



UNIVERSITY OF
SOUTH CAROLINA

SOFTWARE DEFINED NETWORKING

Lab 5: Configuring VXLAN to Provide Network Traffic Isolation

Document Version: **05-25-2020**



Award 1829698

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

Contents

Overview	3
Objectives.....	3
Lab settings	3
Lab roadmap	3
1 Introduction	3
1.1 VXLAN architecture	4
1.2 VXLAN packet format	5
2 Lab topology.....	5
2.1 Lab settings.....	6
2.2 Loading a topology	6
2.3 Load the configuration file	8
2.4 Run the emulation.....	9
2.5 Verify the configuration	10
3 Configuring OSPF on routers r1 and r2.....	12
4 Configuring VXLAN	15
4.1 Run Mininet instances within the containers	16
4.2 Adding entries to the switches' flow tables.....	18
5 Verifying configuration	20
5.1 Performing connectivity test between end-hosts	20
5.2 Verifying VXLAN network identifiers by using Wireshark.....	21
References	28

Overview

This lab presents Virtual eXtensible Local Area Network (VXLAN), a network virtualization scheme that provides a solution for the scalability problems associated with datacenter and large cloud computing deployments. This lab aims to configure VXLAN to isolate network traffic within an emulated environment. Additionally, the user will inspect the packets to identify the fields corresponding to VXLAN network identifiers.

Objectives

By the end of this lab, the user will:

1. Understand the concept of VXLAN.
2. Assign IP addresses to a router interface.
3. Configure a routing protocol.
4. Emulate servers by using docker containers.
5. Push flow tables to configure VXLAN in a switch.
6. Isolate network traffic by using VXLAN.
7. 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 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 services⁷. Those services require on-demand elastic provisioning of computing resources for multi-tenant environments. Such feature is supported by network virtualization, which provide an efficient way to host multiple tenants in the same server and traffic isolation, to avoid a tenant to have access to the data of another tenant.

Isolating network traffic could be done via Layer 2 or Layer 3 networks. For Layer 2 networks, VLANs are often used to segregate traffic. In such scenario a tenant could be identified by its own VLAN. However, VLAN-based network isolation suffers a limitation of 4094 VLANs, which is inadequate considering the high demand of cloud services. Additionally, a tenant could require multiple VLANs, which exacerbates the issue.

On the other hand, layer 3 networks do not provide an extensive solution for multi-tenant networks as well. Two tenants might use the same set of Layer 3 addresses within their networks, which requires the cloud provider to provide isolation in some other form. Further, requiring all tenants to use IP excludes customers relying on direct Layer 2 or non-IP Layer 3 protocols for inter VM communication.

1.1 VXLAN architecture

Hypervisor-based overlay networks is a novel use of Software-defined Network (SDN) capabilities³. This concept does not modify the physical network, which means that networking devices and their configurations remains unchanged. Hypervisor-based virtualized networks are built above such network⁵. The system at the edge of the network works as an interface to these virtual networks. In these networks, many details of the physical network from the devices that connect to the overlays are hidden.

VXLAN (Virtual eXtensible Local Area Network) addresses the above requirements of Layer 2 and Layer 3 data center network infrastructure in the presence of Virtual Machines (VMs) in a multi-tenant environment. VXLAN runs over the existing networking infrastructure and provides a means to increase the number of devices on a Layer 2 network. In summary, VXLAN is a Layer 2 overlay scheme build on the top of a Layer 3 network. Each overlay is unique within the tenant domain and is known as VXLAN segment. The communication is restricted just between VMs within the same VXLAN segment. Each VXLAN segment is identified by a 24-bit segment ID, called the VXLAN Network Identifier (VNI). This allows up to 16 million (2^{24}) VXLAN segments to coexist within the same administrative domain.

Consider Figure 1. VXLAN could be considered as a tunneling scheme to overlay Layer 2 networks on top of Layer 3 networks⁶. The tunnels are stateless, so each segment is encapsulated according to a set of rules. The end point of the tunnel (VXLAN Tunnel End Point or VTEP) is located within the hypervisor on the server that hosts the VM. The traffic is isolated according to the VNI and the end-hosts can communicate as they are located within the same network.

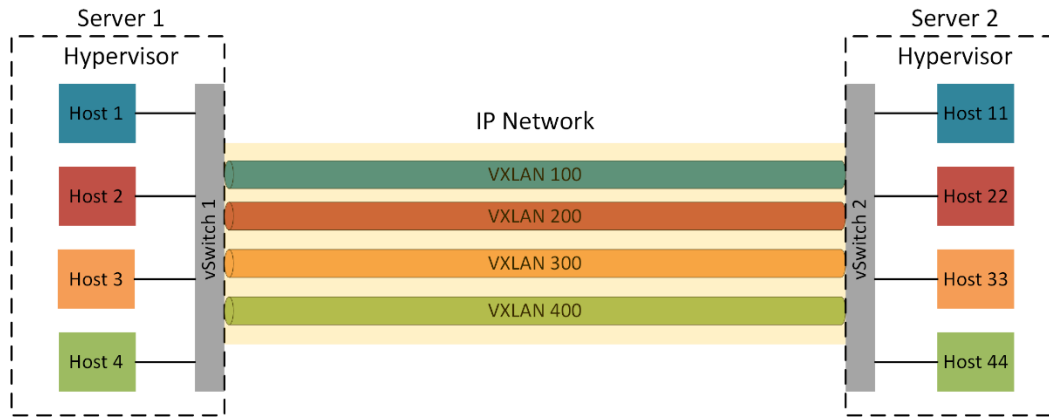


Figure 1. VXLAN overview.

1.2 VXLAN packet format

Figure 2 illustrates the format of a VXLAN packet⁵. The outer header contains the MAC and IP addresses appropriate for sending a unicast packet to the destination switch, acting as a virtual tunnel end point. The VXLAN header follows the outer header and contains a VXLAN Network Identifier of 24 bits in length, sufficient for about 16 million networks.

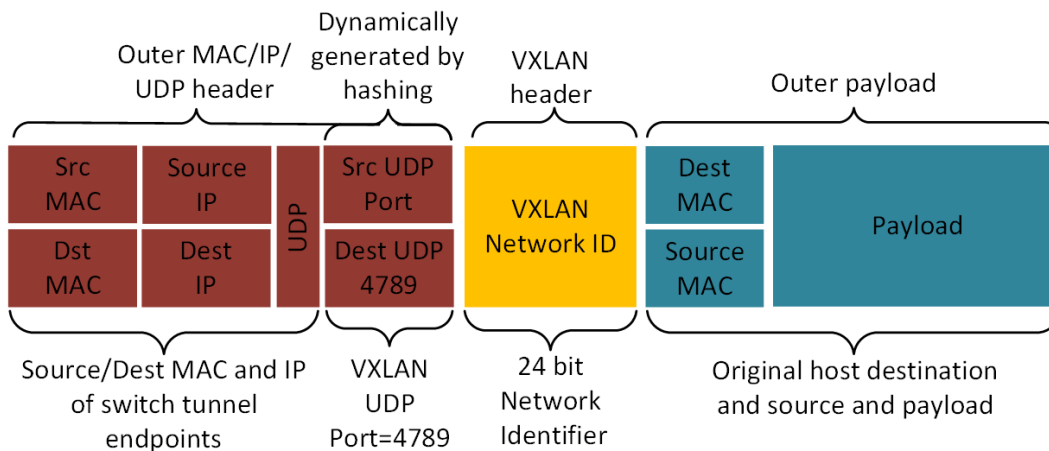


Figure 2. VXLAN packet format.

2 Lab topology

Consider Figure 3. The topology consists of four end-hosts, two switches and two routers. The end hosts and switches are running inside Server 1 and Server 2. Those servers are implemented by Docker⁸ containers which run Mininet instances. Router r1 and router r2 are supported by Free-range Routing (FRR) engine.

