Anomaly Detection in IOT Network Using SDN Principles

Project Report
Submitted in partial fulfilment for the Degree of
Bachelor of Computer Science Applications

by

Ashtapadhi SV

Supervised by

Mr. Binu PK

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

AMRITA VISHWA VIDYAPEETHAM

AMRITAPURI CAMPUS (INDIA)

Term: Feb to June - 2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

AMRITAPURI CAMPUS

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"Anomaly Detection in IOT Network Using SDN Principles"** submitted by **Ashtapadhi S V, Reg. No: AM.SC.U3CSC19023,** in partial fulfillment of the requirements for the award of the Degree Bachelor of Computer Science Applications is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Computing, Amrita Vishwa Vidyapeetham, Amritaprui.

Mr. Binu PK
Assistant Professor
Department of Computer Science and Applications
Amrita Vishwa Vidyapeetham
Amritapuri
India.

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS

AMRITA VISHWA VIDYAPEETHAM

AMRITAPURI CAMPUS

DECLARATION

I, **Ashtapadhi S V, Reg. No: AM.SC.U3CSC19023,** here by declare that this project report entitled **"Anomaly Detection in IOT Network Using SDN Principles",** is the record of the original work done by me under the guidance of **Mr. Binu PK**, Assistant Professor, Department of Computer Science and Applications, Amrita Vishwa Vidyapeetham, Amritapuri. To the best of my knowledge, this work has not formed the basis for the award of any degree/ diploma/ associateship/ fellowship/ or a similar award to any candidate in any University.

Place: Amritapuri **Signature of the Student**

Date: 27-05-2022

COUNTERSIGNED

Mr. Binu PK
Assistant Professor
Department of Computer Scinece and Applications
Amrita Vishwa Vidyapeetham
Amritapuri
India.

Contents

1 Abstract **5**

2 Introduction **5**
- 2.1 Background . . . 5
  - 2.1.1 Software Defined Network (SDN) . . . 5
  - 2.1.2 Software Defined Network and Traditional Networking . . . 6
- 2.2 Motivation . . . 6
- 2.3 Problem Definitions . . . 6
  - 2.3.1 Deauthentication Attack . . . 7
  - 2.3.2 DDoS UDP Packet Flooding . . . 7
  - 2.3.3 Smurf Attack . . . 7

3 Project Description **8**
- 3.1 Scope . . . 9
- 3.2 Project Overview . . . 9

4 Background Study **10**

5 Solution Approach **11**
  - 5.0.1 Deauthentication Attack . . . 12
  - 5.0.2 DDoS Attack . . . 14
  - 5.0.3 Smurf Attack . . . 21

6 Result and Analysis **24**

7 Conclusion **24**

8 References **25**

1 Abstract

IoT is a trend that is driving the ongoing digitization of the society in many new and amazing ways. Today, the Internet of Things has grown so broad that the development of its security should also be taken into concern along with it. As the use of IoT technologies has increased in different sectors, a very huge number of IoT devices are connected to the Internet and thus there is an ever increasing number of ways, that our technology can be hacked or exploited by those with malicious intentions. So, the security of these iot devices becomes very important. In this project, we propose a framework to provide security to IoT networks. To ensure that an IoT network is free from attacks, continuous monitoring of the network is required. This can be achieved using Software Defined Networking (SDN). SDN is an intelligent networking paradigm that can apply authentication and access rules that can open up a way for better security mechanisms. SDN enables the network to be centrally controlled using software applications. In this project, we propose a SDN based framework to detect and mitigate the anomalies that can occur in an IoT network. Using Mininet, an open-source network emulator, we implement a SDN prototype to emulate an IoT network. By applying threat modelling, all possible threats in the network are identified.

2 Introduction

2.1 Background

IoT is growing in importance, both for industrial use and everyday use. It is making our lives better in so many ways, and it will likely continue to do so. IOT helps us work smarter and live smarter.It has a lot to offer and has a potential future, but more often, the question of safety and security in IoT systems arises. To overcome the security challenges of IOT, we first have to understand the nature of IOT security issues.IoT devices are prone to attacks since device security is very poor. Currently, there is no protocol to guarantee the security of IoT devices.So, cybercriminals are taking advantage of this vulnerability. Threats to IoT systems and devices translate to bigger security risks because of certain characteristics that the underlying technology possesses. These characteristics make IoT environments functional and efficient, but they are likely to be abused by hackers. The impact of any attack on the IoT system can be more damaging than one can possibly imagine. Since IoT systems involve physical and virtual components, any cyber-attack can equally impact the physical devices. So developing a framework for the early detection and mitigation of anomalies in an IoT network is very important. Also, attacks on IoT devices may lead to denial of services on IoT networks. Therefore, in this project we are focusing on the early detection of certain denial-of-service attacks.

2.1.1 Software Defined Network (SDN)

To make sure the IoT networks are secure, continuous monitoring of the network is required; which is achieved using Software Defined Networking (SDN). Software-defined networking enables a new way of managing the entire network through a centralized controller. It separates the data plane and the control plane. SDN represents a substantial step forward from traditional networking, it enables Increased control with greater speed and flexibility, Customizable network infrastructure, Robust security. Instead of manually programming multiple vendor-specific hardware devices, developers can control the flow of traffic over a network simply by programming an open standard software-based controller. Networking administrators also have more flexibility in choosing networking equipment, since they can choose a single protocol to communicate with any number of hardware devices through a central controller.

2.1.2 Software Defined Network and Traditional Networking

The key difference between SDN and traditional networking is infrastructure. SDN is software-based, while traditional networking is hardware-based. Because the control plane is software-based, SDN is much more flexible than traditional networking. The SDN also uses the open flow protocol to increase the security of an IOT network. OpenFlow is a programmable network protocol designed to manage and direct traffic among routers and switches from various vendors.

2.2 Motivation

The main goal of this research is detecting and preventing different types of DoS and DDOS attacks in its early stages. IoT provides a promising future in different business sectors. The use of IOT have enhanced the services provided by different sectors like industry, education, agriculture etc. As IOT highly come into use in business, its security is an important concern for the business or industrial development. IoT security is a best practice to ensure the sustainability to IoT business: it provides trust, integrity and control. It protects key assets like devices, identity, data, decisions, commands and actions. Secure IOT brings different business development and opportunities also by protecting from the risks that connectivity brings.

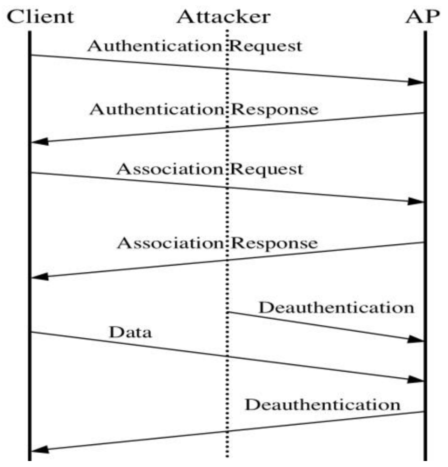
In the near future all the network providers will be switching to the fastest and most robust 5G technology. 5G speed and other connectivity benefits are expected to make businesses more efficient and give consumers access to more information faster than ever before. As IoT adoption continues to expand and more 5G networks are deployed, IoT security is more important than ever. 5G implementation provides malicious actors with new ways to infiltrate organizations' systems, networks and applications. It becomes easier for the attackers to attack the networks or devices, in a more efficient and easier way. To accomplish the goal to ensure security in IoT devices with the growth of 5G technology, a fast and effective method is needed, and that's where comes the need and importance of developing this project for the early detection and mitigation of iot network using SDN.

2.3 Problem Definitions

With a listing of different security concerns in IOT Networks, the main security threats we are concentrating upon in this research work is on Denial-of-Service and Distributed Denial- Of-Service. When an attack, meant to shut down a machine or network, making it inaccessible to its intended users is termed as Denial-of-service attacks. While, when a large number of packets are forwarded to a network device with an intent to either stop the service or decrease the performance from many different sources, they are termed as Distributed Denial-of-service attacks. These attacks need to be detected at an early stage so that the impact of the attack can be controlled significantly and the mitigation process can be made easier.

2.3.1 Deauthentication Attack

Deauthentication is a denial-of-service attack where Deauthentication frames terminate the connection between a client and an Access Point (AP). Here the attacker creates deauthentication frames containing the spoofed MAC address of a victim's AP and forward the deauthentication request to the victim. Deauthentication attack prevent the victim from connecting to their network and this is one of the Denail-of-Service attacks that this project is focusing.

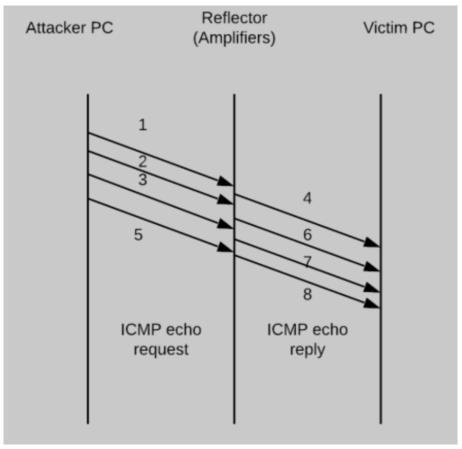


2.3.2 DDoS UDP Packet Flooding

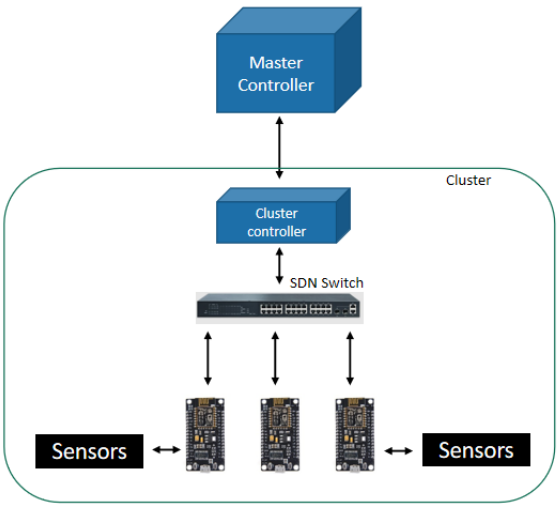
In DDoS attacks, a large number of packets are sent to a host or a group of hosts in a network. If the source addresses of the incoming packets are spoofed, which they usually are, the switch will not find a match and has to forward the packet to the controller. The collection of valid and the DDoS spoofed packets can bind the resources of the controller into continuous processing that exhausts them. This will make the controller unreachable for the newly arrived valid packets and may bring the controller down . Even if there is a backup controller, it has to face the same challenge.

