T.C.

ALTINBAS UNIVERSITY

Institute of Graduate Studies

Information Technologies

DETECTION OF DDOS ATTACKS BASED ON ENTROPY-PCA IN SDN

Master of Science

Hasen Hadi Sadiq AlMomin

Supervisor

Asst. Prof. Abdullahi Abdu Ibrahim

Istanbul, 2020

DETECTION OF DDOS ATTACKS BASED ON ENTROPY-PCA IN SDN

by

Hasen Hadi Sadiq AlMomin

Information Technologies

Submitted to the Graduate School of Science and Engineering

in partial fulfillment of the requirements for the degree of

Master of Science

ALTINBAŞ UNIVERSITY

2020

The thesis titled “DETECTION OF DDOS ATTACKS BASED ON ENTROPY-PCA IN SDN” prepared and presented by “HASEN ALMOMIN” was accepted as a Master of Science Thesis in Information Technologies.

_____ _____

Asst. Prof. Abdullahi Abdu Ibrahim

Supervisor

Thesis Defense Jury Members:

Asst. Prof. Dr. Abdullahi Abdu IBRAHIM	School of Engineering and Natural Sciences, Altinbas University	_____
Asst. Prof. Dr. Çağatay AYDIN	School of Engineering and Natural Sciences, Altinbas University	_____
Asst. Prof. Adil Deniz DURU	Physical Education and Sports, Marmara University	_____

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Approval Date of Institute of Graduate Studies: ____/____/____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Hasen Hadi AlMomin

DEDICATION

This thesis is dedicated to Israa Al Tarakany, who has been my spiritual nourishment through the challenges of graduate school and broader life. I am truly lucky and thankful that I've had you in my life.

This work is also dedicated to my parents, who have taught me that the greatest satisfaction can only come from pushing my own limits.

In addition, I would dedicate this work to the rest of my family

(Dr. Hayder Al-Musawi, Dr.Shaymaa Al-Khafaji, Dr.Ahmed J. Abid, Gussun Almomin, Furat Mumin, Tabarek Rubaye, Marwa Hadi, and Rand Almomen)

They have always been providing help, support and assistance throughout my studies.

ACKNOWLEDGEMENTS

I want to acknowledge everyone who played a role in my academic accomplishments.

First of all, my parents, who have supported me with love and understanding. Without you, I could never have achieved this current level of success.

Second, my advisor, Dr. Abdullahi Abdu Ibrahim, who has provided patient advice and guidance throughout the research process.

Special thanks to my classmate, whom I am proud to have known and honored, Ahmed Mohammed Abbas

I also would like to thank the Iraqi Parliament, represented by the Presidency of the Parliament, and the Director-General of the Communications and Information Technology Department of the Parliament, represented by

Mr. Majid Khudhur,

Director-General,

Mr. Ayad Alrubaiy,

Head of the Networks Department,

And my colleagues in the department. With special thanks to my friends, Dr.Ahmed H. Obeid, Dr. Mohamed Adnan, and Dr. Kawa M. Kaky

Thank you all for your unwavering support.

ABSTRACT

DETECTION OF DETECTION OF DDOS ATTACKS BASED ON ENTROPY-PCA IN SDN

AlMomin, Hasen Hadi Sadiq,

M.Sc., Information Technologies, Altınbaş University,

Supervisor: Abdullahi Abdu Ibrahim

Date: December/2020

Pages: 70

Software Defined-Network (SDN) is still lately attracting much new research of interest. This research is a natural result of many significant companies' tendency to manufacture devices for the SDN network and turn their infrastructure into this network type. SDN networks introduce a new design that works on splitting the control plane from the data plane to allow a broader field to smoothly and efficiently program the network to gain much simplicity compared to the conventional networks. Any change in conventional networks required a re-configuration of a set of resources for the network. Whereas in new SDN network needs one person with knowledge on the control layer (controller) to control all network resources and update rules with lease time. However, there are still security concerns surrounding the new network architecture because it is one failure shot that is attracting many cyber-attacks. One of the most critical attacks that increased lately is the Distributed Denial of Service (DDoS), which works to make the service unavailable for an unknown period. This thesis will suggest a method to detect a DDoS attack that targets one or multiple victims concurrently by combining two algorithms of Machine Learning (ML), which is Entropy and Principal Component Analysis (PCA). Also, we examined the efficiency of our schema through a Mininet emulator and a pox controller and using OVS switches as a switch. We have obtained high detection accuracy to detect DDoS attacks. In

addition, we found that our method achieved the best results compared to other methods of attack detection.

Keywords: SDN, DDoS Attack, Controller, PCA, Entropy, Machine Learning

TABLE OF CONTENTS

Pages

ABSTRACT ..... **vii**

LIST OF TABLES ..... **xii**

LIST OF FIGURES ..... **xiiii**

LIST OF ABBREVIATIONS ..... **xiii**

LIST OF SYMBOLS ..... **xiv**

1. INTRODUCTION **15**

1.1 INTRODUCTION 15

1.2 PROBLEM STATMENT 15

1.3 RESEARCH OBJECTIVR AND CONTRIBUTION 16

1.4 SCOPE 17

2. BACKGROUND AND LITERATURE WORK **18**

2.1 SOFTWARE DEFINED NETWORK 18

2.2 ADVANTAGES AND DISADVANTAGES OF SDN 20

2.3 OPENFLOW PROTOCOL 21

2.4 OPENFLOW PROCEDURE 24

2.5 SDN SECURITY 24

2.6 DDOS ATTACKS 26

2.6.1 DDoS Attacks Types 26

2.7 RELATED WORKS ON DDOS ATTACK DETECTION OVER SDN 27

3. METHODOLOGY AND PROPOSAL METHOD **32**

3.1 ENTROPY 32

3.1.1 Entropy Destination IP Address Variance Impelmentation 33

3.2 PRINCIPAL COMPONENT ANALYSIS (PCA) 36

3.3 OUR PROPOSED METHOD (ENTROPY-PCA) 39

3.4 ATTACK MITIGATION 42

4. SIMULATION AND RESULTS 43
4.1 ENVIRONMENT 43
4.2 MININET 43
4.3 TRAFFIC DATA GENERATION 43
4.4 WIRESHARK 44
4.5 PERFORMANCE METRICS AND RESULTS 44
4.5.1 Confusion Matrix 44
4.6 SIMULATION SCENARIO AND RESULT OF OUR PROPOSED METHOD 47
4.6.1 Single Victim Attack 48
4.6.2 Multiple Victim Attack 50
4.6.3 Accuracy Rate of Entropy-PCA Method 51
4.7 COMPARSION WITH A RELATED PREVIOUS WORK 53
5. CONCLUSION AND FUTURE WORK 55
5.1 CONCLUSION 55
5.2 FUTURE WORK 55
REFERENCES 56
APPENDIX A 62

LIST OF TABLES

Pages

Table 2.1: Advantages of SDN 20

Table 4.1: Confusion Matrix 45

Table 4.2: Total Dataset of our simulations 52

Table 4.3: Performance Metrics 53

LIST OF FIGURES

Pages

Figure 2.1: Altınbaş University 18

Figure 2.2: Comparing Conventional Network and SDN. 18

Figure 2.3: SDN functional Architecture 19

Figure 2.4: Basic architecture of an OpenFlow device 22

Figure 2.5: A Flow Table on OpenFlow device 23

Figure 2.6: Supported Counter Fields 23

Figure 2.7: OpenFlow Flow Processing Procedure 24

Figure 2.8: Comparison of all the parameters of the three ML algorithms. 31

Figure 3.1: Entropy flow chart 31

Figure 3.2: Correlation between multiple attributes value 37

Figure 3.3: Entropy-PCA flow chart. 41

Figure 4.1: Network Topology 47

Figure 4.2: Single DDoS Attack Topology 48

Figure 4.3: Wireshark Output: Attack Traffic on single DDoS Attack host 49

Figure 4.4: Multiple DDoS Attack Topology 50

Figure 4.5: Wireshark Output: Attack Traffic on multiple DDoS Attack hosts 51

Figure 4.6: Comparison of the accuracy rate of detection methods 53

LIST OF ABBREVIATIONS

SDN	:	Software Defined Network
QoS	:	Quality of Service
PCA	:	.Principal component analysis
DDoS	:	Distributed Denial of Service
CN	:	Conventional Network
ONF	:	Open Networking Foundation
SNMP	:	Simple Network Management Protocol
TLS	:	Transport Layer Security
UDP	:	User Datagram Protocol
TCP	:	Transmission Control Protocol
ICMP	:	Internet Control Message Protocol
SVM	:	Support Vector Machine
KNN	:	k-Nearest Neighbors
NB	:	Naïve Bayes
ANN	:	Artificial Neural Networks
CLI	:	Command Line Interface
OVS	:	Open vSwitch
ML	:	Machine Learning
TTL	:	Time to Live

LIST OF SYMBOLS

λ : Eigen Value

V : Eigen Vector

$\hat{x}$: Modeled Subspace

$\tilde{x}$: Residual Subspace

σ : Sigma

X' : DeltaY Subspace

1. INTRODUCTION

1.1 INTRODUCTION

SDNs are a new networking concept that has received much attention as a practical technology concept that works on flexibility in the network. SDN Network is usually divided into two layers instead of three on a conventional network (CN), where the first layer is called the control plane layer, which controls and makes all significant decisions in the network and checks the destination address. In contrast, the second layer is the data plane layer, representing the distributed device that connects the control plane with the end-user and implements the controller's rules in the control plane.

