FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

COMMUNICATIONS NETWORKS II (521377S)

Mininet Exercise

Asanka Amarasinghe, 2307144

Athmajan Vivekananthan, 2305573

Farouk Aouaneche, 2307643

March 2024

CONTENTS

CONTENTS

1 BASICS OF MININET

topo-2sw-2host.py 4 fixedPerf.py 6 task1.py 5 ●

Users > athmajanvivekananthan > Downloads > task1.py > task1ControllerNet

```python
#!/usr/bin/env python
from mininet.net import Mininet
from mininet.node import OVSController, OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.util import dumpNodeConnections

def task1ControllerNet():
    "Create a network from scratch with 2  switches, 4 hosts, and 1 controller"

    net = Mininet( controller=OVSController, switch=OVSSwitch,
                   waitConnected=True )

    info( "*** Creating controllers\n" )
    c1 = net.addController( 'c1', port=1234 )

    info( "*** Creating switches\n" )
    s1 = net.addSwitch( 's1' )
    s2 = net.addSwitch( 's2' )

    info( "*** Creating hosts\n" )
    h1 = net.addHost( 'h1' )
    h2 = net.addHost( 'h2' )
    h3 = net.addHost( 'h3' )
    h4 = net.addHost( 'h4' )


    info( "*** Creating links\n" )
    net.addLink( h1, s1 )
    net.addLink( h2, s1 )
    net.addLink( h3, s2 )
    net.addLink( h4, s2 )
    net.addLink( s1, s2 )

    info( "*** Starting network\n" )

    net.build()
    c1.start()
    s1.start( [ c1 ] )
    s2.start( [ c1 ] )

    info( "Dumping host connections\n" )
    dumpNodeConnections(net.hosts)
    info( "*** Testing network\n" )
    net.pingAll()
    info( "*** Stopping network\n" )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )  # for CLI output
    task1ControllerNet()
```

Figure 1.1. Python Script

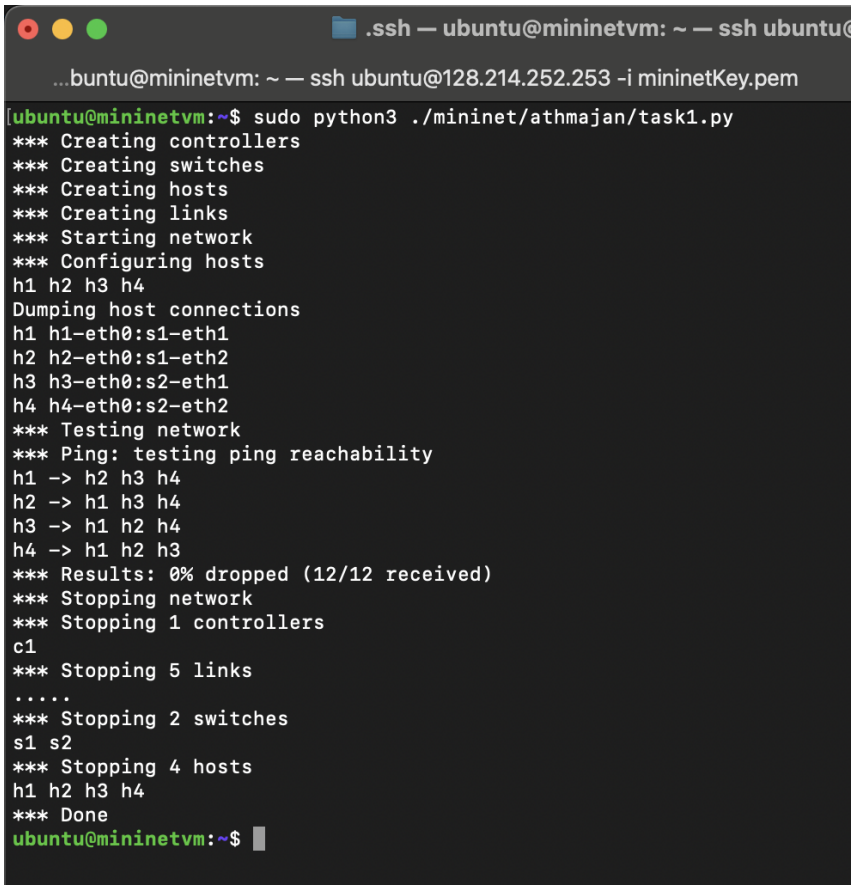

Figure 1.2. Script output

1.1 Question 1

What would be the outcome of the reachability test if there was no link between s1 and s2?

$$\begin{array}{llll} h1 \rightarrow h2 & X & X \\ h2 \rightarrow h1 & X & X \\ h3 \rightarrow X & X & h4 \\ h4 \rightarrow X & X & h3 \end{array}$$

When the link between s1 and s2 was removed h1 and h2 were not reachable from h3 and h4. Similarly h3 and h4 were not reachable from h1 and h2. However h1 was reachable from h2 and vice versa. Likewise h3 was reachable from h4 and vice versa. This is because the connection within each of the switches were available, so the hosts connected within a switch were reachable to each other. Hosts connected between switches which do not have a link were not reachable to each other.

1.2 Question 2

Would there be any difference in the outcome if you replaced the OpenFlow switches with legacy switches? Why? Why not?