2.3.3 Smurf Attack

A smurf attack is a type of denial of service attack in which a system is flooded with spoofed ping messages. This creates high computer network traffic on the victim's network, which often renders it unresponsive.A hacker overloads computers with Internet Control Message Protocol (ICMP) echo requests, also known as pings. The ICMP determines whether data reaches the intended destination at the right time and monitors how well a network transmits data. A smurf attack also sends ICMP pings but is potentially more dangerous because it can exploit vulnerabilities in the Internet Protocol (IP) and the ICMP.



3 Project Description



Here we convey the overall architecture for the early detection and mitigation of attacks in IOT devices using SDN framework. This architecture consists of the following four components : (i) IoT devices, (ii) SDN-enabled switch, (iii) Cluster SDN Controller and (iv) Master SDN Controller.

The IOT device that we use is the microcontroller i.e.; the nodemcu board which is connected to certain sensors to collect real time inputs. These end devices are connected to the SDN enabled switch. The SDN enabled switch supports OpenFlow protocol. The rules and policies for the security purpose will be installed in this switch. Then, according to the geo location, devices will be divided into different clusters and for each cluster there will be a SDN cluster controller which will control all the SDN switches in that network. Then there will be a master controller to which all the cluster controllers will be connected. It controls all the cluster controllers and it has a network wide overview of traffic flow and various events. Whenever an end device detects any anomalous data flow, it will inform the cluster controller and then the cluster controller will inform the master controller. The master controller will take necessary action to mitigate the attack like blocking the data packets and it will also update the rules in the entire network. So that, every device in the network will be secured from similar attacks. We use the mininet emulator to simulate this network to detect and mitigate the attacks in IOT network. Show down with a figure the proposed system.

3.1 Scope

Since IOT is prone to different types of attacks, enhancing its security is very much important. These attacks cause a lot of damage and loss for the genuine users. Attackers initiate the attack, where the Genuine users are not able to access resources, so may not be able to find the information or carry out the actions they need. Even the time-critical actions will not be able to carry out significantly for the genuine users and many more such issues affect the IOT network due to these attacks. So, this project focus on the enhancement of IoT security using SDN principles. In this project, securing IoT devices and networks are efficiently done with the use of SDN. By using SDN, the entire network can be controlled within a centralized controller. Another importance of using SDN is that, instead of manually programming multiple vendor-specific hardware devices, developers can control the flow of traffic over a network simply by programming an open standard software-based controller and also, continuous monitoring of the network is enabled. Thus, IOT networks can be secured in an efficient way.

3.2 Project Overview

After many studies and researches to implement security in IOT using SDN principles, we got familiar with different technologies which became an important part of this project.
1. creation of a simple IOT network using a nodeMcu microcontroller board and certain sensors, to collect real time inputs was done. The real time inputs collected using the sensor readings are later pushed to thingspeak cloud.
2. Since the real implementation of SDN is costly, we are using the Mininet emulator to emulate this IoT network. Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. The switch available in the Mininet supports OpenFlow protocol. Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on a laptop or other PC. It enables complex topology testing, without the need to wire up a physical network. We also uses the Mininet-WiFi which is a fork of Mininet which allows the using of both WiFi Stations and Access Points. Mininet-WiFi provides extended functionalities like wifi features and we can use it in the same way we are using Mininet. Mininet Provides a simple and inexpensive network testbed for developing OpenFlow applications. The SDN uses OpenFlow protocol to increase the security of an IoT network. OpenFlow is a programmable network protocol designed to manage and direct traffic among routers and switches from various

vendors.

4 Background Study

In [1], it addresses the problem of IoT security. With the help of SDN, it aims to prevent the attacks at network level instead of device level. The objective of this paper is to protect the IoT devices from malicious attacks and reduce the damage upon an attack. The attack may be launched from the IoT device itself or the device is the target. In this paper, they propose a framework, Soft Things, for IoT security based on SDN techniques which helps in detection of anomalous behavior and enhanced resilience. It implements the proof of concept on Mininet emulator to detect anomalous traffic of IoT using Support Vector Machine (SVM), a machine learning algorithm and subsequent mitigation of attacks. They have considered various attacks like TCP flooding, ICMP flooding, DDoS and scenarios with IoT device as both target and source of attacks. It also compares the performance of linear and nonlinear SVM for these attack detection in different scenarios. The experimentation is performed using Mininet1 emulation environment which supports OpenFlow protocol. Mininet is used to emulate IoT devices and other functional nodes viz. attacker node. POX2 is used as the controller where algorithms are run to detect traffic behavior before applying suitable actions on flows. They used network topologies considering different scenarios and attack models. For every scenario, they have provided the controller with training data containing normal traffic patterns of the devices and patterns of attacks concerned. Primarily, they consider three scenarios: (i) IoT device is under attack, (ii) Compromised IoT device is being used to launch attack and (iii) DDoS attack by multiple compromised IoT device. As a result, they observed a few false detections when linear SVM is used. For this reason, they observed lower precision of detection in each scenario. On the other hand, the non-linear SVM provided better precision as it uses kernel trick and consequently reduces wrong detections. Thus, it was found that non-linear SVM is a better learning technique for such attack detection. Next, they observe that different attack models in those three scenarios were mitigated within a few seconds, within 2 to 3 seconds. This paper helped us to know more about the usage of different controllers, emulators, protocols and much more in order to detect and mitigate the attacks.

In [2], they have tried to identify various possibilities of DDoS attacks in SDN environment with the help of attack tree and an attack model. Further, an attempt to analyze the impact of various traditional DDoS attacks on SDN components is done. Such analysis helps in identifying the type of DDoS attacks that impose bigger threat on SDN architecture and also the features that could play important role in identification of these attacks are deduced. This paper is organized in different sections. First it discusses research works involving the description of various DDoS attacks in SDN environment. Next, they have classified various DDoS attacks in SDN environment using an attack tree. Further, they modeled DDoS attacks by identifying the attacker, victim, and type of attacks. In next section, they have implemented ICMP flood attack, Smurf attack, UDP flood attack, TCP SYN attack and HTTP flood attack in SDN environment to analyze their impact on SDN architecture and to identify features helpful for detection of these attacks. Then, they analyzed these attacks based on their severity, mainly focusing on how they affect SDN controller and data plane.

[3] stays around the Wi-Fi DOS attack and practical detection of it. While some network equipment has built in features to prevent the attacks; their performance, effectiveness to defend against this attack is not impressive at all. So, this paper provides a fascinating journey to witness how combination of packet crafting skills in scapy and python scripting will assist us to detect this attack. The profound aim of the paper is to Detect the wireless DOS. Here, the researchers first installed certain tools which were required to implement the attack. Then they launched the deauthentication attack by Identifying Wireless Network Interface Name, Turning NIC card into Monitor Mode, Scan for Target Step, change our NIC card Channel to respective channel of Target AP and Broadcasting the deauth Packets. The detection of Wi-Fi Deauthentication Attack was done. Here scapy was used as it can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. As a result of the project we found that, with few lines of scapy and python scripting we can detect deauth attacks in our environment. Through this paper we got an idea on how to implement an attack and detect it.

According to [4], even if the central control is a major advantage of SDN, it is also a single point of failure if it is made unreachable by a Distributed Denial of Service (DDoS) Attack. To mitigate this threat, this paper proposes to use the central control of SDN for attack detection and introduces a solution that is effective and lightweight in terms of the resources that it uses. More precisely, this paper shows how DDoS attacks can exhaust controller resources and provides a solution to detect such attacks based on the entropy variation of the destination IP address. This method is able to detect DDoS within the first five hundred packets of the attack traffic. This paper presented a lightweight and effective solution for detecting DDoS attack in SDN. In addition to early detection of the attack, the solution's addition to the controller is transparent both in terms of resources and functionality. It provides the detection for both the controller and the attacked host. The mitigation methods provided in this paper helped us to solve the errors and got an idea on how to implement some of the steps efficiently.

5 Solution Approach

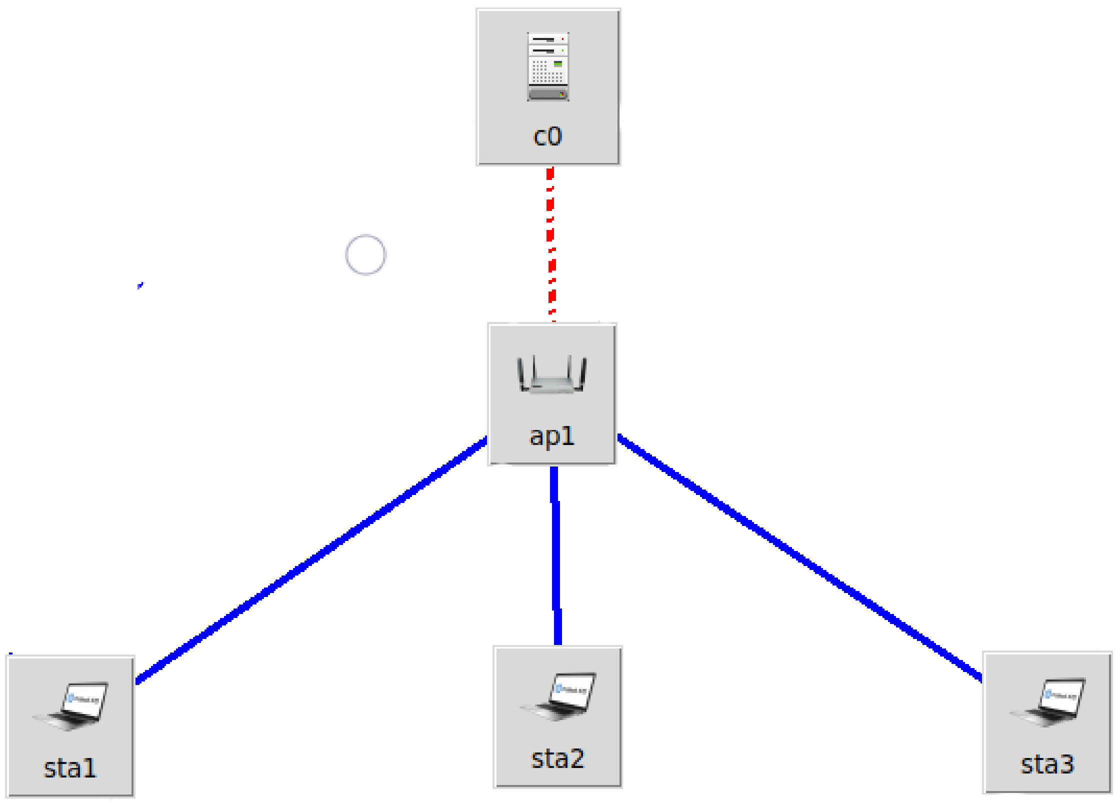
This project provides ways to detect and mitigate different types of attacks that target the IOT devices or network using SDN principles. By continuous monitoring of the IOT network and devices, SDN enables an early detection of the attacks that emerge our IOT system. After modelling and analyzing the behavior pattern of different attacks, we developed the ways to mitigate it. DoS and DDoS attacks were mainly focused to implement and mitigate in our project.

