

Documentation for Setup of SDN Based Threat Vector and Security Solutions Testing Environment

Contents

1. Pre-Requisites for environment setup
 - a. Virtual Box
 - b. Mininet Network Emulator
 - c. Start Mininet
 - d. Python on Mininet
 - e. Scapy on Mininet
 - f. Pytz on Mininet
2. Set up Test Environment
3. Steps to perform the experiment
 - a. Find the Threshold for usual Traffic (normal)
 - b. Detection of DDOS threat using the value of Entropy
4. Additional details and Explanation

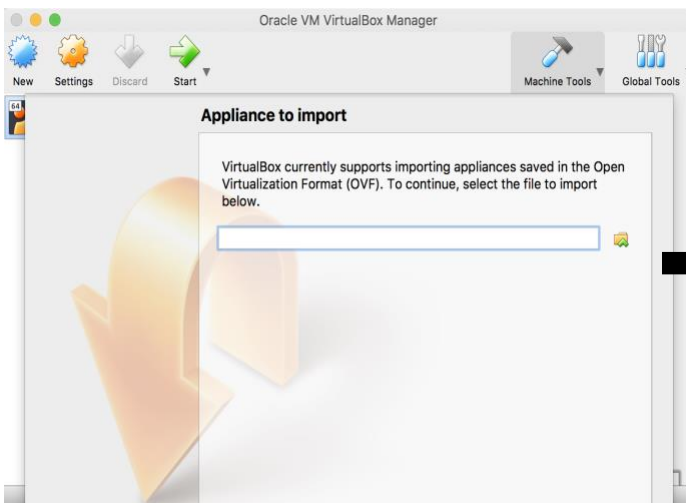
1. Pre-Requisites for environment setup

a. Virtual Box

- Download and install the latest version of Virtual Box for your operating system using the following link:
<https://www.virtualbox.org/wiki/Downloads>

b. Mininet Network Emulator

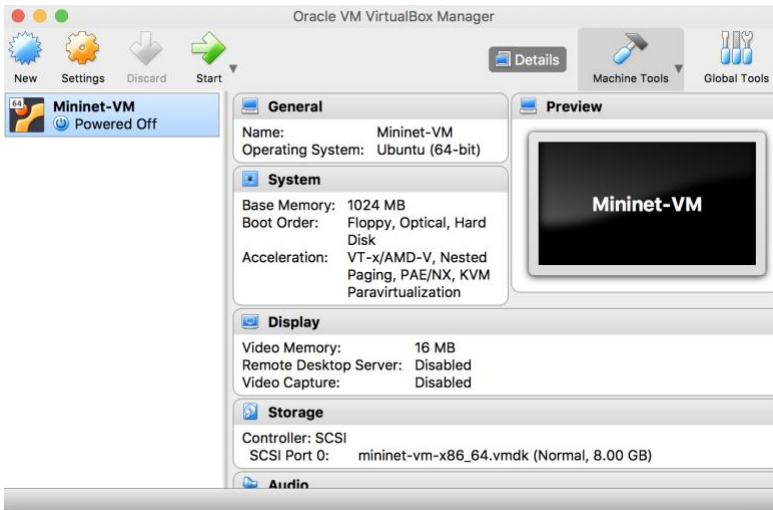
- First ensure that a suitable version of Virtual Box has been installed into your operating system.
- Download a Mininet VM image appropriate for your system using the following link:
<https://github.com/mininet/mininet/wiki/Mininet-VM-Images>
- Open the Virtual Box application and click on *File* → *Import Appliance*.
- Find the '.ovf' Mininet VM image that you just downloaded and click Continue to import it.
- This process should install Mininet on the Virtual Box.



mn-trusty64server-170321-14-17-08	Aug 19, 2018 at 11:11
mininet-2.2.2-170321-ubuntu-14.04.4-server-amd64.ovf	14:29
mininet-vm-x86_64.vmdk	Mar 21, 2017 at 14:29
BASEKIT EVE	Mar 21, 2017 at 14:29

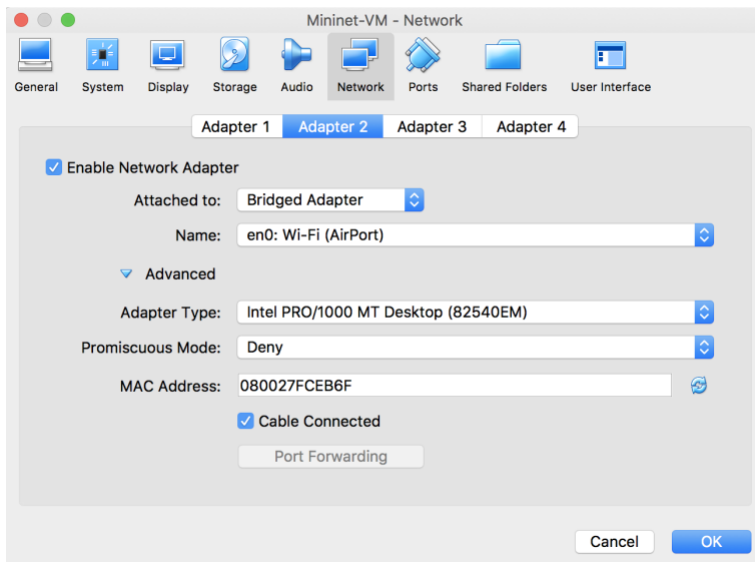
Import Mininet VM Image

- Once successfully installed the Mininet Virtual Machine will be visible in Virtual Box.