In this specific example in the task, there will be no difference in the outcome if we replaced OpenFlow switches with legacy switches. Because the open source controllers are supposed to have legacy device support making it easy to add and configure traditional devices and services. However in other applications where a large scalability and huge flexibility of the network is required would provide different results.

topo-2sw-2host.py 4 fixedPerf.py 6 task1.py 5

Users > athmajanvivekananthan > Downloads > task1.py > task1ControllerNet

```python
#!/usr/bin/env python
from mininet.net import Mininet
from mininet.node import OVSController, OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.util import dumpNodeConnections

def task1ControllerNet():
    "Create a network from scratch with 2  switches, 4 hosts, and 1 controller"

    net = Mininet( controller=OVSController, switch=OVSSwitch,
                   waitConnected=True )

    info( "*** Creating controllers\n" )
    c1 = net.addController( 'c1', port=1234 )

    info( "*** Creating switches\n" )
    s1 = net.addSwitch( 's1' )
    s2 = net.addSwitch( 's2' )

    info( "*** Creating hosts\n" )
    h1 = net.addHost( 'h1' )
    h2 = net.addHost( 'h2',cpu=.5)
    h3 = net.addHost( 'h3',cpu=.5)
    h4 = net.addHost( 'h4' )


    info( "*** Creating links\n" )
    net.addLink( h1, s1 )
    net.addLink( h2, s1, bw=10, delay='5ms', max_queue_size=1000, loss=2)
    net.addLink( h3, s2, bw=10, delay='5ms', max_queue_size=1000, loss=2)
    net.addLink( h4, s2 )
    net.addLink( s1, s2 )

    info( "*** Starting network\n" )

    net.build()
    c1.start()
    s1.start( [ c1 ] )
    s2.start( [ c1 ] )

    info( "Dumping host connections\n" )
    dumpNodeConnections(net.hosts)
    info( "*** Testing network\n" )
    net.pingAll()

    info( "Testing connection speeds between h1 and h2" )
    net.iperf((h1, h2))

    info( "Testing connection speeds between h1 and h3" )
    net.iperf((h1, h3))

    info( "Testing connection speeds between h2 and h3" )
    net.iperf((h2, h3))

    info( "Testing connection speeds between h1 and h4" )
    net.iperf((h1, h4))

    info( "*** Stopping network\n" )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )  # for CLI output
    task1ControllerNet()
```

Figure 1.3. Modified Python Script

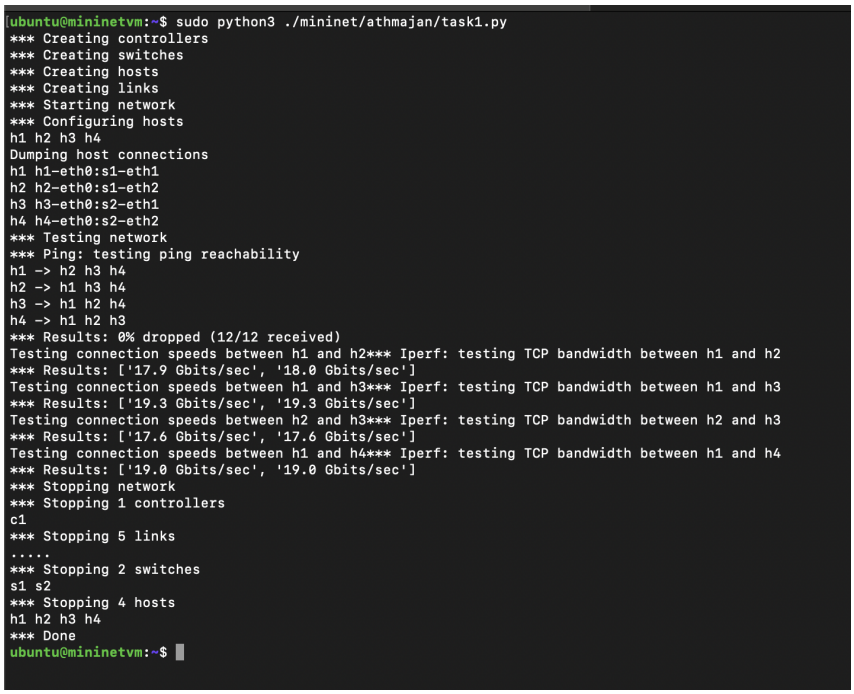

Figure 1.4. Modified Script Output

1.3 Question 3

Explain briefly why you got the outcome you got from the speed tests by comparing the corresponding throughputs.

The answer to this question has been analyzed in three sections. Each cases are presented with two values for forward throughput and backward throughput separated by pipe.

1. **Case I**
   $s1$ and $s2$ were connected. There are no link constraints.

2. **Case II**
   $s1$ and $s2$ were disconnected. Given link constraints were placed for $h2$ and $h3$.

3. **Case III**
   $s1$ and $s2$ were connected. Given link constraints were placed for $h2$ and $h3$.

Table 1.1. Testing connection speeds between hosts for different cases

Connection	Case I BW (Gbps)	Case II BW(Gbps)	Case III BW(Gbps)
h1 $\longleftrightarrow$ h2	18.8|18.8	20.7|20.7	18.1|18.1
h1 $\longleftrightarrow$ h3	18.1|18.2	Cannot reach	18.3|18.4
h2 $\longleftrightarrow$ h3	19.5|19.5	Cannot reach	18.5|18.5
h1 $\longleftrightarrow$ h4	19.3|19.4	Cannot reach	18.9|19.0

As it can be seen in the results presented in Table 1.1, when the link between $s1$ and $s2$ was disconnected, the isolated hosts were unreachable. Therefore throughput measurements were not not available for them.

However when the link between $s1$ and $s2$ was established and the comparing the two cases of with and without the performance constraints on $h2$ and $h3$, it can be seen the likely observation of lower throughputs between $h1$ and $h2$, $h1$ and $h3$, and $h2$ and $h3$. The specified link parameters (bandwidth, delay, loss, and maximum queue size) influence the network performance and can lead to lower throughputs compared to the unconstrained link.

1.4 Question 4

What difference would it make in the speed test if the links between all 4 hosts and their corresponding switches were defined using the same parameters?

A new comparison has been presented in Table 1.2 on top of the same cases (Case I and Case III) from above, including a new case (Case IV) which has the same constraints applied for all links between hosts and their corresponding switches.