5.0.1 Deauthentication Attack

A deauthentication attack is a type of attack which targets the communication between router and the device which effectively disables the WiFi on the device. Deauthentication attack is a created protocol and is being used in real world applications. Deauthencation attack's use a deauthentication frame. This frame sent from a router to a device forces the device to disconnect. Here a device is on the network that shouldn't be on the network. The router sends a deauthentication frame to the device telling it that it has been disconnected.

In this project for performing the deauthentication attack, we created a topology in Mininet-Wifi. We used Mininet-Wifi for this attack because it supports wireless extension and wifi features. Deauthentication is a WiFi attack and hence it needs a WiFi environment to execute.

We created a topology with 3 stations, 1 Access-point and 1 controller. After that, the attack is performed from station 1 to station 3. It is shown in the figure below.



The 2 important things we need to know before performing a deauthentication attack are the mac address of the device we want to disconnect from the network and the mac address of the router that the device is connected to. In our project, this attack was done using Mininet-Wifi. At first, we need to install the aircrack-ng suite. Then in the Linux like terminal of station 1, run the following command- sudo apt install aircrack-ng.

Aircrack-ng utility on Linux is used for scanning and cracking wireless networks encryption. This utility contains a remarkable tool called aireplay-ng, it provides an option to spoof and send deauthentication packets to one or more associated clients with an explicit access point.

The steps to execute deauthentication attack are given below:

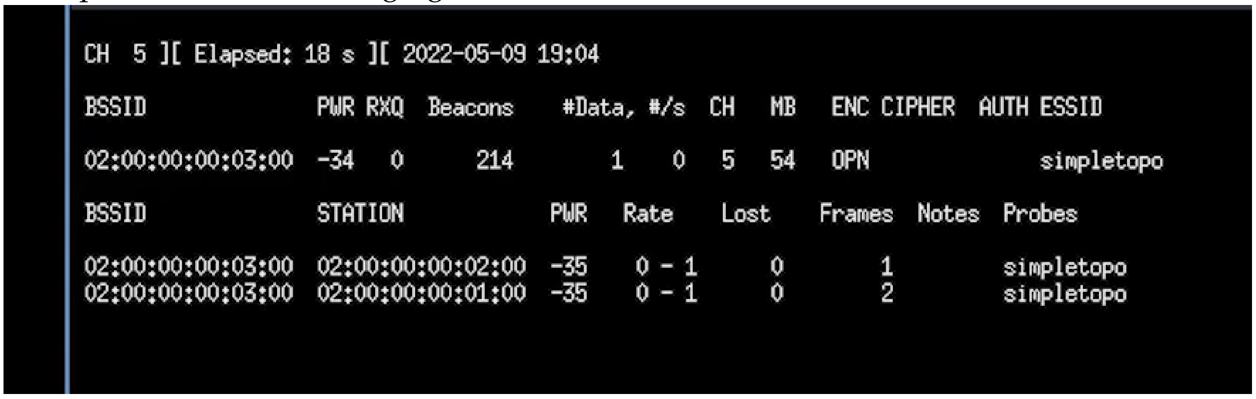
1. Run iwconfig which shows us what our wireless card is called.

2. Next run this command: airmon-ng start sta1-wlan0 where sta1-wlan0 is our network card. This will put our card into monitor mode which allows the card to monitor all traffic on the network.

3. Now that the wireless card is in monitor mode we want to see every router around us. Then we will run iwconfig again as this command will change our network card name.

4. Run this command with your new network card name: airdump-ng wlan Then we will see every single router in range. We need to know what router the device is on. We can identify how close a router is by the PWR column. PWR is the signal strength, how close it is to us. The closer it is, the larger the signal strength. We want to take note of 2 things here: The BSSID (mac address) of the router and the Channel of the router.

5. In order to find and scan for associated clients with a particular access point, Airodump-ng tool was found to achieve such purpose efficiently as it can be depicted in the following figure:

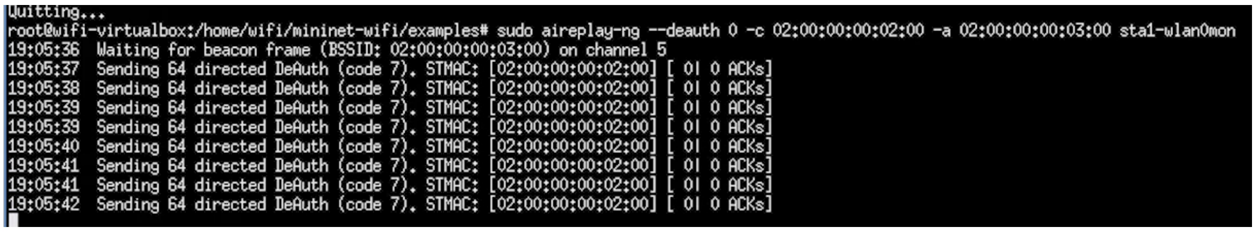
   

   ```
   CH  5 ][ Elapsed: 18 s ][ 2022-05-09 19:04

   BSSID              PWR RXQ  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

   02:00:00:00:03:00  -34   0      214        1    0   5   54   OPN              simpletopo

   BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

   02:00:00:00:03:00  02:00:00:00:02:00  -35    0 - 1     0        1         simpletopo
   02:00:00:00:03:00  02:00:00:00:01:00  -35    0 - 1     0        2         simpletopo
   ```

   Command used : airodump-ng wlan0mon –bssid [routers BSSID here]–channel [routers channel here]

6. Finally, run this command to execute the attack:
   aireplay-ng –deauth 0 -c [DEVICES MAC ADDRESS] -a [ROUTERS MAC ADDRESS] wlan0mon The 0 represents an infinite amount of deauth attacks. If we want to only run 3 deauth attacks we can change this to 3. -c is the client, what you're attacking. This is the devices MAC address.

   -a is the router, what is the router the victim is connected to. The following screenshots demonstrates a practical example for continuous deauthentication packets that is being sent to a particular wireless access point.

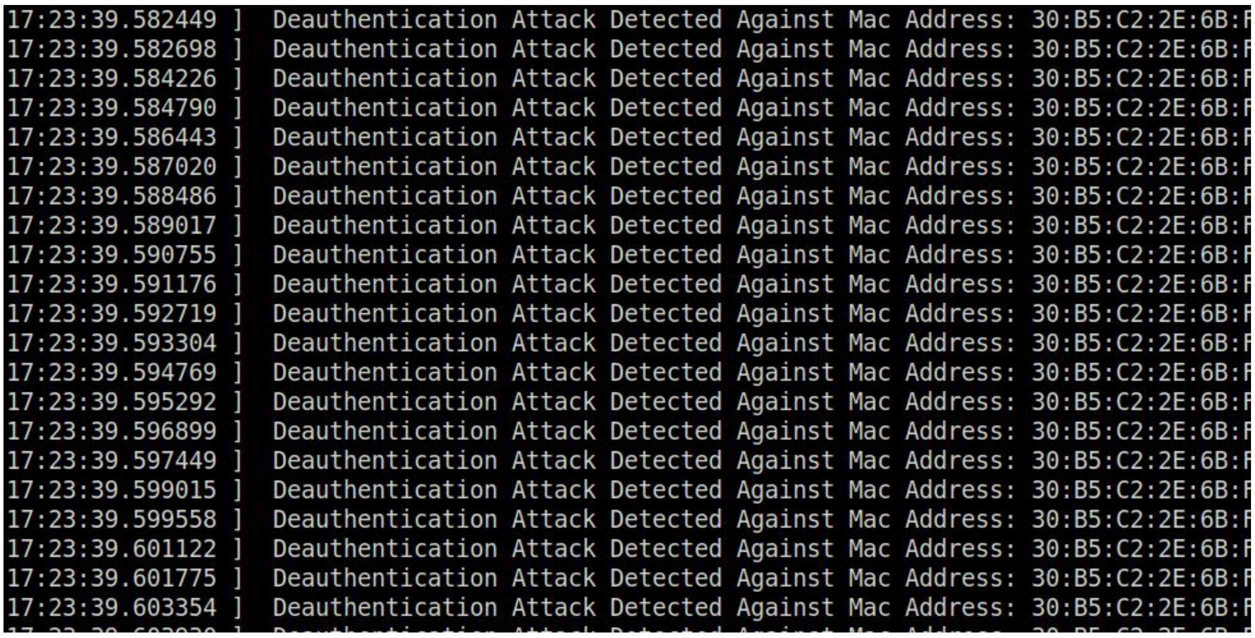


```
Quitting...
root@wifi-virtualbox:/home/wifi/mininet-wifi/examples# sudo aireplay-ng --deauth 0 -c 02:00:00:00:02:00 -a 02:00:00:00:03:00 sta1-wlan0mon
19:05:36  Waiting for beacon frame (BSSID: 02:00:00:00:03:00) on channel 5
19:05:37  Sending 64 directed DeAuth (code 7). STMAC: [02:00:00:00:02:00] [ 0| 0 ACKs]
19:05:38  Sending 64 directed DeAuth (code 7). STMAC: [02:00:00:00:02:00] [ 0| 0 ACKs]
19:05:39  Sending 64 directed DeAuth (code 7). STMAC: [02:00:00:00:02:00] [ 0| 0 ACKs]
19:05:39  Sending 64 directed DeAuth (code 7). STMAC: [02:00:00:00:02:00] [ 0| 0 ACKs]
19:05:40  Sending 64 directed DeAuth (code 7). STMAC: [02:00:00:00:02:00] [ 0| 0 ACKs]
19:05:41  Sending 64 directed DeAuth (code 7). STMAC: [02:00:00:00:02:00] [ 0| 0 ACKs]
19:05:41  Sending 64 directed DeAuth (code 7). STMAC: [02:00:00:00:02:00] [ 0| 0 ACKs]
19:05:42  Sending 64 directed DeAuth (code 7). STMAC: [02:00:00:00:02:00] [ 0| 0 ACKs]
```

Station 3 (sta3) is now disconnected from the access point. We can check that by executing ping command from station 3 to station 2.

