Early detection of DDoS attack against SDN controllers

Software-Defined Networks and Network Function Virtualization

Arushi Mohania (IMT2016117)

Neha Krishna Dasari (IMT2016088)

GitHub link: https://github.com/aMohania/Early-DDoS-Detection-and-Prevention.git

OVERVIEW

- What is SDN?
- DoS and DDoS Attacks
- DDoS in SDN environment
    - Effect of DDoS attack in the SDN environment
    - Central controller vulnerabilities in the SDN environment
    - Importance of mitigating the DDoS attack
- Widely used present-day approaches and their limitations
- Concept of Entropy
    - Why it offers better results especially in a low resource environment
- Software requirements for the project
- Installation instructions
    - Problems that one may face while installing
- Procedure to run the code along with screenshots
    - Problems one may face while running the code
- Relevant code snippets needed to understand the functionality code offers
- References

What is SDN?

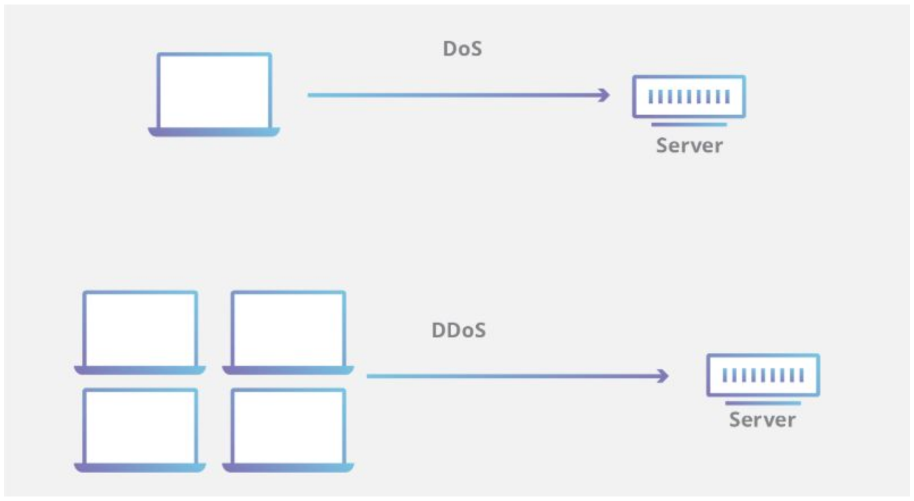
SDN is a developing network architecture that has gained immense popularity over the years. The driving force of SDN lays in the drawbacks in traditional networking approaches and the pressing need for dynamic and programmatically efficient network configuration. The control and forwarding planes are decoupled and this centralised approach raises issues related to security, scalability and elasticity.

DoS and DDoS Attacks

A simple analogy of Distributed Denial-of-Service(DDoS) attack can be a road that is clogged up with traffic jam preventing the regular traffic from reaching its destination.

A common way in which these attacks are executed is the attacker attacking one of the devices in a network thereby using it as their bot in order to send requests to a target whose IP address is known. A lot of requests are sent and the goal is to create an overflow at the target that it begins “denying to service other requests”.

In DDoS attacks, the attack traffic comes from a large number of sources which makes dealing with them even more difficult.



SDN Centralisation: Benefits and Created Vulnerabilities:

The SDN Controller lies at the heart of the SDN architecture. It is logically positioned between Network Elements (NEs) and SDN applications.

SDN’s centralized management approach enables events within the entire network to be collected and aggregated, The resulting broader, more coherent and more accurate image of the network’s status, makes security strategies both easier to enforce and to monitor.

However, a major disadvantage lays in the centralised nature of SDN. If the SDN controller is compromised, it means that the hacker has total control over the network. The SDN controller is a single point of failure and this added a new set of security threats that needed to be taken care of.

DDoS in an SDN environment:

Early detection of DDoS attacks in traditional network architectures is also tough in itself. In an SDN the switches that are responsible for forwarding and this is done by simply looking at their forwarding tables. If a match doesn’t exist they forward the packet to the controller to process it. Now, if there is an overflow at the controller the connection between that and switches is compromised and now there is no connection between the forwarding and processing planes thus exploiting the single point of failure aspect.

Current Approaches:

Most approaches deal with this problem with the help of Machine Learning. Since there is a sudden increase in the number of requests by a spoofed node or a bot, an ML algorithm takes in various parameters that include normal node behaviour and normal network behaviour with respect to traffic rate, and other parameters.

In case of a sudden burst of requests, the idea is that the ML algorithm in place would classify this as an attack and alert the network. There are various variations of this approach addressing problems such as more than one node(usually many) being used as bots and with more suitable parameters. Also given that SDN is a relatively new concept, most of the approaches of traditional networks and attack detection are employed.

Simple mitigation measures include:

- Black Hole Routing (Dropping all requests, both malicious and normal ones)
- Setting a limit on the traffic rate from nodes

When it comes to SDN specific situations, there have been proposals to have another controller over the existing controllers that keeps monitoring these controllers and that can act once anything goes wrong. Variations of this approach have been proposed.

Entropy:

Entropy, as we know is a measure of randomness. The higher the randomness of a network, the higher is its entropy and vice versa. Entropy would be a good metric to help in faster detection of DDoS attacks given that the randomness of the incoming traffic would go down in the event of an attack. If the entropy of the network is less than a certain threshold value quite a few times, we can say that the network might be under attack and take the required measures.

Procedure:

The implementation can be broadly classified into two phases:

1. Packet generation and Initial measure
2. Attack and Detection

In the first phase, we generate packets from one of the hosts which send packets from randomly generated IP addresses to random destinations. An initial measure of entropy is recorded.

In the attack phase, the attack is run from a few selected hosts to a specific target. Due to this, the entropy will come down.

For detection purposes, every 50 packets are considered a set and after every 50 packets, the entropy is calculated. If this calculated entropy is less than the preset threshold, the count is incremented. If this behaviour is observed 5 times, then we can say there is a high chance of DDoS attack has occurred. So, essentially we are looking at 250 packets and this sort of early detection secures the network in a much better way.

Entropy Calculation:

A crucial step is the calculation of entropy. Considering a window size of $n = 50$(as mentioned previously), let,

$Pi =$ The probability that a particular destination IP, i being a destination IP address for one of the 50 packets.

$\Rightarrow Pi = xi/n$

Now, Entropy(H) is given by,

$$H = -\sum_{i=0}^{n}(Pi)log(Pi)$$

We are implementing a research paper through this project. It very well describes the reason behind doing even a small task. The research paper's name is: Early detection of DDoS attack in SDN controllers by Seyed Mohammad Mousavi; Marc St-Hilaire (https://ieeexplore.ieee.org/document/7069319/authors#authors)

Software requirements for the project

- The network emulator used in creating the Network topology is **Mininet**.
- The project is based on Entropy detection before and after the attack and to facilitate that an **Openflow controller** is needed. The research paper we are implementing has used **POX** as it is fast, lightweight and designed as a platform so a custom controller can be built on top of it. Our motivation behind using the POX also involves the factor of our familiarity with Python.
- **Scapy** is used for generating traffic/packets. The UDP packet generation and spoofing of the source IP address will be facilitated by Scapy.

