-W timeout] [hop1 ...] destination
Usage: ping -6 [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
             [-l preload] [-m mark] [-M pmtudisc_option]
             [-N nodeinfo_option] [-p pattern] [-Q tclass] [-s packetsize]
             [-S sndbuf] [-t ttl] [-T timestamp_option] [-w deadline]
             [-W timeout] destination

X
h2 ->
```

Figure 63. Pinging all the hosts.

The test result will be unsuccessful. The purpose of this step is to provide information about the hosts to the controller.

**Step 6.** In the ONOS terminal, type the following command to show the IBGP neighbors that have connected to SDN-IP application.

```
bgp-neighbors
```


root@admin: /home/sdn/SDN_Labs/lab8
File Actions Edit View Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab8

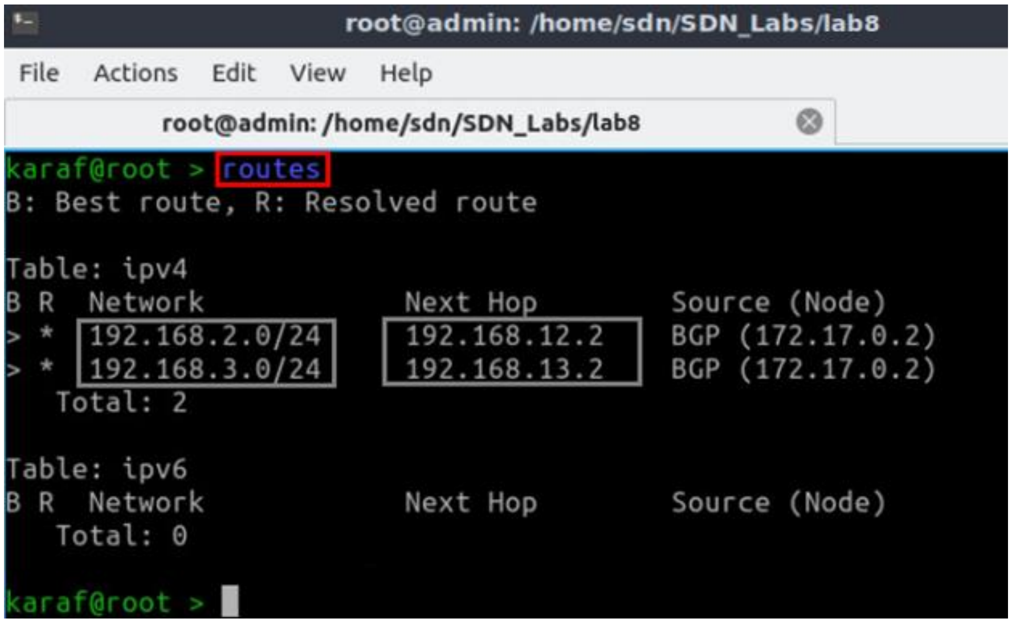
```
karaf@root > bgp-neighbors                                                  19:56:42
BGP neighbor is 192.168.13.1, remote AS 100, local AS 100
  Remote router ID 192.168.13.1, IP /10.0.0.1:42398, BGP version 4, Hold time 180
  Remote AFI/SAFI IPv4 Unicast YES Multicast NO, IPv6 Unicast NO Multicast NO
  Local  router ID 10.0.0.3, IP /10.0.0.3:2000, BGP version 4, Hold time 180
  Local  AFI/SAFI IPv4 Unicast YES Multicast NO, IPv6 Unicast NO Multicast NO
  4 Octet AS Capability: Advertised Received
karaf@root >                                                                19:56:47
```

Figure 64. Viewing IBGP neighbors within the SDN network.

Consider Figure 64. The neighbor 192.168.13.1 corresponds to router r1 in AS 100. This is the internal BGP speaker in the SDN network. The local router ID that the SDN-IP application uses is 10.0.0.3.

**Step 7.** To show the routing table of SDN-IP, type the following command.

```
routes
```



```
root@admin: /home/sdn/SDN_Labs/lab8
File  Actions  Edit  View  Help
root@admin: /home/sdn/SDN_Labs/lab8
karaf@root > routes
B: Best route, R: Resolved route

Table: ipv4
B R  Network             Next Hop         Source (Node)
> *  192.168.2.0/24      192.168.12.2     BGP (172.17.0.2)
> *  192.168.3.0/24      192.168.13.2     BGP (172.17.0.2)
   Total: 2

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

karaf@root >
```

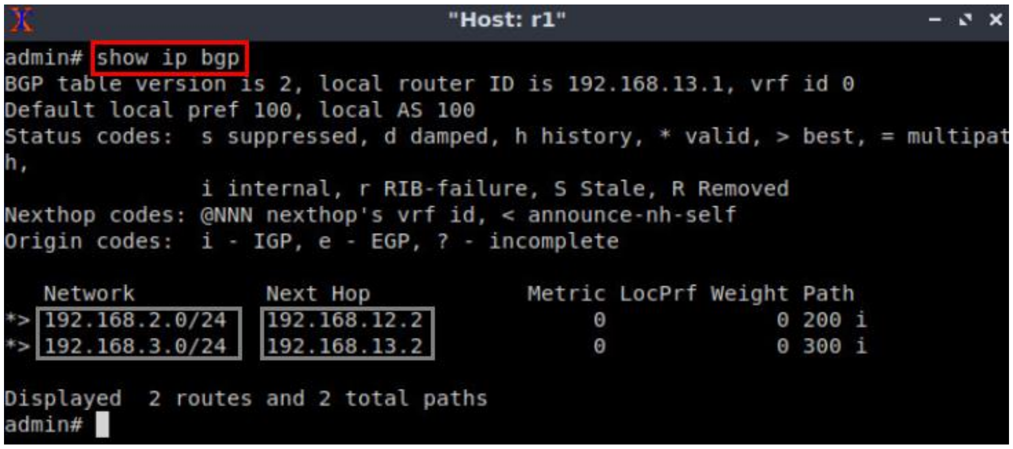
Figure 65. Showing the routing table of the SDN-IP application.

Consider Figure 65. The networks 192.168.2.0/24 and 192.168.3.0/24 are inserted in the routing table of the SDN-IP application.

## 6 Verifying the connectivity between the networks

**Step 1.** Open router r1 terminal and type the following command to show the BGP table.

```
show ip bgp
```



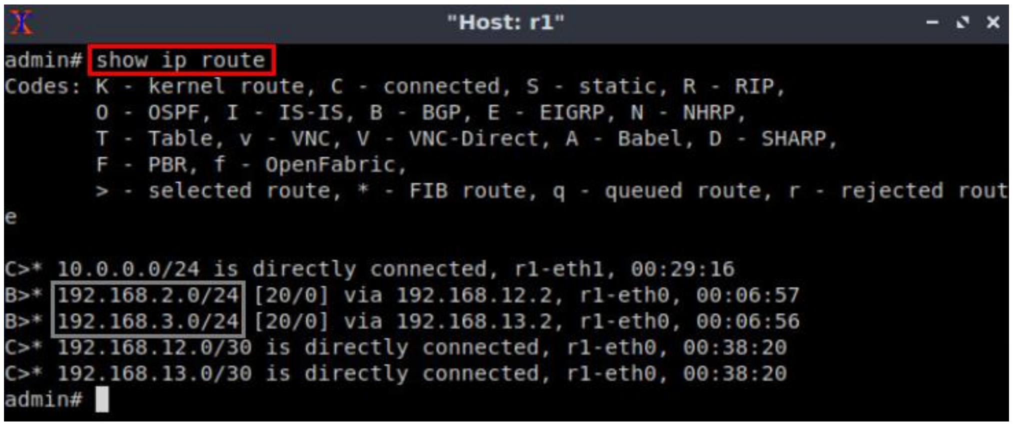
```
"Host: r1"
admin# show ip bgp
BGP table version is 2, local router ID is 192.168.13.1, vrf id 0
Default local pref 100, local AS 100
Status codes:  s suppressed, d damped, h history, * valid, > best, = multipat
h,
               i internal, r RIB-failure, S Stale, R Removed
Nexthop codes: @NNN nexthop's vrf id, < announce-nh-self
Origin codes:  i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 192.168.2.0/24   192.168.12.2             0             0 200 i
*> 192.168.3.0/24   192.168.13.2             0             0 300 i

Displayed  2 routes and 2 total paths
admin#
```

Figure 66. Showing the BGP table of router r1.

Consider Figure 66. The networks 192.168.2.0/24 and 192.168.3.0/24 are inserted in the BGP table of router r1. The next hops to reach these networks are 192.168.12.2 (router r2) and 192.168.13.2 (router r3), respectively.

**Step 2.** In router r1 terminal, type the following command to show the routing table.

```
show ip route
```



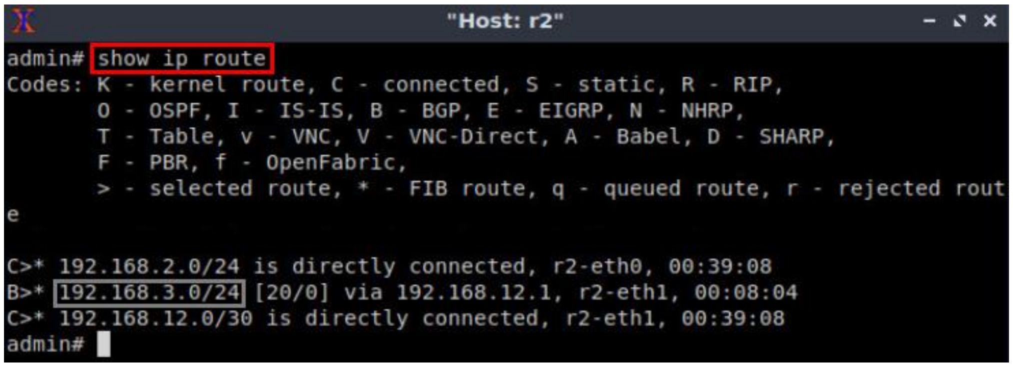
```
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected rout
e

C>* 10.0.0.0/24 is directly connected, r1-eth1, 00:29:16
B>* 192.168.2.0/24 [20/0] via 192.168.12.2, r1-eth0, 00:06:57
B>* 192.168.3.0/24 [20/0] via 192.168.13.2, r1-eth0, 00:06:56
C>* 192.168.12.0/30 is directly connected, r1-eth0, 00:38:20
C>* 192.168.13.0/30 is directly connected, r1-eth0, 00:38:20
admin#
```

Figure 67. Showing the routing table of router r1.

Consider Figure 67. The networks 192.168.2.0/24 and 192.168.3.0/24 advertised by routers r2 and r3, respectively, are added to the routing table of router r1.

**Step 3.** Open router r2 terminal and type the following command to show the routing table.

```
show ip route
```



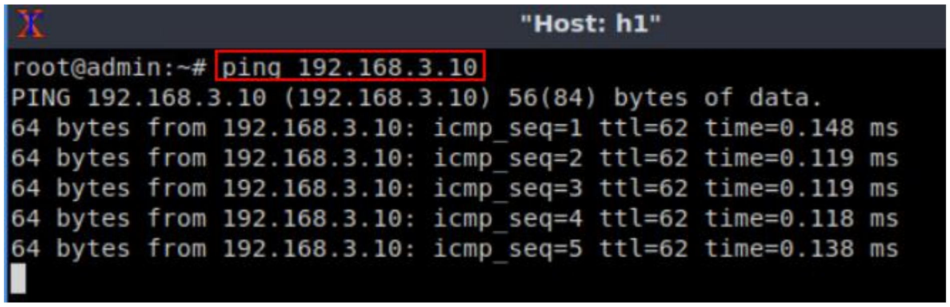
```
admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued route, r - rejected rout
e

C>* 192.168.2.0/24 is directly connected, r2-eth0, 00:39:08
B>* 192.168.3.0/24 [20/0] via 192.168.12.1, r2-eth1, 00:08:04
C>* 192.168.12.0/30 is directly connected, r2-eth1, 00:39:08
admin#
```

Figure 68. Showing the routing table router r2.

Consider Figure 68. The network 192.168.3.0/24 is added to the routing table of router r2, and it is reachable via 192.168.12.1 (router r1). Similarly, you can verify the routing table of router r3.

**Step 4.** Open host h1 terminal and type the following command to test the connectivity with host h2.

```
ping 192.168.3.10
```



```
"Host: h1"
root@admin:~# ping 192.168.3.10
PING 192.168.3.10 (192.168.3.10) 56(84) bytes of data.
64 bytes from 192.168.3.10: icmp_seq=1 ttl=62 time=0.148 ms
64 bytes from 192.168.3.10: icmp_seq=2 ttl=62 time=0.119 ms
64 bytes from 192.168.3.10: icmp_seq=3 ttl=62 time=0.119 ms
64 bytes from 192.168.3.10: icmp_seq=4 ttl=62 time=0.118 ms
64 bytes from 192.168.3.10: icmp_seq=5 ttl=62 time=0.138 ms
```

Figure 69. Pinging host h2 from host h1.

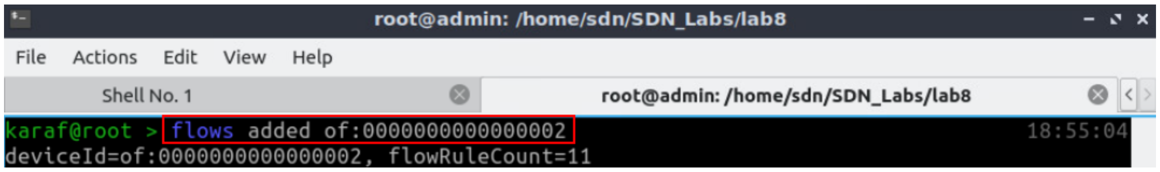
Consider Figure 69. The result shows a successful connectivity test. Thus, BGP is successfully configured and integrated between legacy and SDN networks. Do not stop the test as you will inspect the flow entries that handle the traffic between ASes 200 and 300.

## 7 Inspecting the flow table of the OpenFlow switches

In this section, you will inspect the flow rules installed on the OpenFlow switches to better understand the SDN-IP application and see how the BGP route advertisements are translated into OpenFlow entries.

**Step 1.** On the ONOS terminal, type the following command to inspect the flows in switch s2. Use the `tab` key to autocomplete the OpenFlow port.

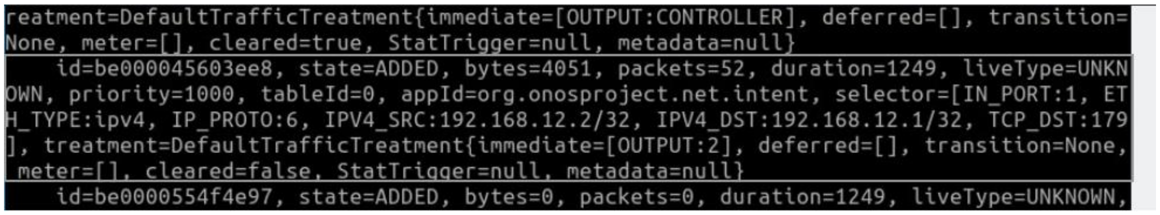
```
flows added of:0000000000000002
```



root@admin: /home/sdn/SDN_Labs/lab8
File
Actions
Edit
View
Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab8

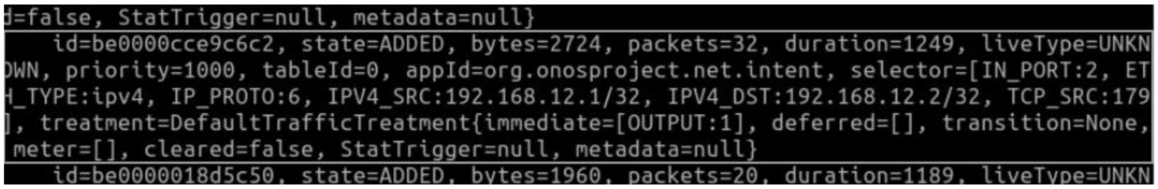
```
karaf@root > flows added of:0000000000000002                        18:55:04
deviceId=of:0000000000000002, flowRuleCount=11
```

Figure 70. Inspecting flow entries on switch s2.



```
reatment=DefaultTrafficTreatment{immediate=[OUTPUT:CONTROLLER], deferred=[], transition=
None, meter=[], cleared=true, StatTrigger=null, metadata=null}
    id=be000045603ee8, state=ADDED, bytes=4051, packets=52, duration=1249, liveType=UNKN
OWN, priority=1000, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:1, ET
H_TYPE:ipv4, IP_PROTO:6, IPV4_SRC:192.168.12.2/32, IPV4_DST:192.168.12.1/32, TCP_DST:179
], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:2], deferred=[], transition=None,
 meter=[], cleared=false, StatTrigger=null, metadata=null}
    id=be0000554f4e97, state=ADDED, bytes=0, packets=0, duration=1249, liveType=UNKNOWN,
```

Figure 71. Inspecting flow entries on switch s2 that handle BGP traffic.



```
d=false, StatTrigger=null, metadata=null}
    id=be0000cce9c6c2, state=ADDED, bytes=2724, packets=32, duration=1249, liveType=UNKN
OWN, priority=1000, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:2, ET
H_TYPE:ipv4, IP_PROTO:6, IPV4_SRC:192.168.12.1/32, IPV4_DST:192.168.12.2/32, TCP_SRC:179
], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:1], deferred=[], transition=None,
 meter=[], cleared=false, StatTrigger=null, metadata=null}
    id=be0000018d5c50, state=ADDED, bytes=1960, packets=20, duration=1189, liveType=UNKN
```

Figure 72. Inspecting flow entries on switch s2 that handle BGP traffic.

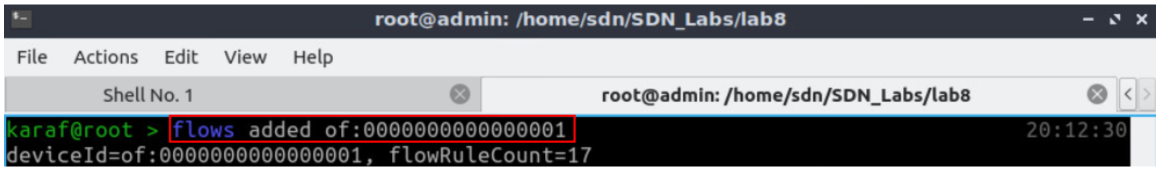
Consider Figures 70, 71, and 72. There are 11 flow rules installed on switch s2. For simplicity, we only show some of the rules that handle BGP traffic. The highlighted rule in figure 71 handles BGP traffic from router r2 to router r1 on switch s2. The match criteria of the rule are as follow:

- `IN_PORT:1` Incoming packets at port 1 (s2-eth1).
- `ETH_TYPE:ipv4` IPv4 packets.
- `IP_PROTO:6` TCP packets.
- `IPV4_SRC: 192.168.12.2/32` The IP address of router r2 (r2-eth1).
- `IPV4_DST: 192.168.12.1/32` The IP address of router r1 (r1-eth0).
- `TCP_SRC: 179` TCP source port where BGP is running.

Packets matching the criteria above will be forwarded out of port 2 (s2-eth2). Likewise, the highlighted rule in figure 72 handles BGP traffic from router r1 to router r2.

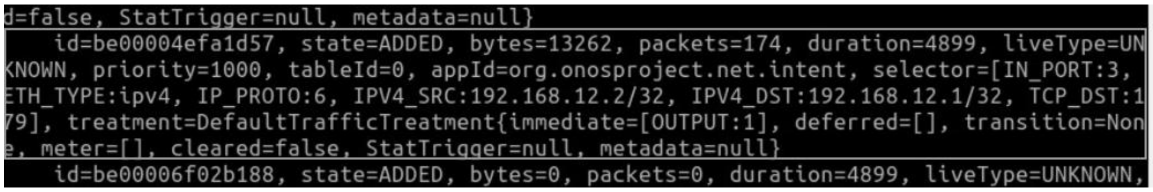
**Step 2.** On the ONOS terminal, type the following command to inspect the flows in switch s1. Use the `tab` key to autocomplete the OpenFlow port.

```
flows added of:0000000000000001
```


root@admin: /home/sdn/SDN_Labs/lab8
File
Actions
Edit
View
Help
Shell No. 1
root@admin: /home/sdn/SDN_Labs/lab8

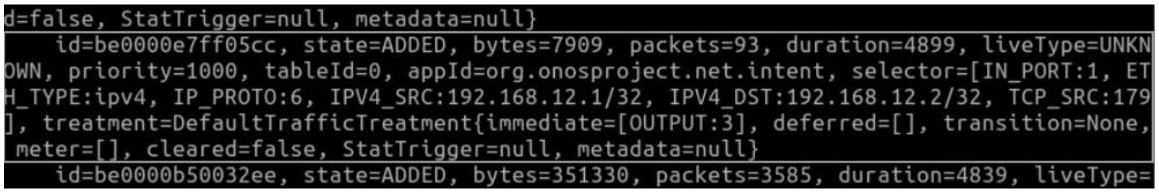
```
karaf@root > flows added of:0000000000000001                          20:12:30
deviceId=of:0000000000000001, flowRuleCount=17
```

Figure 73. Inspecting flow entries on switch s1.



```
d=false, StatTrigger=null, metadata=null}
    id=be00004efa1d57, state=ADDED, bytes=13262, packets=174, duration=4899, liveType=UN
KNOWN, priority=1000, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:3,
ETH_TYPE:ipv4, IP_PROTO:6, IPV4_SRC:192.168.12.2/32, IPV4_DST:192.168.12.1/32, TCP_DST:1
79], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:1], deferred=[], transition=Non
e, meter=[], cleared=false, StatTrigger=null, metadata=null}
    id=be00006f02b188, state=ADDED, bytes=0, packets=0, duration=4899, liveType=UNKNOWN,
```

Figure 74. Inspecting flow entries on switch s1 that handle BGP traffic.



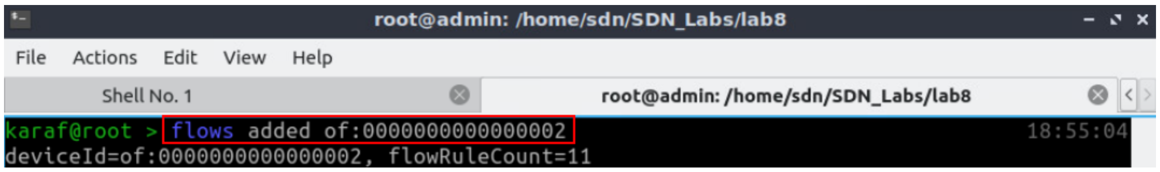
```
d=false, StatTrigger=null, metadata=null}
    id=be0000e7ff05cc, state=ADDED, bytes=7909, packets=93, duration=4899, liveType=UNKN
OWN, priority=1000, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:1, ET
H_TYPE:ipv4, IP_PROTO:6, IPV4_SRC:192.168.12.1/32, IPV4_DST:192.168.12.2/32, TCP_SRC:179
], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:3], deferred=[], transition=None,
 meter=[], cleared=false, StatTrigger=null, metadata=null}
    id=be0000b50032ee, state=ADDED, bytes=351330, packets=3585, duration=4839, liveType=
```

Figure 75. Inspecting flow entries on switch s1 that handle BGP traffic.

Consider Figures 73, 74, and 75. There are 17 flow rules installed on switch s1. For simplicity, we only show some of the rules that handle BGP traffic. The highlighted rule in figure 74 handles BGP traffic from router r2 to router r1 on switch s1. Likewise, the highlighted rule in figure 75 handles BGP traffic from router r1 to router r2 on switch s1. Similarly, you can inspect other flows on switches s1 and s3 that deal with the BGP traffic between routers r1 and r3.

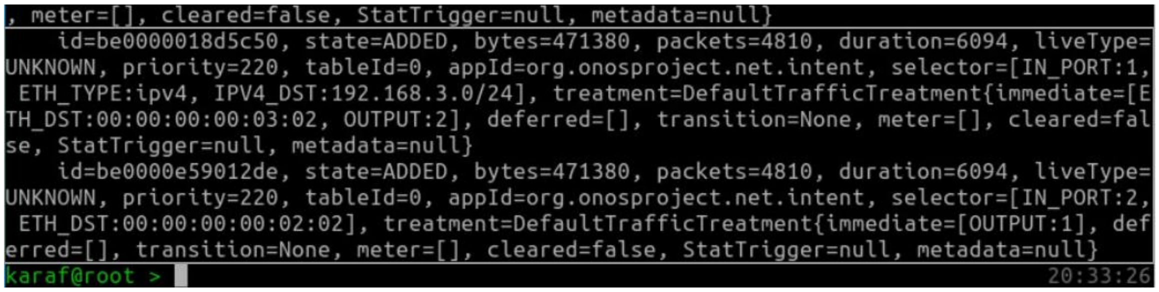
**Step 3.** On the ONOS terminal, type the following command to inspect the flows in switch s2. Use the `tab` key to autocomplete the OpenFlow port.

