SDN-BASED STATELESS FIREWALL

Student Name: Hemendu Roy

Email: hroy6@asu.edu

Submission Date: 03-05-2022

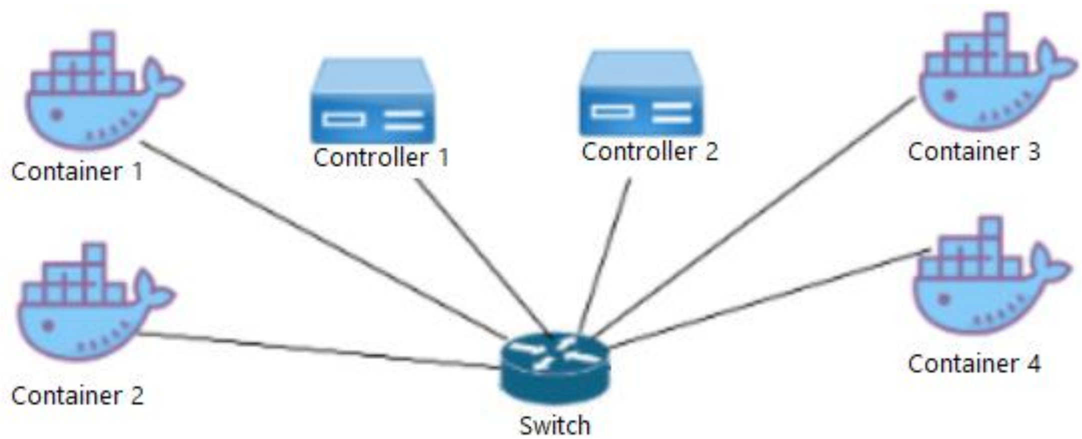
Class Name and Term: CSE 548 Spring 2022

I. PROJECT OVERVIEW

In this project, we're using mininet and containernet to develop a emulate Denial of Service (DoS) attacks in a Software Defined Network. The firewall in question is based on OpenFlow with flow-based policies to accept, propagate or drop packets wherein the first packet is inspected against policies and the rest of the data stream is dealt with only subsequently. We're enhancing this firewall by implementing port security to counter DoS attacks.

II. NETWORK SETUP

Network Topology and Configurations



In this setup, we have set up a mininet environment in containernet with 4 containernet hosts, one OVS switch and two remote controllers as shown in the figure above.

Initial Reachability

Initially, the assigned addresses of each host are as follows.

Container Host	Layer 2 address	Layer 3 address
h1	00:00:00:00:00:01	192.168.2.10
h2	00:00:00:00:00:02	192.168.2.20
h3	00:00:00:00:00:03	192.168.2.30
h4	00:00:00:00:00:04	192.168.2.40

III. Software

Open vSwitch – [1] Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols.

tcpdump – [2] tcpdump is used to capture network traffic on a network interface that match a set of Boolean expressions

Mininet – [3] Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native), in seconds, with a single command.

Containernet – [4] Containernet is a fork of the famous Mininet network emulator and allows to use Docker containers as hosts in emulated network topologies.

IV. Project Description

In this section, detailed descriptions of the project tasks will be illustrated.

Before developing the firewall, we first make sure our working environment is setup correctly.

Checking python installations.

```
root@ubuntu:/home/ubuntu/pox# python --version
Python 2.7.17
root@ubuntu:/home/ubuntu/pox# python3 --version
Python 3.6.9
```

Checking the mininet installation.

```
root@ubuntu:/home/ubuntu/pox# mn --version
2.3.0d5
root@ubuntu:/home/ubuntu/pox# mn --test pingall
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.455 seconds
```

Checking the POX installation.

```
root@ubuntu:/home/ubuntu/pox# ./pox.py -verbose forwarding.hub
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:forwarding.hub:Proactive hub running.
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.17/Feb 27 2021 15:10:58)
DEBUG:core:Platform is Linux-5.3.0-53-generic-x86_64-with-Ubuntu-18.04-bionic
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
^CINFO:core:Going down...
INFO:core:Down.
```

Lastly, we check if OVS is installed correctly.

```
root@ubuntu:/home/ubuntu/pox# ovs-vsctl --version
ovs-vsctl (Open vSwitch) 2.17.90
DB Schema 8.3.0
```

Now that we have verified that our environment is setup correctly, we can begin modifying the firewall rules.

The rules for Layer 2 can be found in */home/ubuntu/pox/l2firewall.config*
The rules for Layer 3 can be found in */home/ubuntu/pox/l3firewall.config*