Installation instructions

Instructions to install Mininet:

```
sudo apt-get install mininet
sudo mn -c
sudo apt-get install git
git clone git://github.com/mininet/mininet
mininet/util/install.sh -a
```

The install.sh will automatically install POX and Wireshark while installing other utilities of Mininet. Therefore, no need to install POX again.

It will be good if the Python version used is 2.7 and 3.X. Some functionalities of POX may not support the 3.X version.

You can change your default Python version by the following commands:

```
alias python=python2.7
```

Instructions to install Scapy:

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

Procedure to run the code

There are three code files that we have written and one we have modified as per the needs of the project.

The codes we have written:

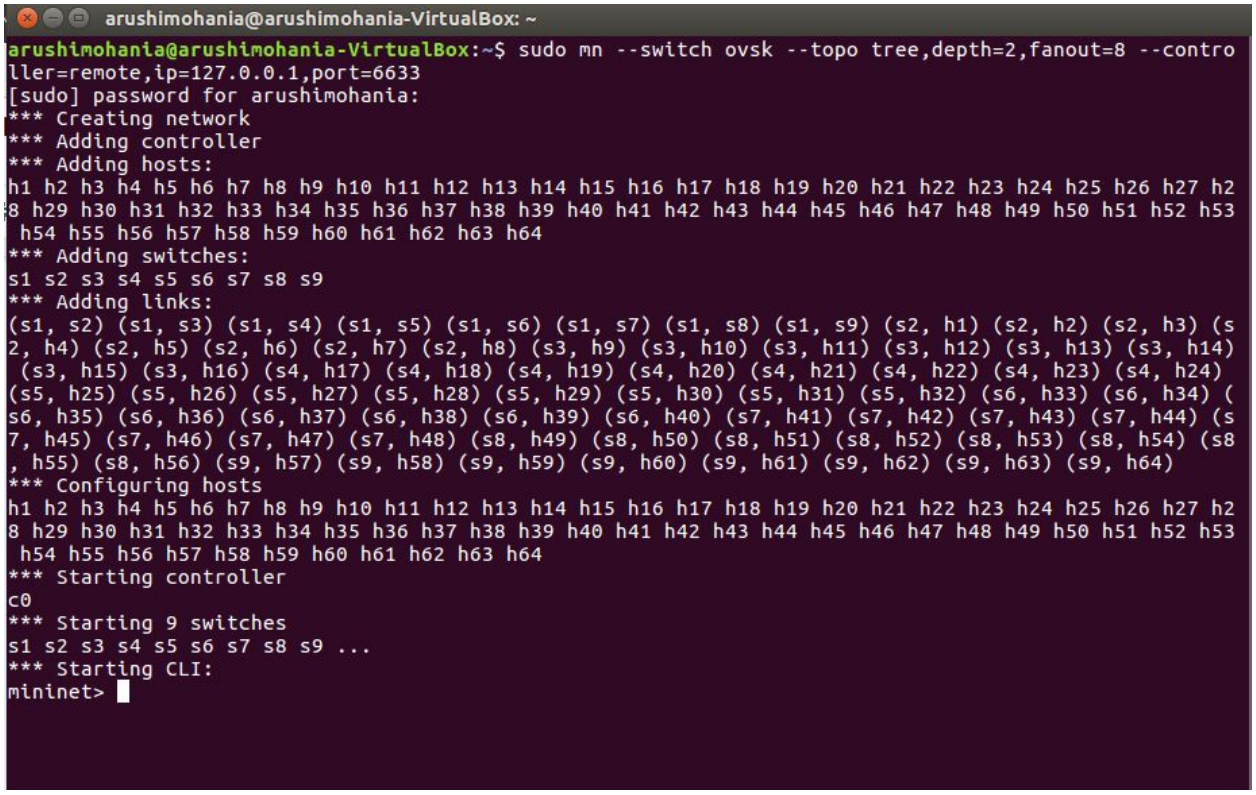
- trafficGen.py
  - It should be placed in mininet/custom
- attackLauncher.py
  - It should be placed in mininet/custom
- detectAttack.py
  - It should be placed in pox/pox/forwarding

The code we have modified:

- l3_modified.py
  - It is present in pox/pox/forwarding as l3_learning.py, you can either edit the code within it or simply make an l3_modified.py file

Before starting we need a network topology, in the research paper we are implementing, 9 switches and 64 hosts are there in the topology. To create a topology with these parameters with the help of mininet, the following command should be run on the terminal:

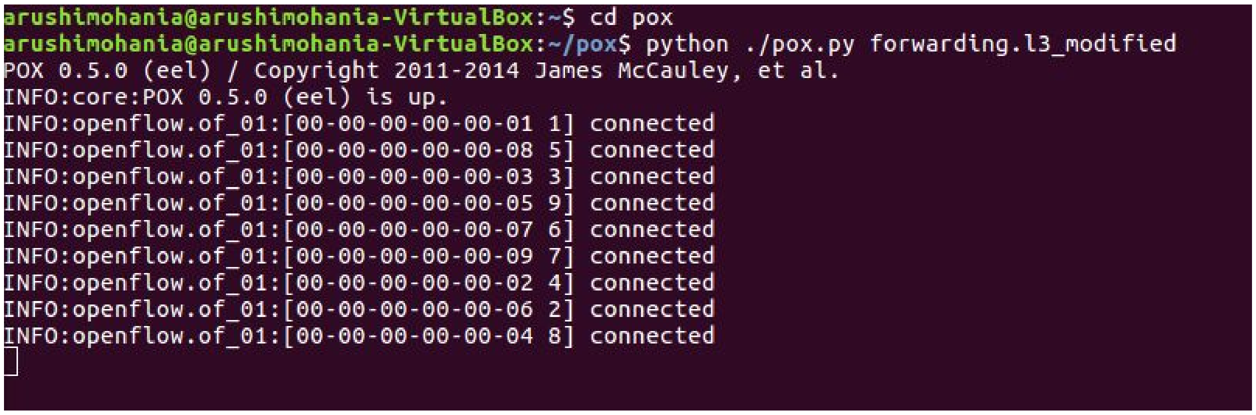
```
sudo mn --switch ovsk --topo tree,depth=2,fanout=8
-controller=remote,ip=127.0.0.1,port=6633
```



```
arushimohania@arushimohania-VirtualBox: ~
arushimohania@arushimohania-VirtualBox:~$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --contro
ller=remote,ip=127.0.0.1,port=6633
[sudo] password for arushimohania:
*** 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 h2
8 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) (s
2, 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) (s
7, 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 h2
8 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>
```

