

# Software-Defined Networking

## Lab 3 OpenFlow and Wireshark

University of Colorado Boulder  
Department of Computer Science

Professor Levi Perigo, Ph.D.

## Lab Summary

Understanding the OpenFlow protocol is critical to understanding SDN. The purpose of this lab is to capture and analyze the OpenFlow protocol messages using Wireshark. Knowing how OpenFlow works and being able to troubleshoot different scenarios within Wireshark, will prove to be a valuable skillset in industry.

## Objective 1 – X11 Forwarding: Setup the SDN (controller/switch) Environment

1. Run Xming software on your laptop.
  2. Start the controllers VM.
  3. Verify that your local machine can talk to the controllers VM.
  4. Run ODL from the CLI of the SSH session on your local machine (this may take a few min. to initialize (use the Lab 0 document for assistance)).
    - a. Provide a screenshot of ODL running on the CLI of the VM. [5 points]

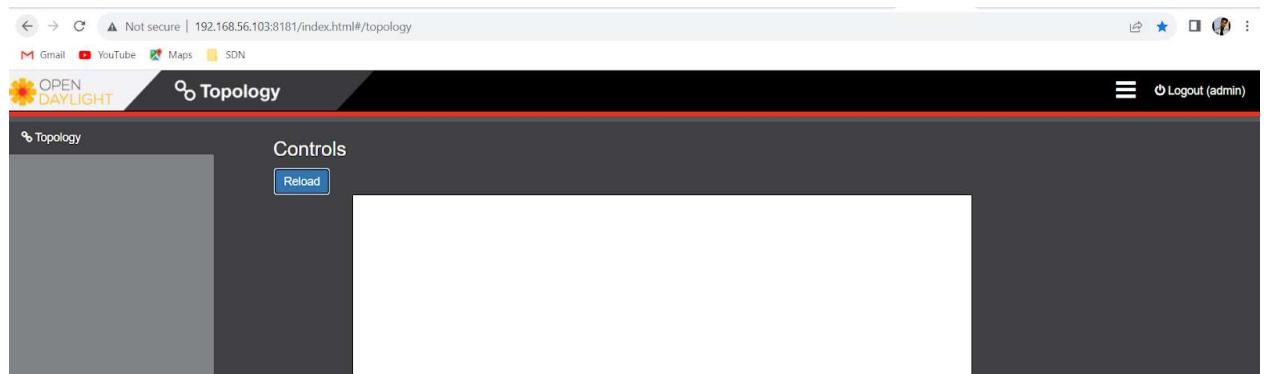
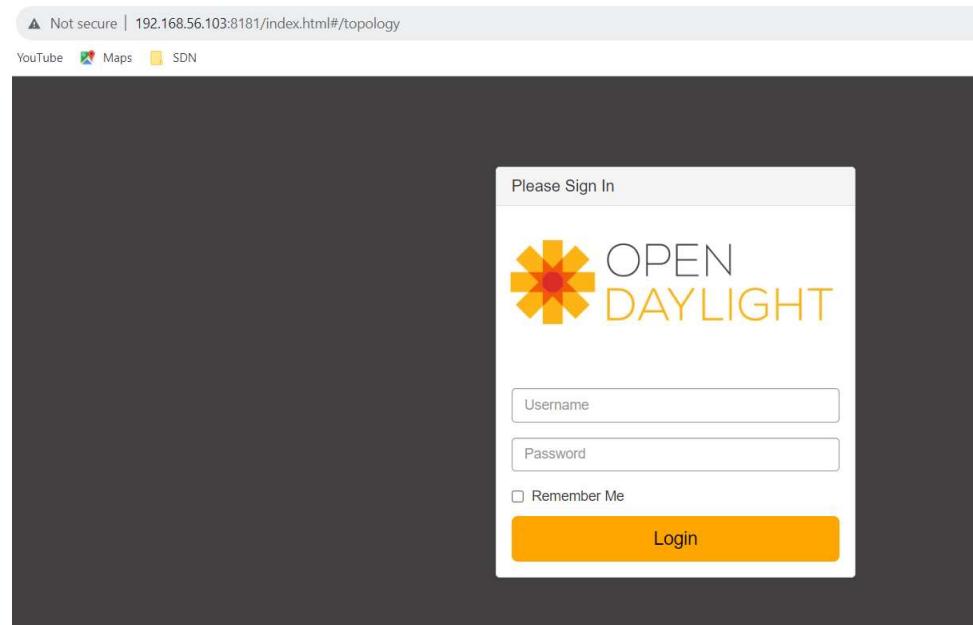
a. Provide a screenshot of ODL running on the CLI of the VM. [5 points]

```
sdn@sdn-controllers:~/karaf-0.8.3$ bin/karaf clean
Apache Karaf starting up. Press Enter to open the shell now...
100% [=====]
Karaf started in 1s. Bundle stats: 13 active, 13 total

  
Hit '<tab>' for a list of available commands  
and '[cmd] --help' for help on a specific command.  
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
opendaylight-user@root>feature:install odl-12switch-switch-ui
opendaylight-user@root>
```

5. Enable the ODL web interface and all the features (use the Lab 0 document for assistance).
  6. Login to the ODL web interface.
    - a. Provide a screenshot of your local machine at the ODL home page. **[10 points]**



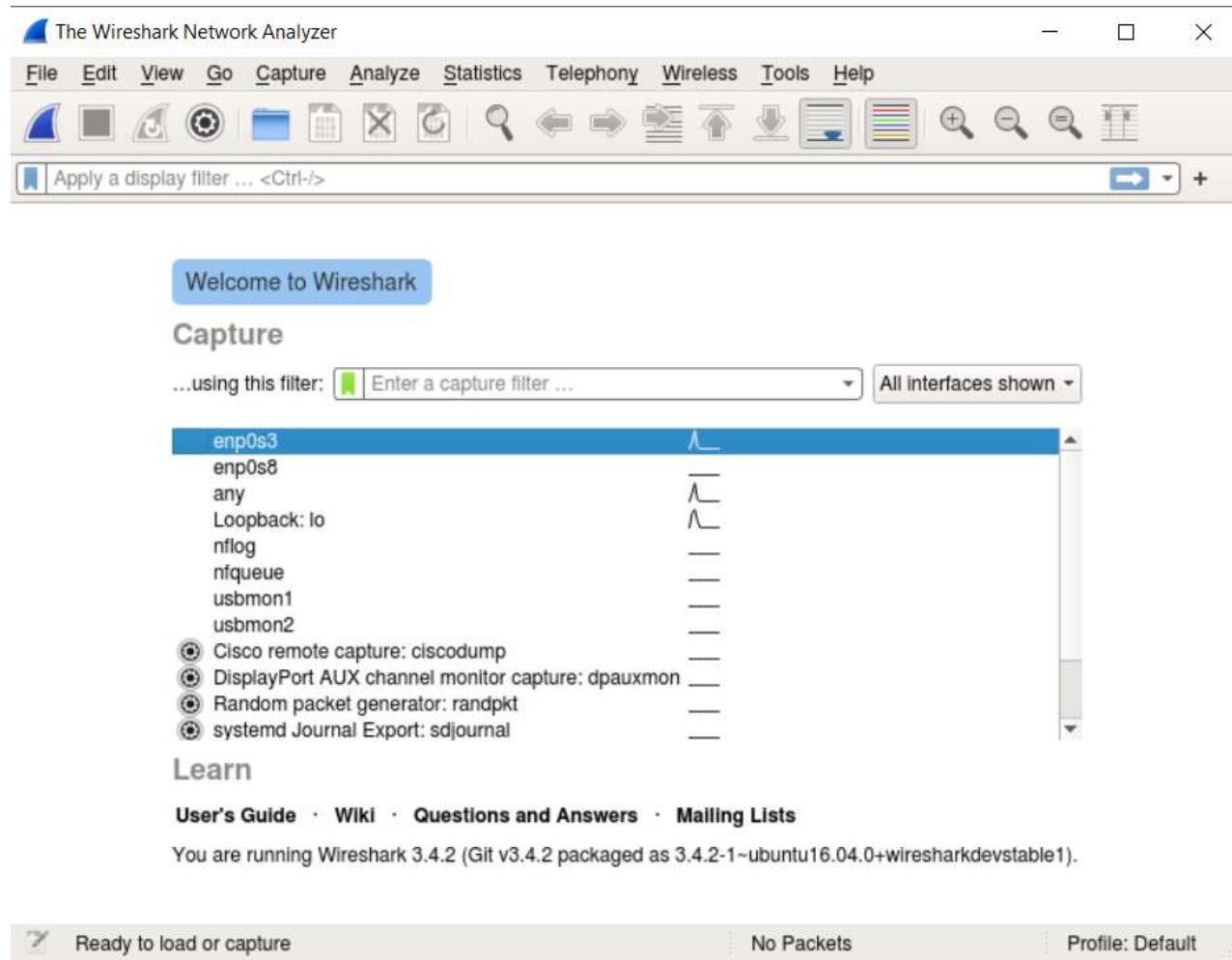
7. Start a new PuTTy/SSH session to the OpenDaylight controller.
  - a. Before you connect, in PuTTy go to Connection > SSH > X11.
  - b. Check the “Enable X11 forwarding” box.
    - i. Note: If using MAC/Linux you can simply SSH with the “-X”.
  - c. Initiate the PuTTy connection.
  - d. Install and initialize Wireshark on the controllers VM (use the Lab 0 document for assistance).
  - e. In the new Wireshark window when you go to “Interfaces” it should be the interfaces on the controller (not your local machine).

