

SDN-BASED STATELESS FIREWALL

Student Name: Hemendu Roy

Email: hroy6@asu.edu

Submission Date: 03-05-2022

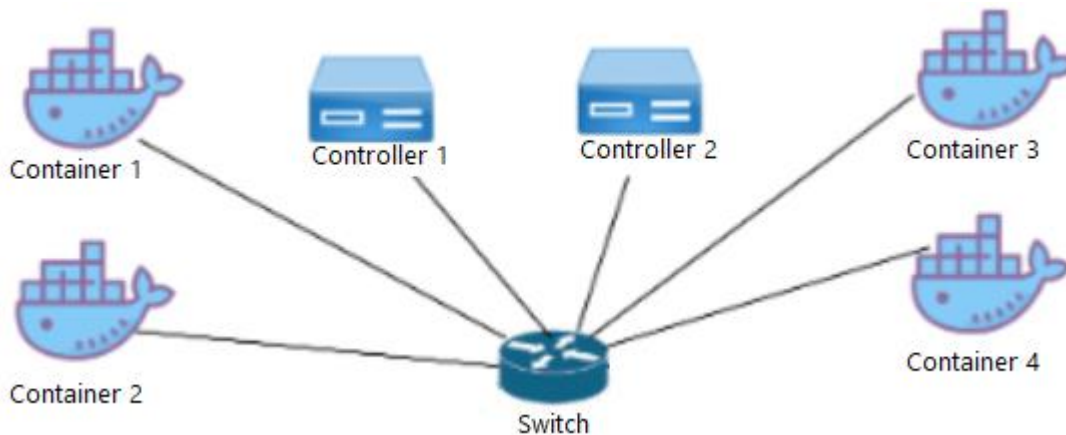
Class Name and Term: CSE 548 Spring 2022

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*.

```
File Edit View Search Terminal Tabs Help
root@ubuntu:/home/ubuntu/pox x root@ubuntu:/home/ubuntu x root@ubuntu:/home/ubuntu/pox x
containernet> xterm h1
containernet> xterm h1
containernet> xterm h2
containernet> xterm h2
containernet> xterm h4
containernet> xterm h2
containernet> xterm h1
containernet> xterm h3
containernet>
Interrupt
containernet>
*** Stopping 2 controllers
c0 c1
*** Stopping 8 terms
*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 4126.005 seconds
root@ubuntu:/home/ubuntu# mn --topo=single,4 --c0h3-eth0:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.
*** 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> xterm h1 h2 h3 h4
containernet>

"Node: h1"
root@ubuntu:/home/ubuntu# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0
    ether 00:00:00:00:00:01 txqueuelen 1000 (Ethernet)
    RX packets 30 bytes 3468 (3.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 310 (310.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:/home/ubuntu#

"Node: h2"
root@ubuntu:/home/ubuntu# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.2 netmask 255.0.0.0 broadcast 0.0.0.0
    ether 00:00:00:00:00:02 txqueuelen 1000 (Ethernet)
    RX packets 29 bytes 3358 (3.3 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 310 (310.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:/home/ubuntu#

"Node: h3"
root@ubuntu:/home/ubuntu# ifconfig
h3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.3 netmask 255.0.0.0 broadcast 0.0.0.0
    ether 00:00:00:00:00:03 txqueuelen 1000 (Ethernet)
    RX packets 30 bytes 3468 (3.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 310 (310.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:/home/ubuntu#

"Node: h4"
root@ubuntu:/home/ubuntu# ifconfig
h4-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.4 netmask 255.0.0.0 broadcast 0.0.0.0
    ether 00:00:00:00:00:04 txqueuelen 1000 (Ethernet)
    RX packets 29 bytes 3358 (3.3 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 310 (310.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:/home/ubuntu#
```