Table 1.2. Testing connection speeds with constraints to all links

Connection	**Case I BW (Gbps)**	**Case III BW(Gbps)**	**Case IV BW(Gbps)**
h1 $\longleftrightarrow$ h2	18.8|18.8	18.1|18.1	18.8|18.9
h1 $\longleftrightarrow$ h3	18.1|18.2	18.3|18.4	18.8|18.8
h2 $\longleftrightarrow$ h3	19.5|19.5	18.5|18.5	18.5|18.6
h1 $\longleftrightarrow$ h4	19.3|19.4	18.9|19.0	19.6|19.6

When all linked were constrained with the same parameters (Case IV), the behaviour of the network throughput is almost similar to the totally unconstrained case (Case I) even though slight reduction in throughput is observed.

2 INTEGRATING A REMOTE CONTROLLER

sers > athmajanvivekananthan > Downloads > task1.py > task1ControllerNet

```python
from mininet.net import Mininet
from mininet.node import OVSController, OVSSwitch, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink
from mininet.util import dumpNodeConnections

def task1ControllerNet():
    "Create a network from scratch with 2  switches, 4 hosts, and 1 controller"
    c2 = RemoteController( 'c2', ip='127.0.0.1', port=6633 )

    net = Mininet( controller=c2, switch=OVSSwitch,
                   waitConnected=False,link=TCLink)

    info( "*** Creating controllers\n" )
    #c1 = net.addController( 'c1', port=6633 )

    info( "*** Creating switches\n" )
    s1 = net.addSwitch( 's1' )
    s2 = net.addSwitch( 's2' )

    info( "*** Creating hosts\n" )
    h1 = net.addHost( 'h1')
    h2 = net.addHost( 'h2')
    h3 = net.addHost( 'h3')
    h4 = net.addHost( 'h4')


    info( "*** Creating links\n" )
    net.addLink( h1, s1)
    net.addLink( h2, s1)
    net.addLink( h3, s2)
    net.addLink( h4, s2)
    net.addLink( s1, s2 )
    info( "*** Starting network\n" )

    net.build()
    c2.start()
    s1.start( [ c2 ] )
    s2.start( [ c2 ] )

    info( "Dumping host connections\n" )
    dumpNodeConnections(net.hosts)
    info( "*** Testing network\n" )
    net.pingAll()

    CLI(net)

    info( "Displaying flow table\n" )
    for switch in net.switches:
        print(switch.dpctl('dump-flows'))


    info( "*** Stopping network\n" )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )  # for CLI output
    task1ControllerNet()
```

Figure 2.1. Python code showing the integration of POX controller

```
ubuntu@mininetvm:~/mininet/custom/pox$ ./pox.py log.level --DEBUG forwarding.hub
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
INFO:forwarding.hub:Hub running.
DEBUG:core:POX 0.1.0 (betta) going up...
DEBUG:core:Running on CPython (2.7.18/Jul 1 2022 12:27:04)
DEBUG:core:Platform is Linux-5.4.0-171-generic-x86_64-with-Ubuntu-20.04-focal
INFO:core:POX 0.1.0 (betta) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
```

Figure 2.2. POX controller Running as Hub

```
ubuntu@mininetvm:~/mininet/athmajan$ sudo python3 task1.py
*** Creating controllers
*** Creating switches
*** Creating hosts
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth1
h4 h4-eth0:s2-eth2
*** Testing network
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
*** Starting CLI:
mininet> exit
Displaying flow table
NXST_AGGREGATE reply (xid=0x4): packet_count=62 byte_count=5028 flow_count=1

NXST_AGGREGATE reply (xid=0x4): packet_count=57 byte_count=4706 flow_count=1

 cookie=0x0, duration=4.069s, table=0, n_packets=62, n_bytes=5028, actions=FLOOD

 cookie=0x0, duration=4.061s, table=0, n_packets=57, n_bytes=4706, actions=FLOOD

*** Stopping network
*** Stopping 0 controllers

*** Stopping 5 links
.....
*** Stopping 2 switches
s1 s2
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
```

Figure 2.3. Snapshot of the code outcome for POX as a Hub

```
ubuntu@mininetvm:~/mininet/custom/pox$ ./pox.py log.level --DEBUG forwarding.l2_learning
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.1.0 (betta) going up...
DEBUG:core:Running on CPython (2.7.18/Jul 1 2022 12:27:04)
DEBUG:core:Platform is Linux-5.4.0-171-generic-x86_64-with-Ubuntu-20.04-focal
INFO:core:POX 0.1.0 (betta) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 2]
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
```

