Software-Defined Networking

Lab 3
OpenFlow and Wireshark

University of Colorado Boulder
Department of Computer Science

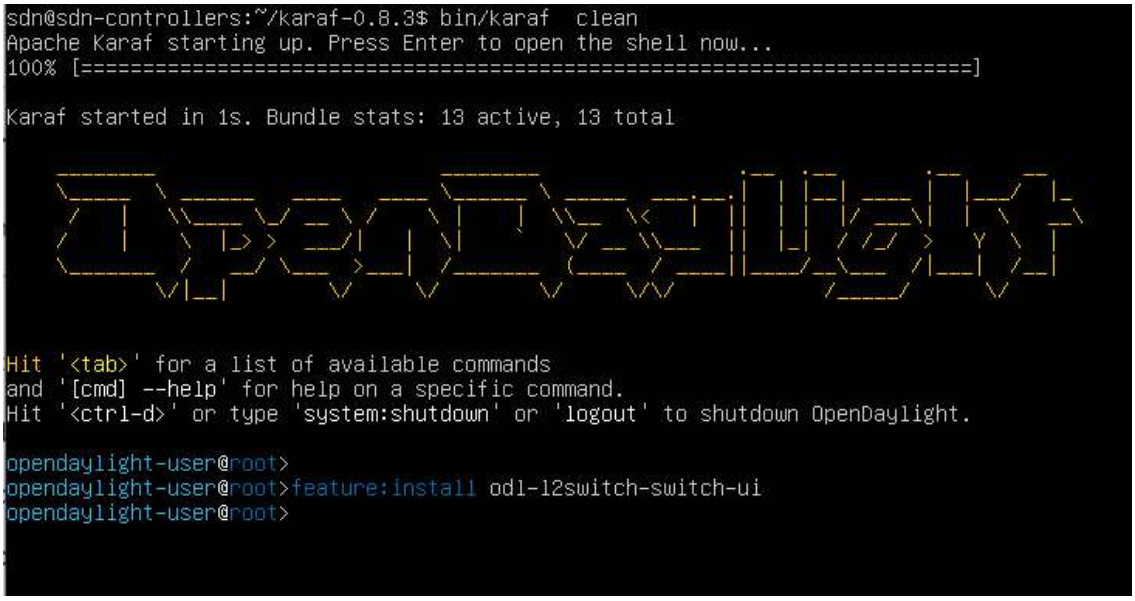
Professor Levi Perigo, Ph.D.

Lab Summary

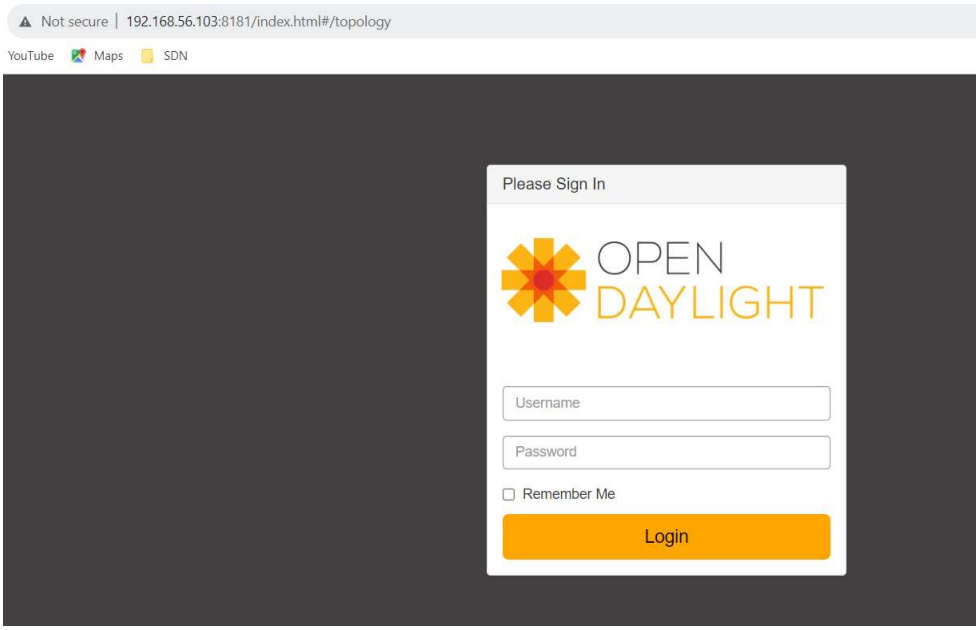
Understanding the OpenFlow protocol is critical to understanding SDN. The purpose of this lab is to capture and analyze the OpenFlow protocol messages using Wireshark. Knowing how OpenFlow works and being able to troubleshoot different scenarios within Wireshark, will prove to be a valuable skillset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

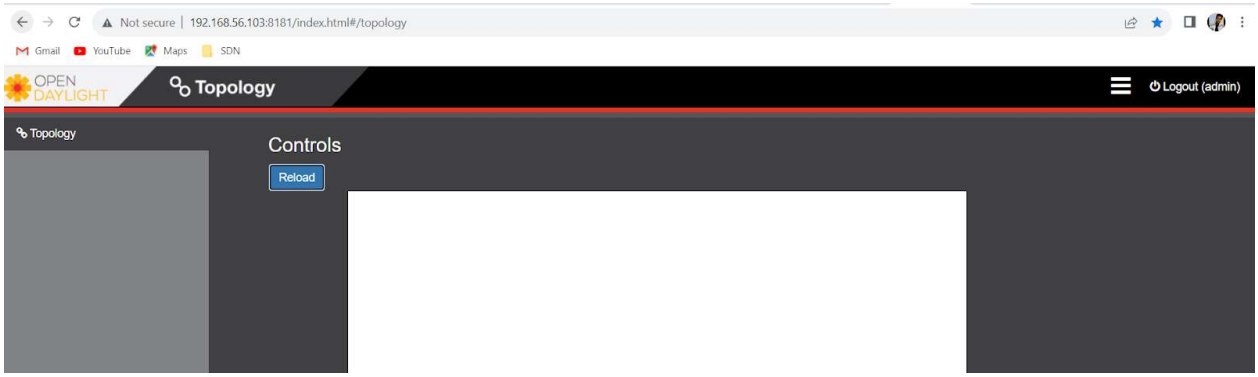
Objective 1 – X11 Forwarding: Setup the SDN (controller/switch) Environment

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**]



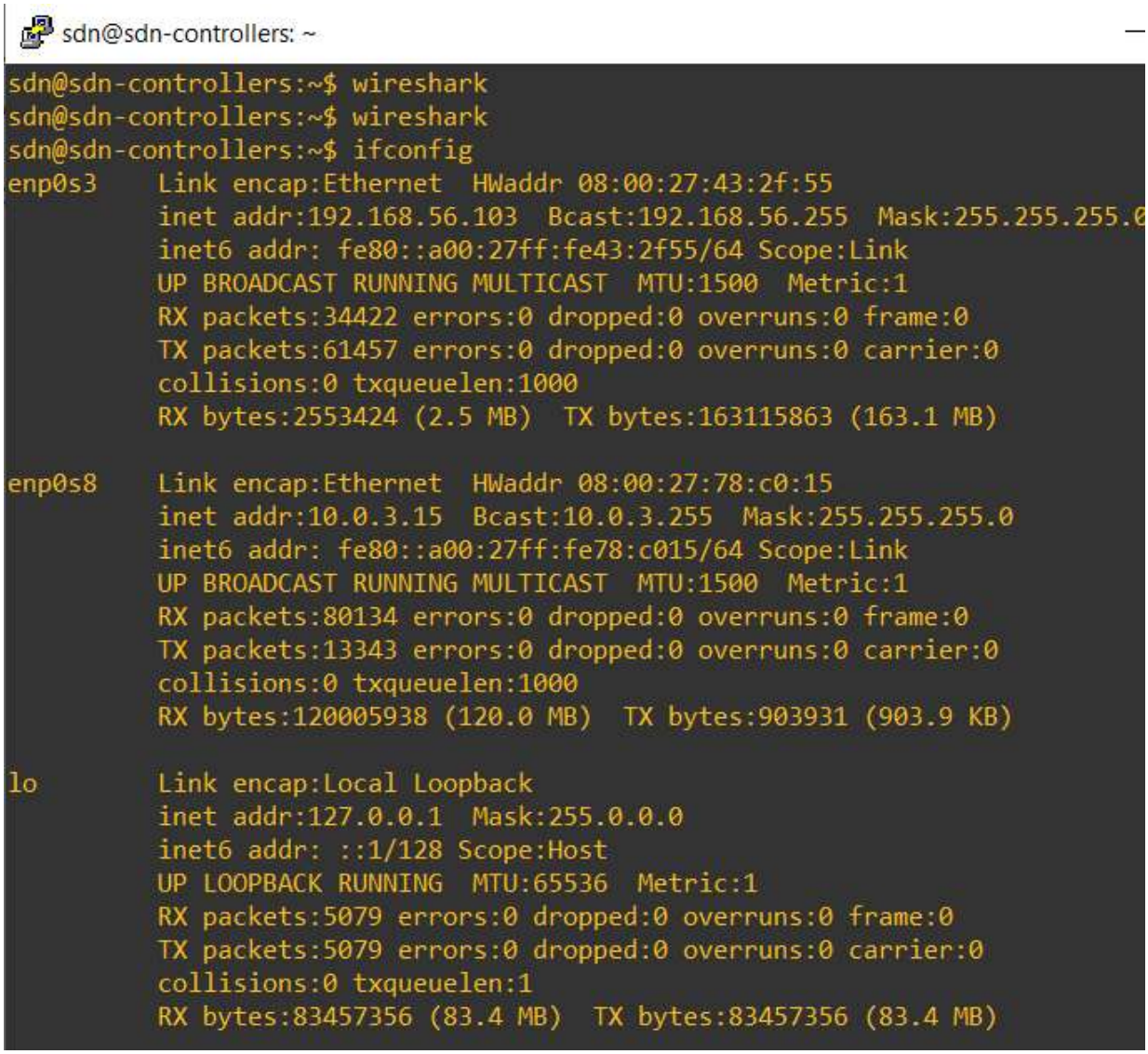
5. Enable the ODL web interface and all the features (use the Lab 0 document for assistance).
6. Login to the ODL web interface.
    a. Provide a screenshot of your local machine at the ODL home page. [**10 points**]