```
flows added of:0000000000000002
```



```
karaf@root > flows added of:0000000000000002                                    18:55:04
deviceId=of:0000000000000002, flowRuleCount=11
```

Figure 76. Inspecting flow entries on switch s2.



```
, meter=[], cleared=false, StatTrigger=null, metadata=null}
    id=be0000018d5c50, state=ADDED, bytes=471380, packets=4810, duration=6094, liveType=
UNKNOWN, priority=220, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:1,
 ETH_TYPE:ipv4, IPV4_DST:192.168.3.0/24], treatment=DefaultTrafficTreatment{immediate=[E
TH_DST:00:00:00:00:03:02, OUTPUT:2], deferred=[], transition=None, meter=[], cleared=fal
se, StatTrigger=null, metadata=null}
    id=be0000e59012de, state=ADDED, bytes=471380, packets=4810, duration=6094, liveType=
UNKNOWN, priority=220, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:2,
 ETH_DST:00:00:00:00:02:02], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:1], def
erred=[], transition=None, meter=[], cleared=false, StatTrigger=null, metadata=null}
karaf@root >                                                                    20:33:26
```

Figure 77. Inspecting flow entries on switch s2 that handle traffic between ASes 200 and 300.

Consider Figure 77. Switch s2 has two flow rules that handle the traffic between ASes 200 and 300. The first flow rule matches against IPv4 packets destined to AS 300 (192.168.3.0/24). The corresponding actions are changing the MAC destination address to interface r3-eth1 (`ETH_DST:00:00:00:00:03:02`), followed by forwarding the packets out of port 1 (`OUTPUT:2`), which corresponds to s2-eth2. The second flow matches against packets destined to router r2 (`ETH_DST:00:00:00:00:02:02`) and forwards them out of port 1 (s2-eth1).

This concludes Lab 8. Stop the emulation and then exit out of MiniEdit and Linux terminal.

## References

1. A. Tanenbaum, D. Wetherall, “*Computer networks*”, 5th Edition, Pearson, 2012.
2. Y. Rekhter, T. Li, and S. Hares. "*A border gateway protocol 4 (BGP-4)*." RFC 4271 (1994).
3. ONOS project, “*SDN-IP*”, [Online]. Available: https://wiki.onosproject.org/display/ONOS/SDN-IP.
4. P. Goransson, C. Black, T. Culver. “*Software defined networks: a comprehensive approach*”. Morgan Kaufmann, 2016.
5. P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.C. Wang, J. Bi. “*Seamless interworking of SDN and IP*”. In Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, pp. 475-476, 2013.

UNIVERSITY OF
SOUTH CAROLINA

# SOFTWARE DEFINED NETWORKING

## Exercise 4: Incremental Deployment of SDN Networks within Legacy Networks

**Document Version: 09-02-2021**

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

## Contents

## 1 Exercise description

Consider Figure 1. The topology consists of two IP networks in Autonomous Systems (ASes) 20 and 30 and one Software Defined Networking (SDN) network in AS 10. The IP networks connect to the SDN network through their Border Gateway Protocol (BGP) routers. Router r1 is a BGP router within the SDN network that communicates with the External BGP (EBGP) routers r2 and r3 via the networks 173.17.12.0/30 and 173.17.13.0/30, respectively. Furthermore, router r1 connects to the ONOS controller via the network 10.0.0.0/24 to propagate the BGP advertisements received from other networks. The SDN network contains two hosts that are in networks 192.168.1.0/24 and 192.168.2.0/24, respectively, and should communicate after configuring the SDN control and data planes.

The goal of this exercise is to configure the SDN switches to interconnect with the legacy networks, as well as to emulate virtual gateways and routing within the SDN network. To interconnect the three ASes, you should configure BGP on the legacy routers and enable the necessary ONOS applications within the SDN network. To emulate virtual gateways and connect hosts h3 and h4, you should reconfigure ONOS with a new network configuration file and enable an additional ONOS application. The topology below is already built and you should use Mininet to emulate it. Additionally, you should use Free Range Routing (FRR) to configure network protocols.

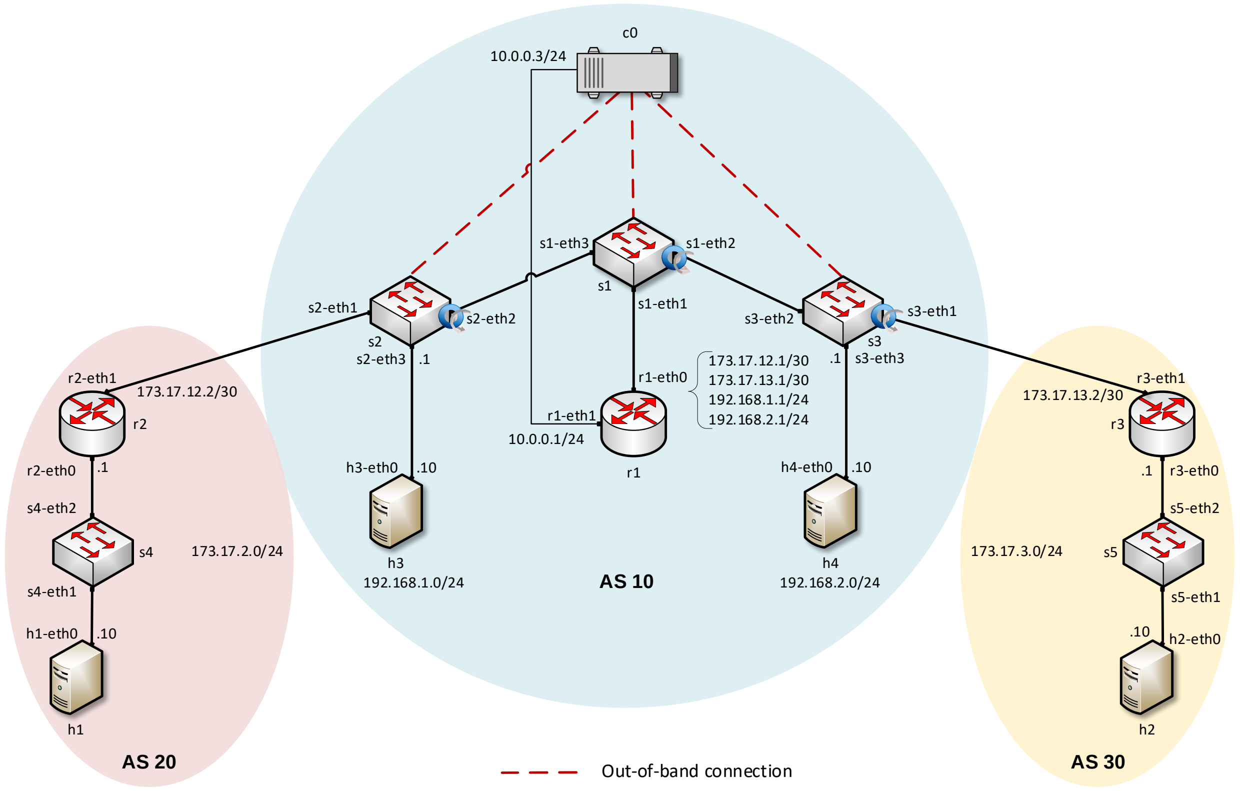


Figure 1. Lab topology.

### 1.1 Topology settings

The devices are already configured according to Table 1.

Table 1. Topology information.

| Device | Interface | MAC Address | IP Address | Subnet | Default gateway |
|---|---|---|---|---|---|
| Router r1 | r1-eth0 | 00:00:00:00:01:01 | 173.17.12.1 | /30 | N/A |
| | | 00:00:00:00:01:01 | 173.17.13.1 | /30 | N/A |
| | | 00:00:00:00:01:01 | 192.168.1.1 | /24 | N/A |
| | | 00:00:00:00:01:01 | 192.168.2.1 | /24 | N/A |
| | r1-eth1 | 00:00:00:00:01:02 | 10.0.0.1 | /24 | N/A |
| Router r2 | r2-eth0 | 00:00:00:00:02:01 | 173.17.2.1 | /24 | N/A |
| | r2-eth1 | 00:00:00:00:02:02 | 173.17.12.2 | /30 | N/A |
| Router r3 | r3-eth0 | 00:00:00:00:03:01 | 173.17.3.1 | /24 | N/A |
| | r3-eth1 | 00:00:00:00:03:02 | 173.17.13.2 | /30 | N/A |
| Switch s2 | s2-eth3 | N/A | 192.168.1.1 | /24 | N/A |
| Switch s3 | s3-eth3 | N/A | 192.168.2.1 | /24 | N/A |
| Host h1 | h1-eth0 | 00:00:00:00:00:01 | 173.17.2.10 | /24 | 173.17.2.1 |
| Host h2 | h2-eth0 | 00:00:00:00:00:02 | 173.17.3.10 | /24 | 173.17.3.1 |
| Host h3 | h3-eth0 | 00:00:00:00:00:03 | 192.168.1.10 | /24 | 192.168.1.1 |
| Host h4 | h4-eth0 | 00:00:00:00:00:04 | 192.168.2.10 | /24 | 192.168.2.1 |
| c0 | N/A | N/A | 172.17.0.2 | /16 | N/A |
| | N/A | N/A | 10.0.0.3 | /24 | N/A |

## 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)** Open MiniEdit and load the topology above. The topology file *Exercise4.mn* of this exercise is in the directory *~/SDN_Labs/Exercise4* as shown in the figure below. Do not run MiniEdit.

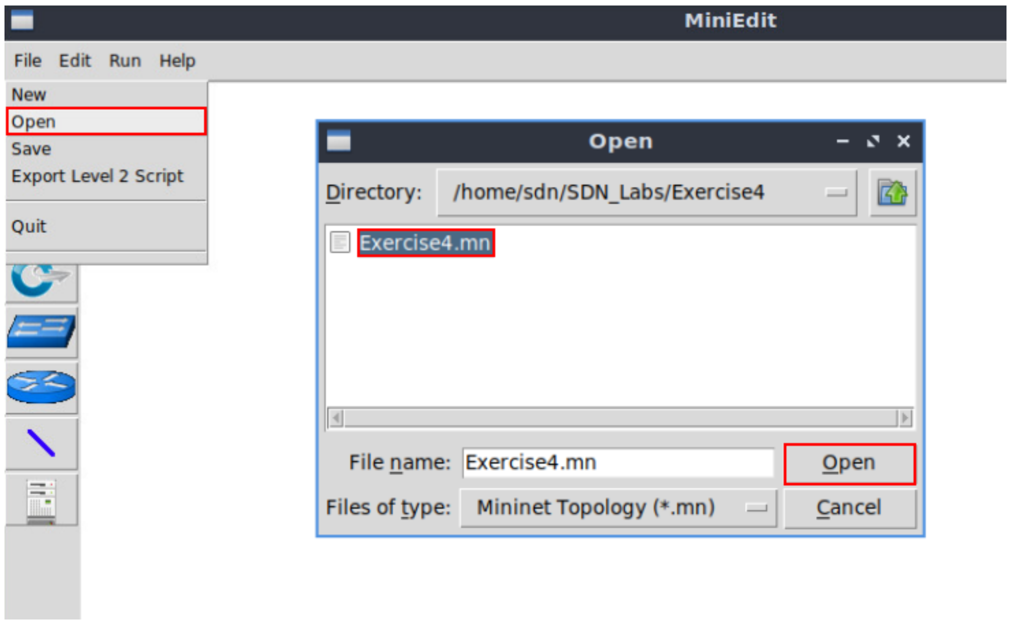
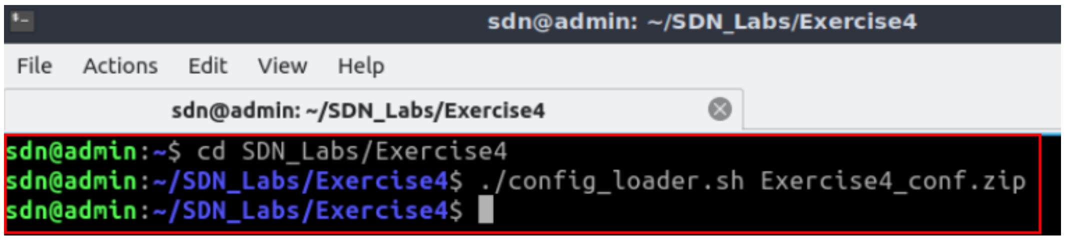


Figure 2. Loading the topology file in Mininet.

**b)** In Linux terminal, run the script responsible for loading the IP addresses to the interfaces of the routers as shown in the figure below. The script *config_loader.sh* takes the IP addresses file *Exercise4_conf.zip* as an argument, and both files are in the directory *~/SDN_Labs/Exercise4.*


sdn@admin: ~/SDN_Labs/Exercise4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/Exercise4

```
sdn@admin:~$ cd SDN_Labs/Exercise4
sdn@admin:~/SDN_Labs/Exercise4$ ./config_loader.sh Exercise4_conf.zip
sdn@admin:~/SDN_Labs/Exercise4$ 
```

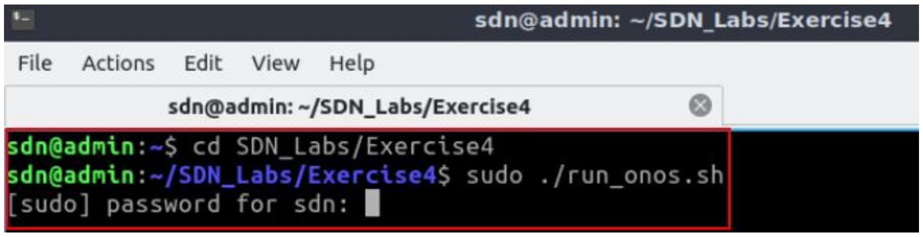
Figure 3. Executing the shell script to load the configuration.

**c)** Run MiniEdit and verify in Mininet terminal that the links conform to the topology figure and settings table above.

**d)** In FRR shell, inspect the routing table and verify the directly connected networks on each router.

**e)** Configure BGP on routers r2 and r3 within the legacy networks. Routers r2 and r3 should peer with router r1 and advertise their local network. Inspect the BGP tables of routers r2 and r3 and report any learned routes. Explain the results.

**f)** In the Linux terminal, navigate to the directory *~/SDN_Labs/Exercise4* and execute, in superuser mode, the script *run_onos.sh* that runs the ONOS controller. When prompted for a password, type `password`. The steps to run ONOS are depicted in the figure below.

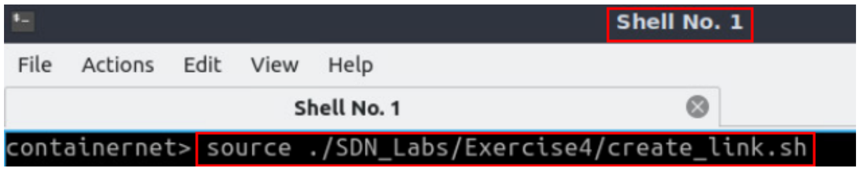

sdn@admin: ~/SDN_Labs/Exercise4
File
Actions
Edit
View
Help
sdn@admin: ~/SDN_Labs/Exercise4

```
sdn@admin:~$ cd SDN_Labs/Exercise4
sdn@admin:~/SDN_Labs/Exercise4$ sudo ./run_onos.sh
[sudo] password for sdn: 
```

Figure 4. Starting the ONOS controller.

**g)** Activate the application that allows the communication between the controller and the OpenFlow switches. Additionally, inspect and list the devices and links observed by ONOS. Verify that the results conform with the topology figure and settings above.

**h)** Create a point-to-point network between the IBGP speaker (router r1) and ONOS. To do that, you should run the script *create_link.sh* located in *~/SDN_Labs/Exercise4/* from within the Mininet terminal as shown in the figure below.



```
containernet> source ./SDN_Labs/Exercise4/create_link.sh
```

Figure 5. Creating a point-to-point network (link) between the IBGP speaker and the ONOS controller.

**i)** Verify the added network by inspecting the interfaces of router r1. Report the added interface along with its IP address.

**j)** Configure BGP on router r1 to peer with the legacy networks and ONOS.

**k)** Activate the ONOS applications responsible for interconnecting the legacy and SDN networks.

**l)** Issue the ONOS command to view the IBGP neighbors of the SDN networks. Validate that the result obtained conforms with the topology above.

**m)** Issue the command to view the routing table of the ONOS application interconnecting the SDN and legacy networks.

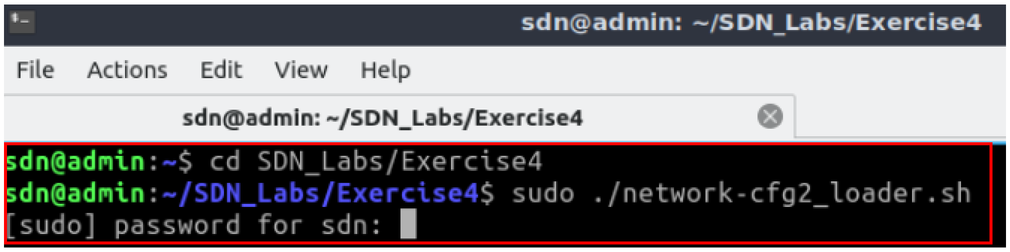
**n)** Inspect the BGP tables of the legacy routers and report the newly learned routes. Compare the results obtained to those in part e and explain the results briefly.

**o)** Test the connectivity between the legacy networks by performing a ping test from host h1 to host h2. Keep the ping test active for the following step.

**p)** List the flow that forwards the BGP traffic from router r2 to router r1 on switch s2 and the flow that forwards the IP packets destined to host h1 on switch s3.

**q)** Test the connectivity between the SDN hosts by performing a ping test from host h3 to host h4.

**r)** ONOS is loaded with a configuration file *network-cfg1.json* necessary for the application that enables the interconnection between the legacy and SDN networks. To enable routing within the SDN network, a new configuration file *network-cfg2_loader.sh* must be loaded into ONOS. The file *network-cfg2_loader.sh* is necessary for the application that enables routing within the SDN network. To load the *network-cfg2_loader.sh* file into ONOS, you should run the script *network-cfg2_loader.sh* located in *~/SDN_Labs/Exercise4/* from within the Linux terminal in superuser mode as shown in the figure below.


sdn@admin: ~/SDN_Labs/Exercise4
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/Exercise4