Figure 2.4. POX controller Running as Learning Switch

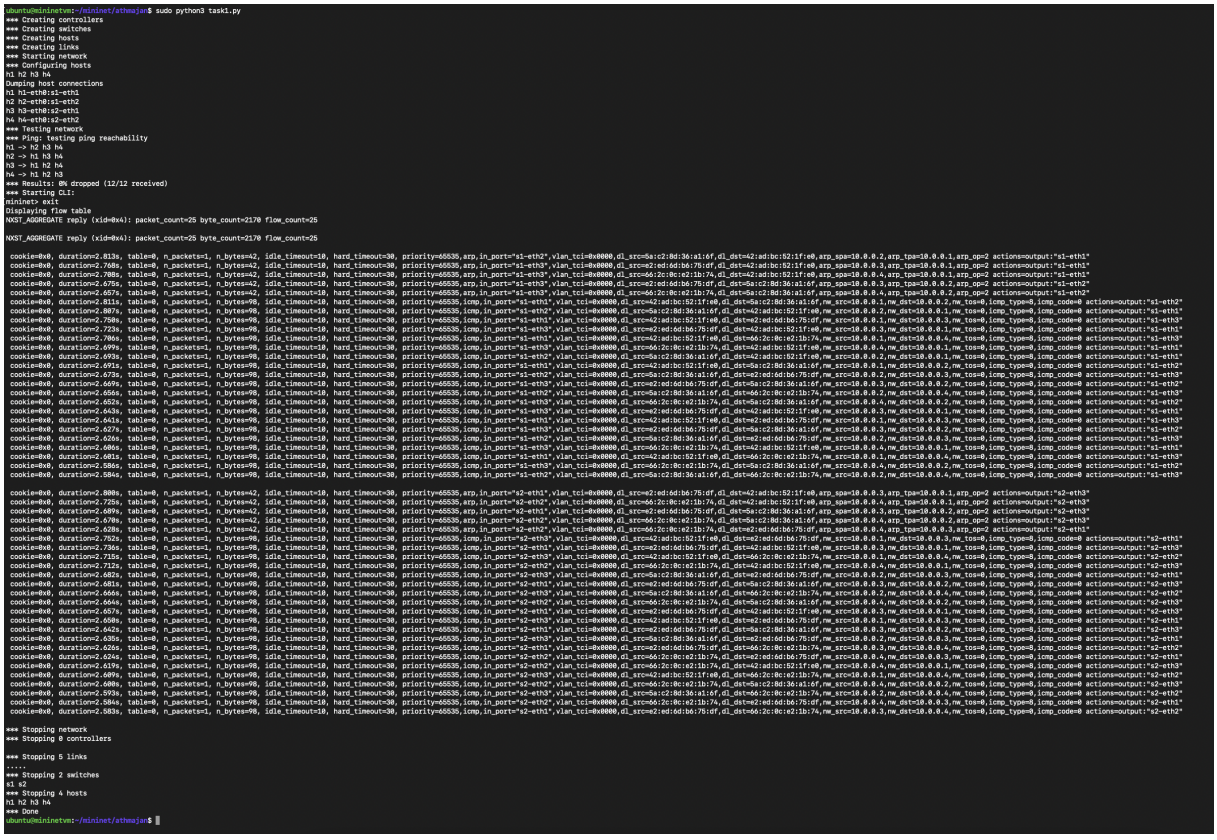

Figure 2.5. Snapshot of the code outcome for POX as a learning switch

2.1 Question 5

Compare the results of dpctl before and after converting the POX controller to a learning switch. What is the difference between the two and why?

When the remote controller is working as a hub it merely forwards the all incoming packets to all port except the one that the packets were received at. This can be seen in the dpctl entry on the flow where the action states ”FLOOD”.

Sample flow Entry for POX controller acting as a hub

```
cookie=0x0, duration=2.417s, table=0, n_packets=56, n_bytes=4536,
actions=FLOOD
```

When acting as a hub, the controller does not learn any MAC addresses nor maintains any forwarding tables. However when the controller is acting as a layer 2 learning switch, it maintains a forwarding table that has information about the MAC addresses and ports.

Sample flow Entry for POX controller acting as a learning switch

```
cookie=0x0, duration=2.609s, table=0, n_packets=1,
n_bytes=98, idle_timeout=10, hard_timeout=30,
priority=65535,icmp,in_port="s2-eth3",vlan_tci=0x0000,
dl_src=42:ad:bc:52:1f:e0,dl_dst=66:2c:0c:e2:1b:74,nw_src=10.0.0.1,
nw_dst=10.0.0.4,nw_tos=0,icmp_type=0,icmp_code=0
actions=output:"s2-eth2"
```

Upon arrival of each packet, the controller learns the MAC address and ports of the incoming feed including the output port information if available in the packet. The flow entry has the output port detail in the action parameter. Therefore similar packet forwarding can be done without having to involve the controller.