Assessments

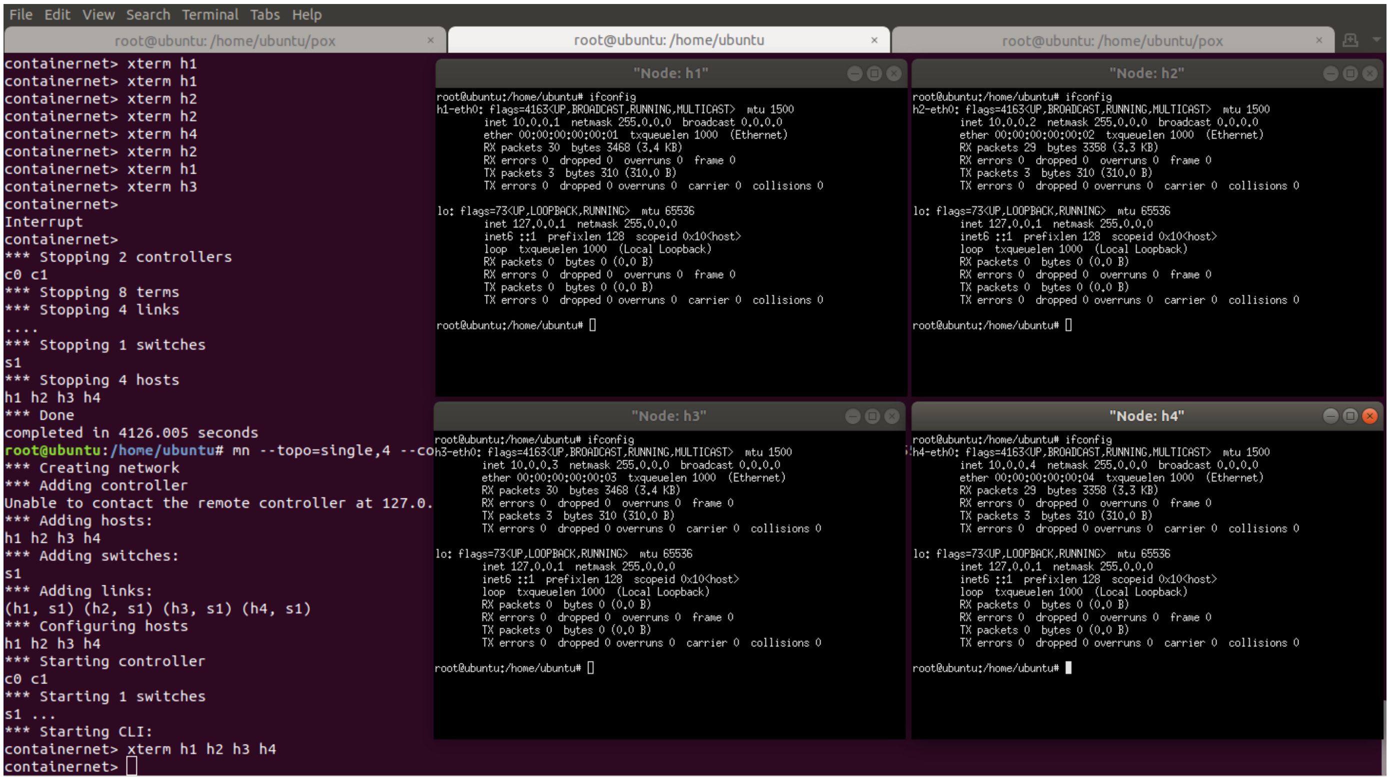
1. Create a mininet based topology with 4 container hosts and one controller switches and run it.

We create the network using the following command
sudo mn --topo=single,4 --controller=remote,port=6633 --controller=remote,port=6655 --switch=ovsk –mac

```
root@ubuntu:/home/ubuntu# mn --topo=single,4 --controller=remote,port=6633 --controller=remote,port=6655 --switch=ovsk --mac
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** 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 c1
*** Starting 1 switches
s1 ...
*** Starting CLI:
containernet>
```

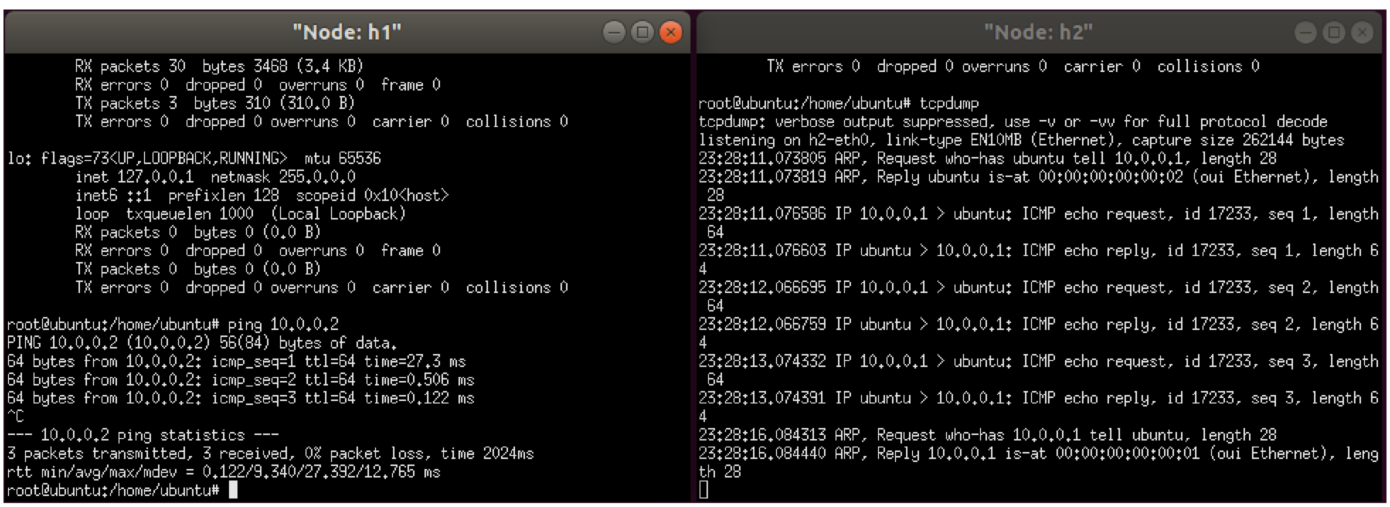
Once created, a containernet CLI prompt appears which can now use to work with our host machines.