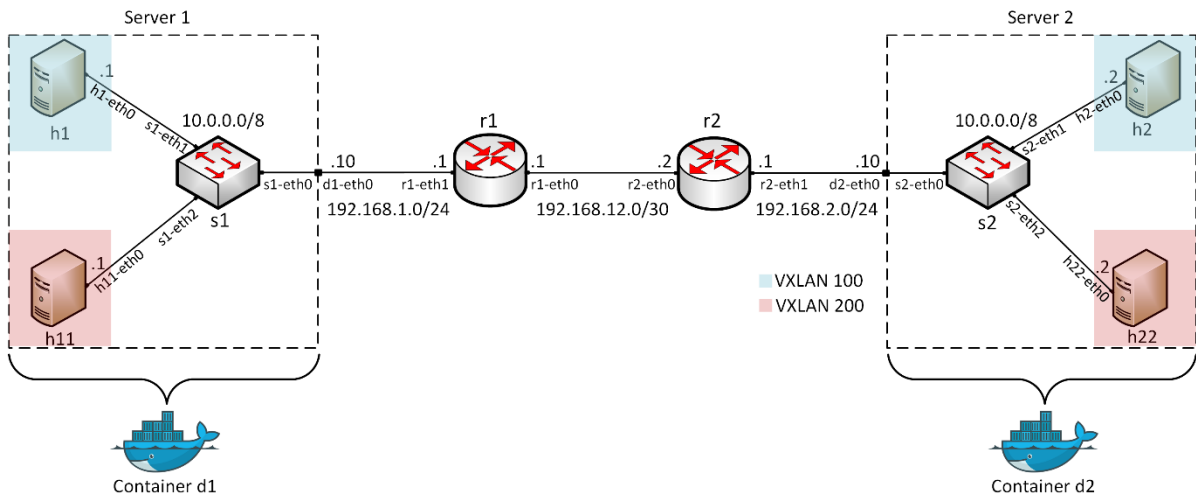


Figure 3. Lab topology.

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	192.168.12.1	/30
	r1-eth1	192.168.1.1	/24
r2	r2-eth0	192.168.12.2	/30
	r2-eth1	192.168.2.1	/24
h1	h1-eth0	10.0.0.1	/8
h11	h11-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8
h22	h22-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 a topology

In this section, the user will open MiniEdit and load the lab topology. MiniEdit provides a Graphical User Interface (GUI) that facilitates the creation and emulation of network topologies in Mininet. This tool has additional capabilities such as: configuring network elements (i.e IP addresses, default gateway), 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`.

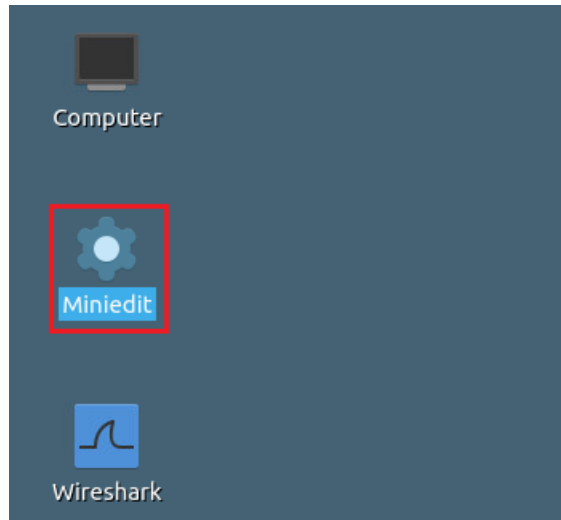


Figure 4. MiniEdit shortcut.

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*.

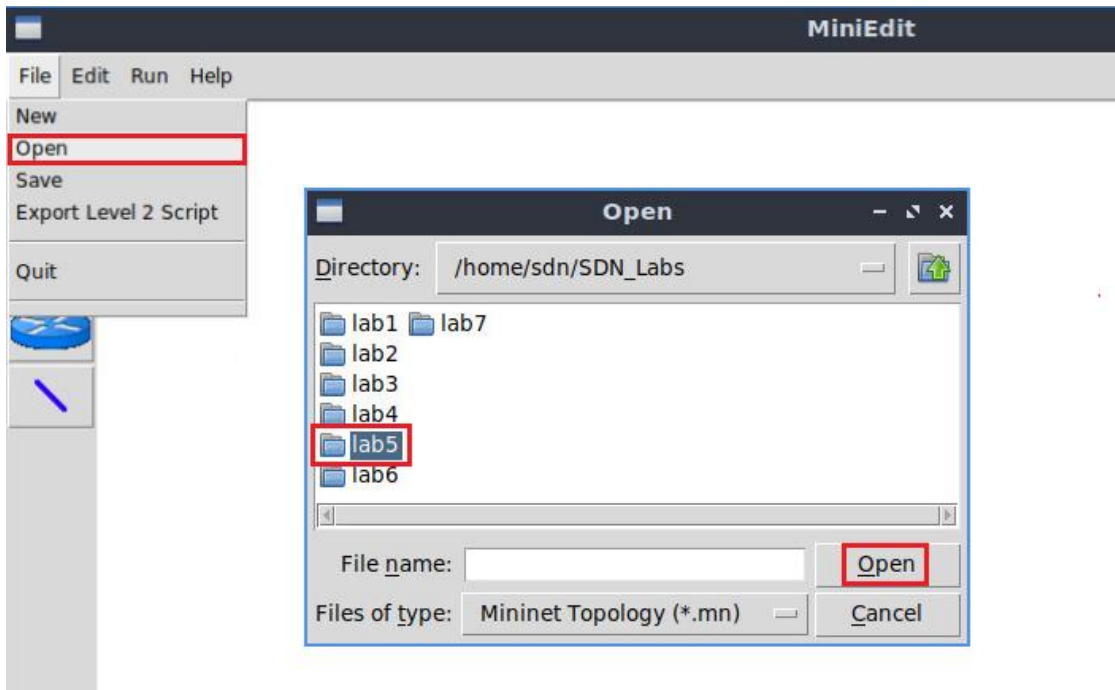


Figure 5. Opening topology.

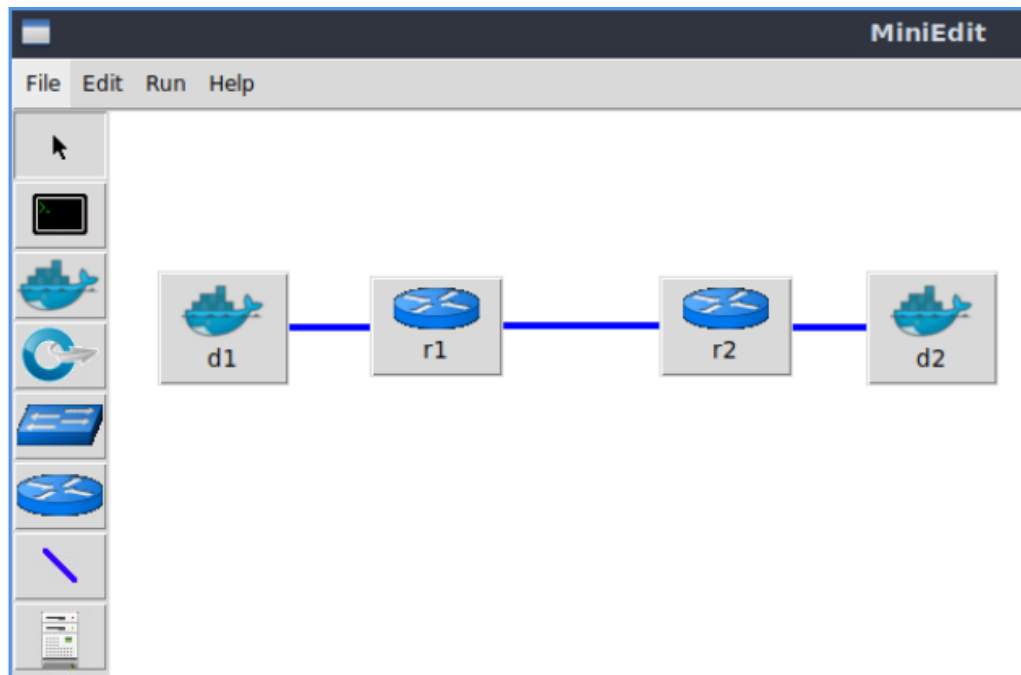


Figure 6. MiniEdit's topology.

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 devices' interfaces, you will execute a script that loads the configuration to the routers and end devices.

