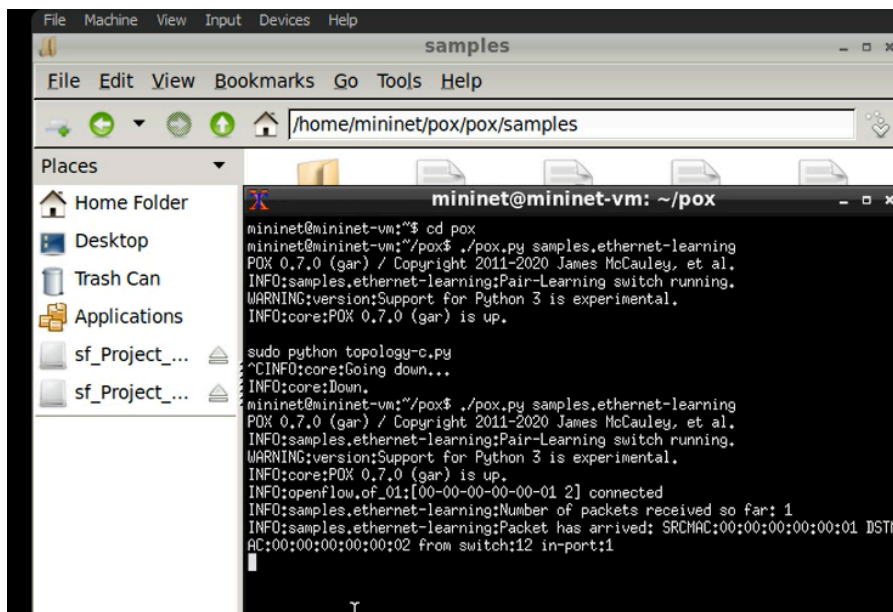
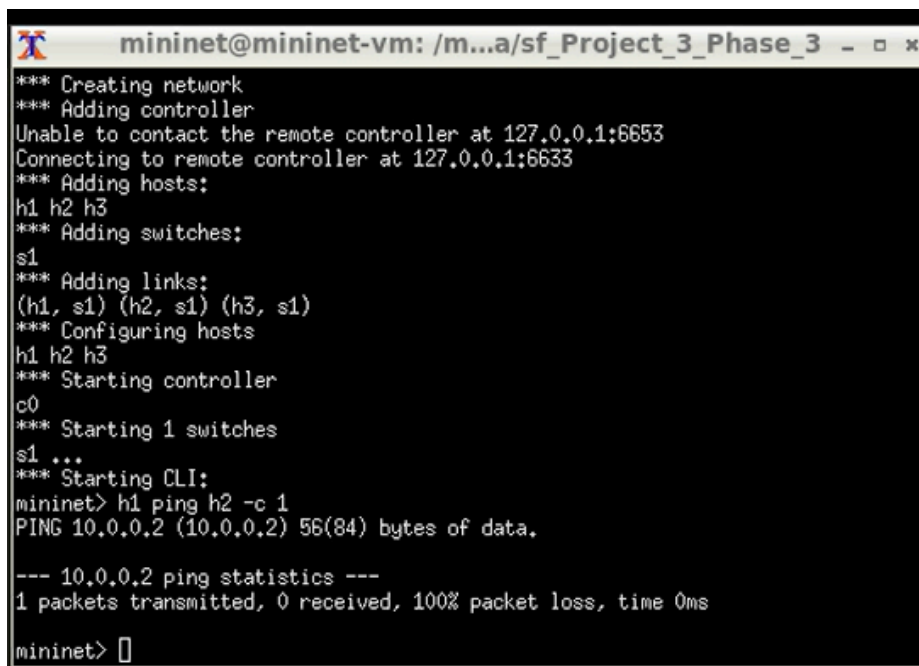


Implementation in Step 2.3:

1. Unedited ethernet-learning.py moved to /home/mininet/pox/pox/samples



2. mininet@mininet-vm: ~/media/sf\_Project\_3\_Phase\_3\$ sudo python topology-c.py



3. Mininet terminal:
  - a. mininet> h1 ping h2 -c 1
    - i. Yes, the ping was successful.
  - b. mininet> dpctl dump-flows
    - i. No rules have been installed yet.

- c. The most likely causes are that the hosts are properly configured and connected within the network, which allows ICMP traffic to flow between them. Additionally, since the network is managed by an SDN controller, there may be delays in pushing flow rules, or the controller may have not yet instructed the switch to install specific rules.