```
sdn@admin:~$ cd SDN_Labs/Exercise4
sdn@admin:~/SDN_Labs/Exercise4$ sudo ./network-cfg2_loader.sh
[sudo] password for sdn:
```

Figure 6. Loading a new network configuration file into ONOS to enable routing.

**s)** Activate the ONOS application that enables routing within the SDN network and connects hosts h3 and h4.

**t)** Test the connectivity between the SDN hosts by performing a ping test from host h3 to host h4. Report the result of the connectivity test.

**u)** List the flow that forwards the IP packets destined to host h4 on switch s2.

**v)** Inspect the two configuration files *network-cfg1.json* and *network-cfg2.json* and check the JSON objects that were added to enable routing in the SDN network.

# SOFTWARE DEFINED NETWORKING

# Lab 9: Configuring Virtual Private LAN Service (VPLS)

**Document Version: 07-03-2021**

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

## Contents

## Overview

The following lab presents Virtual Private Local Area Network Service (VPLS). A VPLS emulates a Local Area Network (LAN) and provides multipoint broadcast over layer 2 circuits between multiple endpoints. This service enables remote sites to share an Ethernet broadcast domain by connecting sites through pseudowires. In this lab, you will configure VPLS in a simple topology using Open Flow switches.

## Objectives

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

1. Understand the operation of VPLS.
2. Enable OpenFlow switches to enable VPLS operation.
3. Use the Open Network Operating System (ONOS) controller to perform VPLS configuration.
4. Verify end-to-end connectivity between the end-hosts attached to the same VPLS and inspect the flow table of the switches.

## Lab settings

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

Table 1**.** Credentials to access the 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 VPLS.
4. Section 4: Verifying connectivity between end-hosts.

## 1 Introduction

Networks that use the data link layer to interconnect devices are referred to as layer 2 networks. A data link layer device operates in the second layer of the Open Systems Interconnection (OSI) model. For example, a typical layer 2 network is an Ethernet network that englobes endpoint devices such as servers, printers, and computers interconnected using one or more Ethernet switches. Ethernet switches enable layer 2 communication by forwarding Ethernet frames within the network.

VPLS works as a layer 2 Virtual Private Network (VPN) service that extends two or more remote customer networks known as VPLS sites. This service can be provided over intermediate networks often referred to as a provider network. A VPLS is delivered transparently, which implies that the remote customer sites seem to be connected in the same Local Area Network (LAN). VPLS enables layer 2 communications between customer networks through intermediate networks[1].

## 1.1 VPLS architecture

Consider Figure 1. A VPLS emulates a LAN and provides layer 2 functionalities by acting as an emulated Ethernet switch within a Wide Area Network (WAN). Once a VPLS instance is created, its primary function is to interconnect two or more remote customer sites. Additionally, VPLS supports Virtual Local Area Networks (VLAN) and Multiprotocol Label Switching (MPLS) encapsulations. When a VPLS instance is configured, it creates a full mesh of pseudowires between the Provider's Edge (PE) routers participating in the VPLS instance. PE routers can replicate and forward broadcast and multicast frames. Therefore, the emulated switch has all the characteristics of a layer 2 switch[2].

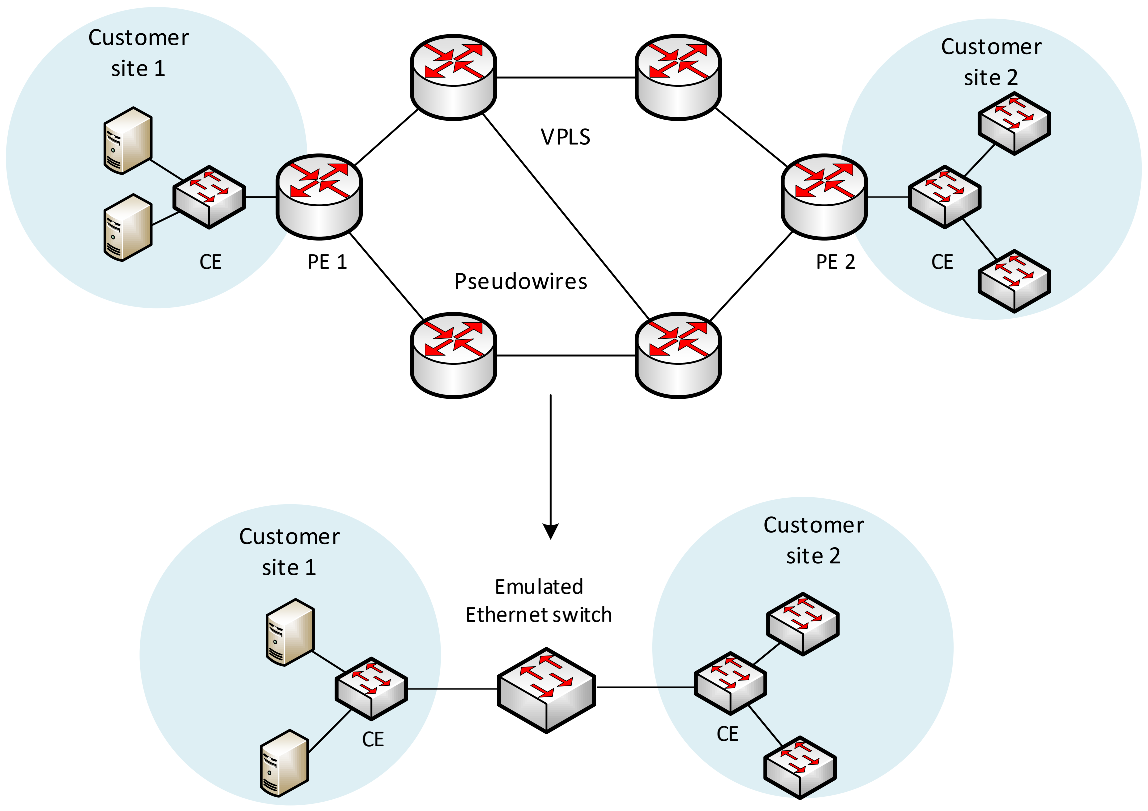


Figure 1. VPLS architecture.

VPLS is implemented as an ONOS application that provides multi-point broadcast layer 2 circuits between multiple endpoints in an OpenFlow network. Additionally, it supports encapsulations such as VLAN and MPLS. To establish VPLS connectivity between two or more end-hosts, they must fulfill the following conditions:

- At least one VPLS must be defined.
- At least the interfaces of two end-host must be configured.
- At least two interfaces must be associated with the same VPLS.

Once the conditions above are satisfied, end-hosts attached to the configured VPLS will send and receive broadcast traffic such as the Address Resolution Protocol (ARP) request messages. This is needed to make sure that all hosts are discovered before establishing unicast communication.

## 2 Lab topology

Consider Figure 2. The topology consists of four end-hosts, two OpenFlow switches, and a controller. This topology presents a scenario where two customers have two remote sites. Hosts h1 and h3 belong to VPLS1, and hosts h2 and h4 belong to VPLS2.

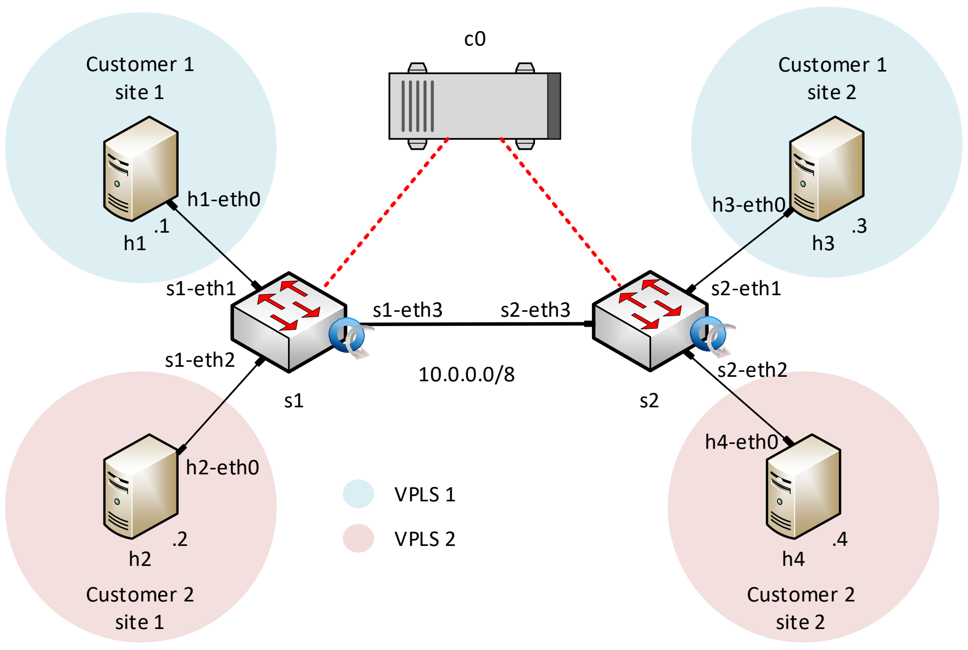


Figure 2. Lab topology.

### 2.1 Lab settings

The devices are already configured according to Table 2.

Table 2**.** Topology information.

| Device | Interface | MAC Address | IP Address | Subnet |
|---|---|---|---|---|
| h1 | h1-eth0 | 00:00:00:00:00:01 | 10.0.0.1 | /8 |
| h2 | h2-eth0 | 00:00:00:00:00:02 | 10.0.0.2 | /8 |
| h3 | h3-eth0 | 00:00:00:00:00:03 | 10.0.0.3 | /8 |
| h4 | h4-eth0 | 00:00:00:00:00:04 | 10.0.0.4 | /8 |
| c0 | N/A | N/A | 172.17.0.2 | /16 |

## 2.2 Loading the 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 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 machine's desktop. Start MiniEdit by clicking on the MiniEdit's shortcut. When prompted for a password, type `password`.

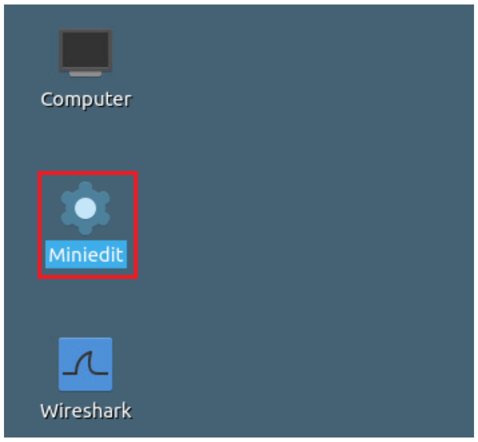


Figure 3. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then *open* to load the lab's topology. Open the *Lab9.mn* topology file stored in the default directory, */home/sdn/SDN_Labs /lab9* and click on *Open*.

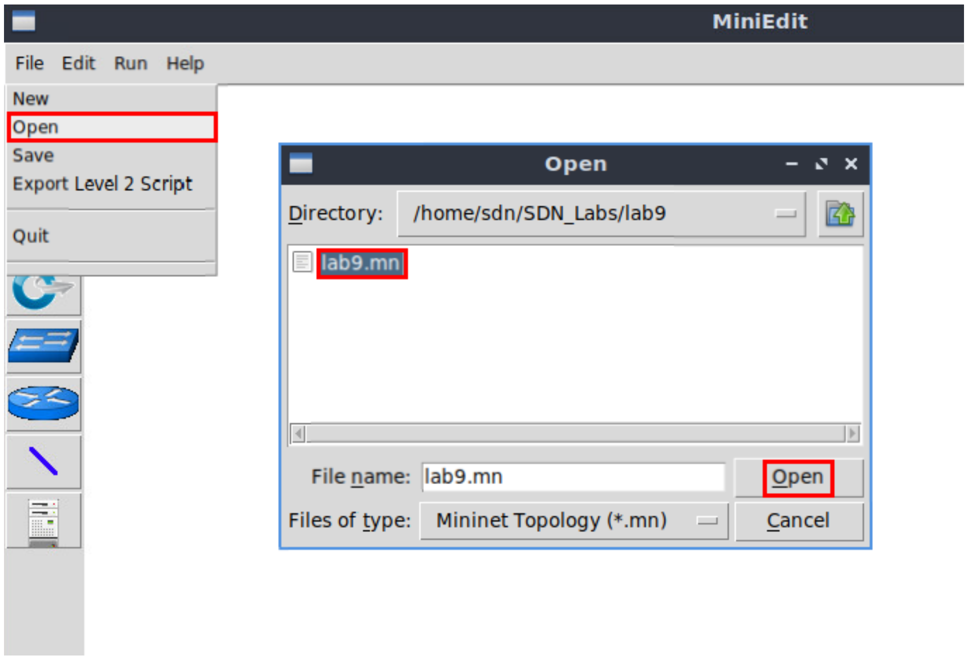


Figure 4. Opening the topology.

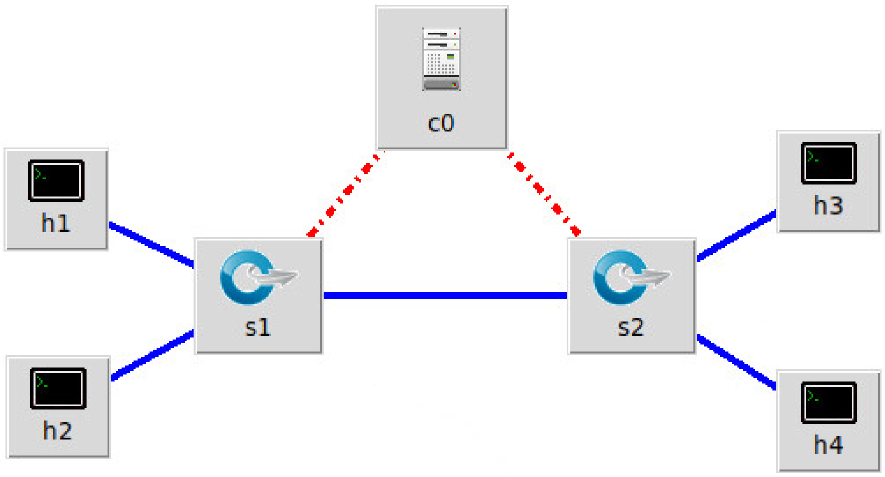


Figure 5. MiniEdit's topology.

**Step 3.** 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.

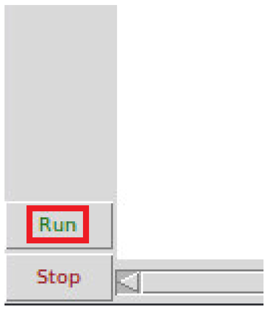


Figure 6. Starting the emulation.

## 2.3 Starting the ONOS controller

**Step 1.** Click on the icon below to open the Linux terminal.

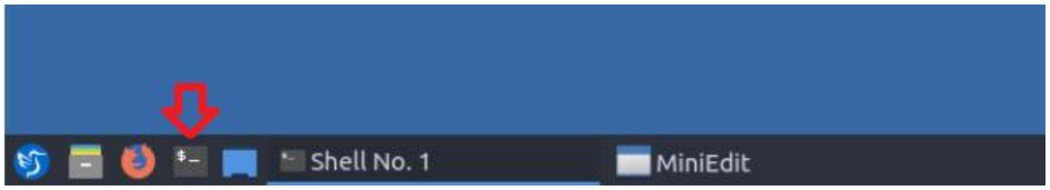
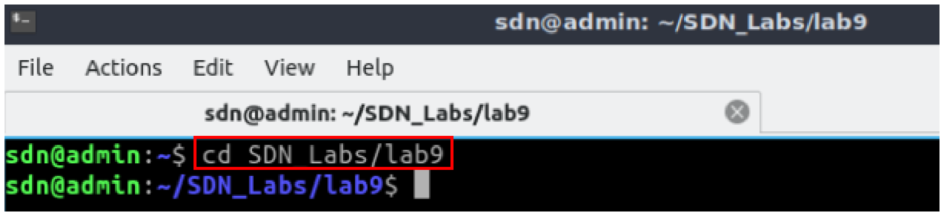


Figure 7. Opening the Linux terminal.

**Step 2.** Navigate into the *SDN_Labs/lab9* directory by issuing the following command. This folder contains the script responsible for starting ONOS. The `cd` command is short for change directory followed by an argument that specifies the destination directory.

```
cd SDN_Labs/lab9
```

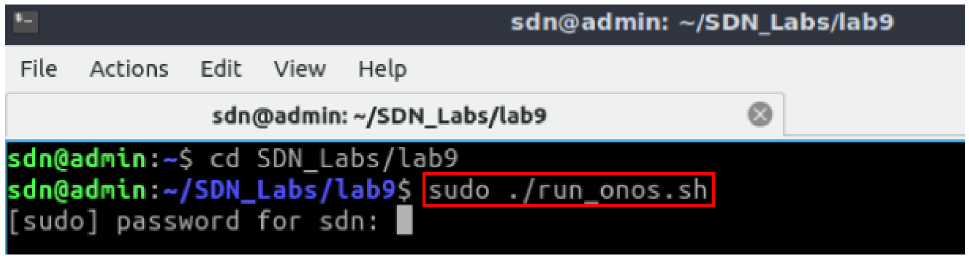


```
sdn@admin:~$ cd SDN_Labs/lab9
sdn@admin:~/SDN_Labs/lab9$ 
```

Figure 8. Entering the *SDN_Labs/lab9* directory.

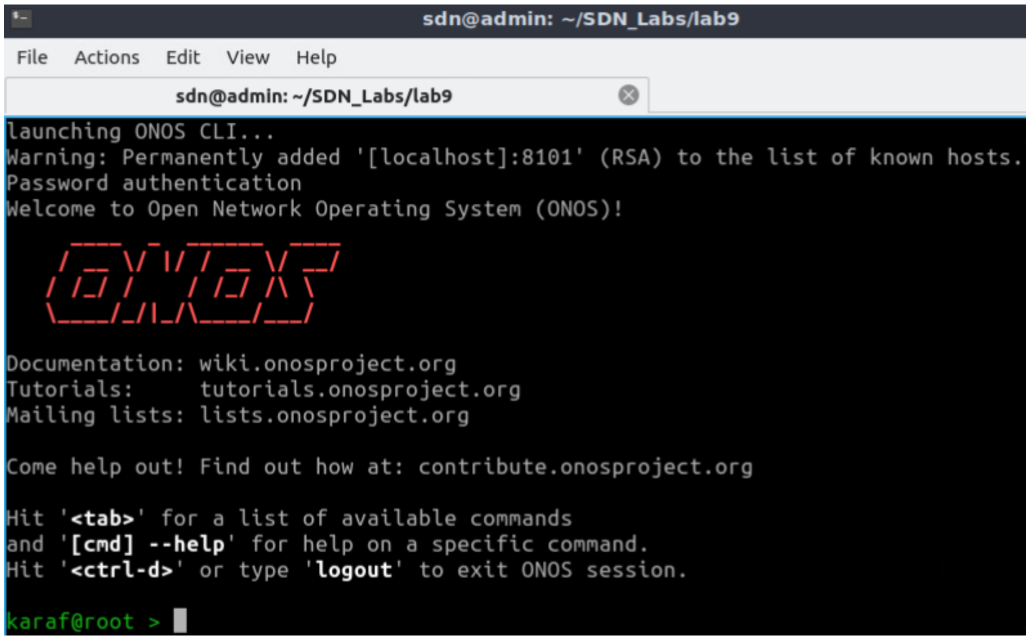
**Step 3.** A script was written to run ONOS and enter its Command Line Interface (CLI). In order to run the script in superuser (root) mode, issue the following command. When prompted for a password, type password. In addition to running ONOS, the script will modify the MAC addresses of the hosts so that they conform with the topology.

```
sudo ./run_onos.sh
```



```
sdn@admin:~$ cd SDN_Labs/lab9
sdn@admin:~/SDN_Labs/lab9$ sudo ./run_onos.sh
[sudo] password for sdn: 
```

Figure 9. Starting the ONOS controller.


sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

```
launching ONOS CLI...
Warning: Permanently added '[localhost]:8101' (RSA) to the list of known hosts.
Password authentication
Welcome to Open Network Operating System (ONOS)!

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

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

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

karaf@root >
```

Figure 10. ONOS CLI.

## 2.4 Verifying the configuration

In this section, you will verify the IP addresses listed in Table 2 and perform a connectivity test.

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

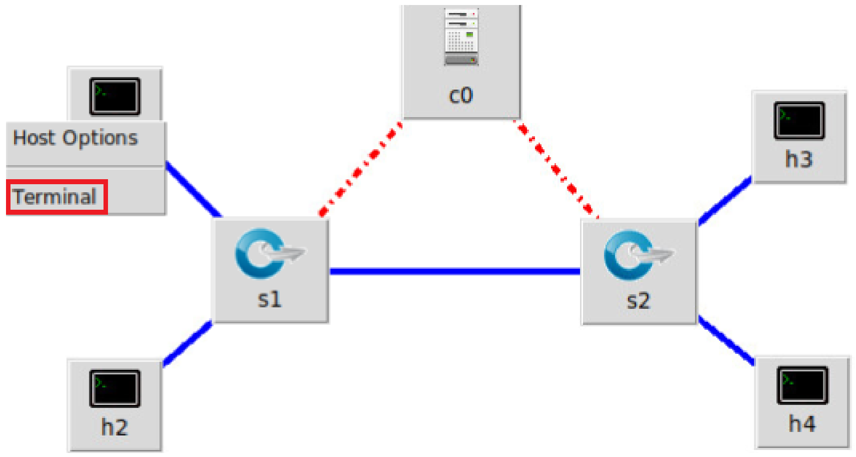
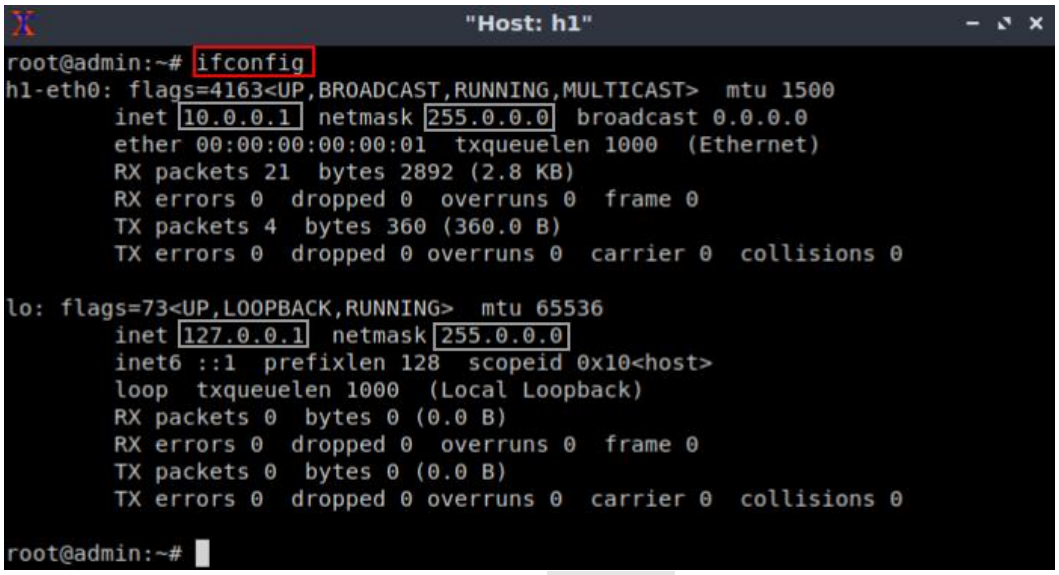


Figure 11. Opening host h1 terminal.

**Step 2**. On host h1 terminal, type the command shown below to verify that the IP address was assigned successfully. You will corroborate that host h1 has two interfaces. Interface *h1-eth0* is configured with the IP address 10.0.0.1 and the subnet mask 255.0.0.0. The loopback interface *lo* is configured with the IP address of 127.0.0.1 with a subnet mask of 255.0.0.0.

```
ifconfig
```



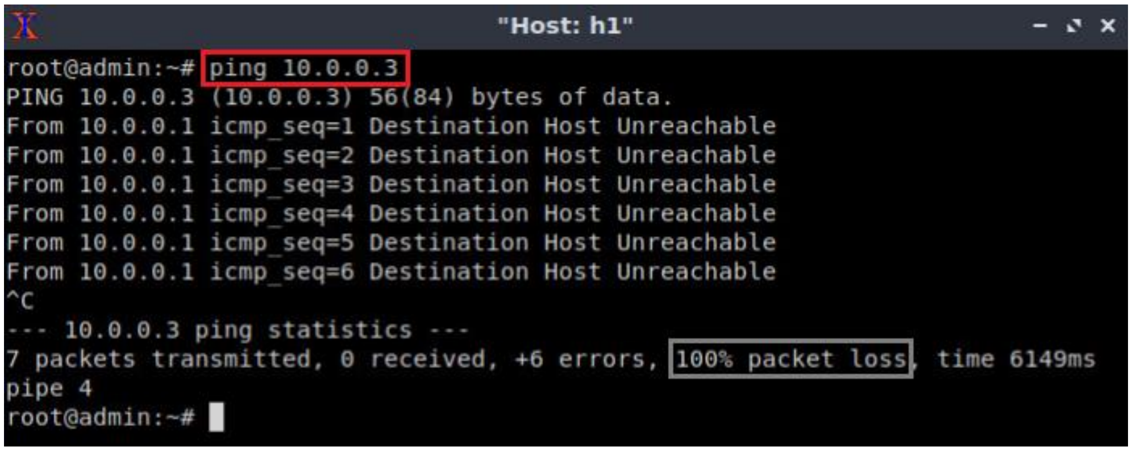
```
root@admin:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 0.0.0.0
        ether 00:00:00:00:00:01  txqueuelen 1000  (Ethernet)
        RX packets 21  bytes 2892 (2.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 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:~# 
```

Figure 12. Output of the `ifconfig` command.

**Step 3**. In order to verify the IP address of other hosts, proceed similarly by repeating steps 1 and 2 on the host’s terminal. Similar results should be observed.

**Step 4**. Test the connectivity between host h1 and host h3 using the `ping` command. On host h1 terminal, type the command specified below.

```
ping 10.0.0.3
```



```
root@admin:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 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
--- 10.0.0.3 ping statistics ---
7 packets transmitted, 0 received, +6 errors, 100% packet loss, time 6149ms
pipe 4
root@admin:~# 
```

Figure 13. Performing a connectivity test on host h1 terminal.

To stop the test, press `Ctrl+c`. The figure above shows an unsuccessful connectivity test and as a result the test had 100% packet loss.

## 3 Configuring VPLS

In this section, you will configure VPLS by enabling the OpenFlow and the VPLS applications in ONOS. Then, you will define two VPLS instances in order to establish a connection between the two customers sites.

## 3.1 Enabling OpenFlow and VPLS applications

**Step 1.** Select the ONOS CLI by clicking on the Linux terminal as shown in the figure below.

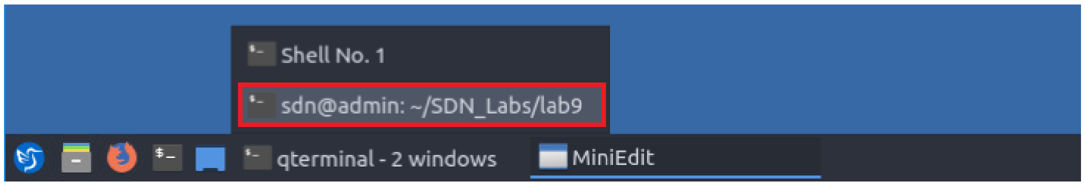
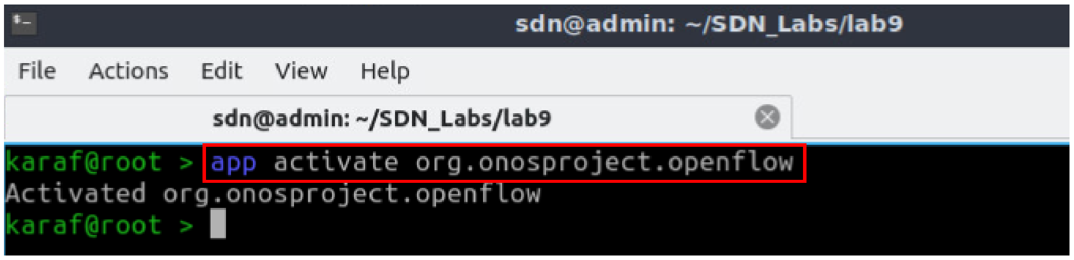


Figure 14. Navigating into the ONOS CLI.

**Step 2.** Activate the OpenFlow application by issuing the following command.

```
app activate org.onosproject.openflow
```

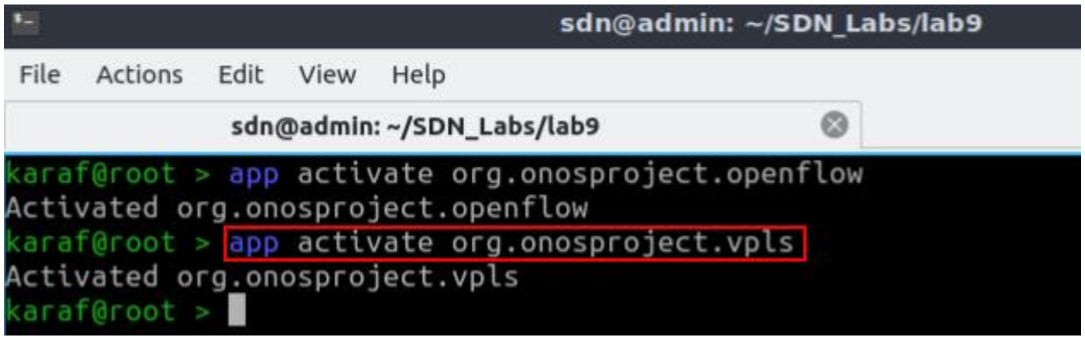

sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

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

Figure 15. Activating the OpenFlow application.

**Step 3.** Activate the VPLS application by typing the command below.