3 CREATING CUSTOM TOPOLOGY USING MINIEDIT

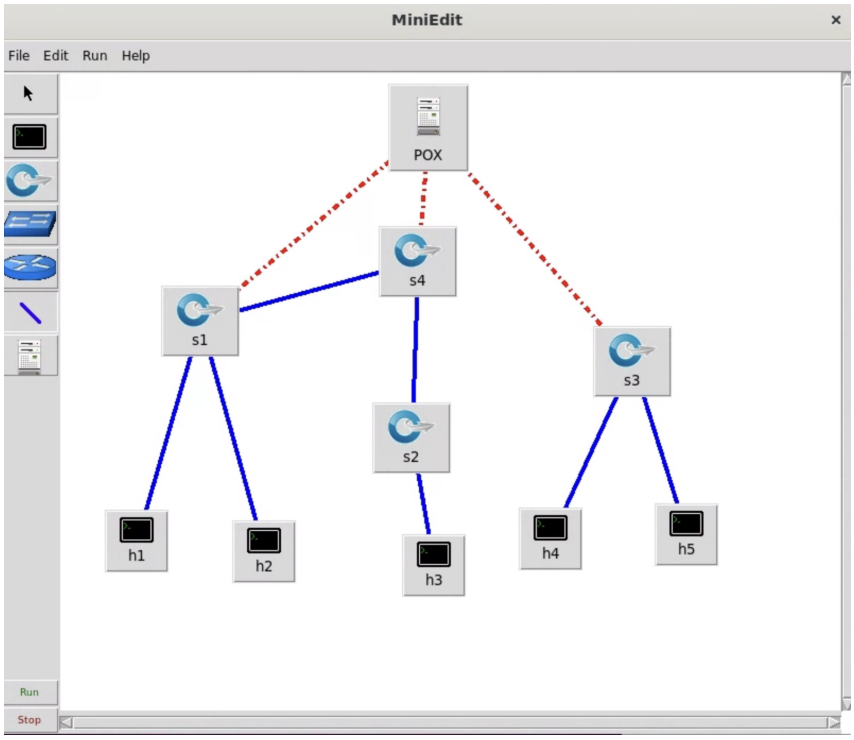

Figure 3.1. Custom topology for exercise 3 - 1

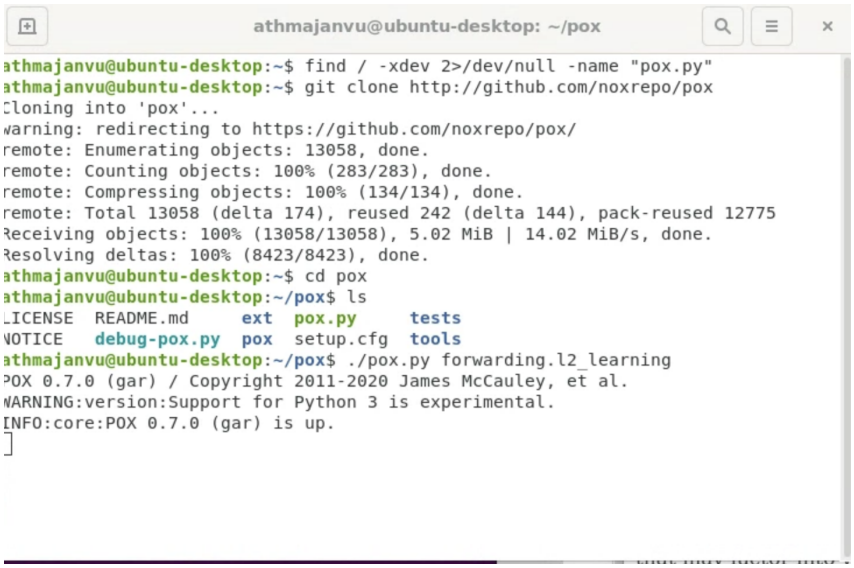

Figure 3.2. POX Controller Running

3.1 Question 6

Now run your network and observe the POX terminal. Briefly explain what you observed on the terminal, what elements can you identify and what do they imply?

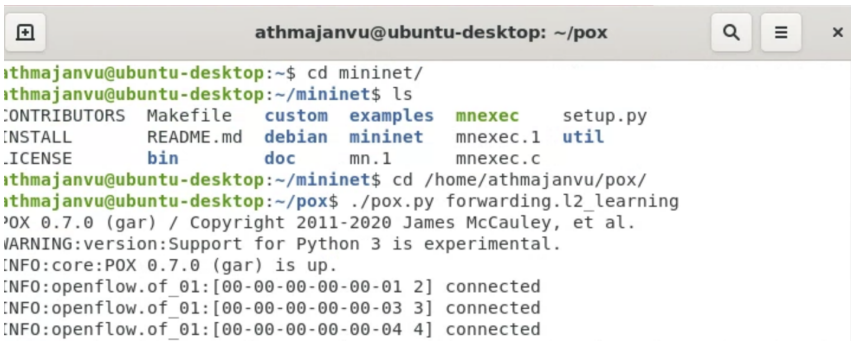

Figure 3.3. POX Controller Output on Terminal

This controller output shows the confirmation on how the connection of OpenFlow switches are complete. The output shows the MAC addresses of the successfully connected devices. In this case those are S1, S3 and S4 respectively as shown in the order in Figure 3.3. This also implies that the controller has no connectivity to S2 which impacted in the ping outputs as described in the following section.

3.2 Question 7

Are you able to reach all nodes on the network? If not why? What modifications do you think are necessary to be able to reach all the hosts? If yes, what modifications did you have to make to reach all the hosts?

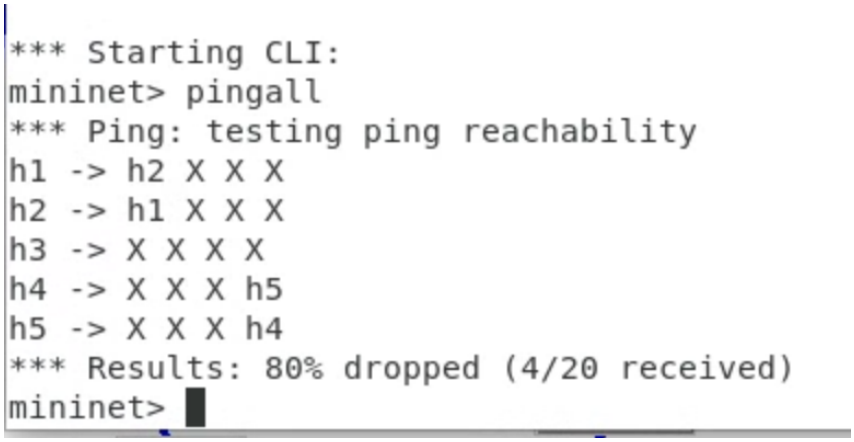

Figure 3.4. Ping output with initial topology as showsn in Figure 3.1

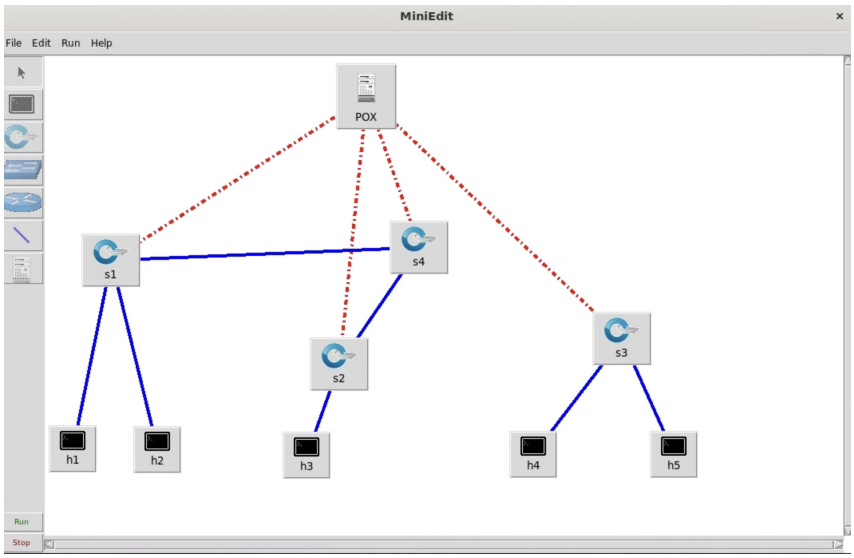

Figure 3.5. Modified Topology to enable full network reachability

```
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 X X
h2 -> h1 h3 X X
h3 -> h1 h2 X X
h4 -> X X X h5
h5 -> X X X h4
*** Results: 60% dropped (8/20 received)
mininet>
```

Figure 3.6. Pingall output after modification to the topology as shown in Figure 3.5

At this point whole network reachability is not achieved. As seen in Figure 3.4, h1 can only reach h2 and vice versa. h3 cannot reach any other nodes. h4 can only reach h5 and vice versa. However based on the topology reachability from h1 and h2 to h3 and vice versa should be available by the looks of it. But since S2 is not connected to the controller, the packets directed to h3 from h1 and h2 and vice versa do not reach their destinations.

