PROJECT REPORT ON

DETECTING DDOS ATTACK BY ENTROPY CALCULATION.

Submited by,
HARSHITH KUMAR A
JUNE 2021

ABSTRACT

Distributed Denial of Service (DDoS) attack poses a severe threat to the Internet. It is difficult to find the exact signature of attacking. Moreover, it is hard to distinguish the difference of an unusual high volume of traffic which is caused by the attack or occurs when a huge number of users occasionally access the target machine at the same time. The attacker makes use of proper service requests to occupy excessive service resources to force the server crash, or to make other legal users unable to attain timely service responses. DDoS detection is the process of distinguishing Distributed Denial of Service (DDoS) attacks from normal network traffic, in order to perform effective attack mitigation.

The entropy detection method is an effective method to detect the DDoS attack. We know that every network in the system has an entropy and increase in the randomness causes entropy to decrease. Here entropy calculation mainly used to calculate the distribution randomness of some attributes in the network packet's headers. Results show that these methods could detect DDOS and after detecting we shall block that specific port in the switch if it drops below certain threshold value, and then bring the port down.

Contents

List of Figures

Chapter 1

INTRODUCTION

The number of Internet users and network services continuously increasing and the dependence on Internet for the critical online services also growing. So there is a high damage cost due to network attacks like Distributed denial of service attack. Contrary to conventional host or service based attacks, Distributed Denial of Service (DDoS) attacks are considered more disruptive in nature. These attacks make targeted services unavailable by sending a significantly large number of malicious access requests to a service provider. When the host under attack has limited computing, memory and network bandwidth, the consequence of DoS attacks could be fairly serious. However, along the development of computer and network technology, the impact of DoS attacks has been significantly mitigated.

1.1 DISTRIBUTED DENIAL OF SERVICE

In computing, a denial-of-service attack (DoS attack) or distributed denial-of-service attack (DDoS attack) is an attempt to make a machine or network resource unavailable to its intended users. It is a "Denial of Service". The server is never compromised, the databases never viewed, and the data never deleted. Throughout and after the attack, the server remains intact.

Confidentiality, Integrity and Availability are the three major components of cyber security. Denial of Service (DoS) and its variant, Distributed Denial of Service (DDoS), are possible threats which exhaust the resources to make it unavailable for the legitimate users, thereby, violating one of the security components- Availability. DoS attacks to networks are numerous and potentially devastating. So far, many types of DoS attacks are identified and most of them are quite effective to stop the communication in the networks. IPv4 as well as IPv6 are quite vulnerable to these attacks. A number of countermeasures are developed to mitigate these attacks.

Most of the hackers keep three things in mind. One is to explore a way through which they can get the secret information. This is to compromise the confidentiality. Second is get access to the confidential information to change or modify it. This involves the compromising of integrity. Third is to compromise the availability. The first two options cannot be enjoyed by novice attackers because it is not easy to gain an unauthorized remote access to a system. Thus, they try to target the availability for which they do not need any administrative privilege on the target system.

Most DoS attacks depend upon the weaknesses in TCP/IP stack protocols. Some of the classical examples of DoS attacks are TCP Syn Flood, UDP Flood, ICMP flood, Smurf and Incomplete HTTP Requests, etc. Attackers either make use of single computer or multiple computers to launch these attacks. The usage

of multiple computers to perform the attack is known as DDoS attack. The different systems are first compromised by using Trojans, worms, etc. and then used by the attackers. These compromised machines are named as zombies while the controller machine is considered as master. This master- zombie relationship works somewhat similar to client-server architecture. It can be very difficult to detect the DDoS attacks because the zombies may be situated across the globe. As a result, they cannot be differentiated from the legitimate traffic

1.2 ENTROPY

In data communications, the term entropy refers to the relative degree of randomness. By definition entropy is the randomness of the occurrence of an event. Let an information source have independent symbols each with the probability of Pi . Then the entropy H is defined as:

$$H = \sum_{i=1}^{n} P_i \log P_i$$

where,

$$P_i = \frac{x_i}{n}$$

and xi, is the frequency of the occurrence of parameter i in the observation of on n events. The main reason for considering entropy is its ability for measuring randomness in a network. The higher the randomness the higher is the entropy and vice versa. For a normal traffic there will be considerable amount of entropy. In the case of DDos attack randomness will decrease, entropy als will decrease with randomness. So, whenever the entropy is less than a threshold value we can say that a DDos attack is occurred. In our case we used 0.5 as threshold value.

Chapter 2

PROPOSED SYSTEM

Software-defined networking(SDN) architectural will separates the data plane from the control plane to make a network more flexible and easier to manage. SDN centralizes management by abstracting the control plane from the data forwarding function in the discrete networking devices. This provides easy network configuration by supporting a programmable interface for applications development related to security, management etc. and the centralized logical controller provides more control over the total network, which has complete network visibility.
But in the case of SDN attacks is much easier when compared to traditional networks, where the network devices have protection from the attacks and limits the occurrence of attacks. DDos is one of the most popular one. For preventing this DDoS threat, we want to use POX for attack detection and want to provide a solution that is effective in terms of the resources used. Using POX controller we will see how DDoS attacks will consume controller resources and provide a solution to detect such attacks based on the entropy variation of the destination IP address.