- i. Provide a screenshot of the Wireshark interfaces that indicates that Wireshark is running on the controller. **[10 points]**

```
sdn@sdn-controllers: ~
sdn@sdn-controllers:~$ wireshark
sdn@sdn-controllers:~$ wireshark
sdn@sdn-controllers:~$ ifconfig
enp0s3      Link encap:Ethernet HWaddr 08:00:27:43:2f:55
             inet addr:192.168.56.103 Bcast:192.168.56.255 Mask:255.255.255.0
             inet6 addr: fe80::a00:27ff:fe43:2f55/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
             RX packets:34422 errors:0 dropped:0 overruns:0 frame:0
             TX packets:61457 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:2553424 (2.5 MB) TX bytes:163115863 (163.1 MB)

enp0s8      Link encap:Ethernet HWaddr 08:00:27:78:c0:15
             inet addr:10.0.3.15 Bcast:10.0.3.255 Mask:255.255.255.0
             inet6 addr: fe80::a00:27ff:fe78:c015/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
             RX packets:80134 errors:0 dropped:0 overruns:0 frame:0
             TX packets:13343 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:120005938 (120.0 MB) TX bytes:903931 (903.9 KB)

lo          Link encap:Local Loopback
             inet addr:127.0.0.1 Mask:255.0.0.0
             inet6 addr: ::1/128 Scope:Host
             UP LOOPBACK RUNNING MTU:65536 Metric:1
             RX packets:5079 errors:0 dropped:0 overruns:0 frame:0
             TX packets:5079 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1
             RX bytes:83457356 (83.4 MB) TX bytes:83457356 (83.4 MB)
```



8. Start the Mininet VM.
9. Verify that your local machine can talk to the Mininet VM.
10. Start a new PuTTy/SSH session to the Mininet VM (using the X11 steps above).
  - a. Login to Mininet.
  - b. Start a new Mininet topology that connects to a remote controller (ODL), uses easy to read MAC addresses, uses OpenFlow v1.3, and uses a network topology that has a single switch and four hosts.
    - i. Provide a screenshot of the command used. **[5 points]**

```
mininet@mininet-ofm:~$ sudo mn --controller=remote,ip=192.168.56.103 --mac --switch=ovsk,protocols=OpenFlow13 --topo=single,4
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.56.103:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> _
```

- c. Within Mininet, issue the command: xterm h1 h2 h3 h4.
- Explain what happened, what each of these new windows are, why this would be beneficial; also provide a screenshot of all four windows (proving you got it working). **[10 points]**

A terminal is opened for each host on the mininet. It would be beneficial if someone has to work on issuing a few commands via hosts to check networks behavior. It can also help to open cli for multiple hosts at once.

```
X "Node: h1"@mininet-ofm
root@mininet-ofm:~# ifconfig
h1-eth0 Link encap:Ethernet HWaddr 00:00:00:00:00:01
          inet addr:10.0.0.1 Bcast:10.255.255.255 Mask:255.0.0.0
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                  RX packets:45 errors:0 dropped:15 overruns:0 frame:0
                  TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:1000
                  RX bytes:3543 (3.5 KB) TX bytes:840 (840.0 B)

lo      Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
                  UP LOOPBACK RUNNING MTU:65536 Metric:1
                  RX packets:711 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:711 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:0
                  RX bytes:1183980 (1.1 MB) TX bytes:1183980 (1.1 MB)

root@mininet-ofm:~#
```

switch and two hosts.

```
X "Node: h2"@mininet-ofm
root@mininet-ofm:~# ifconfig
h2-eth0 Link encap:Ethernet HWaddr 00:00:00:00:00:02
          inet addr:10.0.0.2 Bcast:10.255.255.255 Mask:255.0.0.0
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                  RX packets:47 errors:0 dropped:17 overruns:0 frame:0
                  TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:1000
                  RX bytes:3713 (3.7 KB) TX bytes:840 (840.0 B)

lo      Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
                  UP LOOPBACK RUNNING MTU:65536 Metric:1
                  RX packets:908 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:908 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:0
                  RX bytes:1206548 (1.2 MB) TX bytes:1206548 (1.2 MB)

root@mininet-ofm:~#
```

X "Node: h3"@mininet-ofm

```
root@mininet-ofm:~# ifconfig
h3-eth0    Link encap:Ethernet HWaddr 00:00:00:00:00:03
           inet addr:10.0.0.3 Bcast:10.255.255.255 Mask:255.0.0.0
             UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
             RX packets:40 errors:0 dropped:8 overruns:0 frame:0
             TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:3032 (3.0 KB) TX bytes:840 (840.0 B)

lo        Link encap:Local Loopback
           inet addr:127.0.0.1 Mask:255.0.0.0
             UP LOOPBACK RUNNING MTU:65536 Metric:1
             RX packets:723 errors:0 dropped:0 overruns:0 frame:0
             TX packets:723 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:1184824 (1.1 MB) TX bytes:1184824 (1.1 MB)

root@mininet-ofm:~#
```

X "Node: h4"@mininet-ofm

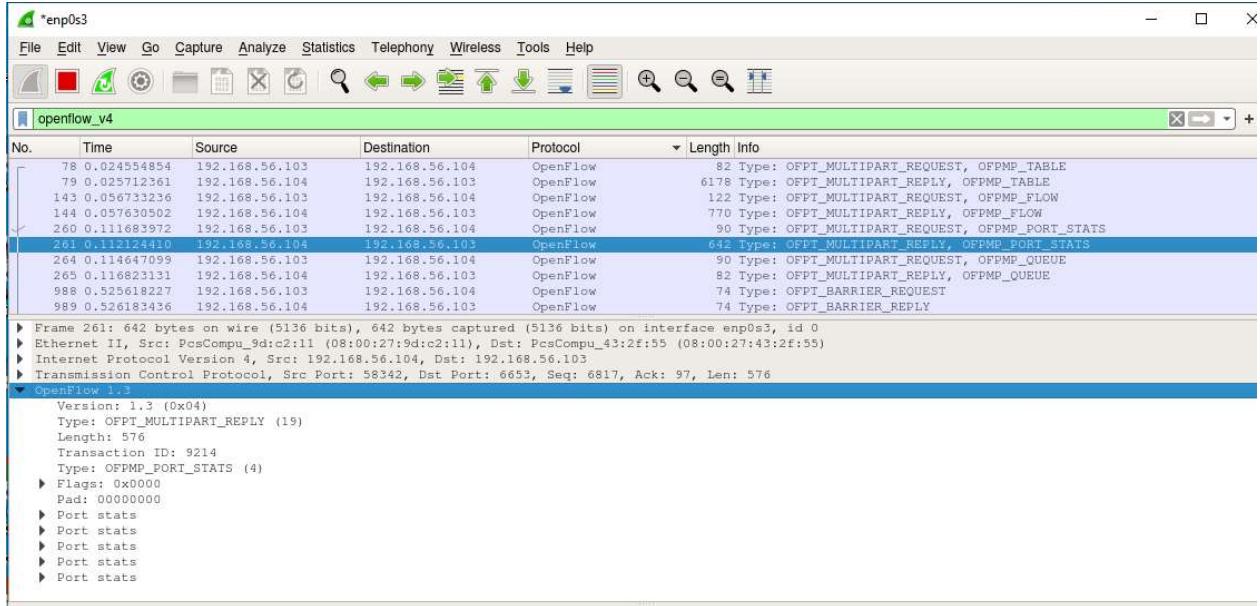
```
root@mininet-ofm:~# ifconfig
h4-eth0    Link encap:Ethernet HWaddr 00:00:00:00:00:04
           inet addr:10.0.0.4 Bcast:10.255.255.255 Mask:255.0.0.0
             UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
             RX packets:49 errors:0 dropped:17 overruns:0 frame:0
             TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:3797 (3.7 KB) TX bytes:840 (840.0 B)

lo        Link encap:Local Loopback
           inet addr:127.0.0.1 Mask:255.0.0.0
             UP LOOPBACK RUNNING MTU:65536 Metric:1
             RX packets:740 errors:0 dropped:0 overruns:0 frame:0
             TX packets:740 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:1185952 (1.1 MB) TX bytes:1185952 (1.1 MB)

root@mininet-ofm:~#
```

## Objective 2 - Capture OpenFlow Messages

1. Open Wireshark on the controller VM (use steps from Objective 1).
2. Create a display filter in Wireshark to only show OpenFlow version 1.3 messages.
  - a. What Wireshark Filter would you use to capture OpenFlow version 1.3 messages? [3 points] —> **openflow\_v4**



3. Start capturing traffic (only showing OpenFlow version 1.3 messages).
4. Start a new Mininet topology that connects to a remote controller (ODL), uses easy to read MAC addresses, uses OpenFlow v1.3, and uses a network topology that has a single switch and two hosts.

- i. Provide a screenshot of the command used. [5 points]

```
mininet@mininet-ofm:~$ sudo mn --mac --controller=remote,ip=192.168.56.103 --switch=ovsk,protocols=OpenFlow13 --topo=single
,[sudo] password for mininet:
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.56.103:6653
*** 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:
mininet> 
```

5. Within Mininet, issue a single ping from Host 1 to Host 2.

- After the ping has completed, stop the Wireshark capture.
- Provide the command used to ping from Host 1 to Host 2. [2 points]

```
mininet> h1 ping h2 -c 1
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=0.464 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.464/0.464/0.464/0.000 ms
mininet>
```

- Provide a screenshot of Wireshark only displaying OpenFlow messages during the Setup/Ping. [2 points]

#### Before Ping - setup stage

8248	12.152180080	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_HELLO
8251	12.155735772	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_HELLO
8252	12.155877415	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
8255	12.159152170	192.168.56.104	192.168.56.103	OpenFlow	98	Type: OFPT_FEATURES_REPLY
8256	12.159713958	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
8257	12.160277904	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
8260	12.173925019	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST
8261	12.174549944	192.168.56.104	192.168.56.103	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY
8262	12.204263447	192.168.56.103	192.168.56.104	OpenFlow	114	Type: OFPT_MULTIPART_REPLY
8263	12.205062375	192.168.56.104	192.168.56.103	OpenFlow	98	Type: OFPT_MULTIPART_REPLY
8264	12.205076245	192.168.56.104	192.168.56.103	OpenFlow	94	Type: OFPT_ERROR
8265	12.205078298	192.168.56.104	192.168.56.103	OpenFlow	274	Type: OFPT_MULTIPART_REPLY
8281	12.272099672	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST
8282	12.272611582	192.168.56.104	192.168.56.103	OpenFlow	6178	Type: OFPT_MULTIPART_REPLY
8284	12.284150761	192.168.56.103	192.168.56.104	OpenFlow	122	Type: OFPT_MULTIPART_REQUEST
8285	12.284500582	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_MULTIPART_REPLY
8286	12.287184235	192.168.56.103	192.168.56.104	OpenFlow	90	Type: OFPT_ROLE_REQUEST
8287	12.287504562	192.168.56.104	192.168.56.103	OpenFlow	90	Type: OFPT_ROLE_REPLY
8288	12.287903745	192.168.56.103	192.168.56.104	OpenFlow	90	Type: OFPT_ROLE_REQUEST
8289	12.288147363	192.168.56.104	192.168.56.103	OpenFlow	90	Type: OFPT_ROLE_REPLY
8290	12.291212495	192.168.56.103	192.168.56.104	OpenFlow	90	Type: OFPT_MULTIPART_REQUEST
8291	12.292097057	192.168.56.104	192.168.56.103	OpenFlow	418	Type: OFPT_MULTIPART_REPLY
8292	12.293245424	192.168.56.103	192.168.56.104	OpenFlow	90	Type: OFPT_MULTIPART_REQUEST
8293	12.308112846	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_MULTIPART_REPLY
8294	12.320299717	192.168.56.103	192.168.56.104	OpenFlow	464	Type: OFPT_SET_CONFIG
8295	12.324173192	192.168.56.103	192.168.56.104	OpenFlow	154	Type: OFPT_FLOW_MOD
8297	12.325555227	192.168.56.103	192.168.56.104	OpenFlow	130	Type: OFPT_FLOW_MOD
8299	12.667879137	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
8301	12.668730454	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
9396	13.739433321	192.168.56.103	192.168.56.104	OpenFlow	452	Type: OFPT_PACKET_OUT
9560	14.240220563	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
9562	14.240800351	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
9571	14.325087990	192.168.56.103	192.168.56.104	OpenFlow	170	Type: OFPT_FLOW_MOD
9572	14.333216898	192.168.56.103	192.168.56.104	OpenFlow	170	Type: OFPT_FLOW_MOD