Mininet VM installed

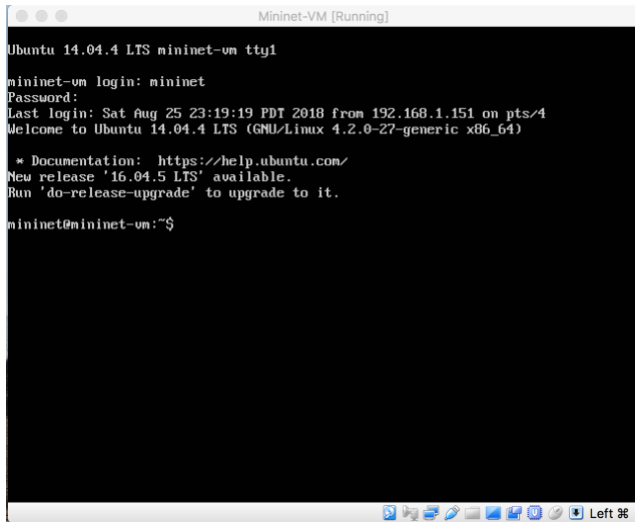
- Select the Mininet-VM and click on *Settings*.
- Within the *Settings* menu, navigate to *Network*.
- Click on *Adapter 2* and then check the box for Enable Network Adapter.
- Change the 'Attached to:' option to be *Bridged Adapter*.
- Set its name to Wi-Fi and click OK. (Note: If not available, set it to the default name)



Mininet-VM Network Settings

c. Start Mininet

- Select the Mininet-VM in Virtual Box and click on Start.
- Mininet will prompt you to enter a login ID and password. The default login ID and password is simply 'mininet'.



```
Mininet-VM [Running]
Ubuntu 14.04.4 LTS mininet-vm tty1
mininet-vm login: mininet
Password:
Last login: Sat Aug 25 23:19:19 PDT 2018 from 192.168.1.151 on pts/4
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
New release '16.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

mininet@mininet-vm:~$
```

Mininet Login

d. Python on Mininet

- Type the following command to check if python is installed on mininet:

```
$ python -V
```

- The above command should display the version of python.
- If python is not installed on mininet, type in the following command to install it manually.

```
$ sudo apt-get install python
```

```
mininet@mininet-vm:~$ python -V
Python 2.7.6
```

If python is installed

e. Scapy on Mininet

- Type the following command to check if scapy is installed on mininet:

```
$ sudo scapy
```

If you see the following output, scapy has been pre-installed.

```
mininet@mininet-vm:~$ sudo scapy
INFO: Can't import python gnuplot wrapper . Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
INFO: No IPv6 support in kernel
WARNING: No route found for IPv6 destination :: (no default route?)
INFO: Can't import python Crypto lib. Won't be able to decrypt WEP.
INFO: Can't import python Crypto lib. Disabled certificate manipulation tools
Welcome to Scapy (2.2.0)
>>>
```

If Scapy is pre-installed

- If you receive the above output, type `quit()` to exit from scapy.
- If an error message pops up instead, you have to install scapy manually.
- To install scapy manually type in the following command:

```
$ sudo apt-get install python-scapy
```

f. Pytz on Mininet

- Type the following command to install pip on mininet:

```
$ sudo apt-get install python-pip
```

Once pip is installed, enter the following command to install pytz

```
$ sudo pip install pytz
```

Pytz is required to display timestamps in your local timezone.

2. Creating the Test Environment

I. For Windows Users

- a. Launch the Xming application.
- b. Start and log into Mininet.
- c. Identify your VM's working IP Address by typing the following command:
`$ ifconfig eth0`
- d. Open the putty application.
- e. In the left navigation bar, click on *SSH* → *X11*.
- f. Then check the box that says *Enable X11 forwarding*. This allows you to use Xming and open Xterm windows.
- g. In the left navigation bar, click on *Session*.
- h. Then under 'Host Name (or IP address)' enter the IP Address you found in step c.
- i. Click *Open* and you will see a command line window on your screen.
- j. The window will prompt you to enter login details. The username and password are once again 'mininet'. You have now opened an SSH window of mininet.

II. For Mac users

- k. Start and log into Mininet.
- l. Identify your VM's working IP Address by typing the following command:

`$ ifconfig eth0`

```
mininet@mininet-vm:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 08:00:27:fc:eb:6f
          inet addr:192.168.1.152  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3155 errors:0 dropped:0 overruns:0 frame:0
          TX packets:166 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:358329 (358.3 KB)  TX bytes:13934 (13.9 KB)
```

Identify IP Address

- m. SSH into mininet from your local host using the following command.

`$ ssh -X mininet@IPAddress`

Where the IP Address is the Address found in the previous step.

You will be prompted to enter a password. The password by default is 'mininet'.

The -X command enables secure X11 forwarding using xwing/xquartz/xterm.

```
$ ssh -X mininet@192.168.1.152
mininet@192.168.1.152's password:
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)

* Documentation:  https://help.ubuntu.com/
```

III. Common Steps for all operating systems

- n. Move to the directory mininet/custom by entering the following command:

```
$ cd mininet/custom
```

- o. Create a new python script for normal traffic generation in this directory:

```
$ vim launchTraffic.py
```

- p. Copy the contents of launchTraffic.py and save the file.

