**B.Tech. Project Work Report (CSIR30)**

**on**

**EARLY DETECTION OF DDOS ATTACK IN SOFTWARE DEFINED NETWORKS USING ENTROPY**

**BY**

**Vaibhav Sahu (*21812022*)**

**Under the Supervision of**

**Dr. Anoop Kumar Patel, Asst. Prof.**



**DEPARTMENT OF COMPUTER ENGINEERING**

**NATIONAL INSTITUTE OF TECHNOLOGY**

**KURUKSHETRA – 136119, HARYANA (INDIA)**

**May-June 2020**



**CERTIFICATE**

I, Vaibhav Sahu hereby certify that the work which is being presented in this B.Tech Project

Work (CSIR30) report entitled “**Early Detection of DDoS Attack in Software Defined** **Network using Entropy**”**,** in partial fulfillment of the requirements for the award ofthe **Bachelor of Technology (Computer Engineering)** is an authentic record of my own work carried out during a period from January, 2021 to April, 2021 under the supervision of **Dr. Anoop Kumar Patel, Assistant Professor**, Computer Engineering Department.

The matter presented in this project report has not been submitted for the award of any other degree elsewhere.

*Signature of Candidate*

**Vaibhav Sahu (21812022)**

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

*Signature with date*

*Faculty Mentor*

**Dr. Anoop Kumar Patel**

Asst. Prof.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Section** |  | **Title** | **Page** |
| **no.** |  |  | **no.** |
|  | **Abstract** |  |  |
|  |  |  |  |
| **1** | **Introduction** |  | **1** |
| **2** | **Motivation** |  | **2** |
| **3** | **Literature Survey** | | **3** |
|  | 3.1 Software Defined Network | | 3 |
|  | 3.2 OpenFlow Protocol | | 5 |
|  | 3.3 SDN Controllers | | 7 |
|  | 3.4 Distributed Denial of Service (DDoS) Attack | | 8 |
| **4** | **Early Detection of DDoS using Entropy** | | **10** |
|  | 4.1 Why Entropy? | | 10 |
|  | 4.2 Early Detection | | 11 |
|  | 4.3 Software and Tools Used | | 14 |
|  | 4.3.1 | Controller | 14 |
|  | 4.3.2 | Network Emulator | 14 |
|  | 4.3.3 | Packet Generation | 14 |
|  | 4.3.4 | Network Setup | 15 |
| **5** | **Data Flow Diagram** | | **17** |
|  | 5.1 Level 0 DFD | | 17 |
|  | 5.2 Level 1 DFD | | 17 |
|  | 5.3 Level 2 DFD | | 18 |
| **6** | **Gantt Chart** |  | 19 |
| **7** | **Experiment Walkthrough** | | **20** |
| **8** | **Results** |  | **25** |
| **9** | **Conclusion** |  | **27** |
|  | **References** |  | **28** |
| **Appendix:** |  |  |  |
|  | A. Detection\_Entropy code | | 29 |
|  | B. L3\_learning updated code | | 30 |
|  | C. Traffic code | | 32 |
|  | D. Attack code | | 33 |

**ABSTRACT**

Basically, as we know (SDN) software-defined architecture reduces the function of n / w controllers by isolating the data plane the control plane. This provides a simple network configuration by supporting a random indication for the development of security related applications, administration etc. The central location controller provides additional control over a full, fully transparent network.

These SDN benefits outline the network at risk and the impact of attacks is sharp compared to traditional networks, where network devices protect against attacks and reduce the occurrence of attacks.

In this project, we want to explore the various attacks that can be launched on SDN in different layers. and we want to explore some of the security options available to mitigate attacks and explore a possible solution to prevent DDoS attacks using entropy.

Basically, a Denial of Service (DDoS) distributed attack is a DoS attack that uses multiple shared attack resources. We know that all networks in the system have an entropy and an increase in randomness causes entropy to decrease.

By preventing this DDoS threat, we want to use POX to gain attack detection and we want to provide an effective solution depending on the resources used.

Specifically, this project shows how DDoS attacks can consume control resources and provide a solution for detecting such attacks depending on the various IP address variables at your destination. Now based on this entropy value, we will block that port from changing if it drops to a certain limit value, then lower the port.

In addition, it is difficult to distinguish the differences of abnormalities a large amount of traffic caused by an attack or incident many users occasionally access the virtual machine at one time. How to get access to an item is an effective way to find out DDoS Attack It is used mainly for random routing calculations of other attributes in the headers of network packets. In this paper, we focus on DDoS attack detection technologies. We are developing a previous detection algorithm and propose two improved findings methods based on the entropy of accumulation and time, respectively. Experimental results suggest that these methods may lead to greater accuracy effective DDoS adoption.

1. **INTRODUCTION**

Software Defined Networking (SDN) is a new networking approach that is introduced with the goal to simplify the network management by separating the data and control planes. SDN has brought with itself programmability within the network control plane. The shift of the control logic from networking devices, like switches and routers, in traditional networks to a centralized unit referred to as the controller permits the physical network hardware to be detached from the control Plane. This separation simplifies the planning of latest protocols and implementation of latest network services like access control, QOS, enforcement of latest policies, bandwidth management, traffic engineering etc. No longer does every chickenfeed got to come at the value of reconfiguring all the network devices.

The centralized structure of the controller could lead on to several security challenges. One of such critical challenges is that the impact of distributed denial of service attacks (DDoS) on SDN networks. In a DDoS attack multiple compromised system are usually infected with a Trojan and are used to target a single or multiple victim in the network. The attack traffic flooding the victim uses many various spoofed source IP addresses. This effectively makes it impossible to prevent the attack by only blocking traffic based on source IP addresses. it's also very difficult to differentiate legitimate user traffic from attack traffic when the attack traffic sources are spread across the web.

2

1. **MOTIVATION**

More than 50% of IT leader say their organization experienced 10 or more DDoS attacks in the past year. 40% of these attacks were at least 10 hours long. Another source estimates there were 7.5 million total DDoS attacks in 2017.

In the past year, DDoS has trended towards larger-scale attacks. 2018 saw the highest-frequency attack yet; the first terabyte-per-second-scale threat. In the first half of 2018, there were seven times more attacks larger than 300 gigabytes-per-second than in the first half of 2017. Unfortunately, the emergence of the shady DDoS-as-a-service industry is likely to cause continued innovation among entrepreneurial cybercriminals.

Falling victim to a DDoS attack is probably even more expensive than you think. As we mentioned above, you can expect a successful DDoS attack to take $50K out of your account. But the numbers could go higher.

So, we need a DDoS Detection algorithm, so we not be able to become a victim and making a light-weighted and early detection can be our priority.

3

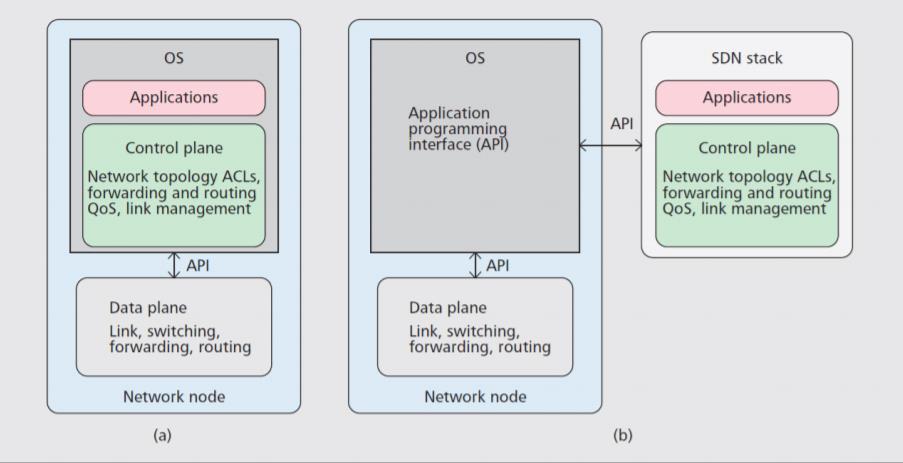
1. **LITERATURE SURVEY**

**3.1 Software Defined Network**

Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today’s applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. Making Network more programmable is the goal of SDN(1)

1. Qiao Yan, F. Richard Yu, Senior Member, IEEE, Qingxiang Gong, and Jianqiang Li: Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 18, NO. 1, FIRST QUARTER 2016

As shown in figure 1(a), the network node operation in a traditional network showing the interaction between the data and control planes. It is the control plane’s responsibility to finds the paths across the network and pass it to the info plane to enable data forwarding. In large networks this might take quite time for reconfiguring the devices to adapt to the traffic demands and network conditions. The main concept of SDN lies in decoupling the control and data planes as shown in figure 1(b).

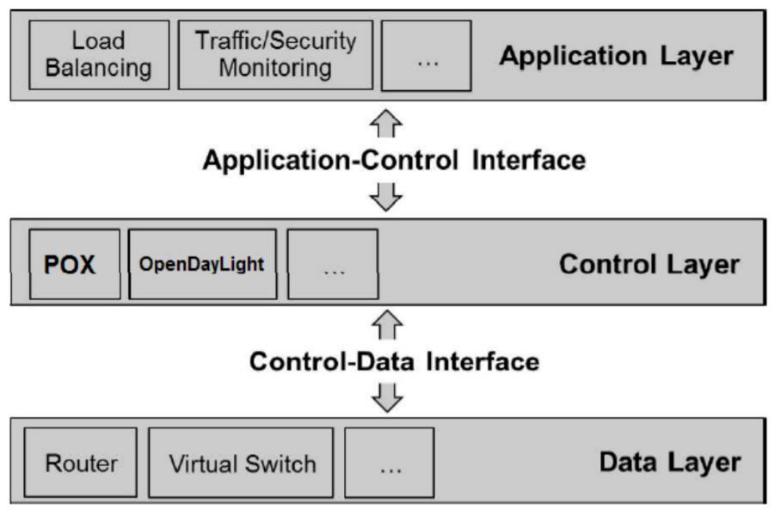


**Figure 1: Comparing Traditional Networks and SDN**

3

SDN uses the idea of centralized network control plane and introduces programmability, which can be used for network management and enables run-time organization for security strategies. In SDN architecture, the three primary functional layers or SDN planes are showed in Figure. 2 and talked about beneath.

* **Application Plane**: It is the top-most layer of the SDN design. It establishes differentSDN applications including different functionalities, for example, policy execution, network management, and security services. The northbound interface is used for an interface between the application plane and control plane and also known as Application-Control Interface.
* **Control Plane**: Control plane deals with the SDN switch through the commands andprovides hardware abstractions to SDN applications. The control plane functionalities used for the system configuration, management, and exchange of routing table information. It encourages network programmability by decoupling from system hardware and runs as a software.
* **Data Plane:** Data plane overseessending the traffic flows dependent on flow ruleswhich are modified by the control plane. Some network devices, like router, switches lie in the data plane. The south bound interface is an inter face between the control between the control plane and data plane which also known as Control-Data Interface.