```
app activate org.onosproject.vpls
```


sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

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

Figure 16. Activating the VPLS application.

## 3.2 Displaying host information

**Step 1.** Open Mininet's CLI by selecting the terminal shown below.

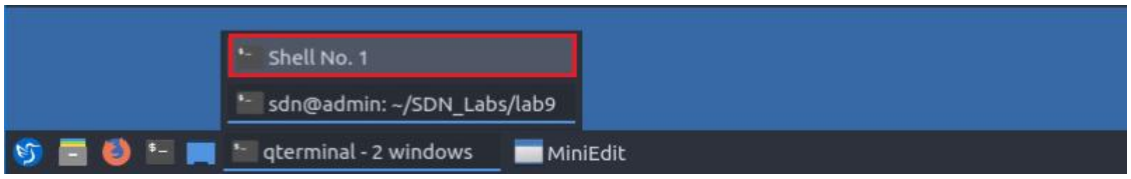
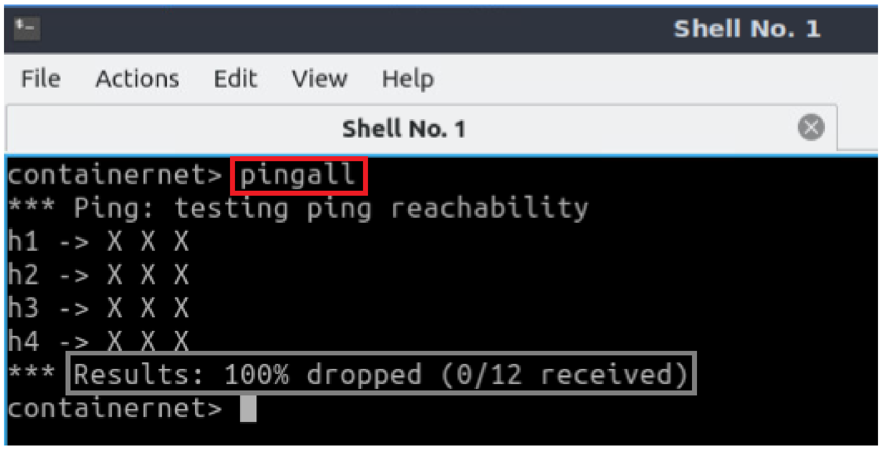


Figure 17. Navigating into Mininet's CLI.

**Step 2.** In the CLI, type the command shown below and wait until the test finishes.

```
pingall
```



```
containernet> pingall
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X X X
h3 -> X X X
h4 -> X X X
*** Results: 100% dropped (0/12 received)
containernet>
```

Figure 18. Performing a connectivity test between all the participants.

The results will show an unsuccessful (X X X) connectivity test among all end-hosts.

**Step 3.** Navigate back to the ONOS CLI by clicking on the Linux terminal as shown in the figure below.

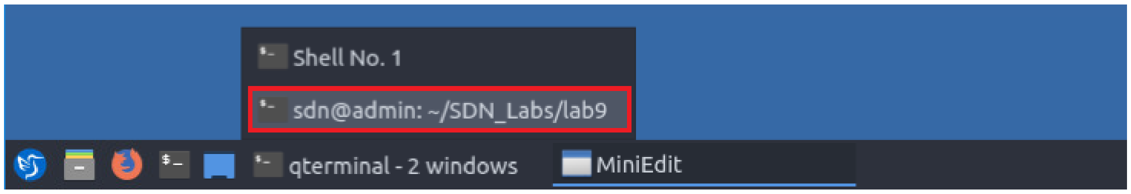
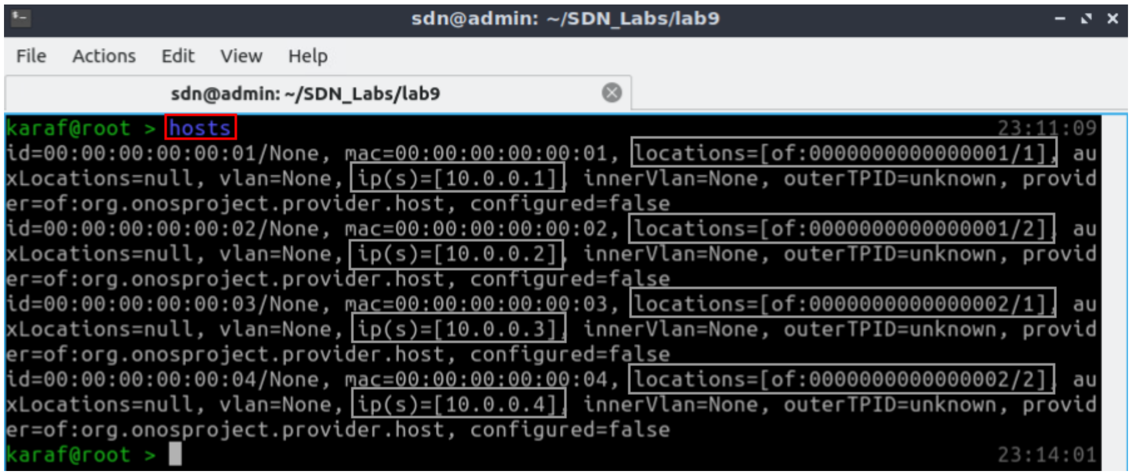


Figure 19. Navigating into ONOS CLI.

**Step 4.** Type the following command to display the hosts information.

```
hosts
```


sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

```
karaf@root > hosts                                                         23:11:09
id=00:00:00:00:00:01/None, mac=00:00:00:00:00:01, locations=[of:0000000000000001/1], au
xLocations=null, vlan=None, ip(s)=[10.0.0.1], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:00:02/None, mac=00:00:00:00:00:02, locations=[of:0000000000000001/2], au
xLocations=null, vlan=None, ip(s)=[10.0.0.2], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:00:03/None, mac=00:00:00:00:00:03, locations=[of:0000000000000002/1], au
xLocations=null, vlan=None, ip(s)=[10.0.0.3], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
id=00:00:00:00:00:04/None, mac=00:00:00:00:00:04, locations=[of:0000000000000002/2], au
xLocations=null, vlan=None, ip(s)=[10.0.0.4], innerVlan=None, outerTPID=unknown, provid
er=of:org.onosproject.provider.host, configured=false
karaf@root >                                                               23:14:01
```

Figure 20. Displaying host information.

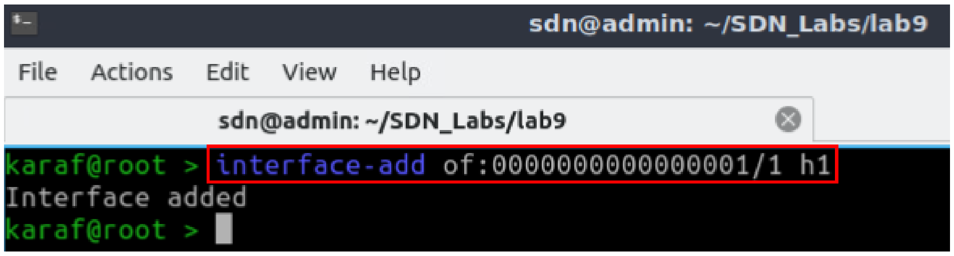
You will visualize the location and the IP address of each host. The location is related to the OpenFlow switch ports. For example, the location of:0000000000000001/2 specifies the OpenFlow switch s1 and the port number 2.

## 3.3 Associating OpenFlow interfaces to the end-hosts

**Step 1.** Considering the information contained in Table 2, associate h1 to its OpenFlow port by typing the following command.

```
interface-add of:0000000000000001/1 h1
```

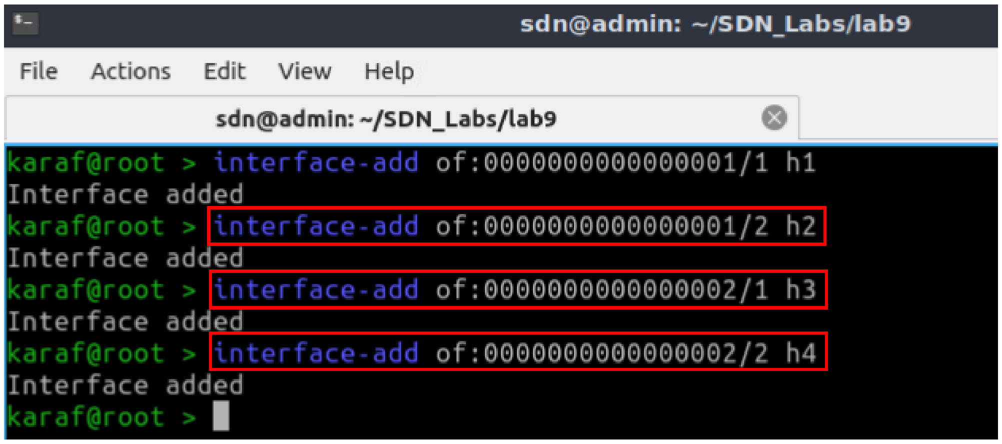
The sequence of zeroes is long and might be a source of mistakes. To have the correct number of zeros, consider using `Tab` to autocomplete.


sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

```
karaf@root > interface-add of:0000000000000001/1 h1
Interface added
karaf@root >
```

Figure 21. Adding an interface.

**Step 2.** Proceed similarly to associate the remaining interfaces. Those steps are summarized in the figure below.


sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

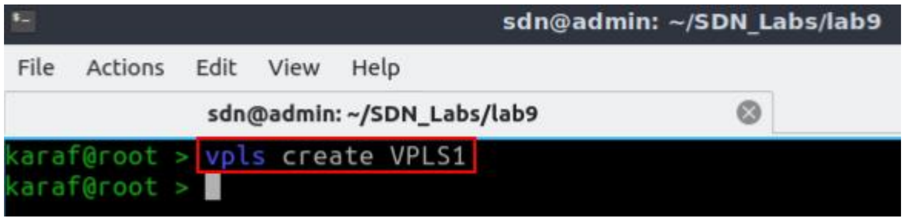
```
karaf@root > interface-add of:0000000000000001/1 h1
Interface added
karaf@root > interface-add of:0000000000000001/2 h2
Interface added
karaf@root > interface-add of:0000000000000002/1 h3
Interface added
karaf@root > interface-add of:0000000000000002/2 h4
Interface added
karaf@root >
```

Figure 22. Adding interfaces.

## 3.4 Creating a VPLS

**Step 1.** To create a new VPLS, type the command below according to the information displayed in Table 2 and the topology.

```
vpls create VPLS1
```

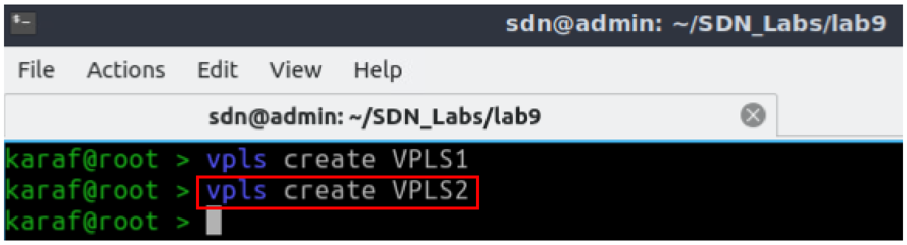

sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

```
karaf@root > vpls create VPLS1
karaf@root >
```

Figure 23. Creating VPLS1.

**Step 2.** Similarly, create another VPLS by typing the following command.

```
vpls create VPLS2
```

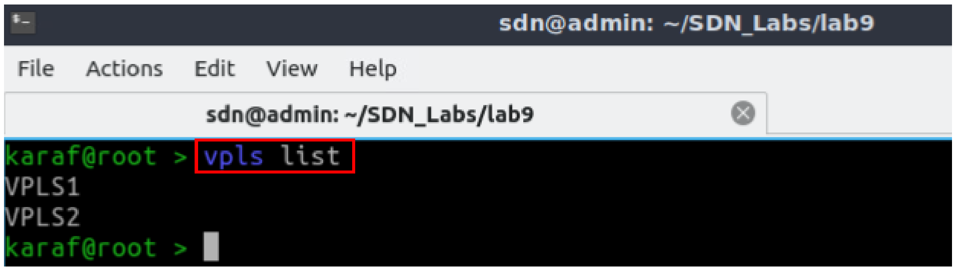

sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

```
karaf@root > vpls create VPLS1
karaf@root > vpls create VPLS2
karaf@root >
```

Figure 24. Creating VPLS2.

**Step 3.** To list the created VPLS, type the following command.

```
vpls list
```


sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

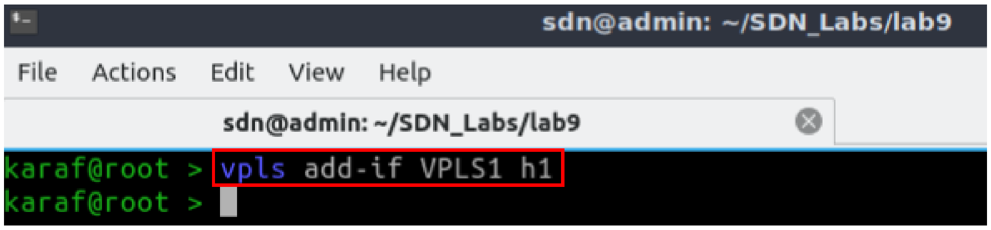
```
karaf@root > vpls list
VPLS1
VPLS2
karaf@root >
```

Figure 25. List of VPLSes.

## 3.5 Adding interfaces to an existing VPLS

**Step 1.** To add host h1 to VPLS1 type the following command.

```
vpls add-if VPLS1 h1
```

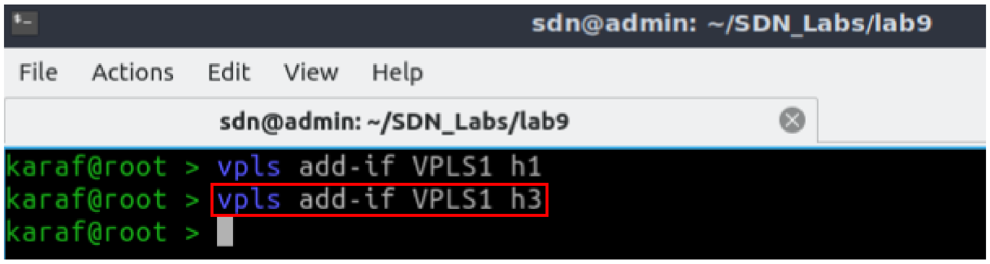

sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

```
karaf@root > vpls add-if VPLS1 h1
karaf@root >
```

Figure 26. Assigning host h1 to VPLS1.

**Step 2.** Add host h3 to VPLS1 by typing the command below. Now, host h1 and host h3 are in the same VPLS.

```
vpls add-if VPLS1 h3
```

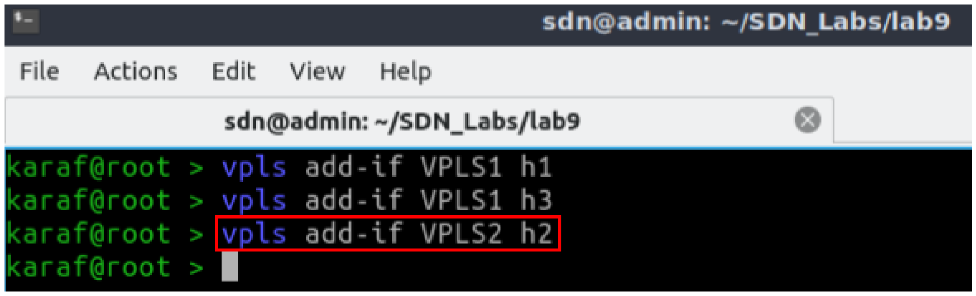

sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

```
karaf@root > vpls add-if VPLS1 h1
karaf@root > vpls add-if VPLS1 h3
karaf@root >
```

Figure 27. Assigning host h3 to VPLS1.

**Step 3.** To add host h2 to VPLS2 type the following command.

```
vpls add-if VPLS2 h2
```

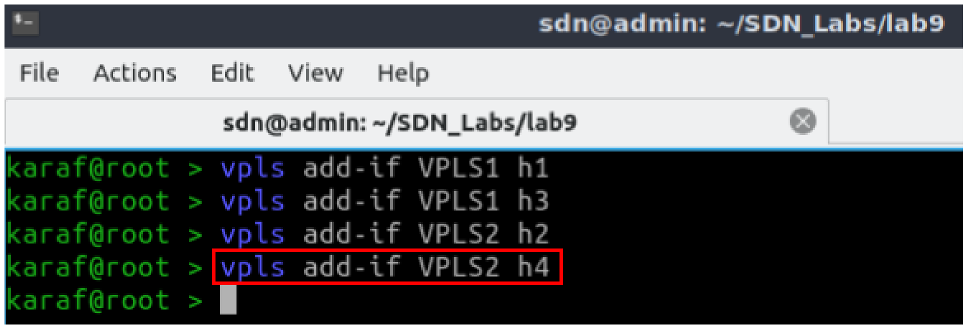

sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

```
karaf@root > vpls add-if VPLS1 h1
karaf@root > vpls add-if VPLS1 h3
karaf@root > vpls add-if VPLS2 h2
karaf@root >
```

Figure 28. Assigning host h2 to VPLS2.

**Step 4.** Similarly, add host h4 to VPLS2 by issuing the following command.

```
vpls add-if VPLS2 h4
```

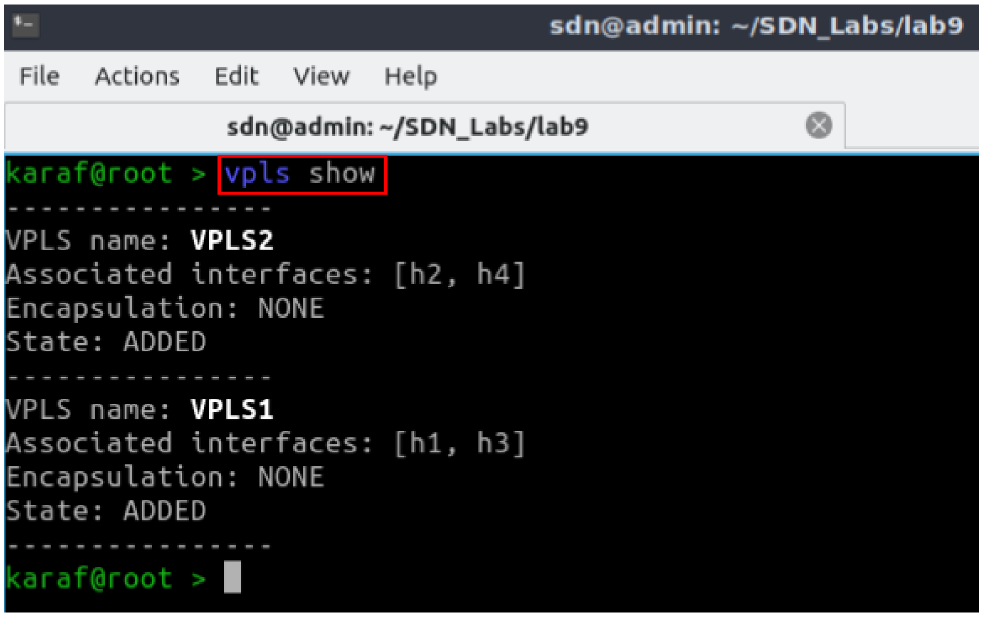

sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

```
karaf@root > vpls add-if VPLS1 h1
karaf@root > vpls add-if VPLS1 h3
karaf@root > vpls add-if VPLS2 h2
karaf@root > vpls add-if VPLS2 h4
karaf@root >
```

Figure 29. Assigning host h4 to VPLS2.

**Step 5.** To verify if the configuration was applied correctly, issue the command below.

```
vpls show
```


sdn@admin: ~/SDN_Labs/lab9
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab9

```
karaf@root > vpls show
----------------
VPLS name: VPLS2
Associated interfaces: [h2, h4]
Encapsulation: NONE
State: ADDED
----------------
VPLS name: VPLS1
Associated interfaces: [h1, h3]
Encapsulation: NONE
State: ADDED
----------------
karaf@root >
```

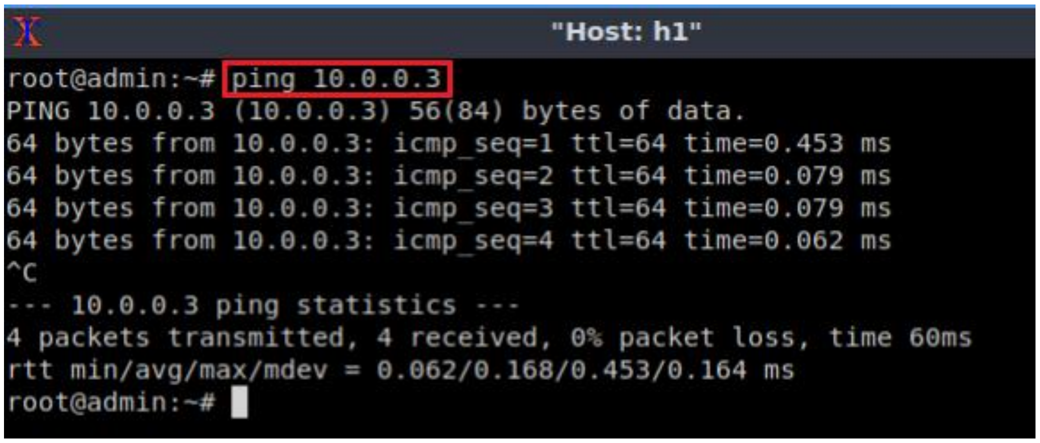
Figure 30. Verifying VPLS configuration.

The output of the figure above shows that VPLS1 is associated with hosts h1 and h3. No encapsulation is used, and the state indicates that the interfaces were added successfully. VPLS2 is associated with hosts h2 and h4, the configuration is the same as VPLS1.

## 4 Verifying connectivity between end-hosts

**Step 1.** Test the connectivity between host h1 and host h3 using the `ping` command. On host h1 terminal, type the command specified below.

```
ping 10.0.0.3
```



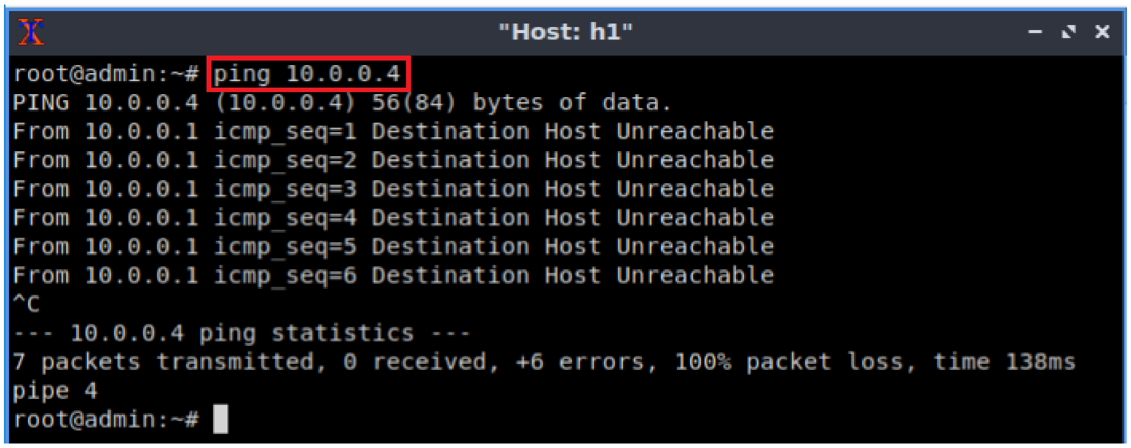
```
root@admin:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.453 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.079 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.079 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.062 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 60ms
rtt min/avg/max/mdev = 0.062/0.168/0.453/0.164 ms
root@admin:~#
```

Figure 31. Connectivity test between hosts h1 and h3

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

**Step 2.** Test the connectivity between host h1 and host h4 using the `ping` command. On host h1 terminal, type the command specified below.