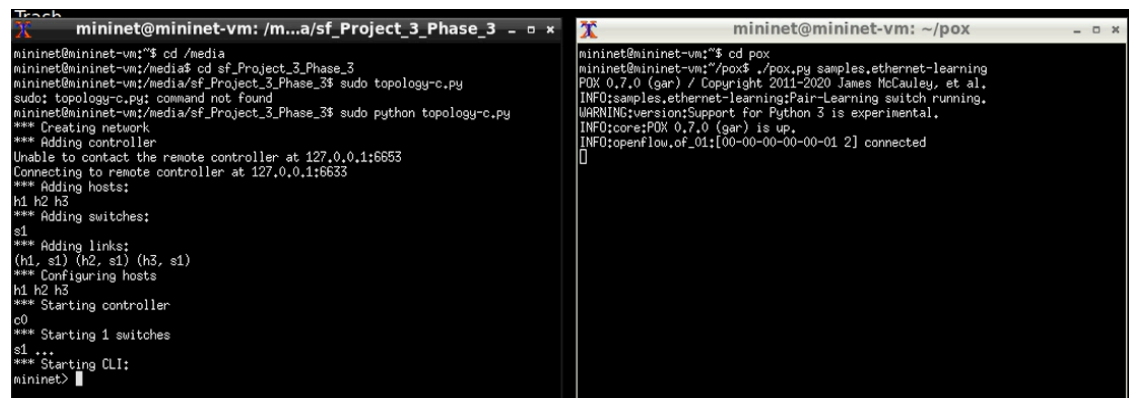
```
mininet> h1 ping h2 -c 1
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet> dpctl dump-flows
*** s1 -----
mininet>
```

- d.
- e. mininet> quit and Ctrl+C shut down the controllers

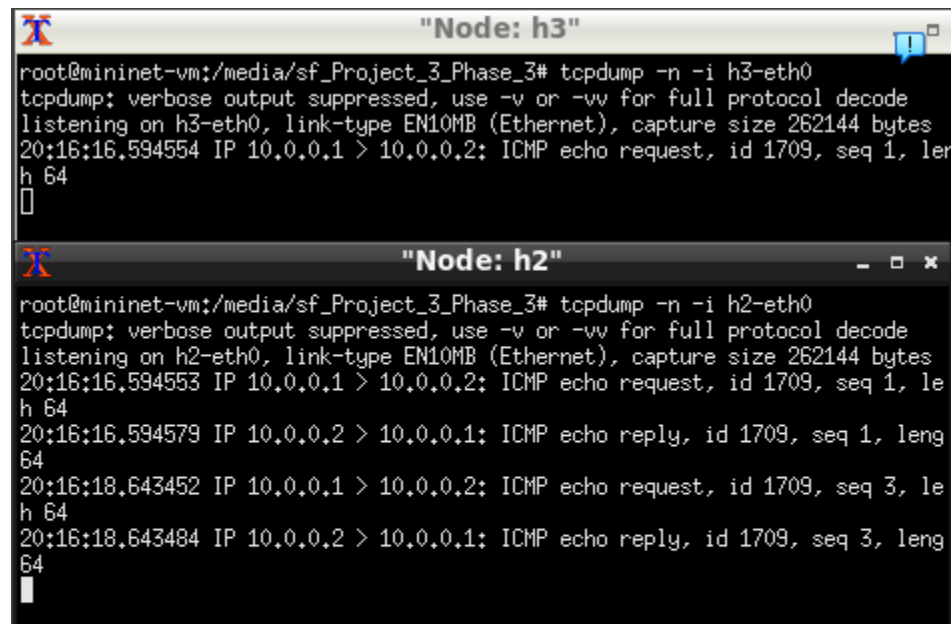
4.



```
mininet@mininet-vm: /media/sf_Project_3_Phase_3
mininet@mininet-vm:~$ cd /media
mininet@mininet-vm:/media$ cd sf_Project_3_Phase_3
mininet@mininet-vm:/media/sf_Project_3_Phase_3$ sudo topology-c.py
sudo: topology-c.py: command not found
mininet@mininet-vm:/media/sf_Project_3_Phase_3$ sudo python topology-c.py
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

```
mininet@mininet-vm: ~/pox
mininet@mininet-vm:~/pox$ ./pox.py samples,ethernet-learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
INFO:samples,ethernet-learning:Pair-Learning switch running.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
```

- a.
- b. See picture in 4a (above)
- c. All three terminals open in quick succession.