Now, to verify that our containernet hosts are up and running, we can use *xterm*.



Moreover, we can use *ifconfig* to observe the assigned MAC addresses and IPs of each host as shown above.

To verify connectivity between the hosts, we can use tcpdump.



Now, our mininet environment is working correctly.

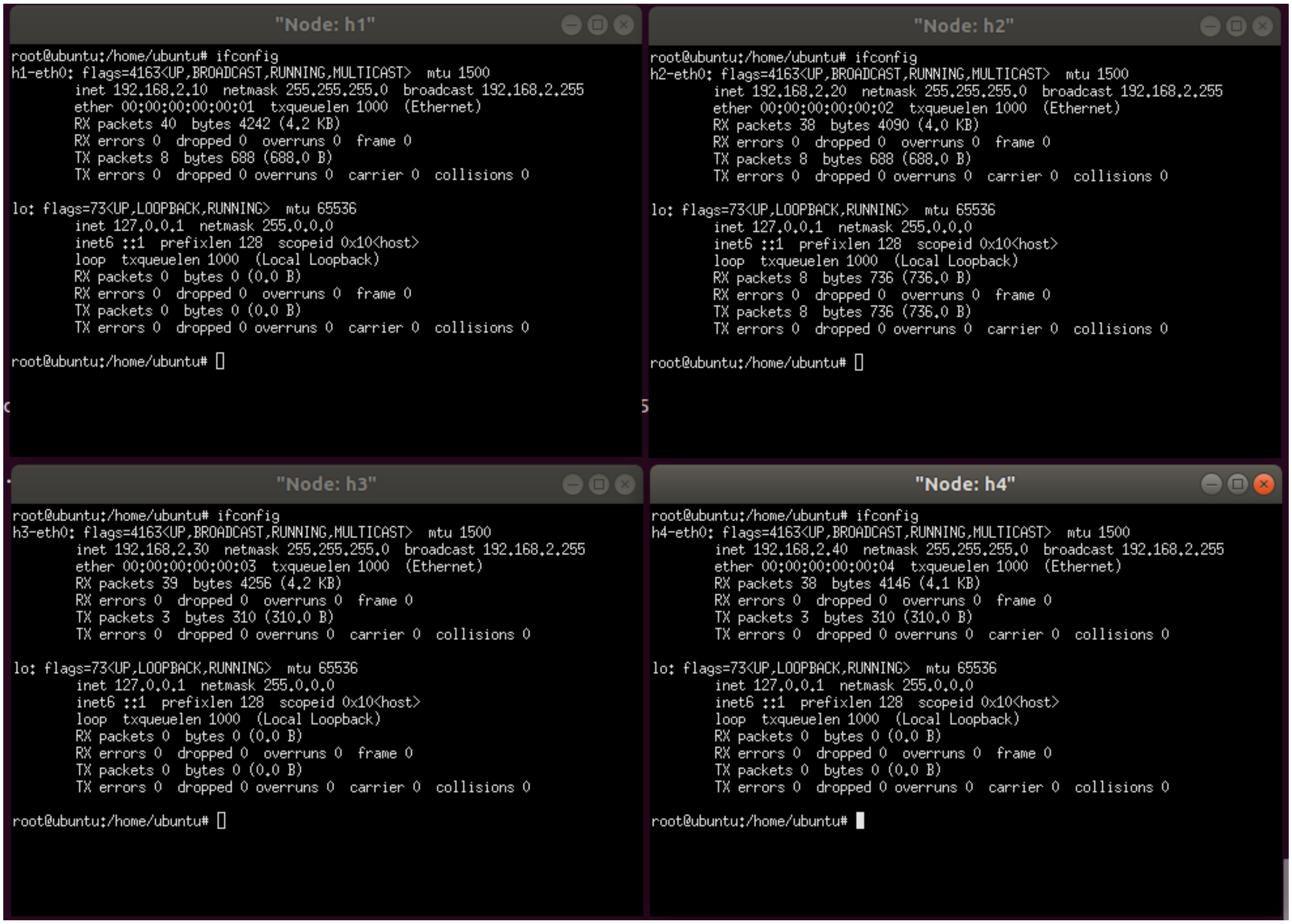
2. Make the interfaces up and assign IP addresses to interfaces of container hosts.