```
ping 10.0.0.4
```



```
root@admin:~# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 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
--- 10.0.0.4 ping statistics ---
7 packets transmitted, 0 received, +6 errors, 100% packet loss, time 138ms
pipe 4
root@admin:~#
```

Figure 32. Connectivity test between hosts h1 and h4.

To stop the test, press `Ctrl+c`. The result in the figure above shows an unsuccessful connectivity test.

**Step 3.** Hold right-click on h2 and select *Terminal*.

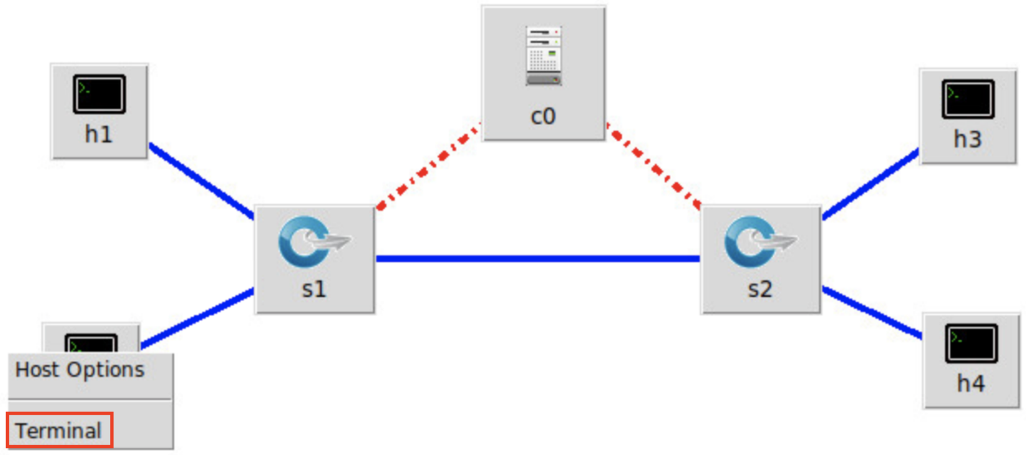
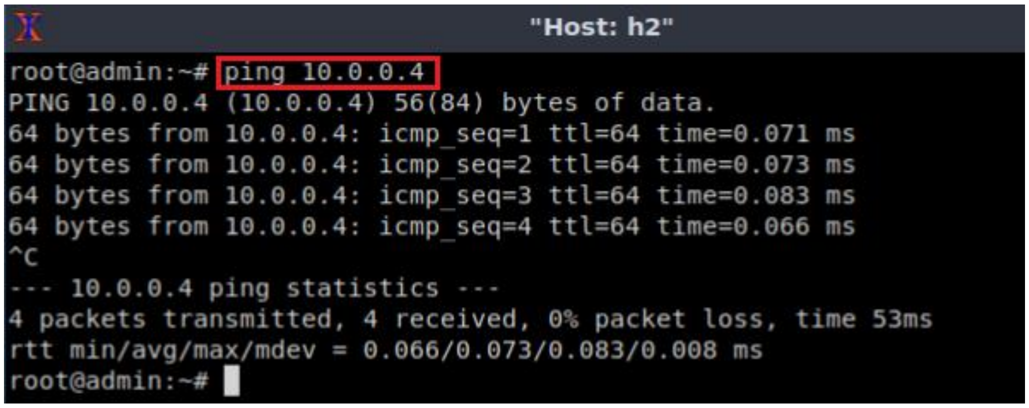


Figure 33. Opening host h2 terminal.

**Step 4.** Test the connectivity between host h2 and host h4 using the `ping` command. On host h2 terminal, type the command specified below.

```
ping 10.0.0.4
```



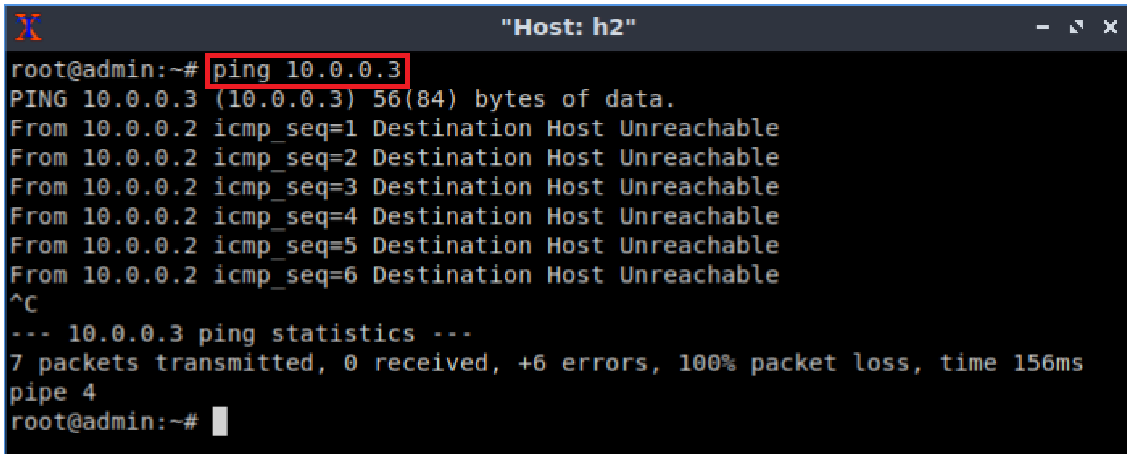
```
"Host: h2"
root@admin:~# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.071 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.073 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.083 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.066 ms
^C
--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 53ms
rtt min/avg/max/mdev = 0.066/0.073/0.083/0.008 ms
root@admin:~#
```

Figure 34. Connectivity test between hosts h2 and h4.

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

**Step 5.** Test the connectivity between host h2 and host h3 using the `ping` command. On host h2 terminal, type the command specified below.

```
ping 10.0.0.3
```



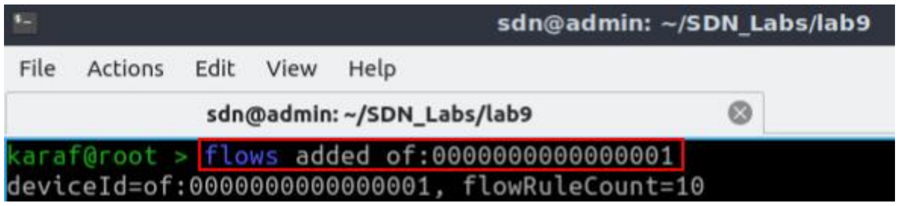
```
"Host: h2"
root@admin:~# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable
From 10.0.0.2 icmp_seq=4 Destination Host Unreachable
From 10.0.0.2 icmp_seq=5 Destination Host Unreachable
From 10.0.0.2 icmp_seq=6 Destination Host Unreachable
^C
--- 10.0.0.3 ping statistics ---
7 packets transmitted, 0 received, +6 errors, 100% packet loss, time 156ms
pipe 4
root@admin:~#
```

Figure 35. Connectivity test between hosts h2 and h3.

To stop the test, press `Ctrl+c`. The result in the figure above shows an unsuccessful connectivity test.

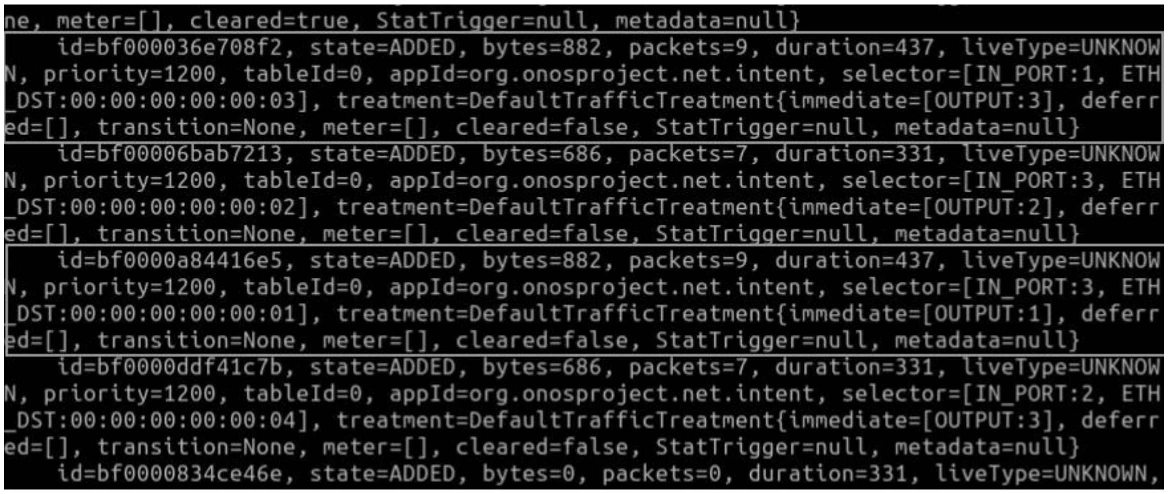
**Step 6.** On the ONOS terminal, issue the following command to display the flows of switch s1.

```
flows added of:0000000000000001
```



```
karaf@root > flows added of:0000000000000001
deviceId=of:0000000000000001, flowRuleCount=10
```

Figure 36. Displaying the added flows on switch s1.



```
ne, meter=[], cleared=true, StatTrigger=null, metadata=null}
    id=bf000036e708f2, state=ADDED, bytes=882, packets=9, duration=437, liveType=UNKNOW
N, priority=1200, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:1, ETH
_DST:00:00:00:00:00:03], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:3], deferr
ed=[], transition=None, meter=[], cleared=false, StatTrigger=null, metadata=null}
    id=bf00006bab7213, state=ADDED, bytes=686, packets=7, duration=331, liveType=UNKNOW
N, priority=1200, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:3, ETH
_DST:00:00:00:00:00:02], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:2], deferr
ed=[], transition=None, meter=[], cleared=false, StatTrigger=null, metadata=null}
    id=bf0000a84416e5, state=ADDED, bytes=882, packets=9, duration=437, liveType=UNKNOW
N, priority=1200, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:3, ETH
_DST:00:00:00:00:00:01], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:1], deferr
ed=[], transition=None, meter=[], cleared=false, StatTrigger=null, metadata=null}
    id=bf0000ddf41c7b, state=ADDED, bytes=686, packets=7, duration=331, liveType=UNKNOW
N, priority=1200, tableId=0, appId=org.onosproject.net.intent, selector=[IN_PORT:2, ETH
_DST:00:00:00:00:00:04], treatment=DefaultTrafficTreatment{immediate=[OUTPUT:3], deferr
ed=[], transition=None, meter=[], cleared=false, StatTrigger=null, metadata=null}
    id=bf0000834ce46e, state=ADDED, bytes=0, packets=0, duration=331, liveType=UNKNOWN,
```

Figure 37. Displaying the added flows on switch s1.

Consider the above two figures. There are 10 flow rules installed on switch s1. For simplicity, we only show some of the rules related to VPLS. The first highlighted rule matches against incoming packets at port s1-eth1 (`IN_PORT:1`) with MAC destination address of host h3 (`ETH_DST:00:00:00:00:00:03`), and forwards them out of port s1-eth3 (`OUTPUT:3`). The second highlighted rule matches against packets from host h3 to h1. Similarly, flow rules for hosts h2 and h4 can be inspected. Furthermore, the flow table of switch s2 contains flow rules similar to those in switch s1.

This concludes Lab 9. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

## References

1. K. Kompella and Y. Rekhter. *"RFC 4761: Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling",* 2007.
2. L. De Ghein, "*MPLS Fundamentals"*, Cisco Press, CCIE No. 1897, 2016.
3. A. Tanenbaum, D. Wetherall, "*Computer networks*", 5th Edition, Pearson, 2012.
4. P. Goransson, C. Black, T. Culver. "*Software defined networks: a comprehensive approach"*. Morgan Kaufmann, 2016.
5. P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.C. Wang, J. Bi. "*Seamless interworking of SDN and IP*", In Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM (pp. 475-476). 2013.
6. ONOS project. "*SDN-IP*". [Online]. Available: https://wiki.onosproject.org/display/ONOS/SDN-IP.

# SOFTWARE DEFINED NETWORKING

# Lab 10: Applying Equal-cost Multi-path Protocol (ECMP) within SDN networks

**Document Version: 07-03-2021**

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

## Contents

## Overview

This lab is an introduction to Equal-cost Multi-path Protocol routing (ECMP) within Software Defined Networking (SDN). The focus in this lab is to apply load balancing within the SDN network so that the traffic is distributed across the links efficiently. Load balancing is achieved via ECMP and is activated via the *segmentrouting* application.

## Objectives

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

1. Understand and apply ECMP.
2. Utilize the sFlow tool and interact with its components.
3. Apply load balancing within the SDN network.
4. Visualize flows through the sFlow dashboard.

## Lab settings

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

Table 1**.** Credentials to access the Client 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: Launching the sFlow tool.
4. Section 4: Starting the ONOS controller.
5. Section 5: Applying load-balancing within the SDN network.
6. Section 6: Testing the load balancer within the SDN network.

## 1 Introduction

### 1.1 Load-balancing via ECMP

Generally, network engineers seek to make use of their available resources and increase the bandwidth of their networks as much as possible. A simple solution for increasing the bandwidth is to upgrade existing infrastructure with better quality; for instance, replacing a 1 Gbps Ethernet link with 10 Gbps. However, this solution is costly and often unwanted when there are cheaper alternatives. Nowadays, it is very common to use parallel links to increase bandwidth. This technique splits network traffic among multiple links to balance the load (i.e., load balancing technique). There are various ways to make the decision on which packet traverses which link. The simplest approach is per-packet load-balancing, where packet 1 is transmitted over link A, packet 2 over link B, and so on alternating between the available links. However, this approach might suffer from several drawbacks, such as degrading the network performance by resending unordered packets often marked as dropped[2].

Equal-cost multi-path routing (ECMP) is a routing technique for forwarding packets along multiple paths of equal cost[3]. One method of selecting which next-hop to use is the hash-threshold approach. In such an approach, the traffic is forwarded based on the hash values of some header fields that remain the same within a traffic flow (e.g., the tuple source IP, destination IP, source port, destination port).

Consider Figure 1. Switch s1 implements load-balancing capability through ECMP. In the forwarding table of switch s1, there are multiple paths to the same destination; for instance, switch s1 can reach host h1 via s1 -> s2 -> h1 or s1 -> s3 -> h1. To balance the load, switch s1 computes the hash of some of the packet's header fields and forwards the packet accordingly. The selected field values of packets of a given traffic flow are always the same. Consequently, packets of a given traffic flow will always take the same path, even when multiple paths are available.

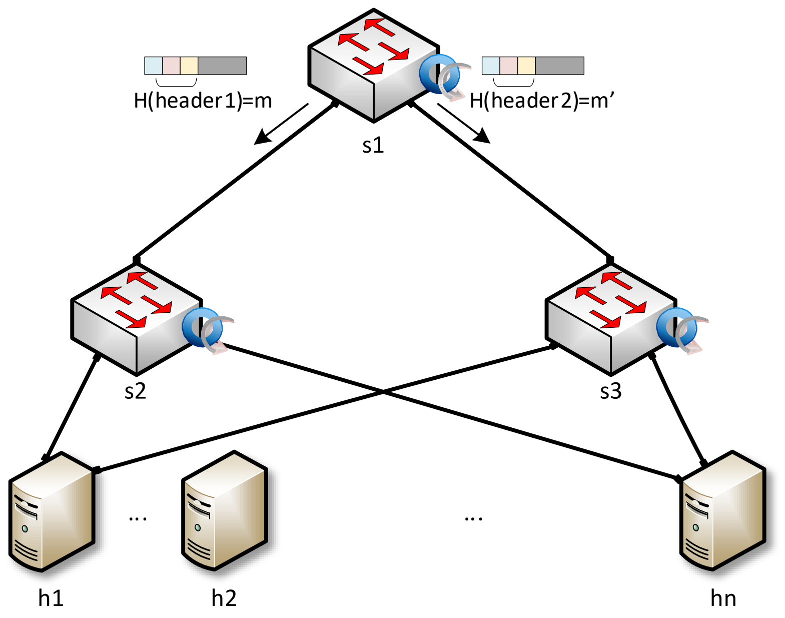


Figure 1. Load-balancing using ECMP.

## 2 Lab topology

Consider Figure 2. The topology consists of four OpenFlow switches, four hosts, and one ONOS controller managing the switches. Hosts h1 and h2 are within the network 10.1.1.0/24, whereas hosts h3 and h4 are within the network 10.1.2.0/24. There are two paths from network 1 to network 2. The goal is to configure the *segmentrouting* application on the switches, where they will balance the load among the hosts in the two networks.

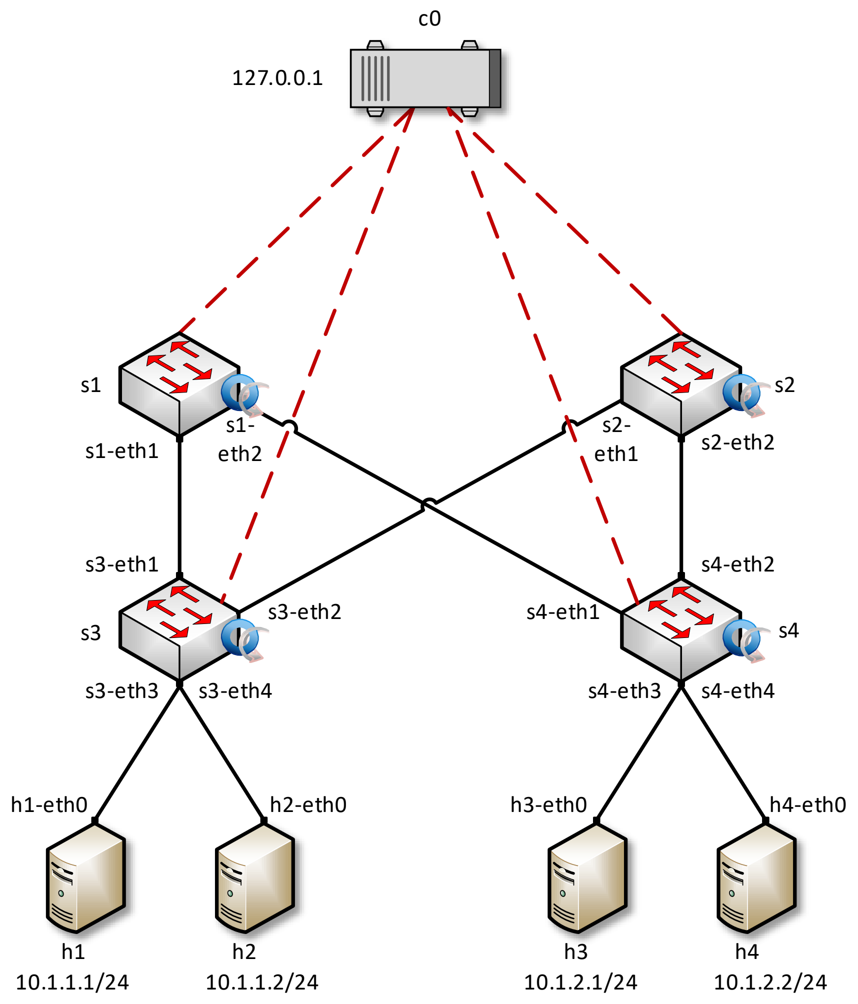


Figure 2. Lab topology.

## 2.1 Lab settings

The devices are already configured according to Table 2.

Table 2. Topology information.

| Device | Interface | IP Address | Subnet | Default gateway |
|---|---|---|---|---|
| Host h1 | h1-eth0 | 10.1.1.1 | /24 | 10.1.1.254 |
| Host h2 | h2-eth0 | 10.1.1.2 | /24 | 10.1.1.254 |
| Host h3 | h3-eth0 | 10.1.2.1 | /24 | 10.1.2.254 |

| | | | | |
|---|---|---|---|---|
| Host h4 | h4-eth0 | 10.1.2.2 | /24 | 10.1.2.254 |
| c0 | N/A | 127.0.01 | /32 | N/A |

## 2.2 Loading the 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 such as: configuring network elements (i.e IP addresses, default gateway), saving the topologies, and exporting layer 2 models.

**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`.

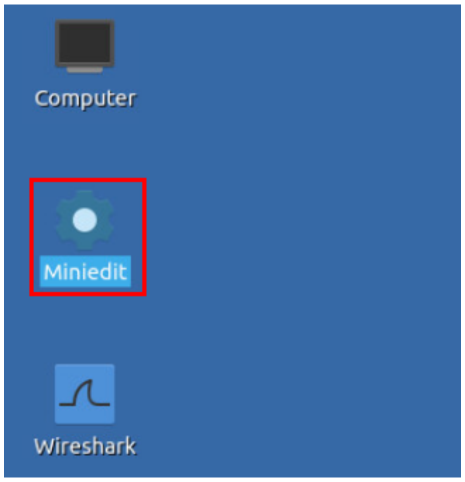


Figure 3. MiniEdit shortcut.

**Step 2.** On MiniEdit's menu bar, click on *File* then *open* to load the lab's topology. Open the *Lab10.mn* topology file stored in the default directory, */home/sdn/SDN_Labs /lab10* and click on *Open*.

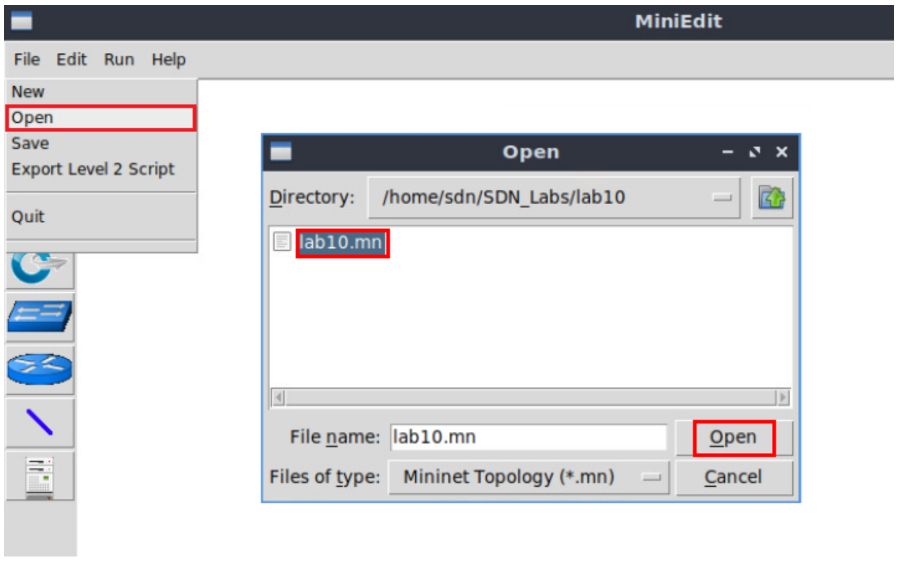


Figure 4. Opening topology.

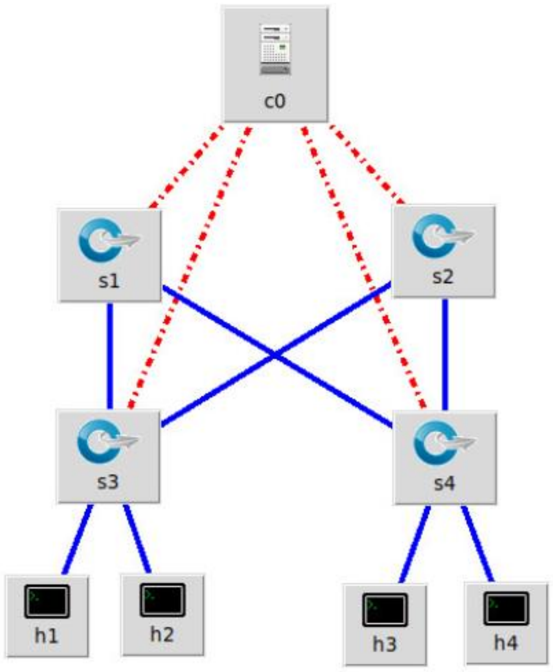


Figure 5. MiniEdit's topology.

**Step 3.** On MiniEdit's menu bar, click on *Edit>Preferences* to view OpenFlow enabled version in our topology.

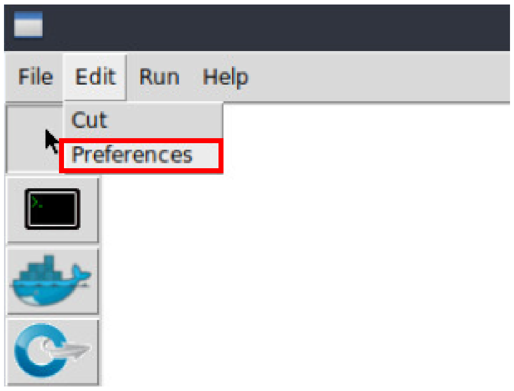
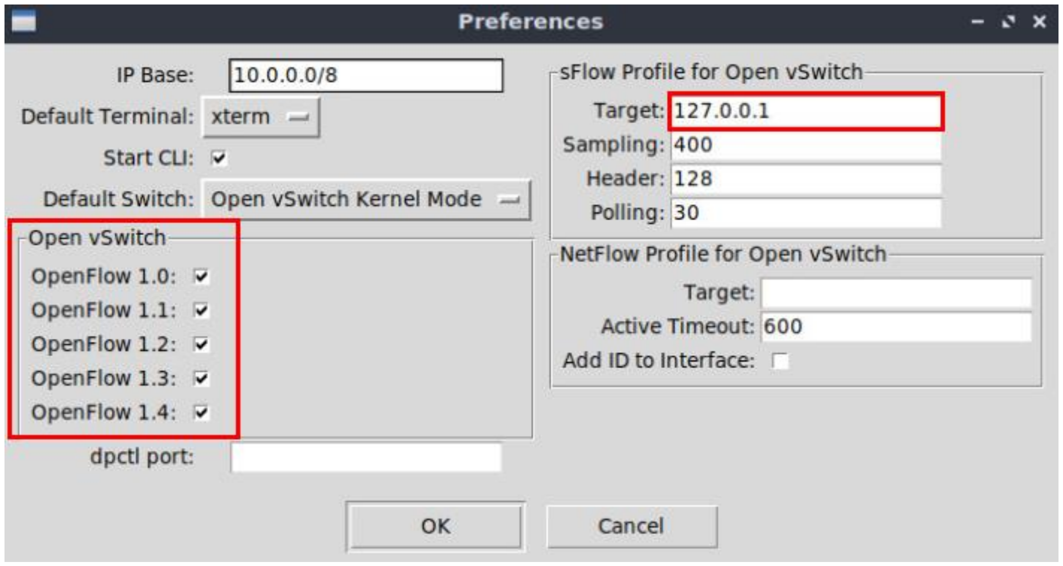


Figure 6. Opening MiniEdit preferences

**Step 4.** Validate that the OpenFlow 1.0, OpenFlow 1.1, OpenFlow 1.2, OpenFlow 1.3, and OpenFlow 1.4 version are enabled in this topology, i.e., the Open vSwitches s1, s2, s3, and s4 support these versions.



Preferences

IP Base: 10.0.0.0/8
Default Terminal: xterm
Start CLI: ✓
Default Switch: Open vSwitch Kernel Mode

Open vSwitch
OpenFlow 1.0: ✓
OpenFlow 1.1: ✓
OpenFlow 1.2: ✓
OpenFlow 1.3: ✓
OpenFlow 1.4: ✓

dpctl port:

sFlow Profile for Open vSwitch
Target: 127.0.0.1
Sampling: 400
Header: 128
Polling: 30

NetFlow Profile for Open vSwitch
Target:
Active Timeout: 600
Add ID to Interface: ☐

Figure 7. Opening MiniEdit preferences.

Consider Figure 7. For this lab, we have enabled OpenFlow 1.0, OpenFlow 1.1, OpenFlow 1.2, OpenFlow 1.3, and OpenFlow 1.4 so that the controller can communicate with the switches and instruct them to perform load balancing in later stages. The target of sFlow is set to the IP address 127.0.0.1, as sFlow is running locally. sFlow is an application used to monitor the traffic flowing across a switch[5].

**Step 5.** Click on Cancel to close the pop-up window without modifying any field.

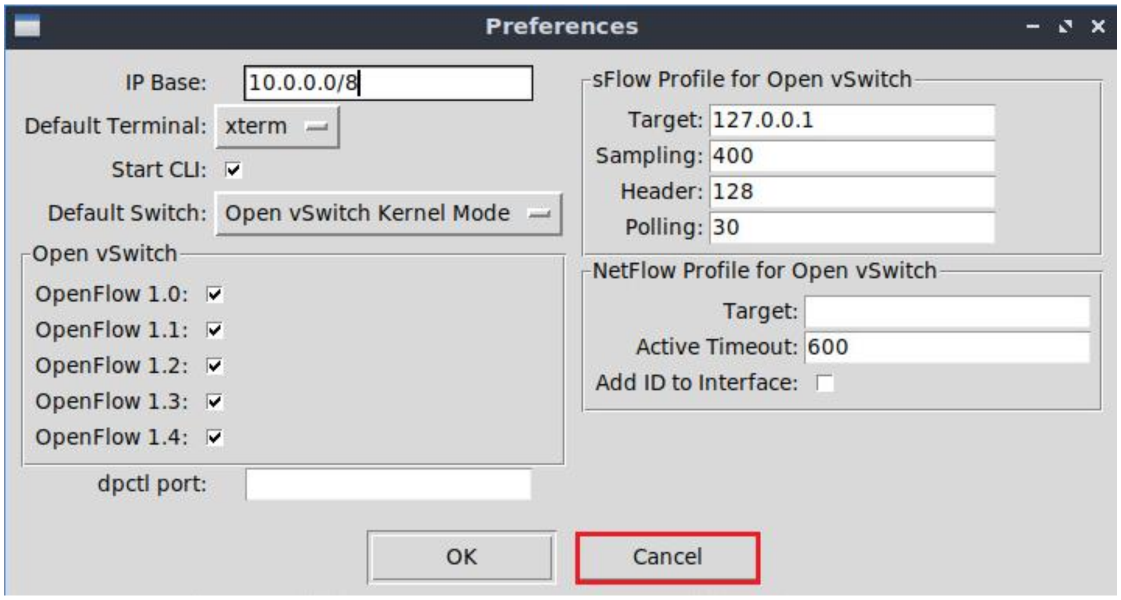


Figure 8. Exiting the pop-up window.

## 2.3 Run the emulation

In this section, you will run the emulation and check the links and interfaces that connect the devices in the given topology. Additionally, you will verify the IP addresses listed in Table 2.

**Step 1.** At this point hosts h1, h2, h3, and h4 are configured. To proceed with the emulation, click on the *Run* button located on the lower left-hand side.

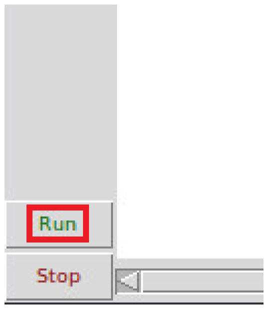
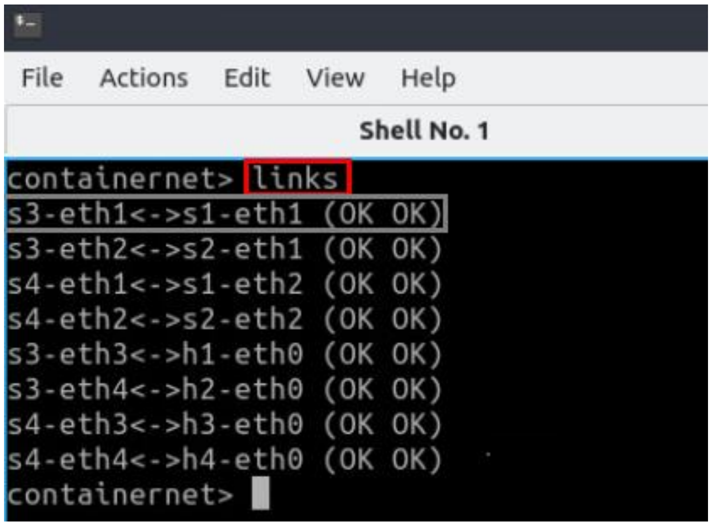


Figure 9. Starting the emulation.

**Step 2.** Issue the following command on Mininet terminal to display the interface names and connections.