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*.

```
"Node: h1"
RX packets 30 bytes 3468 (3.4 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 3 bytes 310 (310.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:/home/ubuntu# ping 10.0.0.2
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=27.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.506 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.122 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2024ms
rtt min/avg/max/mdev = 0.122/9.340/27.392/12.765 ms
root@ubuntu:/home/ubuntu#

"Node: h2"
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:/home/ubuntu# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
23:28:11.073805 ARP, Request who-has ubuntu tell 10.0.0.1, length 28
23:28:11.076586 IP 10.0.0.1 > ubuntu: ICMP echo request, id 17233, seq 1, length 64
23:28:11.076603 IP ubuntu > 10.0.0.1: ICMP echo reply, id 17233, seq 1, length 64
23:28:12.066695 IP 10.0.0.1 > ubuntu: ICMP echo request, id 17233, seq 2, length 64
23:28:12.066759 IP ubuntu > 10.0.0.1: ICMP echo reply, id 17233, seq 2, length 64
23:28:13.074332 IP 10.0.0.1 > ubuntu: ICMP echo request, id 17233, seq 3, length 64
23:28:13.074391 IP ubuntu > 10.0.0.1: ICMP echo reply, id 17233, seq 3, length 64
23:28:16.084313 ARP, Request who-has 10.0.0.1 tell ubuntu, length 28
23:28:16.084440 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01 (oui Ethernet), length 28
^C
```

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.

The image shows four terminal windows, each representing a different node in a network. Each window displays the output of the `ifconfig` command, showing the configuration for the `h1-eth0` (or `h2-eth0`, `h3-eth0`, `h4-eth0`) and `lo` interfaces. The `h1-eth0` interface is configured with IP 192.168.2.10, netmask 255.255.255.0, and broadcast 192.168.2.255. The `h2-eth0` interface is configured with IP 192.168.2.20, netmask 255.255.255.0, and broadcast 192.168.2.255. The `h3-eth0` interface is configured with IP 192.168.2.30, netmask 255.255.255.0, and broadcast 192.168.2.255. The `h4-eth0` interface is configured with IP 192.168.2.40, netmask 255.255.255.0, and broadcast 192.168.2.255. The `lo` interface is configured with IP 127.0.0.1, netmask 255.0.0.0, and broadcast 127.0.0.1. The `h1-eth0` interface is also configured with a loopback address 127.0.0.1 and a netmask of 255.0.0.0. The `h2-eth0` interface is also configured with a loopback address 127.0.0.1 and a netmask of 255.0.0.0. The `h3-eth0` interface is also configured with a loopback address 127.0.0.1 and a netmask of 255.0.0.0. The `h4-eth0` interface is also configured with a loopback address 127.0.0.1 and a netmask of 255.0.0.0.

```

"Node: h1"
root@ubuntu:/home/ubuntu# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.10 netmask 255.255.255.0 broadcast 192.168.2.255
    ether 00:00:00:00:00:01 txqueuelen 1000 (Ethernet)
    RX packets 40 bytes 4242 (4.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8 bytes 688 (688.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:/home/ubuntu#

"Node: h2"
root@ubuntu:/home/ubuntu# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.20 netmask 255.255.255.0 broadcast 192.168.2.255
    ether 00:00:00:00:00:02 txqueuelen 1000 (Ethernet)
    RX packets 38 bytes 4090 (4.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8 bytes 688 (688.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 8 bytes 736 (736.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8 bytes 736 (736.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:/home/ubuntu#

"Node: h3"
root@ubuntu:/home/ubuntu# ifconfig
h3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.30 netmask 255.255.255.0 broadcast 192.168.2.255
    ether 00:00:00:00:00:03 txqueuelen 1000 (Ethernet)
    RX packets 39 bytes 4256 (4.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 310 (310.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:/home/ubuntu#

"Node: h4"
root@ubuntu:/home/ubuntu# ifconfig
h4-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.40 netmask 255.255.255.0 broadcast 192.168.2.255
    ether 00:00:00:00:00:04 txqueuelen 1000 (Ethernet)
    RX packets 38 bytes 4146 (4.1 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 310 (310.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:/home/ubuntu#