To assign IP addresses of our choice, we can use the following set of commands.

h1 ifconfig h1-eth0 192.168.2.10
h2 ifconfig h2-eth0 192.168.2.20
h3 ifconfig h3-eth0 192.168.2.30
h4 ifconfig h4-eth0 192.168.2.40

```
containernet> h1 ifconfig h1-eth0 192.168.2.10
containernet> h2 ifconfig h2-eth0 192.168.2.20
containernet> h3 ifconfig h3-eth0 192.168.2.30
containernet> h4 ifconfig h4-eth0 192.168.2.40
containernet>
```

Once again, we use *ifconfig* to verify the new IPs of the hosts.

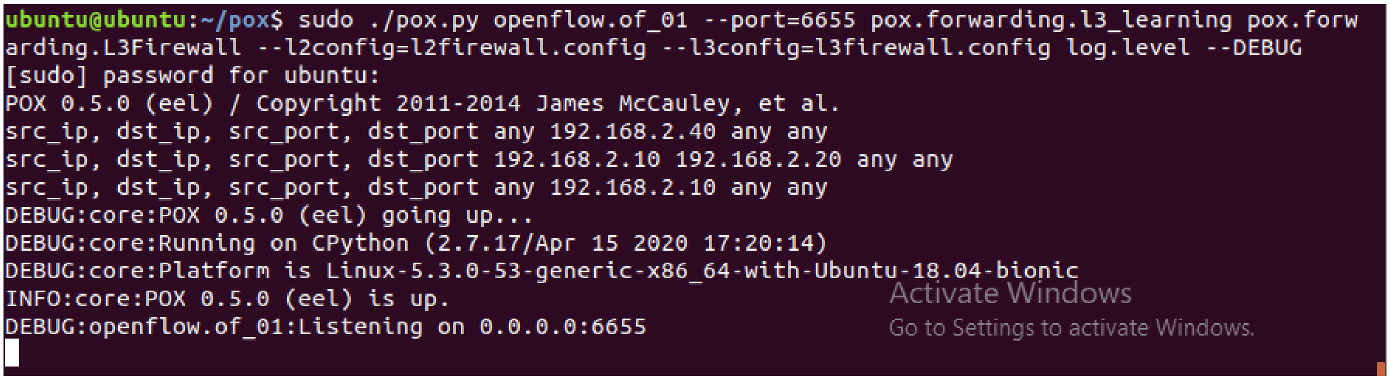


Assessment 3

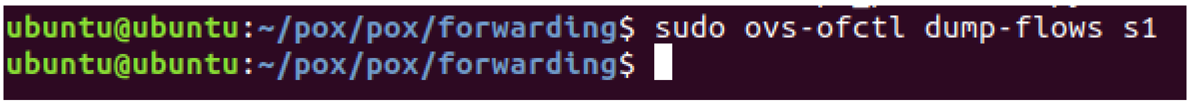
(a) Run l3_learning application in POX controller

We run the command,
sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l3_learning pox.forwarding.L3Firewall --l2config=l2firewall.config --l3config=l3firewall.config log.level –DEBUG

Now, we can see that the POX controller is up and running on localhost and on port 6655