**Figure 2: SDN Functional Architecture**

4

**3.2 OpenFlow Protocol & Switches**

Switches are allowed by OpenFlow to be controlled by an independent controller. Transport Layer Security (TLS) enabled channel is used for communication for switches and controllers. Each switch can contain one or more flow tables which help to keep the flow entries detailed by the controller.

A flow entry includes a header which identifies the individual flow that the packets are matched or not and if it matched then some action are taken into the flow table. Different actions can take like packet forwarding, drop, further lookups in others flow tables, rewriting of the header fields etc.

An OpenFlow switch contains single or group of flow table followed by a secure connection to controller. Each table features a match field, counters, and a group of instructions for every entry. Table 1 shows the fields that a switch can use to find a match in its tables. In the table, there is a metadata field that is defined (second row in Table 1) to carry information from one table to the other when there is more than one table.

Often switches have multiple tables that are linked together. The packet will move from one table to the opposite finding match, carrying the metadata. If a match is found, the metadata tag will be updated according to the table.

Any packets entering the switch are going to be checked against all existing flows within the tables. If a match is found, the action assigned to that entry is applied and the counter for the entry will be updated. Counters cover several components in the switch like counters per flow entry, per port, per table, per queue and other areas. If a match is not found, the packet will be sent to the controller. If there's no field which will be matched, the packets are going to be dropped. Since its header does not have any of the field which is mentioned in Table 1 and considered to be an invalid or illegal packet. Our solution works with IP address which exists in the table.

5

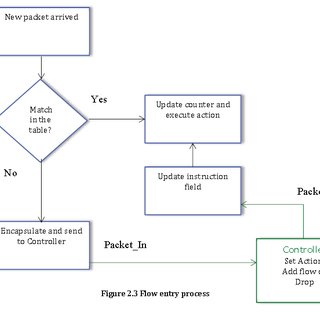
|  |
| --- |
| **Header Field** |
| Ingress Port |
| Metadata |
| Ether src |
| Ether dst |
| Ether type |
| VLAN id |
| VLAN priority |
| MPLS label |
| MPLS traffic class |
| IPv4 src |
| IPv4 dst |
| IPv4 proto/ ARP opcode |
| IPv4 ToS bits |
| TCP / UDP / SCTP src port ICMP Type |
| TCP/ UDP / SCTP dst ICMP code |

**Table 1: Packet header match fields**

New packets can be sent to the controller or the switch which buffer the payload and send only the header. We will refer to it as Packet\_In. Considering the number of switches, time of day, packet length, priority and other factors, the controller must process these packets and send a response with an action to deal with that packet and the packets coming after from the same source.

Figure 3 shows the process of flow entry. If a packet is finding to be marked with drop action, a flow entry will be added. Any packet matching with the flow will be dropped. If packets are not received by flow, the flow entry will time out and after some time it will be removed.

6



**Figure 3: Flow Entry Process**

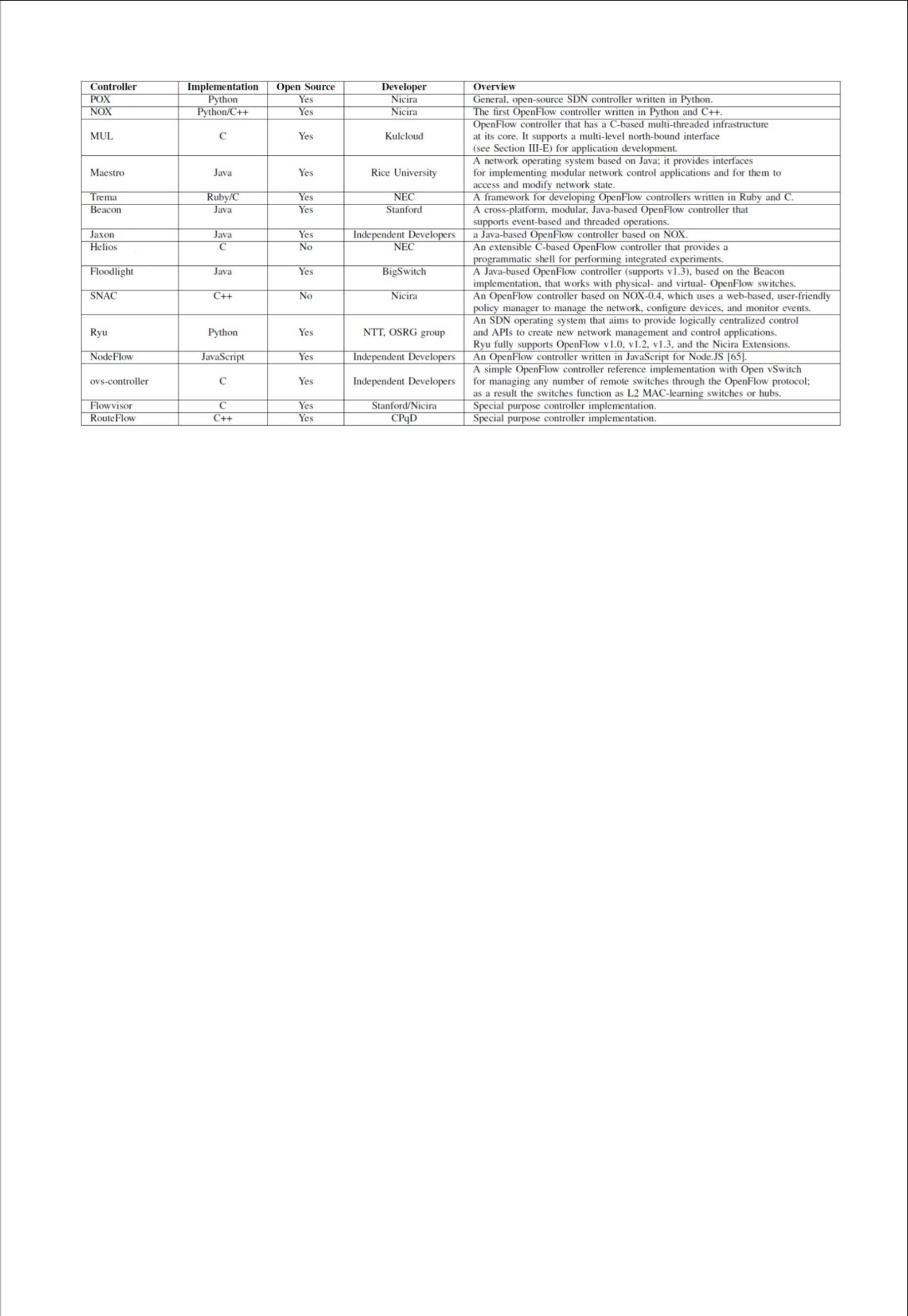
**3.3 SDN Controllers**

As controllers are also known as Brain of SDN. For communicating OpenFlow switches southbound APIs is used and the northbound APIs for SDN applications. Using different modules controllers performs many tasks.

Currently several well-known controller implementations are available that are open source and are written with different programming languages such as python, C++ and Java. Table 2 summarizes the current implementations of different available controllers. The table provides a brief overview of the controller characteristics.

POX is a lightweight OpenFlow controller that's an appropriate platform for SDN research, academic work, education, and experimentation.

7



**Table 2: Current Controller Implementations compliant with the OpenFlow Standard**

**3.4 Distributed Denial of Service (DDoS) Attack**

The Distributed Denial of Service (DDoS) is a well-known malicious attempt to sending heavy traffic for exhausting resources of a computer or a network of computers. The two main goals of the attacker are:

1. Bandwidth depletion.
2. Resource exhaustion.

DDoS attack starts from an attacker planting a Botnet which is a code in compromised PCs. During attack, these codes, and a stream of traffic is directed towards the victim. A more complex attack uses a thin layer of compromised PCs called handler to regulate a bigger number of PCs called agents. The agents are used for generating the attack traffic. Botnet is used for making the attack more concentrated and keeps the perpetrator hidden behind the scene. Figure 4 shows a simple example of DDoS Attack route.

8

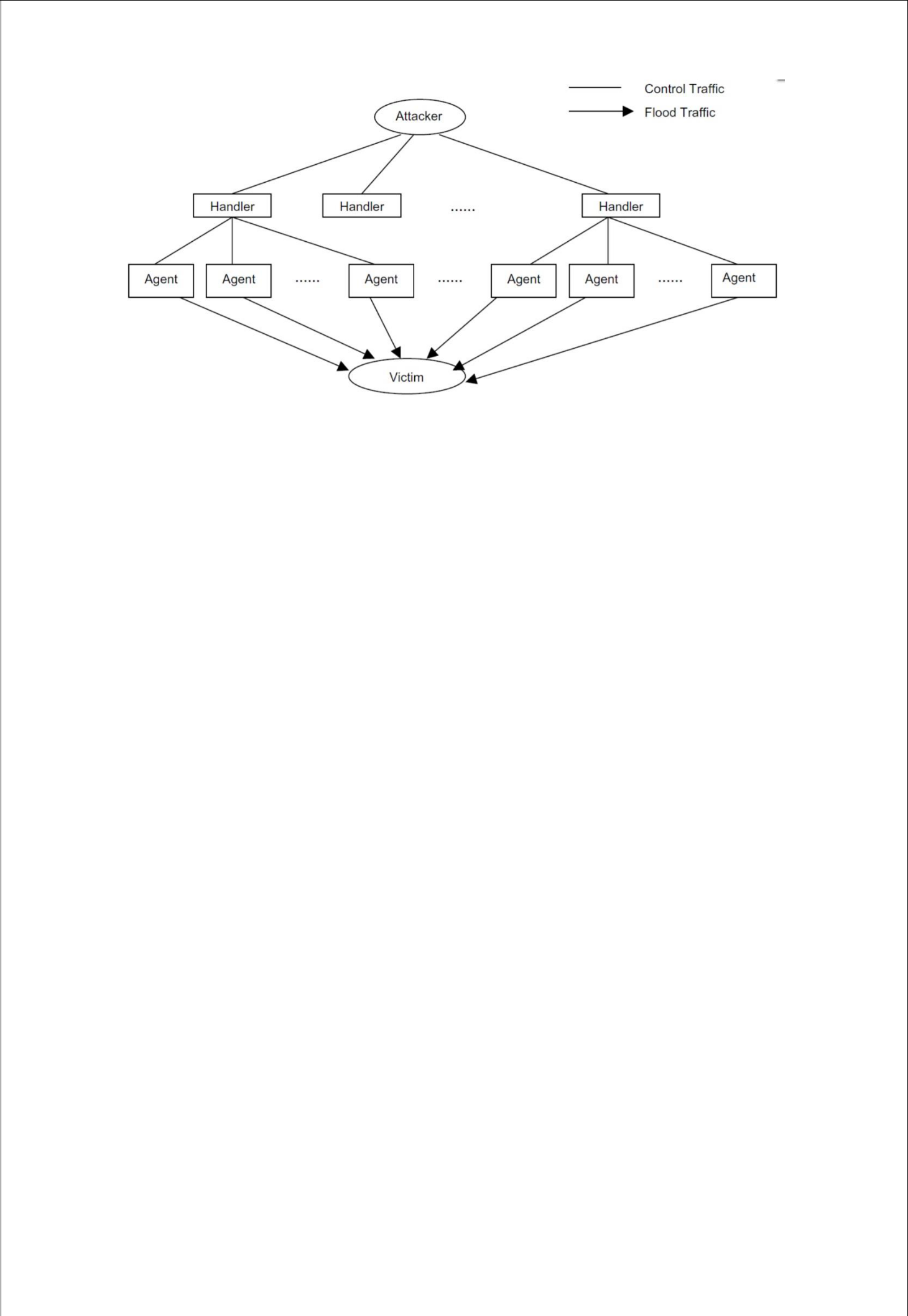
**Figure 4: DDoS Attack Structure**

Handlers are captured by the attackers to find out which hosts can participate within the attacking process or use them to update the codes on the hosts. Each host is controlled by one or variety of handlers and this communication can be run over UDP, TCP or ICMP protocols.