The decouples provide many new services, for instance, traffic selection, applying new policies, access control, bandwidth management, and QoS [1].

1.2 PROBLEM STATEMENT

Many security concerns and problematic issues are showed up on SDN because of the centralization design of SDN. One of the most critical security threats to SDN networks is DDoS (Distributed Denial of Service). DDoS attacks jeopardize data and resources integrity and availability; it performs the attacks by multiple compromised devices called botnets or zombies that use spoofed source IP addresses targeting one or multiple end-users or servers the resources by making them unavailable to legitimate users for some time. The SDN controller is considered a focal point for the attack, so it is an ideal point of attack to knock down the entire network. These type of attacks has a substantial impact on SDN switches and controllers [2].

Each attack packets came on the switch; if the flow table does not match the rule, a new rule is created. The packet header is transferred to the controller after the packet has been stored in its memory. Send many randomly spoofed packets that drain the switch's storage and install numerous new flow entries in the flow table.

Once SDN switches receive a new packet, it will search for a matching rule in the flow table; if there are no exciting rule founds, then the header of that packet will transfer to the controller to decide what to do with this new packet [3]. When the controller replay, a new rule will be

created on the flow table of OpenFlow switches. In a DDoS attack, a large amount of random spoofed IP address that reached the switch with no matching rule that exhaust the switch storage because of these many new requests being processed and stored in the flow table.

Eventually, the combination of spoofed IP addresses from DDoS attacks and legitimate users packet causes the controller to drop down and not handle any more requests. This means that any new legitimate user cannot reach the controller. Since the controller is at the core of the SDN architecture, the whole network will crash as the same security problem will arise, whether or not there is a backup controller [4].

Distributed Denial of Service (DDoS) significantly influences SDN compared to Conventional Network. Hence, a method of detecting these attacks is necessary and adequate mitigation will be taken after this early detection.

Since recently, large companies have started using software-defined networks in their infrastructure, so it is critical to find a lightweight method to detect attacks that do not add additional load to the network. Also, our method will reduce further action for the required time to carry out a mitigation measure [5].

1.3 RESEARCH OBJECTIVE AND CONTRIBUTION

This thesis identified a detection method and analyzed the results of a DDoS attack on SDN environments. A way of protecting SDN switches and controllers against DDoS security threats is proposed by inserting the controller with a lightweight detection code. The significant contributions to this research are:

- We present and implement an efficient method of detecting DDoS attacks by combining two powerful ML algorithms, which are (PCA) Principal Component Analysis and Entropy. This method is a lightweight code that detects a DDoS attack aimed at a single target or multiple network victims.
- Demonstrate how a DDoS attack will completely blind the controller over the SDN network.

- Successfully deployed the combined algorithm with OpenFlow switches via Mininet and POX controllers.
- The effectiveness of the algorithm is evaluated and demonstrated in the Mininet test scenario.

1.4 SCOPE

The organization of this thesis is in the following order:

- **Chapter 2** includes a background of SDN, OpenFlow protocol, and DDoS attacks followed by samples of literature review of previous methods and results for detection.
- **Chapter 3** gives insights into our two algorithms and presents our Entropy-PCA method by step by step description and equations.
- **Chapter 4** discuss the results of our simulation work, along with their detailed analysis.
- **Chapter 5** concludes the thesis's conclusion and the further work that needs to be done in our work.

2. BACKGROUND AND LITERATURE WORK

Figure 2.1: Altınbaş University 10th year banner.

2.1 SOFTWARE-DEFINED NETWORK

SDN offers a unique method of a network that simplifies some networking features such as design and management. Every change in the network in the CN required a change in the configuration of several network resources, which often resulted in high costs[6]. As shown in figure (2.2) below, SDN splits the data plane from the control plane, which is demonstrated in figure (b), while these two-layer usually combine in one layer on the CN that is shown in figure (a) [7]. The application layer is standard between the two different networks.

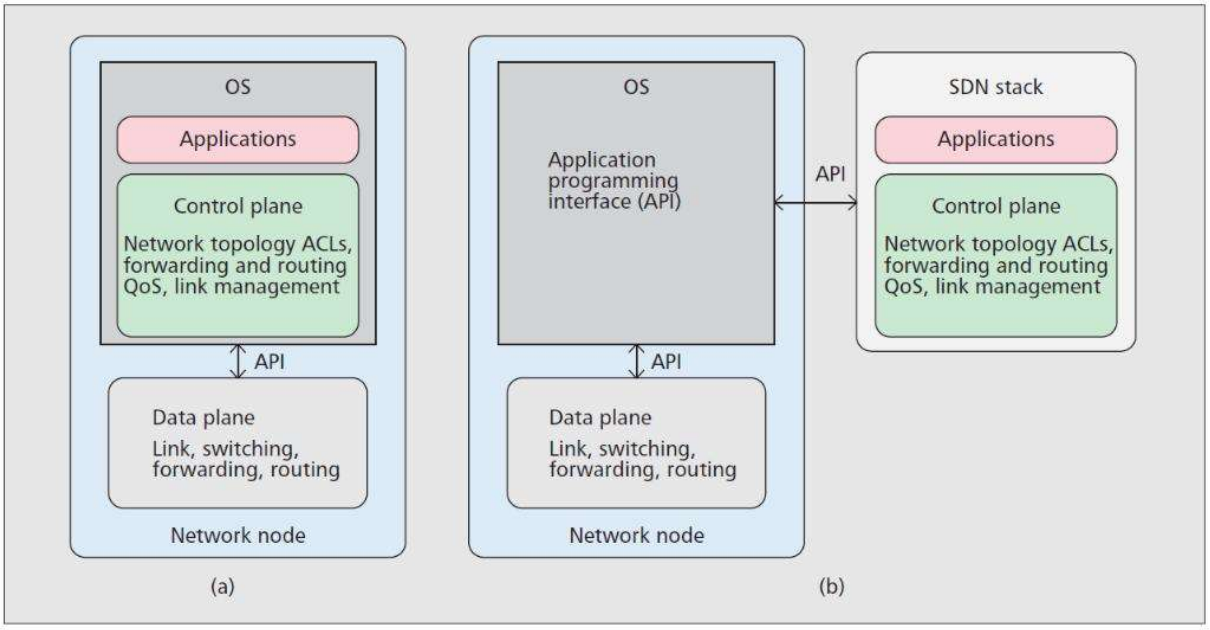

Figure 2.2: Comparing Conventional Network and SDN.

We can consider the conventional network as a hardware-based network, so it depends on the hardware components to make a decision for routing and path selection, such as switches and routers. In contrast, an SDN network is a software-based network that depends on the programming controller to make the decision and install rules on other parts of the network, such as Vswitch. SDN produces a novel way to control the entire network from a single node called a controller on the control plane layer[8]. The controller communicates with other network nodes to install new rules on the data plane's devices through the southbound in OpenFlow protocol [9]. OpenFlow Protocol has two ways of communication one from the controller to the network parts called southbound and the network application to the controller called northbound, as shown in figure (2.3). Control plane applications such as load balancing interact with data planes via application programming interfaces (APIs) at the SDN architecture application layer. The centralization of SDN gives the network the ability to extend the network with new hosts with a minimum effort of configuration and programming and gives the ability to control traffic.