## Ping

No.	Time	Source	Destination	Protocol	Length	Info
10006	14.740869390	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
10007	14.741824916	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
10401	15.330833611	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_MULTIPART_REQ...
10404	15.331622011	192.168.56.104	192.168.56.103	OpenFlow	386	Type: OFPT_MULTIPART_REPL...
10406	15.342729042	192.168.56.103	192.168.56.104	OpenFlow	122	Type: OFPT_MULTIPART_REQ...
10407	15.344560206	192.168.56.104	192.168.56.103	OpenFlow	434	Type: OFPT_MULTIPART_REPL...
10408	15.358671031	192.168.56.103	192.168.56.104	OpenFlow	90	Type: OFPT_MULTIPART_REQ...
10409	15.360687728	192.168.56.104	192.168.56.103	OpenFlow	418	Type: OFPT_MULTIPART_REPL...
10410	15.362114485	192.168.56.103	192.168.56.104	OpenFlow	90	Type: OFPT_MULTIPART_REQ...
10411	15.362590527	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_MULTIPART_REPL...
10592	15.830939072	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
10595	15.831296893	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
11212	17.249770044	192.168.56.104	192.168.56.103	OpenFlow	150	Type: OFPT_PACKET_IN
11214	17.249871293	192.168.56.104	192.168.56.103	OpenFlow	430	Type: OFPT_PACKET_IN
→ 11602	18.250324626	192.168.56.104	192.168.56.103	OpenFlow	206	Type: OFPT_PACKET_IN
→ 11604	18.250413182	192.168.56.104	192.168.56.103	OpenFlow	206	Type: OFPT_PACKET_IN
11636	18.363413888	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_MULTIPART_REQ...
11639	18.364113776	192.168.56.104	192.168.56.103	OpenFlow	1834	Type: OFPT_MULTIPART_REPL...
11675	18.389732710	192.168.56.103	192.168.56.104	OpenFlow	122	Type: OFPT_MULTIPART_REQ...
11676	18.392321658	192.168.56.104	192.168.56.103	OpenFlow	434	Type: OFPT_MULTIPART_REPL...
11729	18.428370514	192.168.56.103	192.168.56.104	OpenFlow	90	Type: OFPT_MULTIPART_REQ...
11731	18.428813079	192.168.56.104	192.168.56.103	OpenFlow	418	Type: OFPT_MULTIPART_REPL...
11732	18.429182031	192.168.56.103	192.168.56.104	OpenFlow	90	Type: OFPT_MULTIPART_REQ...
11733	18.429498948	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_MULTIPART_REPL...
11805	18.740793393	192.168.56.103	192.168.56.104	OpenFlow	452	Type: OFPT_PACKET_OUT

d. What is the logical interface that connects each OpenFlow switch to a controller?

[5 points] → eth2

As observed in the packets received the IP address used by all the switches

connected to the controller are **192.168.56.104**, with different ports

```

mininet> s1 ifconfig
eth0      Link encap:Ethernet HWaddr 08:00:27:86:a6:6d
          UP BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

eth1      Link encap:Ethernet HWaddr 08:00:27:f9:5c:9a
          UP BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

eth2      Link encap:Ethernet HWaddr 08:00:27:9d:c2:11
          inet addr:192.168.56.104 Bcast:192.168.56.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:24523 errors:0 dropped:0 overruns:0 frame:0
          TX packets:27088 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1879262 (1.8 MB) TX bytes:4074879 (4.0 MB)

eth3      Link encap:Ethernet HWaddr 08:00:27:99:58:e6
          inet addr:10.0.5.15 Bcast:10.0.5.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:937 errors:0 dropped:0 overruns:0 frame:0
          TX packets:942 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:90848 (90.8 KB) TX bytes:81902 (81.9 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

```

eth2 physically doesn't exist

6. Which device (switch or controller) initiated the OpenFlow connection?

a. Provide a screenshot of the capture that indicates this. [2 points]

The connection is initiated by the **switch**

No.	Time	Source	Destination	Protocol	Length	Info
8887	13.166577993	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_HELLO ✓
8892	13.175720549	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_HELLO
8893	13.175871780	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
8896	13.176538766	192.168.56.104	192.168.56.103	OpenFlow	98	Type: OFPT_FEATURES_REPLY
8897	13.176729003	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
8898	13.177482023	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
8899	13.185483123	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, ...
8900	13.185996647	192.168.56.104	192.168.56.103	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY, OF...
8901	13.192015256	192.168.56.103	192.168.56.104	OpenFlow	114	Type: OFPT_MULTIPART_REQUEST, ...
8902	13.192320716	192.168.56.104	192.168.56.103	OpenFlow	98	Type: OFPT_MULTIPART_REPLY, OF...
8903	13.192407386	192.168.56.104	192.168.56.103	OpenFlow	94	Type: OFPT_ERROR
8904	13.192483909	192.168.56.104	192.168.56.103	OpenFlow	274	Type: OFPT_MULTIPART_REPLY, OF...

Frame 8887: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface enp0s3, id 0
 ► Ethernet II, Src: PcsCompu\_9d:c2:11 (08:00:27:9d:c2:11), Dst: PcsCompu\_43:2f:55 (08:00:27:43:2f:55)
 ► Internet Protocol Version 4, Src: 192.168.56.104, Dst: 192.168.56.103
 ► Transmission Control Protocol, Src Port: 58464, Dst Port: 6653, Seq: 1, Ack: 1, Len: 16
 ▼ OpenFlow 1.3
 Version: 1.3 (0x04)
 Type: OFPT\_HELLO (0)
 Length: 16
 Transaction ID: 59
 ▼ Element
 Type: OFPHET\_VERSIONBITMAP (1)
 Length: 8
 Bitmap: 00000010

7. Within Wireshark indicate the messages from the switch and the controller that indicate which version of OpenFlow they support.

- a. Provide a screenshot of each. [10 points]

#### Hello message from the switch

No.	Time	Source	Destination	Protocol	Length	Info
8887	13.166577993	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_HELLO
8892	13.175720549	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_HELLO
8893	13.175871780	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
8896	13.176538766	192.168.56.104	192.168.56.103	OpenFlow	98	Type: OFPT_FEATURES_REPLY
8897	13.176729003	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
8898	13.177482023	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
8899	13.185483123	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, ...
8900	13.185996647	192.168.56.104	192.168.56.103	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY, OF...
8901	13.192015256	192.168.56.103	192.168.56.104	OpenFlow	114	Type: OFPT_MULTIPART_REQUEST, ...
8902	13.192320716	192.168.56.104	192.168.56.103	OpenFlow	98	Type: OFPT_MULTIPART_REPLY, OF...
8903	13.192407386	192.168.56.104	192.168.56.103	OpenFlow	94	Type: OFPT_ERROR
8904	13.192483909	192.168.56.104	192.168.56.103	OpenFlow	274	Type: OFPT_MULTIPART_REPLY, OF...

► Frame 8887: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface enp0s3, id 0  
 ► Ethernet II, Src: PcsCompu\_9d:c2:11 (08:00:27:9d:c2:11), Dst: PcsCompu\_43:2f:55 (08:00:27:43:2f:55)  
 ► Internet Protocol Version 4, Src: 192.168.56.104, Dst: 192.168.56.103  
 ► Transmission Control Protocol, Src Port: 58464, Dst Port: 6653, Seq: 1, Ack: 1, Len: 16  
 ▾ OpenFlow 1.3  
 Version: 1.3 (0x04)  
 Type: OFPT\_HELLO (0)  
 Length: 16  
 Transaction ID: 59  
 ▾ Element  
 Type: OFPHET\_VERSIONBITMAP (1)  
 Length: 8  
 Bitmap: 00000010