Therefore by connecting S2 with the controller allows the packets directed towards h3 from h1 and h2 and vice versa. The modified topology can be seen in Figure 3.5. After making the modifications on the topology by connected S2 to the controller the ping output results can be seen as expected in Figure 3.6.

3.3 Flow Dumps of S4

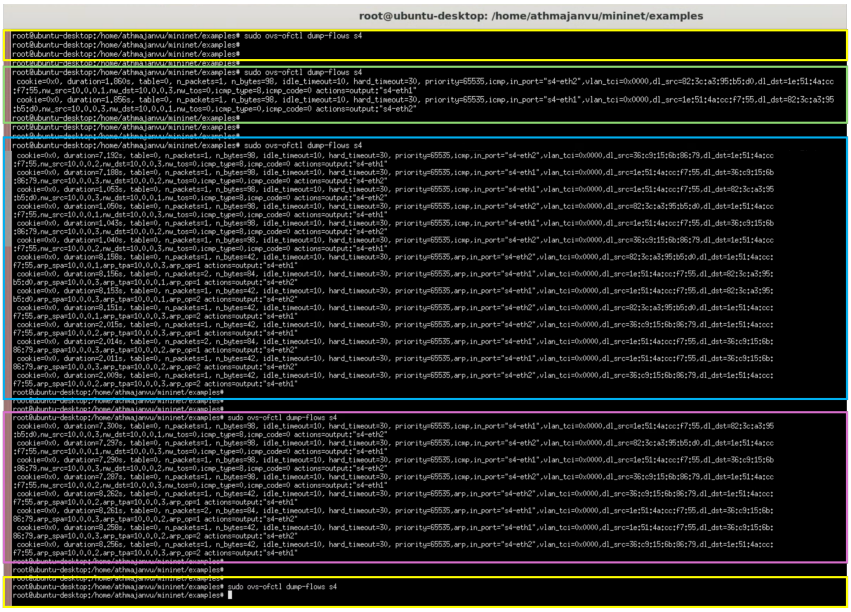

Figure 3.7. Entries in Flow table on s4

Figure 3.7 shows the entries in the S4 flow table. It explains how entries changed using a colour code. As listed in Table the changes in the flow table while a ping test was being done and how the flow table was released of all the entries after the test is done can be clearly seen.

Table 3.1. Explanation of Colour Code used in Figure 3.7

Colour Scenario	Flow Entries
Initial Yellow	None
Green	Flow entries when h1 was pinging other nodes
Cyan	Flow entries when h2 was pinging other nodes
Purple	Flow entries when h3 was pinging other nodes
Final Yellow	Flow entries when h4 and h5 were pinging other nodes and after the test