Hackers use scanning to seek out vulnerable computers within the network. Scanning are often random, supported successful list, local subnet scanning or supported an algorithm designed by the hacker.

1. Application: targeting an application on a host to deny legitimate use.
2. Host: making a host unreachable.
3. Resource: overwhelming a server to stay it sure to the continual stream of fake requests.
4. Network: sending an outsized volume of traffic to a network to exhaust the bandwidth.
5. Infrastructure: simultaneously targeting a website name server in several places.

The attacks are often categorized as application, host, resource, network and infrastructure attacks. A few sorts of DDoS are very frequent in recent years. The pattern of attacks shows that hacker is using certain sorts of methods for launching attacks. Some of these methods will be looked at next. Google and Arbor networks have dedicated a website that follows the attacks and collects statistics from all over the world.

9

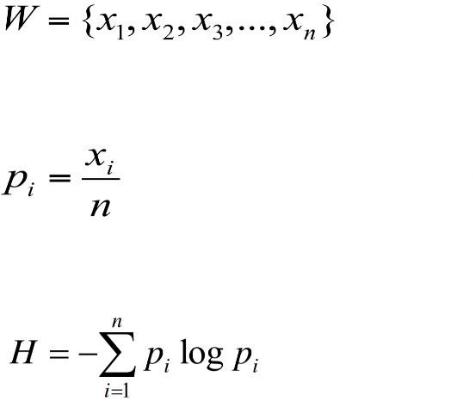
**4.Early Detection of DDoS using Entropy**

In this section, the proposed method for early detection will be explained. Due to the limited resources of the controller, an early detection should be within the first few hundred packets of the attack. First, entropy, its formulas and computation will be discussed.

**4.1 Why Entropy?**

Entropy has an ability to find the randomness that’s why we used it for our detection. The higher the randomness the higher is the entropy and vice versa.

Let W be a set of data with n elements and x is an event in the set. Then, the probability of X happening in W. To measure the entropy, referred to as H, we calculate the probability of all elements in the set and sum.



If all elements have equal probabilities, then entropy value will be highest. The entropy will be lower, if an element appears more than others. The size of W is window size. If a continuous stream of incoming data, in our case the data is packet header, it will be divided into equal sets called windows. Each element and its occurrence are counted in the window. For instance, if the window has 64 elements and, all elements appear only once, the entropy will be 1.80. If one element appears 10 times, the entropy will be 1.64. This property of entropy will be used for calculating the randomness in the SDN controller.

10

The source address is always new, when packets arrive at the controller. This is the reason they come to the controller. They are passed on to the controller, when there has not been an instance of them in the table of the switch.

New packets coming to the controller has destination host is in the network of the controller. The network consists of the switches and hosts that are connected to it. Knowing the packet is new and the destination is in the network by calculating the entropy based on a window size, we can find randomness. Maximum entropy occurs when each packet is destined to precisely one host. Minimum entropy occurs when all the packets in a window are destined for a single host.

Being able to find randomness and have minimum and maximum based on entropy makes it a suitable method for DDoS detection is SDN. Using entropy, it is possible to ascertain its value drop when an outsized number of packets are attacking one host.

**4.2 Early Detection**

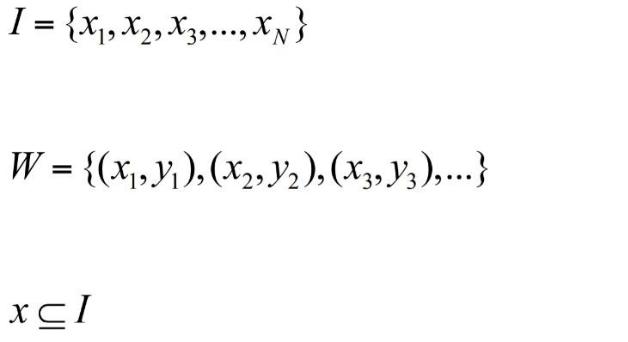
Controller can collect statistics from all OpenFlow switches to detect inactive flows. These flows will be removed if they do not receive any packets for a period. This time period is called time-out in the OpenFlow specification and it can be set to different values.

For a lightweight solution, we propose adding another set of statistics to the controller, it is the entropy of the destination IP address in the controller. The function will determine if a higher than normal rate of incoming packets destined to the same destination.

In the new function, every 50 Packet\_In messages will be analysed for their destination IP address and the entropy of the list will be calculated. The calculated entropy will be compared to a threshold. If the calculated entropy is less than the threshold and it persists for a minimum of 5 consecutive entropy periods and then it will be considered an attack. Detection within 5 entropy periods is 250 packets in the attack, which gives the network and early alert of attack. We tested with values one to five consecutive periods and five has the lowest false negative and positive for early detection.

11

Let I be the IP addresses of all hosts connected to the network, and W be the window containing new packets’ destination IP address x and their number of occurrence y.

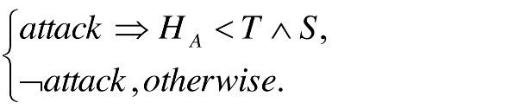


Then the entropy will be at its maximum if each destination IP is unique:



If the above condition doesn't hold, then some IP addresses have appeared quite once.

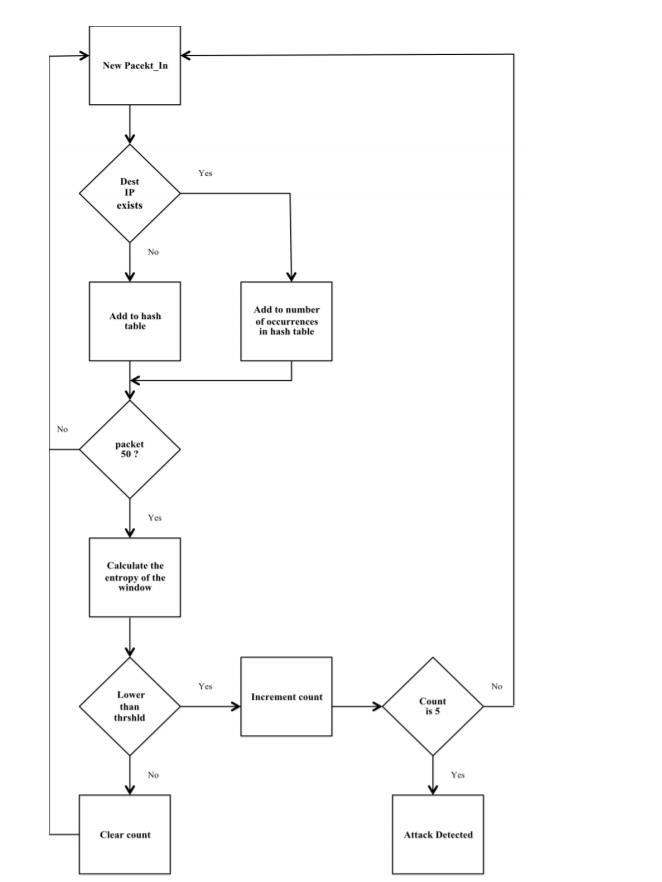
Two conditions are chosen to be the trigger for an attack in our method. One is the threshold and the other is the continuity of the attack. There could be glitches within the network that cause irregularity in normal traffic. If a link to a switch goes down or some hosts become temporarily unavailable, the entropy might fall and trigger a false positive. To avoid false positives of this type, we propose a limit for the number of consecutive low entropy windows. Based on that, the condition for declaring an attack where T is the threshold, S is an array of five windows with lower than T entropy. An attack happened if entropy of attack HA, is smaller than the threshold and, having five consecutive lower than threshold entropies is true. Otherwise, there is no attack:



12

We keep the consecutive-less-entropy is set to be 0. If we found any result in consecutive 5 as higher than entropy is reset to 0.Attack is determined only if the we get less Entropy for 5 consecutive and decreasing consecutively.

The algorithm in Figure 5 is done with the addition of two function is the controller. The first one is called when a new Packet\_In message arrives which accepts the destination IP address as an argument. The second function is used to computes the entropy. This is the only addition to the controller. Just to show the amount of code change in the controller the two functions are shown below.



**Figure 5: DDoS Detection Algorithm**

13

**4.3 Software and Tools Used**

**4.3.1 Controller**

Choosing controller is the first thing in our experiment. There are few famous controllers available. The one that is used in this experiment is POX. Pox is widely used for experiments, it's fast, lightweight and designed as a platform so a custom controller is often built on top of it. It is an improved version of its predecessor NOX, and both are running on Python. POX works on Linux, Mac OS and windows, and its topology discovery. For completeness, three other controllers should be mentioned. Floodlight and Beacon is another widely used controller that is opensource and written in Java.

**4.3.2 Network Emulator**

Mininet is the network emulator that is used for this experiment. It is the quality network emulation tool which will be used for SDN. It can prototype a network on a laptop or PC by using kernel namespace feature. Network namespace provides individual processes with their own network interfaces, ARP tables and routing tables.

Mininet makes use of this feature of the kernel which uses process-based virtualization to run switches and hosts on the kernel. Large networks with different topologies can be emulated and tested. In fact, the code developed in Mininet emulation can be moved to a real production network.

Creating a network in Mininet is as easy of entering the command mn to have a network with 1 switch, 2 hosts and a NOX controller. NOX is the default controller of Mininet. But we use POX controller.

**4.3.3 Packet Generation**

Scapy is used for Packet Generation. Scapy is a powerful tool for packet generating, scanning, sniffing, attacking and packet forging. Scapy is employed here to generate UDP packets and spoof the source IP address of the packets.