```
root@mininet-vm:/media/sf_Project_3_Phase_3# tcpdump -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:16:16.594554 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1709, seq 1, len 64
h 64
^
```

```
root@mininet-vm:/media/sf_Project_3_Phase_3# tcpdump -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:16:16.594553 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1709, seq 1, len 64
20:16:16.594579 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1709, seq 1, len 64
20:16:18.643452 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1709, seq 3, len 64
h 64
20:16:18.643484 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1709, seq 3, len 64
h 64
^
```

- d.

```
"Node: h1"
root@mininet-virtual-machine:/media/sf_Project_3_Phase_3# ping 10.0.0.2 -c 3
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.332 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 1 received, 66.6667% packet loss, time 2051ms
rtt min/avg/max/mdev = 0.332/0.332/0.332/0.000 ms
root@mininet-virtual-machine:/media/sf_Project_3_Phase_3#
```

e. H1:

```
"Node: h2"
root@mininet-virtual-machine:/media/sf_Project_3_Phase_3# tcpdump -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:16:16.594553 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1709, seq 1, len 64
20:16:16.594579 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1709, seq 1, len 64
20:16:18.643452 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1709, seq 3, len 64
20:16:18.643484 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1709, seq 3, len 64
```

H2:

```
"Node: h3"
root@mininet-virtual-machine:/media/sf_Project_3_Phase_3# tcpdump -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:16:16.594554 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1709, seq 1, len 64
```

H3:

- i. In h1, only the third packet was successfully received.  
In h2, ICMP echo replies were requested and sent for packets 1 and 3.  
In h3, an ICMP echo reply was requested for packet 1, but never sent.
  - ii. There is no difference, because only one packet was received.
- f. H3 closed. H2 exited tcpdump without issue.
- g. iperf

```

"Node: h2"
root@mininet-vmt:/media/sf_Project_3_Phase_3# tcpdump -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:16:16.594553 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1709, seq 1, le
h 64
20:16:16.594579 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1709, seq 1, leng
64
20:16:18.643452 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1709, seq 3, le
h 64
20:16:18.643484 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1709, seq 3, leng
64
^C
4 packets captured
4 packets received by filter
0 packets dropped by kernel
root@mininet-vmt:/media/sf_Project_3_Phase_3# iperf -s -p 4000
-----
Server listening on TCP port 4000
TCP window size: 85.3 KByte (default)
-----
[ 6] local 10.0.0.2 port 4000 connected with 10.0.0.1 port 47004
[ ID] Interval      Transfer    Bandwidth
[ 6] 0.0-10.0 sec  44.5 GBytes 38.1 Gbits/sec

```

i. H2:

```

"Node: h1"
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 1 received, 66.6667% packet loss, time 2051ms
rtt min/avg/max/mdev = 0.332/0.332/0.332/0.000 ms
root@mininet-vmt:/media/sf_Project_3_Phase_3# iper -c 10.0.0.2 -p 4000
Command 'iper' not found, did you mean:

  command 'iperf' from deb iperf (2.0.13+dfsg1-1build1)
  command 'piper' from deb piper (0.4-1)
  command 'ipe' from deb ipe (7.2.13-2build1)
  command 'iperl' from deb libapp-repl-perl (0.012-2)

Try: apt install <deb name>
root@mininet-vmt:/media/sf_Project_3_Phase_3# iperf -c 10.0.0.2 -p 4000
-----
Client connecting to 10.0.0.2, TCP port 4000
TCP window size: 493 KByte (default)
-----
[ 5] local 10.0.0.1 port 47004 connected with 10.0.0.2 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec  44.5 GBytes 38.2 Gbits/sec
root@mininet-vmt:/media/sf_Project_3_Phase_3#