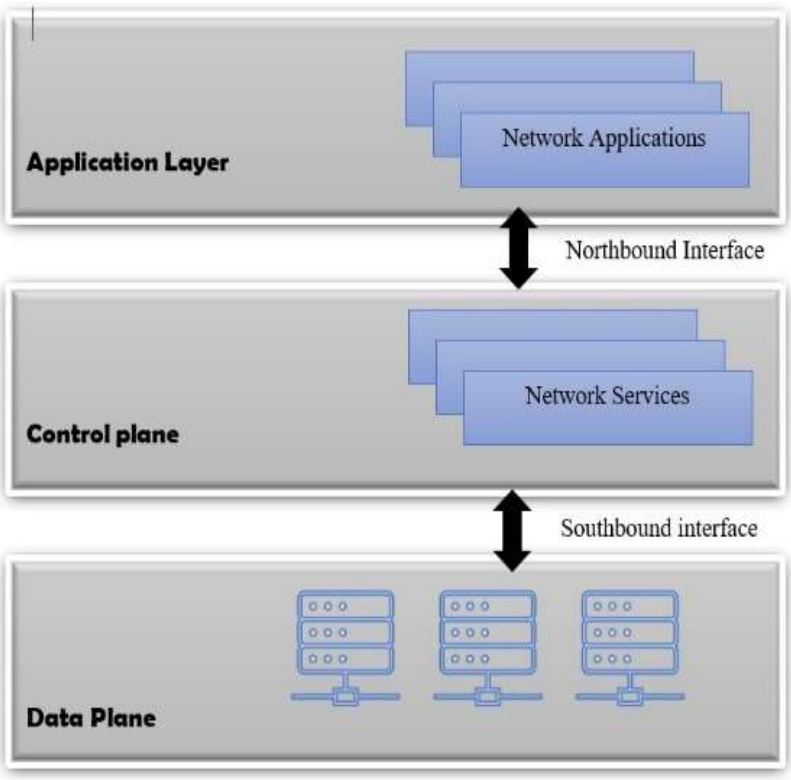

Figure 2.3: SDN functional Architecture**.**

2.2 ADVANTAGES AND DISADVANTAGES OF SDN

SDN separation layers bring many advantages, such as simplicity and control management using programming. Such features boost network efficiency and strengthen the capacity to create policy-driven network oversight and automated configuration [10]. We summarized the features of SDN in contrast with TN in table 2.1 below:

Table 2.1: Advantages of SDN

	SDN	**TN**
Features	- Data and control plane isolated -Programmability	-Complex control management
Configuration	- Automated configuration from centralization point	-Manual configuration for almost every new device
Performance	-Globalized information control	-Limited info
Innovation	- Easy implementation of software at lower cost - Adequate environment for test isolation -A quick process implementation	-Hardware implementation with extra cost -Restricted test environment - A long process through standardization

SDN function with a global vision to the entire network SDN to strengthen decision making and enhance network behavior performance by adaptation programmatically. Furthermore, this centralization gives us the ability to collect the traffic status on the network in real-time. Improving configuration can be considered as an essential feature of SDN through the centralization point. Due to a variation of equipment manufacture of network devices, therefore, TN required a lengthy manual process of configuration procedure, which also needs more effort and is more likely to has error-prone and in order to address these errors, it must take sufficient time to handle and tackle these network troubleshooting. While in SDN, we can make

configuration for all devices automatically by one central point, which is located in the control layer, specifically from the controller device, which works programmatically so the SDN controller can control many TN features such as routing, switching, firewall, load balance, SNMP … et cetera.

SDN will provide a way to maximize network efficiency globally due to unified control throughout the global network. It operates by exchanging information from different network layers architecture. These SDN features lead to well-designed centralized solutions to several difficult performance optimization challenges. For example, application problems like traffic scheduling, mobility management from one end to the other energy-efficient operation, load-balanced packets also support the quality of service (QoS) that can now quickly be addressed to check their performance in the network.

SDN innovation is also used to reduce the time needed to install and expand the network and ease installation with a significant reduction in cost through automated programming for the central control unit. On the other hand, SDN offers a programmable network architecture to promote creativity. The SDN network's role can be used to implement, develop, and deliver new ideas and applications [11]. Experiments on a real system are possible due to the high configurability of SDN since this provides a reasonable separation between layers. The transformation from an experimental to an operational phase of the implementation of technologies can be carried out smoothly.

2.3 OPENFLOW PROTOCOL

OpenFlow is an open-source protocol that SDN architecture uses as a standard protocol to communicate. Open Networking Foundation (ONF) is capable of managing the OpenFlow standards and has defined it as the first open standard protocol dealing with SDN and the transmission between its layers. The controller uses OpenFlow to configure the devices on the data plane layer. OpenFlow version 1.5.1 is the current version, while 1.6 is currently available but only accessible for members of ONF since September 2016 [12].

OpenFlow permits the controller to manages the switches at the data plane layer [13]. Transport Layer Security (TLS) manages transmission between switches and controllers. Each SDN switch should have one or more flow tables to keep the controller's flow entries.

An OpenFlow device is implemented packet forwarding, which identifies by the controller as appeared in figure (2.4) below:

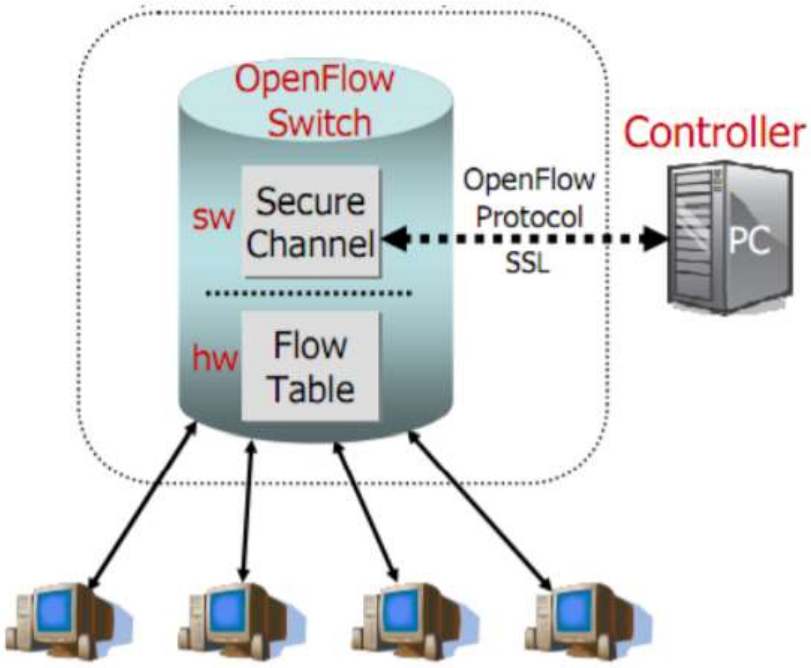

Figure 2.4: Basic architecture of an OpenFlow device**.**

There is a range of matching actions available in OpenFlow, such as drop-out, packet forwarding, additional queries on other flow tables, counter field rewriting, and so on [14]. The flow table is shown in Figure (2.5).

In addition, the OpenFlow device provides a two-way communications channel for the controller, a first-way connection to the controller which is used to request a controller action and to report the network status. In contrast, a second-way connection route from the controller to the OpenFlow device is used for sending messages to update OpenFlow table [15].

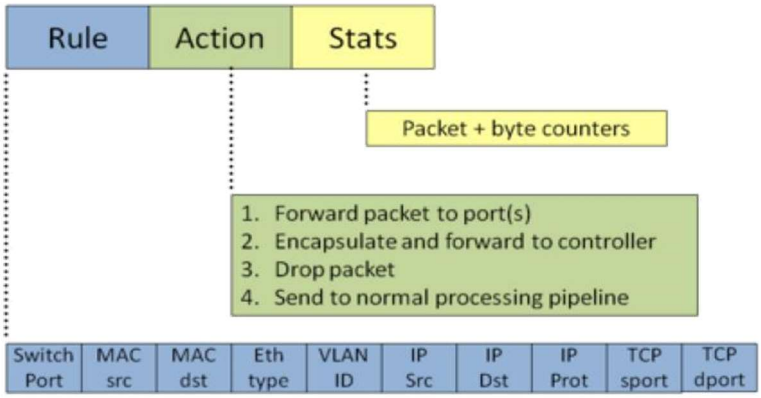

Figure 2.5: A Flow Table on OpenFlow device**.**

The four available statistical scopes in the flow table are kept by a table, flow, port, and py queue counter [16].