After executing the attack, we use wireshark to sniff the packet and analyse it.

For detection of deauthentication attack, we use scapy library in python. Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. It can replace hping, arpspoof, arp-sk, arping, and even some parts of Nmap, tcpdump, and tshark. We can detect the deauthentication attack by Analyzing if sniffed packet is Dot11 Deauth layer. For preventing this attack, new rules are set by the controller to block the packets having Dot11 Deauth layer.



```
17:23:39.582449 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.582698 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.584226 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.584790 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.586443 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.587020 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.588486 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.589017 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.590755 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.591176 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.592719 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.593304 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.594769 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.595292 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.596899 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.597449 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.599015 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.599558 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.601122 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.601775 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
17:23:39.603354 ]  Deauthentication Attack Detected Against Mac Address: 30:B5:C2:2E:6B:F
```

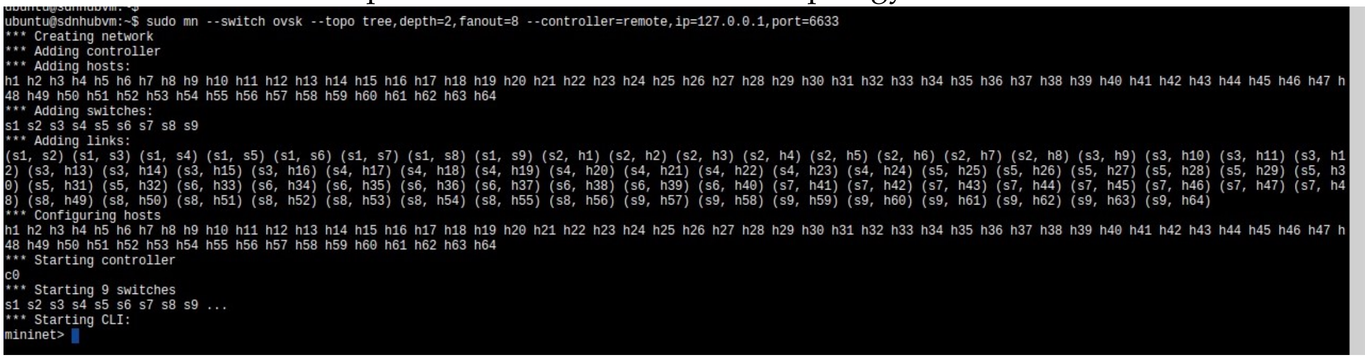
5.0.2 DDoS Attack

We worked on the creating a DDos attack and detection of the attack on the entropy base and thus, preventing the DDos to occur. If the source addresses of incoming are spoofed, then the switch would not find a match and the packet needs to be forwarded to the controller. The collection of DDoS spoofed packets and legitimate packets can bind the controller into continuous processing, which exhausts them. Due to this, the controller is unreachable for the new incoming legitimate packets. This will bring the controller down causing loss to the SDN architecture. For a backup controller, the same challenge is to be faced.

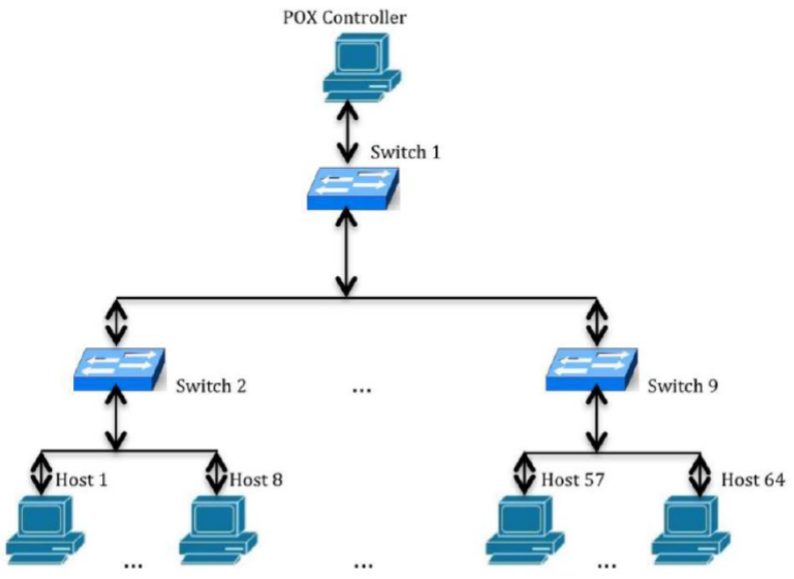
Such kind of attacks can be detected in the early stage by monitoring few hundred of packets considering changes in entropy. The main reason for considering entropy is its ability for measuring randomness in a network. The higher the randomness the higher is the entropy and vice versa. So, whenever the entropy is less than a threshold value we can say that a DDos attack is occurred.

Basically, an Openflow controller is connected to a network. We then observe the entropy of the traffic related to the controller under normal and attack conditions. We used the POX controller for this project, because it runs on Python. The network emulator used is Mininet for creation of network topology. Packet generation is done with the help of Scapy. Where Scapy is used for generation of packets, sniffing, scanning, forging of packet and attacking. Scapy is used for generation of UDP packets and spoofing the source IP address of the packets.

Here first we create a Mininet topology of 9 switches and 64 hosts. We then started pox controller from another terminal and then we can see 9 openflow links are connected for 9 open switches. The creation of topology is shown below:



```
ubuntu@sdnhubvm:~$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,ip=127.0.0.1,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h
48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (s1, s8) (s1, s9) (s2, h1) (s2, h2) (s2, h3) (s2, h4) (s2, h5) (s2, h6) (s2, h7) (s2, h8) (s3, h9) (s3, h10) (s3, h11) (s3, h1
2) (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, h3
0) (s5, h31) (s5, h32) (s6, h33) (s6, h34) (s6, h35) (s6, h36) (s6, h37) (s6, h38) (s6, h39) (s6, h40) (s7, h41) (s7, h42) (s7, h43) (s7, h44) (s7, h45) (s7, h46) (s7, h47) (s7, h4
8) (s8, h49) (s8, h50) (s8, h51) (s8, h52) (s8, h53) (s8, h54) (s8, h55) (s8, h56) (s9, h57) (s9, h58) (s9, h59) (s9, h60) (s9, h61) (s9, h62) (s9, h63) (s9, h64)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h
48 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>
```



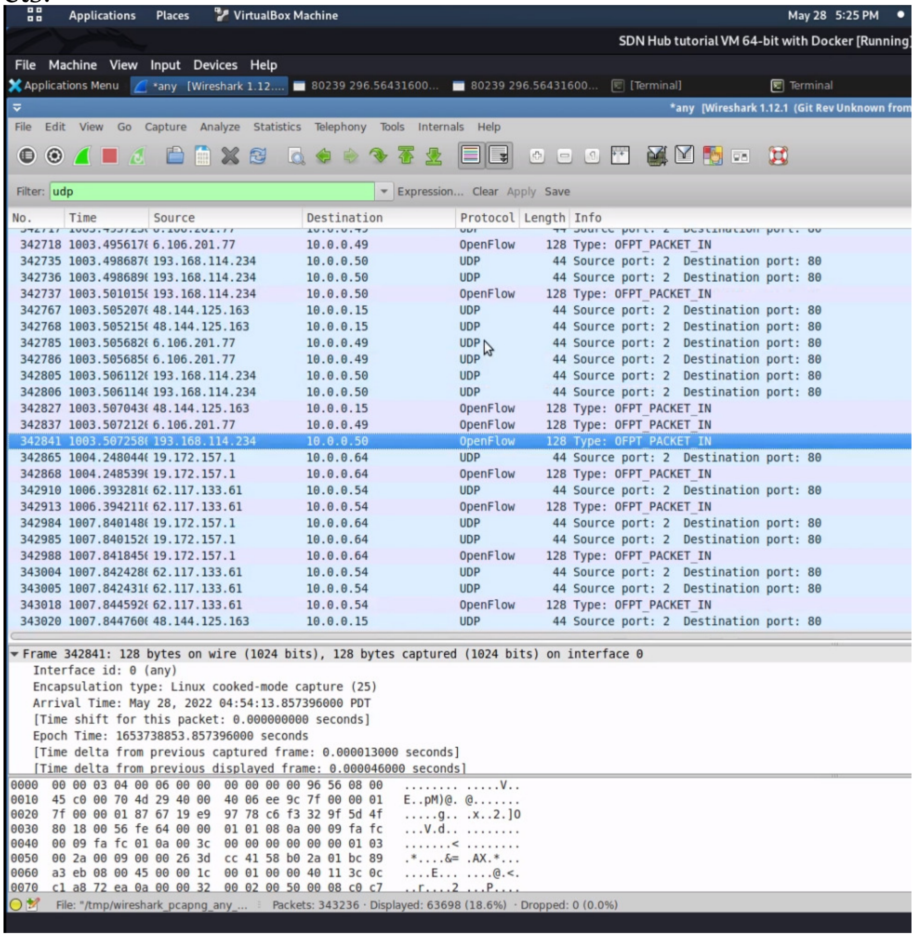
Next, we executed a python code to launch the traffic. this python file generates random source ip and send the packets to random destinations between the host.

We generally give start and end value from the command prompt of one of the node, for example h1.

We import sys module of Python for accessing system specific parameters and functions. The getopt module helps scripts to parse the command line arguments. We use os module to import Popen to fetches shell commands into python program. Scapy module is used to import functions such as sendp, IP, UDP, Ethernet and TCP. The random module is used to import randrange function. Here we used the if_name_ which is used to avoid the running of the main function when called from other programs.

For the implementation of attack,we run the python code to execute attack from few selected hosts to a selected target. Now the entropy value in the controller decreases. Here we give the target ip address from the command prompt of the hosts which are acting like a botnet. The time module provides various time related functions. The logging module helps implement logging system for applications. The logging.getlogger function will suppress all messages that have a lower level of seriousness than error messages, before importing scapy. Here we get the IP address of target from the command prompt of the bonet armies. As in the same way of traffic launch. We create the packet with the random src ip and then send it out through the eth0 with help of sendp function of scapy.

After implementing the attack, we used Wireshark software to analyse the packets.