Python programming language is employed in POX. The code for generating random source IP addresses and host IP addresses is in Python. The function “randrange” is used which is

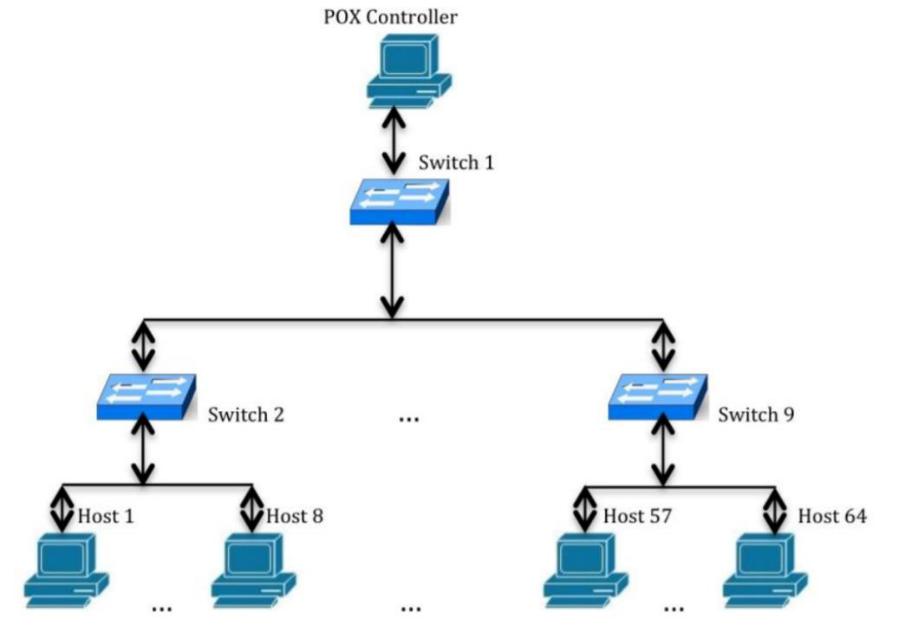
14

inheriting the function “random”. This function produces a consistent random float within the range [0.0, 1.0). It creates an extended period of random number generation which can end in generating random numbers with uniform distribution. These numbers are joined together to make spoofed source IP addresses. Two other parameters that we set in Scapy are: sort of packets and interval of packet generation. UDP packets are used for both attack traffic and normal traffic. The code for generating normal and attack is shown in Appendix.

**4.3.4 Network Setup**

The experiment was done on an ASUS laptop with an Intel Core i5-8300H CPU, 8GB of RAM and 1TB SSHD, and 10/100/100/1000Mbitps network interface. The operating system is Linux Ubuntu 20.04 and Mininet version 2.2.0 was run native on Linux. Mininet 2.2.0 supports OpenFlow version 1.0. Using Mininet, a tree-type network of depth 2 with 9 switches and 64 hosts was created. Figure 12 shows the network. Open Virtual Switch (OVS) was used for network switches. OVS may be a software switch that runs both on hardware and software.

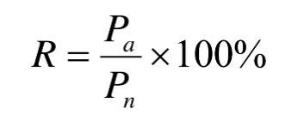
For this work, there is no difference between OpenFlow and OVS switch. Both do the same job, and both are supported in Mininet. In Figure 6, all switches refer to OpenFlow enabled switches. The L3\_learning module of POX was used for the controller.



**Figure 6: Experiment Network with 9 switches and 64 hosts**

15

To find the range for an optimal threshold, we ran a series of experiments to ascertain the effect of an attack on the entropy. The experiments cover an attack to at least one host. To compare different rates of incoming packets, we controlled the speed of normal and attack traffic to extend and reduce the intensity of DDoS on the controller. R is rate of incoming attack packets to normal traffic attacks. Where Pa and Pn are the amount of attack packets and normal traffic packets respectively.

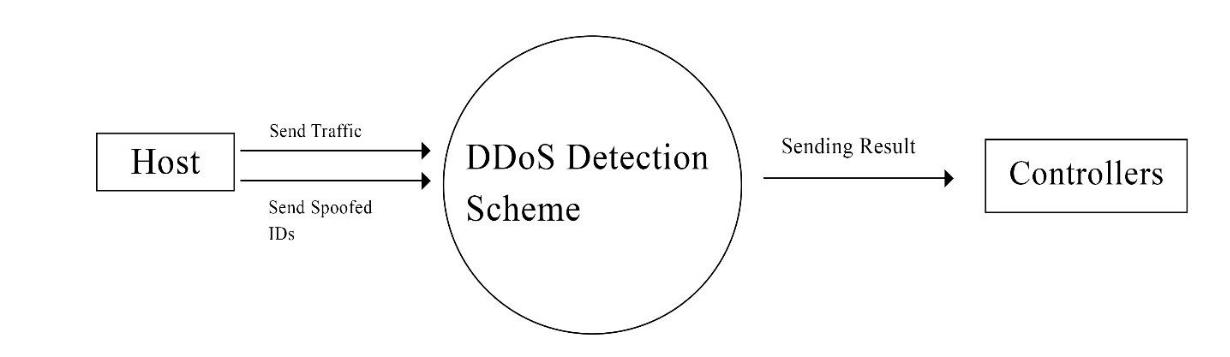


We ran a 25% rate attack on one host for 25 times to seek out an appropriate threshold. This threshold is used for maximum entropy for attack detection. We ran normal traffic from one host to another and entropy is calculated is 25 times and mean is taken for all 25 tests as threshold Entropy.