Open a new terminal and run the following commands:

```
cd pox
python ./pox.py forwarding.l3_modified
```



```
arushimohania@arushimohania-VirtualBox:~$ cd pox
arushimohania@arushimohania-VirtualBox:~/pox$ python ./pox.py forwarding.l3_modified
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:openflow.of_01:[00-00-00-00-00-08 5] connected
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:openflow.of_01:[00-00-00-00-00-05 9] connected
INFO:openflow.of_01:[00-00-00-00-00-07 6] connected
INFO:openflow.of_01:[00-00-00-00-00-09 7] connected
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
INFO:openflow.of_01:[00-00-00-00-00-06 2] connected
INFO:openflow.of_01:[00-00-00-00-00-04 8] connected
```

As discussed earlier in the report, the project is divided into two major parts, which are:

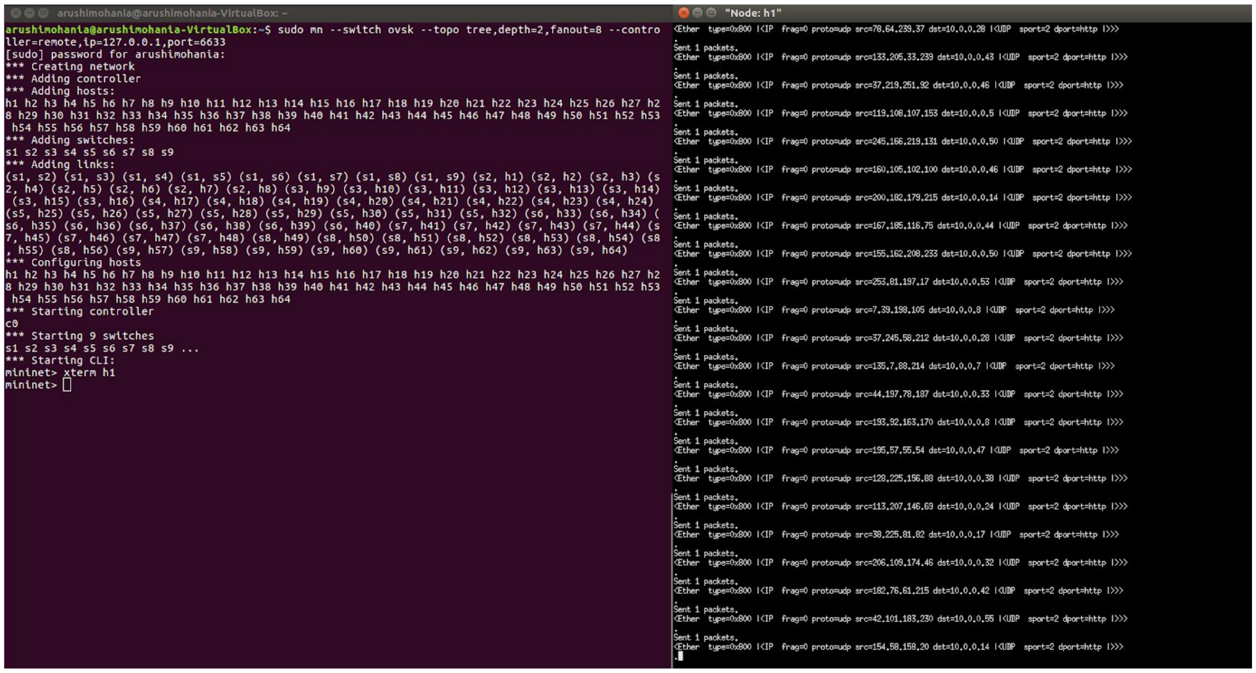
- Packet generation and detection
- Attack, detection and prevention

Therefore, to generate traffic, trafficGen.py should be run. The terminal in which mininet is running, run:

```
xterm h1
```

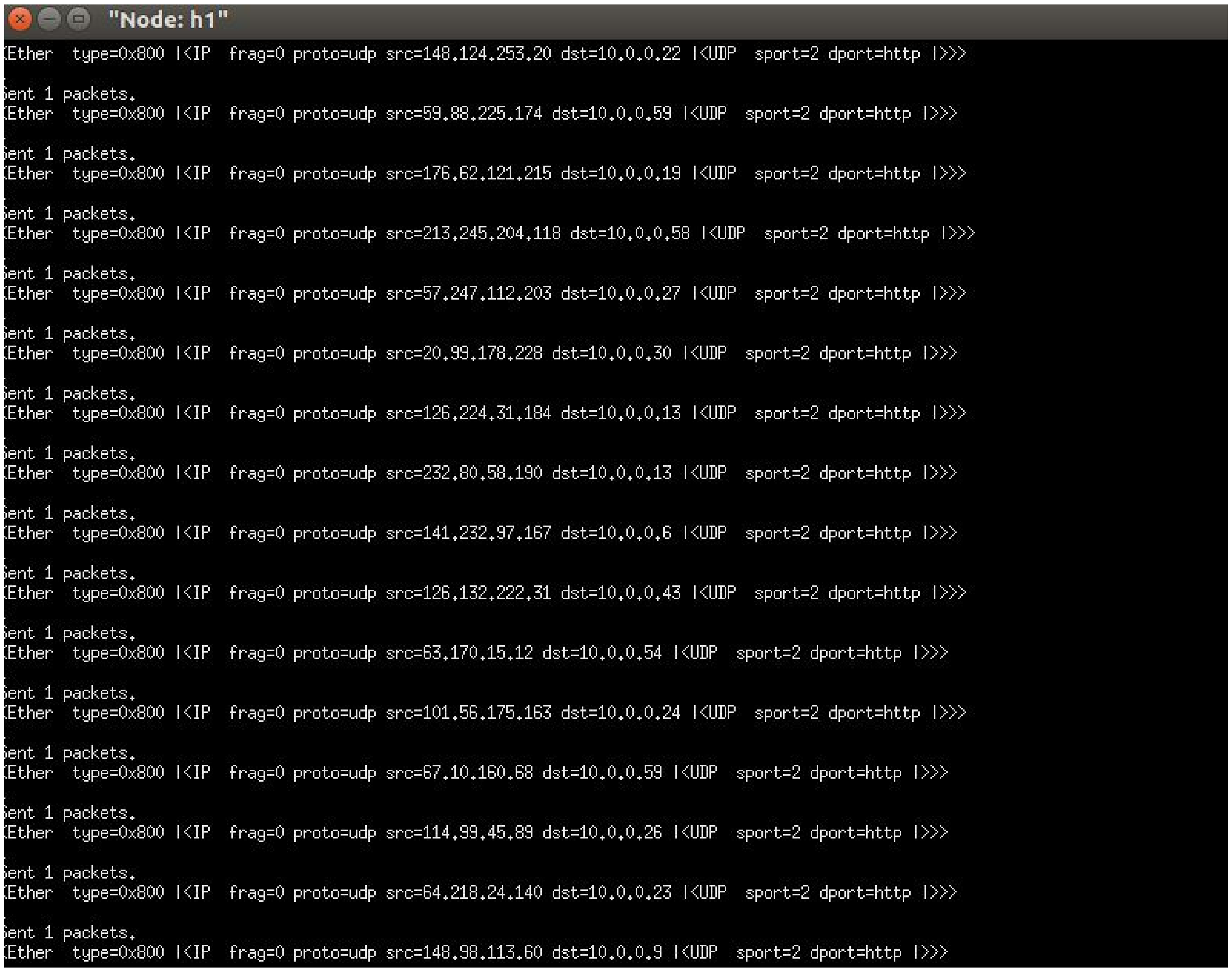
The xterm window will open for node h1, in that run:

```
cd mininet/custom
python trafficLaunch.py -s 2 -e 60
```