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

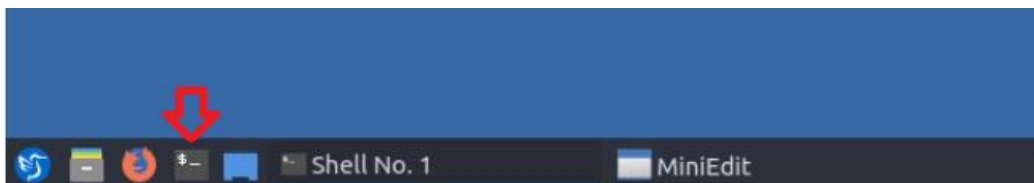
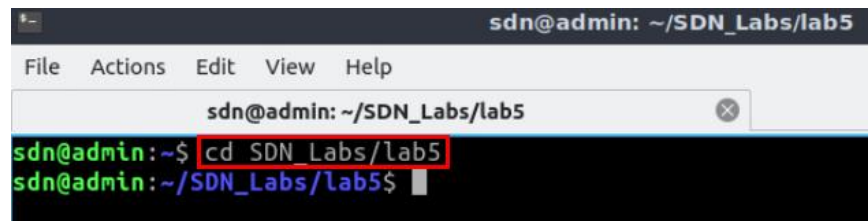


Figure 7. Opening Linux terminal.

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 routers' interfaces. The `cd` command is short for change directory followed by an argument that specifies the destination directory.

```
cd SDN_Labs/lab5
```

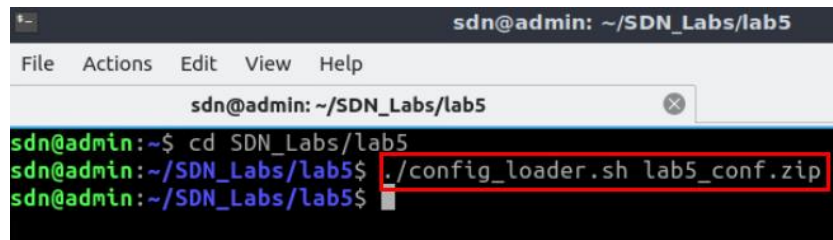



```
sdn@admin: ~/SDN_Labs/lab5
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab5
sdn@admin:~$ cd SDN_Labs/lab5
sdn@admin:~/SDN_Labs/lab5$
```

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
```

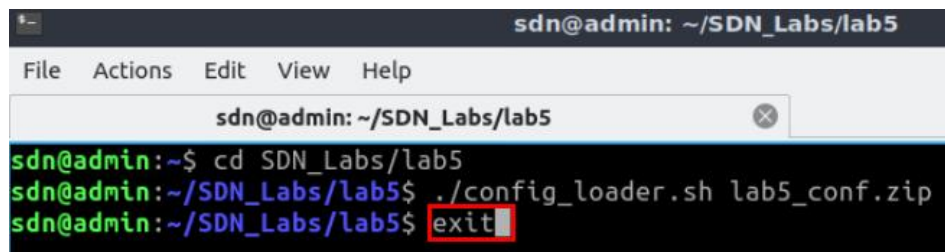


```
sdn@admin: ~/SDN_Labs/lab5
File Actions Edit View Help
sdn@admin: ~/SDN_Labs/lab5
sdn@admin:~$ cd SDN_Labs/lab5
sdn@admin:~/SDN_Labs/lab5$ ./config_loader.sh lab5_conf.zip
sdn@admin:~/SDN_Labs/lab5$
```

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 on the *Run* button located in 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
```

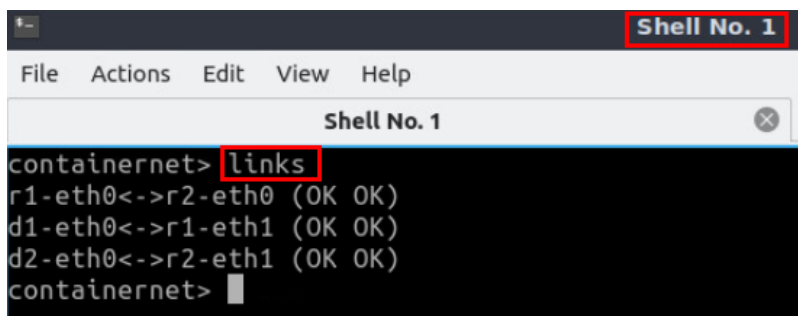


Figure 12. Displaying network interfaces.

In Figure 12, the link displayed within the gray box indicates that interface *eth0* of router *r1* connects to interface *eth0* of router *r2* (i.e., *r1-eth0<->r2-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*.

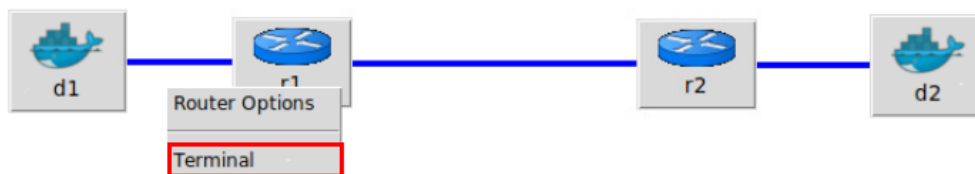


Figure 13. Opening a terminal on router *r1*.

Step 2. In this step, you will start 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
```

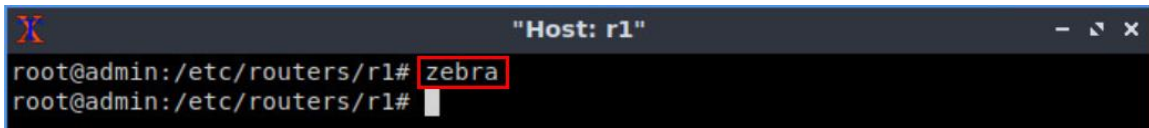


Figure 14. Starting zebra daemon.

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
```

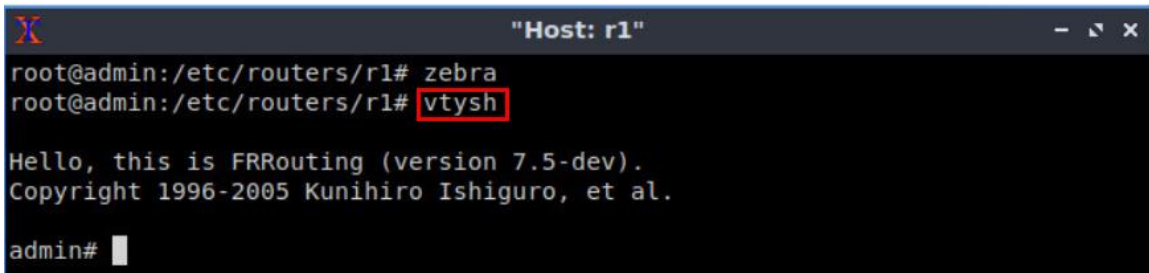


Figure 15. Starting vtysh on router r1.

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
```

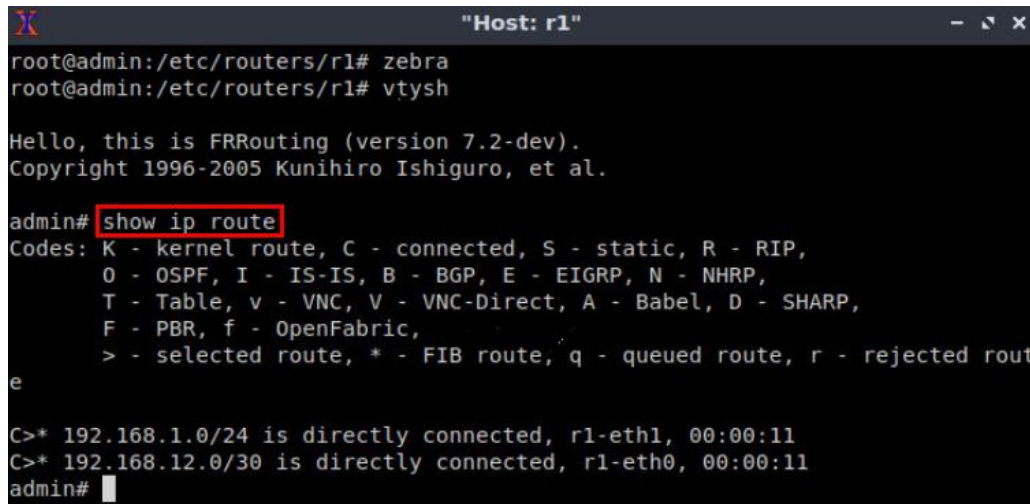


Figure 16. Displaying routing table of router r1.

The output in the figure above shows that the networks 192.168.1.0/24 and 192.168.12.0/30 are directly connected through the interfaces *r1-eth1* and *r1-eth0*, respectively.

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

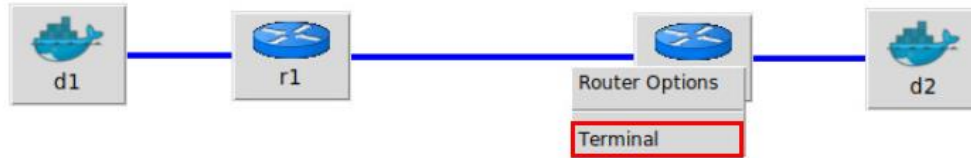


Figure 17. Opening a terminal on router r2.

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. At the end, you will verify all the directly connected networks of router r2.