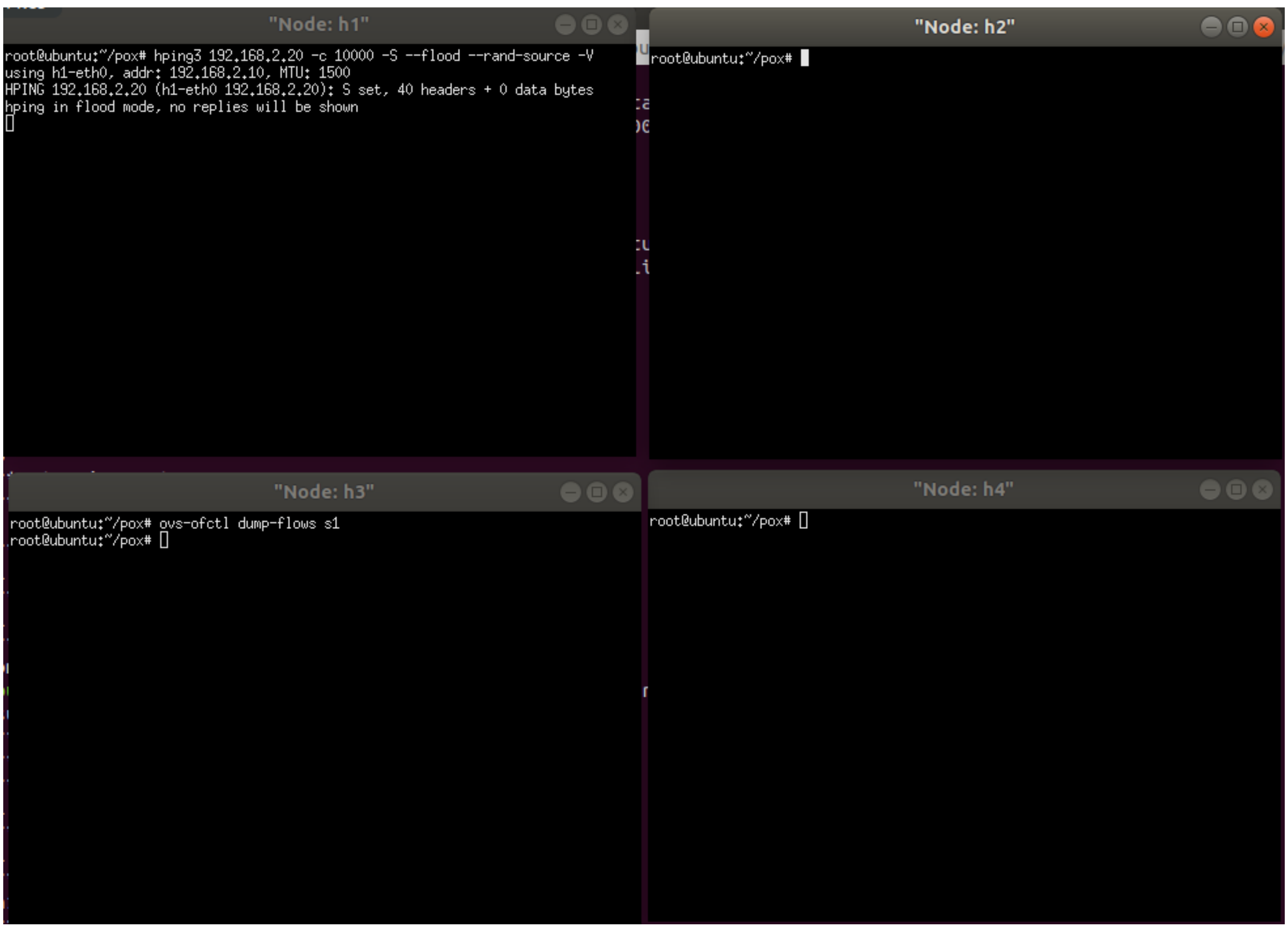


(b) Check openflow flow-entries on switch 1
We run the command *sudo ovs-ofctl dump-flows s1*.
Since we have not transmitted any packets between the hosts yet, the openflow entries are empty as expected