Applications
Places
VirtualBox Machine
May 28 5:25 PM
SDN Hub tutorial VM 64-bit with Docker [Running]
File Machine View Input Devices Help
Applications Menu
*any [Wireshark 1.12....
80239 296.56431600...
80239 296.56431600...
[Terminal]
Terminal
*any [Wireshark 1.12.1 (Git Rev Unknown from
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: udp
Expression... Clear Apply Save

```
No.    Time        Source           Destination  Protocol Length Info
342718 1003.495617( 6.106.201.77     10.0.0.49    OpenFlow   128 Type: OFPT_PACKET_IN
342735 1003.498687( 193.168.114.234  10.0.0.50    UDP         44 Source port: 2  Destination port: 80
342736 1003.498689( 193.168.114.234  10.0.0.50    UDP         44 Source port: 2  Destination port: 80
342737 1003.501015( 193.168.114.234  10.0.0.50    OpenFlow   128 Type: OFPT_PACKET_IN
342767 1003.505207( 48.144.125.163   10.0.0.15    UDP         44 Source port: 2  Destination port: 80
342768 1003.505215( 48.144.125.163   10.0.0.15    UDP         44 Source port: 2  Destination port: 80
342785 1003.505682( 6.106.201.77     10.0.0.49    UDP         44 Source port: 2  Destination port: 80
342786 1003.505685( 6.106.201.77     10.0.0.49    UDP         44 Source port: 2  Destination port: 80
342805 1003.506112( 193.168.114.234  10.0.0.50    UDP         44 Source port: 2  Destination port: 80
342806 1003.506114( 193.168.114.234  10.0.0.50    UDP         44 Source port: 2  Destination port: 80
342827 1003.507043( 48.144.125.163   10.0.0.15    OpenFlow   128 Type: OFPT_PACKET_IN
342837 1003.507212( 6.106.201.77     10.0.0.49    OpenFlow   128 Type: OFPT_PACKET_IN
342841 1003.507258( 193.168.114.234  10.0.0.50    OpenFlow   128 Type: OFPT_PACKET_IN
342865 1004.248044( 19.172.157.1     10.0.0.64    UDP         44 Source port: 2  Destination port: 80
342868 1004.248539( 19.172.157.1     10.0.0.64    OpenFlow   128 Type: OFPT_PACKET_IN
342910 1006.393281( 62.117.133.61    10.0.0.54    UDP         44 Source port: 2  Destination port: 80
342913 1006.394211( 62.117.133.61    10.0.0.54    OpenFlow   128 Type: OFPT_PACKET_IN
342984 1007.840148( 19.172.157.1     10.0.0.64    UDP         44 Source port: 2  Destination port: 80
342985 1007.840152( 19.172.157.1     10.0.0.64    UDP         44 Source port: 2  Destination port: 80
342988 1007.841845( 19.172.157.1     10.0.0.64    OpenFlow   128 Type: OFPT_PACKET_IN
343004 1007.842428( 62.117.133.61    10.0.0.54    UDP         44 Source port: 2  Destination port: 80
343005 1007.842431( 62.117.133.61    10.0.0.54    UDP         44 Source port: 2  Destination port: 80
343018 1007.844592( 62.117.133.61    10.0.0.54    OpenFlow   128 Type: OFPT_PACKET_IN
343020 1007.844760( 48.144.125.163   10.0.0.15    UDP         44 Source port: 2  Destination port: 80

▾ Frame 342841: 128 bytes on wire (1024 bits), 128 bytes captured (1024 bits) on interface 0
    Interface id: 0 (any)
    Encapsulation type: Linux cooked-mode capture (25)
    Arrival Time: May 28, 2022 04:54:13.857396000 PDT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1653738853.857396000 seconds
    [Time delta from previous captured frame: 0.000013000 seconds]
    [Time delta from previous displayed frame: 0.000046000 seconds]

0000  00 00 03 04 00 06 00 00  00 00 00 00 96 56 08 00   ........ .....V..
0010  45 c0 00 70 4d 29 40 00  40 06 ee 9c 7f 00 00 01   E..pM)@. @.......
0020  7f 00 00 01 87 67 19 e9  97 78 c6 f3 32 9f 5d 4f   .....g.. .x..2.]O
0030  80 18 00 56 fe 64 00 00  01 01 08 0a 00 09 fa fc   ...V.d.. ........
0040  00 09 fa fc 01 0a 00 3c  00 00 00 00 00 00 01 03   .......< ........
0050  00 2a 00 09 00 00 26 3d  cc 41 58 b0 2a 01 bc 89   .*....&= .AX.*...
0060  a3 eb 08 00 45 00 00 1c  00 01 00 00 40 11 3c 0c   ....E... ....@.<.
0070  c1 a8 72 ea 0a 00 00 32  00 02 00 50 00 08 c0 c7   ..r....2 ...P....

File: "/tmp/wireshark_pcapng_any_...   Packets: 343236 · Displayed: 63698 (18.6%) · Dropped: 0 (0.0%)
```

For Detecting the Attack, first we make a count of 50 packets in the window and then we calculate the entropy and compare it with threshold that we set and make a count of consecutive entropy value lower than threshold. If this count reaches 5 then we can say that ddos had occurred otherwise not.

For this we written a code for detection and did some changes in the l3_learning module of the pox controller so that it can detect the ddos. These changes are explained below:

Entropy formula : Here we detect the entropy with the help of 2 factors

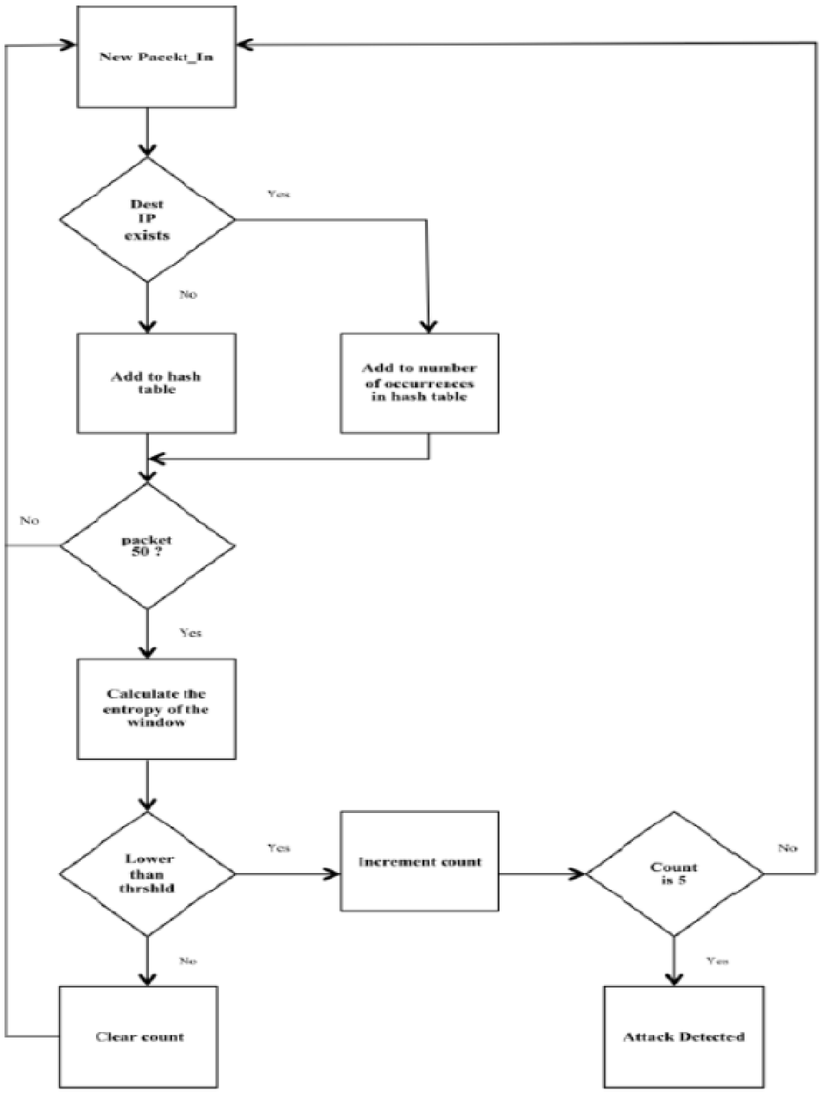
destination ip and no. of the times it repeated. : Here we used the window size to be 50 and Probability of destination ip occurred in the window is given as pi

```
pi=(xi)/n          where x is no. of event in the set and n to be the
```

Now ,

```
entropy H= - sum of all (pi)log(pi)
Where i is from o to n
```

The sequence of steps we use in the detection code is given as a flow below.

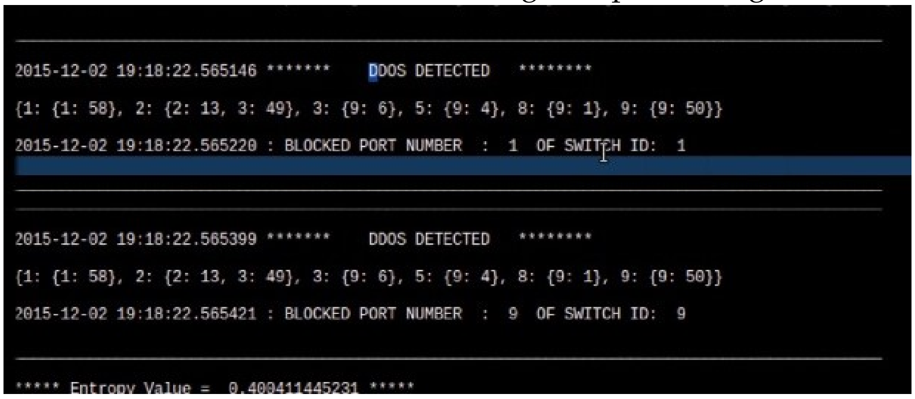


While looking into the python code, In the class Entropy, we defined the entropy dictionary, IP list and destination entropy are as null. The count and value are assigned as 0 and 1 respectively. In the statcolect function, we basically collect statistics related to the detection of attack i.e entropy. Here every packet in message is collected into iplist and count is incremented for every packet in. When the count value reaches 50, the 50 Packet_In messages would be parsed for their destination IP addresses in the hash table. If it is present in the table its value increments otherwise it is listed with 1. Entropy is calculated using this hash table values. In the entropy function, We calculate the probability of destination ip occurred in the window and then we calculate the entropy by the formula :

```
entropy H= - sum of all (pi)log(pi)
```

Then we store this value in entropy dictionary which is used in l3_learning module of pox to compare and say whether attack had occurred or not.

screenshot of Pox controller detecting and preventing the attack is given below:



```
2015-12-02 19:18:22.565146 *******    DDOS DETECTED   ********
{1: {1: 58}, 2: {2: 13, 3: 49}, 3: {9: 6}, 5: {9: 4}, 8: {9: 1}, 9: {9: 50}}
2015-12-02 19:18:22.565220 : BLOCKED PORT NUMBER  :  1  OF SWITCH ID:  1

2015-12-02 19:18:22.565399 *******    DDOS DETECTED   ********
{1: {1: 58}, 2: {2: 13, 3: 49}, 3: {9: 6}, 5: {9: 4}, 8: {9: 1}, 9: {9: 50}}
2015-12-02 19:18:22.565421 : BLOCKED PORT NUMBER  :  9  OF SWITCH ID:  9

***** Entropy Value =  0.400411445231 *****
```

Overall Steps

Finding the threshold of the usual traffic:

1. Creating a Mininet topology by entering the following command:

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

2. In the Mininet terminal of virtual box enter the following command for running the pox controller:

   ```
   $ cd pox

   $ python ./pox.py  forwarding.l3_edited
   ```

3. To determine the IP address of the POX controller, entering the following command in Mininet:

   ```
   $ ifconfig
   ```

   The loopback address is the IP address in the above command.