3.4 Question 8

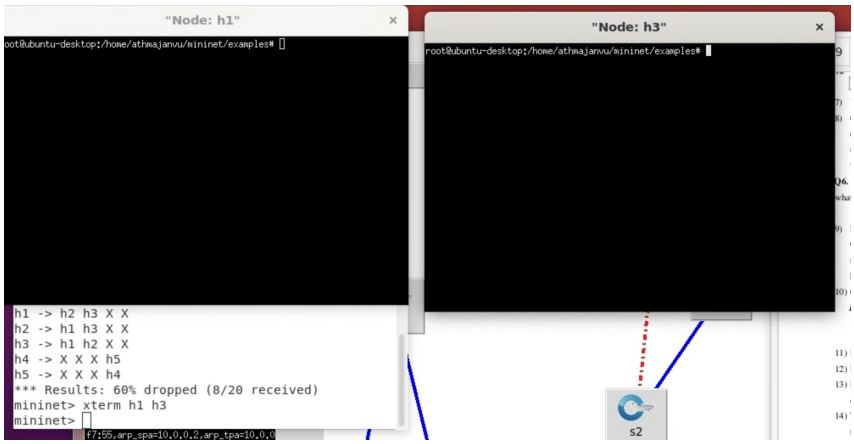

Figure 3.8. Xterm terminal for Nodes h1 and h3

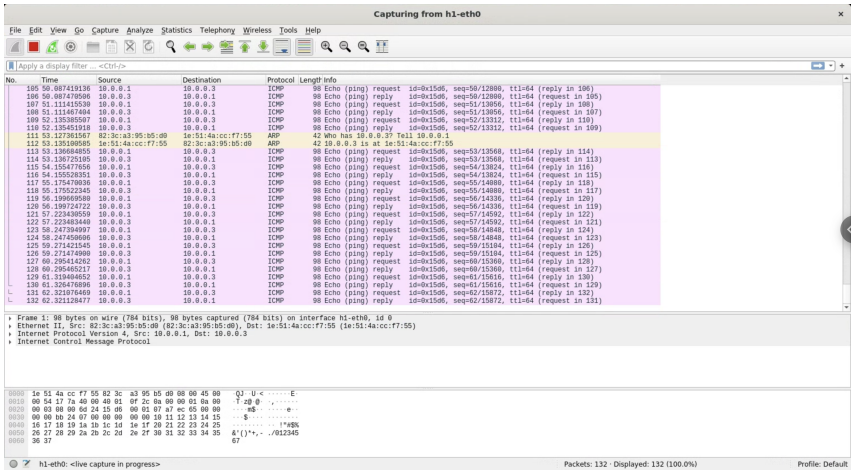

Figure 3.9. Wireshark Output of Node h1

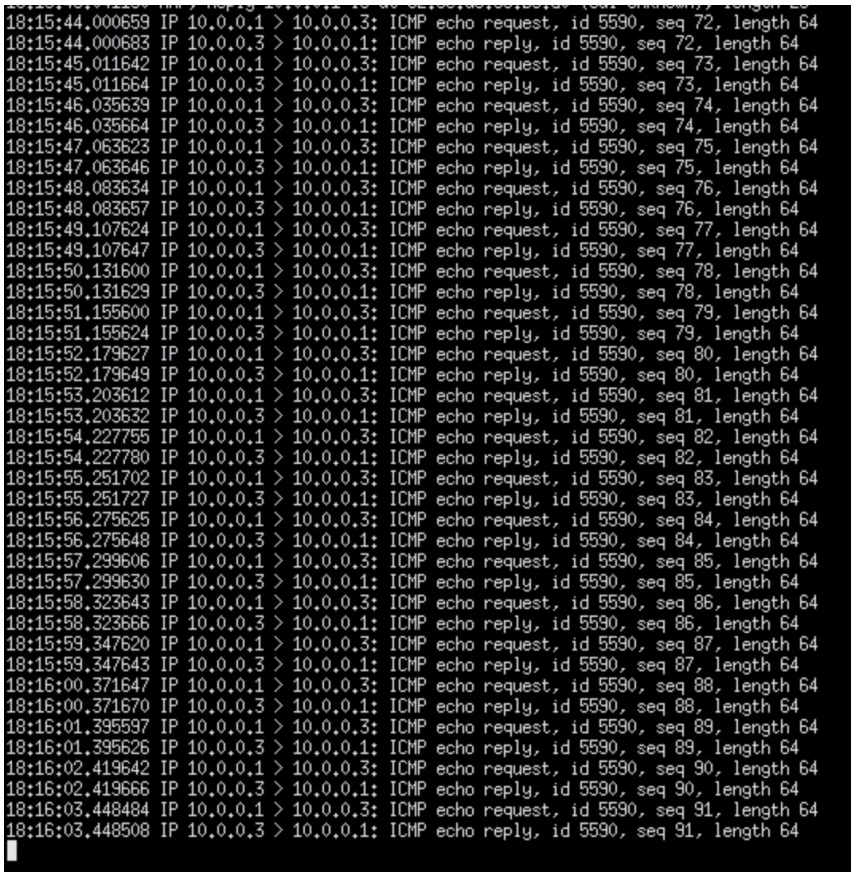

Figure 3.10. TCPDump Output of Node h3

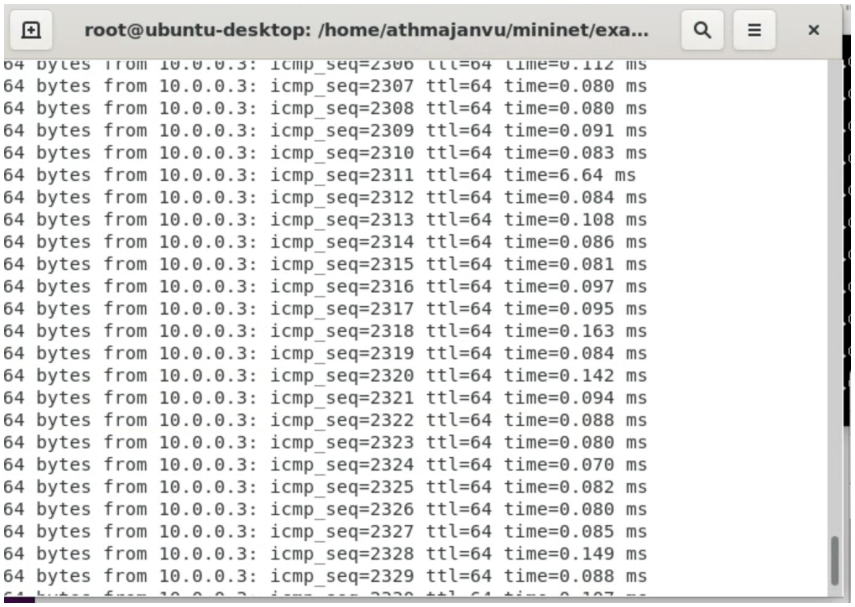

Figure 3.11. Miniedit Console Outputs

Figure 3.11 shows the Miniedit console output for the command $h1 ping h3$. It shows the ping success rate at the end. Even though it is not evident in the Figure 3.11 the console output showed the initial time for the test which was higher and later on as time progresses the time for the tests were gradually reduced meaning the optimization of the path by the controller.

Figure 3.10 shows the TCPdump output on xterm terminal of Node h3. The TCPdump is a holistic view of the Internet Control Message Protocol packets and other packets being sent in the network. Commands can be modified to capture and save the captured file to be analyzed later.

Figure 3.9 shows the Wireshark output of Node h1 when $h1 ping h3$ was going on. Since the controller has the prior update on the MAC addresses, the initial broadcast message to find MAC addresses of the target are not seen in the Wirehark frames. However in addition to the packets being sent via Internet Control Message Protocol, periodically Address Resolution Protocol requests for the MAC addresses of the nodes and their IP addresses. In addition to this a Wireshark entry is a frame that contains frame size, source address, destination address, data field. In the bottom window, hex and ASCII domain of the selected packet.

4 ADDING FLOW ENTRIES TO THE FLOW TABLE

4.1 Question 9

What actions will Mininet take based on this command?

1. Create 3 virtual hosts each with separate IP addresses
2. Created a single (Open vSwitch) switch in the kernel with 3 ports.
3. Connected the three virtual hosts to the switch with a virtual ethernet cable.
4. Set a unique MAC addresses to each host.
5. Configure the OpenFlow switch to connect to the remote controller.

4.2 Question 10

Open the switch terminal and run a simple command to reveal the current flow entries. What do you see in the flow entries? Why ?