- q. Repeat step e and f to create an attack traffic file in the same directory.

```
$ vim launchAttack.py
```

- r. Move to the directory pox/pox/forwarding by entering the following commands:

```
$ cd ../../pox/pox/forwarding
```

- s. Create a new python script detection.py, copy the contents of detection.py script into this file and save it.

```
$ vim detection.py
```

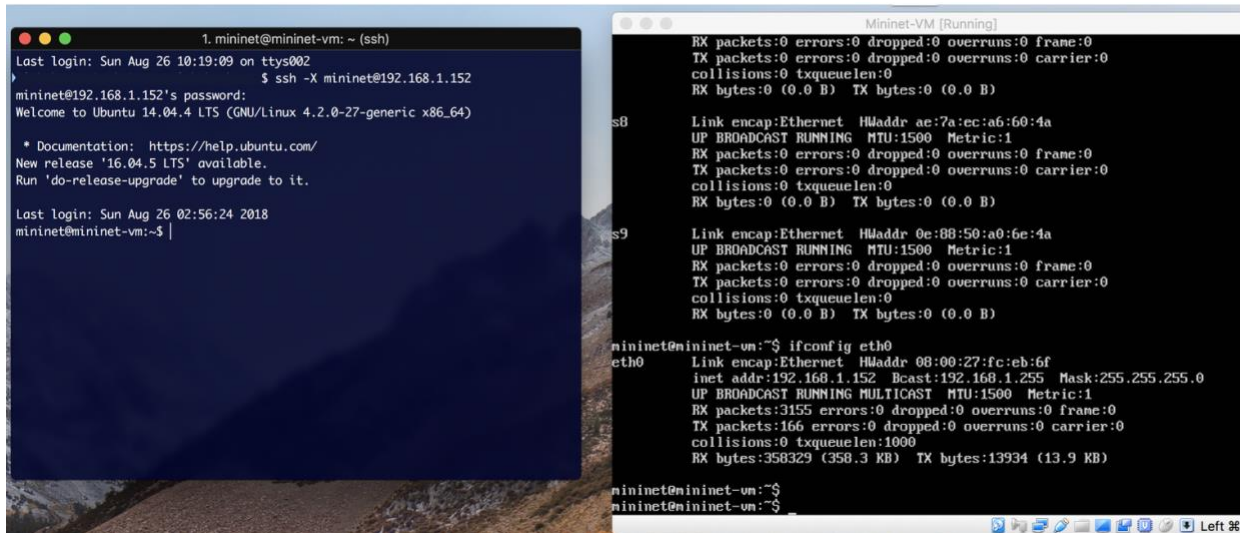
- t. Create a new python script in the same directory. Copy the contents of l3_editing.py to this file and save it.

```
$ vim l3_editing.py
```

3. Steps to perform the experiment

a. Find the Threshold for usual Traffic (normal)

- After the initial setup of the test environment you should see an ssh window of Mininet as well as Mininet running on the virtual box.



```
1. mininet@mininet-vm: ~ (ssh)
Last login: Sun Aug 26 10:19:09 on ttys002
$ ssh -X mininet@192.168.1.152
mininet@192.168.1.152's password:
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
New release '16.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Aug 26 02:56:24 2018
mininet@mininet-vm:~$

Mininet-VM [Running]
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:0
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

Link encap:Ethernet HWaddr ac:7a:ec:a6:60:4a
UP BROADCAST RUNNING 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:0
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

Link encap:Ethernet HWaddr 0e:8b:50:a0:6e:4a
UP BROADCAST RUNNING 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:0
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

mininet@mininet-vm:~$ ifconfig eth0
eth0      Link encap:Ethernet HWaddr 08:00:27:fc:eb:6f
          inet addr:192.168.1.152 Bcast:192.168.1.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:3155 errors:0 dropped:0 overruns:0 frame:0
          TX packets:166 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:358329 (358.3 KB) TX bytes:13934 (13.9 KB)

mininet@mininet-vm:~$
mininet@mininet-vm:~$
```

SSH window and virtual box window running mininet

- Move to the pox directory in the Virtual Box terminal running Mininet by entering the following command:

```
$ cd pox
```

- In the Mininet terminal of Virtual Box enter the following command to run the pox controller:

```
$ python ./pox.py forwarding.l3_editing
```

- Simultaneously, in the ssh window running Mininet, create a Mininet topology by entering the following command:

```
$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,ip=127.0.0.1,port=6633
```

This command creates a network and adds controllers, hosts, switches and links between them.