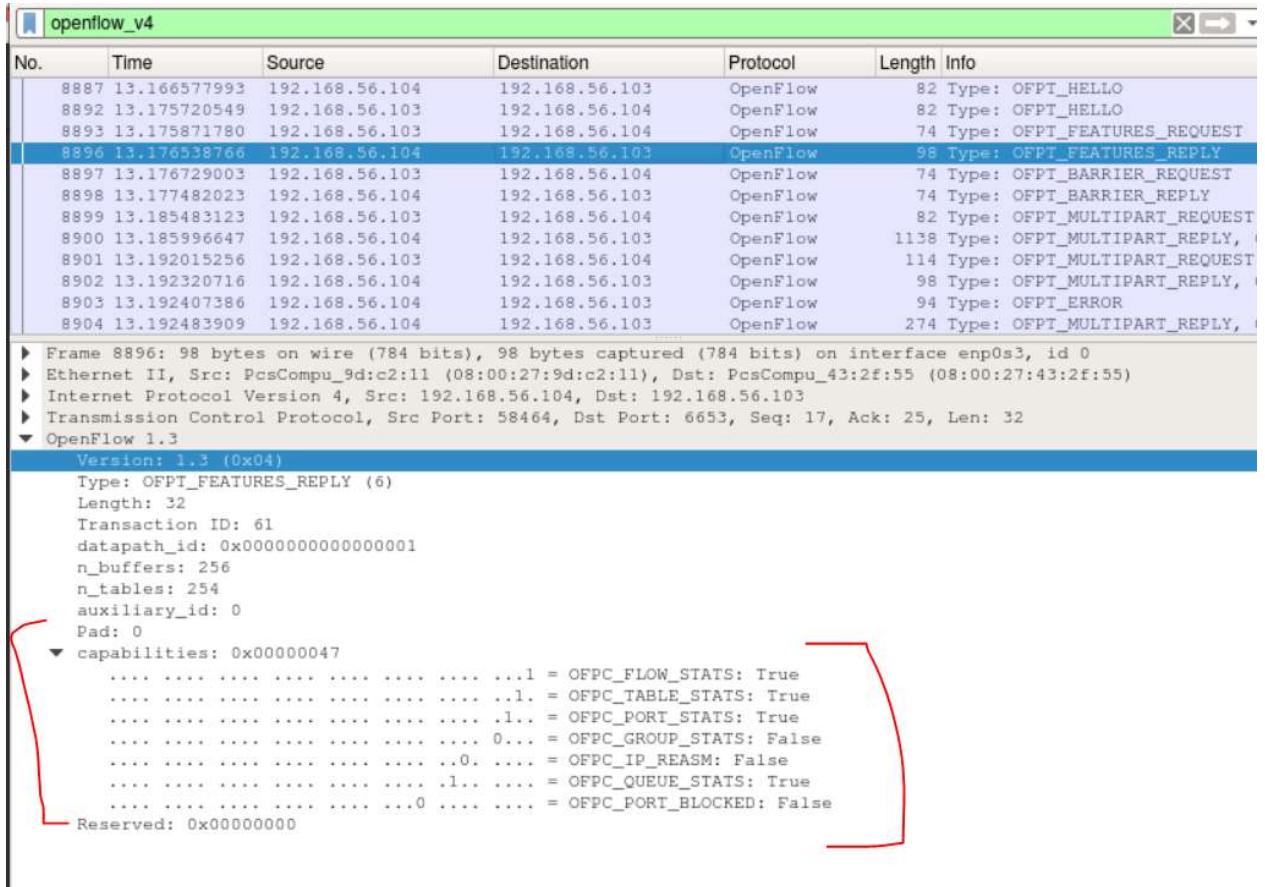
#### Hello message from controller

No.	Time	Source	Destination	Protocol	Length	Info
8887	13.166577993	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_HELLO
8892	13.175720549	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_HELLO
8893	13.175871780	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
8896	13.176538766	192.168.56.104	192.168.56.103	OpenFlow	98	Type: OFPT_FEATURES_REPLY
8897	13.176729003	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
8898	13.177482023	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
8899	13.185483123	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, ...
8900	13.185996647	192.168.56.104	192.168.56.103	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY, OF...
8901	13.192015256	192.168.56.103	192.168.56.104	OpenFlow	114	Type: OFPT_MULTIPART_REQUEST, ...
8902	13.192320716	192.168.56.104	192.168.56.103	OpenFlow	98	Type: OFPT_MULTIPART_REPLY, OF...
8903	13.192407386	192.168.56.104	192.168.56.103	OpenFlow	94	Type: OFPT_ERROR
8904	13.192483909	192.168.56.104	192.168.56.103	OpenFlow	274	Type: OFPT_MULTIPART_REPLY, OF...

► Frame 8892: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface enp0s3, id 0  
 ► Ethernet II, Src: PcsCompu\_43:2f:55 (08:00:27:43:2f:55), Dst: PcsCompu\_9d:c2:11 (08:00:27:9d:c2:11)  
 ► Internet Protocol Version 4, Src: 192.168.56.103, Dst: 192.168.56.104  
 ► Transmission Control Protocol, Src Port: 6653, Dst Port: 58464, Seq: 1, Ack: 17, Len: 16  
 ▾ OpenFlow 1.3  
 Version: 1.3 (0x04)  
 Type: OFPT\_HELLO (0)  
 Length: 16  
 Transaction ID: 60  
 ▾ Element  
 Type: OFPHET\_VERSIONBITMAP (1)  
 Length: 8  
 Bitmap: 00000012

8. Within Wireshark indicate the message that shows what features the switch supports.

- a. Provide a screenshot of the Wireshark message, indicating the features. [5 points]



The screenshot shows a Wireshark capture titled "openflow\_v4". The packet list pane displays several OpenFlow messages, including an OFPT\_HELLO, OFPT\_FEATURES\_REQUEST, OFPT\_FEATURES\_REPLY, OFPT\_BARRIER\_REQUEST, OFPT\_BARRIER\_REPLY, OFPT\_MULTIPART\_REQUEST, OFPT\_MULTIPART\_REPLY, OFPT\_MULTIPART\_REQUEST, OFPT\_MULTIPART\_REPLY, OFPT\_ERROR, and OFPT\_MULTIPART\_REPLY. The details pane for the selected OFPT\_FEATURES\_REPLY packet (Frame 8896) shows its structure. A red bracket highlights the "capabilities" field, which contains a series of bits indicating supported statistics: OFPC\_FLOW\_STATS, OFPC\_TABLE\_STATS, OFPC\_PORT\_STATS, OFPC\_GROUP\_STATS, OFPC\_IP\_REASM, OFPC\_QUEUE\_STATS, and OFPC\_PORT\_BLOCKED. The "OFPC\_GROUP\_STATS" bit is shown as False.

No.	Time	Source	Destination	Protocol	Length	Info
8887	13.166577993	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_HELLO
8892	13.175720549	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_HELLO
8893	13.175871780	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
8896	13.176538766	192.168.56.104	192.168.56.103	OpenFlow	98	Type: OFPT_FEATURES_REPLY
8897	13.176729003	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
8898	13.177482023	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
8899	13.185483123	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST
8900	13.185996647	192.168.56.104	192.168.56.103	OpenFlow	1138	Type: OFPT_MULTIPART_REPLY, Len: 32
8901	13.192015256	192.168.56.103	192.168.56.104	OpenFlow	114	Type: OFPT_MULTIPART_REQUEST
8902	13.192320716	192.168.56.104	192.168.56.103	OpenFlow	98	Type: OFPT_MULTIPART_REPLY, Len: 32
8903	13.192407386	192.168.56.104	192.168.56.103	OpenFlow	94	Type: OFPT_ERROR
8904	13.192483909	192.168.56.104	192.168.56.103	OpenFlow	274	Type: OFPT_MULTIPART_REPLY, Len: 32

► Frame 8896: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0  
 ► Ethernet II, Src: PcsCompu\_9d:c2:11 (08:00:27:9d:c2:11), Dst: PcsCompu\_43:2f:55 (08:00:27:43:2f:55)  
 ► Internet Protocol Version 4, Src: 192.168.56.104, Dst: 192.168.56.103  
 ► Transmission Control Protocol, Src Port: 58464, Dst Port: 6653, Seq: 17, Ack: 25, Len: 32  
 ▾ OpenFlow 1.3  
 Version: 1.3 (0x04)  
 Type: OFPT\_FEATURES\_REPLY (6)  
 Length: 32  
 Transaction ID: 61  
 datapath\_id: 0x0000000000000001  
 n\_buffers: 256  
 n\_tables: 254  
 auxiliary\_id: 0  
 Pad: 0  
 ▾ capabilities: 0x00000047  
 .... .... .... .... 1 = OFPC\_FLOW\_STATS: True  
 .... .... .... .... 1. = OFPC\_TABLE\_STATS: True  
 .... .... .... .... 1.. = OFPC\_PORT\_STATS: True  
 .... .... .... .... 0... = OFPC\_GROUP\_STATS: False  
 .... .... .... .... 0. .... = OFPC\_IP\_REASM: False  
 .... .... .... .... 1. .... = OFPC\_QUEUE\_STATS: True  
 .... .... .... .... 0 .... = OFPC\_PORT\_BLOCKED: False  
 Reserved: 0x00000000

9. Within Wireshark indicate how many flow tables the switch supports in the OpenFlow pipeline?

- a. Provide a screenshot highlighting the answer from the capture. [5 points]

254

No.	Time	Source	Destination	Protocol	Length Info
15243	17.530347931	192.168.56.104	192.168.56.103	OpenFlow	82 Type: OFPT_HELLO
15244	17.538346834	192.168.56.103	192.168.56.104	OpenFlow	82 Type: OFPT_HELLO
15250	17.539571974	192.168.56.103	192.168.56.104	OpenFlow	74 Type: OFPT_FEATURES_REQUEST
15252	17.540577466	192.168.56.104	192.168.56.103	OpenFlow	98 Type: OFPT_FEATURES_REPLY
15253	17.542355268	192.168.56.103	192.168.56.104	OpenFlow	74 Type: OFPT_BARRIER_REQUEST
15254	17.542936509	192.168.56.104	192.168.56.103	OpenFlow	74 Type: OFPT_BARRIER_REPLY
15256	17.582556183	192.168.56.103	192.168.56.104	OpenFlow	82 Type: OFPT_MULTIPART_REQUEST, OFPMP_DESC
15257	17.583676643	192.168.56.104	192.168.56.103	OpenFlow	1138 Type: OFPT_MULTIPART_REPLY, OFPMP_DESC
15259	17.67320982	192.168.56.103	192.168.56.104	OpenFlow	114 Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
► Frame 15252: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0					
► Ethernet II, Src: PcsCompu_9d:c2:11 (08:00:27:9d:c2:11), Dst: PcsCompu_43:2f:55 (08:00:27:43:2f:55)					
► Internet Protocol Version 4, Src: 192.168.56.104, Dst: 192.168.56.103					
► Transmission Control Protocol, Src Port: 45376, Dst Port: 6653, Seq: 17, Ack: 25, Len: 32					
▼ OpenFlow 1.3					
Version: 1.3 (0x04)					
Type: OFPT_FEATURES_REPLY (6)					
Length: 32					
Transaction ID: 4741					
datapath_id: 0x0000000000000001					
n_buffers: 256					
n_tables: 254					
auxiliary_id: 0					
Pad: 0					
▼ capabilities: 0x000000047					
..... .1. .... .1 = OFPC_FLOW_STATS: True					
..... .1. .... .1. = OFPC_TABLE_STATS: True					
..... .1. .... .0... = OFPC_PORT_STATS: True					
..... .0... = OFPC_GROUP_STATS: False					
..... .0... = OFPC_IP_REASM: False					
..... .1... = OFPC_QUEUE_STATS: True					
..... .0 ... .0 = OFPC_PORT_BLOCKED: False					
Reserved: 0x00000000					

10. The controller sent a OFPT\_MULTIPART\_REQUEST, and the switch replied with a “REPLY.”

Explain what a multipart OpenFlow message is, and why/when they are used, also indicate, by showing in the capture, one switch reply to a multipart request. [5 points]

Multipart Openflow message is a way of sending extra statistical information, requested by the controller to the switch. It is an efficient way to extract information like - **Port stats, Table stats, Table features , Queue Statistics and Group Statistics**, since it uses a single request and the stats are sent back immediately. They are carried in a single TCP entity, which makes it efficient rather than sending multiple request each for a different statistics, reducing the overall traffic in the out of band network