```

Host: r2
root@admin:/etc/routers/r2# zebra
root@admin:/etc/routers/r2# vtysh

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

admin# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       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-eth1, 00:00:04
C>* 192.168.12.0/30 is directly connected, r2-eth0, 00:00:04
admin#
    
```

Figure 18. Displaying routing table of router r2.

3 Configuring OSPF on routers r1 and r2

In this section, you will configure OSPF routing protocol in router r1 and router r2. First, you will enable the OSPF daemon on routers r3 and r4. 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 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 19. Exiting the vtysh session.

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

```
ospfd
```

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

Figure 20. Starting OSPF daemon.

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

```
vttysh
```

```
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 21. Star Starting vtyshon router r1.

Step 4. To enable router r1 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 22. Enabling configuration mode on router r1.

Step 5. In order to configure 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
```

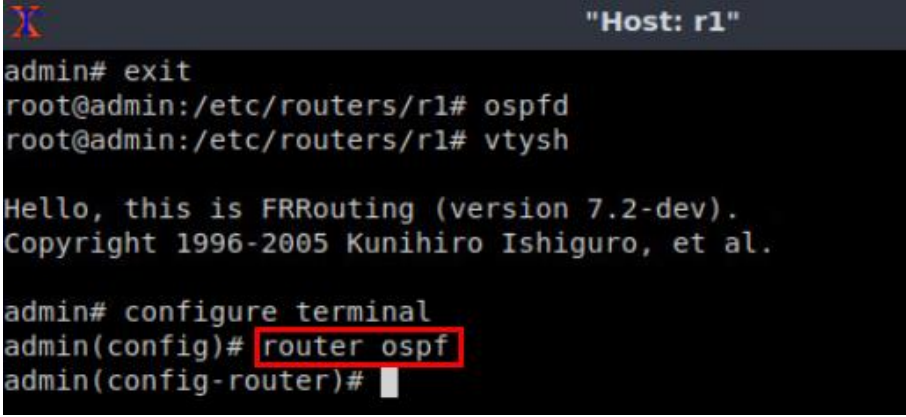
A terminal window titled "Host: r1" showing the configuration process. The user enters 'admin# exit', then 'root@admin:/etc/routers/r1# ospfd', and 'root@admin:/etc/routers/r1# vtysh'. A message displays: 'Hello, this is FRRouting (version 7.2-dev). Copyright 1996-2005 Kunihiro Ishiguro, et al.' The user then enters 'admin# configure terminal', followed by 'admin(config)# router ospf', which is highlighted with a red box. The prompt changes to 'admin(config-router)#'.

Figure 23. Configuring OSPF on router r1.

Step 6. In this step, you will enable all the interfaces of router r1 to participate in the OSPF routing process, i.e., all the attached networks will be advertised to OSPF neighbors. The advertised networks are associated with area 0. To advertise all connected networks in the same command, the network 0.0.0.0/0 will be used. This network address matches all IP addresses.

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

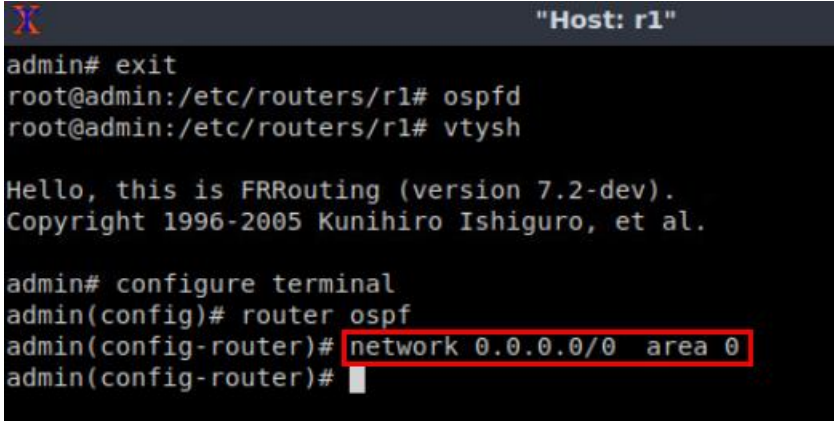
A terminal window titled "Host: r1" showing the configuration process. The user enters 'admin# exit', then 'root@admin:/etc/routers/r1# ospfd', and 'root@admin:/etc/routers/r1# vtysh'. A message displays: 'Hello, this is FRRouting (version 7.2-dev). Copyright 1996-2005 Kunihiro Ishiguro, et al.' The user then enters 'admin# configure terminal', followed by 'admin(config)# router ospf', and 'admin(config-router)# network 0.0.0.0/0 area 0', which is highlighted with a red box. The prompt changes to 'admin(config-router)#'.

Figure 24. 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
```

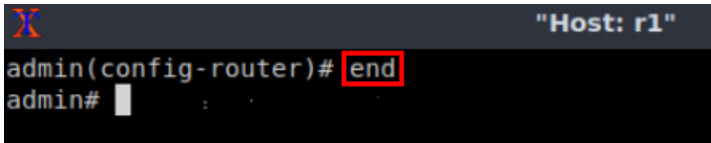
A terminal window titled "Host: r1" showing the user entering 'admin(config-router)# end', which is highlighted with a red box. The prompt changes to 'admin#'.

Figure 25. 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 26. Exiting from the configuration mode.

Step 9. Type the following command to verify the routing table of router r1.

```

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

O> 192.168.1.0/24 [110/10] is directly connected, r1-eth1, 00:18:17
C>* 192.168.1.0/24 is directly connected, r1-eth1, 01:07:34
O>* 192.168.2.0/24 [110/20] via 192.168.12.2, r1-eth0, 00:06:27
O  192.168.12.0/30 [110/10] is directly connected, r1-eth0, 00:18:17
C>* 192.168.12.0/30 is directly connected, r1-eth0, 01:07:34
admin#
    
```

Figure 27. Verifying the routing table of router r1.

Consider Figure 27. The network 192.168.2.0/24 is learned via OSPF (O>*) and it is reachable via the next hop 192.168.12.2 (route r2).

4 Configuring VXLAN

In this section, the user will start the networks within the containers d1 and d2. Both containers run a Mininet topology as depicted in Figure 3. In container d1, the topology consists in two hosts (h1 and h11) connected to a switch (s1). Similarly, container d2 runs a topology with two hosts (h2 and h22) 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 multitenant 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, h11, h2 and h22 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. In container d1 terminal, type the following python script to start a Mininet instance that consists of two hosts connected to a switch.

```
python start_server1.py
```

```

root@d1:~# python start_server1.py
* Starting ovsdb-server
* Configuring Open vSwitch system IDs
* Starting ovs-vswitchd
* Enabling remote OVSDB managers
*** Error setting resource limits. Mininet's performance may be affected.
*** Adding controller
*** Adding hosts
*** Adding switch
*** Creating links
*** Starting network
*** Configuring hosts
h1 h11
*** 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 ', h11.name, 'has IP address = 10.0.0.1 and MAC address = 00:00:00:00:00:01
*** Running CLI
*** Starting CLI:

```

Figure 28. Starting a Mininet instance within container d1.

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

Notice that host h1 and host h11 have the same IP addresses and MAC addresses. These hosts will be isolated b using VXLAN.