Scope	Values	Size (bits)
Per Table	Active Entries	32
	Packet Lookups	64
	Packet Matches	64
Per Flow	Received Packets	64
	Received Bytes	64
	Duration (seconds)	32
	Duration (nanoseconds)	32
Per Port	Received Packets	64
	Transmitted Packets	64
	Received Bytes	64
	Transmitted Bytes	64
	Receive Drops	64
	Transmit Drops	64
	Receive Errors	64
	Transmit Errors	64
	Receive Frame	64
	Alignment Errors	64
	Receive Overrun Errors	64
	Receive CRC Errors	64
	Collisions	64
Per Queue	Transmit Packets	64
	Transmit Bytes	64
	Transmit Overrun Errors	64

Figure 2.6: Supported Counter Fields [6]**.**

2.4 OPENFLOW PROCEDURE

The details of the steps to route the path flow among two switches across two hosts within SDN networks are explained by Jad Naous et al. [13]. During the first initial phases, the switch flow tables are blank, as shown in Figure (2.7), consisting of 8 steps. If a packet comes in step 1, the controller is forwarded to step 2 directly and because the switch flow table does not match. The received packet is checked by the controller and determines the actions to be followed, such as drop or forward it, and then the controller produces new flow input back towards the switches. The switches now receive the flow entry in the path the packet is going through in step 3, and then the packet is sent to the host receiver in step 4 and step 5 sequentially. On the other three steps, every new packet with the exact flow is pathed directly because of the matching rule in flow entry[14].

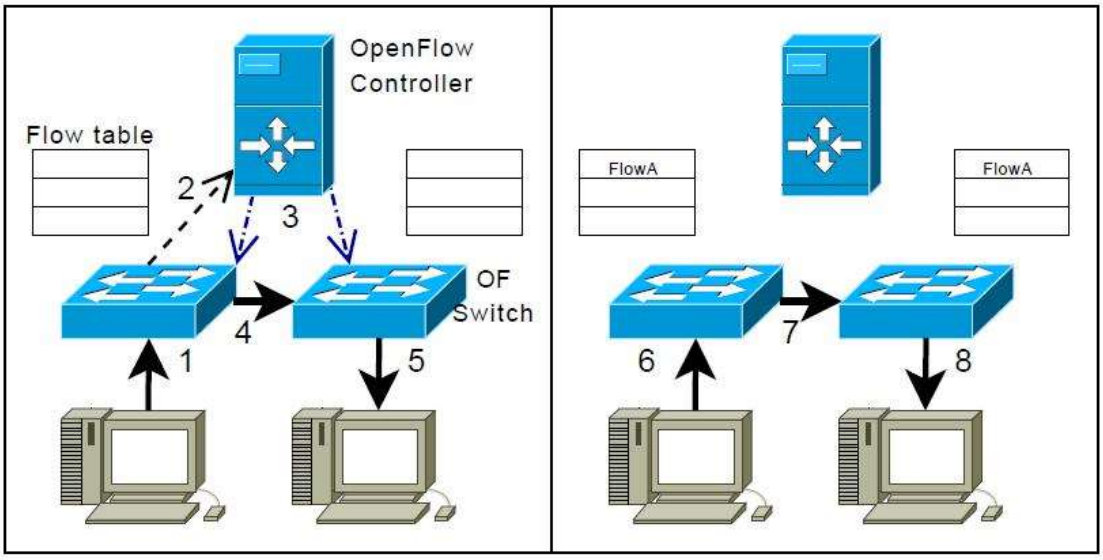

Figure 2.7: OpenFlow Flow Processing Procedure [7]**.**

2.5 SDN SECURITY

Indeed, the process of separating the data layer from the control layer led to a qualitative leap in the world of networks. However, this step led to the emergence of new gaps that did not exist in conventional networks, but interest began to grow during the previous period, and there was a real tendency to address security and reliability issues within SDN networks. Therefore, many

researchers began to address these dangers, weaknesses, and threats that resulted from this new technology, and these researches offer some solutions that should be taken into consideration since the first step in building the network, which may contribute to avoiding and confronting these threats and reducing their risks [17].

Conventional networks have natural immunity against common attacks due to the nature of their closed devices, durable design, software homogeneity, and decentralized control. For example, suppose an attacker takes advantage of a weakness of devices manufactured by a company. In that case, the network will only be affected in the part that contains devices belonging to the same company, as for the rest of the devices, it will not be affected since it belongs to other manufacturers. As for the SDN, the existence of the standard OpenFlow protocol among all companies will increase the risk of threats and spread common failures among all companies [18]. SDN networks produced an excellent idea in the world of networks. However, they simultaneously increased the risks and threats, which necessitated more research and discussions of security issues, reliability, and solutions that must be considered in these networks' design.

The controller considers a favorable spot to monitor the traffics and detect abnormal traffic because all network packets are pathed over the controller [9]. Besides, the programmability feature makes the mitigation process deployed much faster and efficiently. To diminish the impact of the attacks, the controller can install or update flow rules. As mentioned in the abstract, the controller is a preferred attack spot for an attacker. Multiple network controllers could use to avoid the collapse of the network.

DDoS is one of the highest threat security concerns [19]. As previously stated, the data plane device includes a flow table within the OpenFlow procedure. The switch communicates with the controller through a secure transport layer protocol and asks about new packets that do not exist on the switch's flow table to decide what action should be taken. Each flow entry has a matching procedure that guides the data plane. Once the packet reaches the switch, the action defined in the flow entry will be performed, and the header checks if a matching rule exists on the flow table. If not, the packet header would be sent to the controller. In the controller, the inbound header is processed to set a new rule on the flow table switch [20].

2.6 DDOS ATTACKS

DDoS is known to be one of the most critical security risks to many researchers and technology undertakings. Attacks of DDoS threaten data integrity and access to services, rendering legitimate users unavailable resources via multiple compromise systems (zombies or botnet) typically infected by a Trojan virus that aims to overload systems with flood packets one or multiple device victims. The SDN controller will be a proper spot for the entire network to such attacks. A spoofed IP addresses used by this type of intrusion attacks. The DDoS attack also has a critical impact on the controls and switches [15][21].

Each packet attack arrived at the switch; if no rule matched in the flow table, a new regulation would be created. The packet header is transferred to the controller after the packet is stored in the memory[22][23]. The switch memory will drain up because of the limitation of flow entries required to install when the attackers send a large amount of flood packet towards their victims.

Eventually, the server sinks to the point of the controller's inability to manage packages that arrived from a mixture of spoofed DDoS packets and legitimate user packets [24]. Consequently, any new legitimate user cannot access the controller. The whole network will also collapse because the controller works as an SDN architecture core since the same security problem will arise, whether or not there is a backup controller.

2.6.1 DDoS Attack Types

DDoS attacks take various forms; in this section, we will briefly discuss the three highest reported types of these widely used attacks.

- User Datagram Protocol (UDP) Flood

It considers as the most commonly used attack, in which botnets or zombies transmit a large number of packets with a random or specific application port. However, the ping will return to the spoofed source IP address since no waiting time exists [25]. The process overwhelmed the victim with more requests than the device victim can handle until the device become unreachable for all legitimate users.

In this study case, we will simulate this type of attack due to its importance and considered it one of the most challenging types to detect.

- Transmission Control Protocol (TCP) SYN Flood

TCP SYN is based on manipulation and understanding of the TCP three-way handshake connection initialization process. The attack works via directing the victim's device with TCP SYN flood requests until the victim cannot handle it as usual [26]. In typical operation, the TCP SYN has to dedicate a port and a CPU process to listen for SYN-ACK. Nevertheless, the SYN-ACK is never sent when it comes to this type of attack.

- ICMP (Ping) Flood

Ping is another basic DDoS attack form that floods the victim with a vast amount of ICMP echo request packets. The receiving server accepts requests from the source IP address and sends an ICMP echo replay back to the same source IP address [27]. ICMP flood usually targeting a bandwidth of the network instead of the server's capacity. When the threshold capacity is hit, the server is unable to respond because of network congestion.