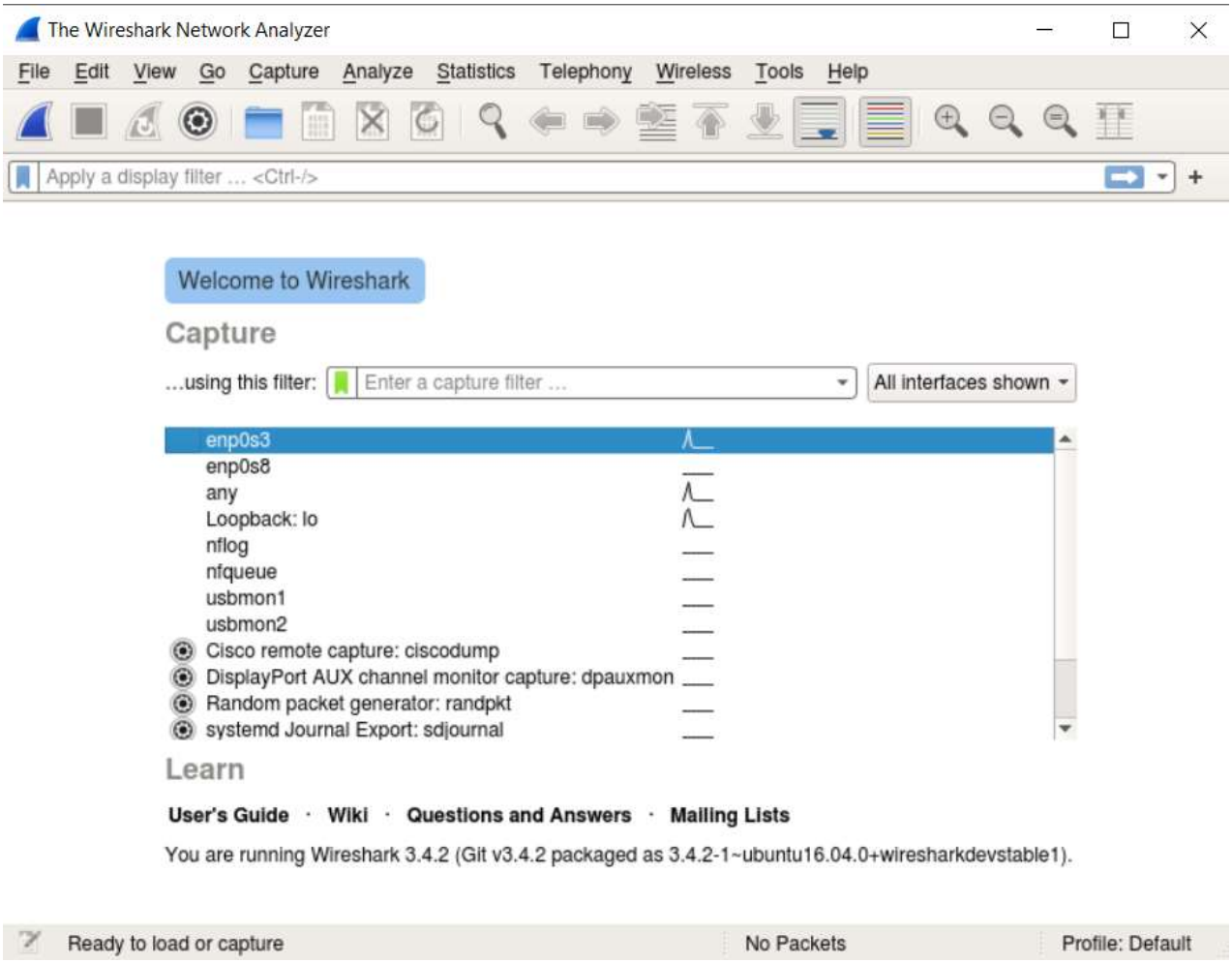




7. Start a new PuTTy/SSH session to the OpenDaylight controller.
    a. Before you connect, in PuTTy go to Connection > SSH > X11.
    b. Check the “Enable X11 forwarding” box.
        i. Note: If using MAC/Linux you can simply SSH with the “-X”.
    c. Initiate the PuTTy connection.
    d. Install and 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**]





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**]