Step 2. 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)
h11-eth0<->s1-eth2 (OK OK)
mininet>

```

Figure 29. 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 h11 is connected to the switch s1 (*h11-eth0<->s1-eth2*).

Step 3. Similarly, in container d2 terminal, type the following python script to start a Mininet instance that consists in two 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
h2 h22
*** Starting controller

*** Starting 1 switches
s2 ..
Host h2 has IP address = 10.0.0.2 and MAC address = 00:00:00:00:00:02
Host h22 has IP address = 10.0.0.2 and MAC address = 00:00:00:00:00:02
*** Running CLI
*** Starting CLI:
mininet>

```

Figure 30. Starting a Mininet instance within container d2.

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

Notice that host h2 and host h22 have the same IP addresses and MAC addresses. These hosts will be isolated b using VXLAN.

Step 4. In container d2 terminal, run the following command to verify the devices in the topology:

```
links
```

```

X root@d2: ~
mininet> links
h2-eth0<->s2-eth1 (OK OK)
h22-eth0<->s2-eth2 (OK OK)
mininet>

```

Figure 31. Verifying the links between the devices in container d2.

The figure above shows that the host h2 and switch s2 are connected via the interface pair *h2-eth0<->s2-eth1*. Similarly, host h22 is connected to the switch s2 (*h22-eth0<->s2-eth2*).

4.2 Adding entries to the switches' flow tables

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 h2 and from h11 to h22.

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

```
sh cat flow1.txt | nl
```

```

X 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,dl_dst=00:00:00:00:00:01,actions=output:1
 5 table=1,tun_id=200,dl_dst=00:00:00:00:00:01,actions=output:2
 6 table=1,tun_id=100,dl_dst=00:00:00:00:00:02,actions=output:10
 7 table=1,tun_id=200,dl_dst=00:00:00:00:00:02,actions=output:10
 8 table=1,tun_id=100,arp,nw_dst=10.0.0.1,actions=output:1
 9 table=1,tun_id=200,arp,nw_dst=10.0.0.1,actions=output:2
10 table=1,tun_id=100,arp,nw_dst=10.0.0.2,actions=output:10
11 table=1,tun_id=200,arp,nw_dst=10.0.0.2,actions=output:10
12 table=1,priority=100,actions=drop
mininet>

```

Figure 32. Flow table in container d1.

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

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

```

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

```

Figure 33. Adding flow entries to switch s1.

Step 3. In this step, you will configure a VXLAN tunnel endpoint (VTEP) that will enable outgoing traffic from switch s1 to the outer network. A script is written to facilitate this process. To execute the script, type the following command.

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

```

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

```

Figure 34. Enabling outgoing traffic in switch s1.

VTEP is the device responsible for encapsulating and de-encapsulating layer 2 traffic. This device is the connection between the overlay and the underlay network. In this case, the VTEP is configured to provide connectivity between the switches and the containers' egress interfaces.

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
```

```

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

```

Figure 35. Adding flow entries to switch s2.

Step 5. Similarly, in container d2, type the command below to configure a VXLAN tunnel endpoint (VTEP) in order that enables outgoing traffic from switch s2 to the outer network, issue the following command:

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

```

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

```

Figure 36. Enabling outgoing traffic in switch s2.

5 Verifying configuration

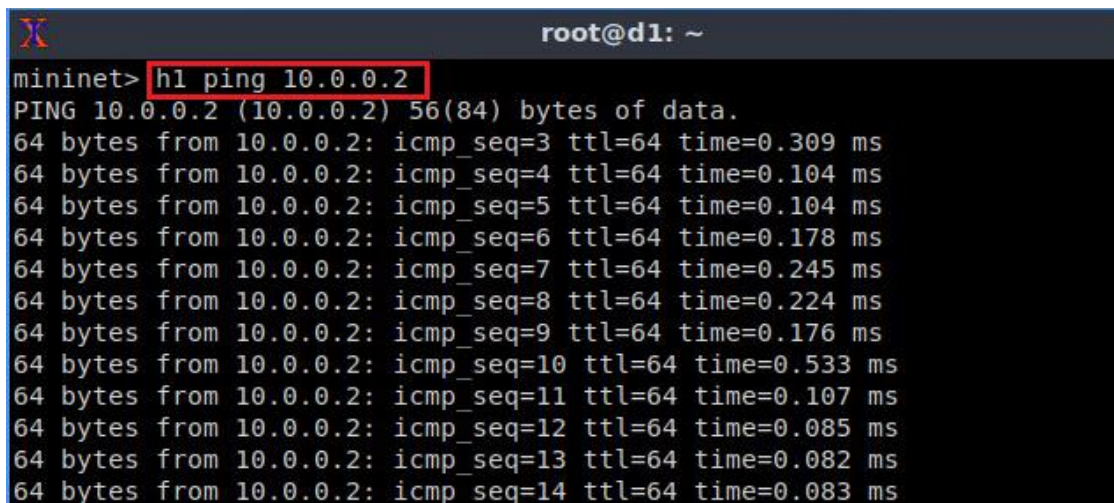
In this section, the user will verify that the VXLAN tags were applied accordingly. Notice that the traffic between h1 and h2 has the VXLAN tag 100 and, the traffic between h11 and h22 corresponds to the VXLAN tag 200. This tag is known as the VXLAN Network Identifier (VNI). The VNI is used to identify 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 h2, also between h11 and h22.

Step 1. In container d1 terminal, issue the following command to verify the connectivity between host h1 and host h11. Notice that `h1` specifies host 1 as the source.

```
h1 ping 10.0.0.2
```



```

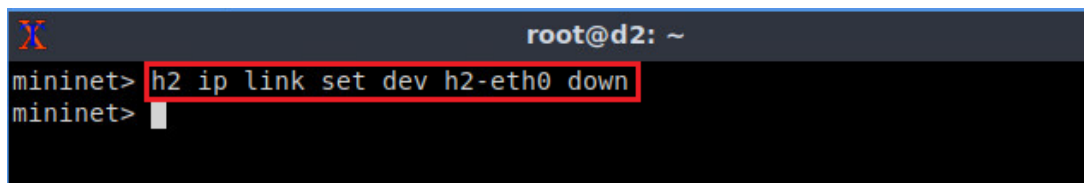
root@d1: ~
mininet> h1 ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.309 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.104 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.104 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.178 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 37. Performing a connectivity test between host h1 and host h2.

Consider Figure 37. 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 h2.



```

root@d2: ~
mininet> h2 ip link set dev h2-eth0 down
mininet>

```

Figure 38. Disabling h2 network interface.

Step 3. Click on container d1 terminal. The user 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 39. Verifying connectivity between host h1 and host h2.

Step 4. In container d1 terminal, issue the following command to test the connectivity between host h11 and host h22. Notice that `h11` specifies host 11 as the source.

```
h11 ping 10.0.0.2
```

```

root@d1: ~
mininet> h11 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=0.258 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.094 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.108 ms

```

Figure 40. Performing a connectivity test between host h11 and host h22.

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 Wireshark network analyzer. The identifiers are used by the switch to isolate network traffic.

Step 1. Click on router r1 terminal and issue the following command to exit the vtsh session.

```
exit
```

```

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

```

Figure 41. Exiting from vtysh.

Step 2. In router r1 terminal, start Wireshark dissector by issuing the following command. A new window will emerge.

```
wireshark
```

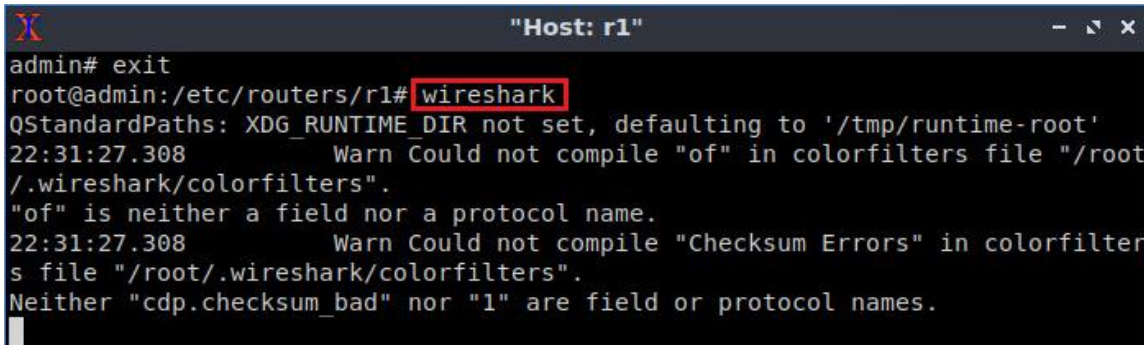


Figure 42. Starting Wireshark network analyzer.