4. Now opening xterm for a host by typing the following command:

   ```
   mininet>xterm h1
   ```

5. In the xterm window of host h1 running the following command:

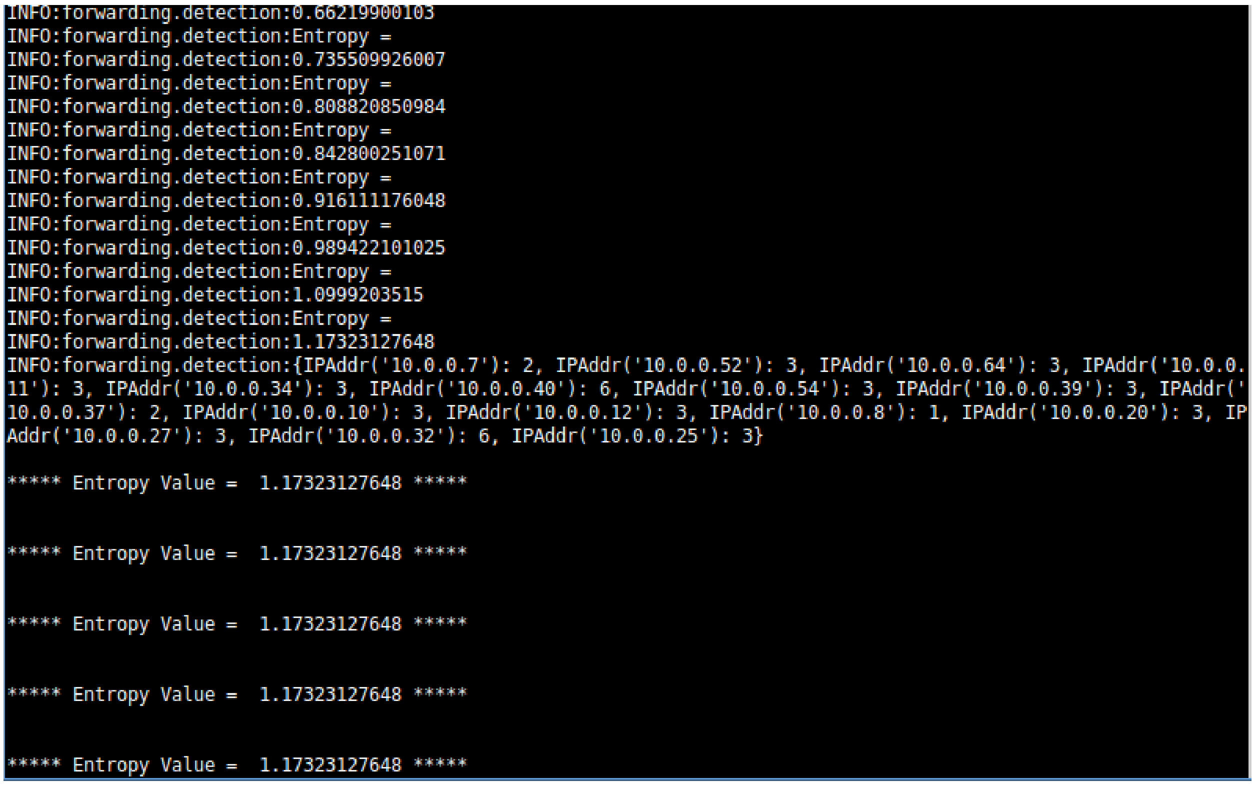
   ```
   # python launchTraffic.py {s 2 {e 65
   ```

Detection of DDoS threat using the value of Entropy:

On xterm window of h64 entering the following commands:

```
# script h64.txt

# tcpdump {v
```

Entropy value before the DDoS attack:

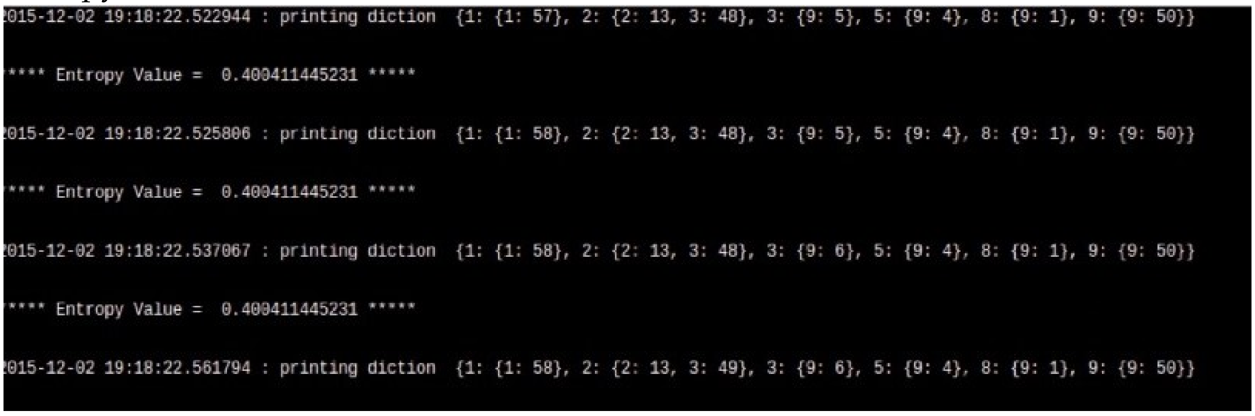


```
INFO:forwarding.detection:0.66219900103
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.735509926007
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.808820850984
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.842800251071
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.916111176048
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.989422101025
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.0999203515
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.17323127648
INFO:forwarding.detection:{IPAddr('10.0.0.7'): 2, IPAddr('10.0.0.52'): 3, IPAddr('10.0.0.64'): 3, IPAddr('10.0.0.
11'): 3, IPAddr('10.0.0.34'): 3, IPAddr('10.0.0.40'): 6, IPAddr('10.0.0.54'): 3, IPAddr('10.0.0.39'): 3, IPAddr('
10.0.0.37'): 2, IPAddr('10.0.0.10'): 3, IPAddr('10.0.0.12'): 3, IPAddr('10.0.0.8'): 1, IPAddr('10.0.0.20'): 3, IP
Addr('10.0.0.27'): 3, IPAddr('10.0.0.32'): 6, IPAddr('10.0.0.25'): 3}

***** Entropy Value =  1.17323127648 *****


***** Entropy Value =  1.17323127648 *****


***** Entropy Value =  1.17323127648 *****


***** Entropy Value =  1.17323127648 *****


***** Entropy Value =  1.17323127648 *****
```

Now repeating step 5. on h1 and parallelly entering the following commands to run the attack traffic from h4 and h6 xterm windows to attack on 56

```
# python launchAttack.py 10.0.0.56
```

Entropy value after the DDoS attack:



```
015-12-02 19:18:22.522944 : printing diction  {1: {1: 57}, 2: {2: 13, 3: 48}, 3: {9: 5}, 5: {9: 4}, 8: {9: 1}, 9: {9: 50}}

**** Entropy Value =  0.400411445231 *****

015-12-02 19:18:22.525806 : printing diction  {1: {1: 58}, 2: {2: 13, 3: 48}, 3: {9: 5}, 5: {9: 4}, 8: {9: 1}, 9: {9: 50}}

**** Entropy Value =  0.400411445231 *****

015-12-02 19:18:22.537067 : printing diction  {1: {1: 58}, 2: {2: 13, 3: 48}, 3: {9: 6}, 5: {9: 4}, 8: {9: 1}, 9: {9: 50}}

**** Entropy Value =  0.400411445231 *****

015-12-02 19:18:22.561794 : printing diction  {1: {1: 58}, 2: {2: 13, 3: 49}, 3: {9: 6}, 5: {9: 4}, 8: {9: 1}, 9: {9: 50}}
```

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 hosts stop sending attack packets, the switches are started again by the POX controller after shutting them down during the attack.