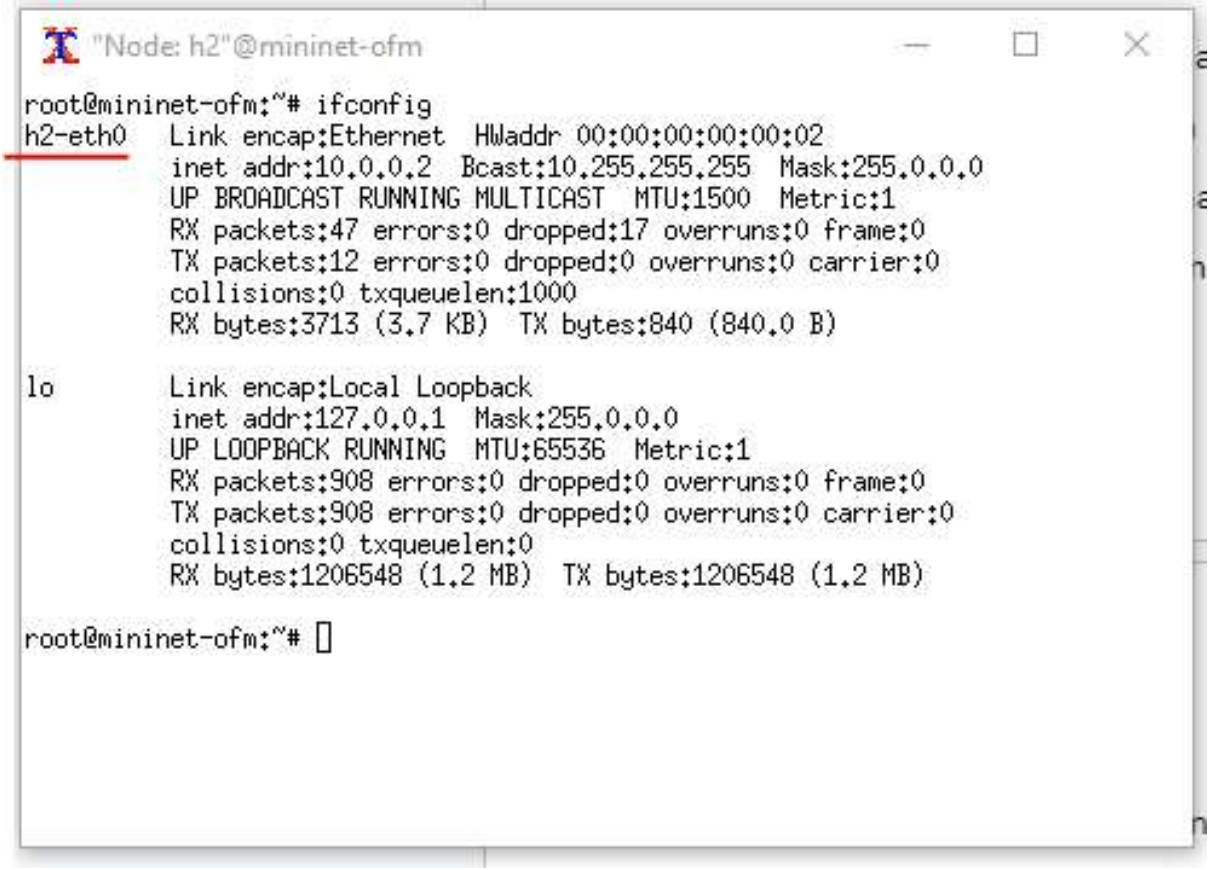
```
mininet@mininet-ofm:~$ sudo mn --controller=remote,ip=192.168.56.103 --mac --swi
tch=ovsk,protocols=OpenFlow13  --topo=single,4
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.56.103:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(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.

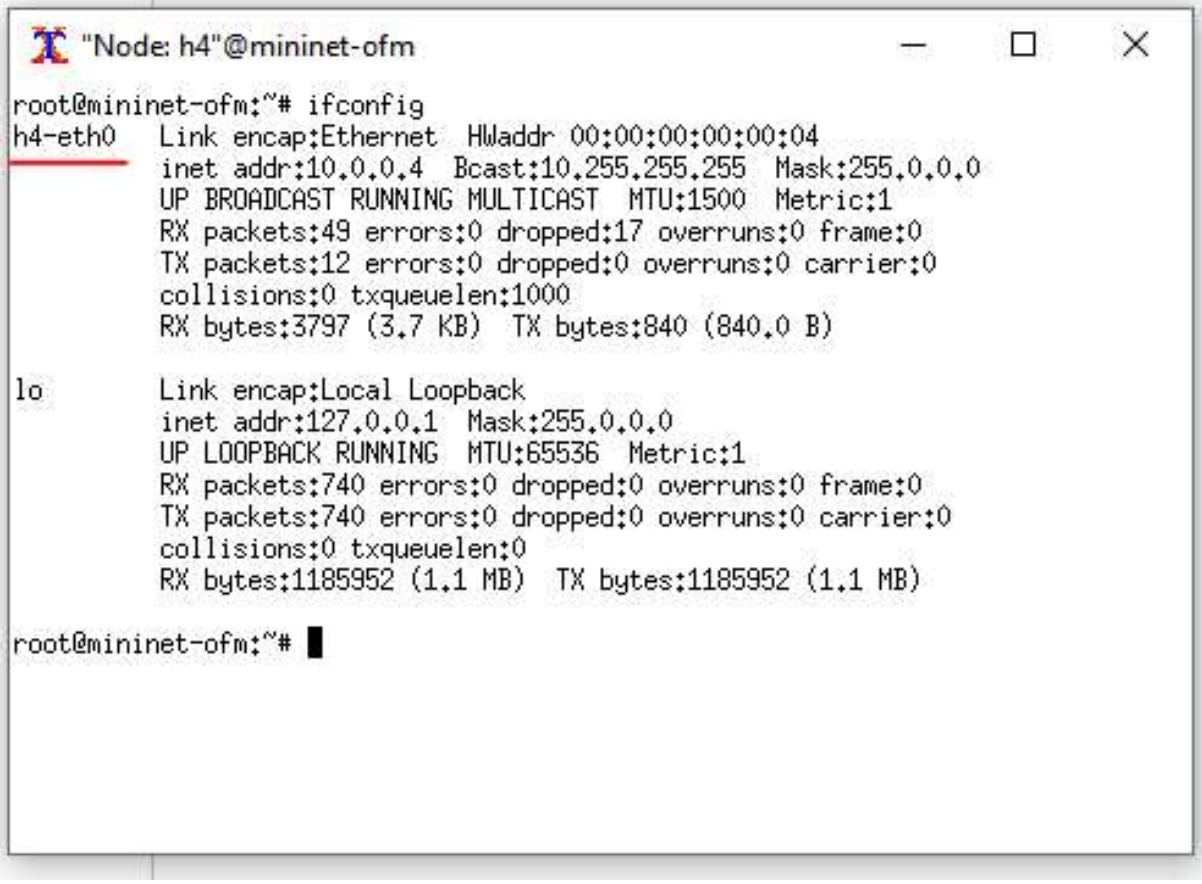
   i. Explain what happened, what each of these new windows are, why this would be beneficial; also provide a screenshot of all four windows (proving you got it working). [**10 points**]

   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.

switch and two hosts.

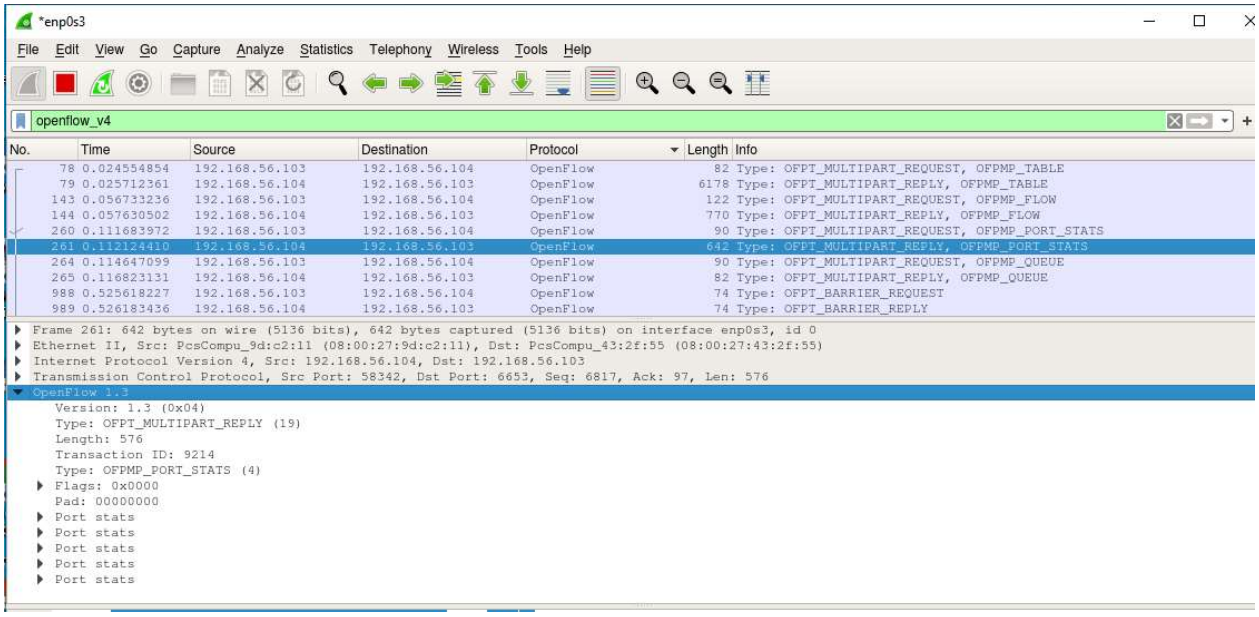



"Node: h3"@mininet-ofm
root@mininet-ofm:~# ifconfig
h3-eth0 Link encap:Ethernet HWaddr 00:00:00:00:00:03
inet addr:10.0.0.3 Bcast:10.255.255.255 Mask:255.0.0.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:40 errors:0 dropped:8 overruns:0 frame:0
TX packets:12 errors:0 dropped:0 overruns:0 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:~#


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

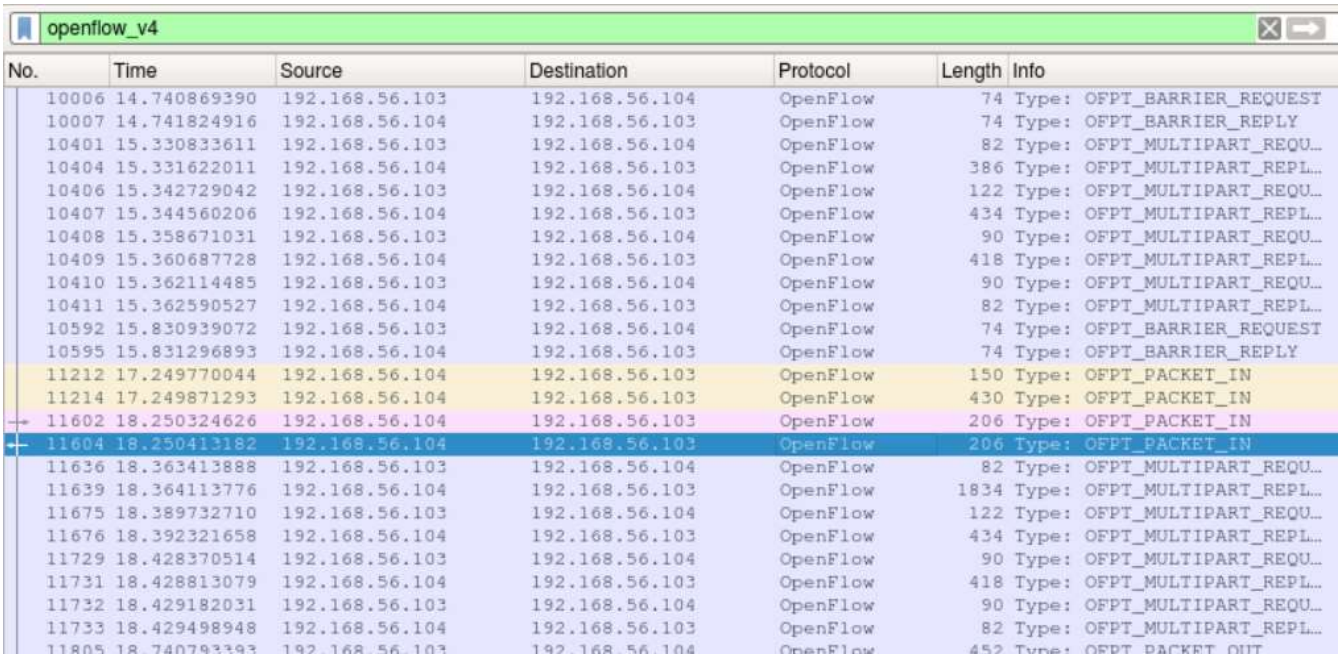
```
mininet> h1 ping h2 -c 1
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.464 ms

--- 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>
```

    c. 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_REQUE…
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_REQU…
8261	12.174549944	192.168.56.104	192.168.56.103	OpenFlow	1138	Type: OFPT_MULTIPART_REPL…
8262	12.204263447	192.168.56.103	192.168.56.104	OpenFlow	114	Type: OFPT_MULTIPART_REQU…
8263	12.205062375	192.168.56.104	192.168.56.103	OpenFlow	98	Type: OFPT_MULTIPART_REPL…
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_REPL…
8281	12.272099672	192.168.56.103	192.168.56.104	OpenFlow	82	Type: OFPT_MULTIPART_REQU…
8282	12.272611582	192.168.56.104	192.168.56.103	OpenFlow	6178	Type: OFPT_MULTIPART_REPL…
8284	12.284150761	192.168.56.103	192.168.56.104	OpenFlow	122	Type: OFPT_MULTIPART_REQU…
8285	12.284500582	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_MULTIPART_REPL…
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_REQU…
8291	12.292097057	192.168.56.104	192.168.56.103	OpenFlow	418	Type: OFPT_MULTIPART_REPL…
8292	12.293245424	192.168.56.103	192.168.56.104	OpenFlow	90	Type: OFPT_MULTIPART_REQU…
8293	12.308112846	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_MULTIPART_REPL…
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



openflow_v4

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_REQU…
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_REQU…
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_REQU…
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_REQU…
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_REQU…
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_REQU…
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_REQU…
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_REQU…
11733	18.429498948	192.168.56.104	192.168.56.103	OpenFlow	82	Type: OFPT_MULTIPART_REPL…

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**

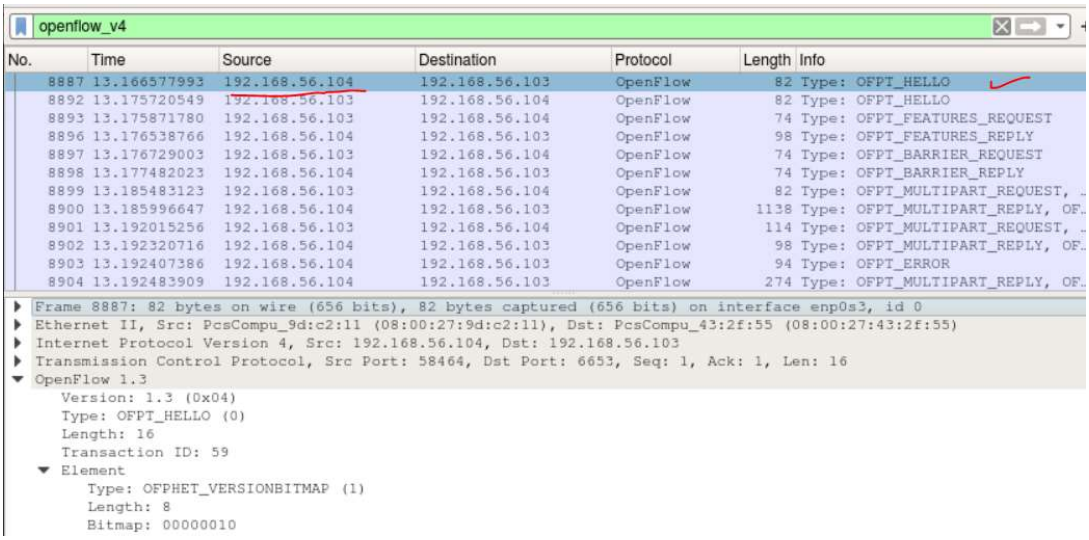
```
mininet> s1 ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:86:a6:6d
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
   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