127.0.0.1 is the loopback address. You can find this address by entering the command:

```
$ ifconfig lo
```



```

1. mininet@mininet-vm: ~ (ssh)
mininet@mininet-vm:~$ sudo mm --switch ovsk --topo tree,depth=2,fanout=8 --controller-remote,ip=127.0.0.1,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h
23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h
43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h
63 h64
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (s1, s8) (s1, s9) (s2, h1)
(s2, h2) (s2, h3) (s2, h4) (s2, h5) (s2, h6) (s2, h7) (s2, h8) (s3, h9) (s3, h1
0) (s3, h11) (s3, h12) (s3, h13) (s3, h14) (s3, h15) (s3, h16) (s4, h17) (s4, h1
8) (s4, h19) (s4, h20) (s4, h21) (s4, h22) (s4, h23) (s4, h24) (s5, h25) (s5, h
26) (s5, h27) (s5, h28) (s5, h29) (s5, h30) (s5, h31) (s5, h32) (s6, h33) (s6, h
34) (s6, h35) (s6, h36) (s6, h37) (s6, h38) (s6, h39) (s6, h40) (s7, h41) (s7, h
42) (s7, h43) (s7, h44) (s7, h45) (s7, h46) (s7, h47) (s7, h48) (s8, h49) (s8, h
50) (s8, h51) (s8, h52) (s8, h53) (s8, h54) (s8, h55) (s8, h56) (s9, h57) (s9, h
58) (s9, h59) (s9, h60) (s9, h61) (s9, h62) (s9, h63) (s9, h64)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h
23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h
43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h
63 h64
*** Starting controller
c0
*** Starting 9 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 ...
*** Starting CLI:
mininet>

mininet@mininet-vm:~$ python ./pox.py ifconfig lo
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:87163 errors:0 dropped:0 overruns:0 frame:0
TX packets:87163 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:4369776 (4.3 MB) TX bytes:4369776 (4.3 MB)

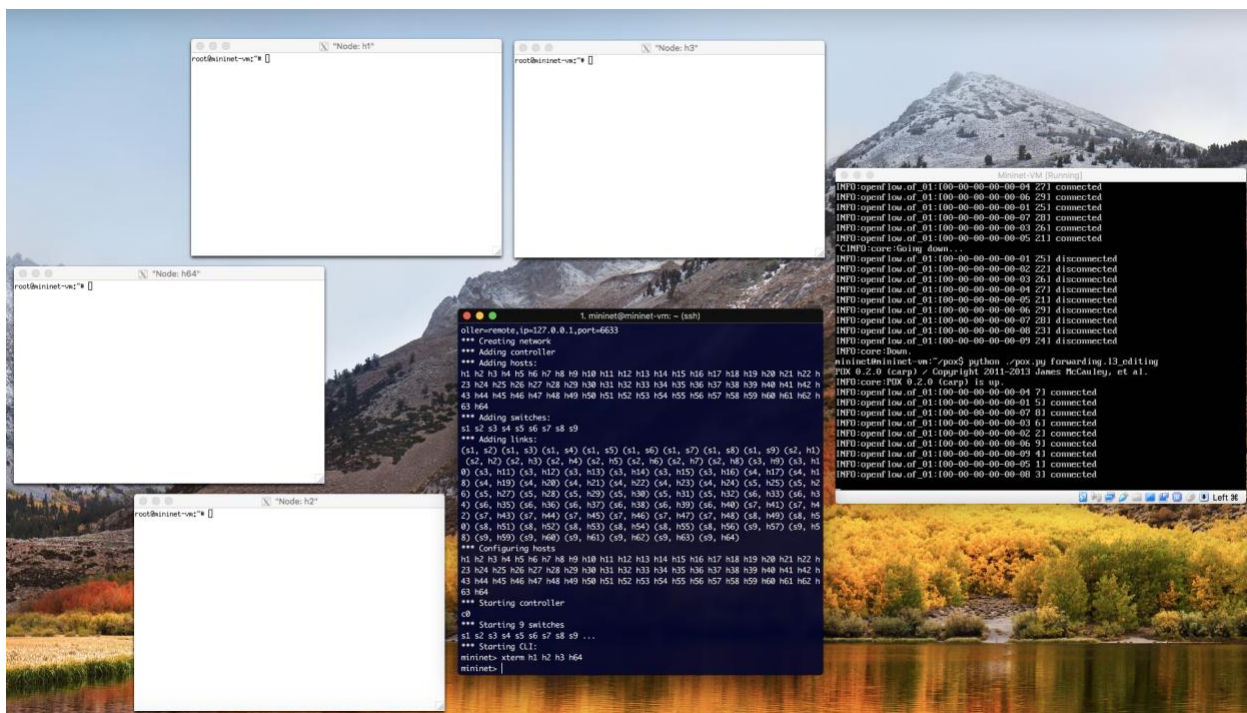
mininet@mininet-vm:~$ python ./pox.py forwarding, i3_editing
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-05 1] connected
INFO:openflow.of_01:[00-00-00-00-00-02 2] connected
INFO:openflow.of_01:[00-00-00-00-00-08 3] connected
INFO:openflow.of_01:[00-00-00-00-00-09 5] connected
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
INFO:openflow.of_01:[00-00-00-00-00-04 7] connected
INFO:openflow.of_01:[00-00-00-00-00-07 8] connected
INFO:openflow.of_01:[00-00-00-00-00-03 6] connected
INFO:openflow.of_01:[00-00-00-00-00-06 9] connected

```

Run Pox controller and create mininet topology

- In your ssh window, type the following command to open xterm windows:

mininet> xterm h1 h2 h3 h64



Open xterm windows after running above command

- In the xterm window of h1 move to the mininet/custom directory:

cd mininet/custom

- Then run the launchTraffic code using the following command:

python ./launchTraffic.py -s 2 -e 65

Here -s and -e indicate where the start and end parameters are.

2 and 65 are the arguments passed to the gendest function that generates the dst IP address. More info can be seen in the 'Additional details and Explanation' section of the document.