```

Assessment 3

(a) Run `I3_learning` application in POX controller

We run the command,

```

sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.I3_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

```

ubuntu@ubuntu:~/pox$ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l3_learning pox.forwarding.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 192.168.2.40 any any
src_ip, dst_ip, src_port, dst_port 192.168.2.10 192.168.2.20 any any
src_ip, dst_ip, src_port, dst_port any 192.168.2.10 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

```

(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

```

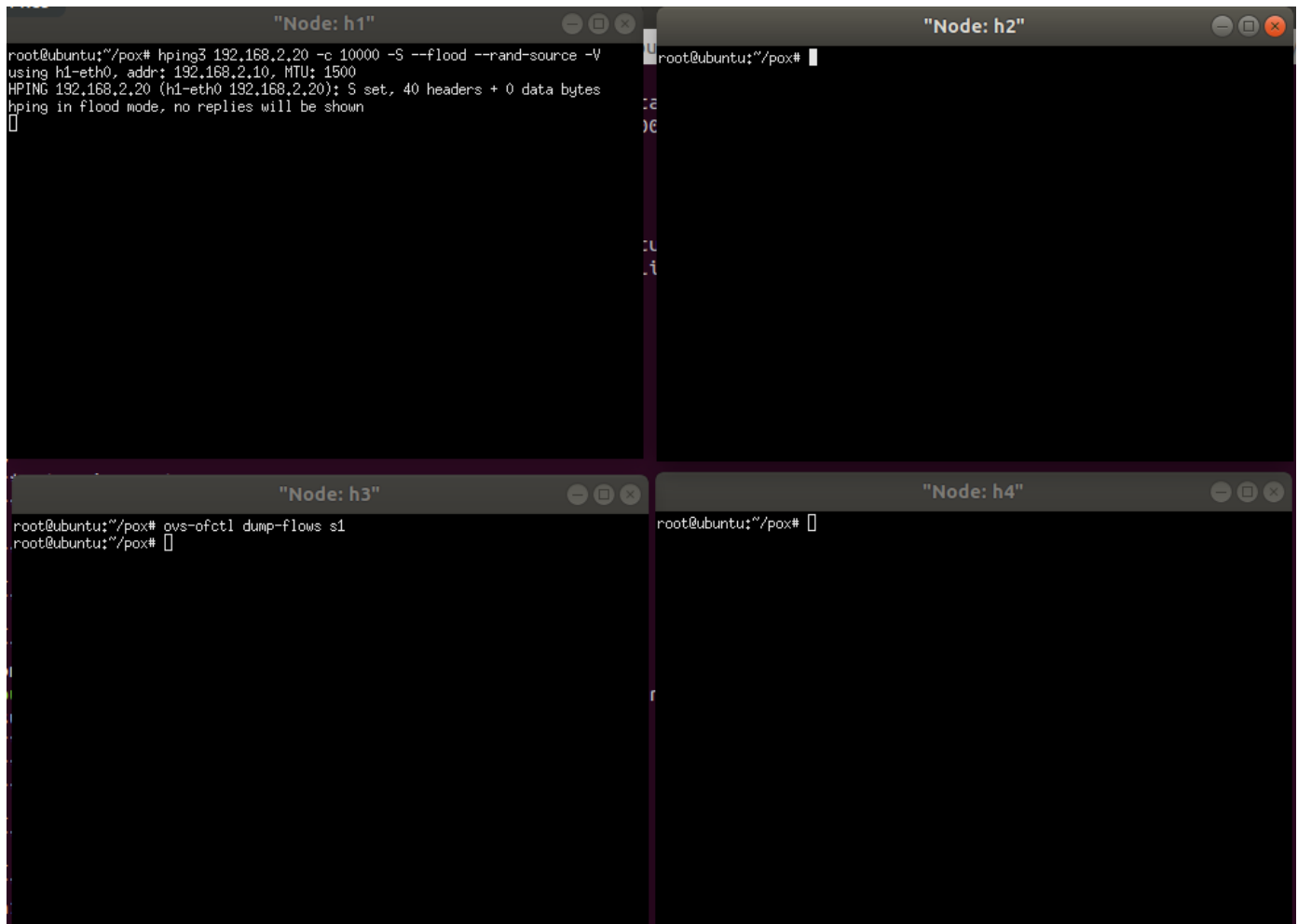
ubuntu@ubuntu:~/pox/pox/forwarding$ sudo ovs-ofctl dump-flows s1
ubuntu@ubuntu:~/pox/pox/forwarding$

```

(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.

The image displays four terminal windows arranged in a 2x2 grid, each representing a different node in a network simulation.

- Top-left window ("Node: h1"):** Shows a command to flood host 3 using hping3. The output indicates that hping is in flood mode and no replies will be shown.
- Top-right window ("Node: h3"):** Shows a command to ping host 4 (192.168.2.40). The output shows six failed ping attempts, each resulting in "Destination Host Unreachable".
- Bottom-left window ("Node: h2"):** Shows a command to dump OpenFlow flow entries on switch s1. The output is empty, indicating no flow entries are currently present.
- Bottom-right window ("Node: h4"):** Shows a command to dump OpenFlow flow entries on switch s1. The output is empty, indicating no flow entries are currently present.

(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.

The image shows a terminal window for "Node: h3" displaying the output of the command `ovs-ofctl dump-flows s1`. The output lists numerous flow entries, each with details such as cookie, duration, table, n_packets, n_bytes, idle_timeout, priority, and actions. The entries are truncated on the right side of the terminal window, indicating a large number of flows.

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 switch, 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.forwarding.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