```
links
```



```
containernet> links
s3-eth1<->s1-eth1 (OK OK)
s3-eth2<->s2-eth1 (OK OK)
s4-eth1<->s1-eth2 (OK OK)
s4-eth2<->s2-eth2 (OK OK)
s3-eth3<->h1-eth0 (OK OK)
s3-eth4<->h2-eth0 (OK OK)
s4-eth3<->h3-eth0 (OK OK)
s4-eth4<->h4-eth0 (OK OK)
containernet> 
```

Figure 10. Displaying network interfaces.

In the figure above, the link displayed within the gray box indicates that interface *eth1* of switch s3 connects to interface *eth1* of switch s1 (i.e., *s3-eth1<->s1-eth1*).

**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.

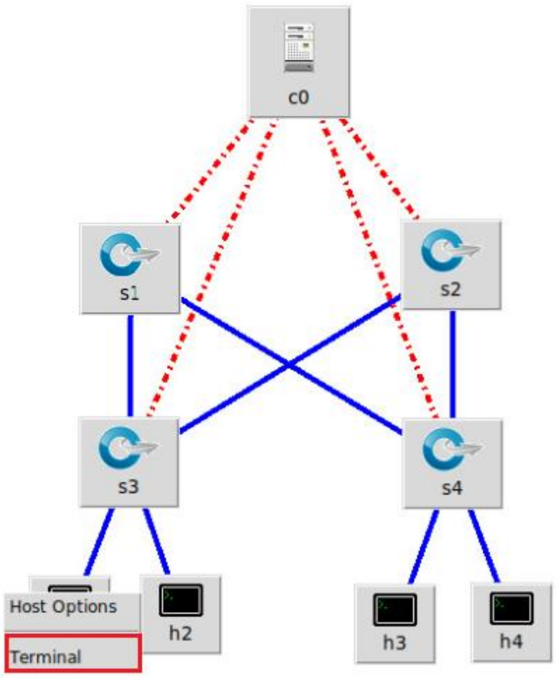
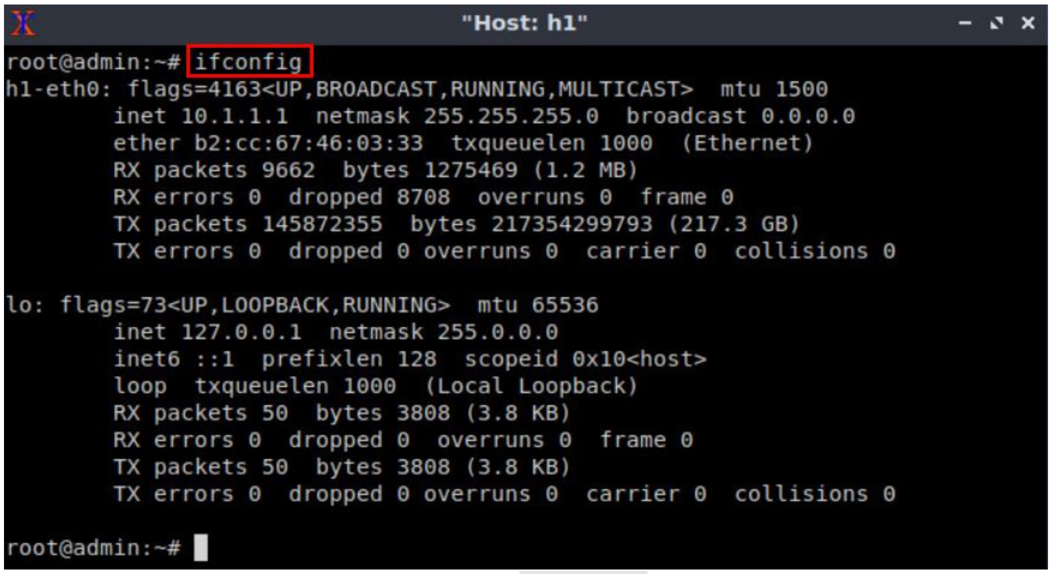


Figure 11. Opening a terminal on host h1.

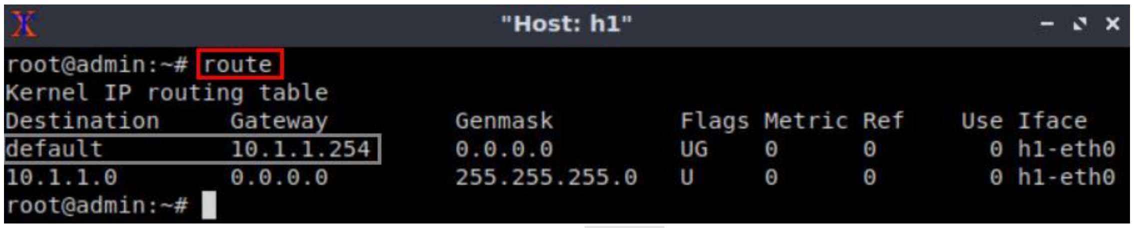
**Step 4**. On host h1 terminal, type the command shown below to verify that the IP address was assigned successfully. You will corroborate that host h1 has two interfaces, *h1-eth0* and *lo.* The interface *h1-eth0* is configured with the IP address 10.1.1.1 and the subnet mask 255.255.255.0. The interface lo is configured with the IP address of 127.0.0.1 and the subnet mask of 255.0.0.0.

```
ifconfig
```


```
root@admin:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.1.1  netmask 255.255.255.0  broadcast 0.0.0.0
        ether b2:cc:67:46:03:33  txqueuelen 1000  (Ethernet)
        RX packets 9662  bytes 1275469 (1.2 MB)
        RX errors 0  dropped 8708  overruns 0  frame 0
        TX packets 145872355  bytes 217354299793 (217.3 GB)
        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 50  bytes 3808 (3.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 50  bytes 3808 (3.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@admin:~#
```
Figure 12. Output of `ifconfig` command.

**Step 5**. On host h1 terminal, type the command shown below to verify that the default gateway IP address is 10.1.1.254.

```
route
```


```
root@admin:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.1.1.254      0.0.0.0         UG    0      0        0 h1-eth0
10.1.1.0        0.0.0.0         255.255.255.0   U     0      0        0 h1-eth0
root@admin:~#
```
Figure 13. Output of `route` command.

**Step 6**. In order to verify the IP address and default gateway of other hosts, proceed similarly by repeating steps 3 to 5 on the host's terminal. Similar results should be observed.

## 3 Launching the sFlow tool

In this section, you will launch sFlow, a tool for monitoring network traffic, to see how the traffic is balanced between the links in later sections.

**Step 1.** Go to the opened Linux terminal.

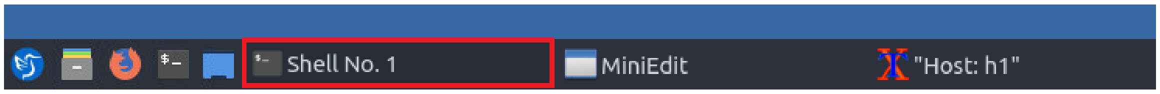


Figure 14. Opening Linux terminal.

**Step 2.** Click on *File>New Tab* to open an additional tab in the Linux terminal. Alternatively, you may press `Ctrl+Shift+T`.

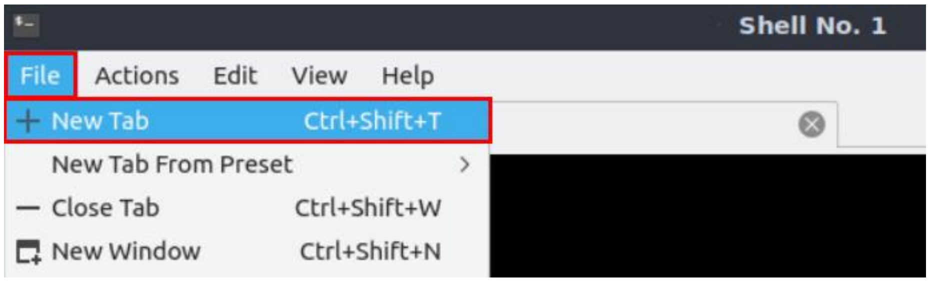
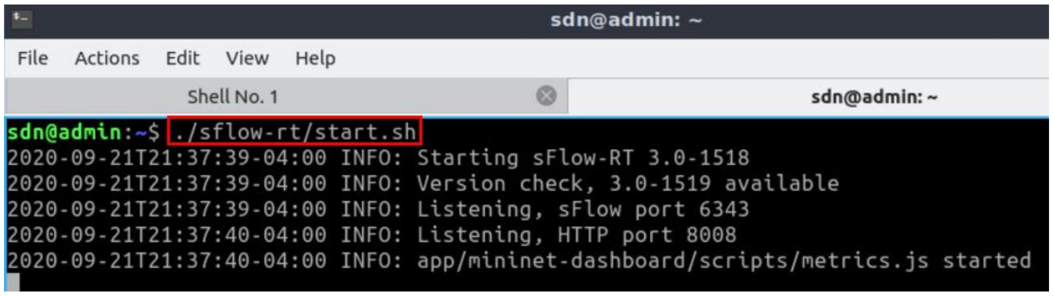


Figure 15. Opening an additional tab.

**Step 3.** Issue the following command to start sFlow.

```
./sflow-rt/start.sh
```



```
sdn@admin:~$ ./sflow-rt/start.sh
2020-09-21T21:37:39-04:00 INFO: Starting sFlow-RT 3.0-1518
2020-09-21T21:37:39-04:00 INFO: Version check, 3.0-1519 available
2020-09-21T21:37:39-04:00 INFO: Listening, sFlow port 6343
2020-09-21T21:37:40-04:00 INFO: Listening, HTTP port 8008
2020-09-21T21:37:40-04:00 INFO: app/mininet-dashboard/scripts/metrics.js started
```

Figure 16. Starting sFlow tool.

Consider Figure 16. After you execute the command above, sFlow will be running in the background. In the next steps, you will visualize some of sFlow capabilities using its dashboard that is accessible through a browser.

**Step 4.** Click on the Firefox button to start the browser.

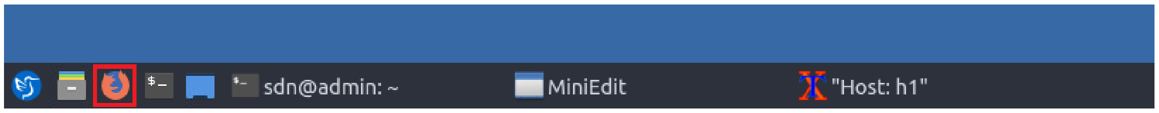


Figure 17. Opening Firefox browser.

**Step 5.** sFlow runs locally and includes a dashboard. In order to launch the dashboard, navigate to the following URL.

```
127.0.0.1:8008
```

- 127.0.0.1: loopback IP address, also referred to as the localhost.
- 8008: port number through which sFlow dashboard is accessed.

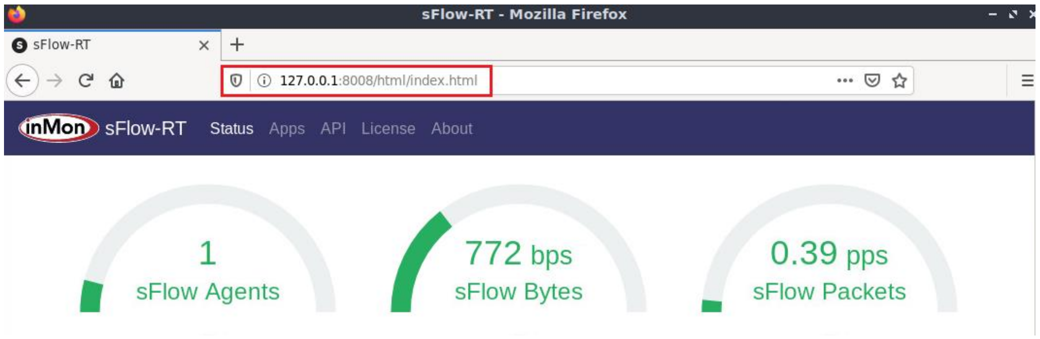


Figure 18. Navigating to sFlow dashboard.

Note that once you write the URL 127.0.0.1:8008, it will automatically be redirected to 127.0.0.1:8008/html/index. The numbers displayed regarding sFlow Agents, Bytes, and Packets fluctuates based on the monitored traffic.

**Step 6.** Click on the `Apps` tab to go to the available sFlow applications. From there, click on `mininet-dashboard` application button.

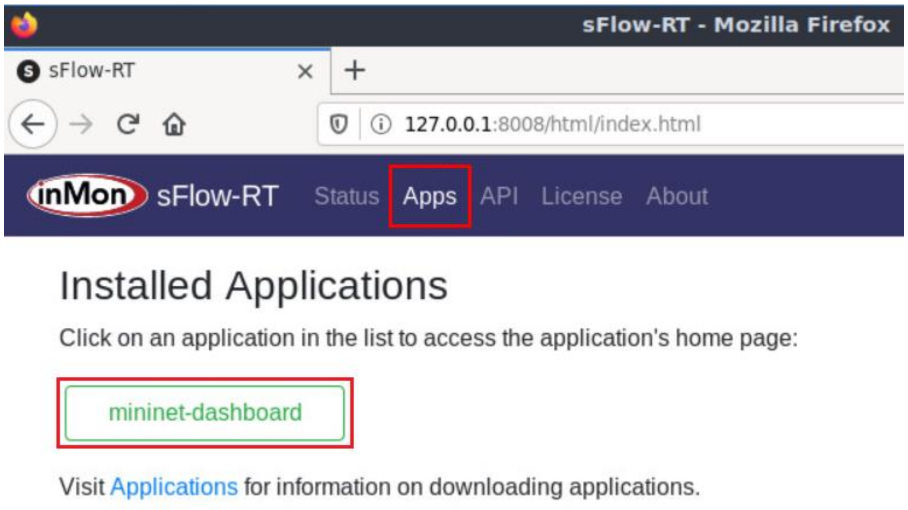

sFlow-RT - Mozilla Firefox
sFlow-RT
127.0.0.1:8008/html/index.html
inMon sFlow-RT
Status
Apps
API
License
About
Installed Applications
Click on an application in the list to access the application's home page:
mininet-dashboard
Visit Applications for information on downloading applications.

Figure 19. Navigating to mininet-dashboard application.

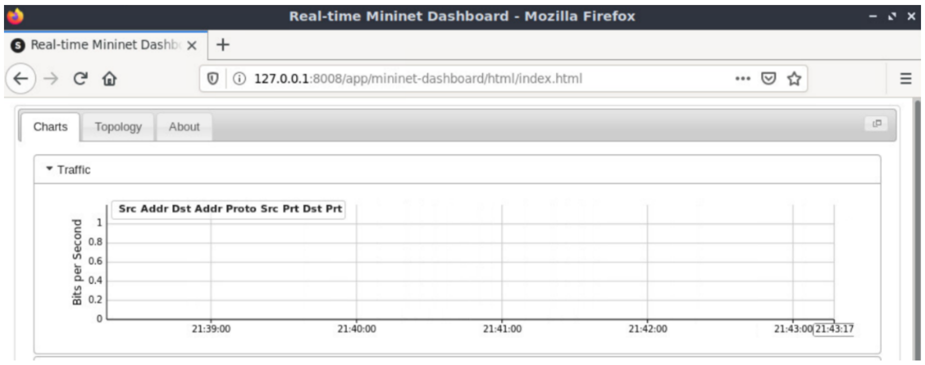

Real-time Mininet Dashboard - Mozilla Firefox
Real-time Mininet Dashb
127.0.0.1:8008/app/mininet-dashboard/html/index.html
Charts
Topology
About
Traffic
Src Addr Dst Addr Proto Src Prt Dst Prt
Bits per Second
1
0.8
0.6
0.4
0.2
0
21:39:00
21:40:00
21:41:00
21:42:00
21:43:00
21:43:17

Figure 20. Mininet-dashboard application.

Consider Figure 20. The *mininet-dashboard* application displays statistics about the current running topology in Mininet. Currently, Mininet is not attached to sFlow. Thus, no information will be displayed. In later sections, you will attach Mininet to sFlow and visualize the network traffic.

## 4 Starting the ONOS controller

In this section, you will start the ONOS controller and activate the OpenFlow application so that the controller discovers the devices, hosts, and links in the topology.

**Step 1.** Go back to the Linux terminal.

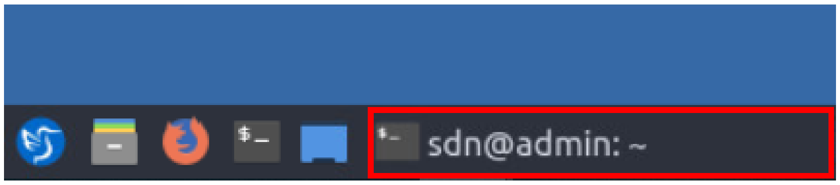
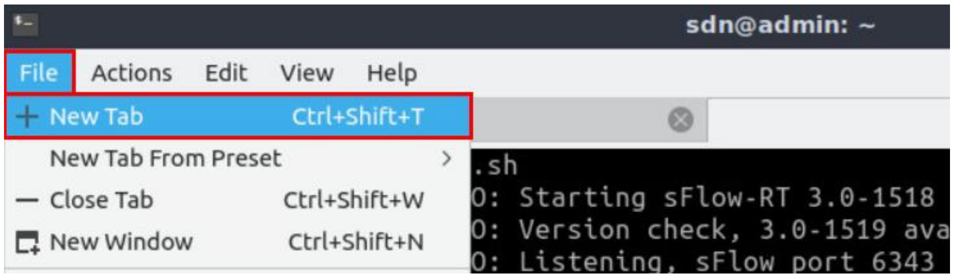


Figure 21. Opening Linux terminal.

**Step 2.** Click on *File>New Tab* to open an additional tab in the Linux terminal. Alternatively, you may press `Ctrl+Shift+T`.

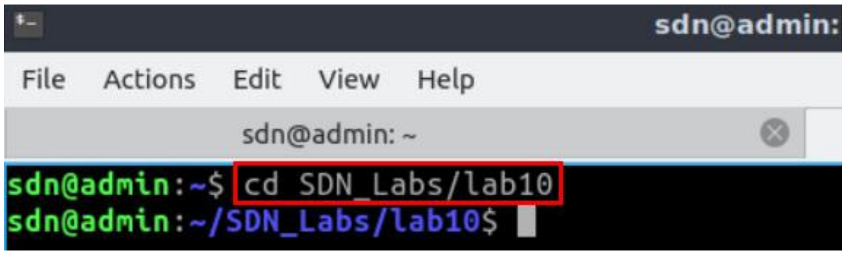


```
.sh
O: Starting sFlow-RT 3.0-1518
O: Version check, 3.0-1519 ava
O: Listening, sFlow port 6343
```

Figure 22. Opening an additional tab.

**Step 3.** Navigate into *SDN_Labs/lab10* directory by issuing the following command.