- Now you should be able to see the pox controller generating a list of values for entropy as shown below. The least value obtained is the threshold entropy for normal traffic. To avoid false positives and negatives due to loss of a switch we choose an entropy value of 1.00 instead of 1.14. This implies 10% fault tolerance.

```
2. mininet@mininet-vm: ~/pox (ssh)
***** Entropy Value = 1.16354027518 *****

***** Entropy Value = 1.16354027518 *****

INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.134050949081
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.207361874058
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.34141282314
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.414723748117
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.488034673094
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.543952273441
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.617263198418
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.690574123395
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.724553523481
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.797864448458
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.908362698933
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.964280299279
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.03759122426
INFO:forwarding.detection:{IPAddr('10.0.0.40'): 9, IPAddr('10.0.0.48'): 3, IPAddr('10.0.0.34'): 9, IPAddr('10.0.0.16'): 3, IPAddr('10.0.0.13'): 3, IPAddr('10.0.0.9'): 2, IPAddr('10.0.0.31'): 3, IPAddr('10.0.0.64'): 3, IPAddr('10.0.0.44'): 1, IPAddr('10.0.0.20'): 3, IPAddr('10.0.0.53'): 6, IPAddr('10.0.0.4'): 2, IPAddr('10.0.0.46'): 3}

***** Entropy Value = 1.03759122426 *****

***** Entropy Value = 1.03759122426 *****
```

Entropy values

- Entropy is calculated on the basis of a list of IP addresses that packets are being sent to. All the destination IP addresses have the same first three octets of 10.0.0. The fourth octet is randomized in the launchTraffic.py file. These IP addresses correspond to the 64 hosts in our mininet topology. For every 50 packets sent, a dictionary is maintained. The dictionary comprises the destination IP addresses and the

number of times a packet has been sent to that address. Finally, entropy is calculated as follows: $\text{Entropy} = -(\text{count}/50) * \log_{10}(\text{count}/50)$, Where 50 represents the number of packets counted in the dictionary.

b. Detection of DDOS Threat using the value of Entropy

- On the h2 and h3 xterm windows enter the following command to move to the mininet/custom directory:

```
$ cd mininet/custom
```

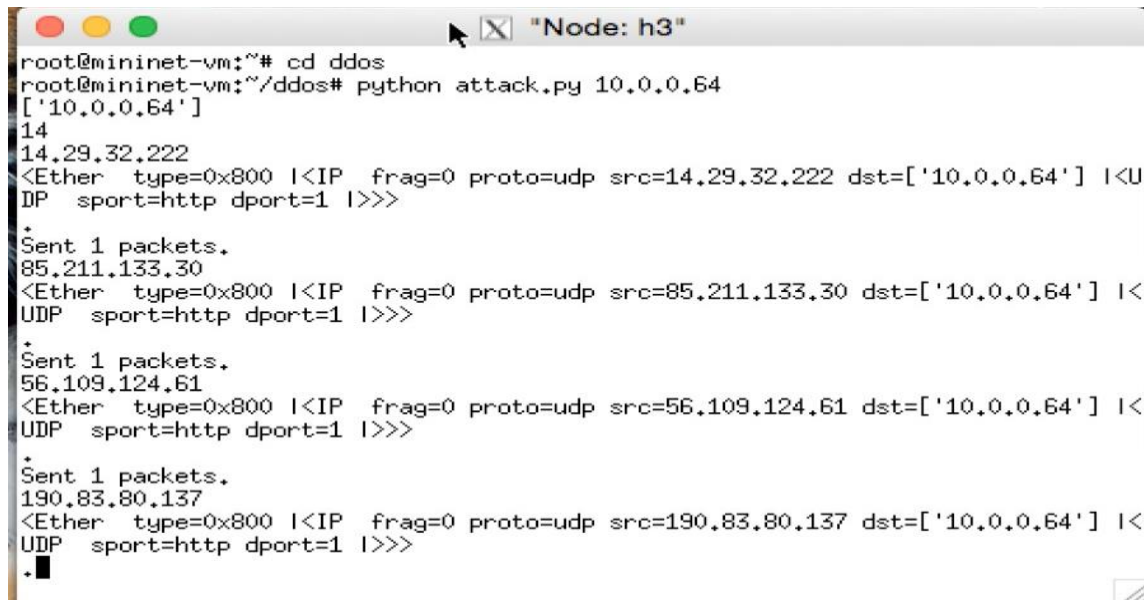
- Enter the following command to run launchTraffic.py in the h1 xterm window:

```
$ python ./launchTraffic.py -s 2 -e 65
```

- Simultaneously enter the following commands in the h2 and h3 xterm windows to run launchAttack.py:

```
$ python ./launchAttack.py 10.0.0.64
```

Here 10.0.0.64 is the destination IP address that attack packets are going to be sent to.



```
root@mininet-vm:~# cd ddos
root@mininet-vm:~/ddos# python attack.py 10.0.0.64
['10.0.0.64']
14
14.29.32.222
<Ether type=0x800 |<IP frag=0 proto=udp src=14.29.32.222 dst=['10.0.0.64'] |<U
UDP sport=http dport=1 |>>>
*
Sent 1 packets.
85.211.133.30
<Ether type=0x800 |<IP frag=0 proto=udp src=85.211.133.30 dst=['10.0.0.64'] |<
UDP sport=http dport=1 |>>>
*
Sent 1 packets.
56.109.124.61
<Ether type=0x800 |<IP frag=0 proto=udp src=56.109.124.61 dst=['10.0.0.64'] |<
UDP sport=http dport=1 |>>>
*
Sent 1 packets.
190.83.80.137
<Ether type=0x800 |<IP frag=0 proto=udp src=190.83.80.137 dst=['10.0.0.64'] |<
UDP sport=http dport=1 |>>>
.█
```