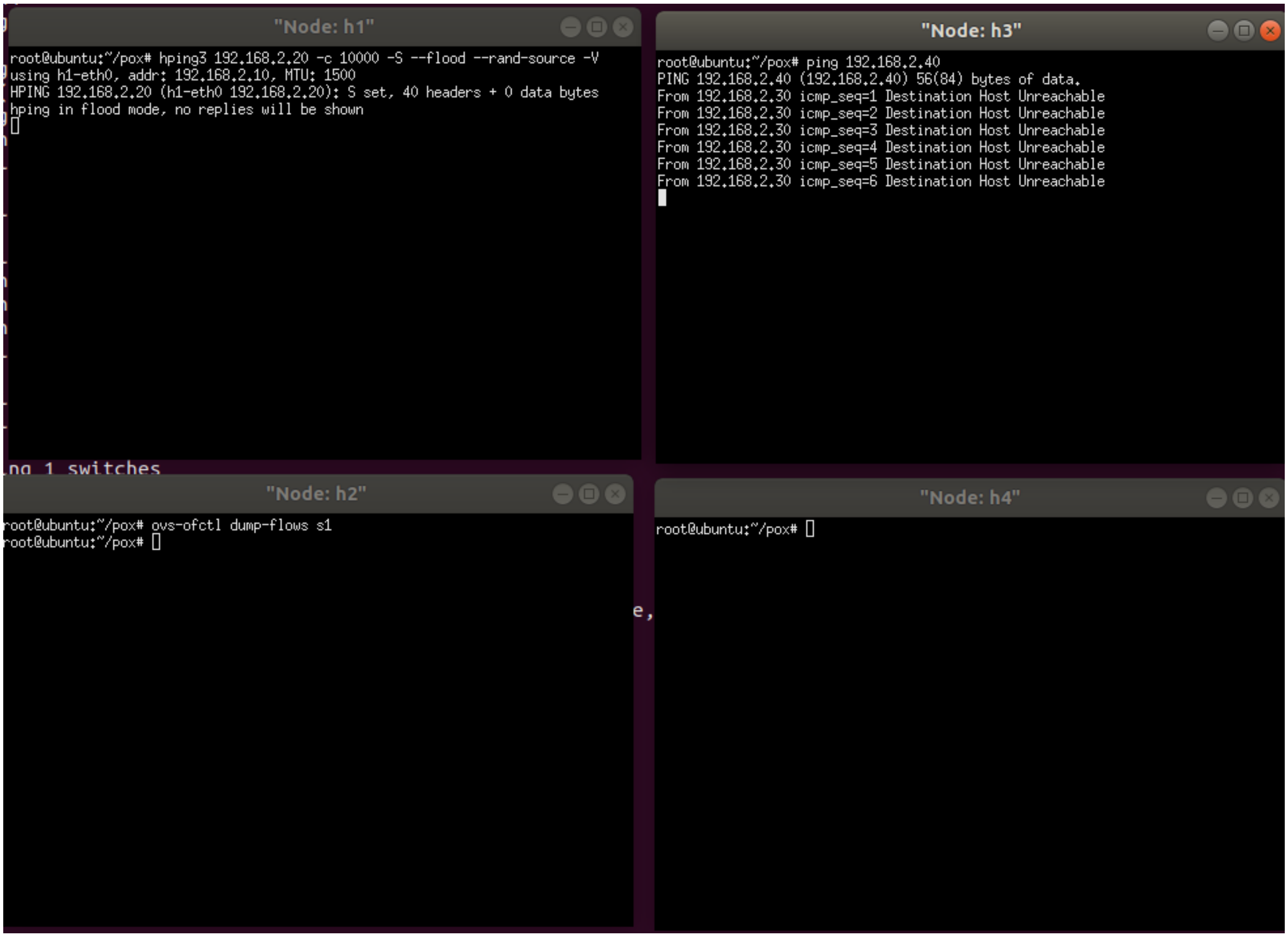


(c) Start flooding from any container host to container host #2

We start flooding from host 1 to host 2 using the command,
hping3 192.168.2.20 -c 10000 -S –flood –rand-source -V



We can see that host 3 is not able to ping host 4 because the controller has been overloaded as a result of the DoS attack from host 1.



(d) Check openflow flow-entries on switch 1

We can see thousands of entries when we dump the flows as a result of the DoS attack from host 1.

Assessment 4

- You should illustrate (through screenshots and descriptions) your implemented program codes
- You should demo how your implementation can mitigate the DoS through a sequence of screenshots with explanation.
- You should submit the source codes of your implementation.

The code for the DoS detection has been implemented using Python. It can be found in the file, L3Firewall.py which will be detailed in the Appendix.

To maintain what flows have been observed by the swtich, a record of the MAC address, source IP ,destination IP and switch port will be stored for every network transaction.

For example, if a packet were to be sent from source MAC address 00:00:00:00:00:01 with source IP 192.168.1.10 to destination IP 192.168.2.20 over switch port 1, a the mapping dict will be updated as follows
{"00:00:00:00:00:01":[" 192.168.1.10"," 192.168.1.20","1"]}

Subsequent records will be added similarly with the MAC addresses of new transactions as the dictionary key and the value, an array containing the source and destination IPs and the switch port.

First, we will consider the scenario of IP addresses being spoofed and later, we will demonstrate DoS mitigation despite MAC address spoofing as part of the bonus section.

If a host IP address were to be spoofed, the only difference in our new dictionary records would be the source IP i.e. **RecordTable['<MAC address>'][0]**
The rest of the entries would be identical to the record that was generated before spoofing.

To test our code against this, we can perform the following steps.

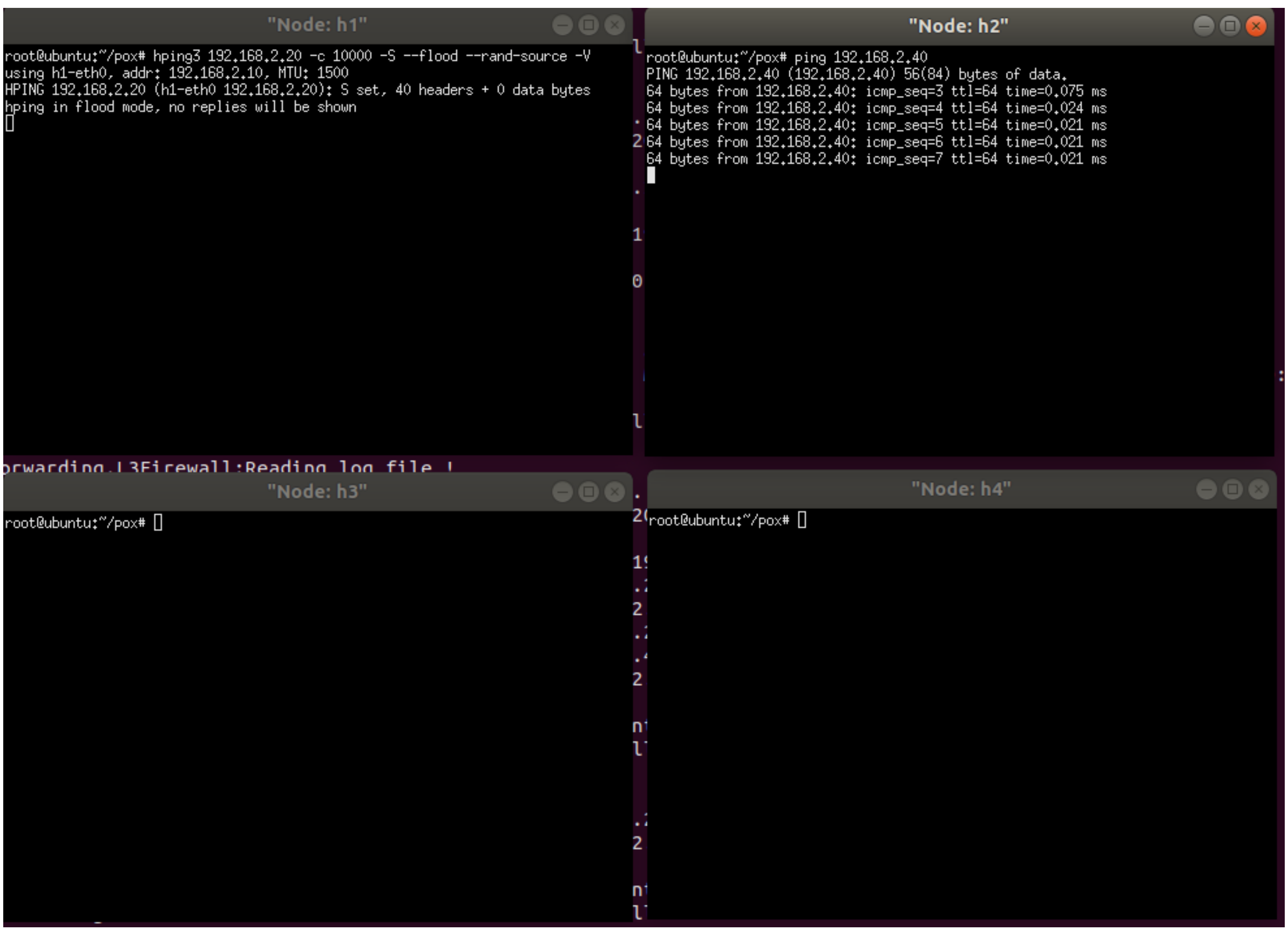
Run POX controller,

```
ubuntu@ubuntu:~/pox$ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l3_learning pox.forw
arding.L3Firewall --l2config=l2firewall.config --l3config=l3firewall.config log.level --DEBUG
[sudo] password for ubuntu:
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
src_ip, dst_ip, src_port, dst_port any 192.168.2.20 any any
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.17/Apr 15 2020 17:20:14)
DEBUG:core:Platform is Linux-5.3.0-53-generic-x86_64-with-Ubuntu-18.04-bionic
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6655
```