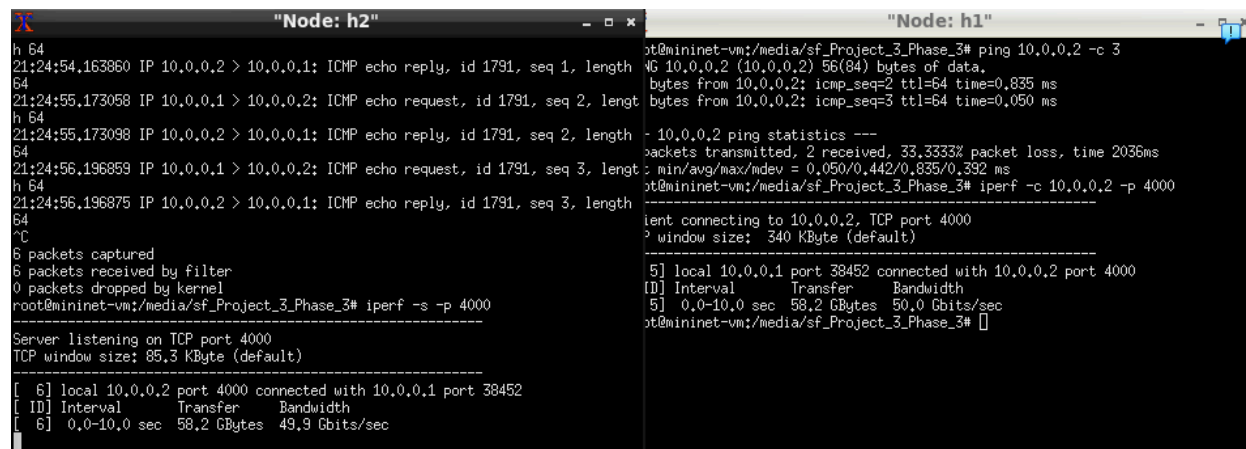
```

- ii. H1:
- iii. Yes, the iperf command worked as intended.
- iv. 77,909 packets were sent to the controller (1,067,726 - 989,817)
- v. Four rules are installed in s1.
- vi. The first two rules are from when h1 attempted to ping h2. This can be determined because of the older timestamp (duration value), as well as

the smaller value of the bytes (n\_bytes)

The second two rules are from the iPerf, which is determined by the newer timestamp (duration), as well as the much larger value in n\_bytes.

- vii. See 4g-i and 4g-ii.
  - h. Mininet shuts down with quit and the controller shuts down with Ctrl+C.
5. Step 5 code is noted clearly in ethernet-learning.py
6. Ping and iPerf
- a. iPerf worked without issue.
  - b. 68,787 packets were sent to the controller (1,408,524 - 1,339,737)
  - c. Two rules are installed in s1.
  - d. The two rules are as listed in the writeup:
    - i. dl\_src = Packet's source MAC address, dl\_dst = Packet's destination MAC address. Set the action's output port to the one that reaches the destination host.
    - ii. dl\_src = Packet's destination MAC address, dl\_dst = Packet's source MAC address. Set the action's output port to the one that reaches the source host.



```
h 64
21:24:54.163860 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1791, seq 1, length 64
21:24:55.173058 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1791, seq 2, length 64
21:24:55.173098 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1791, seq 2, length 64
21:24:56.196859 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 1791, seq 3, length 64
21:24:56.196875 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 1791, seq 3, length 64
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
root@mininet-vmt:/media/sf_Project_3_Phase_3# iperf -s -p 4000
Server listening on TCP port 4000
TCP window size: 85.3 KByte (default)

[ 6] local 10.0.0.2 port 4000 connected with 10.0.0.1 port 38452
[ ID] Interval      Transfer    Bandwidth
[ 6] 0.0-10.0 sec  58.2 GBytes 49.9 Gbits/sec

bt@mininet-vmt:/media/sf_Project_3_Phase_3# ping 10.0.0.2 -c 3
P 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.835 ms
bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.050 ms

- 10.0.0.2 ping statistics ---
packets transmitted, 2 received, 33.333% packet loss, time 2036ms
: min/avg/max/ndev = 0.050/0.442/0.835/0.392 ms
bt@mininet-vmt:/media/sf_Project_3_Phase_3# iperf -c 10.0.0.2 -p 4000
tent connecting to 10.0.0.2, TCP port 4000
? window size: 340 KByte (default)

[ 5] local 10.0.0.1 port 38452 connected with 10.0.0.2 port 4000
[ ID] Interval      Transfer    Bandwidth
[ 5] 0.0-10.0 sec  58.2 GBytes 50.0 Gbits/sec
bt@mininet-vmt:/media/sf_Project_3_Phase_3#
```

- e.
7. Controller and topology exited and restarted
- a. Ping All
    - i. 5 packets were sent to the controller.

ii.

```
mininet@mininet-vm: /m...a/sf_Project_3_Phase_3 - □ ×
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> h1 X
h3 -> h1 h2
*** Results: 50% dropped (3/6 received)
mininet> dpctl dump-flows
*** s1 -----
cookie=0x0, duration=79.905s, table=0, n_packets=1, n_bytes=98, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
cookie=0x0, duration=79.905s, table=0, n_packets=1, n_bytes=98, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
cookie=0x0, duration=69.878s, table=0, n_packets=1, n_bytes=98, dl_src=00:00:00:00:00:03, dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
cookie=0x0, duration=69.877s, table=0, n_packets=1, n_bytes=98, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
cookie=0x0, duration=59.850s, table=0, n_packets=1, n_bytes=98, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
cookie=0x0, duration=59.850s, table=0, n_packets=1, n_bytes=98, dl_src=00:00:00:00:00:03, dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
mininet>
```

iii.