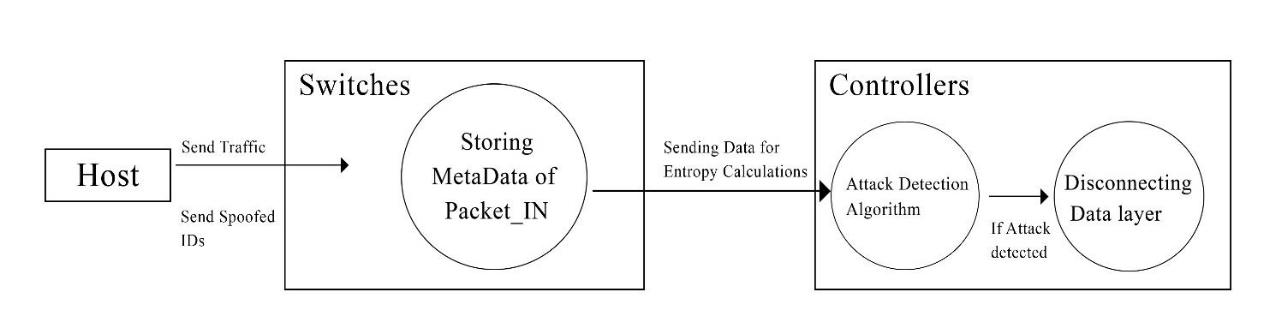
16

1. **DATA FLOW DIAGRAM**

**5.1 Level 0 DFD**

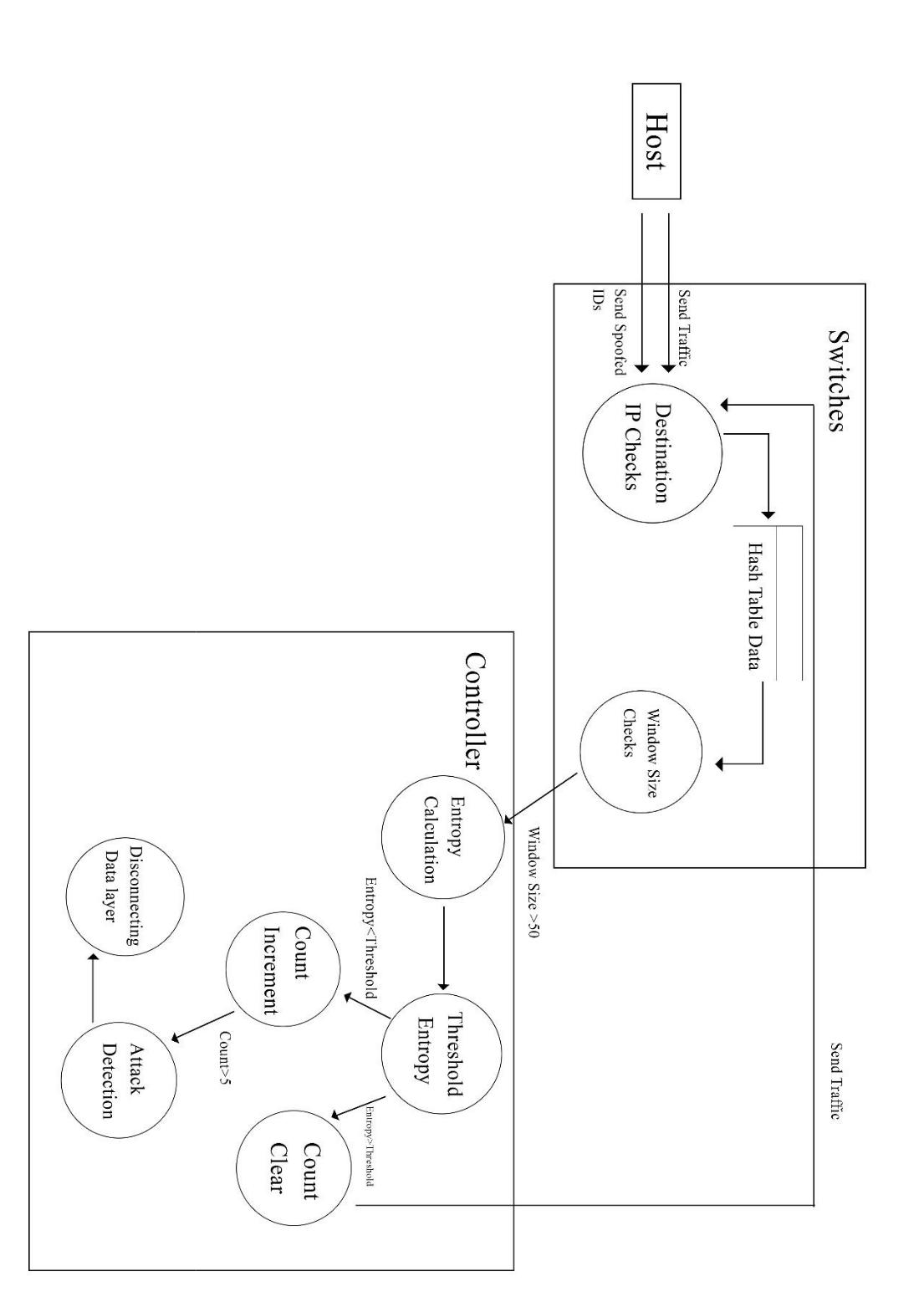
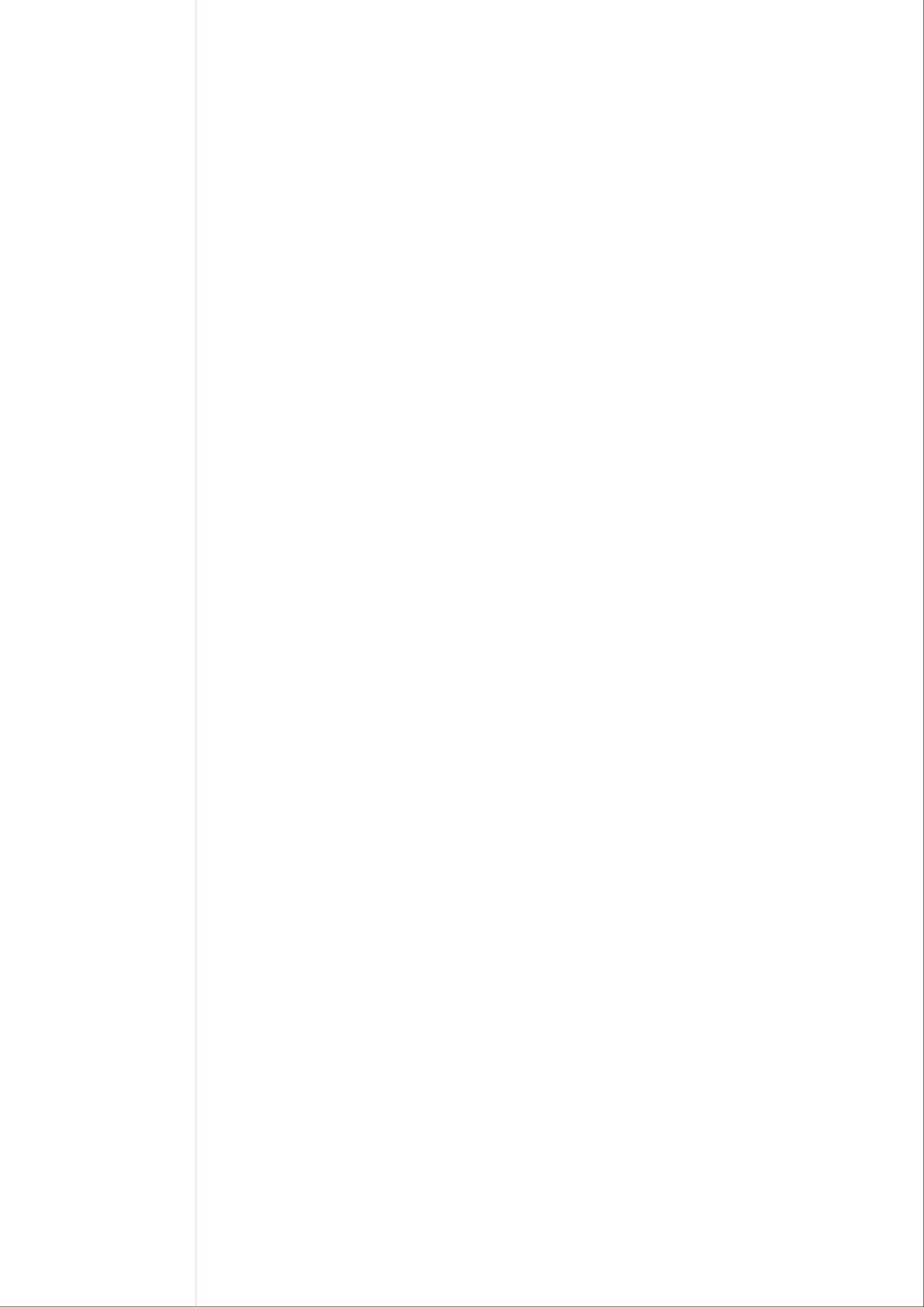


**5.2 Level 1 DFD**



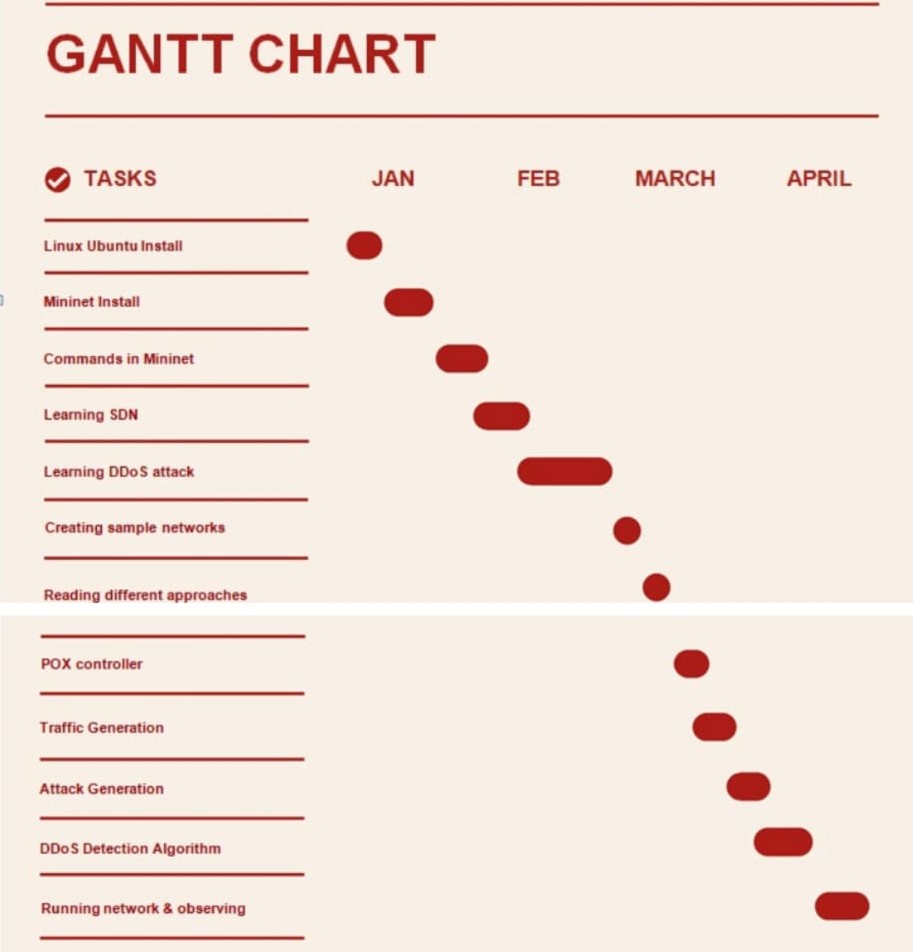
17

**5.3 Level 2 DFD**



18

1. **GANTT CHART**

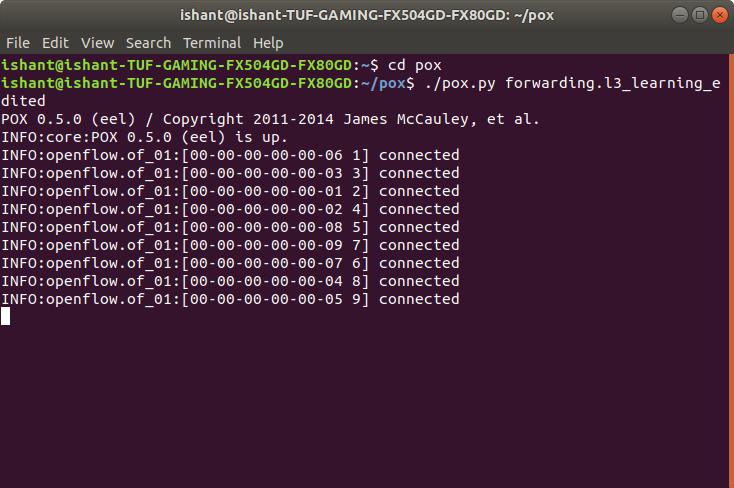


19

1. **EXPERIMENT WALKTHROUGH**

In this experiment we are trying to distinguish between DDoS and normal traffic. So, using Mininet, a tree-type network of depth two with five switches and 16 hosts was created, upon which we would test this model. POX is used as the controller of choice. The normal and attack packets are generated from a fixed number of hosts (IPs) outside the network we created. The detector module deployed on top of the controller captures the packets and determines if there is an attack.

1. First, we copy our detection.py in pox/pox/forwarding where we edited l3\_forwarding to capture packets and detect entropy.
2. Make a copy of the attack.py and traffic.py into accessible location in your mininet/custom folder.
3. We open terminal and activate our pox controller.



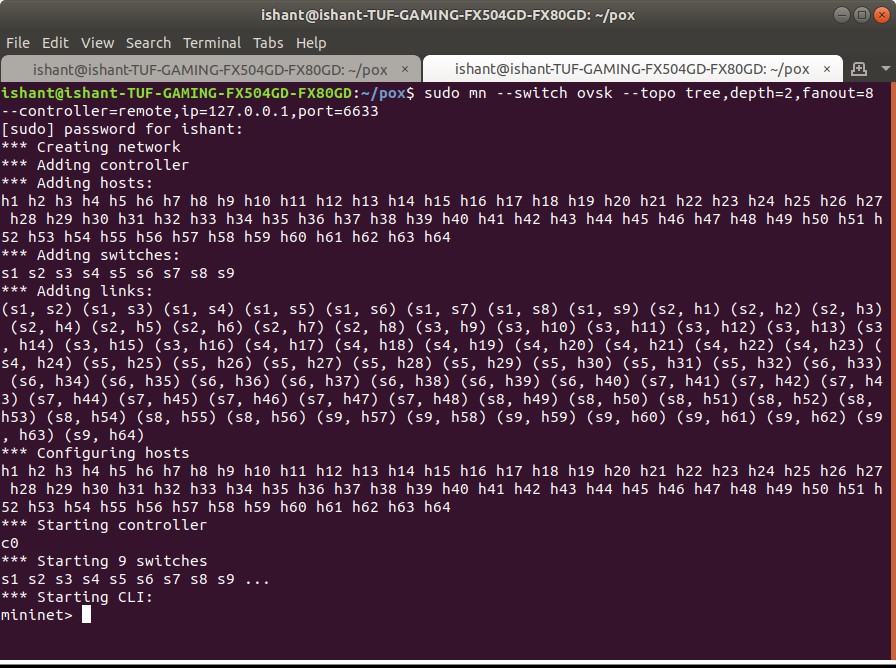
20

1. In another terminal tab we create an mininet topology using:

Sudo mn –switch ask –top tree, depth=2, fanout=8 –controller=remote, ip=127.0.0.1, port=6633

It means it creates a mininet topology with controller with port address=6633 which connects it with pox controller and IP address with 127.0.0.1 and fanout of 8 creating 9 open vSwitch kernel which shows same function as OpenFlow switches, where s1 connects with c0(controller) and 8 other switches and all other 8 switches (s2, ….., s9) connects with each 8 hosts(h1, h2, …………………., h63, h64).