2.7 RELATED WORKS ON DDOS ATTACK DETECTION OVER SDN

Lots of research has been conducted into the SDN environment focused on DDoS attacks. In [28], The suggested: JESS Joint Entropy as a detection method depends on entropy and protocol experiments. The target IP address, packet size, TCP flags, and the randomness of the attacker Network are measured for accurate access to the attacker information, such as the source IP address. The attack is observed, and where entropy data vary in normal and attacked transport, a mitigation module begins execution. Random traffic is generated to determine the normal attack. The packet then starts to drop when the ratio of entropy value is higher than the threshold set.

In [4], entropy often used traffic to identify, which may become a false alert if many destination IP traffic was targeted and regarded as normal traffic. This article demonstrates how DDoS attacks could exhaust controller resources by providing a solution for the detection of those attacks based on an entropy variation of the IP address. This approach will track DDoS in the first five hundred attack traffic packets. A lightweight and efficacious solution for detecting

DDoS attacks was presented. Through using the core role performed by the controller in the SDN, the destination IP address was used to detect attacks in the first 250 packets of malicious packets. Therefore, the threshold chosen for attack traffic is the minimum possible.

Nevertheless, this threshold detection rate was 96%. Besides the early detection of the attack, the approach is straightforward in terms of both resources and features. This gives both the controller and the targeted host the detection.

The authors in [29] present a comprehensive framework to detect and mitigate DDoS attacks. The architecture is therefore not suitable for small networks. They propose a new DDoS attack detection and mitigation architecture for a wide-range network that includes a clever SDN-built community. Their proposed architecture meets the application-specific criteria of DDoS attack detection and mitigation. This paper provides two major contributions. They first include a detailed analysis and debate on DDoS-based SDN attacks' detection and prevention mechanisms and identify the attacks regarding detection techniques. Second, they suggest a constructive SDN Protection System (ProDefense) exploit SDN features for network security. They explain how such an architecture could be used to protect smart city applications.

Two issues have been discussed through the concept of an effective prevention and mitigation method for DDoS attacks. ProDefence provides device-specific network traffic threshold requirements for their proposed framework. This allows flexible standards for identifying DDoS attacks to be introduced. ProDefense also uses a distributed controller framework, which enables load balancing and eliminates system failure possibilities. ProDefense is intended to be used in a large spectrum of applications, including smart grid, e-governance, and physical networks. The tolerance for network attacks varies across all these applications. In order to ensure the rapid implementation of network rules, ProDefense will require continuity between all controllers. In all programs or scripting languages compatible with the SDN controller used can be implemented a ProDefense code.

Nevertheless, users use node.js to implement key ProDefense modules such as threshold sensors and adaptive filters. They picked node.js since it is an async IO model that enables simultaneous interactions with distributed controllers and monitoring systems without blocking the SDN

application. The document still restricts the conduct of comprehensive experiments using the ProDefense framework to study Smart City applications' efficiency.

The approach used in [30], throughout this document, they have introduced a method to detect and prevent DDOS attacks from SDN using the switch to collect the flow statistics and port stats periodically. If the measured rate of a packet is higher than the defined threshold, the source cannot send additional packets by modifying the forwarding logic from the flow table established by the switch. They have taken the method in which traffic statistics are being obtained from different switches. Upon selection, they measured the rate and bandwidth of the packets, which shot high values when the attack occurs. The sudden rise would detect the attack that is then avoided by modifying the host node's forwarding logic to drop the packets rather than forwarding them. Following that, no more packets are sent, and then the transmission rule in the flow table is disabled. They are finding out about improvements in packet rate and bandwidth for intrusion detection and attack prevention.

Therefore, they figure out the increase in the packet rate and bandwidth thread to detect the attack. In order not to forward the packets coming from malicious hosts, they adjust the forwarding logic of the switch flow table.

Reference [31] used 12 features to select the trained database and test ML detection algorithms individually. DDoS attacks in SDN have been detected with machine-based learning models in this study. First, in normal and DDoS attack traffic, spectacular SDN features for the data set were obtained. A new dataset was then generated with the current dataset using a function selection methodology. Feature selection algorithms have been chosen to optimize the systems, make the analysis simpler, and require shorter training times. All datasets were trained and evaluated with Vector Machine Support Models (SVM), Naive Bayes (NB), Artificial Neural Network (ANN), and K-Nearest Neighbors (KNNs) classifications, which had been generated with and without feature selection methods. The test results demonstrated that use the KNN classificatory wrapper selection achieved the highest accuracy rate in the DDoS attachment detection (98.3%). That research shows that machine learning and functional selection methods can give high correctness for detecting DDoS attacks in SDN with good load and time reduction. However, these tests lacked the types of DDoS attacks on subnet IP addresses.

FADM presents [32], an active featherweight scheme in order to discover and reduce DDoS flood attacks in the SDN environment. FADM is a simple and effective SDN DDoS attack detection and mitigation mechanism. Initially, the information on network traffic is obtained from the SDN controller and sFlow agencies. Network functionality is then calculated using an entropy-based approach, and a network abnormality is defined using the SVM classificatory. The timeliness and accuracy of attack detection are effectively improved through the use of these methods together. We propose an effective attack mitigation strategy based on white lists and traffic transfer to maintain the main network functionality. They have used the CT-based approach and the sFlow-based method in a specific network scenario to increase data collection quality. They have then used the entropy approach to quantify network changes and use the SVM classification to classify the network status as normal or anomalous. They offer an effective mitigation function based on the white list and the dynamic changing of the forwarding rules to secure the network for providing normal service in the event of DDoS attacks.

Conclusions from this study suggest that FADM can reliably determine multiple DDoS flood attacks that allow the network to recover quickly. Furthermore, the results show that FADM's overhead is low. It is unable to analyze DDoS attacks and botnets on application layers by using SDN and ML attributes.

This paper [33] shows SAFETY, a new early detection and mitigation solution for TCP SYN floods. Security uses SDN with the entropy method for random flow data to evaluate programming and a wide-visibility approach. Entropy information includes the target address and few TCP flag attributes. To demonstrate the effectiveness and efficiency of SAFETY, they install it in the Floodlight controller as an extension module and test it under various conditions. The experimental results indicate that SAFETY substantially increases the processing time up to (13%) compared to other methods. Other parameters are tested and enhanced in different situations, such as CPU Usage on the controller and the attack detection time. They reveal that the entropy can efficiently measure the degree of the randomness of the received packets on the SDN controller, based on traffic features such as the clubbing of destination IP addresses and the TCP flags and after a time-dependent window sequence.

The paper emphasizes the fact that it must not only be assumed to be a single victim target but must start operating on a comprehensives defense system that detects all types of DDoS attacks.

At the same time, reference [8] highlights the several vulnerabilities deliberately attempting to disrupt legit users' connections to TCP / IP application and transport layer services. The purpose of this paper is to propose a technique for the detection and analysis of synchronous and non-synchronous traffic flows by observing network in time. In addition, this technique uses legitimate and malicious traffic authentication from traffic sources using CAPTCHA in different ways. The proposal job will detect any unusual or faked IP addresses by using both synchronous and non-synchronous traffic flow reported information overtime slots. Besides, it marked address pairs authenticated using the CAPTCHA challenge-response mechanism while other packets are being dropped.

Reference [34] demonstrates the use of Attack Detection of various ML algorithms in SDN. Three classification algorithms (Neural Network, SVM, and Bayesian) have been tested. The controller maintains an Access Control List (ACL), which divides data flood into normal traffic or attacks it using the ML classification devices. Also, they found that SVM algorithms were better than other ML algorithms, as shown in figure (2.8) below. One such approach is to use algorithms for machine learning to identify relations as legitimate and illegitimate. In order to detect suspicious and harmful connections, we have two machine learning algorithms: the SVM and Neural Network (NN) classificatory.