```
mininet@mininet-vm: ~/pox
mininet@mininet-vm:~/pox$ ./pox.py samples.ethernet-learning
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
INFO:samples.ethernet-learning:Pair-Learning switch running.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:samples.ethernet-learning:Number of packets received so far: 1
INFO:samples.ethernet-learning:Packet has arrived: SRCMAC:00:00:00:00:00:01 DSTH
AC:00:00:00:00:00:02 from switch:12 in-port:1
INFO:samples.ethernet-learning:Packet flooded to 2 ports
INFO:samples.ethernet-learning:Number of packets received so far: 2
INFO:samples.ethernet-learning:Packet has arrived: SRCMAC:00:00:00:00:00:02 DSTH
AC:00:00:00:00:00:01 from switch:12 in-port:2
INFO:samples.ethernet-learning:Number of packets received so far: 3
INFO:samples.ethernet-learning:Packet has arrived: SRCMAC:00:00:00:00:00:01 DSTH
AC:00:00:00:00:00:03 from switch:12 in-port:1
INFO:samples.ethernet-learning:Packet flooded to 2 ports
INFO:samples.ethernet-learning:Number of packets received so far: 4
INFO:samples.ethernet-learning:Packet has arrived: SRCMAC:00:00:00:00:00:03 DSTH
AC:00:00:00:00:00:01 from switch:12 in-port:3
INFO:samples.ethernet-learning:Number of packets received so far: 5
INFO:samples.ethernet-learning:Packet has arrived: SRCMAC:00:00:00:00:00:02 DSTH
AC:00:00:00:00:00:03 from switch:12 in-port:2
```

4 messages were flooded (2 times, 2 ports each)

- b. No packets were sent to the controller. The rules were already established on the switch, which made it much easier to handle the ping all command.

## 2.5.2 (extra credit)

```
root@mininet-vm:/home/mininet/pox/pox/samples# ping 10.99.0.1 -c 3
PING 10.99.0.1 (10.99.0.1) 56(84) bytes of data.
From 10.0.1.100 icmp_seq=1 Destination Host Unreachable
From 10.0.1.100 icmp_seq=2 Destination Host Unreachable
From 10.0.1.100 icmp_seq=3 Destination Host Unreachable
```

- 1.
2. No, in our case, the source MAC address should not be changed because the source is initiating the communication. The source MAC address uniquely identifies the source device on the local network segment. When a host sends a packet, it expects responses to be sent back to its own MAC address. Changing the source MAC address would disrupt this expected behavior and potentially cause communication issues.

Description of self-learning algorithm:

The self-learning algorithm implemented in this controller enables the switches to dynamically learn the MAC addresses of devices connected to them by observing the source MAC addresses of incoming packets. Whenever a switch receives a packet, it checks the source MAC address and the port on which the packet arrived. It then updates its MAC address table to associate the MAC address with the corresponding input port. This allows the switch to build a mapping of MAC addresses to ports, facilitating efficient packet forwarding without relying on a centralized routing table.

PSEUDOCODE:

1. Import necessary libraries/modules
2. Define global variables:
  - mac\_address\_tables: a dictionary to store MAC address tables for each switch
  - flood\_counter: a dictionary to keep track of flooding
3. Define helper functions:
  - handle\_arp\_request(packet, packet\_input\_port, switch\_ID): Handles ARP requests
  - handle\_icmp\_packet(packet, packet\_input\_port, switch\_ID): Handles ICMP packets (Echo Requests)
  - handle\_ipv4\_packet(packet, packet\_input\_port, switch\_ID): Handles IPv4 packets
4. Define \_handle\_PacketIn(event) function:
  - Extract information from the event (packet, ports, MAC addresses, switch ID)
  - Log packet arrival information
  - Check the packet type:
    - If it's an ARP request:
      - Call handle\_arp\_request function
    - If it's an ICMP packet:
      - Call handle\_icmp\_packet function
    - If it's an IPv4 packet:
      - Call handle\_ipv4\_packet function
5. Define launch() function:
  - Register PacketIn event listener
  - Log that the switch is running