2.1 SOFTWARE-DEFINED NETWORKING

Software-defined networking (SDN) technology is an approach to network management that enables dynamic, programmatically efficient network configuration in order to improve network performance and monitoring, making it more like cloud computing than traditional network management. SDN is meant to address the fact that the static architecture of traditional networks is decentralized and complex while current networks require more flexibility and easy troubleshooting. SDN attempts to centralize network intelligence in one network component by disassociating the forwarding process of network packets (data plane) from the routing process (control plane). The control plane consists of one or more controllers, which are considered the brain of the SDN network where the whole intelligence is incorporated. However, the intelligent centralization has its own drawbacks when it comes to security, scalability and elasticity and this is the main issue of SDN.
Features and benefits :

- Reduce complexity by decoupling the control and data planes, while making automation highly secure and scalable.
- Deploy applications and services faster by leveraging open APIs. Easily integrate third-party products.
- Eliminate manual configuration. Provision and manage data centers, campuses, and wide-area networks.

- Centralize configuration, management, control, monitoring, service delivery, and cloud automation.

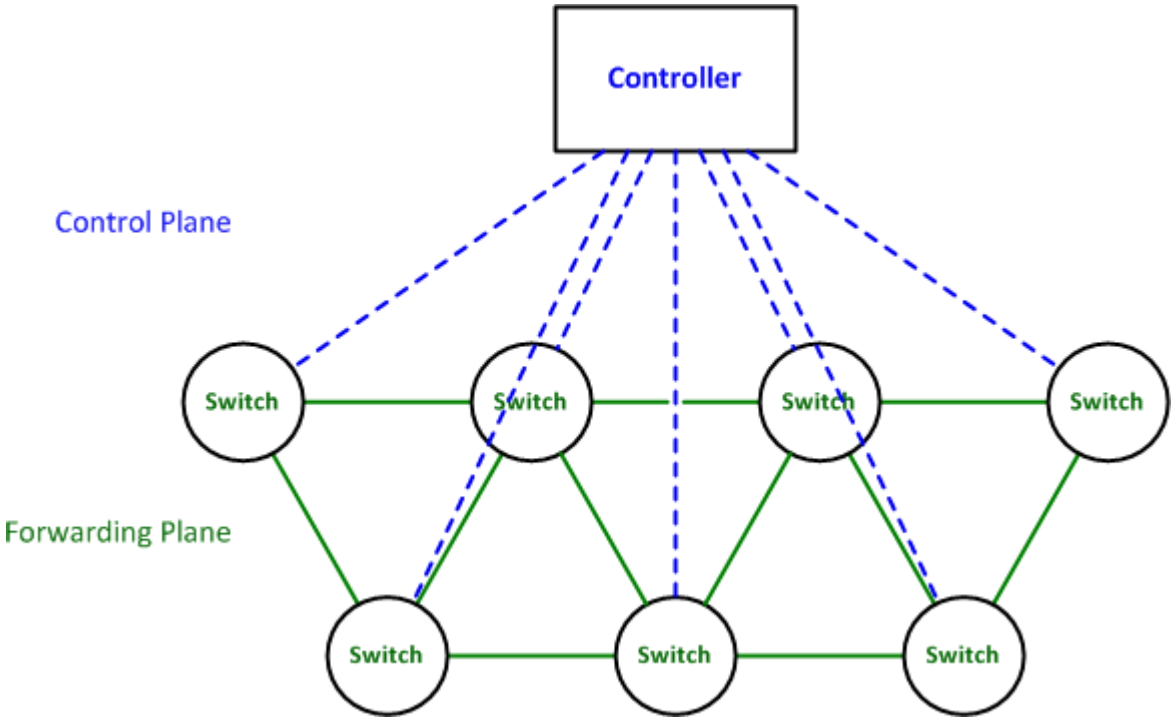

Figure 2.1: SDN controller

The use of OpenFlow makes the controller a primary victim for the attacker because of the following reasons :

- In OpenFlow protocol there is no standard for security implementation and developers of products are implementing their own proprietary methods.
- The programmable aspect of SDN also makes them much more vulnerable to numerous malicious attacks and code exploits.
- Errors related to configuration of SDN can be more serious than traditional network errors.

2.2 MININET

Mininet is open source software that is used to simulate a software defined network (SDN) and its compatible controllers, switches and hosts. It comes in handy for networking enthusiasts and beginners to get some experience in working with the fast growing SDN technology. To put it simply, a SDN separates the control plane from the data forwarding plane, making network devices such as switches and routers fully programmable and, hence, the network behaves according to the users' requirements. By default, Mininet provides OVS switches and OVS controllers. However, it has the support to install other/preferred SDN controllers and switches instead of the defaults. The primary feature that distinguishes SDN devices from traditional network devices is the scope for customising protocols and functions.
Mininet supports the Openflow protocol, which provides an interface between the control plane and the data forwarding plane. Openflow protocols are used to control the packet flow as per the API written on the controller. Mininet also has support for a variety of topologies and ensures the availability of custom

topologies. The CLI (command line interface) provided by Mininet is comfortable to use after a bit of practice.

2.3 POX CONTROLLER

There are many types of controllers. Some of the popular OpenFlow controllers are NOX, POX, Beacon, Floodlight, Ryu and Open Daylight, etc. POX is very popular for prototyping and for its simple structures. Others are used for the production network. POX Is defined as an OpenFlow controller that was derived and developed through the use of python programming language that basically aim to provide an efficient and easy environment for performing research investigations and tests in SDN networks. POX relies on component-based model in which the whole network elements as well as activities are recognized as separate components that are able to be isolated and utilized every time and place the need is. The location of POX is specifically between network components at one side and the applications on the other side.
POX is a Python based open source OpenFlow or Software Defined Networking (SDN) Controller. POX is used for faster development and prototyping of new network applications. POX controller comes pre installed with the mininet virtual machine. Using POX controller you can turn dumb openflow devices into hub, switch, load balancer, firewall devices. The POX controller allows easy way to run SDN experiments. POX can be passed different parameters according to real or experimental topologies, thus allowing you to run experiments on real hardware, testbeds or in mininet emulator.