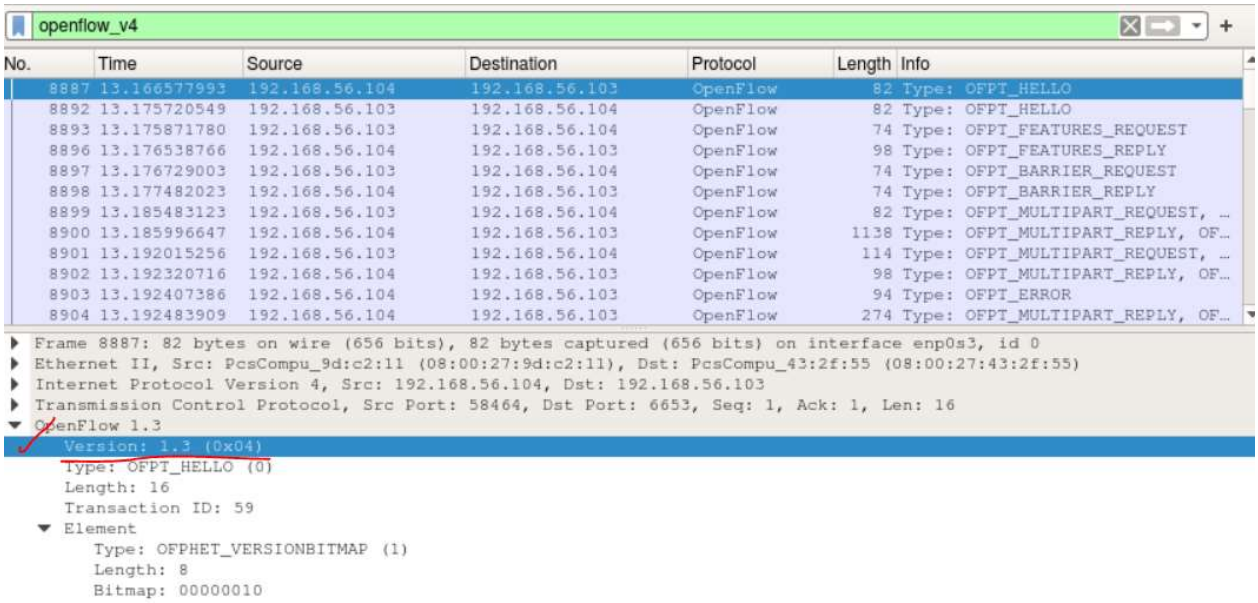
   a. Provide a screenshot of the capture that indicates this. [**2 points**]

The connection is initiated by the **switch**

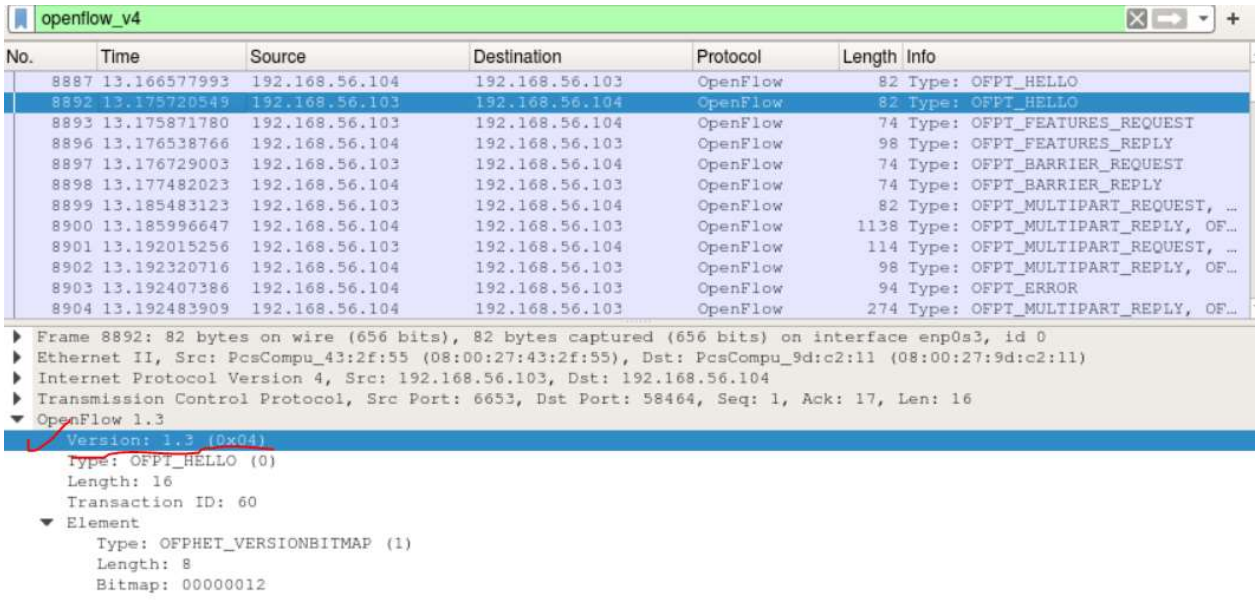


7. Within Wireshark indicate the messages from the switch and the controller that indicate which version of OpenFlow they support.
    a. Provide a screenshot of each. [**10 points**]