Next, bring up mininet topology

Now, when we flood the controller from host 1, host 2 should still be able to ping other hosts because a dict entry will already be made for the MAC address of host 1. When new IPs using the same MAC are detected, the packets will be dropped.

We can observe that even though the controller is being flooded, we can ping from host 2 to host 4 which proves that the DoS attack has been mitigated



The POX controller also notifies us that a Fake IP address has been detected because an existing IP is already associated with the same MAC address

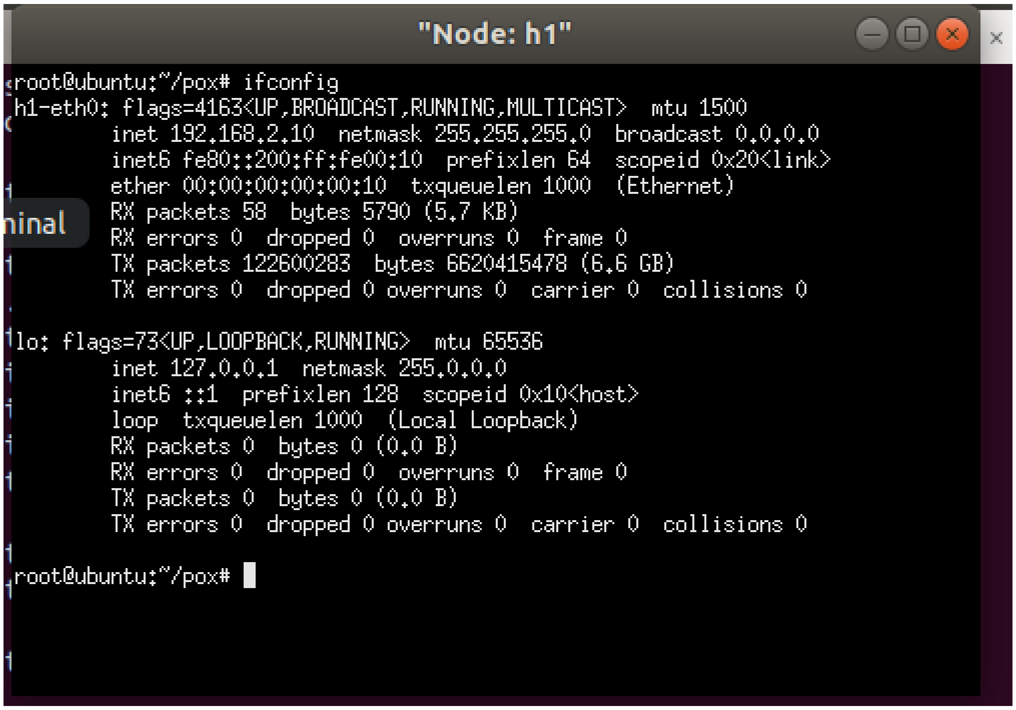
```
ubuntu@ubuntu:~/pox$ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l3_learning pox.forw
arding.L3Firewall --l2config=l2firewall.config --l3config=l3firewall.config log.level --DEBUG
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.17/Apr 15 2020 17:20:14)
DEBUG:core:Platform is Linux-5.3.0-53-generic-x86_64-with-Ubuntu-18.04-bionic
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6655
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:forwarding.l3_learning:1 1 ARP request 192.168.2.10 => 192.168.2.20
DEBUG:forwarding.l3_learning:1 1 learned 192.168.2.10
DEBUG:forwarding.l3_learning:1 1 flooding ARP request 192.168.2.10 => 192.168.2.20
DEBUG:forwarding.l3_learning:1 2 ARP reply 192.168.2.20 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 2 learned 192.168.2.20
DEBUG:forwarding.l3_learning:1 2 flooding ARP reply 192.168.2.20 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 1 IP 103.148.119.145 => 192.168.2.20
DEBUG:forwarding.l3_learning:1 1 learned 103.148.119.145
DEBUG:forwarding.l3_learning:1 1 installing flow for 103.148.119.145 => 192.168.2.20 out port 2
DEBUG:forwarding.L3Firewall:Inside checkSecPort function
DEBUG:forwarding.L3Firewall:New entry is: 00:00:00:00:00:01, 103.148.119.145, 192.168.2.20, 1
DEBUG:forwarding.L3Firewall:Safe...proceeding flow
DEBUG:forwarding.l3_learning:1 1 IP 217.44.104.185 => 192.168.2.20
DEBUG:forwarding.l3_learning:1 1 learned 217.44.104.185
DEBUG:forwarding.l3_learning:1 1 installing flow for 217.44.104.185 => 192.168.2.20 out port 2
DEBUG:forwarding.L3Firewall:Inside checkSecPort function
DEBUG:forwarding.L3Firewall:Fake IP! MAC 00:00:00:00:00:01 IP 103.148.119.145 port 192.168.2.20,
start 192.168.2.20 1
DEBUG:forwarding.L3Firewall:new log entered
```