2.4 SCAPY

Scapy is a powerful interactive packet manipulation tool, packet generator, network scanner, network discovery, packet sniffer, etc. It can for the moment replace hping, parts of nmap, arpspoof, arp-sk, arp-ing, tcpdump, tshark, p0f, etc. These kind of tools do two things : sending packets and receiving answers. That's what scapy does : you define a set of packets, it sends them, receives answers, matches requests with answers and returns a list of packet couples (request, answer) and a list of unmatched packets. Scapy uses the python interpreter as a command board. That means that you can use directly python language. If you give a file as parameter when you run scapy, your session will be saved when you leave the interpretor, and restored the next time you launch scapy.

Chapter 3

IMPLEMENTATION

Network topology adopted in this project was a tree topology which consisted of 9 switches and 64 hosts; switches employed were OpenFlow-enabled switches which support OpenFlow protocol and expected to be connected to the POX controller.

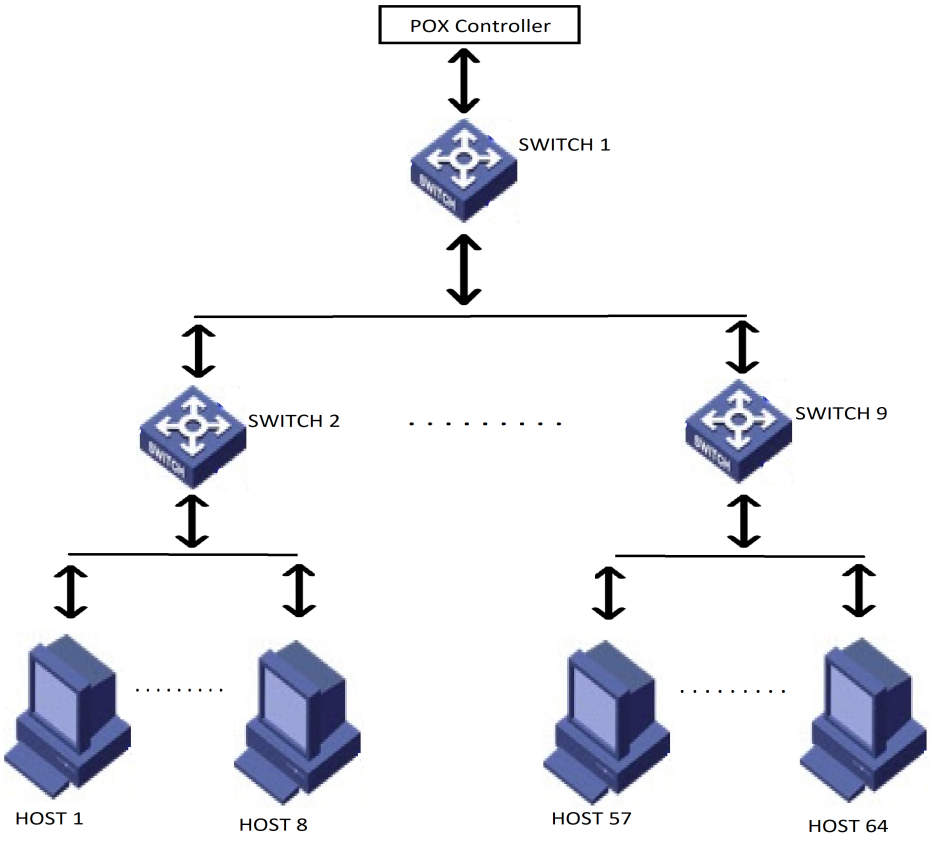

Figure 3.1: Mininet Topology

Setup of the network used is shown in Figure 3.1 where POX controller was connected to Tree topology with one switchs (s1-s9) and 64 hosts (h1-h64). Openflow controller is connected to a network. The performance of the POX controller was reported. We then observe the entropy of the traffic related to the controller under normal and attack conditions.
In these case, POX controller was modified to call the detection algorithm in a new module called 'l3 edited' also, host (h1) was used to generate the normal traffic and host (h2) was used to generate DDoS attack traffic. The idea behind using these tests is to show the behavior of the networks when it deals with normal and attack traffic.

Normal and DDoS attack traffic are generated using Scapy which is a powerful tool for packet manipulation, Scapy is started programmatically by two different python codes for normal and attack traffic flows, these codes generate large number of random spoofed source IP addresses. In normal traffic we basically, generates random source ip and send the packets to random destinations between the hosts. We generally give start and end value from the command prompt of one of the node for example h1. While in attack traffic, all packets from the spoofed IP addresses are sent to victim host that has the IP (10.0.0.64). The attack traffic has a higher rate than normal traffic. Now the entropy value in the controller decreases.