Launching the Attack

- Observe the entropy values in the pox controller. The value decreases below the threshold values for normal traffic as determined earlier. Thus, we can detect the attack within the first 250 packets of malicious traffic attacking a particular host in the SDN network.


```

2018-08-27 14:01:23.515356+04:00 : printing diction {1: {1: 2}, 9: {9: 1}}

INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.073310924977
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.146621849954
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.22414986036
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.297460785337
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.331440185424
INFO:forwarding.detection:{IPAddr('10.0.0.63'): 3, IPAddr('10.0.0.26'): 3, IPAddr('10.0.0.64'): 40, IPAddr('10.0.0.21'): 3, IPAddr('10.0.0.30'): 1}

***** Entropy Value = 0.331440185424 *****

2018-08-27 14:01:23.523457+04:00 : printing diction {1: {1: 2}, 9: {9: 2}}

***** Entropy Value = 0.331440185424 *****

```

Change in entropy value of controller

- The dictionary seen in the figure below denotes the switch ID (in our mininet topology), the port of the switch and the number of times that specific port has been activated. If the count for any port exceeds 50, the system raises a warning stating that a DDOS attack has been detected.

```

***** Entropy Value = 0.236932581971 *****

2018-08-27 14:03:40.569790+04:00 : printing diction {1: {1: 83}, 2: {1: 16, 3: 70}, 3: {9: 2}, 4: {9: 2}, 5: {9: 1}, 6: {9: 4}, 7: {9: 3}, 8: {9: 1}, 9: {9: 70}}

-----

2018-08-27 14:03:40.571726+04:00 ***** DDOS DETECTED *****

{1: {1: 83}, 2: {1: 16, 3: 70}, 3: {9: 2}, 4: {9: 2}, 5: {9: 1}, 6: {9: 4}, 7: {9: 3}, 8: {9: 1}, 9: {9: 70}}

2018-08-27 14:03:40.572630+04:00 : BLOCKED PORT NUMBER : 1 OF SWITCH ID: 1

-----

```

DDOS Attack detected

- On successful completion of the experiment, stop running the Mininet topology by entering the following command:

```
mininet> exit
```

4. Additional Details and Explanation

When you first run the pox controller you will notice that the controller doesn't seem to do or display anything on the screen.

If you run the command: `$ python ./pox.py -verbose forwarding.l3_editing`, you will notice the following being displayed.

```
mininet@mininet-vm:~/pox$ python ./pox.py -verbose forwarding.l3_editing
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Oct 26 2016 20:30:19)
DEBUG:core:Platform is Linux-4.2.0-27-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
```

Debug statements give us more information

Thus, we see that the controller is listening on the port 0.0.0.0:6633. However, it doesn't have any switches on that port to connect to.

Take a look at the following command that we entered during the testing of the environment:

```
$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,ip=127.0.0.1,port=6633
```

The above command sets up a Mininet tree topology. It has a depth of two which includes a core switch and its children. The fanout parameter declares that each switch has 8 children and 8 hosts connected to it. Thus, the core switch has 8 children to give us a total of 9 switches as seen in the image below. The core switch has no host connected to it, however, all 8 children have 8 hosts connected to them. Thus, we have a total of $(8 * 8 = 64)$ hosts in our topology.

```
mininet@mininet-vm:~$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,ip=127.0.0.1,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (s1, s8) (s1, s9) (s2, h1) (s2, h2) (s2, h3) (s2, h4) (s2, h5) (s2, h6) (s2, h7) (s2, h8) (s3, h9) (s3, h10) (s3, h11) (s3, h12) (s3, h13) (s3, h14) (s3, h15) (s3, h16) (s4, h17) (s4, h18) (s4, h19) (s4, h20) (s4, h21) (s4, h22) (s4, h23) (s4, h24) (s5, h25) (s5, h26) (s5, h27) (s5, h28) (s5, h29) (s5, h30) (s5, h31) (s5, h32) (s6, h33) (s6, h34) (s6, h35) (s6, h36) (s6, h37) (s6, h38) (s6, h39) (s6, h40) (s7, h41) (s7, h42) (s7, h43) (s7, h44) (s7, h45) (s7, h46) (s7, h47) (s7, h48) (s8, h49) (s8, h50) (s8, h51) (s8, h52) (s8, h53) (s8, h54) (s8, h55) (s8, h56) (s9, h57) (s9, h58) (s9, h59) (s9, h60) (s9, h61) (s9, h62) (s9, h63) (s9, h64)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Starting controller
c0
*** Starting 9 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 ...
*** Starting CLI:
```

Mininet tree topology set up

Each of the 9 switches are created on port 6633 and have an IP address that matches the loopback IP address specified in the statement just entered.

The hosts, however, have their own private IP addresses that begin with 10.0.0.1 and end at 10.0.0.64.

These values may appear familiar as they correspond with the argument we pass to the launchAttack.py file. (More on this later)

Now that the pox controller has connected to the 9 switches just created, we run the launchTraffic.py code.

The command we enter to run the code is as follows:

```
$ python ./launchTraffic.py -s 2 -e 65
```

This command runs the launchTraffic.py code with the given arguments, 2 and 65.

The launchTraffic.py code actually tries to simulate regular traffic using the host IP addresses.

The sourceIPgen() function generates random IP addresses with values in each of the 4 octets being randomized.