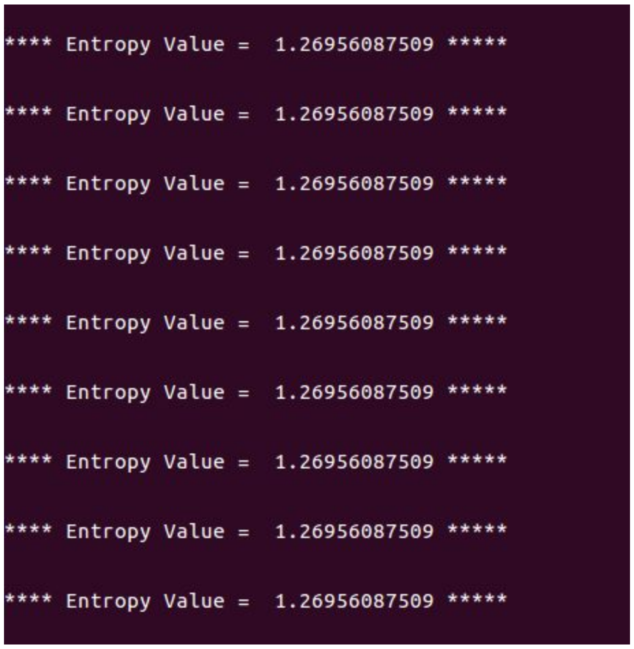
```
arushimohania@arushimohania-VirtualBox:~$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --contro
ller=remote,ip=127.0.0.1,port=6633
[sudo] password for arushimohania:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h2
8 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) (s
2, 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) (s
7, 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 h2
8 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> xterm h1
mininet> 
```

```
<Ether  type=0x800 |<IP  frag=0 proto=udp src=78.64.239.37 dst=10.0.0.28 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=133.205.33.239 dst=10.0.0.43 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=37.219.251.92 dst=10.0.0.46 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=119.108.107.153 dst=10.0.0.5 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=245.166.219.131 dst=10.0.0.50 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=160.105.102.100 dst=10.0.0.46 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=200.182.179.215 dst=10.0.0.14 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=167.185.116.75 dst=10.0.0.44 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=155.162.208.233 dst=10.0.0.50 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=253.81.197.17 dst=10.0.0.53 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=7.39.198.105 dst=10.0.0.8 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=37.245.58.212 dst=10.0.0.28 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=135.7.88.214 dst=10.0.0.7 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=44.197.78.187 dst=10.0.0.33 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=193.92.163.170 dst=10.0.0.8 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=195.57.55.54 dst=10.0.0.47 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=128.225.156.88 dst=10.0.0.38 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=113.207.146.69 dst=10.0.0.24 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=38.225.81.82 dst=10.0.0.17 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=206.109.174.46 dst=10.0.0.32 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=182.76.61.215 dst=10.0.0.42 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=42.101.183.230 dst=10.0.0.55 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=154.58.158.20 dst=10.0.0.14 |<UDP  sport=2 dport=http |>>>
.
```



```
Ether  type=0x800 |<IP  frag=0 proto=udp src=148.124.253.20 dst=10.0.0.22 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=59.88.225.174 dst=10.0.0.59 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=176.62.121.215 dst=10.0.0.19 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=213.245.204.118 dst=10.0.0.58 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=57.247.112.203 dst=10.0.0.27 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=20.99.178.228 dst=10.0.0.30 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=126.224.31.184 dst=10.0.0.13 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=232.80.58.190 dst=10.0.0.13 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=141.232.97.167 dst=10.0.0.6 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=126.132.222.31 dst=10.0.0.43 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=63.170.15.12 dst=10.0.0.54 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=101.56.175.163 dst=10.0.0.24 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=67.10.160.68 dst=10.0.0.59 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=114.99.45.89 dst=10.0.0.26 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=64.218.24.140 dst=10.0.0.23 |<UDP  sport=2 dport=http |>>>

ent 1 packets.
Ether  type=0x800 |<IP  frag=0 proto=udp src=148.98.113.60 dst=10.0.0.9 |<UDP  sport=2 dport=http |>>>
```

Entropy after traffic generation:



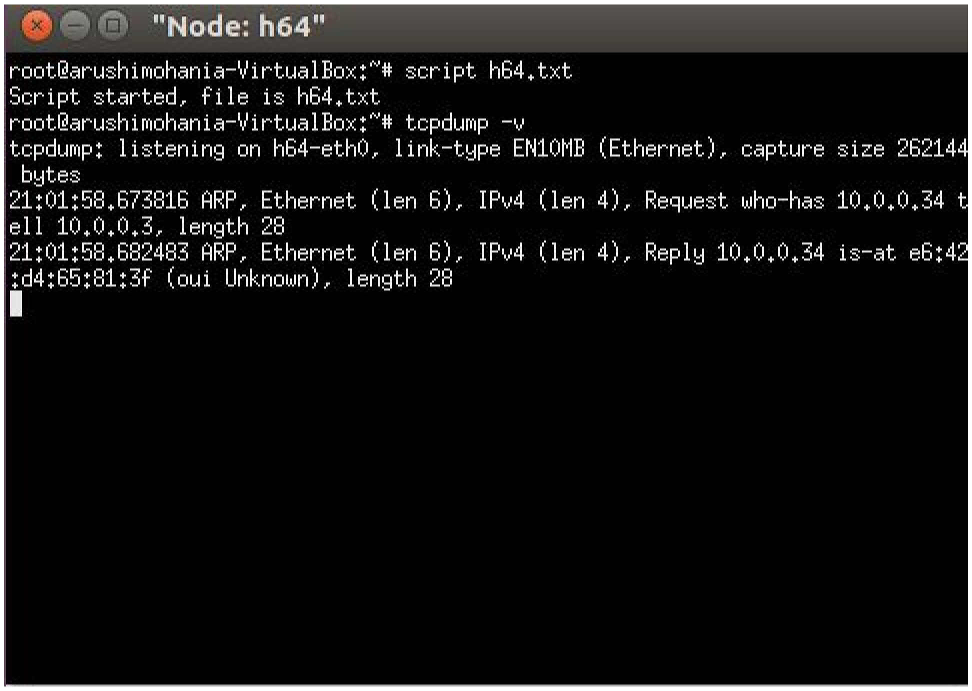
```
**** Entropy Value =  1.26956087509 *****

**** Entropy Value =  1.26956087509 *****

**** Entropy Value =  1.26956087509 *****

**** Entropy Value =  1.26956087509 *****

**** Entropy Value =  1.26956087509 *****

**** Entropy Value =  1.26956087509 *****

**** Entropy Value =  1.26956087509 *****

**** Entropy Value =  1.26956087509 *****

**** Entropy Value =  1.26956087509 *****
```