Each host has IP address as 10.0.0.1………………64.

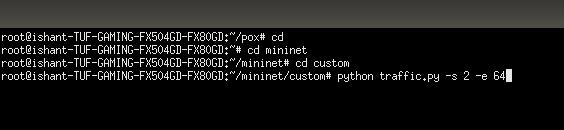


1. Open the Xterm of host h1 and launch the normal traffic in the mininet network. Let it run for few minutes to generate entropy values.

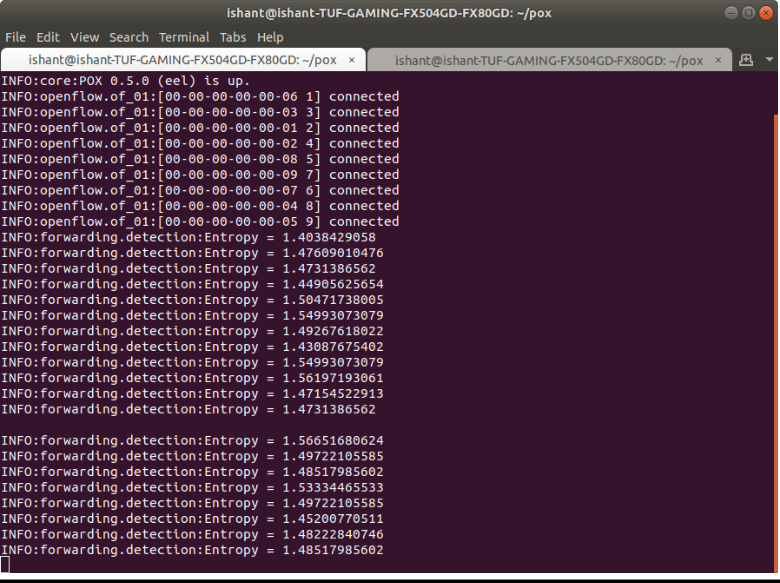
$python traffic.py -s X -e Y

21

Its stats that as starting point will be with IP address10.0.0.X and ending host will be 10.0.0.Y



6. We observe Entropy Values in POX terminal Tab.

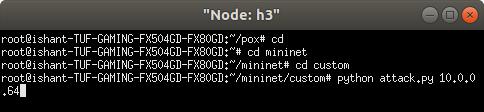
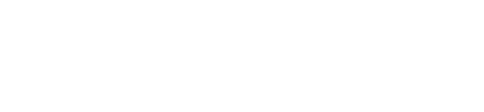
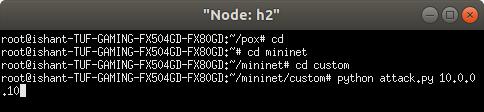
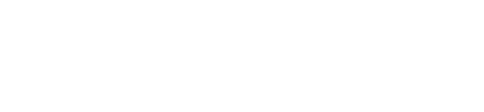


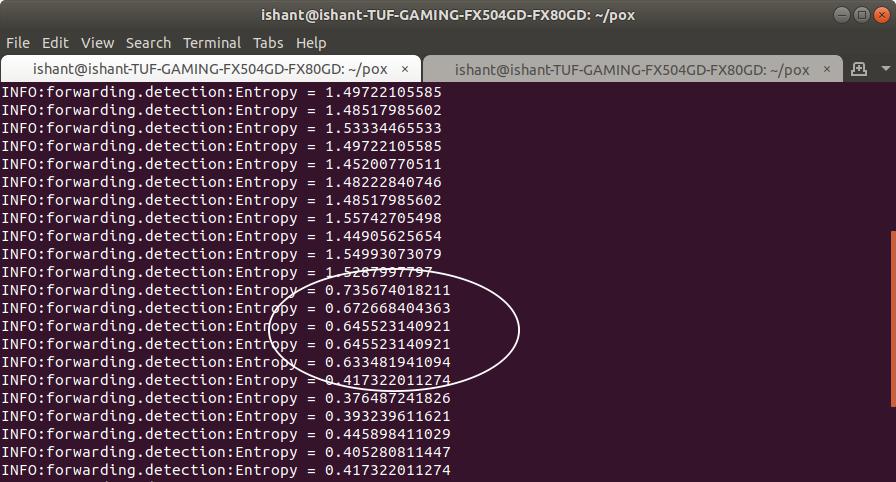
22

1. Now we attack using Xterm of h2 and h3 to observe Entropy Value.

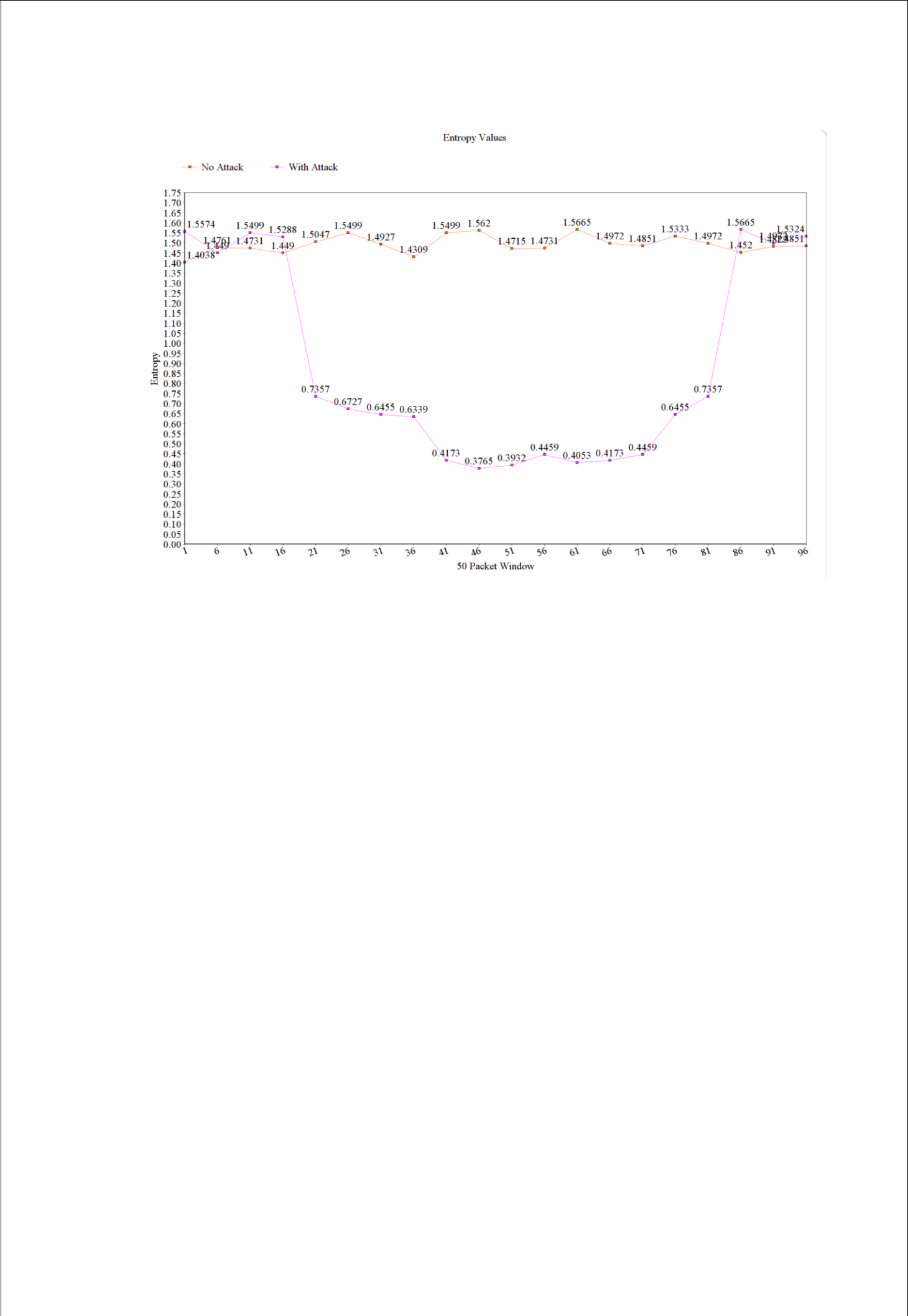
$python attack.py 10.0.0.Z

It takes one value as input to create traffic in destination IP address.