The gendest() function generates random IP addresses that correspond to the host IP addresses in our created topology. Thus, the first three octets are 10.0.0 while the fourth octet is randomized between the 2 arguments passed to the code. Thus, we generate destination IP addresses ranging from 10.0.0.2 to 10.0.0.64 using the randrange() function.

```
*
Sent 1 packets.
<Ether type=0x800 |<IP frag=0 proto=udp src=77.194.46.66 dst=10.0.0.4 |<UDP s
port=2 dport=http |>>>
+^C
Sent 1 packets.
<Ether type=0x800 |<IP frag=0 proto=udp src=212.47.48.154 dst=10.0.0.12 |<UDP
sport=2 dport=http |>>>
*
Sent 1 packets.
<Ether type=0x800 |<IP frag=0 proto=udp src=71.73.65.221 dst=10.0.0.27 |<UDP
sport=2 dport=http |>>>
*
Sent 1 packets.
<Ether type=0x800 |<IP frag=0 proto=udp src=100.14.102.182 dst=10.0.0.33 |<UDP
sport=2 dport=http |>>>
```

[Packets being sent from random source to random destination under normal traffic](#)

The launchTraffic.py code send network packets from random source IP addresses to random destination IP addresses that correspond to our network hosts. This simulates regular network traffic.

While launchTraffic.py is running, we see a list of entropy values being generated in our pox controller. After every 50 statements where the entropy value is logged to the display, we see Entropy values increasing and then a list of IP addresses being displayed as in the image below.

```

***** Entropy Value = 1.09643920189 *****

***** Entropy Value = 1.09643920189 *****

***** Entropy Value = 1.09643920189 *****
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.073310924977
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.207361874058
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.280672799035
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.353983724012
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.427294648989
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.500605573966
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.611103824441
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.645083224527
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.718394149504
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.791705074481
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.865015999458
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.898995399545
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.972306324522
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.0456172495
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.11892817448
INFO:forwarding.detection:{IPAddr('10.0.0.63'): 3, IPAddr('10.0.0.10'): 9, IPAddr('10.0.0.30'): 3, IPAddr('10.0.0.64'): 3, IPAddr('10.0.0.29'): 3, IPAddr('10.0.0.33'): 3, IPAddr('10.0.0.37'): 6, IPAddr('10.0.0.43'): 1, IPAddr('10.0.0.14'): 3, IPAddr('10.0.0.53'): 3, IPAddr('10.0.0.24'): 3, IPAddr('10.0.0.7'): 1, IPAddr('10.0.0.47'): 3, IPAddr('10.0.0.55'): 3, IPAddr('10.0.0.34'): 3}

***** Entropy Value = 1.11892817448 *****

***** Entropy Value = 1.11892817448 *****

```

Entropy values and IP List

Take a look at the dictionary being displayed in the image above. It displays a number of host IP addresses along with a number associated with it.

This dictionary maps the host IP addresses the number of times a network packet has been sent to them.

Entropy is calculated on these count values.

The entropy function in the detection.py file calculates entropy as follows:

Entropy = $-(\text{count}/50) * \log_{10}(\text{count}/50)$

The 50 here corresponds with the number of packets that have been sent. This is an arbitrary interval taken by the programmer. Thus, the value of entropy finally displayed after the IP list is created is the sum of the entropy values calculated on each IP address in the dictionary.


```
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.073310924977
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.207361874058
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.280672799035
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.353983724012
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.427294648989
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.500605573966
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.611103824441
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.645083224527
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.718394149504
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.791705074481
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.865015999458
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.898995399545
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.972306324522
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.0456172495
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.11892817448
```

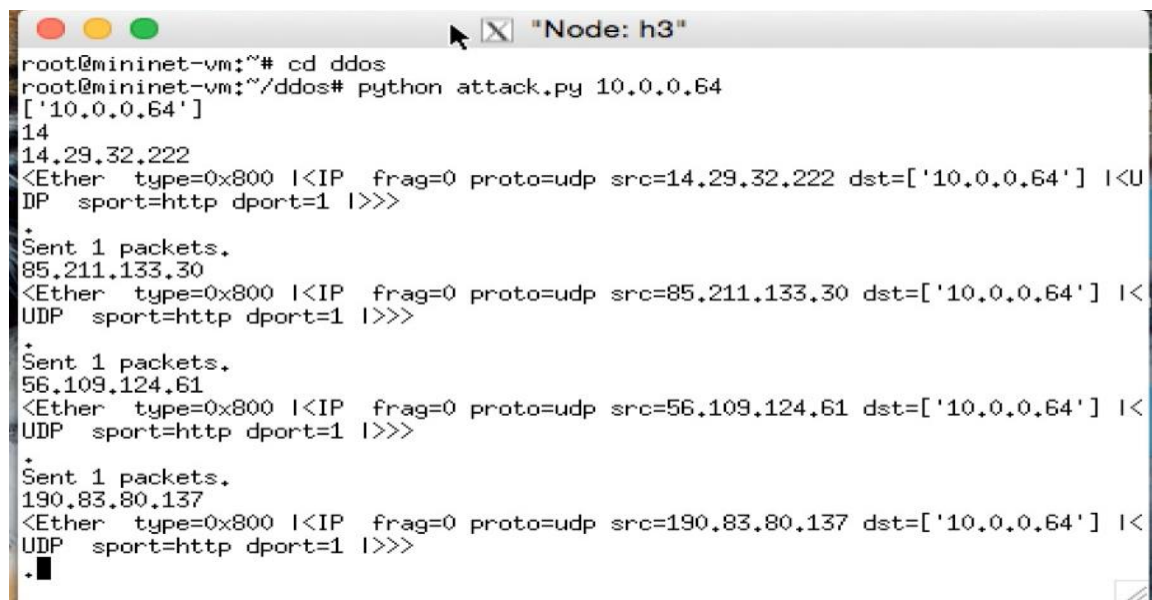
Entropy values for each IP address are added to each other until we get the final value at the bottom of the image.