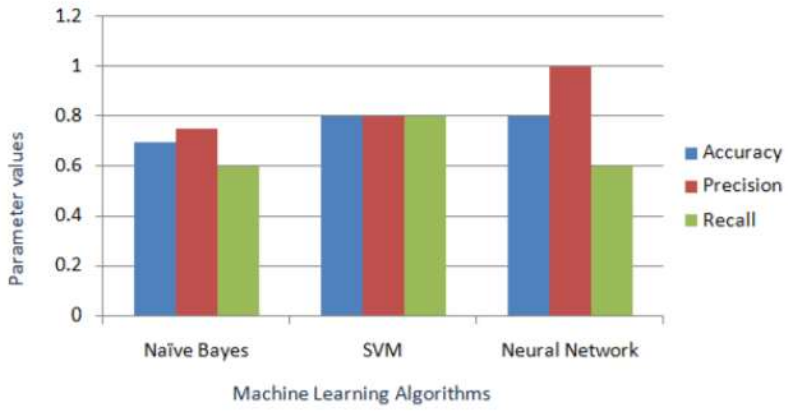

Figure 2.8: Comparison of all the parameters of the three ML algorithms.

3. METHODOLOGY AND PROPOSAL METHOD

This detection method is based on two ML algorithms, which are Entropy and PCA.

3.1 ENTROPY

Entropy or Shannon-Wiener ML is an algorithm technique with an essential concept as one of the ML-algorithms for computer science and information theory. Entropy is used to determine a random variable's randomness, the destination IP address for this scenario [32]. Entropy has a direct relationship with randomness. Whenever that is randomization, the value of entropy also increases. The entropy value is (1) when the traffic is equally spread among every destination in the network and (0) when the traffic requests the same destination. So, the value of entropy range (0,1).

Entropy detection is similar to the detection method in [35]. Later, entropy and PCA algorithms are combined to detect several types of DDoS attacks.

A fixed-size window is used to collect traffic packets and analyze entropy. The entropy computation is simplified with a fixed-size window. Because entropy accuracy may decrease in a fixed-time window via the low traffic loading period. Instead, the algorithm uses a window, which is measured by using (n) packet numbers and (n) by the window size. The packets are grouped into each window in accordance with their IP address. In each group, all packets have the same destination address, but several sources addresses may exist. The destination's IP address is used as the feature metrics, whereas the randomness is measured by the frequency of each specific destination address. Let n be the total number of packets in a window and the Entropy formula calculated as follow:

$$H = -\sum_{i=1}^{n} p_i \log p_i \tag{3.1}$$

Where P the probability of each element in the window (W).

$$W = \{(z_1, y_1), (z_2, y_2), (z, y_3) \ldots\} \tag{3.2}$$

Where W shows the window, and z_i Denotes the IP address of the destination, and y refers to the number of times. The study proposes a window size of 50 packets for gathering statistics.

To calculate the probability, we use:

$$P_i = \frac{z_i}{n} \tag{3.3}$$

Where Zi represents the total address requested over a specific time, we expect to be spread out the traffic all over a possible destination in normal traffic. Whereas in DDoS attacks, the traffic will focus on a specific destination IP address, or sometimes several destination IP addresses, which reduces the entropy value accordingly. When the entropy value drops, this indicates that a network attack may occur.

A fast detection method is essential in the SDN networks, and the attacks are detected in their first steps. SDN networks are more exposed than conventional networks to DDoS attacks. The window should not be too large to detect at early detection. In addition, a small window will add additional computation. A fifty-window size in our proposed method is used, as suggested by [4] in this thesis.

3.1.1 Entropy Destination IP Address Variance Implementation

The following algorithm flow chart highlights the measures related to the application of Entropy Variation of Destination IP address.

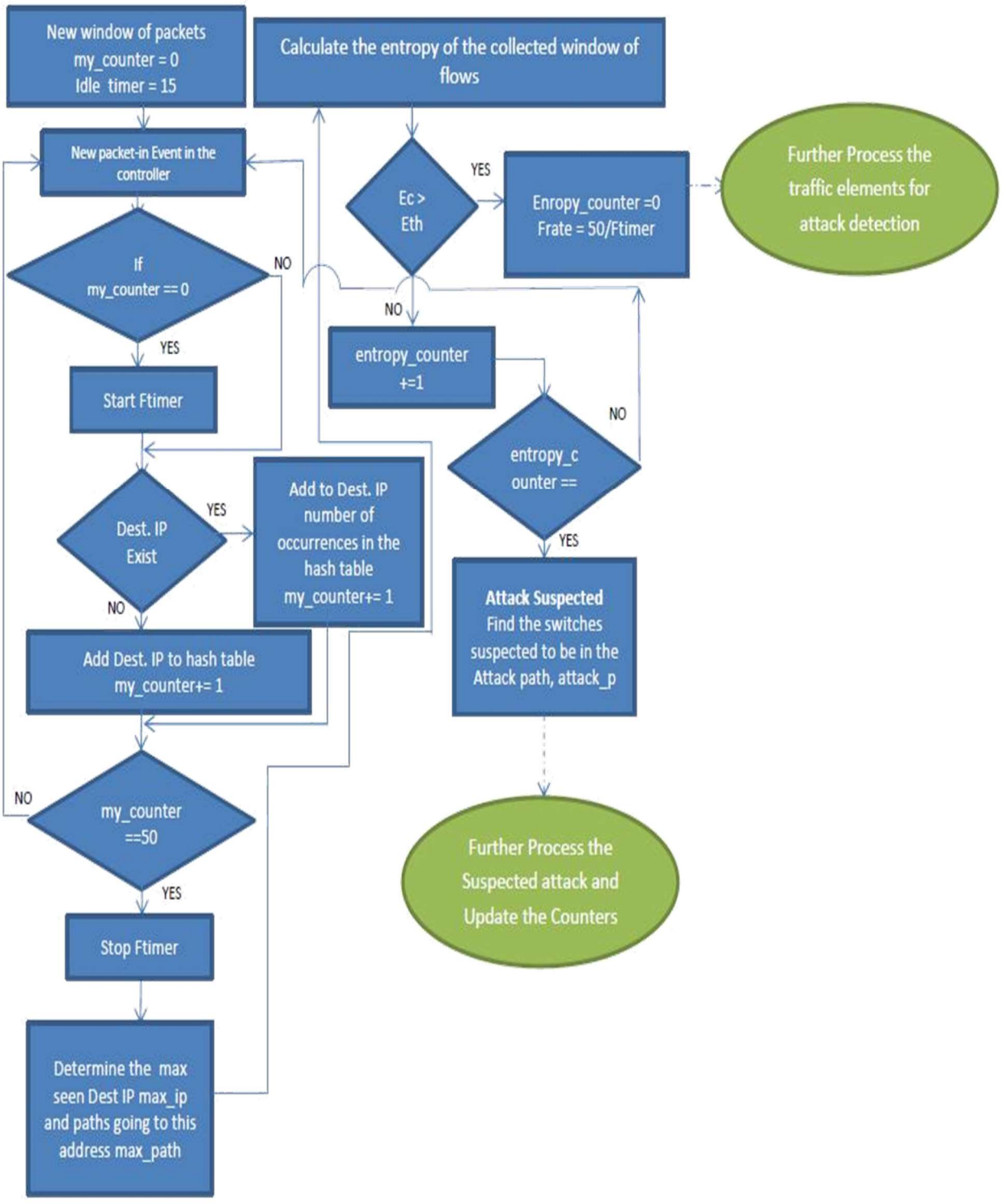

Figure 3.1: Entropy flow chart.

In the first step, the controller collects a window of 50 packages. These packets represent a 50-flow initiation request window that has been sent via switches to the controller. A timer calculates the time it takes to collect fifty packets from that window. This temporary timer is applied based on the flow rate in the next detection phase.

The controller calculates the shortest path for each flow, as indicated earlier, and installs the rule for the path flows. The controller does not maintain measured paths by default. A module to detect calculated paths (path-to-Stat) is inserted to identify the path for an attack in the next steps of the detection algorithm. Once a window of 50 collected packets is reached, the IP of the destination for a single packet is checked to see how often this IP that collected has been duplicated. A repeated destination IP address is validated as (max IP), and the following processing paths are stored as (max-path) [36].

The entropy function is then requested to calculate entropy. To calculate the entropy the function takings IP of destination addresses, how often repeats the frequently used source IP addresses, and calculates each destination's frequency. Current Entropy (Ec) is derived from the calculated Frequencies with eq (3.2). Whenever an attacker aims for a network host, there is a dramatic increase in the number of new flows for specific addresses. Thus, entropy will start declining [37][38]. The initial entropy threshold (Eth) in normal to low network traffic conditions is calculated at the beginning of the algorithm as the default entropy. The entropy calculated (Ec) is used in comparison with this threshold. If the calculated entropy is five consecutive times below the threshold, an attack is suspected.