Now, again open the terminal in which mininet is running, this time run:

```
xterm h64
```

In the xterm window of the node h64, run:

```
script h64.txt
tcpdump -v
```



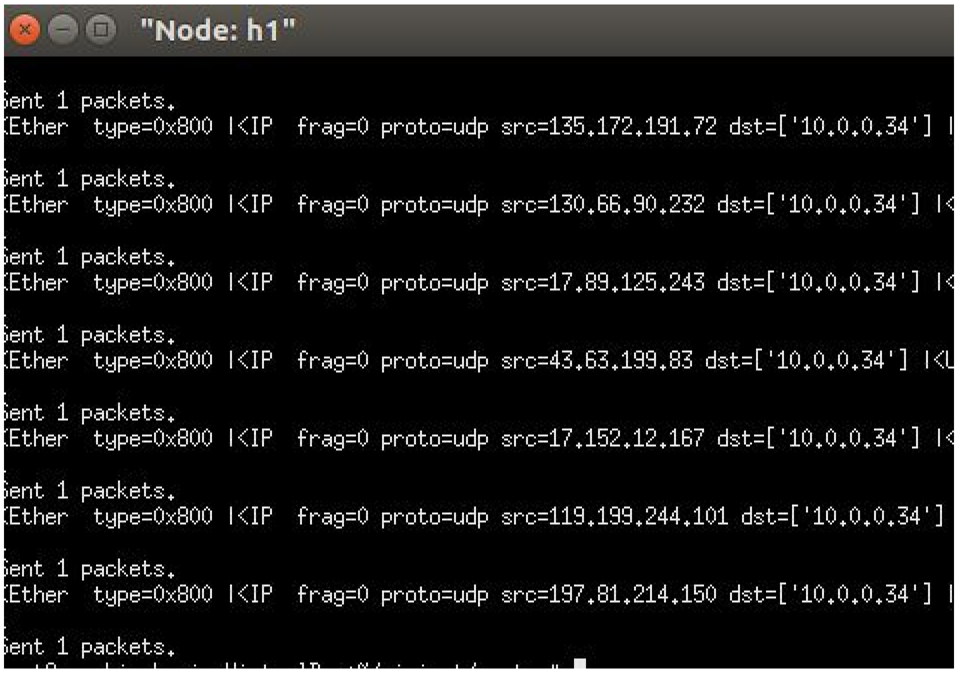
```
root@arushimohania-VirtualBox:~# script h64.txt
Script started, file is h64.txt
root@arushimohania-VirtualBox:~# tcpdump -v
tcpdump: listening on h64-eth0, link-type EN10MB (Ethernet), capture size 262144
 bytes
21:01:58.673816 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.0.34 t
ell 10.0.0.3, length 28
21:01:58.682483 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.0.0.34 is-at e6:42
:d4:65:81:3f (oui Unknown), length 28
```

Now, it time to attack, we will be attacking node h64 using node h1, h2 and h3. Therefore, on the terminal with mininet running, run:

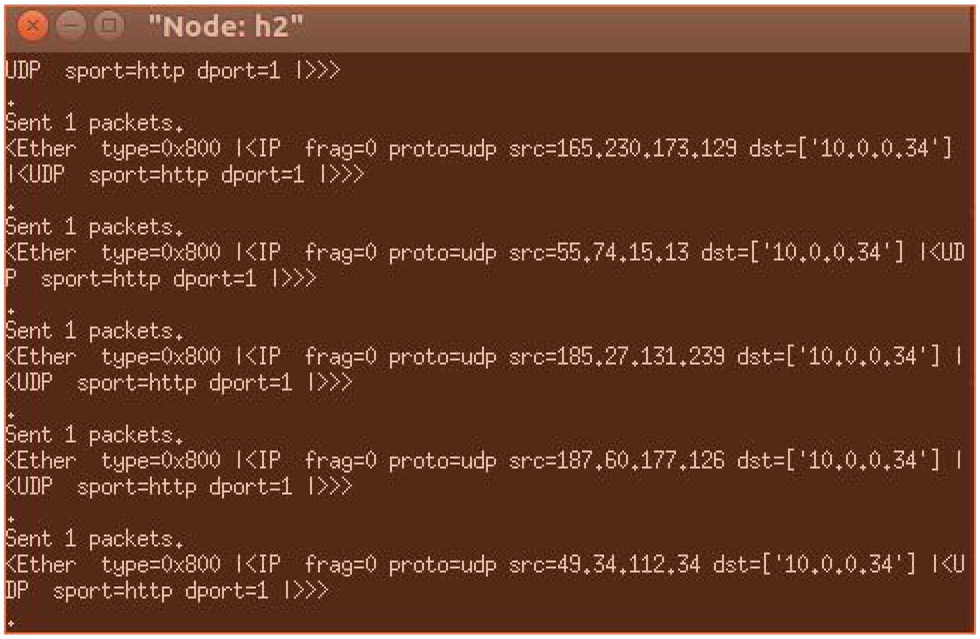
```
xterm h2 h3
```

Now, on the xterm windows of h1, h2 and h3, run:

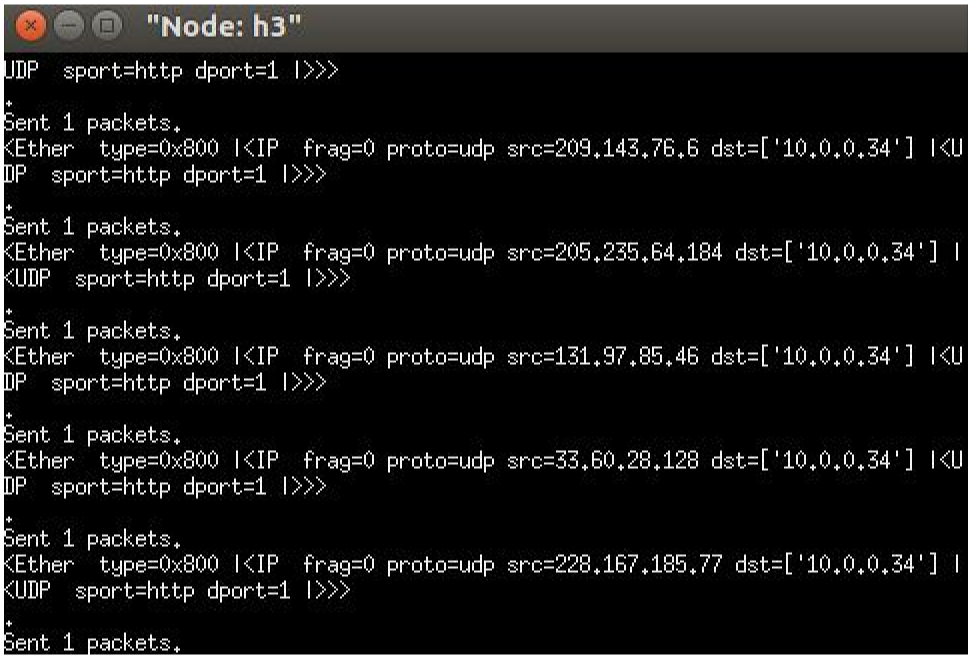
```
cd mininet/custom
python attackLauncher.py 10.0.0.34
```



```
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=135.172.191.72 dst=['10.0.0.34'] |
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=130.66.90.232 dst=['10.0.0.34'] |<
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=17.89.125.243 dst=['10.0.0.34'] |<
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=43.63.199.83 dst=['10.0.0.34'] |<U
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=17.152.12.167 dst=['10.0.0.34'] |<
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=119.199.244.101 dst=['10.0.0.34']
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=197.81.214.150 dst=['10.0.0.34'] |
.
Sent 1 packets.
```