### Request Packet for Table stats and Role in the same TCP packet

```

8903 13.1924073786 192.168.56.104    192.168.56.103    OpenFlow      94 Type: OFPT_ERROR
8904 13.192483909  192.168.56.104    192.168.56.103    OpenFlow      274 Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
8906 13.220235506  192.168.56.103    192.168.56.104    OpenFlow      106 Type: OFPT_ROLE_REQUEST
8907 13.220714668  192.168.56.104    192.168.56.103    OpenFlow      6178 Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE
8909 13.220755909  192.168.56.104    192.168.56.103    OpenFlow      90 Type: OFPT_ROLE_REPLY
8910 13.226848544  192.168.56.103    192.168.56.104    OpenFlow      90 Type: OFPT_ROLE_REQUEST

▶ Frame 8906: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface enp0s3, id 0
▶ Ethernet II, Src: PcsCompu_43:2f:55 (08:00:27:43:2f:55), Dst: PcsCompu_9d:c2:11 (08:00:27:9d:c2:11)
▶ Internet Protocol Version 4, Src: 192.168.56.103, Dst: 192.168.56.104
▶ Transmission Control Protocol, Src Port: 6653, Dst Port: 58464, Seq: 97, Ack: 1397, Len: 40
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REQUEST (18)
  Length: 16
  Transaction ID: 4
  Type: OFPPMP_TABLE (3)
  ▼ Flags: 0x0000
    .... .... .... 0 = OFPMPF_REQ_MORE: 0x0
  Pad: 00000000
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_ROLE_REQUEST (24)
  Length: 24
  Transaction ID: 5
  Role: OFPCR_ROLE_NOCHANGE (0x00000000)
  Pad: 00000000
  Generation ID: 0x0000000000000000

```

## Reply

11. Within Wireshark select the OpenFlow message that includes the Ping request message.

- a. Provide a screenshot of this message highlighting the Ping request OpenFlow message. [5 points]

No.	Time	Source	Destination	Protocol	Length	Info
10049	15.383937692	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
10050	15.384447618	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
10252	15.874092873	192.168.56.104	192.168.56.103	OpenFlow	150	Type: OFPT_PACKET_IN
10254	15.874187807	192.168.56.104	192.168.56.103	OpenFlow	430	Type: OFPT_PACKET_IN
10261	15.991098254	192.168.56.103	192.168.56.104	OpenFlow	452	Type: OFPT_PACKET_OUT
10481	16.491669508	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
10482	16.492274346	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
+ 10706	16.876727248	192.168.56.104	192.168.56.103	OpenFlow	206	Type: OFPT_PACKET_IN
+ 10708	16.876873134	192.168.56.104	192.168.56.103	OpenFlow	206	Type: OFPT_PACKET_IN
11224	17.877339713	192.168.56.104	192.168.56.103	OpenFlow	206	Type: OFPT_PACKET_IN
▶ Frame 10706: 206 bytes on wire (1648 bits), 206 bytes captured (1648 bits) on interface enp0s3, id 0						
▶ Ethernet II, Src: PcsCompu_9d:c2:11 (08:00:27:9d:c2:11), Dst: PcsCompu_43:2f:55 (08:00:27:43:2f:55)						
▶ Internet Protocol Version 4, Src: 192.168.56.104, Dst: 192.168.56.103						
▶ Transmission Control Protocol, Src Port: 58468, Dst Port: 6653, Seq: 15269, Ack: 1561, Len: 140						
▼ OpenFlow 1.3						
Version: 1.3 (0x04)						
Type: OFPT_PACKET_IN (10)						
Length: 140						
Transaction ID: 0						
Buffer ID: OFP_NO_BUFFER (4294967295)						
Total length: 98						
Reason: OFPR_ACTION (1)						
Table ID: 0						
Cookie: 0x2b0000000000002f						
Match						
Pad: 0000						
▼ Data						
▶ 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						
Type: 8 (Echo (ping) request)						
Code: 0						
Checksum: 0x4901 [correct]						
[Checksum Status: Good]						
Identifier (BE): 11762 (0x2df2)						
Identifier (LE): 61997 (0xf22d)						
Sequence Number (BE): 2 (0x0002)						
Sequence Number (LE): 512 (0x0200)						
[Response frame: 10708]						
Timestamp from icmp data: Sep 23, 2023 10:21:33.168439000 MDT						
[Timestamp from icmp data (relative): 32828.750611448 seconds]						
Data (48 bytes)						

- b. What type of OpenFlow message is it? [3 points]

It is a **PACKET\_IN** message, indicating that the message is sent from the switch to the controller.

- c. What is the Buffer\_ID field, and what does it mean? [3 points]

When a packet is received by the switch and if it is unable to process it immediately, it will buffer the packet and an associated buffer\_id is generated associated with the location of the packet in the buffer.

Only a portion of the packet header required by the controller can be sent in a PACKET\_IN message, along with the buffer\_id, can be sent to the controller. The controller can then send decision in PACKET\_OUT message, along with buffer\_id.

This makes the process efficient since the whole packet is no longer required to be sent to the controller.

d. Indicate the src MAC and dst MAC, and src IP and dst IP in this OpenFlow packet.

Are these the same or different from the actual src/dst of the packet in Wireshark? Why? [3 points]

srcIP and dstIP are different from the hostSrcIp and hostDestIP.

Similarly, it is true for L2, srcMAC and dstMAC are different from hostSrcMac and hostDestMac.

This is because a PACKET\_IN is basically a table\_miss which is a message from the switch to the controller. The Openflow protocol has one holistic goal of managing flow across switches and it doesn't do real forwarding.

A table\_miss (PACKET\_IN) would cause the switch to ask the controller about the way the packet has to be processed hence the srcIP and destIP is of the switch and controller VMs. Hence, the corresponding MACs are also not of the hosts connected to the switch

12. After the Ping (request/reply) is successful, what does the controller do? What message

does it use to do this? Provide a screenshot of the message. [15 points]

Frame ID	Source MAC	Destination MAC	Source IP	Destination IP	Protocol	Type	Details
16625 25.482706641	192.168.56.104	192.168.56.103	OpenFlow	150	Type: OFPT_PACKET_IN		
16627 25.482832000	192.168.56.104	192.168.56.103	OpenFlow	150	Type: OFPT_PACKET_IN		
16629 25.483233328	192.168.56.104	192.168.56.103	OpenFlow	206	Type: OFPT_PACKET_IN		
16631 25.483526830	192.168.56.104	192.168.56.103	OpenFlow	206	Type: OFPT_PACKET_IN		
16635 25.498421018	192.168.56.103	192.168.56.104	OpenFlow	258	Type: OFPT_FLOW_MOD		
16640 25.626461368	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_TABLE		
16644 25.627695114	192.168.56.104	192.168.56.103	OpenFlow	386	Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE		

Priority: 10  
Buffer ID: OFP\_NO\_BUFFER (4294967295)  
Out port: OFPP\_ANY (4294967295)  
Out group: OFPG\_ANY (4294967295)  
Flags: 0x0000  
Pad: 0000

Match

Type: OFPMT\_OXM (1)  
Length: 24  
OXM field  
Class: OFPXMC\_OPENFLOW\_BASIC (0x8000)  
0000 011. = Field: OFPXMT\_OFB\_ETH\_DST (3)  
.... ...0 = Has mask: False  
Length: 6  
Value: 00:00:00\_00:00:02 (00:00:00:00:00:02)

OXM field  
Class: OFPXMC\_OPENFLOW\_BASIC (0x8000)  
0000 100. = Field: OFPXMT\_OFB\_ETH\_SRC (4)  
.... ...0 = Has mask: False  
Length: 6  
Value: 00:00:00\_00:00:01 (00:00:00:00:00:01)

Instruction

Type: OFPIT\_APPLY\_ACTIONS (4)  
Length: 24  
Pad: 00000000

Action

Type: OFPAT\_OUTPUT (0)  
Length: 16  
Port: 2  
Max length: OFPCML\_NO\_BUFFER (65535)  
Pad: 000000000000

OpenFlow 1.3  
Version: 1.3 (0x04)  
Type: OFPT\_FLOW\_MOD (14)  
Length: 96

2 flows are installed with match criteria as:

## Flows

### 1. Match srcMAC: 0000.0000.0001 destMAC: 0000.0000.0002

#### Action: Egress on Port 2

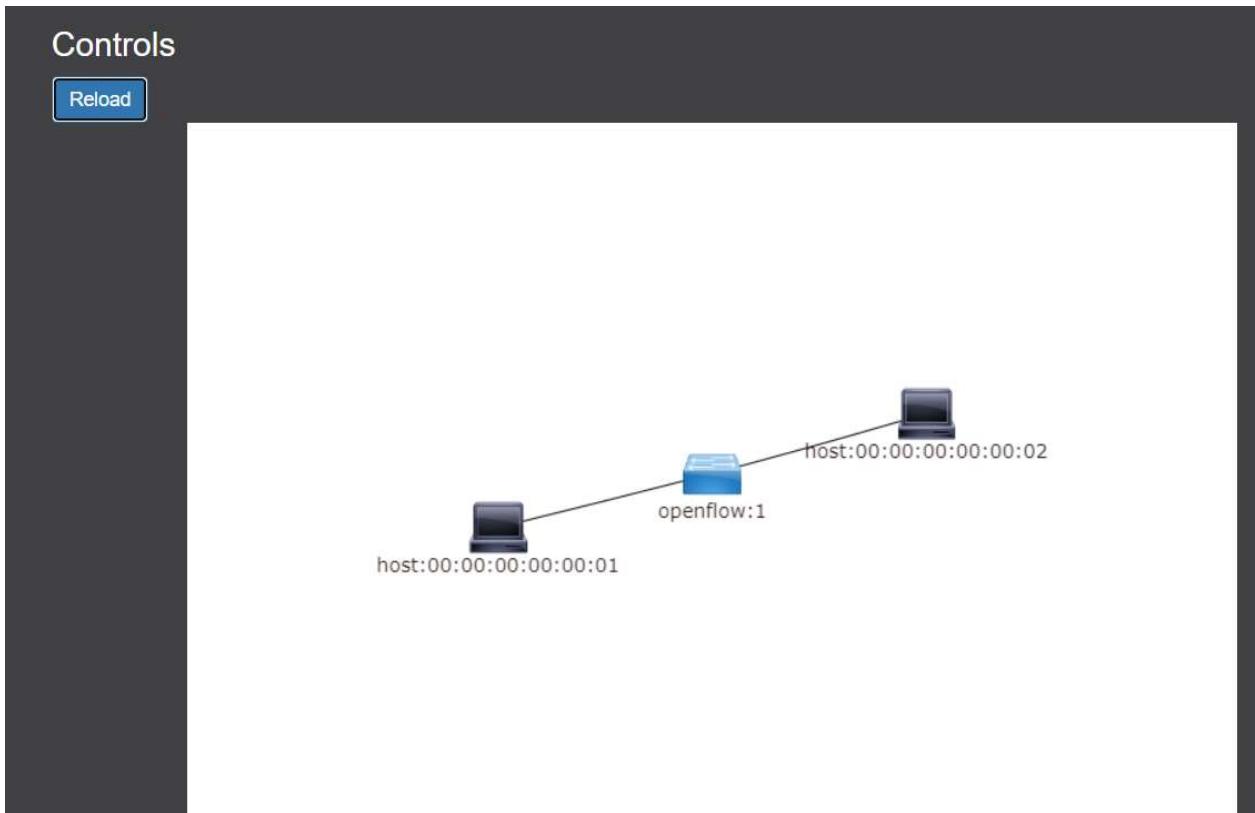
16631 25.483526836 192.168.56.104	192.168.56.103	OpenFlow	206 Type: OFPT_PACKET_IN
16638 25.496421018 192.168.56.103	192.168.56.104	OpenFlow	258 Type: OFPT_FLOW_MOD
16640 25.626461368 192.168.56.103	192.168.56.104	OpenFlow	82 Type: OFPT_MULTIPART_REQUEST, OFPMP_TABLE
16644 25.627695114 192.168.56.104	192.168.56.103	OpenFlow	386 Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE
<pre>Table ID: 0 Command: OFPFC_ADD (0) Idle timeout: 600 Hard timeout: 300 Priority: 10 Buffer ID: OFPP_NO_BUFFER (4294967295) Out port: OFPP_ANY (4294967295) Out group: OFPG_ANY (4294967295) ▶ Flags: 0x0000 Pad: 0000 ▼ Match   Type: OFPMT_OXM (1)   Length: 24   ▼ OXM field     Class: OFPXMC_OPENFLOW_BASIC (0x8000)     0000 011. = Field: OFPXMT_OFB_ETH_DST (3)     .... ...0 = Has mask: False     Length: 6     Value: 00:00:00_00:00:01 (00:00:00:00:00:01)   ▼ OXM field     Class: OFPXMC_OPENFLOW_BASIC (0x8000)     0000 100. = Field: OFPXMT_OFB_ETH_SRC (4)     .... ...0 = Has mask: False     Length: 6     Value: 00:00:00_00:00:02 (00:00:00:00:00:02)   ▼ Instruction     Type: OFPIT_APPLY_ACTIONS (4)     Length: 24     Pad: 00000000     ▼ Action       Type: OFPAT_OUTPUT (0)       Length: 16       Port: 1       Max length: OFPCML_NO_BUFFER (65535)       Pad: 000000000000</pre>			