3.1 ALGORITHM

Every new packet that comes to a openflow-enabled switch is send to the controller as a PacketIn event. During a DDoS the attacker sends a huge amount of traffic from numerous compromised devices, often distributed globally in what is referred to as a botnet, through the network consuming the resource as well as the bandwidth.
A large number of nodes send packets to the victim host. Hence, a detector placed at the controller collects the packets sent to a switch. Each new packet in the network is sent to the controller for specifying a flow for the packet and similar ones. This analyses the source IPs of the packets and counts the number of packets that each sender (source IP) sends to a host in a given network to produce a source-IP distribution table. The table is later then used to detect DoS/DDoS and determine who is issuing the attack.
First, make a count of 50 packets in a window and then calculate the entropy and compare it with threshold and make a count of consecutive entropy value lower than threshold. If this count reaches 5 then DDoS had occurred otherwise not. For this purpose we will modify "l3 learning" module of the pox controller so that it can detect the DDos. Here we will call our class entropy calculation inside the module.

3.2 FLOWCHART

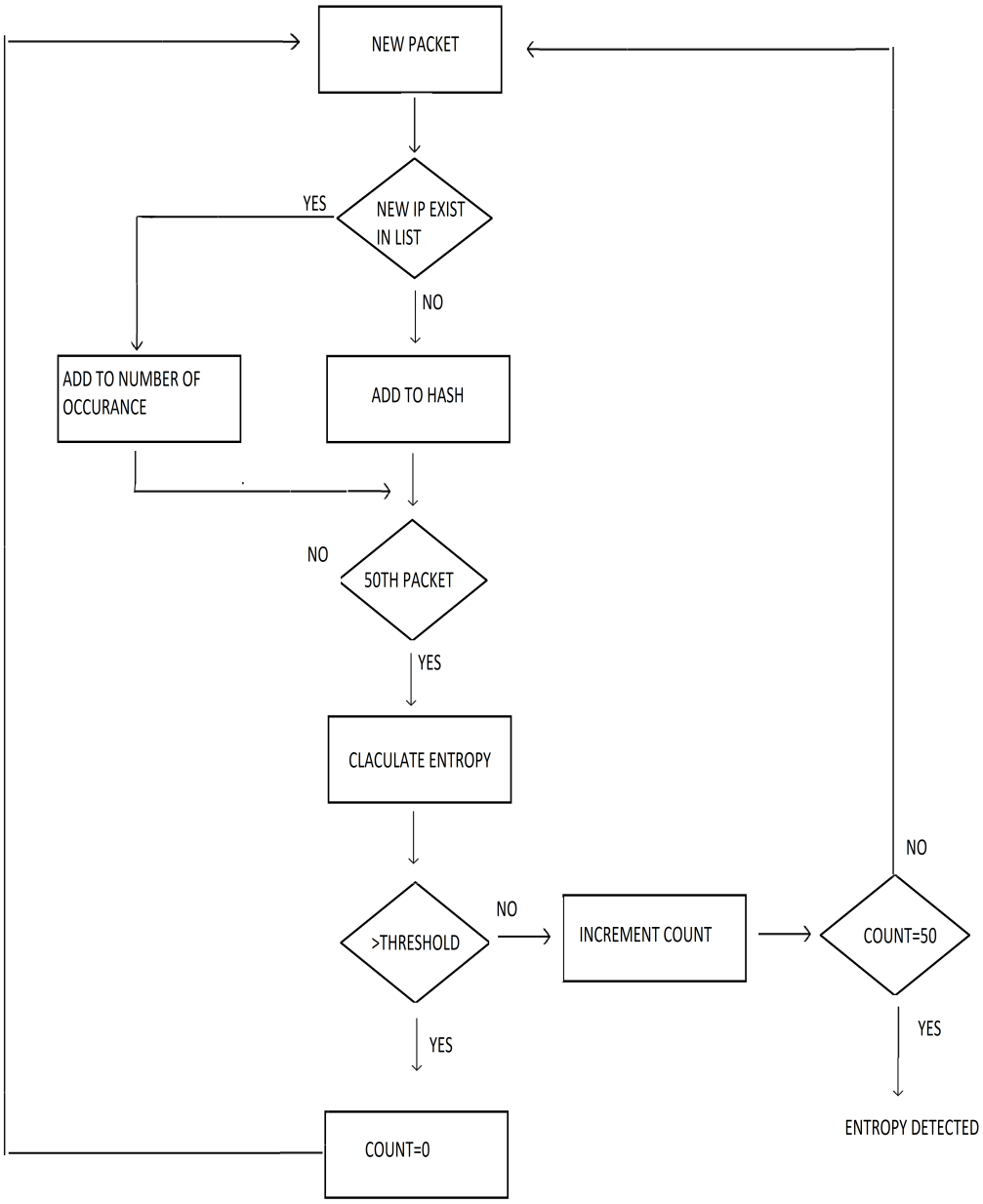

Figure 3.2: Algorithem

Chapter 4

CONCLUSION

Some of the key ideas of SDN are the introduction of dynamic programmability in forwarding devices through open southbound interfaces, the decoupling of the control and data plane, and the global view of the network by logical centralization of the 'network brain'. While data plane elements became dumb, highly efficient and programmable packet forwarding devices, the control plane elements, are now represented by a single entity, the controller.

A Distributed Denial of Service (DDoS) attack is an attempt to make the services of network unavailable by exhausting the resources. The effect of this DDoS attack is even worse in an SDN than the traditional one. Hence within a short span of time if the attacker bursts too much packets into a network it will damage the controller, followed by the collapse of the network.