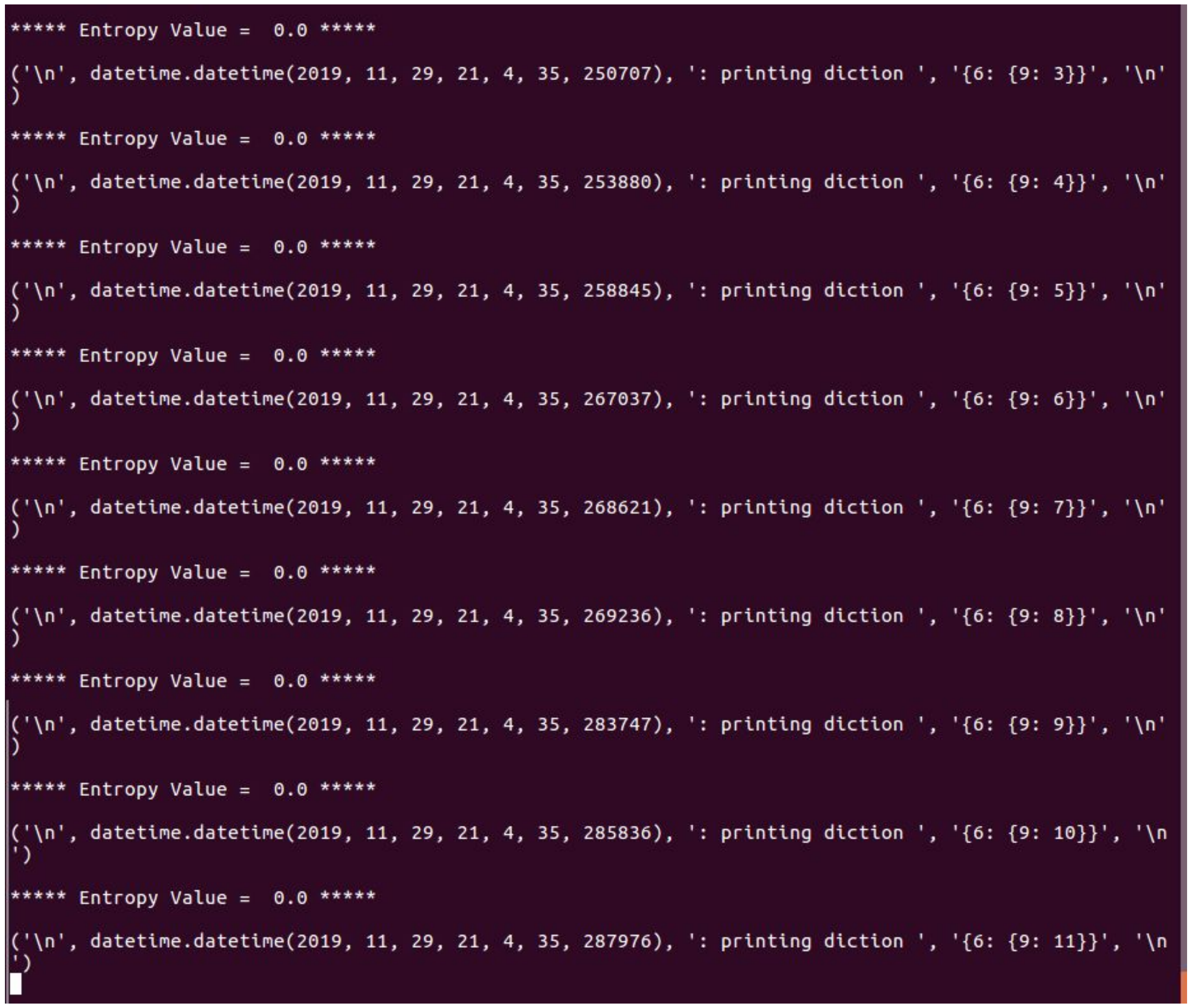


```
UDP  sport=http dport=1 |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=165.230.173.129 dst=['10.0.0.34']
|<UDP  sport=http dport=1 |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=55.74.15.13 dst=['10.0.0.34'] |<UD
P  sport=http dport=1 |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=185.27.131.239 dst=['10.0.0.34'] |
<UDP  sport=http dport=1 |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=187.60.177.126 dst=['10.0.0.34'] |
<UDP  sport=http dport=1 |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=49.34.112.34 dst=['10.0.0.34'] |<U
DP  sport=http dport=1 |>>>
.
```



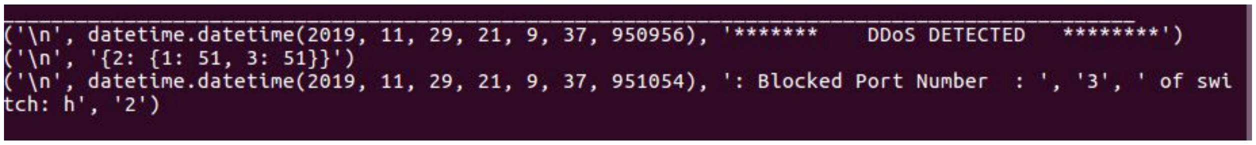
```
UDP  sport=http dport=1 |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=209.143.76.6 dst=['10.0.0.34'] |<U
DP  sport=http dport=1 |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=205.235.64.184 dst=['10.0.0.34'] |
<UDP  sport=http dport=1 |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=131.97.85.46 dst=['10.0.0.34'] |<U
DP  sport=http dport=1 |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=33.60.28.128 dst=['10.0.0.34'] |<U
DP  sport=http dport=1 |>>>
.
Sent 1 packets.
<Ether  type=0x800 |<IP  frag=0 proto=udp src=228.167.185.77 dst=['10.0.0.34'] |
<UDP  sport=http dport=1 |>>>
.
Sent 1 packets.
```

Entropy after the attack:

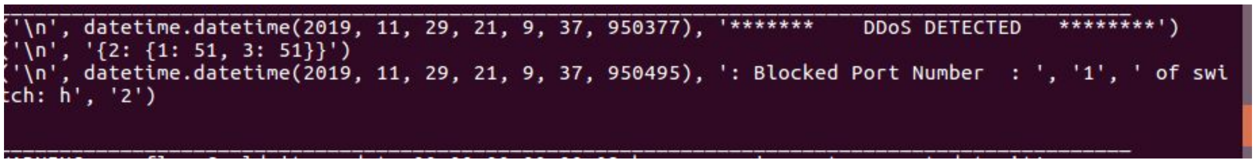


```
***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 4, 35, 250707), ': printing diction ', '{6: {9: 3}}', '\n'
)

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 4, 35, 253880), ': printing diction ', '{6: {9: 4}}', '\n'
)

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 4, 35, 258845), ': printing diction ', '{6: {9: 5}}', '\n'
)

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 4, 35, 267037), ': printing diction ', '{6: {9: 6}}', '\n'
)

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 4, 35, 268621), ': printing diction ', '{6: {9: 7}}', '\n'
)

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 4, 35, 269236), ': printing diction ', '{6: {9: 8}}', '\n'
)

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 4, 35, 283747), ': printing diction ', '{6: {9: 9}}', '\n'
)

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 4, 35, 285836), ': printing diction ', '{6: {9: 10}}', '\n
')

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 4, 35, 287976), ': printing diction ', '{6: {9: 11}}', '\n
')
```

DDoS Attack detection:

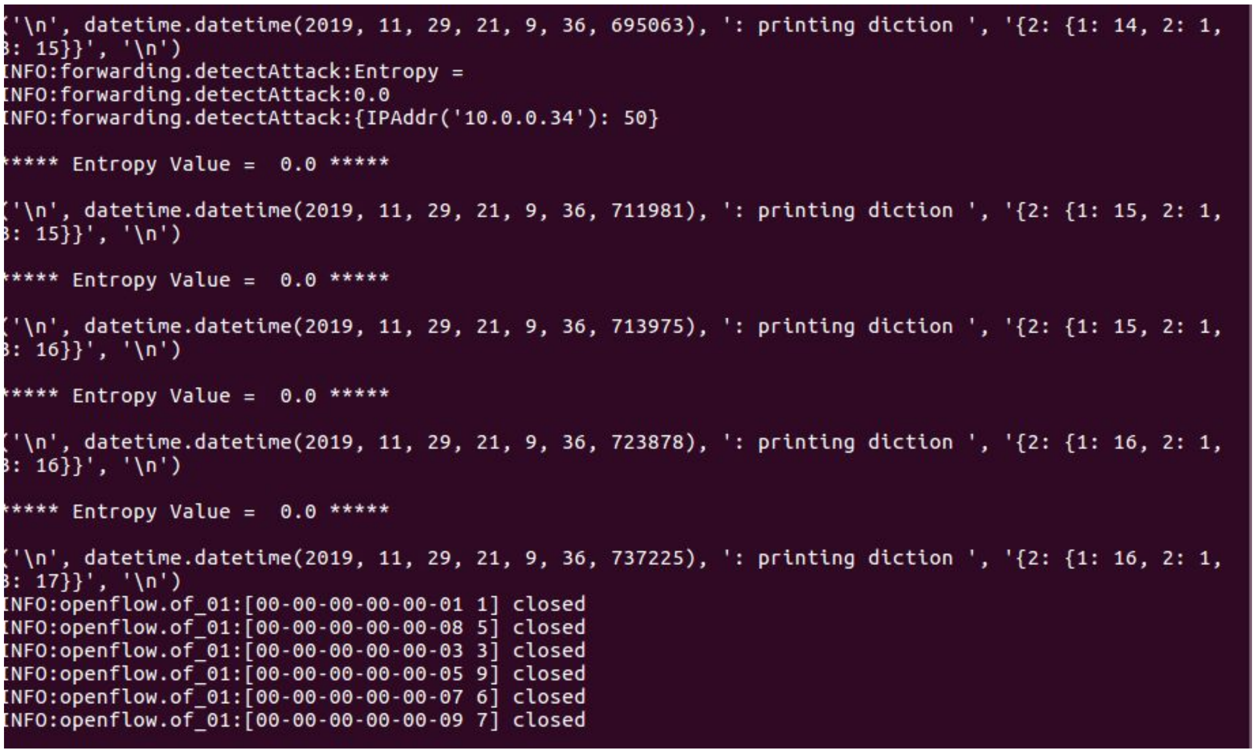


```
('\n', datetime.datetime(2019, 11, 29, 21, 9, 37, 950956), '*******    DDoS DETECTED   ********')
('\n', '{2: {1: 51, 3: 51}}')
('\n', datetime.datetime(2019, 11, 29, 21, 9, 37, 951054), ': Blocked Port Number  : ', '3', ' of swi
tch: h', '2')
```



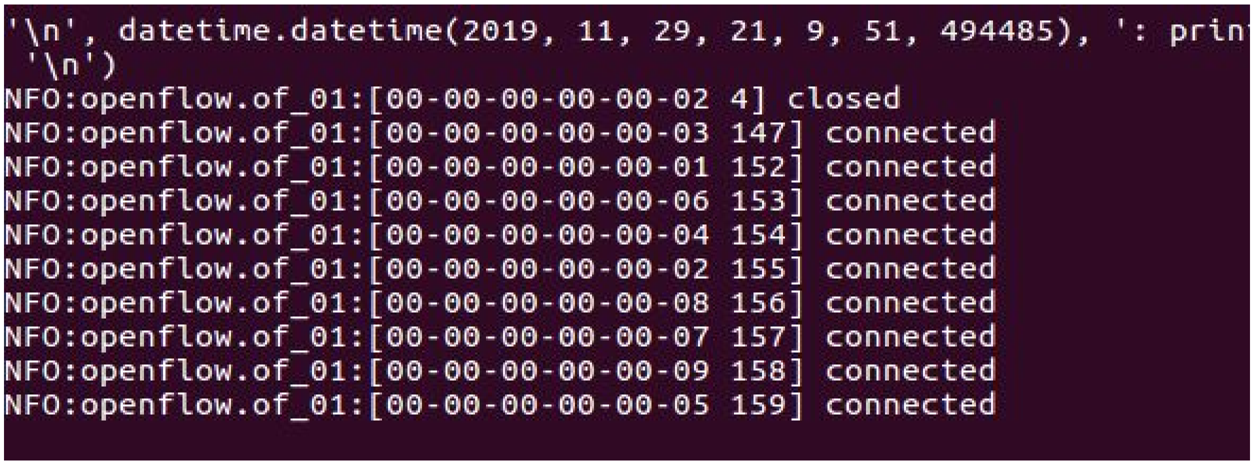
```
('\n', datetime.datetime(2019, 11, 29, 21, 9, 37, 950377), '*******    DDoS DETECTED   ********')
('\n', '{2: {1: 51, 3: 51}}')
('\n', datetime.datetime(2019, 11, 29, 21, 9, 37, 950495), ': Blocked Port Number  : ', '1', ' of swi
tch: h', '2')
```