### 2. Match srcMAC: 0000.0000.0002 destMAC: 0000.0000.0001

#### Action: Egress on Port 1

These flow installed would allow following more icmp packets to directly forwarded without involvement of controller, allowing hardware level forwarding, OFPT\_FLOW\_MOD modifies the flow table.

13. Within the ODL web interface, provide a screenshot of the network topology showing all hosts and switches. [5 points]



## Objective 3 - Capture OpenFlow Failure Messages

### 1. Clean up and clear Mininet

- What command do you use in Mininet to clean it up/clear it? [2 points]

`sudo mn --clean`

```

mininet@mininet-ofm:~$ sudo mn --clean
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core1t-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest m
nexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core1t-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtes t mnexec ivs ryu-manager 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/conn* /tmp/vlogs* /tmp/*.* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+*' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
mininet@mininet-ofm:~$ 
  
```

2. After you clear the Mininet process, refresh the ODL topology. What happened? Why?

[3 points]

Once mininet is cleared, the switches go down and the ODL topology has nothing to show in the dashboard. Hence, there is no topology shown.

Once, a clean message was issued, the switch actively sent the links going down to the controller, allowing the controller to realize the switch was going down.

No.	Time	Source	Destination	Protocol	Length	Info
34100	130.485890639	192.168.56.103	192.168.56.104	OpenFlow	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_STATS
34101	130.486312266	192.168.56.104	192.168.56.103	OpenFlow	418	Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_STATS
34102	130.486864841	192.168.56.103	192.168.56.104	OpenFlow	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_QUEUE
34103	130.487181721	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_MULTIPART_REPLY, OFPMP_QUEUE
34104	130.498668995	192.168.56.103	192.168.56.104	OpenFlow	203	Type: OFPT_SET_CONFIG
34105	130.500893585	192.168.56.103	192.168.56.104	OpenFlow	218	Type: OFPT_FLOW_MOD
34107	130.503452803	192.168.56.103	192.168.56.104	OpenFlow	191	Type: OFPT_PACKET_OUT
34108	130.504186663	192.168.56.103	192.168.56.104	OpenFlow	202	Type: OFPT_PACKET_OUT
34120	130.921039001	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
34121	130.921713334	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
35094	131.910168607	192.168.56.103	192.168.56.104	OpenFlow	452	Type: OFPT_PACKET_OUT
35126	132.410279056	192.168.56.103	192.168.56.104	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
35128	132.410916932	192.168.56.104	192.168.56.103	OpenFlow	74	Type: OFPT_BARRIER_REPLY
35130	132.508520636	192.168.56.103	192.168.56.104	OpenFlow	170	Type: OFPT_FLOW_MOD
35131	132.511369683	192.168.56.103	192.168.56.104	OpenFlow	170	Type: OFPT_FLOW_MOD
35419	132.821080541	192.168.56.104	192.168.56.103	OpenFlow	146	Type: OFPT_PORT_STATUS
35452	132.831271743	192.168.56.104	192.168.56.103	OpenFlow	146	Type: OFPT_PORT_STATUS
35454	132.832139851	192.168.56.104	192.168.56.103	OpenFlow	146	Type: OFPT_PORT_STATUS

```

▶ Transmission Control Protocol, Src Port: 58526, Dst Port: 6653, Seq: 8117, Ack: 1425, Len: 80
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_PORT_STATUS (12)
  Length: 80
  Transaction ID: 0
  Reason: OFPPR_DELETE (1)
  Pad: 00000000000000
  ▼ Port
    Port no: 2
    Pad: 00000000
    Hw addr: aa:a9:e9:d1:28:5e (aa:a9:e9:d1:28:5e)
    Pad: 0000
    Name: s1-eth2
    ▶ Config: 0x00000001
    ▼ State: 0x00000001
      .....1 = OFPPS_LINK_DOWN: True
      .....0 = OFPPS_BLOCKED: False
      .....0 = OFPPS_LIVE: False
    ▶ Current: 0x00000000
    ▶ Advertised: 0x00000000
    ▶ Supported: 0x00000000
    ▶ Peer: 0x00000000
    Curr speed: 0
    Max speed: 0
  
```

3. Use the same steps above to start a new Wireshark capture
4. Use the same steps above to initiate a new Mininet topology. Use the same topology used before in step 2.4, but this time instead of using OpenFlow 1.3 use version 1.2.
- a. Did the switch connect to the controller? Why or why not? How do you know?

Provide screenshots of the capture to support your answer. [15 points]

Hello message sent from the switch to the controller (this message is because the wireshark doesn't know this version, hence the warning)

```

▶ Frame 13038: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface enp0s3, id 0
▶ Ethernet II, Src: PcsCompu_9d:c2:11 (08:00:27:9d:c2:11), Dst: PcsCompu_43:2f:55 (08:00:27:43:2f:55)
▶ Internet Protocol Version 4, Src: 192.168.56.104, Dst: 192.168.56.103
▶ Transmission Control Protocol, Src Port: 58530, Dst Port: 6653, Seq: 1, Ack: 1, Len: 16
▼ .000 0011 = Version: 1.2 (0x03)
  ▼ [Expert Info (Warning/Undecoded): Unsupported version not dissected]
    [Unsupported version not dissected]
    [Severity level: Warning]
    [Group: Undecoded]

```

### the hello message sent by controller to switch

```

▶ Frame 13043: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface enp0s3, id 0
▶ Ethernet II, Src: PcsCompu_43:2f:55 (08:00:27:43:2f:55), Dst: PcsCompu_9d:c2:11 (08:00:27:9d:c2:11)
▶ Internet Protocol Version 4, Src: 192.168.56.103, Dst: 192.168.56.104
▶ Transmission Control Protocol, Src Port: 6653, Dst Port: 58530, Seq: 1, Ack: 17, Len: 16
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_HELLO (0)
  Length: 16
  Transaction ID: 21
  ▼ Element
    Type: OFPHET_VERSIONBITMAP (1)
    Length: 8
    Bitmap: 00000012

```

Clearly they do not support a common version. Hence the controller send a message back indicating there is a version mismatch

Frame	Time	Source IP	Destination IP	Protocol	Length	Type
13045	16.050330055	192.168.56.104	192.168.56.103	OpenFlow	155	OFPT_ERROR
13046	16.050410488	192.168.56.104	192.168.56.103	TCP	66	58530 → 6653 [FIN, ACK] Seq=106 Ack=17
13047	16.051478547	192.168.56.103	192.168.56.104	TCP	66	6653 → 58520 [FIN, ACK] Seq=17 Ack=107
13048	16.051744684	192.168.56.104	192.168.56.103	TCP	66	58530 → 6653 [ACK] Seq=107 Ack=18 Win=1
13059	16.087307833	192.168.56.104	192.168.56.103	TCP	74	58532 → 6653 [SYN] Seq=0 Win=29200 Len=8
13060	16.087322634	192.168.56.103	192.168.56.104	TCP	74	6653 → 58532 [SYN, ACK] Seq=0 Ack=1 Win=1

```

▶ Frame 13045: 155 bytes on wire (1240 bits), 155 bytes captured (1240 bits) on interface enp0s3, id 0
▶ Ethernet II, Src: PcsCompu_9d:c2:11 (08:00:27:9d:c2:11), Dst: PcsCompu_43:2f:55 (08:00:27:43:2f:55)
▶ Internet Protocol Version 4, Src: 192.168.56.104, Dst: 192.168.56.103
▶ Transmission Control Protocol, Src Port: 58530, Dst Port: 6653, Seq: 17, Ack: 17, Len: 89
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_ERROR (1)
  Length: 89
  Transaction ID: 0
  Type: OFPET_HELLO_FAILED (0)
  Code: OFPHFC_INCOMPATIBLE (0)
  Data: We support version 0x03, you support versions 0x01, 0x04, no common versions.