On successful completion of the above steps, terminating tcpdump on h64 by entering ‘control/command + c’

Stop running Mininet topology by entering the following command:

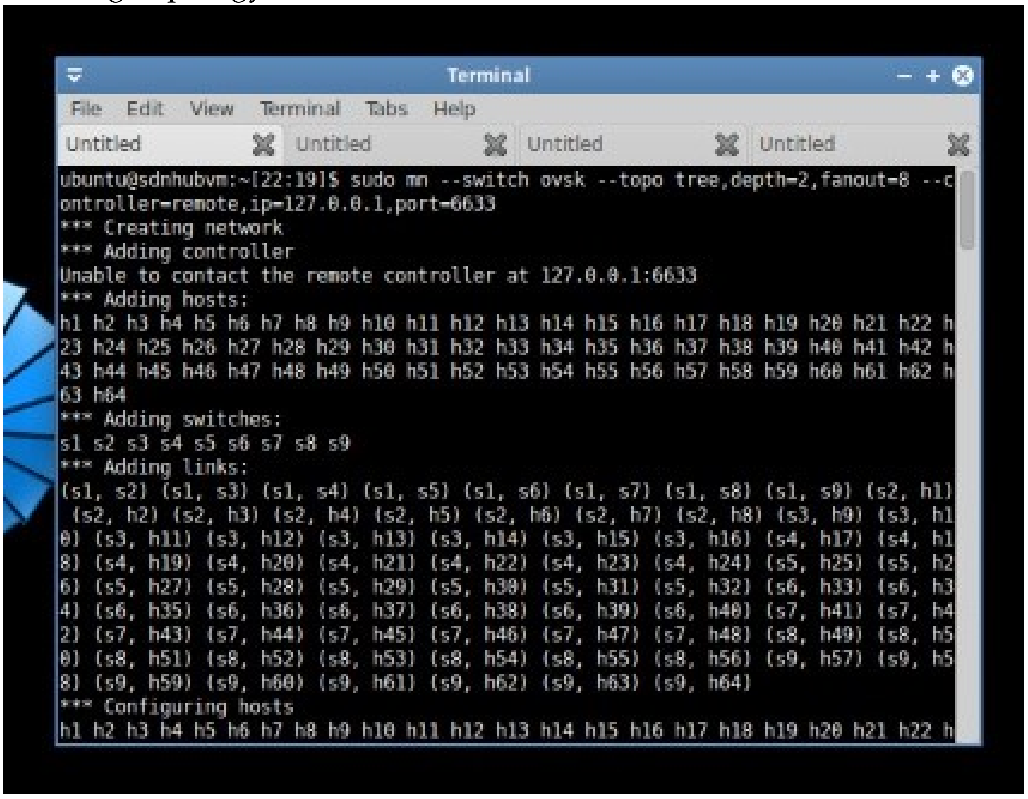
```
mininet>exit
```

5.0.3 Smurf Attack

In this attack, the attacker forges ICMP echo request packets with the IP address of the victim as the source address and broadcasts therequest on the network, making the computers in the network to send repliesto the ICMP echo requests.In a multi-access broadcast network, the number of replies could be overwhelming as hundreds of computer may listen to the broadcast.

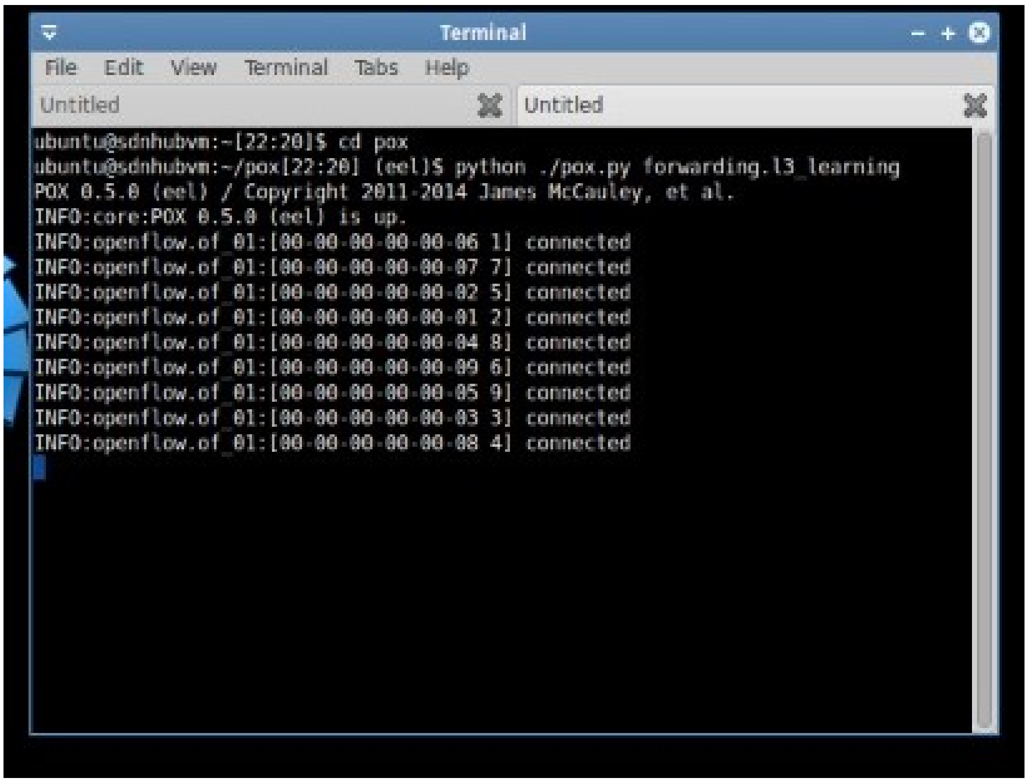
For implementing this attack, same topology which is used for the DDoS attack is used.

Creating Topology :



```
ubuntu@sdnhubvm:~[22:19]$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --c
ontroller=remote,ip=127.0.0.1,port=6633
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h
23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h
43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h
63 h64
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (s1, s8) (s1, s9) (s2, h1)
 (s2, h2) (s2, h3) (s2, h4) (s2, h5) (s2, h6) (s2, h7) (s2, h8) (s3, h9) (s3, h1
0) (s3, h11) (s3, h12) (s3, h13) (s3, h14) (s3, h15) (s3, h16) (s4, h17) (s4, h1
8) (s4, h19) (s4, h20) (s4, h21) (s4, h22) (s4, h23) (s4, h24) (s5, h25) (s5, h2
6) (s5, h27) (s5, h28) (s5, h29) (s5, h30) (s5, h31) (s5, h32) (s6, h33) (s6, h3
4) (s6, h35) (s6, h36) (s6, h37) (s6, h38) (s6, h39) (s6, h40) (s7, h41) (s7, h4
2) (s7, h43) (s7, h44) (s7, h45) (s7, h46) (s7, h47) (s7, h48) (s8, h49) (s8, h5
0) (s8, h51) (s8, h52) (s8, h53) (s8, h54) (s8, h55) (s8, h56) (s9, h57) (s9, h5
8) (s9, h59) (s9, h60) (s9, h61) (s9, h62) (s9, h63) (s9, h64)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h
```

Starting the Pox Controller :



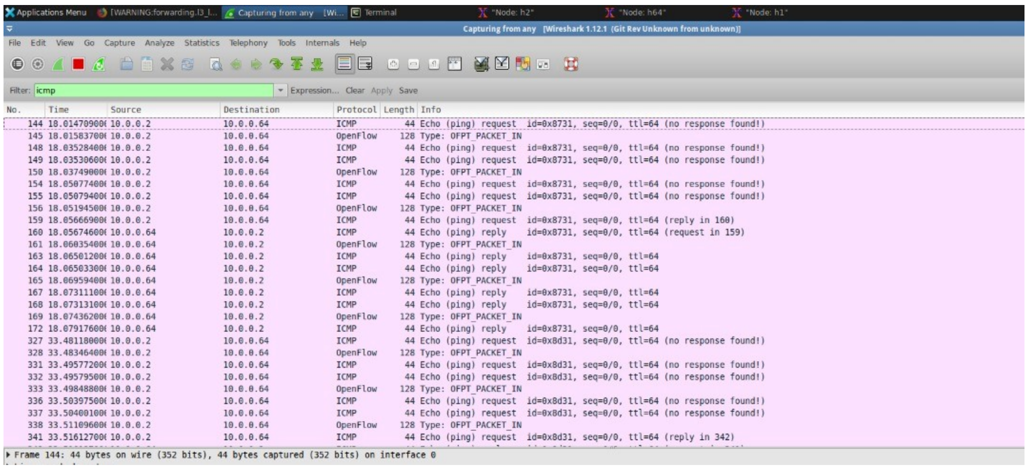
```
ubuntu@sdnhubvm:~[22:20]$ cd pox
ubuntu@sdnhubvm:~/pox[22:20] (eel)$ python ./pox.py forwarding.l3_learning
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-06 1] connected
INFO:openflow.of_01:[00-00-00-00-00-07 7] connected
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-04 8] connected
INFO:openflow.of_01:[00-00-00-00-00-09 6] connected
INFO:openflow.of_01:[00-00-00-00-00-05 9] connected
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:openflow.of_01:[00-00-00-00-00-08 4] connected
```

Following are the steps we used to perform the Smurf Attack:

1. Target IP address is to be identified by the attacker PC through nmap.
2. Intermediary site (a broadcast address) is to be identified by attacker which helps in amplifying attack.
3. Large amount of traffic/packets (ICMP request) will be sent by attacker to the broadcast address at particular intermediary sites.
4. These intermediaries will provide broadcast to all hosts which are there in a subnet(255.255.255.0).
5. Hosts will reply to network and will send the ICMP request to the target PC IP address. The target PC IP address will then reply with ICMP reply packets to all the ICMP request packets. Thus, the denial in service attack (ICMP Smurf Attack) is completed.

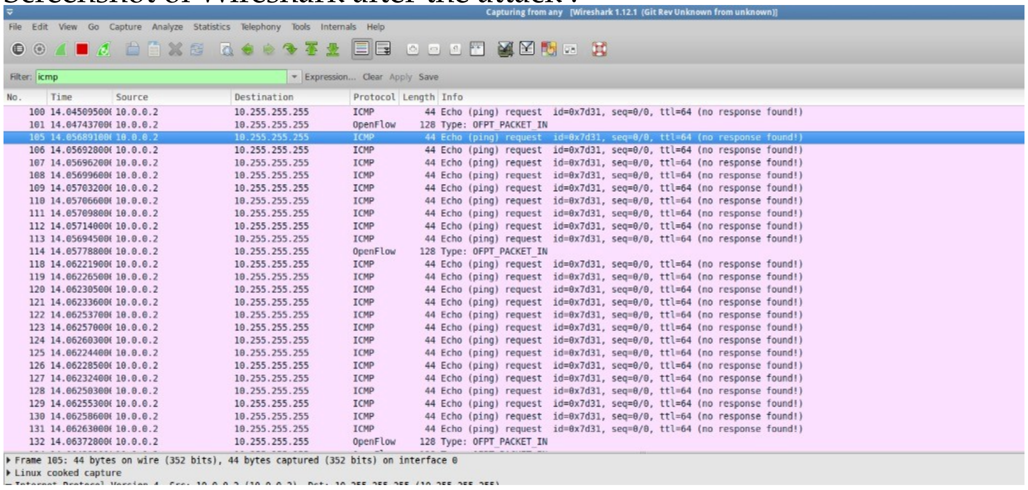
We used Wireshark for sniffing the packets to analyse them.