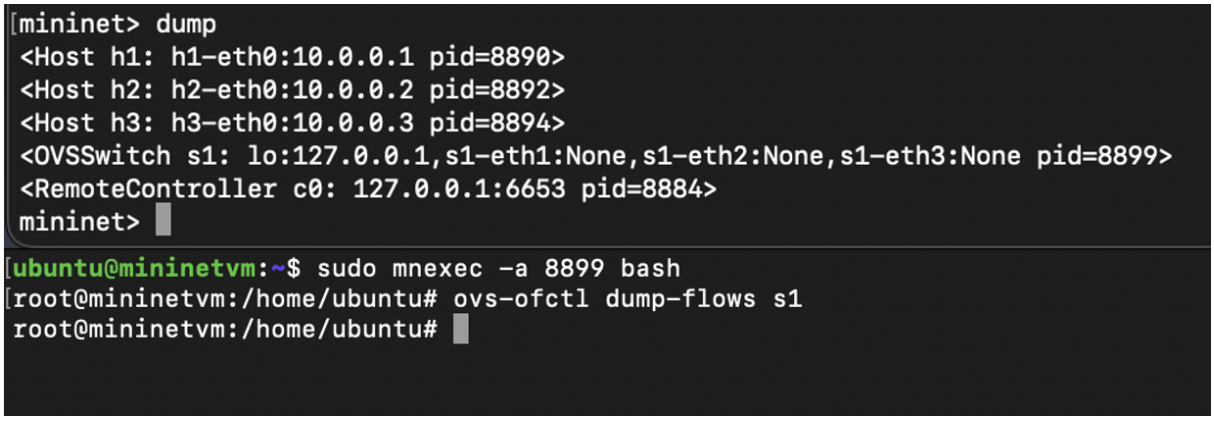

Figure 4.1. CLI output for Question 10 - Flow Entries

No flow entries were available to be seen. This is because the controller which is remote and has no way of actively controlling the network. Therefore the switch is merely forwarding the packets without maintaining any flow tables.

4.3 Question 11

Expected outcome when running $ovs - ofctldump - flowss1$ is similar to question 10 to see the flow entries of s1. But the result is blank flow entries again similar to question 10. The reason behind this is that the remote controller has no active way of writing flow entries to the switch.

4.4 Question 12

```
mininet> h1 ping -c4 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3052ms
pipe 4
mininet>
```

Figure 4.2. Ping results from h1 to h2 for Question 12

Since there are no flow entries defined in the switch the packets received from h1 directed towards h2 are dropped. Therefore we do not see a connectivity from h1 to h2.

4.5 Question 13

- Letting all nodes flood in hub mode

```
sh ovs-ofctl add-flow s1 dl_type=0x806,nw_proto=1,actions=flood
```

- Enable exchange from h1 to h2

```
sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,
dl_dst=00:00:00:00:00:02,actions=output:2
```

- Enable exchange from h1 to h2

```
sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,
dl_dst=00:00:00:00:00:03,actions=output:3
```

- Enable exchange from h3 to h1

```
sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:03,
dl_dst=00:00:00:00:00:01,actions=output:1
```

- Enable exchange from h3 to h2

```
sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:03,
dl_dst=00:00:00:00:00:02,actions=output:2
```

- Enable exchange from h2 to h3

```
sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:02,
dl_dst=00:00:00:00:00:03,actions=output:3
```

- Enable exchange from h2 to h1

```
sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:02,
dl_dst=00:00:00:00:00:01,actions=output:1
```

4.6 Question 14

```
root@mininetvm:/home/ubuntu# ovs-ofctl dump-flows s1
 cookie=0x0, duration=1051.784s, table=0, n_packets=68, n_bytes=5656, actions=FLOOD
 cookie=0x0, duration=917.688s, table=0, n_packets=0, n_bytes=0, dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=884.823s, table=0, n_packets=0, n_bytes=0, dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=881.775s, table=0, n_packets=0, n_bytes=0, dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=877.884s, table=0, n_packets=0, n_bytes=0, dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=874.750s, table=0, n_packets=0, n_bytes=0, dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=871.139s, table=0, n_packets=0, n_bytes=0, dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
root@mininetvm:/home/ubuntu#
```

Figure 4.3. Flow entries after enabling exchange between nodes

Now since the packet exchange has been enabled between the nodes, the switch has the updated flow table with the correct flow entries as manually specified.

4.7 Question 15

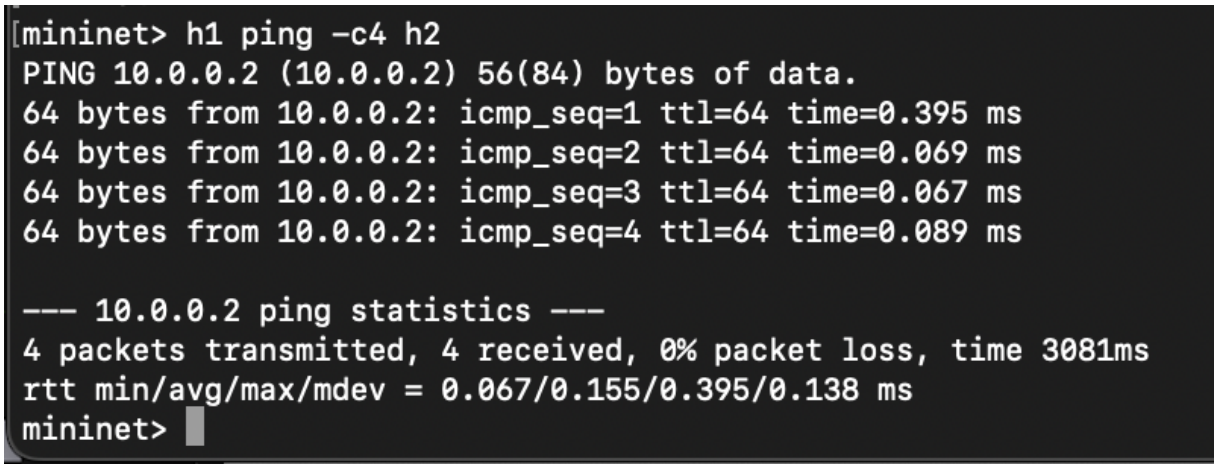

Figure 4.4. Ping output after flow entries

Now h1 is able to ping h2 since the switch knows to forward the received packet appropriately without dropping like it was before in Question 12 even without the controller present.