```

The hello message itself failed. However, the controller can downgrade to Openflow1.0, but since we have issued the command, that mininet has to use 1.2 version, it doesn't help.

```

Completed in 2.59 seconds
mininet@mininet-ofm:~$ sudo mn --mac --controller=remote,ip=192.168.56.103 --switch=ovsk,protocols=OpenFlow12 --topo=single,2
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.56.103:6653

```

Since there is no controller connected and the switch has no flow , the packets are simply dropped. Thus ping won't work

## Objective 4 - OpenFlow Failure Messages (Hardware) [Team]

1. Each team will select one Dell/ABMX server. Label your server with your team number/name.
2. For this objective you have to choose a version of Ubuntu image, and do a fresh install of it onto the server.
3. Once Ubuntu is installed, install Open vSwitch on the server.
4. Paste screenshots of the Ubuntu version and OvS version on your server. **[5 points]**

### Ubuntu Version-

```
team2@team2:~$ cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.3 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.3 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
team2@team2:~$
```

### OVS Version -

```
team2@team2:~$ ovs-vsctl --version
ovs-vsctl (Open vSwitch) 2.17.7
DB Schema 8.3.0
team2@team2:~$
```

5. Physically connect your server to an SDN controller. The controller IP is 10.20.30.2/24 and it connected to the ‘Adv. NGN – VPN Mgmt. Switch’ on FastEthernet1/48 and the Mgmt-2 Switch on FastEthernet1/0/1.
6. Connect two laptops to your server. Create an OvS bridge and add the two server interfaces connected to the laptops to the bridge. Before establishing an OpenFlow connection to the controller, ping between two hosts connected to the OVS. Are the pings successful? Show the OpenFlow table of the OVS and explain why the pings are/aren’t successful. **[5 points]**

```
team2@team2:~$ sudo ovs-ofctl dump-flows ovsbridge
cookie=0x0, duration=23.634s, table=0, n_packets=1, n_bytes=60, priority=0 actions=NORMAL
team2@team2:~$
```

Ping works because the OVS switch uses regular switching operation and transparent bridges do not require any configuration. There is no controller, thus the default flow is used, **action=NORMAL** which causes the switch to learn MAC address, perform flooding and thus ping packets work.

7. Write a Python script that –

- Automates the steps required to connect your OvS to the controller.
- Verifies and displays if the OpenFlow connection is successful.
- Indicates the OpenFlow failover mode of the switch.

Provide screenshots of the output of your script and attach it with your submission. [15

points]

```
team2@team2:~/Downloads$ python3 createOVSBridge.py
Creating a new bridge - ovsbridge
Executed - sudo ovs-vsctl add-br ovsbridge : 0
Executed - sudo ovs-vsctl add-port ovsbridge eno1 : 0
Executed - sudo ovs-vsctl add-port ovsbridge eno2 : 0
Executed - sudo ovs-vsctl set-fail-mode ovsbridge secure : 0
Executed - sudo ovs-vsctl set-controller ovsbridge tcp:192.168.1.7:6633 : 0
ovsbridge is created
Wait for the switch to connect to the controller 192.168.1.7:6633
##### STATUS #####
OVS Bridge ovsbridge connected to the controller sucessfully
OVS Bridge ovsbridge fail-mode set to SECURE
team2@team2:~/Downloads$
```

8. Issue a continuous ping between hosts.

- Show the flow table entry from OVS that indicates the ICMP traffic between hosts. [3 points]

**Controller added flows to the switch**

```
team2@team2: $ sudo ovs-ofctl dump-flows ovsbridge
cookie=0x0, duration=11.366s, table=0, n_packets=9, n_bytes=638, in_port=eno1,dl_src=5c:26:0a:24:8f:a0,
dl_dst=5c:26:0a:24:8d:7a actions=output:eno2
cookie=0x0, duration=11.364s, table=0, n_packets=9, n_bytes=638, in_port=eno2,dl_src=5c:26:0a:24:8d:7a,
dl_dst=5c:26:0a:24:8f:a0 actions=output:eno1
team2@team2: $
```

### Connection to the controller

```
team2@team2:~$ sudo ovs-vsctl set controller ovsbridge tcp:192.168.1.1:6633
team2@team2:~$ sudo ovs-vsctl list controller ovsbridge
_uuid          : 646aa1f6-12f0-44d5-a0d9-00418654f681
connection_mode : []
controller_burst_limit: []
controller_queue_size: []
controller_rate_limit: []
enable_async_messages: []
external_ids    : {}
inactivity_probe: []
is_connected    : true
local_gateway   : []
local_ip        : []
local_netmask   : []
max_backoff     : []
other_config    : {}
role            : other
status          : {sec_since_connect="14", state=ACTIVE}
target          : "tcp:192.168.1.7:6633"
type            : []
```

- b. Drop the connection between the OVS/controller.

- Do the pings fail or continue? Are there flow table entries in the OVS or not? Explain why? **[10 points]**

```
C:\Users\kiran>ping 21.21.21.2

Pinging 21.21.21.2 with 32 bytes of data:
Reply from 21.21.21.2: bytes=32 time=9ms TTL=128
Reply from 21.21.21.2: bytes=32 time<1ms TTL=128
Reply from 21.21.21.2: bytes=32 time<1ms TTL=128
Reply from 21.21.21.2: bytes=32 time<1ms TTL=128

Ping statistics for 21.21.21.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 9ms, Average = 2ms
```

Ping works because the OVS behaves like a L2 switch

```
team2@team2:~$ sudo ovs-appctl fdb/show ovsbridge
  port  VLAN  MAC          Age
    2      0  5c:26:0a:24:8d:7a   33
    1      0  5c:26:0a:24:8f:a0   24
team2@team2:~$ sudo ovs-ofctl dump-flows ovsbridge
team2@team2:~$ sudo ovs-ofctl dump-flows ovsbridge
team2@team2:~$ sudo ovs-ofctl dump-flows ovsbridge
team2@team2:~$ sudo ovs-appctl fdb/flush ovsbridge
table successfully flushed
team2@team2:~$ sudo ovs-appctl fdb/show ovsbridge
  port  VLAN  MAC          Age
team2@team2:~$ sudo ovs-appctl fdb/show ovsbridge
  port  VLAN  MAC          Age
    1      0  5c:26:0a:24:8f:a0   0
    2      0  5c:26:0a:24:8d:7a   0
team2@team2:~$ sudo ovs-appctl fdb/show ovsbridge
  port  VLAN  MAC          Age
    1      0  5c:26:0a:24:8f:a0   0
    2      0  5c:26:0a:24:8d:7a   0
team2@team2:~$
```

Flow tables are not present in the switch, but it behaves as an L2 switch.

As it can be seen above the flow table is empty, and the mac table is flushed as well.

When we start a ping the switch learns the mac-addresses basically performing regular L2 forwarding.

9. Establish the connection between OVS and the controller.

- Issue a continuous ping between hosts.
- Configure OVS to be in failover secure mode.
- Drop the connection between the OVS/controller.
  - Do the pings continue? Why/why not? Show the flow table to help with your explanation [5 points]

Ping works.

```
l_dst=5c:26:0a:24:8f:a0 actions=output:eno1
team2@team2:~$ sudo ovs-vsctl set-fail-mode ovsbridge secure
team2@team2:~$ sudo ovs-vsctl set-controller ovsbridge tcp:192.168.1.7:6634
team2@team2:~$ sudo ovs-ofctl dump-flows ovsbridge
  cookie=0x0, duration=195.860s, table=0, n_packets=74, n_bytes=5336, in_port=eno1,dl_src=5c:26:0a:24:8f:a0,dl_dst=5c:26:0a:24:8d:7a actions=output:eno2
  cookie=0x0, duration=142.494s, table=0, n_packets=69, n_bytes=5022, in_port=eno2,dl_src=5c:26:0a:24:8d:7a,dl_dst=5c:26:0a:24:8f:a0 actions=output:eno1
team2@team2:~$
```

In the failover secure mode the flow tables are preserved even when the connection with the controller is lost.

We configured the controller port as wrong, thus causing the controller connection to be lost,

but the flows are still preserved. The switch can still do regular forwarding of the packets, since the flows are preserved allowing the data plane to remain unaffected.

When the switch comes back up it can send a BARRIER request to check if it is in sync with the controller allowing the flows to be updated if necessary.

## Objective 5 – Python script to capture, display and save

1. For this objective you have to create a Python script that captures packets using tcpdump between the Mininet switches and the ODL controller.
2. Initialize a linear Mininet topology with two switches and connect it to the ODL controller.
3. Your script should capture OpenFlow packets and identify when a switch has made a successful OpenFlow connection to the controller.
  - a. How will you identify a switch has been successfully connected to ODL from the tcpdump captures? **[5 points]**

### **Algorithm -**

1. Check if the packet contains TCP header
2. Check if the packet is intended to the controller and destination port is 6653
3. Check if the TCP packet has a payload
4. Extract the Feature Request code (05) from the payload and record the destination IP and destination port.  
*This indicates that the hello packets were exchanged and the controller is expecting more information from the switch, related to ports and table support. —> connection established*
5. Check if the Openflow payload has Feature Reply code (06) and extract data\_path\_id. Also, record the source Ip and source Port of the packet.  
*This gives dpid of the switch*

4. The script should print out the dpid of the connected switch and also save this information in a file connected.txt in JSON format.

Format –

```
{ "<dpid>" : { "ip_address" : "<IP address of switch>", "status" : "<connected/not connected>" } }
```

5. Paste screenshots of the script output and the text file saved. [20 points]

**Script output ->**

```
sdn@sdn-controllers:~$ sudo python3 openflowMonitor.py
192.168.56.104 -----Openflow Hello Message-----> 192.168.56.103
192.168.56.104 -----Openflow Hello Message-----> 192.168.56.103
192.168.56.103 -----Openflow Hello Message-----> 192.168.56.104
192.168.56.103 -----Openflow Feature Request-----> 192.168.56.104
{('192.168.56.104', '42380'): 'Connected'}
192.168.56.103 -----Openflow Hello Message-----> 192.168.56.104
{('192.168.56.104', '42380'): 'Connected'}
192.168.56.103 -----Openflow Feature Request-----> 192.168.56.104
{('192.168.56.104', '42382'): 'Connected', ('192.168.56.104', '42380'): 'Connected'}
192.168.56.104 -----Openflow Feature Reply-----> 192.168.56.103
{('192.168.56.104', '42382'): 'Connected', ('192.168.56.104', '42380'): 'Connected'}
{('192.168.56.104', '42380'): '0000000000000002'}
192.168.56.104 -----Openflow Feature Reply-----> 192.168.56.103
{('192.168.56.104', '42382'): 'Connected', ('192.168.56.104', '42380'): 'Connected'}
{('192.168.56.104', '42382'): '0000000000000001', ('192.168.56.104', '42380'): '0000000000000002'}
192.168.56.103 -----Openflow Modification-----> 192.168.56.104
{('192.168.56.104', '42382'): 'Connected', ('192.168.56.104', '42380'): 'Connected'}
{('192.168.56.104', '42382'): '0000000000000001', ('192.168.56.104', '42380'): '0000000000000002'}
192.168.56.103 -----Openflow Modification-----> 192.168.56.104
{('192.168.56.104', '42382'): 'Connected', ('192.168.56.104', '42380'): 'Connected'}
{('192.168.56.104', '42382'): '0000000000000001', ('192.168.56.104', '42380'): '0000000000000002'}
```

**File generated -**

```
^Csdn@sdn-controllers:~$ cat switch_Records.json
{
    "0000000000000001": {
        "socket_port": "42382",
        "status": "Connected",
        "ip_address": "192.168.56.104"
    },
    "0000000000000002": {
        "socket_port": "42380",
        "status": "Connected",
        "ip_address": "192.168.56.104"
    }
}sdn@sdn-controllers:~$
```

6. Connect another OvS in the Mininet VM to the controller (in addition to the two switches connected before). Provide screenshot of the commands used for this purpose.