We have implemented a method to detect DDoS attacks successfully. Here , every packet which has entropy value less than the threshold value, is stored in a dictionary and if an entry comes more than 5 times, it is assumed as a DDoS attack. This knobe value of 5 depends on the topology. Hence, a topology smaller in size may show DDoS attack for normal traffic also.

As a result of the experiment, Entropy calculation for DDoS detection turns out to be a effective solution for the detection of anomaly in network traffic in a SDN architecture. This can be a efficient method to adopt at software defined data centers.

Appendix A

CODES

A.1 ENTROPY DETECTING

```
from pox.core import core

log = core.getLogger()

class Entropy(object):
      count = 0
      destFrequency = {}
      destIP = []
      destEntropy = []
      value = 1

      def collectStats(self, element):
          l = 0
          self.count +=1
          self.destIP.append(element)
          if self.count == 50:
             for i in self.destIP:
                 l +=1
                 if i not in self.destFrequency:
                    self.destFrequency[i] = 0
                 self.destFrequency[i] += 1
             self.findEntropy(self.destFrequency)
             log.info(self.destFrequency)
             self.destFrequency = {}
             self.destIP = []
             l = 0
             self.count = 0
```

```
        def findEntropy (self, lists):
            l = 50
            entropyList = []
            for k,p in lists.items():
                c = p/float(l)
                c = abs(c)
                entropyList.append(-c * math.log(c, 10))

                log.info('Entropy = ')
                log.info(sum(entropyList))

                self.destEntropy.append(sum(entropyList))

            if(len(self.destEntropy)) == 80:
                print (self.destEntropy)
                self.destEntropy = []
            self.value = sum(entropyList)

        def __init__(self):
```

A.2 NORMAL TRAFFIC LAUNCHING

```
import sys
import getopt
import time
from os import popen
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import sendp, IP, UDP, Ether, TCP
from random import randrange
def sourceIPgen():
    not_valid = [10,127,254,1,2,169,172,192]
    first = randrange(1,256)
    while first in not_valid:
        first = randrange(1,256)
    ip = ".".join([str(first),str(randrange(1,256)),str(randrange(1,256)
          str(randrange(1,256))])
    return ip
def gendest(start, end):
    first = 10
```

```
    second =0; third =0;
    ip = ".".join([str(first),str(second),str(third),
           str(randrange(start,end))])
    return ip
def main(argv):
    print argv
    try:
        opts, args = getopt.getopt(sys.argv[1:],'s:e:',['start=','end=']
    except getopt.GetoptError:
        sys.exit(2)
    for opt, arg in opts:
        if opt =='-s':
            start = int(arg)
        elif opt =='-e':
            end = int(arg)
    if start == '':
        sys.exit()
    if end == '':
        sys.exit()
    interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()
    for i in xrange(1000):
        packets = Ether()/IP(dst=gendest(start, end),src=sourceIPgen())
                  /UDP(dport=80,sport=2)
        print(repr(packets))
        sendp( packets,iface=interface.rstrip(),inter=0.1)
if __name__ == '__main__':
  main(sys.argv)
```

A.3 ATTACK LAUNCH

```
import sys
import time
from os import popen
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import sendp, IP, UDP, Ether, TCP
from random import randrange
import time
def sourceIPgen():
```

```
  not_valid = [10,127,254,255,1,2,169,172,192]
  first = randrange(1,256)
  while first in not_valid:
    first = randrange(1,256)
  ip = ".".join([str(first),str(randrange(1,256)), str(randrange(1,256)
        str(randrange(1,256))])
  print (ip)
  return ip
def main():
  for i in range (1,5):
    mymain()
    time.sleep (10)
def mymain():
  dstIP = sys.argv[1:]
  print (dstIP)
  src_port = 80
  dst_port = 1
  interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()
  for i in xrange(0,500):
    packets = Ether()/IP(dst=dstIP,src=sourceIPgen())
                  /UDP(dport=dst_port,sport=src_port)
    print(repr(packets))
    sendp( packets,iface=interface.rstrip(),inter=0.025)
if __name__=="__main__":
  main()
```

Appendix B

SCREENSHOTS

Figure B.1: Creating tree topology which consisted of 9 switches and 64 hosts using mininet.
usning the command :