8. As we attack, we saw a drop in entropy value which helps us to detect DDos Attack.

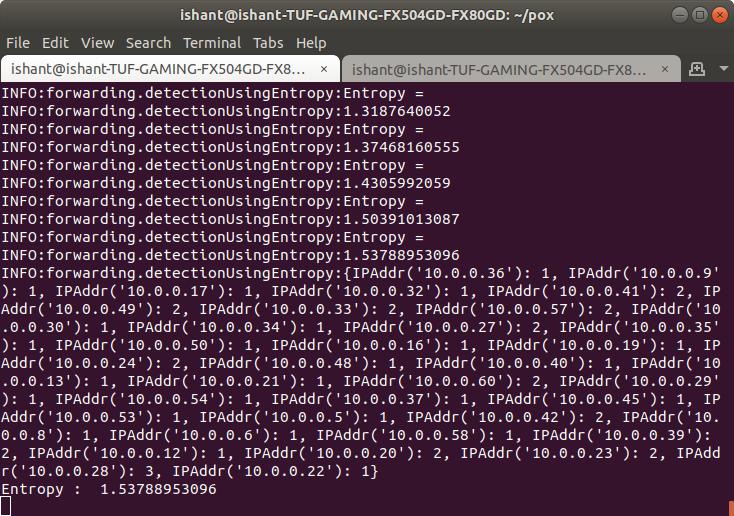
23

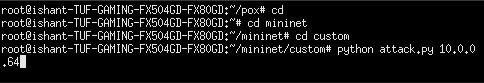


24

1. **RESULT**

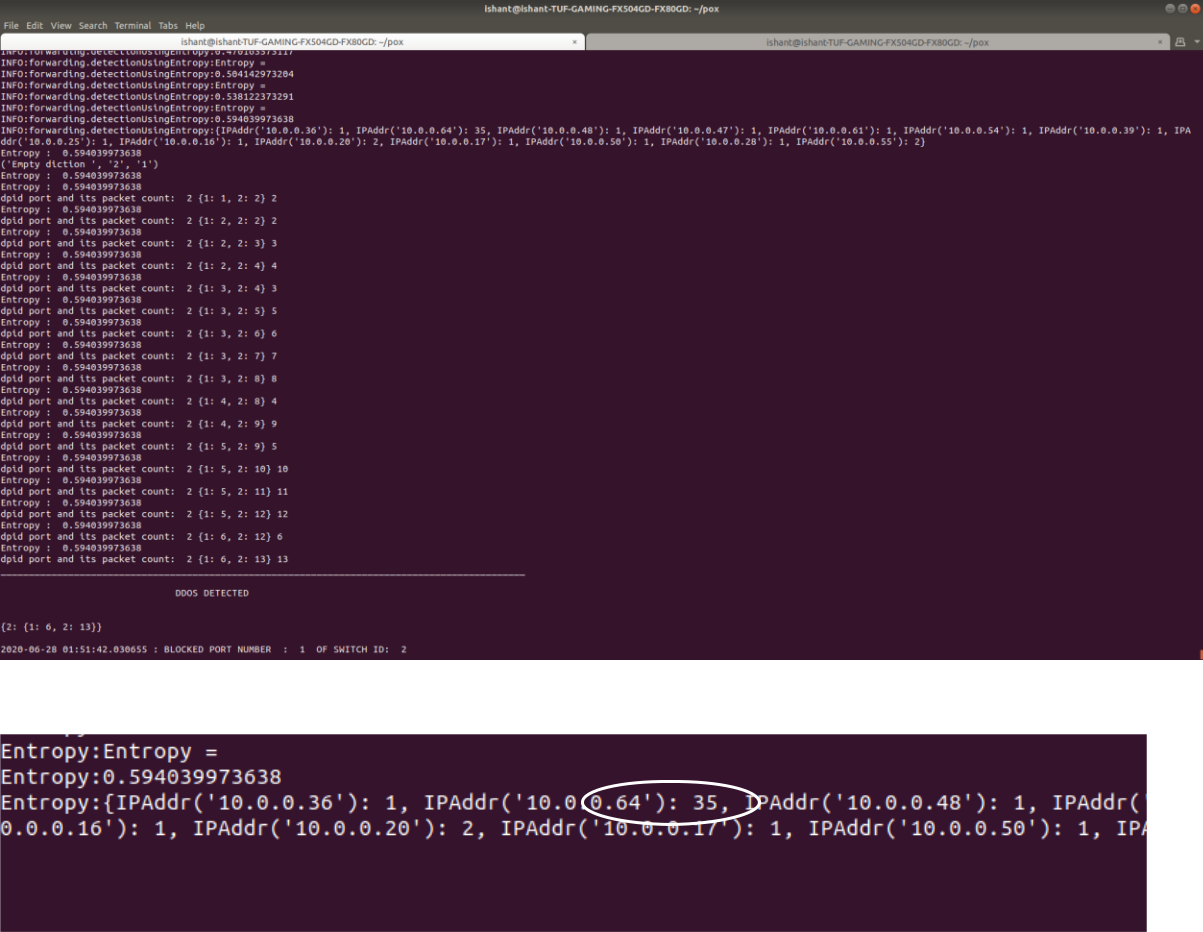
In this project, we first normal traffic to create a threshold entropy and then we attack and since its increases traffic in the victim host it decreases the entropy value and if threshold entropy is larger, we check it for 5 times and DDoS detected with giving port id of attacker.



It creates a 1.5378 of entropy as threshold entropy as no traffic attack. After this we again run traffic and attack on host 64.

25

Then it detects DDoS and show increasing packets in port 64.



26

1. **CONCLUSION**

As a result of the experiment, Entropy 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.

Protecting the operating system of SDN (i.e. the controller) by detecting DDoS attacks were the center of this research. The challenge in detecting any threats to the controller is early detection. Although the term “early” can be used loosely in detecting an attack, we quantified the early detection to the first 250 packets of traffic as minimum and 500 packets maximum.

There are many different methods for detecting attacks and each method is used differently. In this research, we focused on a solution that works particularly well for SDN, based on its specifications, points of strength and limitations. We made use of the fact that SDN specification dictates the forwarding of new packets to the controller. We took into account the abilities of the controller and its broad view of the whole network and used that for adding entropy statistics collection.

Finally, understanding the importance of keeping the controller always connected to the network, we came up with a solution to detect any threat at its very beginning. Entropy has been used in DDoS detection in non-SDN network but, to the best of our knowledge, it has not been used in SDN and this is the first solution of its kind in SDN. One contribution of this research is covering controller security in SDN. The topic of SDN is new and the reason might be limited deployment of this structure as a production network. The contributions of this research are:

* Showed how DDoS attack can overwhelm the controller in SDN architecture.
* Proposed a lightweight and fast DDoS detection mechanism based on entropy, to protect the controller.
* Implement the proposed mechanism using Mininet and POX controller.
* Showed the effectiveness of the solution through extensive simulations

27

**REFERENCES**

1. Qiao Yan, F. Richard Yu, Senior Member, IEEE, Qingxiang Gong, and Jianqiang Li: Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 18, NO. 1, FIRST QUARTER 2016
2. Ijaz Ahmad, Suneth Namal, Mika Ylianttila, Senior Member, IEEE, and Andrei Gurtov, Senior Member, IEEE: Security in Software Defined Networks: A Survey IEEE COMMUNICATION SURVEYS & TUTORIALS, VOL. 17, NO. 4, FOURTH QUARTER 2015
3. Prashant Kumar, Meenakshi Tripathi, Member, IEEE, Ajay Nehra, Mauro Conti , Senior Member, IEEE, and Chhagan Lal: SAFETY: Early Detection and Mitigation of TCP SYN Flood Utilizing Entropy in SDN IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, VOL. 15, NO. 4, DECEMBER 2018
4. Kübra Kalkan, Levent Altay, Gürkan Gür, and Fatih Alagöz: JESS: Joint Entropy-Based DDoS Defense Scheme in SDN IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 36, NO. 10, OCTOBER 2018
5. Zhenpeng Liu1,2, Yupeng He1 , Wensheng Wang3 , Bin Zhang2,\*: DDoS Attack Detection Scheme Based on Entropy and PSO-BP Neural Network in SDN
6. Maryam Kia: Early Detection and Mitigation of DDoS Attacks In Software Defined Networks Ryerson University Toronto, Ontario, Canada, 2015
7. POX Controller: https://github.com/noxrepo/pox/
8. Mininet Install and commands: <http://mininet.org/download/>

28

**APPENDIX**

**A. Detection Entropy Code**

import math

from pox.core import core

log = core.getLogger()

class Entropy(object):

count = 0

entDic = {}

ipList = []

dstEnt = []

value = 1

def statcolect(self, element):

#print "Self values"

#print "count is " + str(self.count)

#print "Length of IP list is"

#print len(self.ipList)

#print "\*\*\*\*\*\*\*\*\*"

l = 0

self.count +=1

self.ipList.append(element)

if self.count == 50:

for i in self.ipList:

l +=1

if i not in self.entDic:

self.entDic[i] =0

self.entDic[i] +=1

self.entropy(self.entDic)

log.info(self.entDic)

self.entDic = {}

self.ipList = []

l = 0

self.count = 0

def entropy (self, lists):

#print "Entropy called"

l = 50

elist = []

for k,p in lists.items():

29

'''

log.info("p is")

log.info(p)

log.info("P is obtained from")

log.info(k)

log.info("l is")

log.info(l)

'''

c = p/float(l)

#log.info("Value of c is ")

#log.info(c)

c = abs(c)

elist.append(-c \* math.log(c, 10))

log.info('Entropy = ')

log.info(sum(elist))

#log.info("\*\*\*\*")

self.dstEnt.append(sum(elist))

if(len(self.dstEnt)) == 80:

print self.dstEnt

self.dstEnt = []

self.value = sum(elist)

def \_\_init\_\_(self):

pass

**B. Traffic Generation Code**

def generateSourceIP():

#not valid for first octet of IP address

not\_valid = [10, 127, 254, 1, 2, 169, 172, 192]

#selects a random number in the range [1,256)

first = randrange(1, 256)

while first in not\_valid:

first = randrange(1, 256)

#eg, ip = "100.200.10.1"

ip = ".".join([str(first), str(randrange(1,256)), str(randrange(1,256)), str(randrange(1,256))])

return ip

#start, end: given as command line arguments. eg, python traffic.py -s 2 -e 65 def generateDestinationIP(start, end):

30

first = 10

second = 0;

third = 0;

#eg, ip = "10.0.0.64"

ip = ".".join([str(first), str(second), str(third), str(randrange(start,end))])

return ip

def main(argv):

#print argv

#getopt.getopt() parses command line arguments and options 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()

#open interface eth0 to send packets

interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()

for i in xrange(1000):

packets = Ether() / IP(dst = generateDestinationIP (start, end), src = generateSourceIP ()) / UDP(dport = 80, sport = 2)

print(repr(packets))

#rstrip() strips whitespace characters from the end of interface sendp(packets, iface = interface.rstrip(), inter = 0.1)

31

if \_\_name\_\_ == '\_\_main\_\_':

main(sys.argv)

**C. Attack Code**

def generateSourceIP():

not\_valid = [10, 127, 254, 255, 1, 2, 169, 172, 192]

first = randrange(1, 256)

while first in not\_valid:

first = randrange(1, 256)

#print first

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):

launchAttack()

time.sleep (10)

def launchAttack():

#eg, python attack.py 10.0.0.64, where destinationIP = 10.0.0.64

destinationIP = sys.argv[1:]

#print destinationIP

interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()

for i in xrange(0, 500):

packets = Ether() / IP(dst = destinationIP, src = generateSourceIP()) / UDP(dport = 1, sport = 80)

print(repr(packets))

#send packets with interval = 0.025 s

sendp(packets, iface = interface.rstrip(), inter = 0.025)

32

if \_\_name\_\_=="\_\_main\_\_":

main()

**D. L3\_Learning Updated Code**

import os

import datetime

from pox.core import core

import pox

from pox.lib.packet.ethernet import ethernet, ETHER\_BROADCAST from pox.lib.packet.ipv4 import ipv4 from pox.lib.packet.arp import arp

from pox.lib.addresses import IPAddr, EthAddr from pox.lib.util import str\_to\_bool, dpid\_to\_str from pox.lib.recoco import Timer

import pox.openflow.libopenflow\_01 as of

from pox.lib.revent import \*

import itertools

import time–

from .detectionUsingEntropy import Entropy

diction = {}

ent\_obj = Entropy()

set\_Timer = False

defendDDOS=False

log = core.getLogger()

FLOW\_IDLE\_TIMEOUT = 10

ARP\_TIMEOUT = 60 \* 2

MAX\_BUFFERED\_PER\_IP = 5

MAX\_BUFFER\_TIME = 5

class Entry (object):

def \_\_init\_\_ (self, port, mac):

self.timeout = time.time() + ARP\_TIMEOUT

self.port = port

self.mac = mac

def \_\_eq\_\_ (self, other):

if type(other) == tuple:

return (self.port,self.mac)==other

else:

return (self.port,self.mac)==(other.port,other.mac)