[5 points]

```
mininet@mininet-ofm:~$ sudo ovs-vsctl add-br ovs1
mininet@mininet-ofm:~$ sudo ovs-vsctl set bridge ovs1 protocols=OpenFlow13
mininet@mininet-ofm:~$ sudo ovs-vsctl set-controller ovs1 tcp:192.168.56.103:6653
mininet@mininet-ofm:~$ █
```

7. Paste screenshots of the script output and the updated text file after the new switch connects to the controller. [5 points]

## Cli output of the code –

```
{('192.168.56.104', '57554'): 'Connected', ('192.168.56.104', '57556'): 'Connected'}
{('192.168.56.104', '57554'): '0000000000000002', ('192.168.56.104', '57556'): '0000000000000001'}
192.168.56.103 --Openflow Modification--> 192.168.56.104
{('192.168.56.104', '57554'): 'Connected', ('192.168.56.104', '57556'): 'Connected'}
{('192.168.56.104', '57554'): '0000000000000002', ('192.168.56.104', '57556'): '0000000000000001'}
192.168.56.104 --Openflow Hello Message--> 192.168.56.103
{('192.168.56.104', '57554'): 'Connected', ('192.168.56.104', '57556'): 'Connected'}
{('192.168.56.104', '57554'): '0000000000000002', ('192.168.56.104', '57556'): '0000000000000001'}
192.168.56.103 --Openflow Hello Message--> 192.168.56.104
{('192.168.56.104', '57554'): 'Connected', ('192.168.56.104', '57556'): 'Connected'}
{('192.168.56.104', '57554'): '0000000000000002', ('192.168.56.104', '57556'): '0000000000000001'}
192.168.56.103 --Openflow Feature Request--> 192.168.56.104
{('192.168.56.104', '57558'): 'Connected', ('192.168.56.104', '57554'): 'Connected', ('192.168.56.104', '57556'): 'Connected'}
{('192.168.56.104', '57554'): '0000000000000002', ('192.168.56.104', '57556'): '0000000000000001'}
192.168.56.104 --Openflow Feature Reply--> 192.168.56.103
{('192.168.56.104', '57558'): 'Connected', ('192.168.56.104', '57554'): 'Connected', ('192.168.56.104', '57556'): 'Connected'}
{('192.168.56.104', '57554'): '00009fe11ad3548', ('192.168.56.104', '57554'): '0000000000000002', ('192.168.56.104', '57556'): '0000000000000001'}
192.168.56.103 --Openflow Modification--> 192.168.56.104
{('192.168.56.104', '57558'): 'Connected', ('192.168.56.104', '57554'): 'Connected', ('192.168.56.104', '57556'): 'Connected'}
{('192.168.56.104', '57554'): '00009fe11ad3548', ('192.168.56.104', '57554'): '0000000000000002', ('192.168.56.104', '57556'): '0000000000000001'}
```

```
^Csdn@sdn-controllers:~$ cat switch_Records.json
```

```
"0000000000000002": {
    "ip_address": "192.168.56.104",
    "socket_port": "57554",
    "status": "Connected"
},
"000096fe11ad3548": {
    "ip_address": "192.168.56.104",
    "socket_port": "57558",
    "status": "Connected"
},
"0000000000000001": {
    "ip_address": "192.168.56.104",
    "socket_port": "57556",
    "status": "Connected"
}
}
}sdn@sdn-controllers:~$ █
```

8. Describe a use case wherein a file like connected.txt would be useful for network management. [2 points]

**connected.txt / switch\_Records.txt** would be useful to check which switches are connected, which are disconnected. A script can be written that can monitor this file and

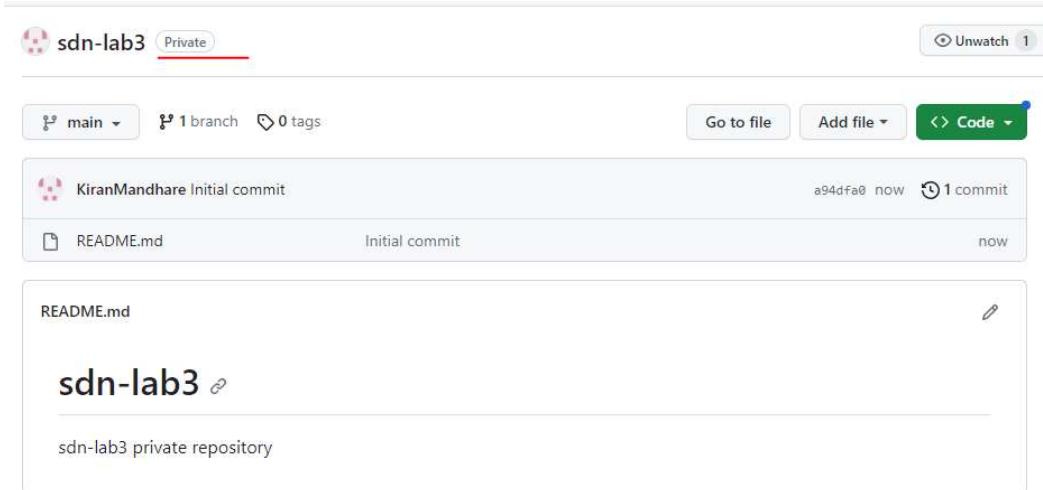
generate alerts, or take some troubleshooting actions allowing to write self healing code or monitoring piece of code. This txt file can also be used to create a dashboard and can be used to understand the current health of different switches in the network.

This txt file for 3-4 switches doesn't serve a great purpose but as soon as we run into scalability such scripts come handy for understanding the network at given instance of time.

## Objective 6 – Version control using Git and GitHub

Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files. GitHub provides hosting service for software development version control using Git.

1. Create a free account on GitHub if you don't already have one.
2. Set up a **private** repository with the name sdn-lab3. Provide a screenshot of the repository created as well as its private setting. **[5 points]**



The screenshot shows a GitHub repository page for 'sdn-lab3'. The repository is marked as 'Private'. It has 1 branch and 0 tags. There is 1 commit by 'KiranMandhare' titled 'Initial commit', dated 'now', with a commit hash 'a94dfa0'. The README.md file contains the text 'sdn-lab3'.

3. Now push the scripts you created in Objectives 4 and 5 from your local system to your private repo on GitHub using git commands. Provide screenshots of the git commands used and of the scripts present in your GitHub repo. **[5 points]**

Initialize the local repository and point it to the remote repository using **remote add origin** command

```
kiran@DESKTOP-9CMJKPK MINGW64 /d/sdn-lab3
$ git init
Initialized empty Git repository in D:/sdn-lab3/.git/
kiran@DESKTOP-9CMJKPK MINGW64 /d/sdn-lab3 (master)
$ git remote add origin https://github.com/KiranMandhare/sdn-lab3.git
```

Good practice to pull before adding and making sure that the local repository is in sync with the remote repository. **git pull** → pulls latest changes and merges with local repository

```
kiran@DESKTOP-9CMJKPK MINGW64 /d/sdn-lab3 (master)
$ git pull https://github.com/KiranMandhare/sdn-lab3.git master
From https://github.com/KiranMandhare/sdn-lab3
 * branch            master      -> FETCH_HEAD
```

set the **git pull** command to pull from the remote repository and from master branch

```
kiran@DESKTOP-9CMJKPK MINGW64 /d/sdn-lab3 (master)
$ git branch --set-upstream-to=origin/master master
branch 'master' set up to track 'origin/master'.
```

check **git status** → see if there are new files that are not pushed to remote repository

```
kiran@DESKTOP-9CMJKPK MINGW64 /d/sdn-lab3 (master)
$ ls -ltr
total 1
-rw-r--r-- 1 kiran 197609 41 Sep 25 04:49 README.md

kiran@DESKTOP-9CMJKPK MINGW64 /d/sdn-lab3 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    createOVSBridge.py
    openflow.py

nothing added to commit but untracked files present (use "git add" to track)
```

**git add .** → add all new files to staging area

**git commit -m** → saves the update in the local repository

**git push** → pushes all the commits to the remote repository

**git push origin master** → pushes master local branch to the configured origin, which is the url of the remote repository

```
kiran@DESKTOP-9CMJKPK MINGW64 /d/sdn-lab3 (master)
$ git add .

kiran@DESKTOP-9CMJKPK MINGW64 /d/sdn-lab3 (master)
$ git commit -m "python code"
[master 6726320] python code
 2 files changed, 84 insertions(+)
 create mode 100644 createOVSBridge.py
 create mode 100644 openflow.py

kiran@DESKTOP-9CMJKPK MINGW64 /d/sdn-lab3 (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.45 KiB | 1.46 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/KiranMandhare/sdn-lab3.git
 a94dfa0..6726320  master -> master
```

The screenshot shows a GitHub repository page for 'sdn-lab3'. The repository is private, as indicated by the 'Private' badge. It has 1 branch and 0 tags. The most recent commit was made by Kiran Mandhare 1 hour ago, with a commit message of 'Initial commit'. The repository contains four files: README.md, createOVSBridge.py, openflow.py, and sdn-lab3Doc.pdf. The README.md file is shown in its entirety.

KiranMandhare / sdn-lab3

Code Issues Pull requests Actions Projects Security Insights Settings

sdn-lab3 Private Unwatch 1

master 1 branch 0 tags Go to file Add file Code

Kiran Mandhare and Kiran Mandhare lab doc		4abd43b 1 hour ago	4 commits
README.md	Initial commit	17 hours ago	
createOVSBridge.py	createOVSBridge file tweaked	7 hours ago	
openflow.py	python code	16 hours ago	
sdn-lab3Doc.pdf	lab doc	1 hour ago	

README.md

sdn-lab3

sdn-lab3 private repository

4. Describe a use case of version control/Git from the perspective of a network engineer. [2 points]

Git can be used to maintain a network as code, and this can help allow other developers to reuse the same automation rather than reinventing the wheels. It can also allow multiple network developers to collaborate on the tasks, allowing collaborative automation, which can be huge and more manageable.

Total Score = \_\_\_\_\_ / 230