After executing the above command on router r1 terminal, Wireshark window will open, where you monitor different interfaces related to router r1.

Step 3. Click on interface *r1-eth0* then on the icon located on upper left-hand side to start capturing packets on this interface.

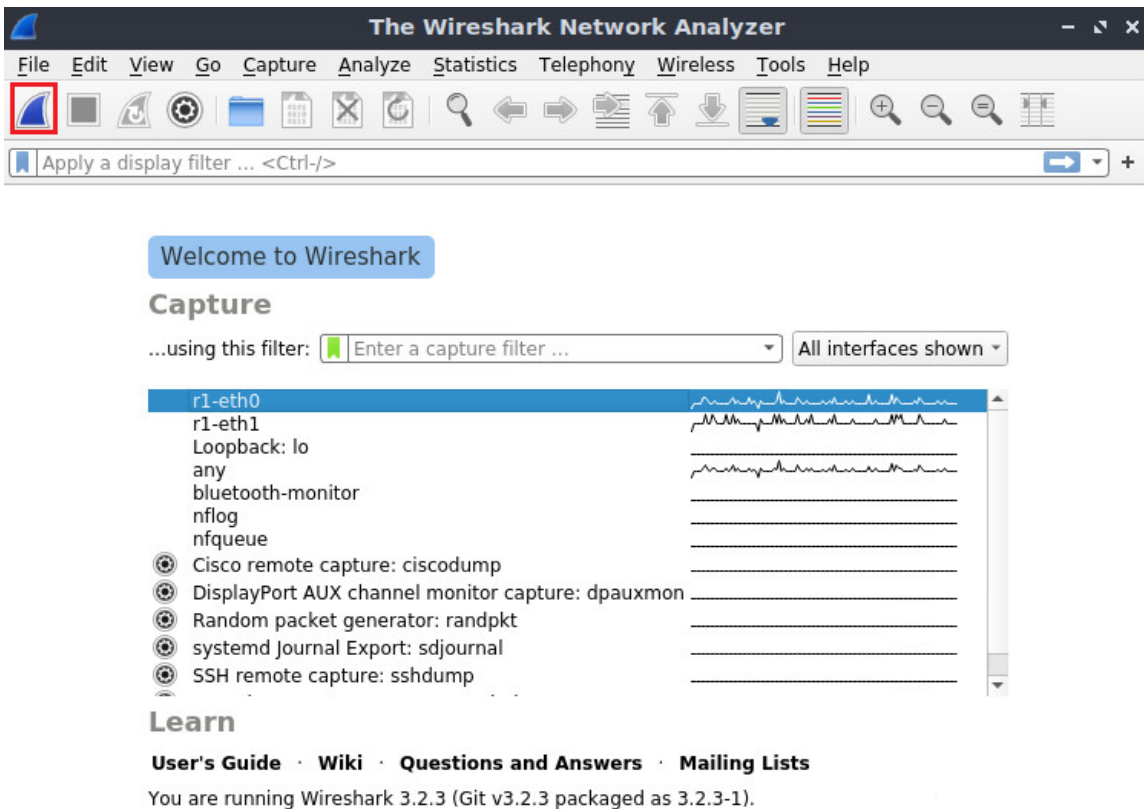


Figure 43. Starting packet capturing on interface r1-eth0.

Step 4. In the filter box located in upper left-hand side, type *vxlan* in order to filter the packets that contains VXLAN tags.

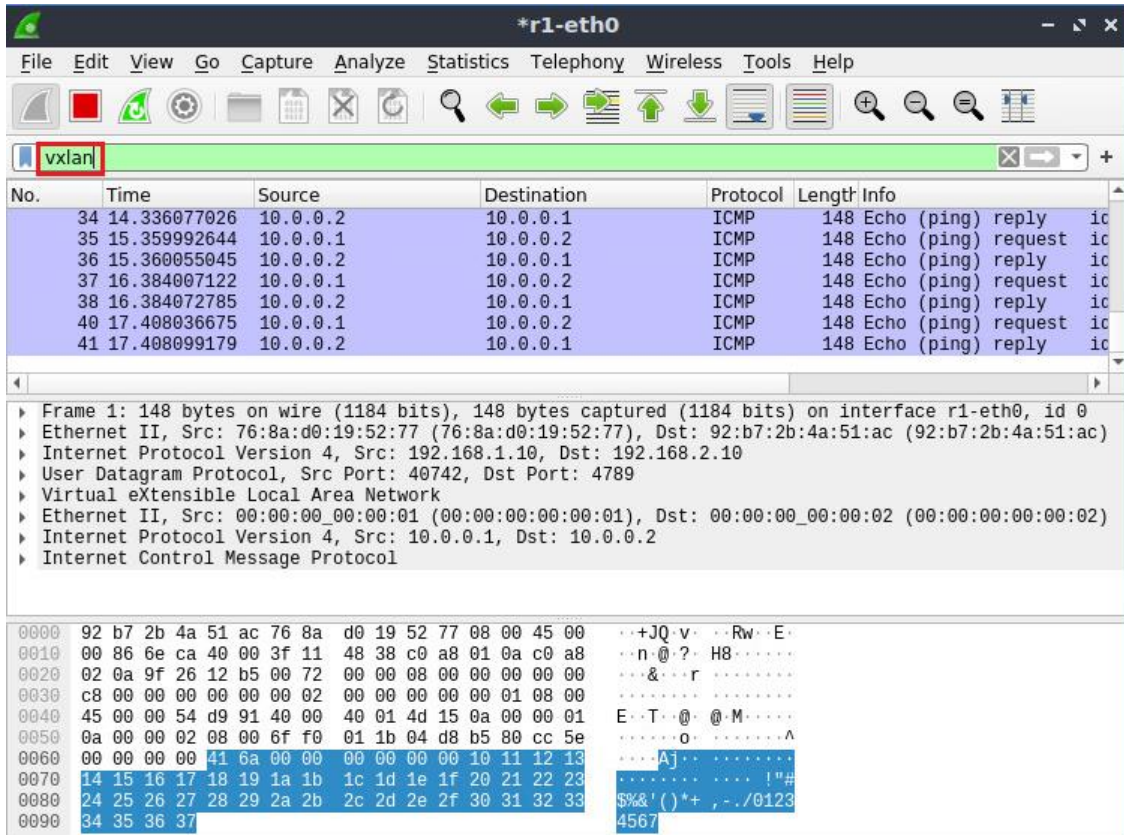


Figure 44. Filtering network traffic.

Step 5. Click on the arrow located on leftmost 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 h11 to h22.