Hello message from the switch

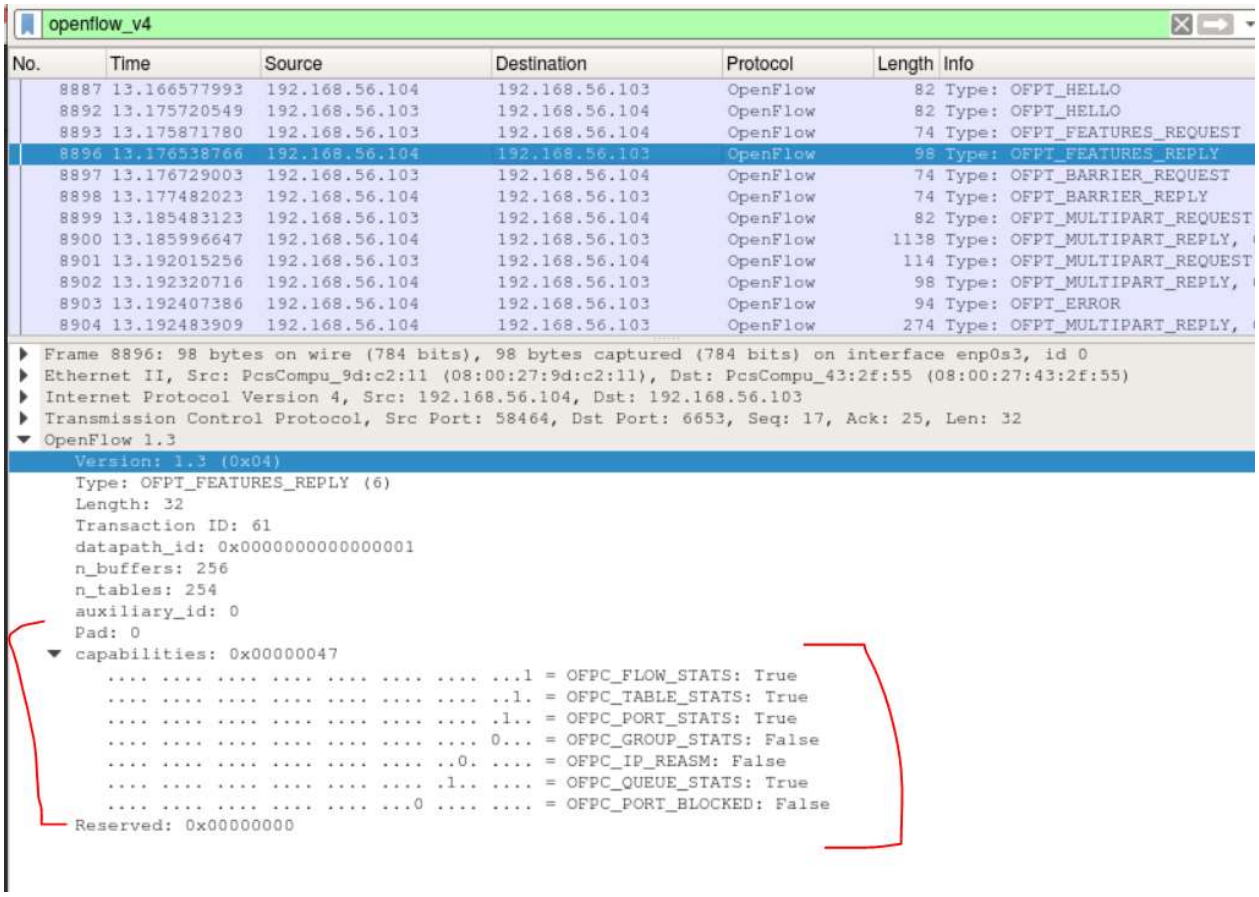


Hello message from controller



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

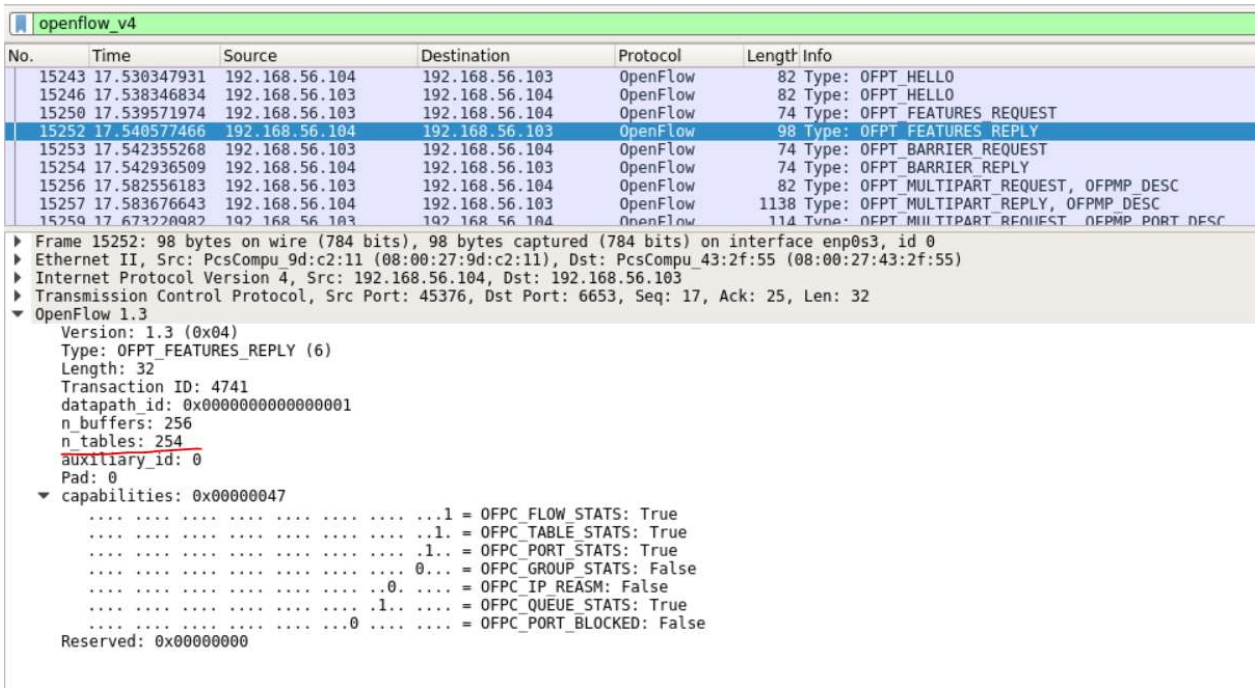
a. Provide a screenshot of the Wireshark message, indicating the features. [**5 points**]



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

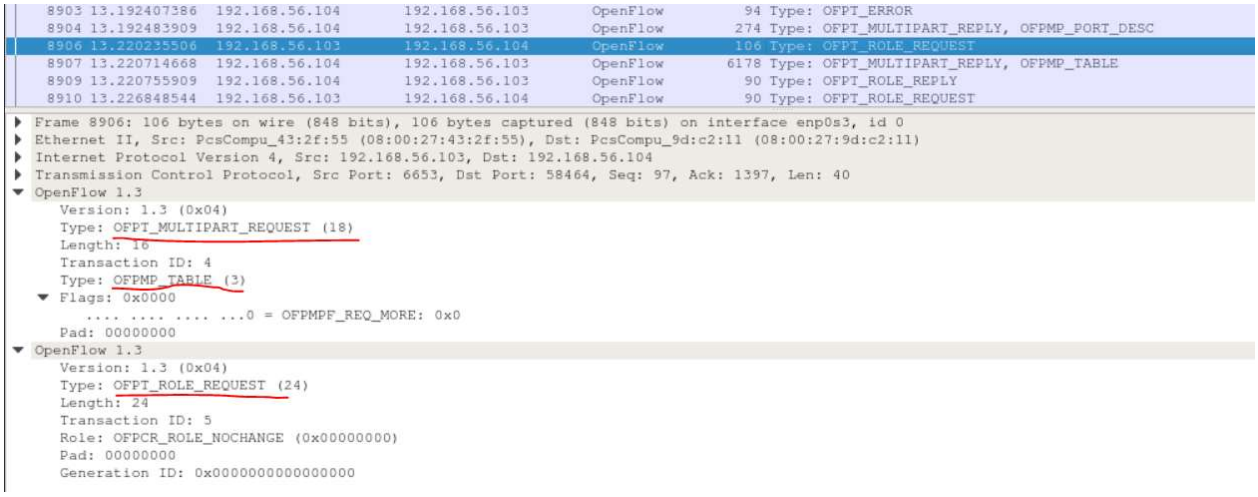


10. The controller sent a OFPT_MULTIPART_REQUEST, and the switch replied with a “REPLY.”

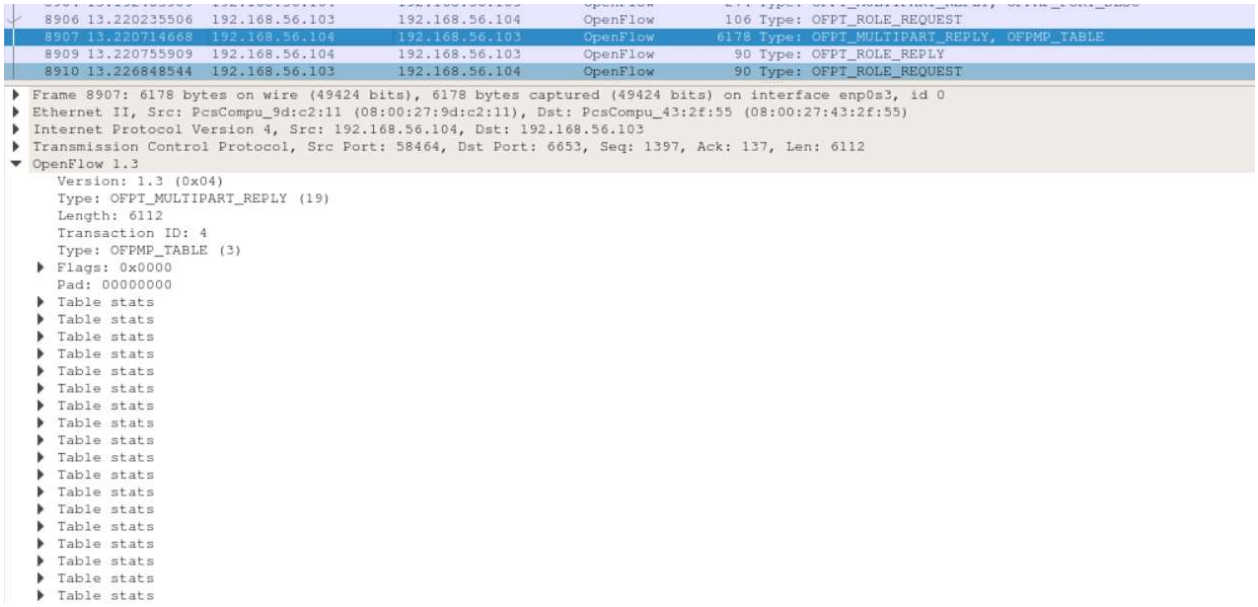
    Explain what a multipart OpenFlow message is, and why/when they are used, also indicate, by showing in the capture, one switch reply to a multipart request. [**5 points**]

    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

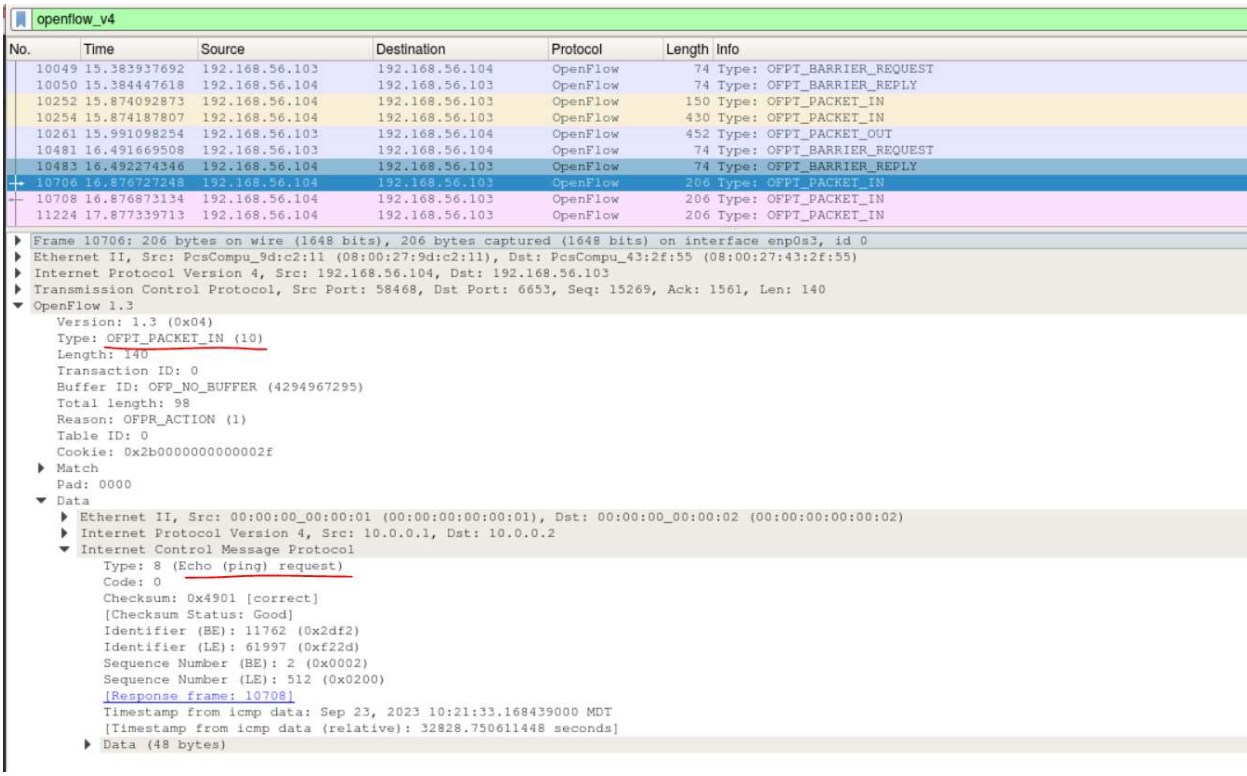
Request Packet for Table stats and Role in the same TCP packet



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**]



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.