The entropy threshold will be updated to the calculated entropy when no attack is detected to avoid further false alarm detections. This approach makes it possible to change the detection algorithm dynamically depends on the current traffic pattern. Furthermore, The Entropy threshold is restored to its default value if an attack is happened to increase detection sensitivity and consciousness. Since simple traffic patterns may shift entropy quickly, the detection algorithm is suspected of attacking only if the entropy is computed in five windows lower than the threshold.

Although entropy is a successful method of detection, it is not capable of detecting many scenarios. When the demand for a certain network destination such as the webserver or e-mail

server suddenly increases for legitimate traffic, the Entropy detection method continually reports false alarms of an ongoing attack. In addition, entropy may not show a significant decrease in the attacker distributed between many victims and result in a false negative report. The proposed detection method includes other detection algorithms combined in this project to overcome the above limitations of entropy detection.

3.2 PRINCIPAL COMPONENT ANALYSIS (PCA)

As mentioned in the previous chapter, entropy can detect a single victim of a DDoS attack. However, because of entropy limitations, particularly in multiple victims, It cannot be considered an effective independent DDoS Attack detection scheme [39][40]. Multiple DDoS attacks target various destinations simultaneously, which could lead to insignificant entropy changes, and we need another solution to detect this type of attack. Therefore, a combination of Entropy and PCA suggested producing a comprehensive detection method to detect the two types of DDoS attacks over SDN.

PCA is an ML algorithm method of coordination transformation that maps the measured data to a new axis for data reduction, which reduces data size. It is used for many computer science applications, such as image processing. It is also used to detect anomalies traffic in [21] as an effective way to separate network traffic into disjoint spaces corresponding to normal and anomalous network conditions, while [22] used PCA algorithm standalone method for detecting multiple DDoS attacks.

Since PCA records approximately 99 percent of all variability of the nominal traffic data; thus, by calculating the attribute value, the legal user traffic level must be similar to the principal component. For instance, suppose we have normal traffic coming from a source IP 128.238 with TTL value 56 want to reach a website such as Google.com. When this type of normal traffic rises, then the traffic rate at the google edge router increases simultaneously with the source IP prefix 128.238 and TTL value 56 and vice-versa. Thus, the IP prefix of the value attributes 128.238 and TTL value 56 are intrinsically dependent. The normal traffic measurement points in figure (3.2) are usually close to the extracted PC without a DDoS attack [39].

As DDoS attacking traffic with (130.30) prefix spoofing source IP with random TTL does not follow normal traffic correlation since DDoS generally sends low TTL traffic, existing traffic

metering points will differ from these main components just like illustrated in figure (3.2) with dark points. The greater the difference, the more aggressive the traffic.

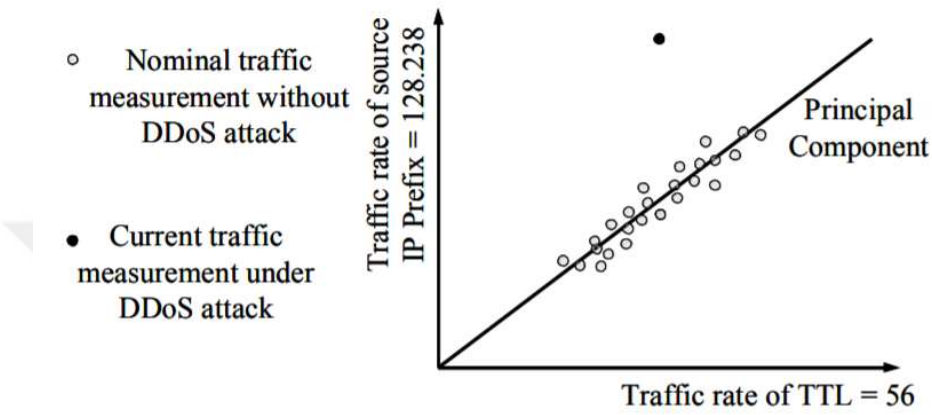

Figure 3.2: Correlation between multiple attributes values.

Since DDoS attackers try to start as much traffic as possible, the value of attributes shared with traffic attacks has increased significantly. In addition, the attacker usually does not know the normal traffic target profile to imitate the legitimacy of traffic that breaks normal traffic attribute values. Based on the above explanation, we used the PCA algorithm to isolate traffic from abnormal traffic [41].

We followed [19], a highly efficient method for detecting multiple DDoS attacks by using PCA. Detecting abnormalities can be found in decouples X to normal and anomalous components, which denote it as modeled ($\hat{x}$) and residual parts ($\tilde{x}$) of X as:

$$x = \hat{x} + \tilde{x} \tag{3.4}$$

The X PC is the linear combination of original variables and orthogonal to (X -1)th PC and has X^{th} maximum variance. Origin-Destination pair (OD Pair) represents a packet's origin node and destination node. (OD Flow p.) contains all traffic features for the OD pair. The numeral of sequential time intervals of interest t.

Let matrix (X) be the $t \times p$ where shows the times of all OD-flows, which denotes the time series of all the OD flows. Each column i denotes the time-series of $(i - \text{th})$ OD flow, and each row j represents an instance of all the OD flows at the time period j.

For matrix $X^T X$, which also calculates the covariance between flows as:

$$x^t x_{v_i} = \lambda_i v_i \tag{3.5}$$

Where $\{\lambda_i, i = 1, \cdots, \mathrm{p}\}$ are the eigenvalues corresponding to eigenvectors v_i. Since $X^T X$ is symmetrically positive definite, its orthogonal vectors, and the corresponding eigenvalues are not negative. As a result, the eigenvectors and the eigenvalues are set in order from large to small, as $\lambda_1 \geq \lambda_2 \geq \lambda_3 \ldots \geq \lambda_P$.

It can be shown to use the Rayleigh Quotient of $X^T X$ that corresponds to the maximum energy of the residual. We can write k - th principal components vk as:

$$vk = \arg\max_{\|v\|=1} \left\| x - \sum_{i=1}^{k-1} (X v_i v_i^T) v \right\| \tag{3.6}$$

Thus, Calculation of all PC $\{v_i\}_{i=1}^{P}$ is equivalent to eigenvectors of $X^T X$. For the examination of transformed data, the PC space can be used. The contribution of the principal axis i as a function of time is given by x_{v_i}, and can be normalized to unit length by dividing by $\sigma_i = \sqrt{\lambda_i}$. Thus, we have each principal axis i,

$$u_i = \frac{x_{v_i}}{\sigma_i}, \quad i = 1, \cdots, \mathrm{p} \tag{3.7}$$

They u_i are vectors of size t and orthogonal by construction. The equation above shows that all the OD pairs when weighed by v_iProduce one dimension of the transformed data. u_i captures the $i - \text{th}$ strongest temporal trend common the all OD pairs, and the set of $\{\mathrm{u_i}\}_{\mathrm{i=1}}^{\mathrm{P}}$ captures the time-varying trends common to the OD pairs, refer to them as the Eigen-flow of X. The set of principal components $\{v_i\}_{i=1}^{P}$ can be arranged in order as columns of a principal matrix V, which has size $P \times P$. Likewise, we can form the $t \times p$ matrix U in which column i is u_i, that V, U and σ_i can be arranged to write each OD flow X_i as:

$$\frac{x_i}{\sigma_i} = U(V^T)_{i,} \quad i = 1, \cdots, \mathrm{p} \tag{3.8}$$

To select a subspace that finds only r singular values are nonnegligible implies that X effectively resides on a dimensional subspace. Here, the original matrix is approximated as:

$$X' \approx \sum_{i=1}^{r} \sigma i u_i v_i^T \tag{3.9}$$

We can then write $\hat{x}$ and $\tilde{x}$ as:

$$\hat{x} = PP_x^T = Cx \; and \; \tilde{x} = (I - PP^T)x = \tilde{C}x \tag{3.9}$$

where the matrix **C = PPT** represents the linear operator that performs projection onto the normal subspace **S**, and $\tilde{\boldsymbol{C}}$ likewise projects onto the anomaly subspace $\tilde{\boldsymbol{S}}$.