33

def \_\_ne\_\_ (self, other):

return not self.\_\_eq\_\_(other)

def isExpired (self):

if self.port == of.OFPP\_NONE: return False

return time.time() > self.timeout

def dpid\_to\_mac (dpid):

return EthAddr("%012x" % (dpid & 0xffFFffFFffFF,))

class l3\_switch (EventMixin):

def \_\_init\_\_ (self, fakeways = [], arp\_for\_unknowns = False, wide = False):

self.fakeways = set(fakeways)

self.wide = wide

self.arp\_for\_unknowns = arp\_for\_unknowns

self.outstanding\_arps = {}

self.lost\_buffers = {}

self.arpTable = {}

self.\_expire\_timer = Timer(5, self.\_handle\_expiration, recurring=True)

core.listen\_to\_dependencies(self)

def \_handle\_expiration (self):

empty = []

for k,v in self.lost\_buffers.iteritems():

dpid,ip = k

for item in list(v):

expires\_at,buffer\_id,in\_port = item

if expires\_at < time.time():

v.remove(item)

po = of.ofp\_packet\_out(buffer\_id = buffer\_id, in\_port = in\_port)

core.openflow.sendToDPID(dpid, po)

if len(v) == 0: empty.append(k)

for k in empty:

del self.lost\_buffers[k]

def \_send\_lost\_buffers (self, dpid, ipaddr, macaddr, port):

if (dpid,ipaddr) in self.lost\_buffers:

bucket = self.lost\_buffers[(dpid,ipaddr)]

del self.lost\_buffers[(dpid,ipaddr)]

log.debug("Sending %i buffered packets to %s from %s"

* (len(bucket),ipaddr,dpid\_to\_str(dpid)))

34

po = of.ofp\_packet\_out(buffer\_id=buffer\_id,in\_port=in\_port)

po.actions.append(of.ofp\_action\_dl\_addr.set\_dst(macaddr))

po.actions.append(of.ofp\_action\_output(port = port))

core.openflow.sendToDPID(dpid, po)

def \_handle\_openflow\_PacketIn (self, event):

dpid = event.connection.dpid

inport = event.port

packet = event.parsed

global set\_Timer

global defendDDOS

global blockPort

timerSet =False

global diction

def preventing():

global diction

global set\_Timer

if not set\_Timer:

set\_Timer =True

if len(diction) == 0:

print("Empty diction ",str(event.connection.dpid), str(event.port)) diction[event.connection.dpid] = {} diction[event.connection.dpid][event.port] = 1

elif event.connection.dpid not in diction: diction[event.connection.dpid] = {} diction[event.connection.dpid][event.port] = 1

else:

if event.connection.dpid in diction:

if event.port in diction[event.connection.dpid]:

temp\_count=0

temp\_count =diction[event.connection.dpid][event.port] temp\_count = temp\_count+1 diction[event.connection.dpid][event.port]=temp\_count #print

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

print "dpid port and its packet count: ", str(event.connection.dpid), str(diction[event.connection.dpid]), str(diction[event.connection.dpid][event.port])

#print

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

else:

diction[event.connection.dpid][event.port] = 1

def \_timer\_func ():

global diction

35

global set\_Timer

if set\_Timer==True:

for k,v in diction.iteritems():

for i,j in v.iteritems():

if j >=5:

print

“\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_”

print “\n DDOS DETECTED \n”

print “\n”,str(diction)

print “\n”,datetime.datetime.now(),”: BLOCKED PORT NUMBER : “,str(i), “OF SWITCH ID : “, str(k)

print

“\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_”

os.\_exit(0)

dpid = k

msg = of.ofp\_packet\_out(in\_port=i)

core.openflow.sendToDPID(dpid,msg)

diction={}

if not packet.parsed:

log.warning("%i %i ignoring unparsed packet", dpid, inport)

return

if dpid not in self.arpTable:

self.arpTable[dpid] = {}

for fake in self.fakeways:

self.arpTable[dpid][IPAddr(fake)] = Entry(of.OFPP\_NONE, dpid\_to\_mac(dpid))

if packet.type == ethernet.LLDP\_TYPE:

return

if isinstance(packet.next, ipv4):

log.debug("%i %i IP %s => %s", dpid,inport, packet.next.srcip,packet.next.dstip)

ent\_obj.collectStats(event.parsed.next.dstip)

print "Entropy : ",str(ent\_obj.value)

if ent\_obj.value <1.0:

preventing()

if timerSet is not True:

Timer(1, \_timer\_func, recurring=True)

timerSet=False

else:

timerSet=False

self.\_send\_lost\_buffers(dpid, packet.next.srcip, packet.src, inport)

if packet.next.srcip in self.arpTable[dpid]:

36

if self.arpTable[dpid][packet.next.srcip] != (inport, packet.src): log.info("%i %i RE-learned %s", dpid,inport,packet.next.srcip) if self.wide:

msg = of.ofp\_flow\_mod(command=of.OFPFC\_DELETE) msg.match.nw\_dst = packet.next.srcip msg.match.dl\_type = ethernet.IP\_TYPE event.connection.send(msg)

else:

log.debug("%i %i learned %s", dpid,inport,packet.next.srcip)

self.arpTable[dpid][packet.next.srcip] = Entry(inport, packet.src)

dstaddr = packet.next.dstip

if dstaddr in self.arpTable[dpid]:

prt = self.arpTable[dpid][dstaddr].port

mac = self.arpTable[dpid][dstaddr].mac

if prt == inport:

log.warning("%i %i not sending packet for %s back out of the "

"input port" % (dpid, inport, dstaddr))

else:

log.debug("%i %i installing flow for %s => %s out port %i"

% (dpid, inport, packet.next.srcip, dstaddr, prt))

actions = []

actions.append(of.ofp\_action\_dl\_addr.set\_dst(mac))

actions.append(of.ofp\_action\_output(port = prt))

if self.wide:

match = of.ofp\_match(dl\_type = packet.type, nw\_dst = dstaddr)

else:

match = of.ofp\_match.from\_packet(packet, inport)

msg = of.ofp\_flow\_mod(command=of.OFPFC\_ADD,

idle\_timeout=FLOW\_IDLE\_TIMEOUT,

hard\_timeout=of.OFP\_FLOW\_PERMANENT,

buffer\_id=event.ofp.buffer\_id,

actions=actions,

match=match)

event.connection.send(msg.pack())

elif self.arp\_for\_unknowns:

if (dpid,dstaddr) not in self.lost\_buffers:

self.lost\_buffers[(dpid,dstaddr)] = []

bucket = self.lost\_buffers[(dpid,dstaddr)]

entry = (time.time() + MAX\_BUFFER\_TIME,event.ofp.buffer\_id,inport)

bucket.append(entry)

while len(bucket) > MAX\_BUFFERED\_PER\_IP: del bucket[0]

self.outstanding\_arps = {k:v for k,v in self.outstanding\_arps.iteritems() if v > time.time()}

if (dpid,dstaddr) in self.outstanding\_arps:

37

return

self.outstanding\_arps[(dpid,dstaddr)] = time.time() + 4

r = arp()

r.hwtype = r.HW\_TYPE\_ETHERNET

r.prototype = r.PROTO\_TYPE\_IP

r.hwlen = 6

r.protolen = r.protolen

r.opcode = r.REQUEST

r.hwdst = ETHER\_BROADCAST

r.protodst = dstaddr

r.hwsrc = packet.src

r.protosrc = packet.next.srcip

1. = ethernet(type=ethernet.ARP\_TYPE, src=packet.src, dst=ETHER\_BROADCAST)

e.set\_payload(r)

log.debug("%i %i ARPing for %s on behalf of %s" % (dpid, inport,

r.protodst, r.protosrc))

msg = of.ofp\_packet\_out()

msg.data = e.pack()

msg.actions.append(of.ofp\_action\_output(port = of.OFPP\_FLOOD))

msg.in\_port = inport

event.connection.send(msg)

elif isinstance(packet.next, arp):

a = packet.next

log.debug("%i %i ARP %s %s => %s", dpid, inport, {arp.REQUEST:"request",arp.REPLY:"reply"}.get(a.opcode, 'op:%i' % (a.opcode,)), a.protosrc, a.protodst)

if a.prototype == arp.PROTO\_TYPE\_IP:

if a.hwtype == arp.HW\_TYPE\_ETHERNET:

if a.protosrc != 0:

if a.protosrc in self.arpTable[dpid]:

if self.arpTable[dpid][a.protosrc] != (inport, packet.src): log.info("%i %i RE-learned %s", dpid,inport,a.protosrc) if self.wide:

msg = of.ofp\_flow\_mod(command=of.OFPFC\_DELETE) msg.match.dl\_type = ethernet.IP\_TYPE msg.match.nw\_dst = a.protosrc event.connection.send(msg)

else:

log.debug("%i %i learned %s", dpid,inport,a.protosrc)

self.arpTable[dpid][a.protosrc] = Entry(inport, packet.src)

self.\_send\_lost\_buffers(dpid, a.protosrc, packet.src, inport)

if a.opcode == arp.REQUEST:

38

if a.protodst in self.arpTable[dpid]:

if not self.arpTable[dpid][a.protodst].isExpired():

r = arp()

r.hwtype = a.hwtype

r.prototype = a.prototype

r.hwlen = a.hwlen

r.protolen = a.protolen

r.opcode = arp.REPLY

r.hwdst = a.hwsrc

r.protodst = a.protosrc

r.protosrc = a.protodst

r.hwsrc = self.arpTable[dpid][a.protodst].mac

e = ethernet(type=packet.type, src=dpid\_to\_mac(dpid), dst=a.hwsrc)

e.set\_payload(r)

log.debug("%i %i answering ARP for %s" % (dpid, inport, r.protosrc))

msg = of.ofp\_packet\_out()

msg.data = e.pack()

msg.actions.append(of.ofp\_action\_output(port =

of.OFPP\_IN\_PORT))

msg.in\_port = inport

event.connection.send(msg)

return

log.debug("%i %i flooding ARP %s %s => %s" % (dpid, inport, {arp.REQUEST:"request",arp.REPLY:"reply"}.get(a.opcode, 'op:%i' % (a.opcode,)), a.protosrc, a.protodst))

msg = of.ofp\_packet\_out(in\_port = inport, data = event.ofp, action = of.ofp\_action\_output(port = of.OFPP\_FLOOD))

event.connection.send(msg)

def launch (fakeways="", arp\_for\_unknowns=None, wide=False):

fakeways = fakeways.replace(","," ").split()

fakeways = [IPAddr(x) for x in fakeways]

if arp\_for\_unknowns is None:

arp\_for\_unknowns = len(fakeways) > 0

else:

arp\_for\_unknowns = str\_to\_bool(arp\_for\_unknowns)

core.registerNew(l3\_switch, fakeways, arp\_for\_unknowns, wide)

39