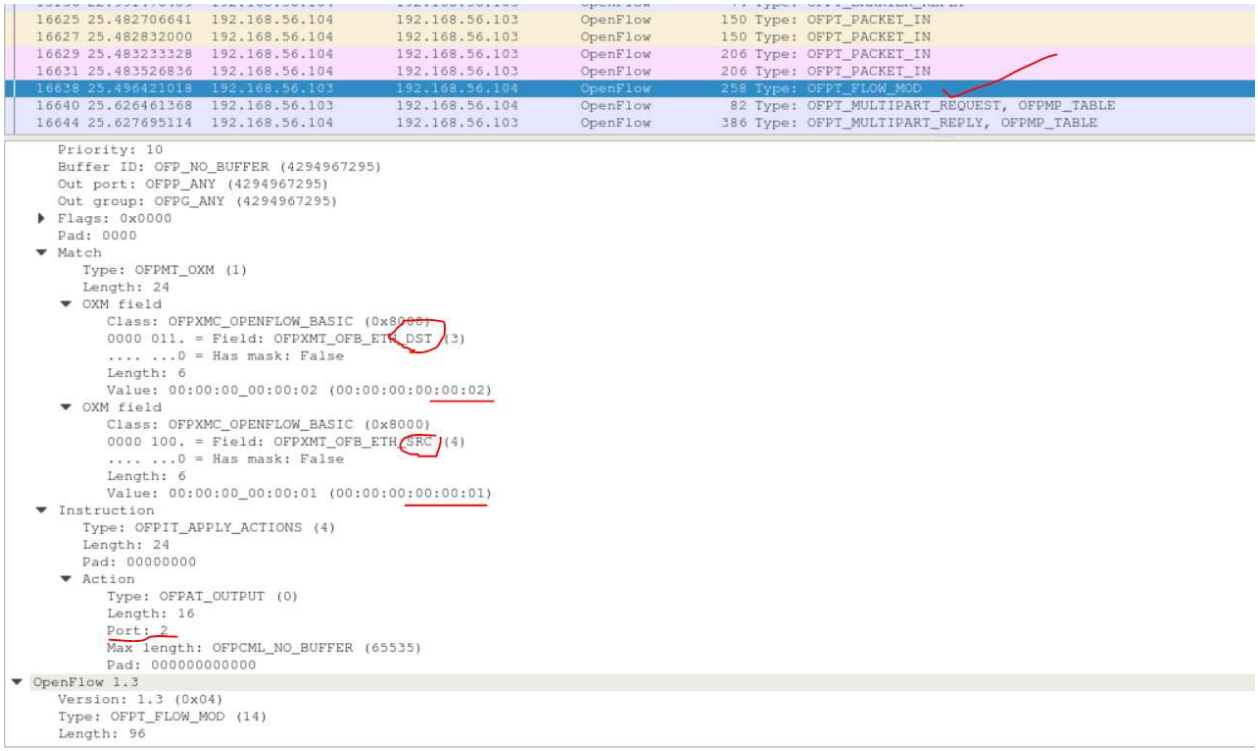
d. Indicate the src MAC and dst MAC, and src IP and dst IP in this OpenFlow packet. Are these the same or different from the actual src/dst of the packet in Wireshark? Why? [**3 points**]

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**]

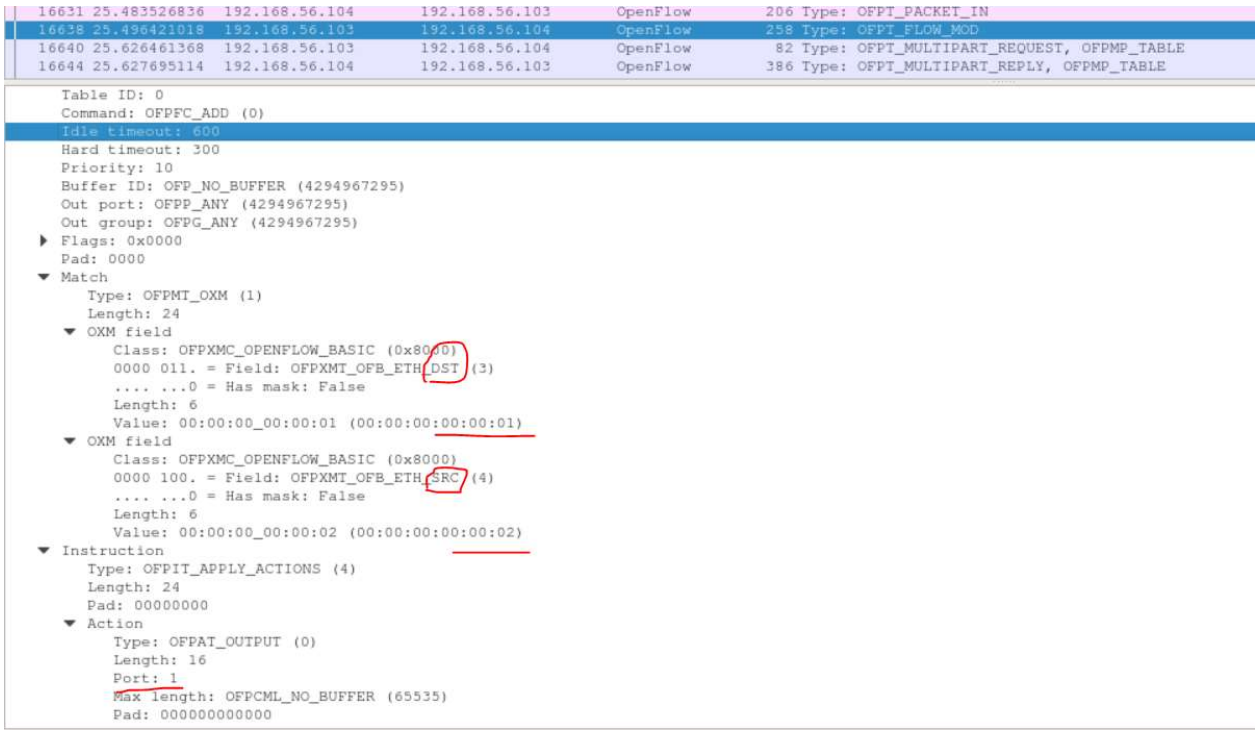


2 flows are installed with match criteria as:

Flows

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

   Action: Egress on Port 2

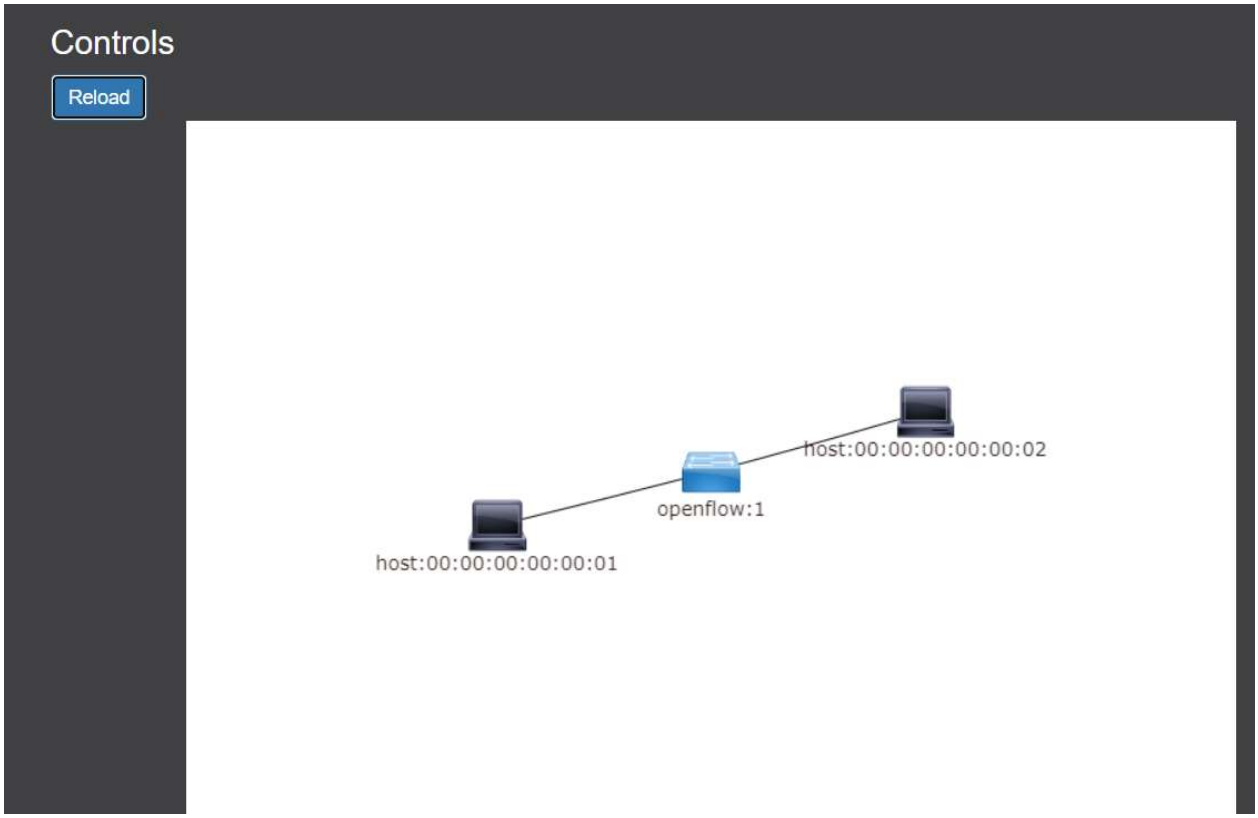


2. **Match** srcMAC: 0000.0000.0002 destMAC: 0000.0000.0001

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

    sudo mn –clean

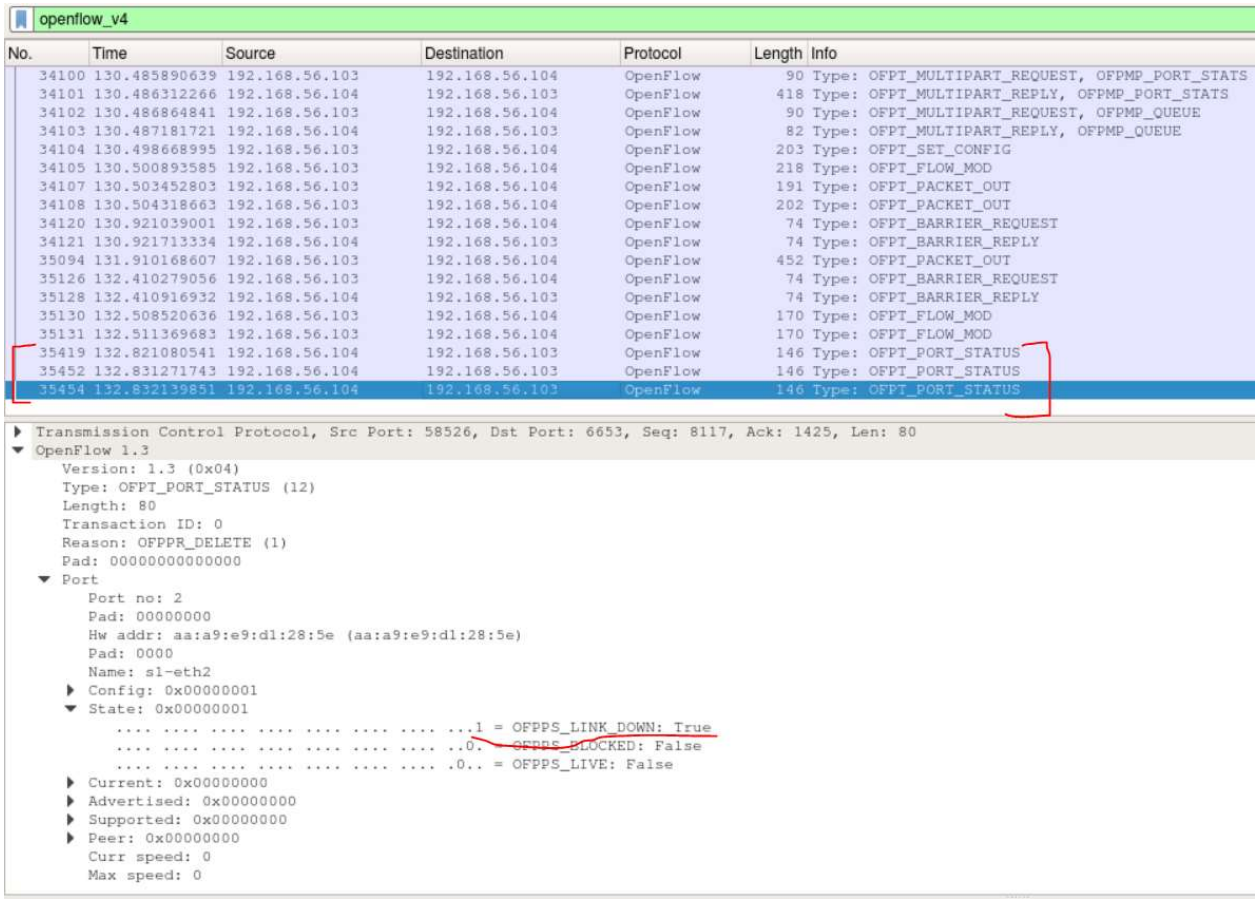
```
mininet@mininet-ofm:~$ sudo mn --clean
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest m
nexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-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/vconn* /tmp/vlogs* /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**]

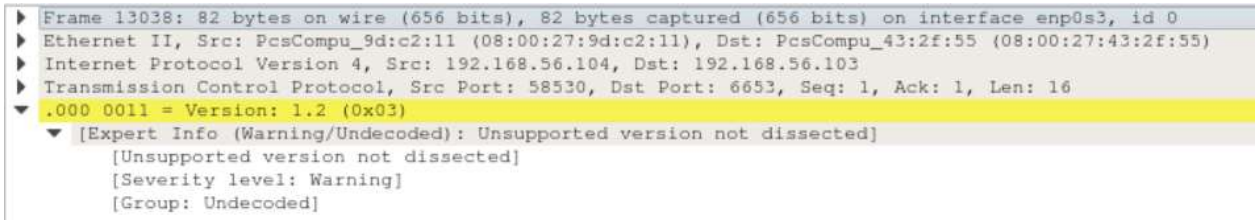
   Once mininet is cleared, the switches go down and the ODL topology has nothing to show in the dashboard. Hence, there is no topology shown.

   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.

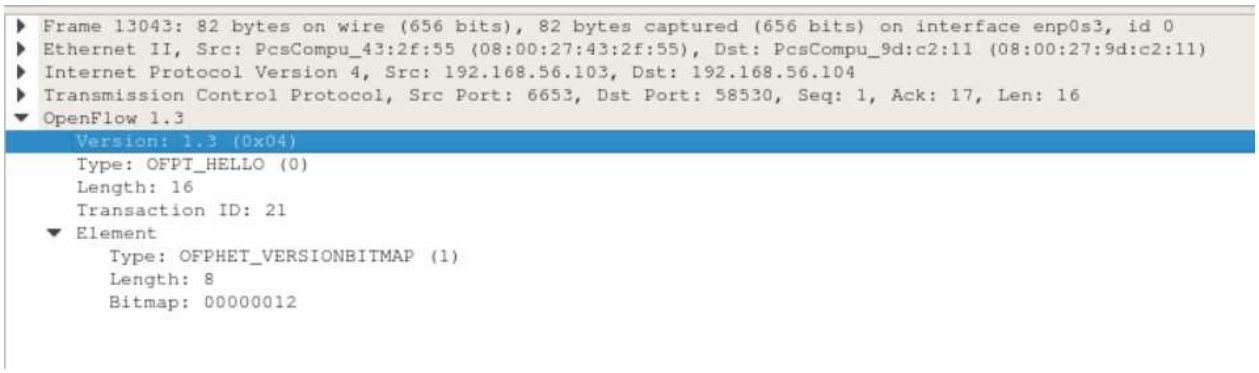


3. Use the same steps above to start a new Wireshark capture
4. Use the same steps above to initiate a new Mininet topology. Use the same topology used before in step 2.4, but this time instead of using OpenFlow 1.3 use version 1.2.
   a. Did the switch connect to the controller? Why or why not? How do you know? Provide screenshots of the capture to support your answer. [**15 points**]

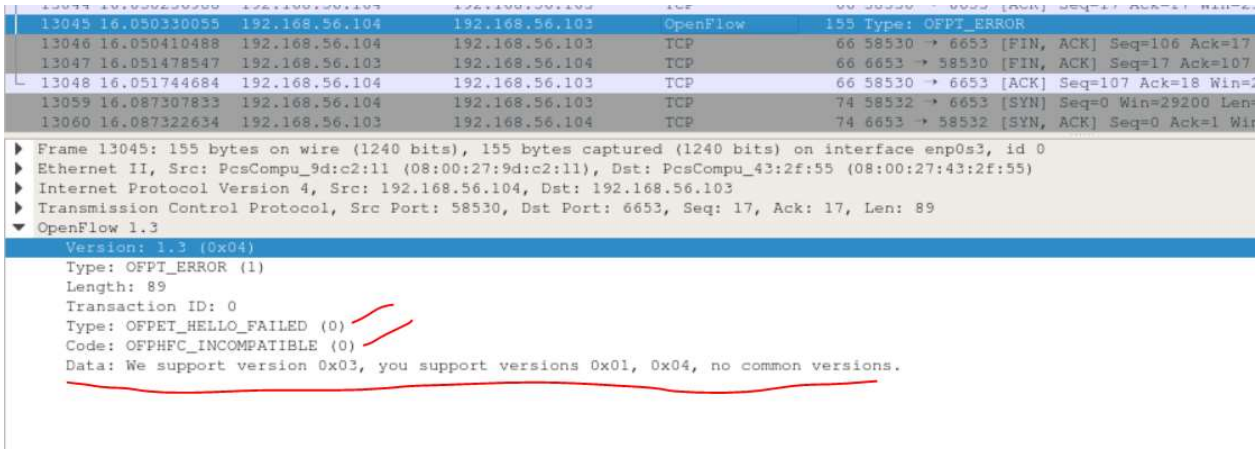
Hello message sent from the switch to the controller (this message is because the wireshark doesn't know this version, hence the warning)



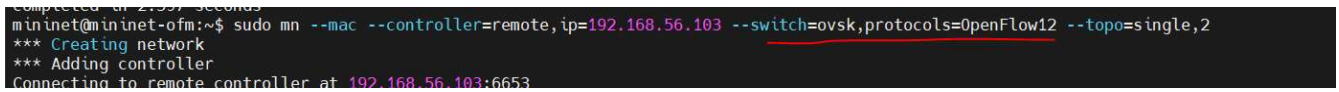
the hello message sent by controller to switch



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



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.



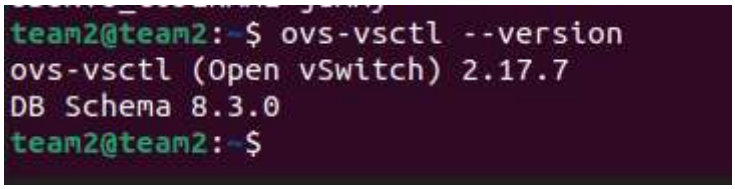
Since there is no controller connected and the switch has no flow , the packets are simply dropped. Thus ping won't work

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-

   OVS Version -



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 –
    a. Automates the steps required to connect your OvS to the controller.
    b. Verifies and displays if the OpenFlow connection is successful.
    c. 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 conroller sucessfully
OVS Bridge ovsbridge fail-mode set to SECURE
team2@team2:~/Downloads$
```

8. Issue a continuous ping between hosts.
    a. 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 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.

   i. 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.
    a. Issue a continuous ping between hosts.
    b. Configure OVS to be in failover secure mode.
    c. Drop the connection between the OVS/controller.
        i. 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