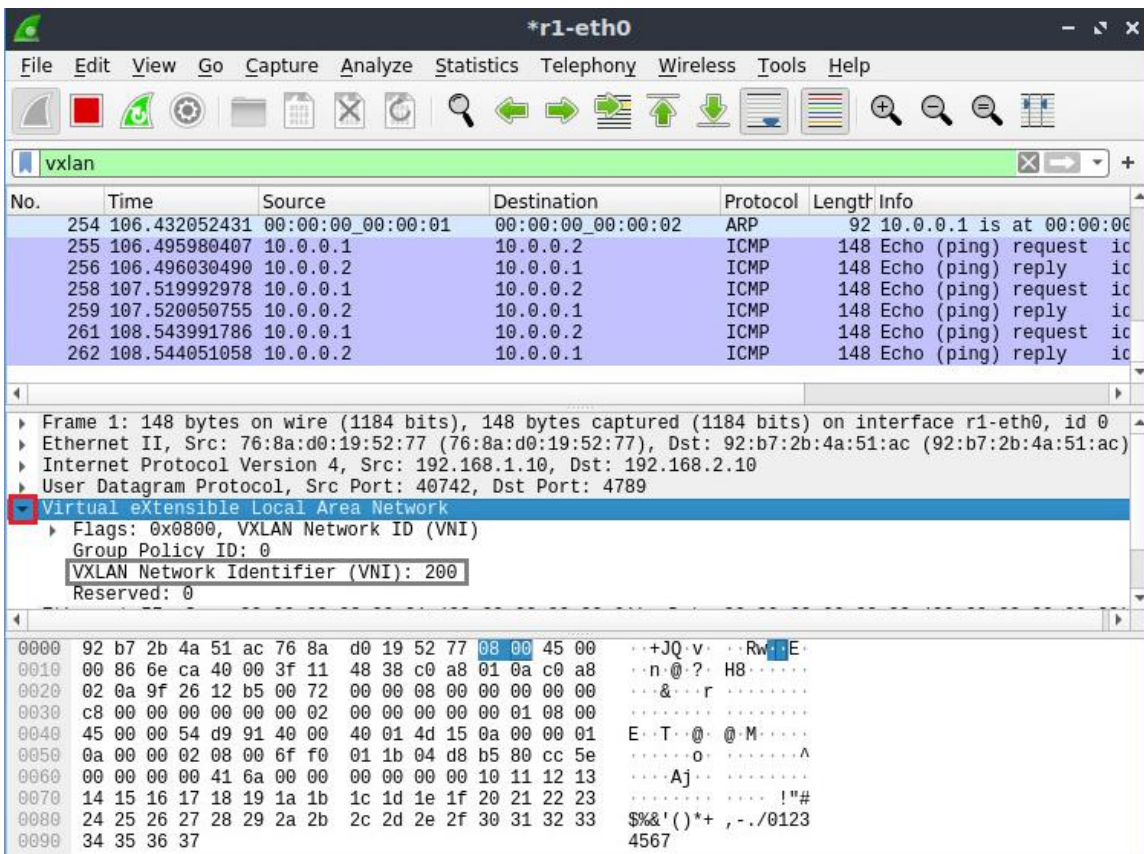


Figure 45. Verifying VXLAN network identifier.

Step 6. To stop packet capturing, click on the red button located on the upper left-hand side.

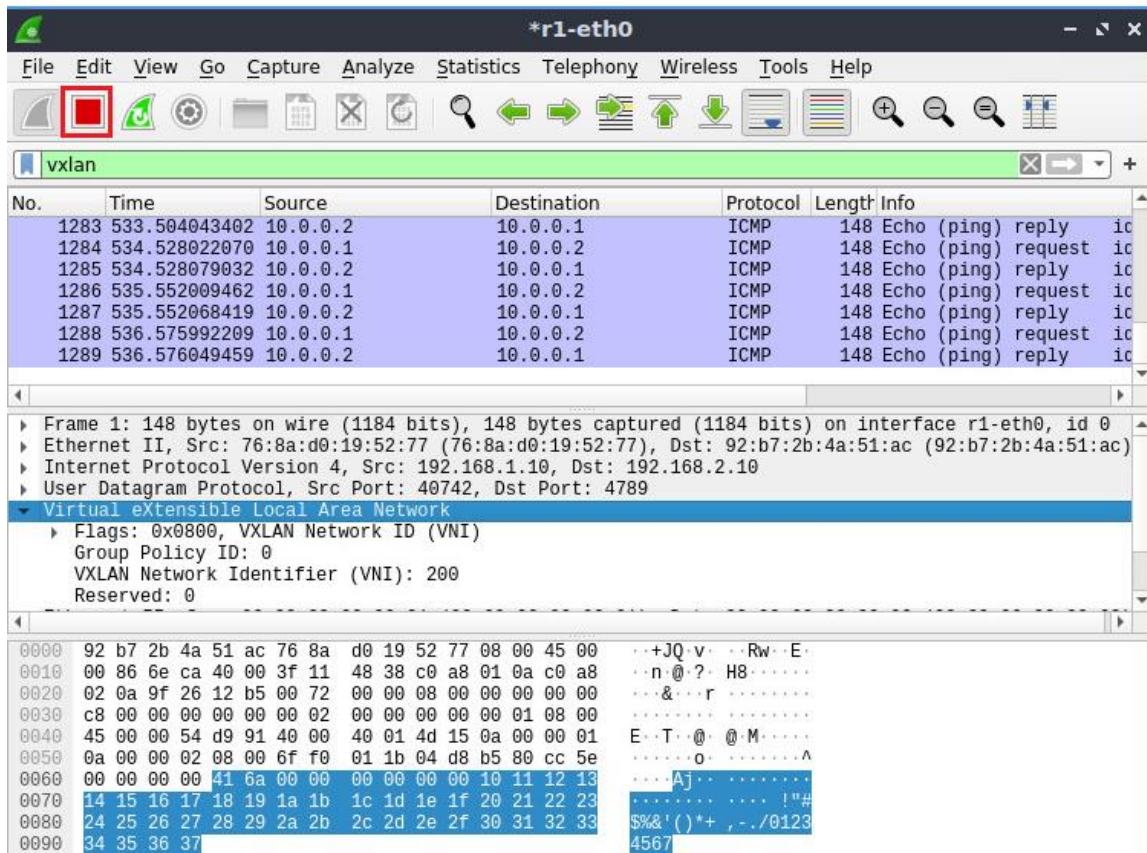


Figure 46. Stopping packet capturing.

Step 7. In container d1, press `ctrl+c` to stop the test.

Step 8. In container d2 terminal, re-enable the network interface in host h2 by issuing the following command:

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

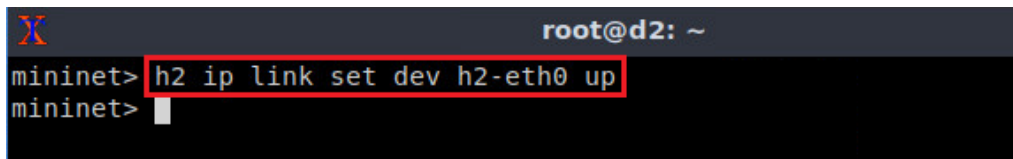


Figure 47. Enabling interface h2-eth0.

Step 9. Perform a connectivity test between h1 and h2 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 48. Performing a connectivity test between host h1 and host h2.

Consider Figure 48. The results show a successful connectivity test.

Step 10. In Wireshark window, start packet capturing by clicking on the button located on upper left-hand side.

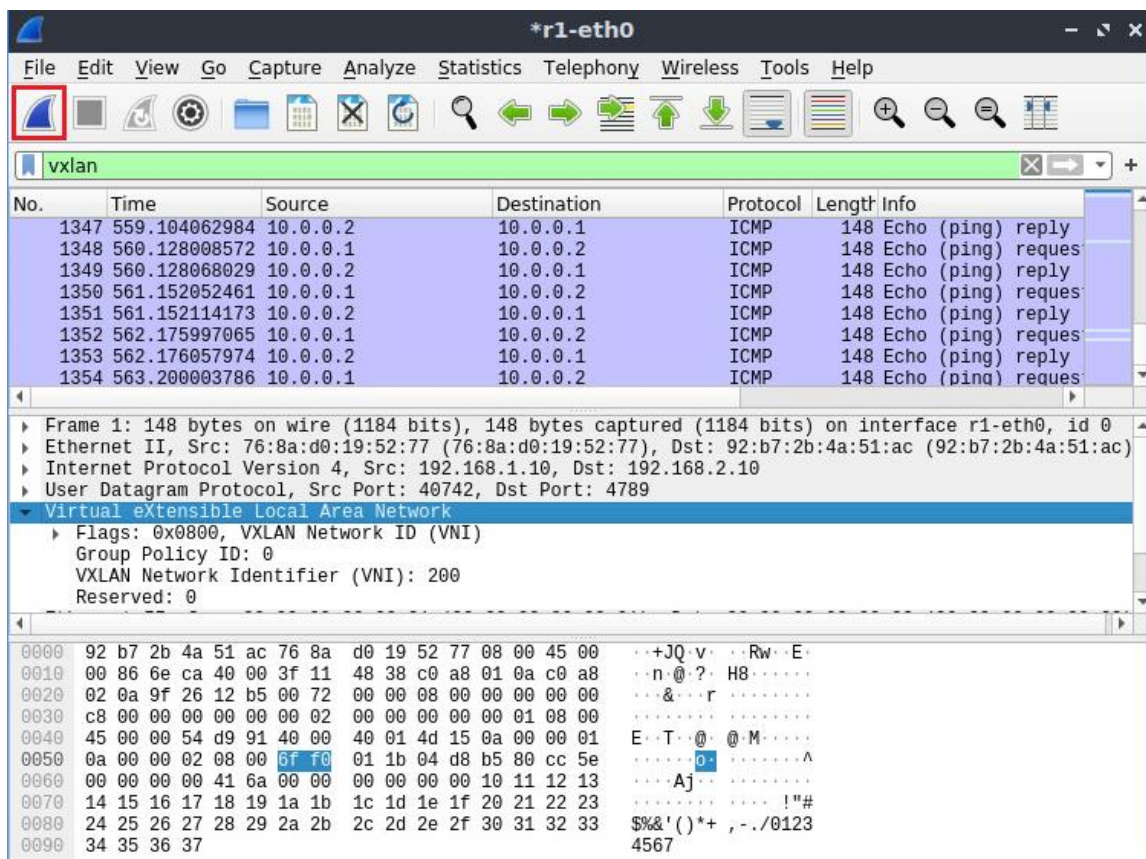


Figure 49. Starting packet capturing.