Screenshot of Wireshark before the attack :


Applications Menu
[WARNING:forwarding.l3_l...
Capturing from any [Wi...
Terminal
"Node: h2"
"Node: h64"
"Node: h1"
Capturing from any [Wireshark 1.12.1 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

```
Filter: icmp

No.  Time         Source      Destination  Protocol Length Info
 144 18.01470900( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8731, seq=0/0, ttl=64 (no response found!)
 145 18.01583700( 10.0.0.2    10.0.0.64    OpenFlow  128 Type: OFPT_PACKET_IN
 148 18.03528400( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8731, seq=0/0, ttl=64 (no response found!)
 149 18.03530600( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8731, seq=0/0, ttl=64 (no response found!)
 150 18.03749000( 10.0.0.2    10.0.0.64    OpenFlow  128 Type: OFPT_PACKET_IN
 154 18.05077400( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8731, seq=0/0, ttl=64 (no response found!)
 155 18.05079400( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8731, seq=0/0, ttl=64 (no response found!)
 156 18.05194500( 10.0.0.2    10.0.0.64    OpenFlow  128 Type: OFPT_PACKET_IN
 159 18.05666900( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8731, seq=0/0, ttl=64 (reply in 160)
 160 18.05674600( 10.0.0.64   10.0.0.2     ICMP       44 Echo (ping) reply    id=0x8731, seq=0/0, ttl=64 (request in 159)
 161 18.06035400( 10.0.0.64   10.0.0.2     OpenFlow  128 Type: OFPT_PACKET_IN
 163 18.06501200( 10.0.0.64   10.0.0.2     ICMP       44 Echo (ping) reply    id=0x8731, seq=0/0, ttl=64
 164 18.06503300( 10.0.0.64   10.0.0.2     ICMP       44 Echo (ping) reply    id=0x8731, seq=0/0, ttl=64
 165 18.06959400( 10.0.0.64   10.0.0.2     OpenFlow  128 Type: OFPT_PACKET_IN
 167 18.07311100( 10.0.0.64   10.0.0.2     ICMP       44 Echo (ping) reply    id=0x8731, seq=0/0, ttl=64
 168 18.07313100( 10.0.0.64   10.0.0.2     ICMP       44 Echo (ping) reply    id=0x8731, seq=0/0, ttl=64
 169 18.07436200( 10.0.0.64   10.0.0.2     OpenFlow  128 Type: OFPT_PACKET_IN
 172 18.07917600( 10.0.0.64   10.0.0.2     ICMP       44 Echo (ping) reply    id=0x8731, seq=0/0, ttl=64
 327 33.48118000( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8d31, seq=0/0, ttl=64 (no response found!)
 328 33.48346400( 10.0.0.2    10.0.0.64    OpenFlow  128 Type: OFPT_PACKET_IN
 331 33.49577200( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8d31, seq=0/0, ttl=64 (no response found!)
 332 33.49579500( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8d31, seq=0/0, ttl=64 (no response found!)
 333 33.49848800( 10.0.0.2    10.0.0.64    OpenFlow  128 Type: OFPT_PACKET_IN
 336 33.50397500( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8d31, seq=0/0, ttl=64 (no response found!)
 337 33.50400100( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8d31, seq=0/0, ttl=64 (no response found!)
 338 33.51109600( 10.0.0.2    10.0.0.64    OpenFlow  128 Type: OFPT_PACKET_IN
 341 33.51612700( 10.0.0.2    10.0.0.64    ICMP       44 Echo (ping) request  id=0x8d31, seq=0/0, ttl=64 (reply in 342)

▸ Frame 144: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface 0
```

Screenshot of Wireshark after the attack :


Capturing from any [Wireshark 1.12.1 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

```
Filter: icmp

No.  Time         Source      Destination      Protocol Length Info
 100 14.04509500( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 101 14.04743700( 10.0.0.2    10.255.255.255   OpenFlow  128 Type: OFPT_PACKET_IN
 105 14.05689100( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 106 14.05692800( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 107 14.05696200( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 108 14.05699600( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 109 14.05703200( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 110 14.05706600( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 111 14.05709800( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 112 14.05714000( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 113 14.05694500( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 114 14.05778800( 10.0.0.2    10.255.255.255   OpenFlow  128 Type: OFPT_PACKET_IN
 118 14.06221900( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 119 14.06226500( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 120 14.06230500( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 121 14.06233600( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 122 14.06253700( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 123 14.06257000( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 124 14.06260300( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 125 14.06224400( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 126 14.06228500( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 127 14.06232400( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 128 14.06250300( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 129 14.06255300( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 130 14.06258600( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 131 14.06263000( 10.0.0.2    10.255.255.255   ICMP       44 Echo (ping) request  id=0x7d31, seq=0/0, ttl=64 (no response found!)
 132 14.06372800( 10.0.0.2    10.255.255.255   OpenFlow  128 Type: OFPT_PACKET_IN

▸ Frame 105: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface 0
▸ Linux cooked capture
```

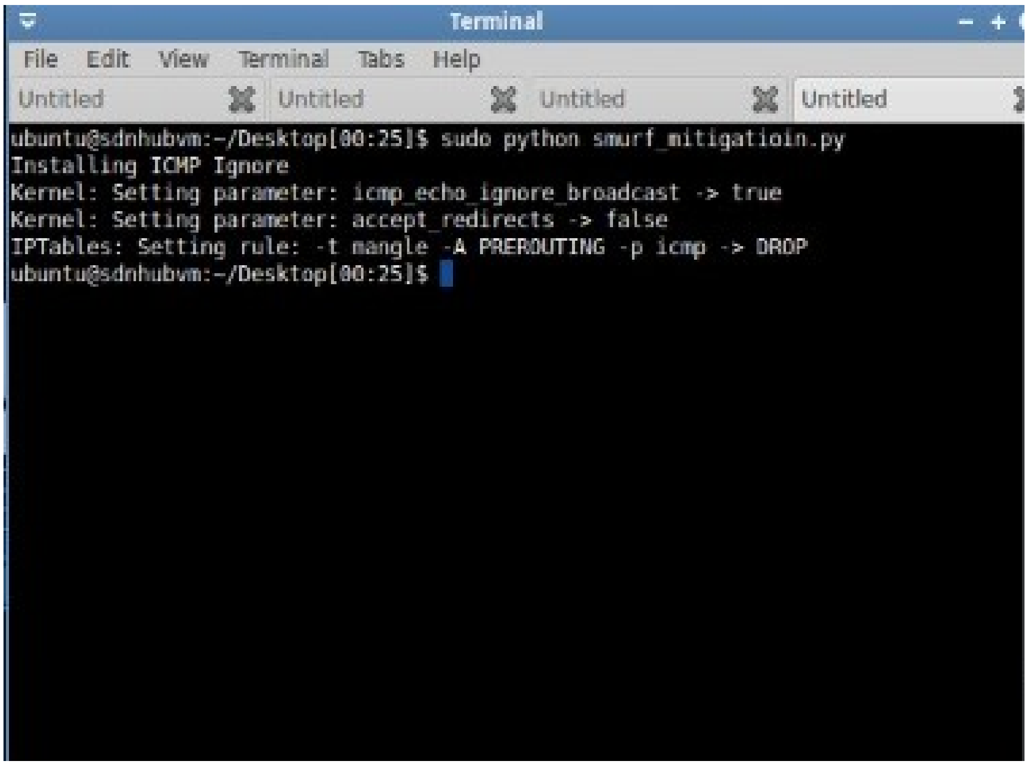
For Mitigating this attack, We created a python code for the controller to add rules into the IP table. In the IP table,

1. we set ICMP Echo ignore Broadcast to true using the function :

```
os.system("echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts")
```

2. We set Accept Redirect to false using :

```
os.system("echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirect")
```



```
ubuntu@sdnhubvm:~/Desktop[00:25]$ sudo python smurf_mitigatioin.py
Installing ICMP Ignore
Kernel: Setting parameter: icmp_echo_ignore_broadcast -> true
Kernel: Setting parameter: accept_redirects -> false
IPTables: Setting rule: -t mangle -A PREROUTING -p icmp -> DROP
ubuntu@sdnhubvm:~/Desktop[00:25]$
```

6 Result and Analysis

In this project, we implemented Deauth, DDoS and Smurf attacks on a virtual topology created using Mininet emulator and Successfully detected and prevention measures are discussed. De-authentication attack is detected by determining the presence of Dot11Deauth frames. DDoS attack is detected using the entropy value. Smurf attack can be prevented by adding rules to the IP table. We faced failure while implementing De-authentication attack on Mininet. For this reason, we identified Mininet-Wifi which supports wifi-extention for the implementation od deauth. We observe that the three attacks are detected within seconds after the execution of the attacks.

7 Conclusion

Ensuring security is an essential subject when coming into IOT. This project presents an effective way to detect and mitigate the attacks approaching the IOT networks at an early stage using SDN principles. In this project, different attacks were modelled to implement it on the IoT network. Here we identified different DoS and DDoS attacks. After the attack was initiated, we identified the behavior pattern of each attack and studied the impact of each attack on IoT network which we used in this project. Each attack had different behavior. So, by analyzing the behavior we can quickly understand the type of attacks.

After modelling and behavior analysis of attacks we identify the attack and mitigate it. For this we use packet filtering in the network device. We use SDN principles and OpenFlow, so that if any device in the network identifies an attack or anomaly, this information will be passed to the master controller which will share this with all the clusters connected to it and updates the rules in the IoT network and secures the network from the specific attack.

The importance of this project increases with the increasing usage of IOT devices. By the development of different mitigation techniques for different attacks which is approaching IOT, we can pave the way for the growth of IOT in a safe and efficient manner.

8 References

[1] S. S. Bhunia and M. Gurusamy, "Dynamic attack detection and mitigation in IoT using SDN," 2017 27th International Telecommunication Networks and Applications Conference (ITNAC), 2017, pp. 1-6, doi: 10.1109/ATNAC.2017.8215418.

[2] N. Dayal and S. Srivastava, "Analyzing behavior of DDoS attacks to identify DDoS detection features in SDN," 2017 9th International Conference on Communication Systems and Networks (COMSNETS), 2017, pp. 274-281, doi: 10.1109/COMSNETS.2017.7945387.

[3] Poudél, Roshan. (2020). Practically Detecting WiFi Deauthentication Attack, 802.11 Deauth Packets using Python and Scapy. 10.13140/RG.2.2.18826.49602.

[4] Mousavi, Seyed St-Hilaire, Marc. (2015). Early detection of DDoS attacks against SDN controllers. 2015 International Conference on Computing, Networking and Communications, ICNC 2015. 77-81. 10.1109/ICCNC.2015.7069319.