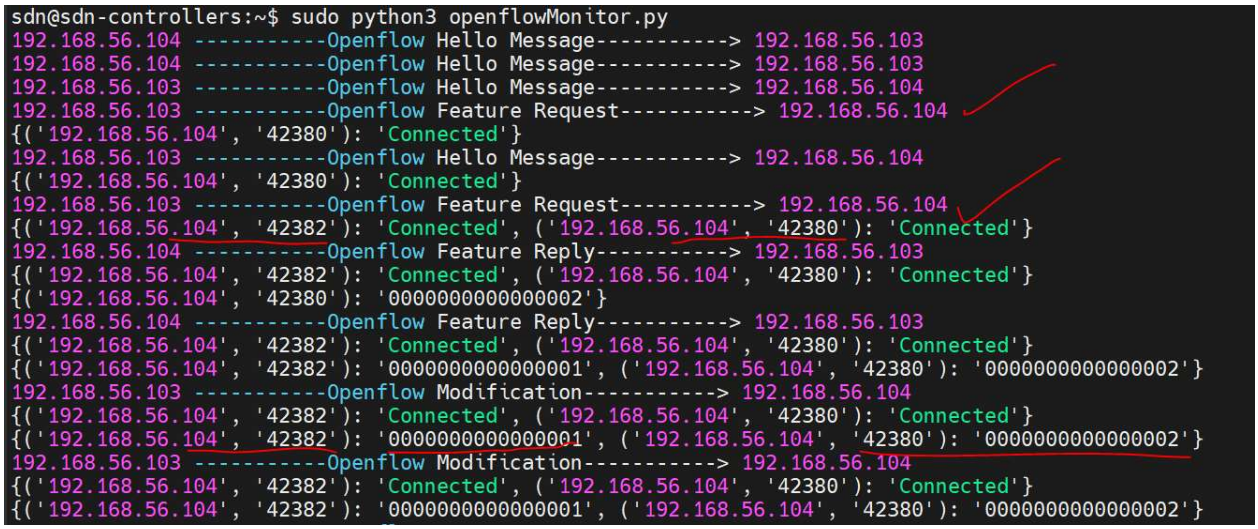
4. The script should print out the dpid of the connected switch and also save this information in a file connected.txt in JSON format.

   Format –

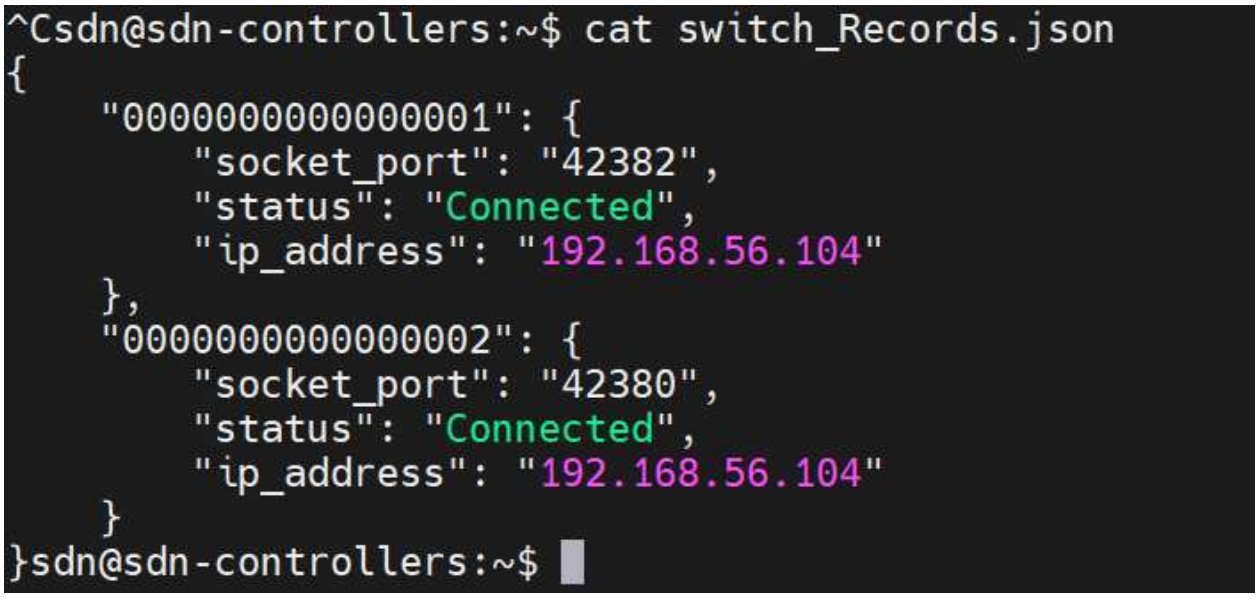
   { “<dpid>” : { “ip_address” : “<IP address of switch>”, “status” : “<connected/not connected>” } }

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

   Script output ->



   File generated -



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', '57558'): '000096fe11ad3548', ('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', '57558'): '000096fe11ad3548', ('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

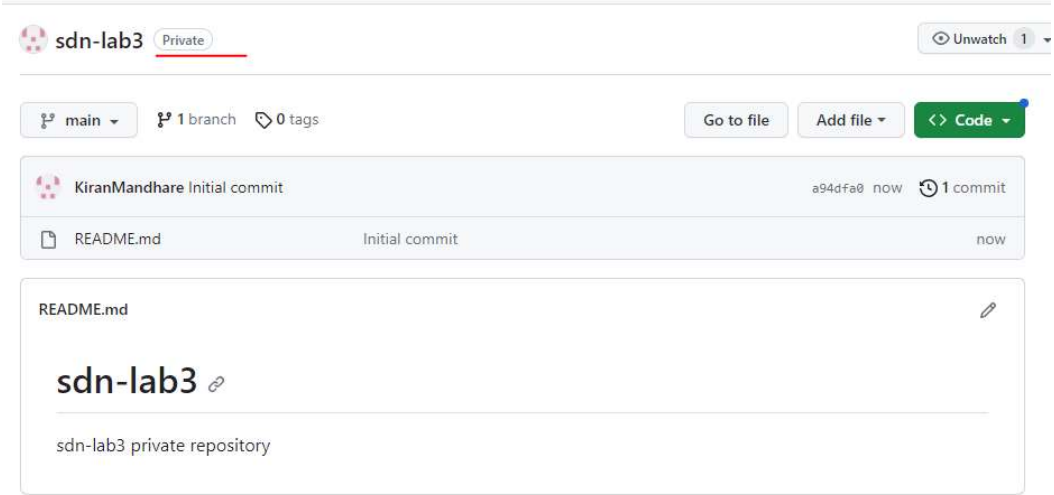
generate alerts, or take some troubleshooting actions allowing to write self healing code or monitoring piece of code. This txt file can also be used to create a dashboard and can be used to understand the current health of different switches in the network.

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**]



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 repsoitory

```
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

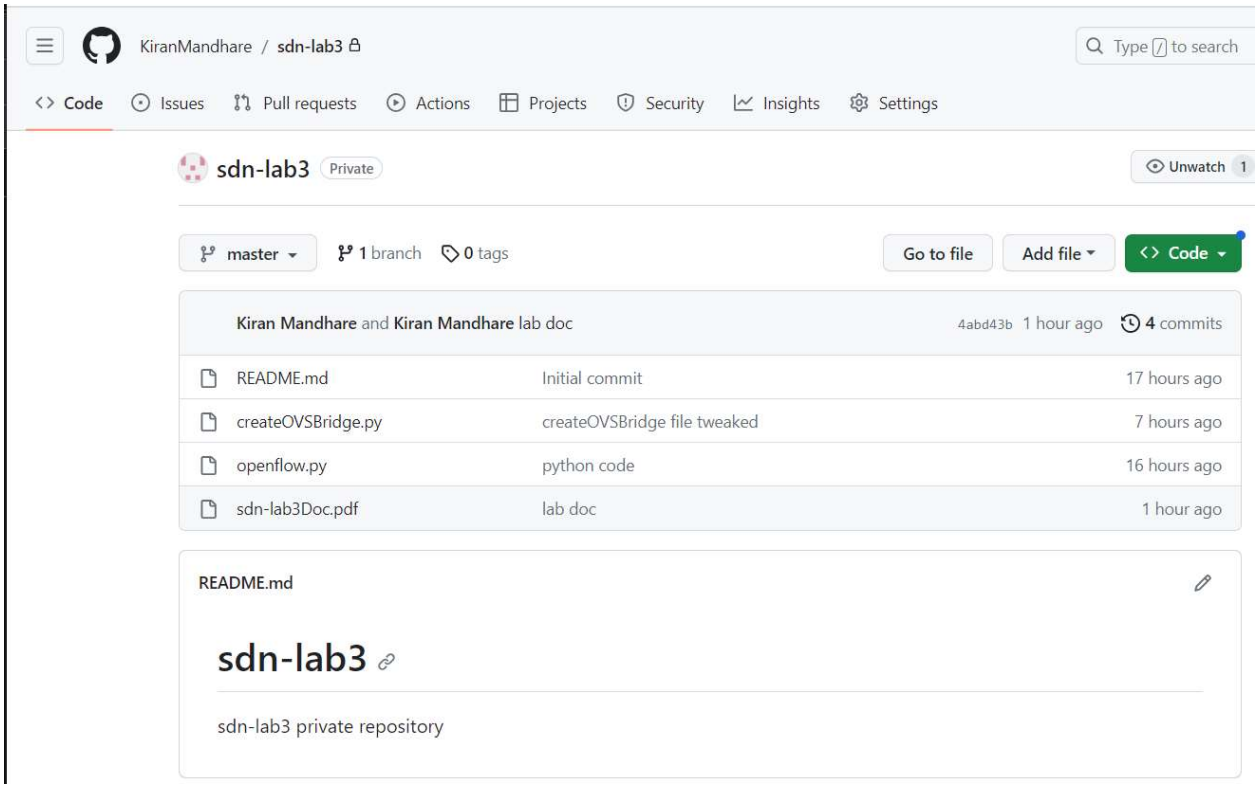
```
kiran@DESKTOP-9CMJKPK MINGW64 /d/sdn-lab3 (master)
$ git branch --set-upstream-to=origin/master master
branch 'master' set up to track 'origin/master'.
```

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
```



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