Similarly, we can check if morphing a MAC address can also be mitigated. Now, we will check if a new MAC address is already associated with an existing IP address.

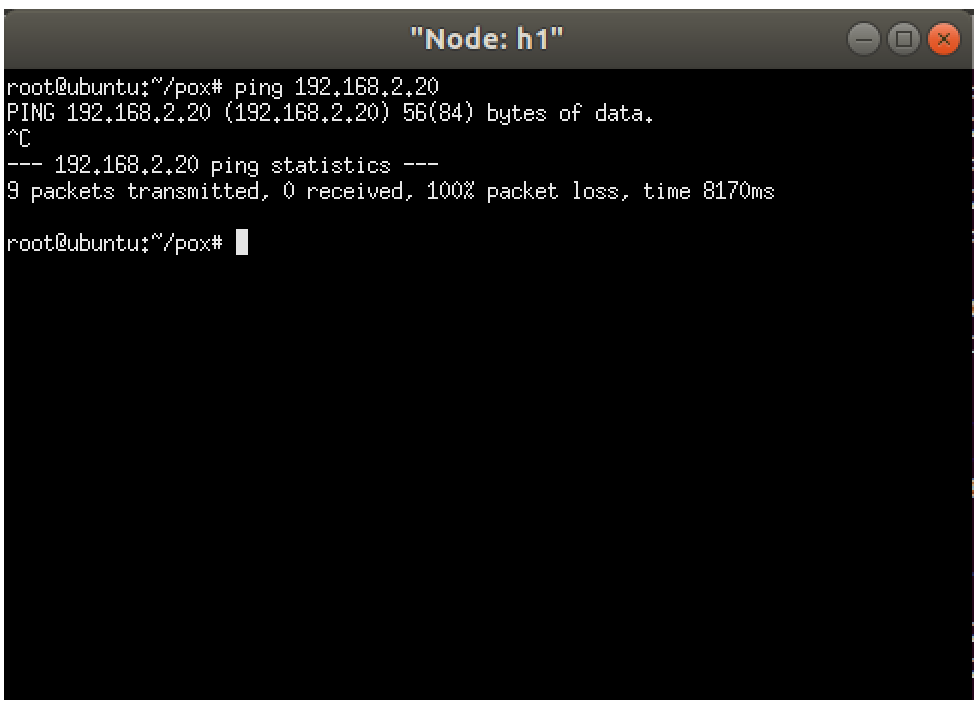
To change the MAC address of a containernet host, run the command,

py h1.setMAC('00:00:00:00:00:10')

We can see that the MAC address has changed.



We can see that if we try to ping from host 1 to host 2, all packets are blocked. Therefore, MAC address spoofing has also been mitigated.



V. Conclusion

I learnt about what the mininet and containernet tools do and how to use them.

More importantly, I understood the theory behind a flow based firewall and how to setup and configure one.

I also learnt how to operate the Open vSwtich tool using its CLI commands.

I learnt how to mitigate DoS attacks both by IP spoofing as well as MAC spoofing.

VI. APPENDIX B: ATTACHED FILES

Demo Video - https://www.youtube.com/watch?v=GY7Ts9JUBr8

VII. REFERENCES

[1] Open vSwitch - https://www.openvswitch.org/
[2] tcpdump Linux man page - https://linux.die.net/man/8/tcpdump
[3] Mininet - http://mininet.org/
[4] Containernet - https://containernet.github.io/