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

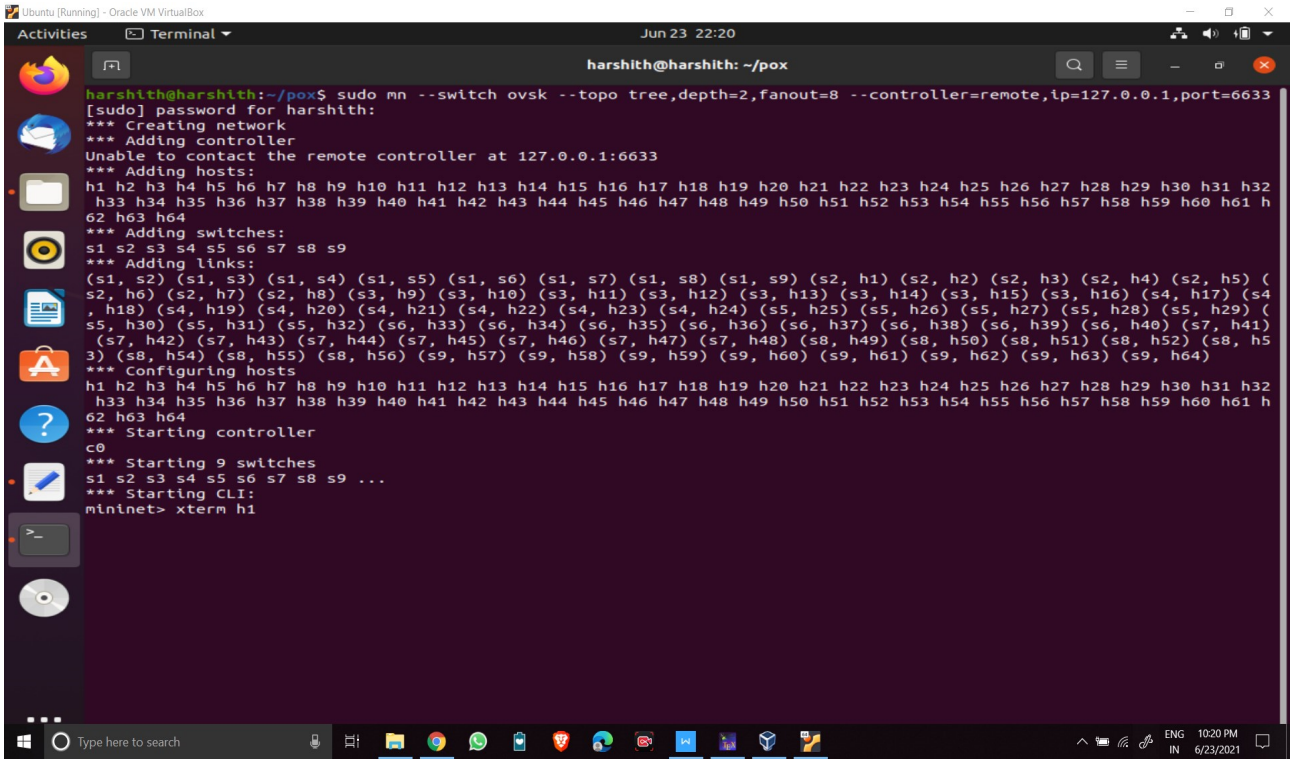

Figure B.1: Creating topology

Figure B.2: Simultaneously, In the Mininet terminal of Virtual Box enter the following command to run the pox controller:

```
$ cd pox $ python3 ./pox.py forwarding.l3_detectionEntropy
```

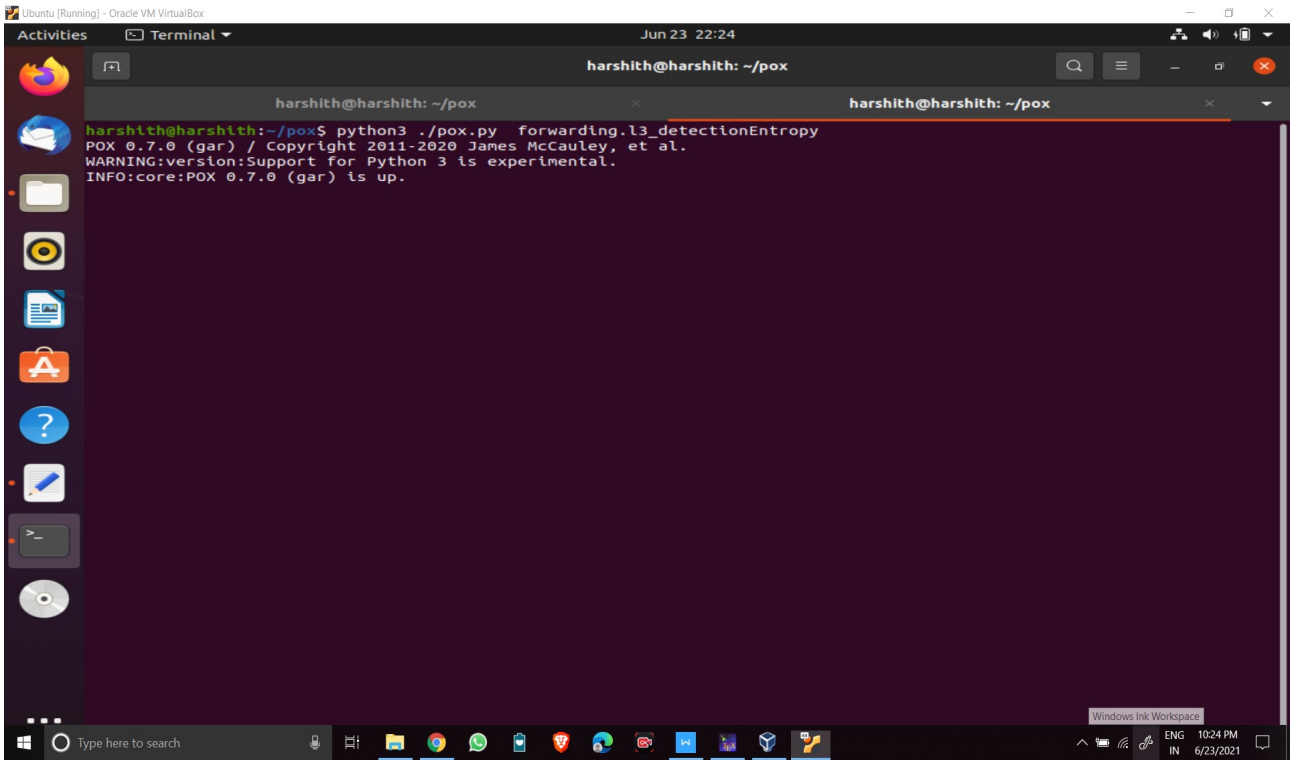

Figure B.2: Pox Controller

Figure B.3: Run the traffic generation script from one of the host which generates random source IP and send the packets to random destinations. Observe the entropy in POX controller terminal.

```
$ python3 Trafficlauch.py  -s 2 -e 65
```