```

completed in 19.54228 seconds
ubuntu@ubuntu:~/pox$ sudo mn --topo=single,4 --controller=remote,port=6655 --switch=ovsk --mac
[sudo] password for ubuntu:
*** Creating network
*** Adding controller
*** 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:
containernet>

```

Activate Windows
Go to Settings to activate Windows.

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

```

"Node: h1"
root@ubuntu:~/pox# hping3 192.168.2.20 -c 10000 -S --flood --rand-source -V
using h1-eth0, addr: 192.168.2.10, MTU: 1500
HPING 192.168.2.20 (h1-eth0 192.168.2.20): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown

"Node: h2"
root@ubuntu:~/pox# ping 192.168.2.40
PING 192.168.2.40 (192.168.2.40) 56(84) bytes of data:
64 bytes from 192.168.2.40: icmp_seq=3 ttl=64 time=0.075 ms
64 bytes from 192.168.2.40: icmp_seq=4 ttl=64 time=0.024 ms
64 bytes from 192.168.2.40: icmp_seq=5 ttl=64 time=0.021 ms
64 bytes from 192.168.2.40: icmp_seq=6 ttl=64 time=0.021 ms
64 bytes from 192.168.2.40: icmp_seq=7 ttl=64 time=0.021 ms

.
1
0

"Node: h3"
root@ubuntu:~/pox#

"Node: h4"
root@ubuntu:~/pox#

```

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.forwarding.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.

```

"Node: h1"
root@ubuntu:~/pox# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.10 netmask 255.255.255.0 broadcast 0.0.0.0
    inet6 fe80::200:ff:fe00:10 prefixlen 64 scopeid 0x20<link>
    ether 00:00:00:00:00:10 txqueuelen 1000 (Ethernet)
    RX packets 58 bytes 5790 (5.7 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 122600283 bytes 6620415478 (6.6 GB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ubuntu:~/pox#

```

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.

```

"Node: h1"
root@ubuntu:~/pox# ping 192.168.2.20
PING 192.168.2.20 (192.168.2.20) 56(84) bytes of data.
^C
--- 192.168.2.20 ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8170ms

root@ubuntu:~/pox#

```

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 vSwitch 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/>