We see that under normal traffic, the entropy value doesn't go below 1. This is the threshold value for entropy under normal traffic.

Now let's take a look at the statement used to run launchAttack.py.

```
$ python ./launchAttack.py 10.0.0.64
```

The argument being passed to LaunchAttack.py matches the IP address of the 64th host. LaunchAttack.py sends network packets from random source IP addresses to a single host. This host is determined by the argument passed. Thus, attack packets are sent specifically to host 64.



```
root@mininet-vm:~# cd ddos
root@mininet-vm:~/ddos# python attack.py 10.0.0.64
['10.0.0.64']
14
14.29.32.222
<Ether type=0x800 |<IP frag=0 proto=udp src=14.29.32.222 dst=['10.0.0.64'] |<U
DP sport=http dport=1 |>>>
*
Sent 1 packets.
85.211.133.30
<Ether type=0x800 |<IP frag=0 proto=udp src=85.211.133.30 dst=['10.0.0.64'] |<
UDP sport=http dport=1 |>>>
*
Sent 1 packets.
56.109.124.61
<Ether type=0x800 |<IP frag=0 proto=udp src=56.109.124.61 dst=['10.0.0.64'] |<
UDP sport=http dport=1 |>>>
*
Sent 1 packets.
190.83.80.137
<Ether type=0x800 |<IP frag=0 proto=udp src=190.83.80.137 dst=['10.0.0.64'] |<
UDP sport=http dport=1 |>>>
*
■
```

Attack packets are only sent to host 64

When launchAttack.py is run, we see that the IP list dictionary generated has dramatic changes in its count values. Host 10.0.0.64 now has extremely high values of count as attack packets are being sent to it. Thus, every 50 packets, host 64 has received several packets whereas the other hosts have received far fewer.

```
2018-08-27 14:01:23.515356+04:00 : printing diction {1: {1: 2}, 9: {9: 1}}

INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.073310924977
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.146621849954
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.22414986036
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.297460785337
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.331440185424
INFO:forwarding.detection:{IPAddr('10.0.0.63'): 3, IPAddr('10.0.0.26'): 3, IPAddr('10.0.0.64'): 40, IPAddr('10.0.0.21'): 3, IPAddr('10.0.0.30'): 1}

***** Entropy Value = 0.331440185424 *****

2018-08-27 14:01:23.523457+04:00 : printing diction {1: {1: 2}, 9: {9: 2}}

***** Entropy Value = 0.331440185424 *****
```

Host 10.0.0.64 has a count of 40 whereas the other hosts have far fewer.

When one host receives many more packets than the other host, the value of entropy tends to decrease.

Thus, we see the entropy fall below the earlier threshold of 1.

The decrease in entropy value serves as a signal that network traffic is irregular.

Take a look at the image above and you will see another dictionary being displayed. This dictionary is only created when the entropy values decreases below 0.5.

This is actually a dictionary that contains dictionary values within it.

The key values correspond to the switches in our mininet topology.

The value dictionaries have keys that correspond to the port in the relevant switch. The final value shows us the number of times those ports have been activated.

This serves as another good measure for irregular network traffic and helps us detect a DDOS attack.

When the count for any port of any switch exceeds 50, a warning is displayed in the pox controller.

```
2018-08-29 11:05:18.152062+04:00 ***** DDOS DETECTED *****
{1: {1: 79}, 2: {1: 12, 3: 67}, 3: {9: 1}, 4: {9: 1}, 5: {9: 1}, 6: {9: 2}, 7: {9: 2}, 8: {9: 1}, 9: {9: 71}}
2018-08-29 11:05:18.153069+04:00 : BLOCKED PORT NUMBER : 1 OF SWITCH ID: 1
-----
2018-08-29 11:05:18.154537+04:00 ***** DDOS DETECTED *****
{1: {1: 79}, 2: {1: 12, 3: 67}, 3: {9: 1}, 4: {9: 1}, 5: {9: 1}, 6: {9: 2}, 7: {9: 2}, 8: {9: 1}, 9: {9: 71}}
2018-08-29 11:05:18.155381+04:00 : BLOCKED PORT NUMBER : 3 OF SWITCH ID: 2
-----
2018-08-29 11:05:18.156312+04:00 ***** DDOS DETECTED *****
{1: {1: 79}, 2: {1: 12, 3: 67}, 3: {9: 1}, 4: {9: 1}, 5: {9: 1}, 6: {9: 2}, 7: {9: 2}, 8: {9: 1}, 9: {9: 71}}
2018-08-29 11:05:18.157228+04:00 : BLOCKED PORT NUMBER : 9 OF SWITCH ID: 9
```

DDOS attack detected

Take a look at the image above. We see that port number 1 of switch 1 has been blocked to prevent the attack from causing more damage. If you look at the dictionary you will notice that port 1 of switch 1 has been activated 79 times.

Similar port 3 of switch 2 has been activated 67 times while port 9 of switch 9 has been activated 71 times.

These are direct indicators of an attack taking place and the program blocks these ports once it realizes that the network is under attack.