```
cd SDN_Labs/lab10
```

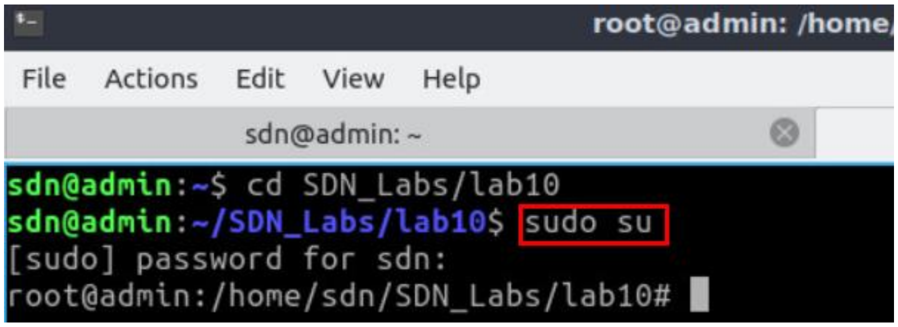


```
sdn@admin:~$ cd SDN_Labs/lab10
sdn@admin:~/SDN_Labs/lab10$
```

Figure 23. Entering the *SDN_Labs/lab10* directory.

**Step 4.** Issue the command below to execute programs with the security privileges of the superuser (root). When prompted for a password, type `password`.

```
sudo su
```

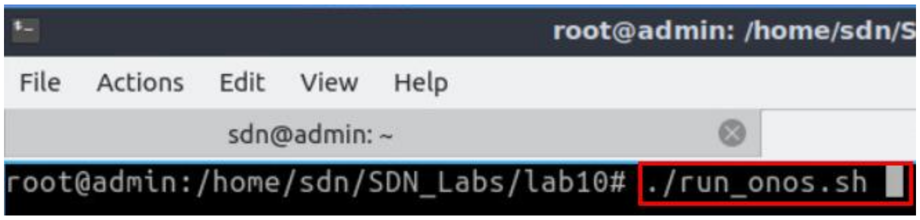


```
sdn@admin:~$ cd SDN_Labs/lab10
sdn@admin:~/SDN_Labs/lab10$ sudo su
[sudo] password for sdn:
root@admin:/home/sdn/SDN_Labs/lab10#
```

Figure 24. Switching to root mode.

**Step 5.** A script was written to run ONOS and enter its Command Line Interface (CLI). In order to run the script, issue the following command.

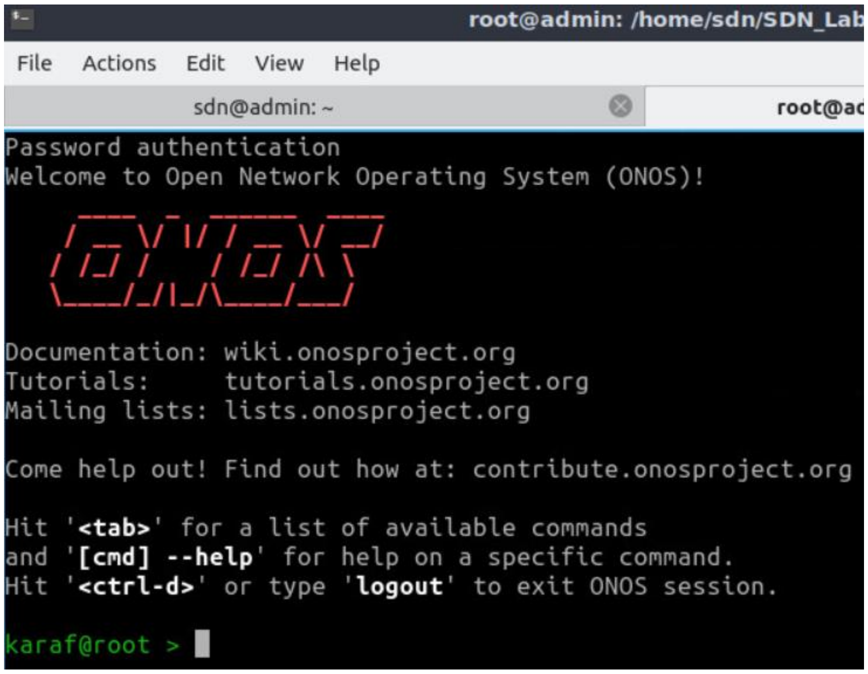
```
./run_onos
```



```
root@admin:/home/sdn/SDN_Labs/lab10# ./run_onos.sh
```

Figure 25. Starting the ONOS controller.

Once the script finishes executing and ONOS is ready, you will be able to execute commands on ONOS CLI as shown in the figure below. Note that this script may take a couple of minutes.

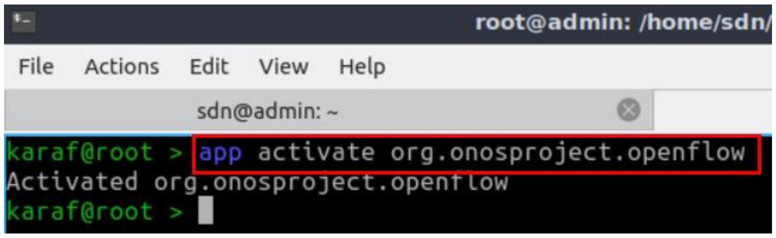


```
Password authentication
Welcome to Open Network Operating System (ONOS)!

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

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

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

karaf@root >
```

Figure 26. ONOS CLI.

**Step 6.** In the ONOS terminal, issue the following command to activate the OpenFlow application.

```
app activate org.onosproject.openflow
```



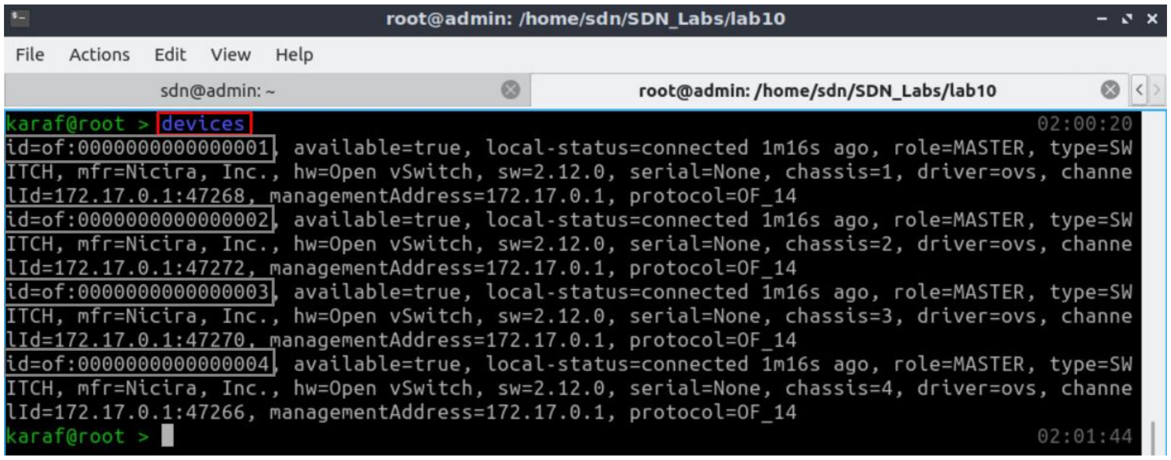
```
karaf@root > app activate org.onosproject.openflow
Activated org.onosproject.openflow
karaf@root >
```

Figure 27. Activating OpenFlow application.

Note that when you activate any ONOS application, you may have to wait few seconds so that the application gives the correct output.

**Step 7.** To display the list of all currently known devices (OVS switches), type the following command.

```
devices
```

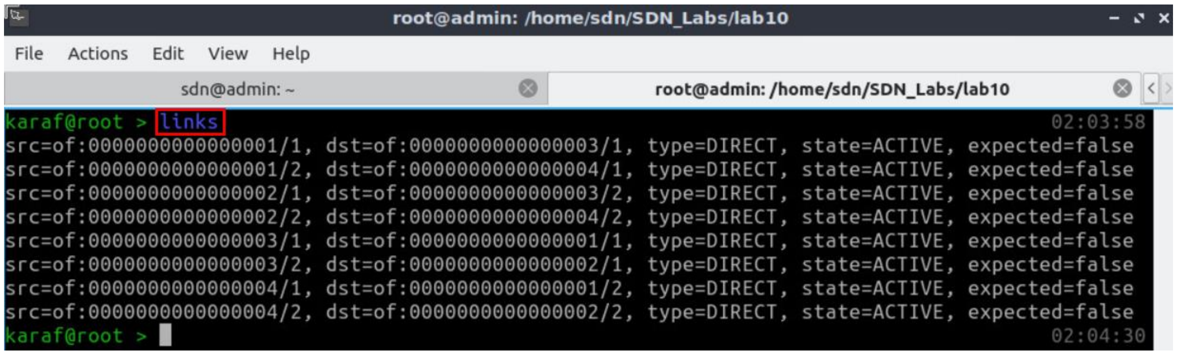

root@admin: /home/sdn/SDN_Labs/lab10
File Actions Edit View Help
sdn@admin: ~
root@admin: /home/sdn/SDN_Labs/lab10

```
karaf@root > devices                                                     02:00:20
id=of:0000000000000001, available=true, local-status=connected 1m16s ago, role=MASTER, type=SW
ITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=1, driver=ovs, channe
lId=172.17.0.1:47268, managementAddress=172.17.0.1, protocol=OF_14
id=of:0000000000000002, available=true, local-status=connected 1m16s ago, role=MASTER, type=SW
ITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=2, driver=ovs, channe
lId=172.17.0.1:47272, managementAddress=172.17.0.1, protocol=OF_14
id=of:0000000000000003, available=true, local-status=connected 1m16s ago, role=MASTER, type=SW
ITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=3, driver=ovs, channe
lId=172.17.0.1:47270, managementAddress=172.17.0.1, protocol=OF_14
id=of:0000000000000004, available=true, local-status=connected 1m16s ago, role=MASTER, type=SW
ITCH, mfr=Nicira, Inc., hw=Open vSwitch, sw=2.12.0, serial=None, chassis=4, driver=ovs, channe
lId=172.17.0.1:47266, managementAddress=172.17.0.1, protocol=OF_14
karaf@root >                                                             02:01:44
```

Figure 28. Displaying the currently known devices (switches).

**Step 8.** To display the list of all currently known links, type the following command.

```
links
```


root@admin: /home/sdn/SDN_Labs/lab10
File Actions Edit View Help
sdn@admin: ~
root@admin: /home/sdn/SDN_Labs/lab10

```
karaf@root > links                                                                  02:03:58
src=of:0000000000000001/1, dst=of:0000000000000003/1, type=DIRECT, state=ACTIVE, expected=false
src=of:0000000000000001/2, dst=of:0000000000000004/1, type=DIRECT, state=ACTIVE, expected=false
src=of:0000000000000002/1, dst=of:0000000000000003/2, type=DIRECT, state=ACTIVE, expected=false
src=of:0000000000000002/2, dst=of:0000000000000004/2, type=DIRECT, state=ACTIVE, expected=false
src=of:0000000000000003/1, dst=of:0000000000000001/1, type=DIRECT, state=ACTIVE, expected=false
src=of:0000000000000003/2, dst=of:0000000000000002/1, type=DIRECT, state=ACTIVE, expected=false
src=of:0000000000000004/1, dst=of:0000000000000001/2, type=DIRECT, state=ACTIVE, expected=false
src=of:0000000000000004/2, dst=of:0000000000000002/2, type=DIRECT, state=ACTIVE, expected=false
karaf@root >                                                                        02:04:30
```

Figure 29. Displaying the currently known links.

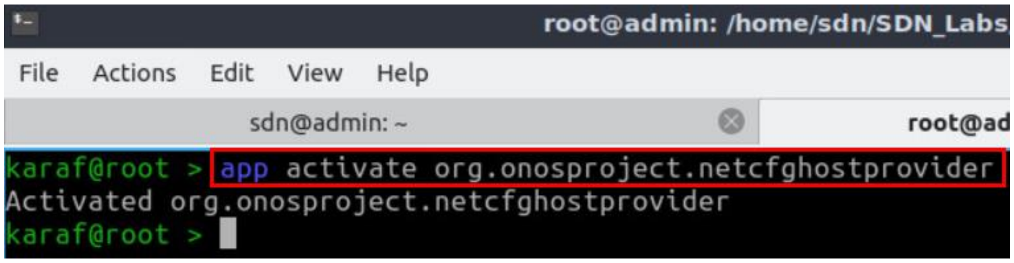
## 5 Applying load-balancing within the SDN network.

## 5.1 Activating the *segmentrouting* application

In this section, you will activate the *segmentrouting* application along with two applications it depends on, namely, *netcfghostprovider* and *netcfglinksprovider*. The latter two applications are used to configure host and link information in ONOS.

**Step 1.** Before activating the *segmentrouting* application you must start the *netcfghostprovider* application. In order to activate it, type the following command.

```
app activate org.onosproject.netcfghostprovider
```

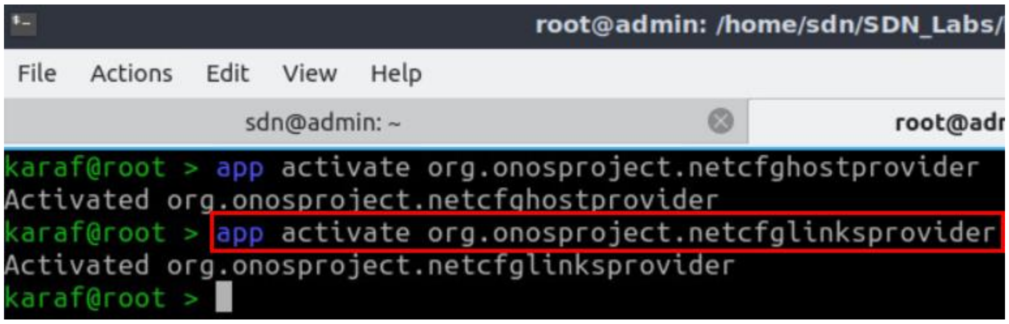


```
karaf@root > app activate org.onosproject.netcfghostprovider
Activated org.onosproject.netcfghostprovider
karaf@root >
```

Figure 30. Activating ONOS *netcfghostprovider* application.

**Step 2.** The *segmentrouting* application also depends on another application, namely, *netcfglinksprovider*. In order to activate it, type the following command.

```
app activate org.onosproject.netcfglinksprovider
```

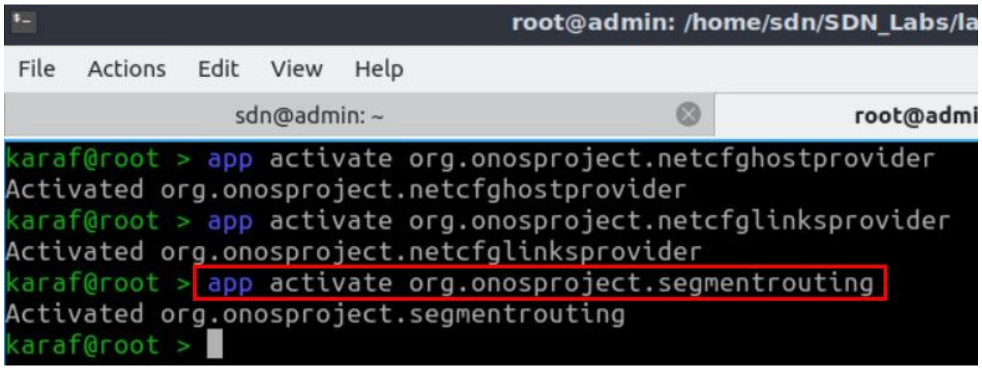


```
karaf@root > app activate org.onosproject.netcfghostprovider
Activated org.onosproject.netcfghostprovider
karaf@root > app activate org.onosproject.netcfglinksprovider
Activated org.onosproject.netcfglinksprovider
karaf@root >
```

Figure 31. Activating ONOS *netcfglinksprovider* application.

**Step 3.** Type the following command to activate the *segmentrouting* application.

```
app activate org.onosproject.segmentrouting
```

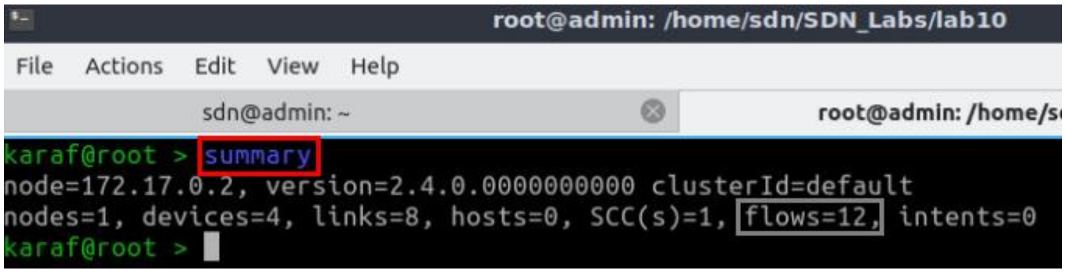

root@admin: /home/sdn/SDN_Labs/la
File Actions Edit View Help
sdn@admin: ~
root@admi

```
karaf@root > app activate org.onosproject.netcfghostprovider
Activated org.onosproject.netcfghostprovider
karaf@root > app activate org.onosproject.netcfglinksprovider
Activated org.onosproject.netcfglinksprovider
karaf@root > app activate org.onosproject.segmentrouting
Activated org.onosproject.segmentrouting
karaf@root >
```

Figure 32. Activating ONOS *segmentrouting* application.

**Step 4.** Type the following command to view a summary of ONOS and the current topology.

```
summary
```

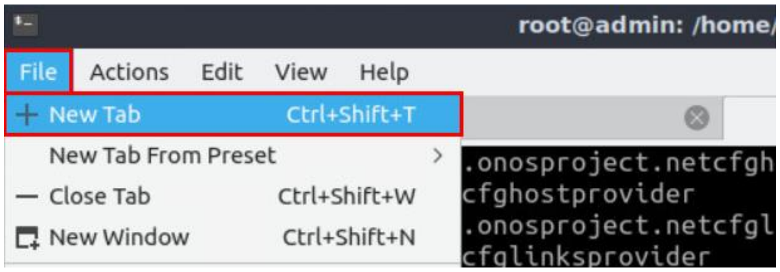

root@admin: /home/sdn/SDN_Labs/lab10
File Actions Edit View Help
sdn@admin: ~
root@admin: /home/s

```
karaf@root > summary
node=172.17.0.2, version=2.4.0.0000000000 clusterId=default
nodes=1, devices=4, links=8, hosts=0, SCC(s)=1, flows=12, intents=0
karaf@root >
```

Figure 33. Summary of ONOS and the current topology.

Consider Figure 33. ONOS shows that there is currently a total of 12 flows inserted in the switches. When *segmentrouting* application takes effect in later steps, it will insert more flows to the switches to achieve its capabilities.

**Step 5.** In the same Linux terminal, Click on *File>New Tab* to open an additional tab in the Linux terminal. Alternatively, you can press `Ctrl+Shift+T`.

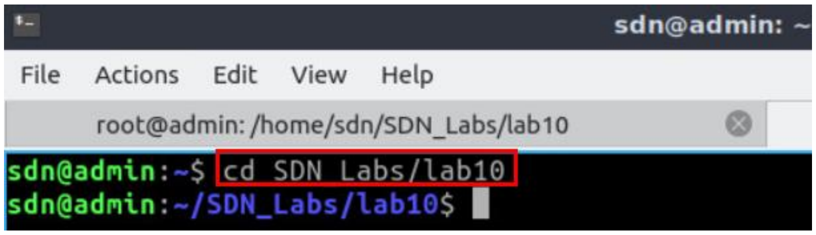


```
.onosproject.netcfgh
cfghostprovider
.onosproject.netcfgl
cfglinksprovider
```

Figure 34. Opening an additional tab.

**Step 6.** Navigate into *SDN_Labs/lab10* directory by issuing the following command.

```
cd SDN_Labs/lab10
```

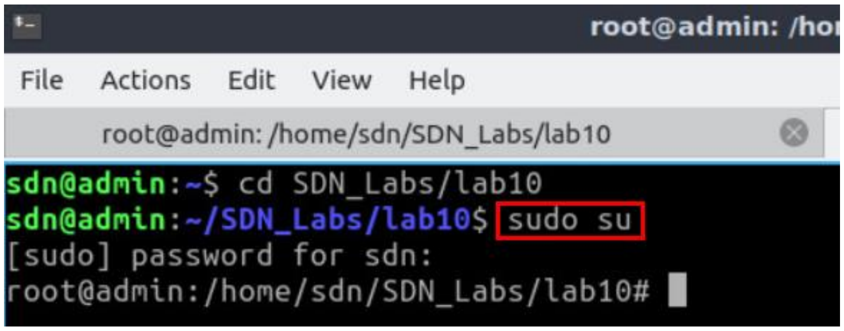

sdn@admin: ~
File Actions Edit View Help
root@admin: /home/sdn/SDN_Labs/lab10

```
sdn@admin:~$ cd SDN_Labs/lab10
sdn@admin:~/SDN_Labs/lab10$
```

Figure 35. Entering the *SDN_Labs/lab10* directory.

**Step 7.** Issue the command below to execute programs with the security privileges of the superuser (root). When prompted for a password, type `password`.