Prevention by closing/shutting the switches:



```
('\n', datetime.datetime(2019, 11, 29, 21, 9, 36, 695063), ': printing diction ', '{2: {1: 14, 2: 1,
3: 15}}', '\n')
INFO:forwarding.detectAttack:Entropy =
INFO:forwarding.detectAttack:0.0
INFO:forwarding.detectAttack:{IPAddr('10.0.0.34'): 50}

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 9, 36, 711981), ': printing diction ', '{2: {1: 15, 2: 1,
3: 15}}', '\n')

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 9, 36, 713975), ': printing diction ', '{2: {1: 15, 2: 1,
3: 16}}', '\n')

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 9, 36, 723878), ': printing diction ', '{2: {1: 16, 2: 1,
3: 16}}', '\n')

***** Entropy Value =  0.0 *****

('\n', datetime.datetime(2019, 11, 29, 21, 9, 36, 737225), ': printing diction ', '{2: {1: 16, 2: 1,
3: 17}}', '\n')
INFO:openflow.of_01:[00-00-00-00-00-01 1] closed
INFO:openflow.of_01:[00-00-00-00-00-08 5] closed
INFO:openflow.of_01:[00-00-00-00-00-03 3] closed
INFO:openflow.of_01:[00-00-00-00-00-05 9] closed
INFO:openflow.of_01:[00-00-00-00-00-07 6] closed
INFO:openflow.of_01:[00-00-00-00-00-09 7] closed
```

After the attack is finished, the switches are reconnected by the POX controller.



```
'\n', datetime.datetime(2019, 11, 29, 21, 9, 51, 494485), ': prin
 '\n')
NFO:openflow.of_01:[00-00-00-00-00-02 4] closed
NFO:openflow.of_01:[00-00-00-00-00-03 147] connected
NFO:openflow.of_01:[00-00-00-00-00-01 152] connected
NFO:openflow.of_01:[00-00-00-00-00-06 153] connected
NFO:openflow.of_01:[00-00-00-00-00-04 154] connected
NFO:openflow.of_01:[00-00-00-00-00-02 155] connected
NFO:openflow.of_01:[00-00-00-00-00-08 156] connected
NFO:openflow.of_01:[00-00-00-00-00-07 157] connected
NFO:openflow.of_01:[00-00-00-00-00-09 158] connected
NFO:openflow.of_01:[00-00-00-00-00-05 159] connected
```

The problems one may encounter:

- While implementing the network topology in the mininet on one terminal and going inside the pox on another terminal, when the `python ./pox.py forwarding.l3_modified` is run multiple times then an error saying, controller already in use may occur. To resolve it run the following commands on the same terminal:

  ```
  sudo netstat -lpn |grep :6633
  ```

  An output of this form will be expected:

  ```
  tcp6       0     0 :::6633  :::*     LISTEN       6782/java
  ```

  The process Id, which is 6782, now this is the process that is using port 6633. To Kill the process, type:

  ```
  sudo kill 6782
  ```

- There is a high possibility of incurring this error:

  ```
  INFO:packet:(ipv6) warning IP packet data incomplete (114 of
  316)
  INFO:packet:(dns) parsing questions: incomplete name
  ```

  This error will occur after running the following command:
  `python ./pox.py forwarding.l3_modified`

  The POX source code has some bugs which lead to the occurrence of the above errors. It can be corrected by making the following changes:

1. Add at line 244 in packet.dns (pox/pox/lib/packet/)

   ```
   elif r.qtype == 28:
       assert isinstance(r.rddata, IPAddr6)
       return s + r.rddata.toRaw()
   ```

2. Change at line 4431 in openflow.libopenflow_01.py the value of OFP_DEFAULT_MISS_SEND_LEN to a bigger value (default is 128). This error happens because DHCP packets are truncated. For my project, I have set it to 1000. It is present in pox/pox/openflow. Also, edit the .pyc file corresponding to it or relocate it

Relevant code snippets and explanation needed to understand the functionality code offers

Code 1: trafficGen.py

This code is used in traffic generation from a particular node. A random source IP address is generated and packets are sent to a random destination host. The start and end value to be used in the last destination IP octet is given through command line arguments. In the results, I have used 2 and 60 respectively.

The invalid first octate values of the IP address are excluded. They are: [1,2,10,127,169,172,192,254].

In order to send the packets, enable/open the ethernet interface eth0 and to read the IP address and the contents use ifconfig and awk respectively. They can be used with the help of the popen command.

The packets that are created and sent out from eth0 use sendp to facilitate it. It takes source IP, destination IP. Also, UDP source port is assigned to 2 while the destination port address is assigned to 80. The interval between two packets is set to 0.1 seconds.

To run this code go to /mininet/custom in an xterm window of a node then run python trafficGen.py -s 2 -e 60

Code 2: attackLauncher.py

The attack is launched on a particular node by a few chosen hosts in the network. They run this code in their xterm window and launch the attack.

Here, in sendp the interval between two packets is reduced to 0.025. The reason behind doing this was given in the research paper that for this ratio of attack vs traffic packet lead to better detection of the DDoS attack because of the entropy values. Though their value assigned to the threshold value of entropy corresponding to this ratio failed terribly, it should have been 1.31 but with that value, the false-positive increased drastically. Therefore, I tweaked the value and finally decided to keep it at 0.5. This will actually be used in the l3_modified.py. I have just mentioned it here to justify the reason behind taking certain values.

Code 3: detectAttack.py

The basic idea is to do the discrete-time analysis in after a certain number of packets, the entropy has to be calculated. In the research paper, they have set the number of packets value to be 50. Therefore, after every 50 packets, a duration or unit of time will be considered. If in a unit time the value of entropy is below the threshold then again the entropy will be calculated. If it continues to stay below the threshold for next 5 intervals or for 250 packets as a whole, then a DDoS attack is considered.

Also, these changes are reflected in the l3_learning now l3_modified in order to detect the DDoS attack.

The entropy calculation is explained before and the same has been used in the code too!

Code 4: l3_modified.py

In class Entropy, we have defined the entropy dictionary, IP list and destination entropy are as null. The count and value are assigned values as 0 and 1 respectively.

In the statcolect function, we basically collect statistics related to the detection of attack i.e entropy. It's calculated using the hash table values.

And obviously, the value in entropy dictionary which is used in l3_learning module of pox to compare and say whether the attack had occurred or not is stored.

In l3_learning module**, (l3_modified)** we import the entropy function from detectAttack file and then make set_timer and defendddos as false as pox has a predefined method for defending ddos we are making it off.

In isinstance class of pox we will extract the entropy values of the windows from entropy dictionary and compare it with a threshold value, we used 0.5 and whenever entropy is less than the threshold we implement preventing class otherwise we will set the timer as false.

In the timer function, we check the count value and if it is greater than equal to 5 then we can say that a ddos had occurred and we block the port of the switch by sending the

```
msg = of.ofp_packet_out(in_port=i)
core.openflow.sendToDPID(dpid,msg)
```

References:

- https://ieeexplore.ieee.org/document/7069319/authors#authors
- http://www.iieta.org/journals/rces/paper/10.18280/rces.060201
- http://windysdn.blogspot.com/2013/09/start-and-stop-pox-controller.html
- http://0x5f.blogspot.com/2015/11/pox-controller-fixes.html