Step 11. A notification window will be prompted. Click on *Continue without Saving* to proceed.

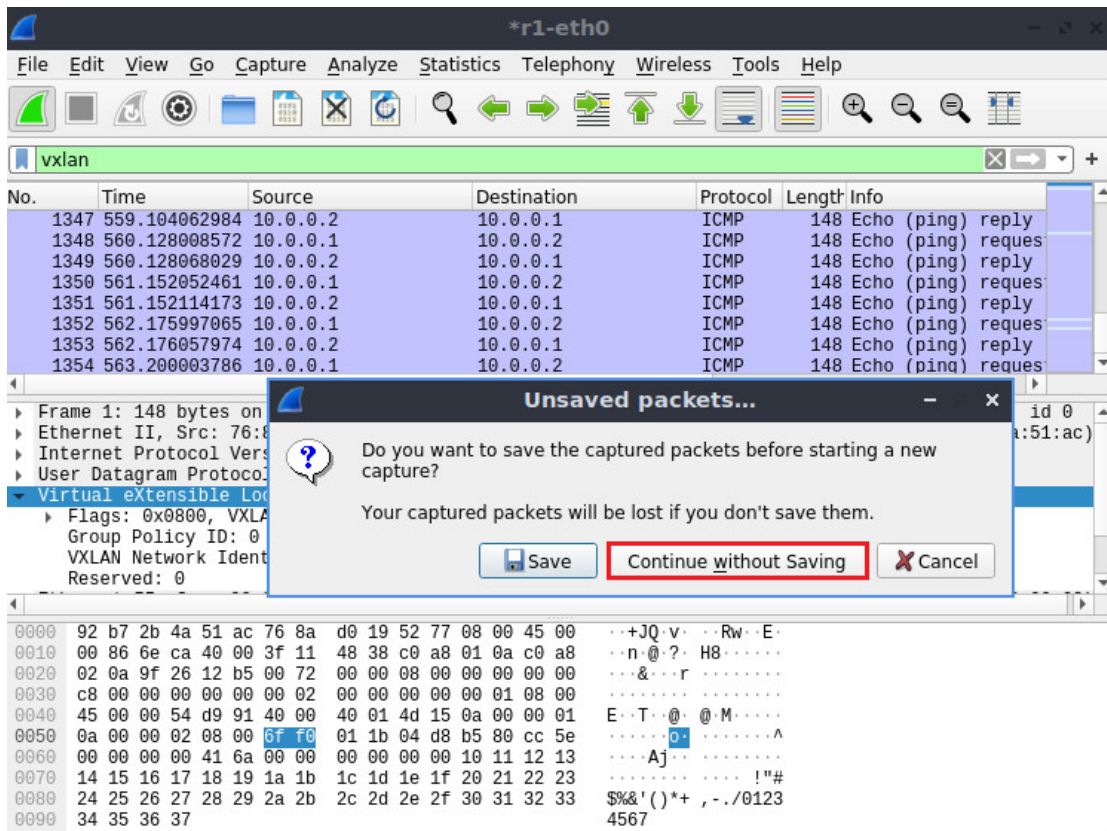


Figure 50. Closing without saving previous packet capture.

Step 12. Verify that the VXLAN Network Identifier is 100. Notice that such tag corresponds to the traffic from h1 to h2.

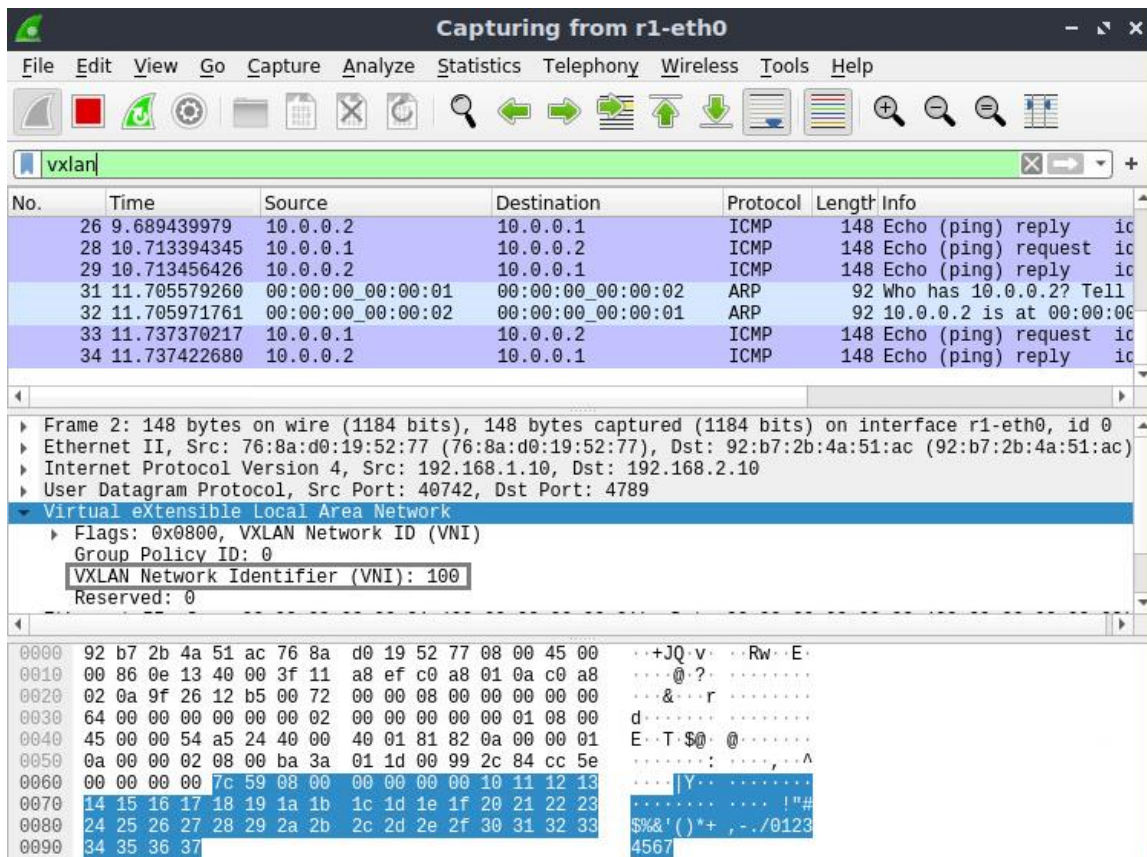


Figure 51. Verifying VXLAN network identifier.

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

References

1. Mininet walkthrough, [Online]. Available: <http://mininet.org>.
2. Peuster, Manuel, Johannes Kampmeyer, and Holger Karl. "Containernet 2.0: A rapid prototyping platform for hybrid service function chains." *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018.
3. 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.
4. P. Goransson, C. Black, T. Culver. "Software defined networks: a comprehensive approach". Morgan Kaufmann, 2016.
5. 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.
6. Mahalingam, Mallik, et al. "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks." *RFC* 7348 (2014): 1-22.
7. 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.

8. Qu, Xiaorong, Weiguo Hao, and Yuanbin Yin. "*L3 gateway for VXLAN*." U.S. Patent No. 8,923,155. 30 Dec. 2014.
9. Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." *Linux journal* 2014.239 (2014): 2.
10. Linux foundation collaborative projects, "*FRRouting: what's in your router*", [Online]. <https://frrouting.org/>