Figure B.4: Now repeat above step on h1 and parallelly enter the following command to run the attack traffic from h4 and h6 xterm windows to attack on h56. python launchAttack.py 10.0.0.56

```
$ python3 attackclaunch.py 10.0.0.2
```

Figure B.5: Observing the entropy values in the POX controller. The value decreases below the threshold value (which is equal to 0.5 here) for normal traffic. Thus, we can detect the attack within the first 250 packets of malicious type of traffic attacking a host in the SDN network. After the hostsstop sending attack packets, the switches are started again by the POX controller after shutting them down during the attack.

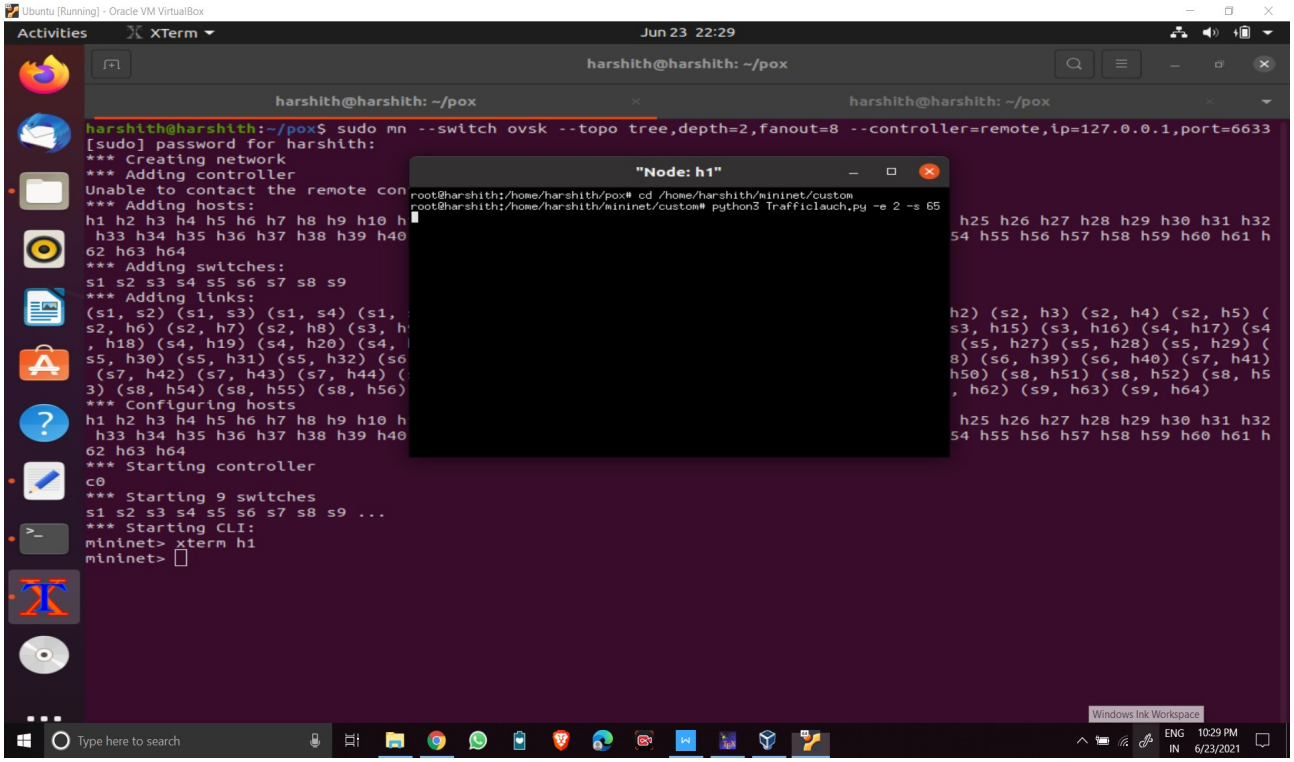

Figure B.3: Traffic Generation

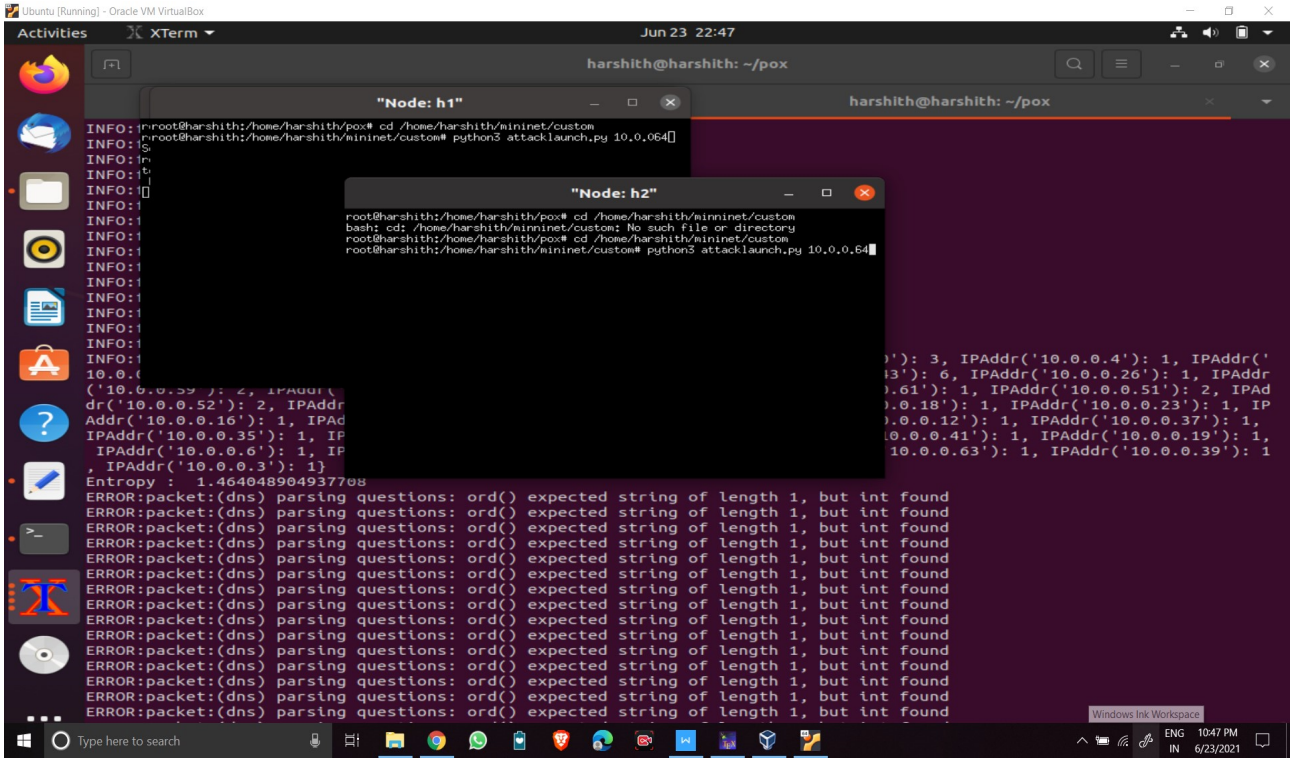

Figure B.4: Launch Attack

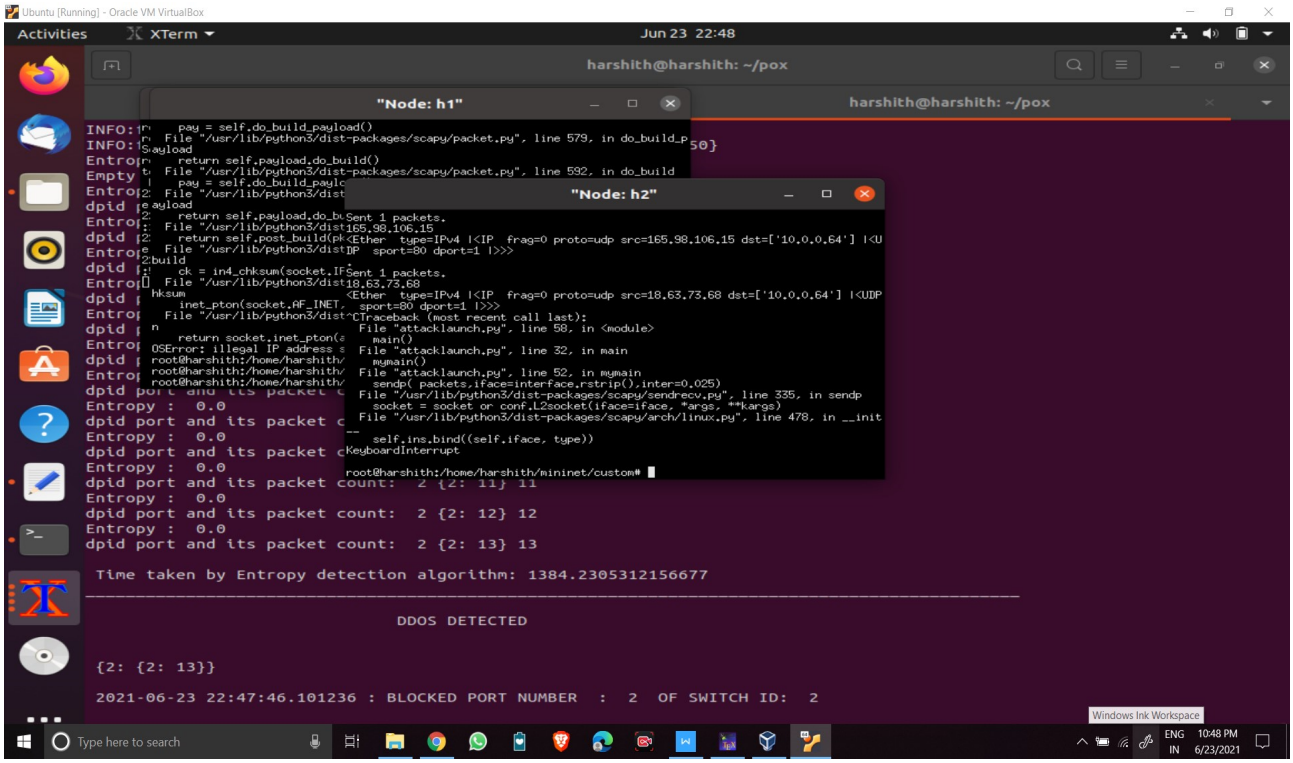

Figure B.5: Entropy value after DDoS attack