```
sudo su
```

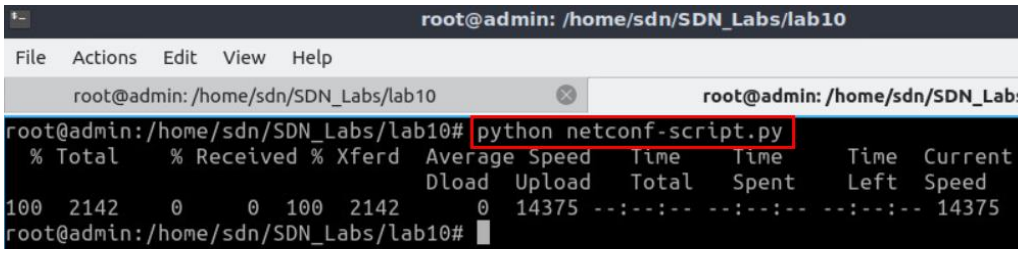

root@admin: /ho
File Actions Edit View Help
root@admin: /home/sdn/SDN_Labs/lab10

```
sdn@admin:~$ cd SDN_Labs/lab10
sdn@admin:~/SDN_Labs/lab10$ sudo su
[sudo] password for sdn:
root@admin:/home/sdn/SDN_Labs/lab10#
```

Figure 36. Switching to root mode.

**Step 8.** A script was written to perform two tasks. First, to load the network configuration needed for the segmentrouting application. Second, to connect the current topology with sFlow and visualize network traffic. In order to run the script, issue the following command.

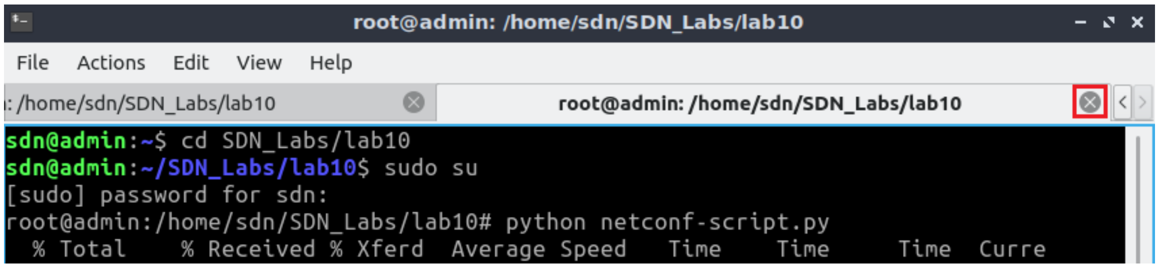
```
python netconf-script.py
```


root@admin: /home/sdn/SDN_Labs/lab10
File Actions Edit View Help
root@admin: /home/sdn/SDN_Labs/lab10
root@admin: /home/sdn/SDN_Lab

```
root@admin:/home/sdn/SDN_Labs/lab10# python netconf-script.py
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2142    0     0  100  2142      0  14375 --:--:-- --:--:-- --:--:-- 14375
root@admin:/home/sdn/SDN_Labs/lab10#
```

Figure 37. Executing the script to load the configuration and connect to sFlow.

**Step 9.** Close the current tab by clicking on ☒ as indicated in the figure below.


root@admin: /home/sdn/SDN_Labs/lab10
File Actions Edit View Help
: /home/sdn/SDN_Labs/lab10
root@admin: /home/sdn/SDN_Labs/lab10

```
sdn@admin:~$ cd SDN_Labs/lab10
sdn@admin:~/SDN_Labs/lab10$ sudo su
[sudo] password for sdn:
root@admin:/home/sdn/SDN_Labs/lab10# python netconf-script.py
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curre
```

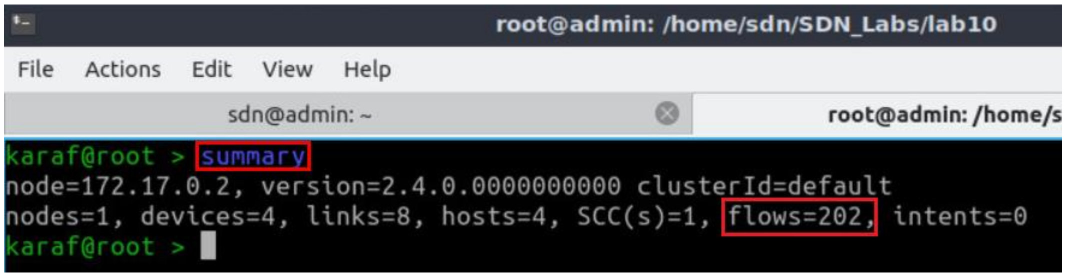
Figure 38. Closing a tab in the Linux terminal.

## 5.2 Verifying the loaded configuration

In this section, you will verify the flows inserted by the *segmentrouting* application and test some of its available commands.

**Step 1.** In the ONOS terminal, write the following command to view a summary of ONOS and the current topology.

```
summary
```



File Actions Edit View Help

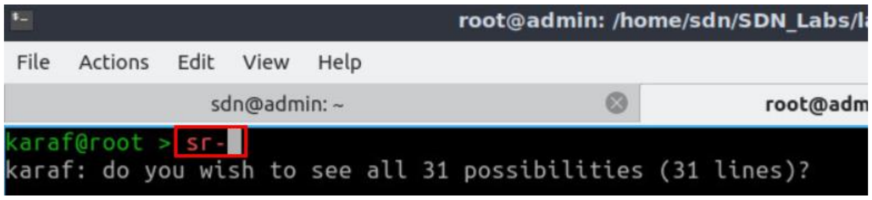
```
karaf@root > summary
node=172.17.0.2, version=2.4.0.0000000000 clusterId=default
nodes=1, devices=4, links=8, hosts=4, SCC(s)=1, flows=202, intents=0
karaf@root >
```

Figure 39. Summary of ONOS and the current topology.

Consider Figure 39. After loading the network configuration into ONOS, the *segmentrouting* application takes effect and the flows are inserted into the switches.

**Step 2.** The *segmentrouting* application offers a number of commands. To view these commands, type the following command followed by `TAB`.

```
sr-
```

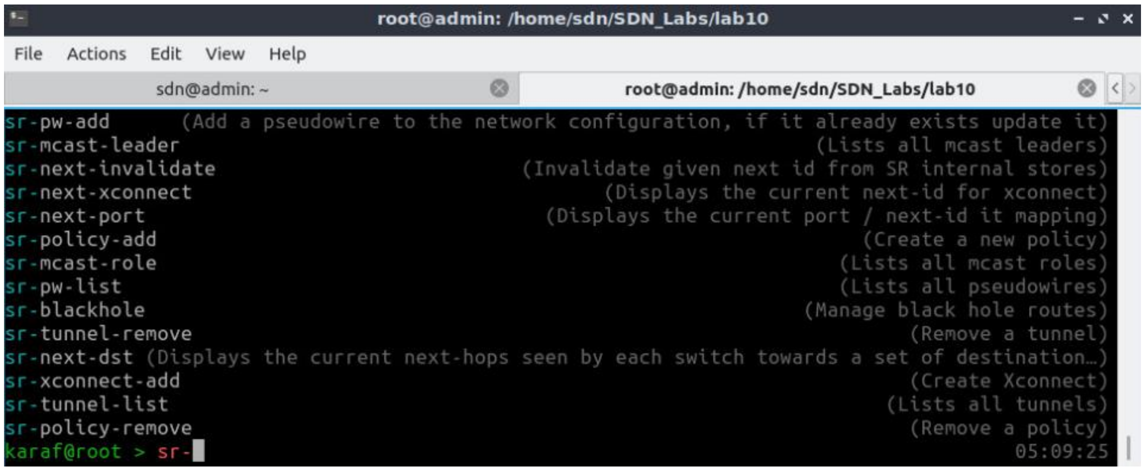


File Actions Edit View Help

```
karaf@root > sr-
karaf: do you wish to see all 31 possibilities (31 lines)?
```

Figure 40. Displaying the commands offered by *segmentrouting* application.

**Step 3.** To confirm displaying all 31 possibilities, click the `TAB` key one more time. This will display all ONOS commands that start with `sr-` on the left-hand side of the CLI, as well as their explanation on the right-hand side of the CLI.

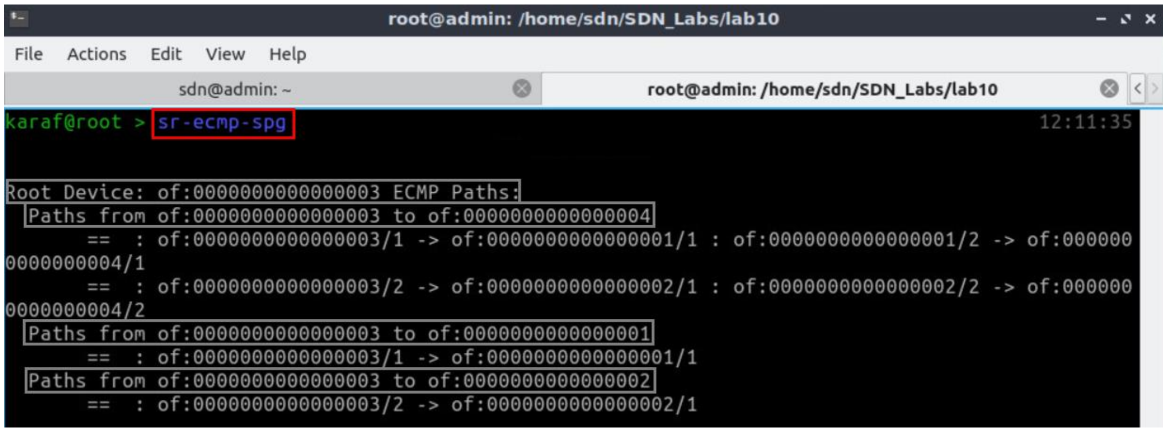

root@admin: /home/sdn/SDN_Labs/lab10
File Actions Edit View Help
sdn@admin: ~
root@admin: /home/sdn/SDN_Labs/lab10

```
sr-pw-add      (Add a pseudowire to the network configuration, if it already exists update it)
sr-mcast-leader                                                       (Lists all mcast leaders)
sr-next-invalidate                                 (Invalidate given next id from SR internal stores)
sr-next-xconnect                                           (Displays the current next-id for xconnect)
sr-next-port                                       (Displays the current port / next-id it mapping)
sr-policy-add                                                               (Create a new policy)
sr-mcast-role                                                              (Lists all mcast roles)
sr-pw-list                                                                 (Lists all pseudowires)
sr-blackhole                                                            (Manage black hole routes)
sr-tunnel-remove                                                                 (Remove a tunnel)
sr-next-dst (Displays the current next-hops seen by each switch towards a set of destination…)
sr-xconnect-add                                                                 (Create Xconnect)
sr-tunnel-list                                                                 (Lists all tunnels)
sr-policy-remove                                                                 (Remove a policy)
karaf@root > sr-                                                                         05:09:25
```

Figure 41. Displaying the commands offered by *segmentrouting* application.

**Step 4.** To view a list of all possible paths between the switches, type the following command.

```
sr-ecmp-spg
```


root@admin: /home/sdn/SDN_Labs/lab10
File Actions Edit View Help
sdn@admin: ~
root@admin: /home/sdn/SDN_Labs/lab10

```
karaf@root > sr-ecmp-spg                                                                 12:11:35


Root Device: of:0000000000000003 ECMP Paths:
  Paths from of:0000000000000003 to of:0000000000000004
       ==  : of:0000000000000003/1 -> of:0000000000000001/1 : of:0000000000000001/2 -> of:000000
0000000004/1
       ==  : of:0000000000000003/2 -> of:0000000000000002/1 : of:0000000000000002/2 -> of:000000
0000000004/2
  Paths from of:0000000000000003 to of:0000000000000001
       ==  : of:0000000000000003/1 -> of:0000000000000001/1
  Paths from of:0000000000000003 to of:0000000000000002
       ==  : of:0000000000000003/2 -> of:0000000000000002/1
```

Figure 42. Displaying a list of all possible paths between the switches.

Consider Figure 42. The command displays the ID of all the visible switches in the topology, i.e., switches s1, s2, s3, and s4, for each one, it displays all possible paths with other switches. For instance, switch s3 is connected to switch s4 via two different paths, and to switches s1 and s2 via one path.

## 6 Testing the load balancer within the SDN network

In this section, you will verify the load balancing feature supported by the *segmentrouting* application. In particular, you will launch several flows between two hosts and visualize (using sFlow) how these flows are balanced between the available links. To do that, you will use iPerf3, a tool for active measurements. iPer3 supports various features, such as establishing a Transmission Control Protocol (TCP) or User Datagram Protocol (UDP), specifying the sending rate, and launching parallel flows[4].

**Step 1.** Click on the opened Firefox browser.

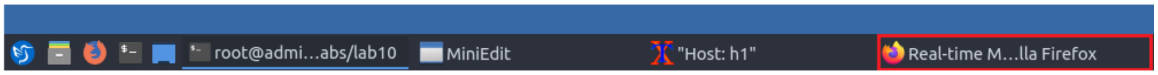


Figure 43. Opening Firefox browser.

**Step 2.** In sFlow dashboard, click on the Topology tab to display the current Mininet topology.

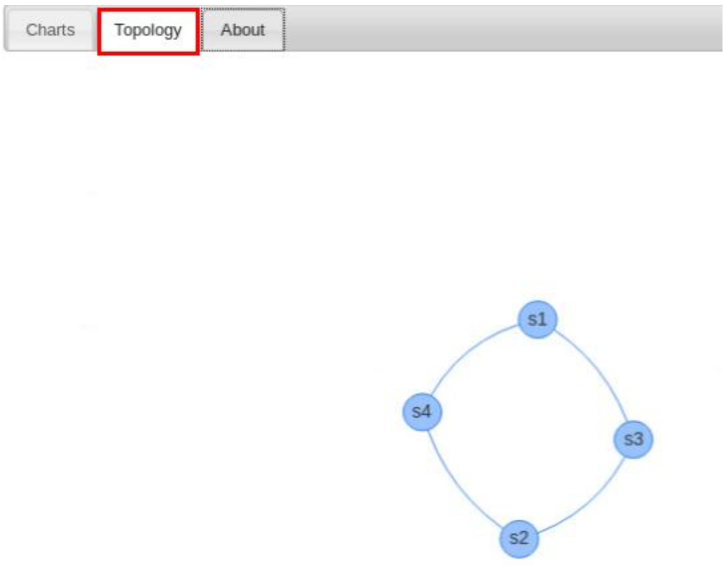


Figure 44. Visualizing the topology using sFlow.

Consider Figure 44. After running the script `netconf-script.py`, sFlow will recognize the topology in Mininet. As a result, the four switches will be displayed with the links connecting them.

**Step 3.** Go back to MiniEdit.

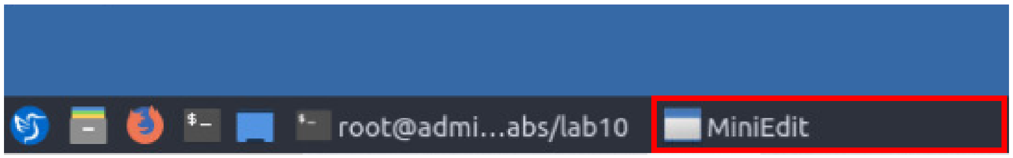


Figure 45. Opening MiniEdit.

**Step 4**. Hold right-click on host h3 and select *Terminal*. This opens the terminal of host h3 and allows the execution of commands on that host.

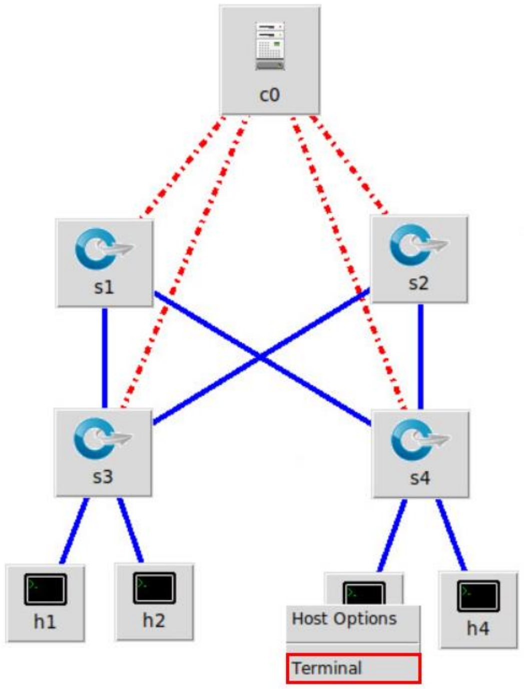
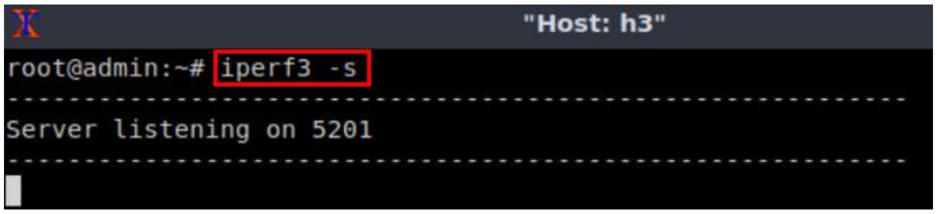


Figure 46. Opening a terminal on host h3.

**Step 5.** To launch iPerf3 in server mode, run the command `iperf3 -s` on host h3 terminal as shown in the figure below.

```
iperf3 -s
```



```
"Host: h3"
root@admin:~# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Figure 47. Running iPerf3 server on host h3.

Consider Figure 47. The parameter `-s` in the command indicates that the host is configured as a server. Now, the server is listening on port 5201 waiting for incoming connections.

**Step 6.** Open host h1 terminal as shown in the below figure.

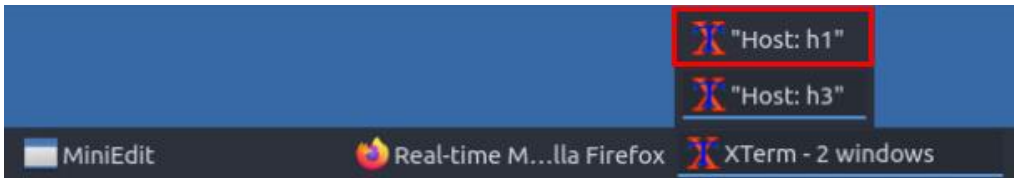
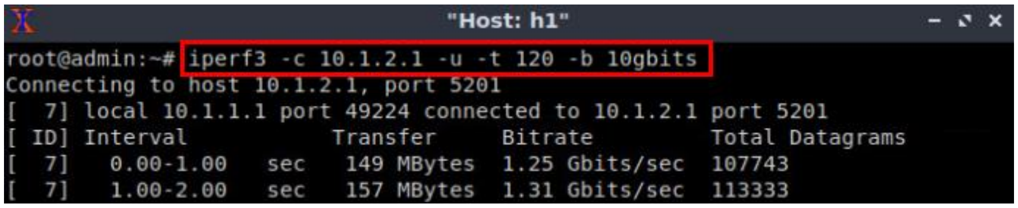


Figure 48. Opening host h1 terminal.

**Step 7.** To launch iPerf3 in client mode, run the command `iperf3 -c` on host h1 terminal as shown in the figure below.

```
iperf3 -c 10.1.2.1 -u -t 120 -b 10gbits
```



```
"Host: h1"
root@admin:~# iperf3 -c 10.1.2.1 -u -t 120 -b 10gbits
Connecting to host 10.1.2.1, port 5201
[  7] local 10.1.1.1 port 49224 connected to 10.1.2.1 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  7]   0.00-1.00   sec   149 MBytes  1.25 Gbits/sec  107743
[  7]   1.00-2.00   sec   157 MBytes  1.31 Gbits/sec  113333
```

Figure 49. Running iPerf3 client on host h1.

Consider Figure 49. The parameter `-c` in the command above indicates that the host is configured as a client. The parameter `10.1.2.1` is the server's (host h3) IP address. The parameter `-u` is to specify UDP traffic. The parameter `-t` specifies the time interval (in seconds) of the iPerf3 test. The parameter `-b` sets the bandwidth (sending rate).

**Step 8.** Click on the opened Firefox browser.

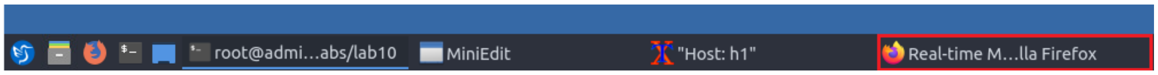


Figure 50. Opening Firefox browser.

**Step 9.** Visualize the traffic in Mininet using sFlow.

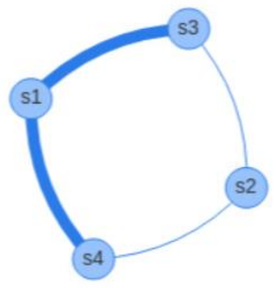


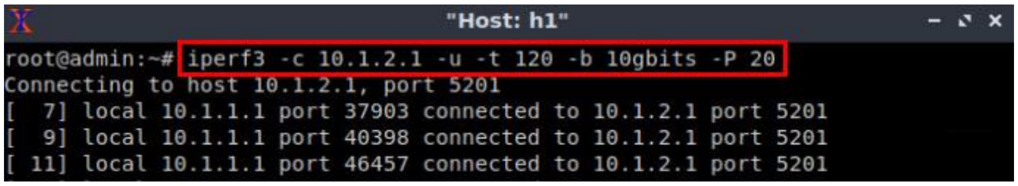
Figure 51. Visualizing the traffic using sFlow.

Consider Figure 51. Traffic passing from host h1 to host h3 will take the path s3 -> s1 -> s4 and finally reach host h3 (thick blue line). There is no load balancing since there is only one active flow.

Note that you can interrupt the iPerf3 test on host h1 terminal by pressing `Ctrl+c`, or wait until it is done.

**Step 10.** On host h1 terminal, perform the same iPerf3 test as in step 8 while increasing the number of parallel flows. The *segmentrouting* application will apply per-flow load balancing.

```
iperf3 -c 10.1.2.1 -u -t 120 -b 10gbits -P 20
```


```
root@admin:~# iperf3 -c 10.1.2.1 -u -t 120 -b 10gbits -P 20
Connecting to host 10.1.2.1, port 5201
[  7] local 10.1.1.1 port 37903 connected to 10.1.2.1 port 5201
[  9] local 10.1.1.1 port 40398 connected to 10.1.2.1 port 5201
[ 11] local 10.1.1.1 port 46457 connected to 10.1.2.1 port 5201
```
Figure 52. Running iPerf3 client on host h1.

Consider Figure 52. The parameter `-P` specifies the number of parallel client threads.

**Step 11.** Click on the opened Firefox browser to view sFlow dashboard.

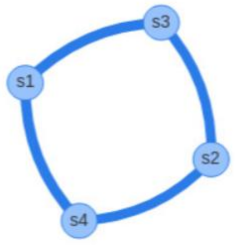


Figure 53. Visualizing the traffic using sFlow.

Consider Figure 53. The 20 parallel flows are running between hosts h1 and h3. This requires the *segmentrouting* application to balance the traffic (per-flow) between all the paths from switch s3 to switch s4. On average, half of the flows will follow one path, while the other half will follow the other path. Thus, traffic will flow in two directions, 1- s3 -> s1 -> s4, and 2- s3 -> s2 -> s4.

This concludes Lab 10. Stop the emulation and then exit out of MiniEdit and Linux terminal.

## References

1. A. Tanenbaum, D. Wetherall, “*Computer networks*”, 5th Edition, Pearson, 2012.
2. Noction, “BGP and equal-cost multipath (ECMP)”. [Online]. Available: https://www.noction.com/blog/equal-cost-multipath-ecmp
3. C. Hopps, “Analysis of an equal-cost multi-path algorithm”. RFC 2992, 2000. [Online]. Available: https://www.hjp.at/doc/rfc/rfc2992.html
4. iPerf, “iPerf - The ultimate speed test tool for TCP, UDP and SCTP”. [Online]. Available: https://iperf.fr/
5. sFlow, “About sFlow”. [Online]. Available: https://sflow.org/about/index.php