A useful statistic for detecting abnormal changes in $\tilde{x}$ is the squared prediction error (SPE):

$$SPE \; = \; \|\tilde{x}\| \tag{3.10}$$

We may consider network traffic to be normal if SPE ≤ δ2 α where δ2 α denotes the threshold for the SPE at the 1 − α confidence level.

3.3 OUR PROPOSED METHOD (ENTROPY-PCA)

While entropy is powerful to detect single-victim attacks, and multi-victim attacks are detective by PCA. In order to detect different types of DDoS attack services, we found it necessary to mix between two algorithms.

According to [4], a small false alarm DDoS attack detection threshold is best detected (1), while [42] is also detected on a PCA DeltaY subspace threshold (1) with five occurrences. The network normally operates when the Entropy-PCA operation is above its limits. Any changes below any of these rates would shift the network service's status into a malicious attack detected by DDoS [40].

We combined the two algorithms to present a simple one technique to detect either type of DDoS attacks, as shown in the flow chart below:

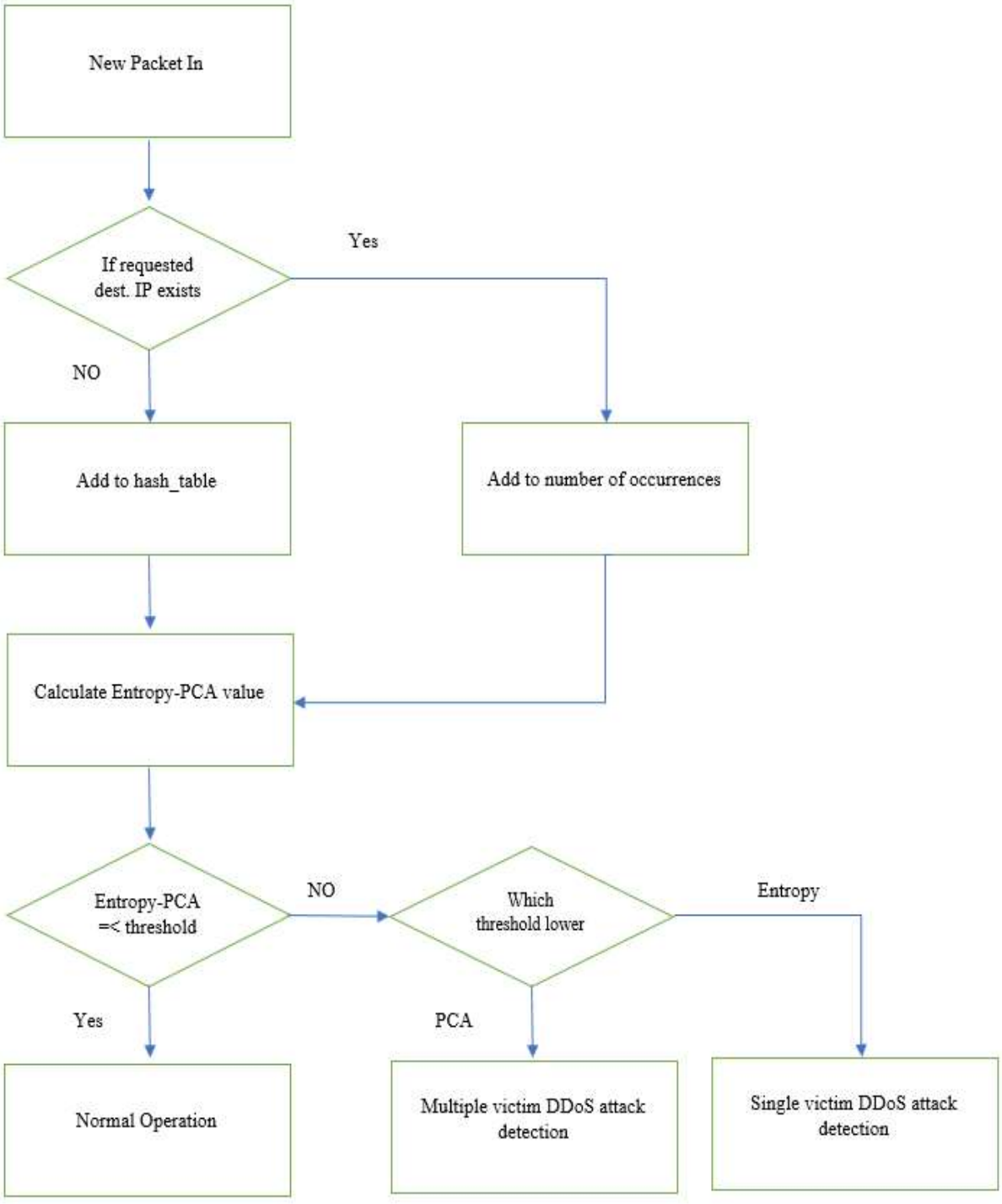

Figure 3.3: Entropy-PCA flow chart.

SDN switch sends a packet_in message to the controller when received a requested destination IP address that does not match any rule in the OpenFlow table and asks the controller for matching action. The controller looks up for the destination IP requested and determined. If the destination IP exists, it will add occurrence to this specific IP, while if there is no existing IP address, it will add to the hash index to address further requested to this IP.

A small, lightweight code adds to the controller, as previously mentioned, to calculate Entropy-PCA values and compared it with the pre-set threshold for both Entropy and PCA separately. While the Entropy-PCA value is above the threshold, the normal operation status is flagged, and the controller will send a packet_out message back to the switch with the path to the requested IP address. The SDN switch will receive this packet_out from the controller with instructed information to reach a specific IP address and save it on the OpenFlow table to deal with any repeated information. Nevertheless, a module is added to record the measured paths to detect the repeated IP destination and detect the attack's possibility on the controller.

So, if the Entropy-PCA value is lower than the threshold for one of them, an attack will be detected by which threshold is lower. If the entropy threshold is the lowest, this means that there is an attack on the network with one victim, but if the PCA threshold that has been violated means that the attack is multiple and involves more than one victim on the network, and mitigation measures should be taken in both cases.

3.4 ATTACK MITIGATION

Attack mitigation mechanisms are outside the scope of this thesis research, but we will briefly explain the most important methods. Many mechanisms are settling and mitigating attacks on the network after discovering the possibility of attacks. These techniques are used to mitigate attacks, including drop packet or block source IP address that appears to us in the attack detection report and for a specified period of time [43][44].

All of the techniques used to mitigate attacks are temporary techniques to buy more time to allow the network administrator to analyze the network in detail. However, these techniques may also sometimes affect the network's legitimate users, but they remain useful instead of forcing the controllers.

4. SIMULATION AND RESULTS

4.1 ENVIRONMENT

Simulation and test scenarios for the proposed Entropy-PCA approach described in this thesis to detect DDoS attacks were implemented on a personal computer Sony Vaio with ubuntu 18.04.4 LTS OS with a quad-core Intel Core i7-2.00 GHz processor and 8 G of memory.

The Suggestion method is implemented in the Mininet virtualized network environment using Python language over the pox controller and OpenFlow Switches. During the simulation, the Scapy scripts [45] tools generate legitimate traffics and attack network hosts. In contrast, Wireshark software was used to analyze the data traffic.

4.2 MININET

Mininet is a well-known Open Source license simulation tool for SDN. Mininet can communicate easily with the SDN network using the Mininet CLI, configure it, deploy it on real hardware, and build a customize prototype SDN topologies[46]. Mininet allows a set of configuration commands that could apply to any virtual link as a real world. It uses the POX controller as a default controller, OpenFlow switches, and Xterm as a terminal emulator to run commands. Mininet has several advantages, including the speed of operating a complete network within a few seconds, the ability to work with a wide range of topologies, supporting OpenFlow protocol [47]. The program can be run on a virtual machine for Windows or a real Linux environment, easy to deal with, and a mini edit tool that builds topologies with a graphical user interface and so on.

4.3 TRAFFIC DATA GENERATION

Scapy is a Python interpreter that allows for creating, forging, or decrypting network packets, collecting and analyzing packets, and dissecting packets [48]. Also, it permits to inject packets into the network. A wide variety of network protocols supports Scapy, and packets are managed and manipulated through Scapy. Scapy is also capable of sending, receiving, and sniffing packets.

In this thesis, Scapy programmed python over the PyCharm emulator to generate dummy traffic to train the data set first [49].

By default, Mininet assigned IP addresses automatically starting from (10.0.0.1) to host one and increasing for each new host. For instance, host 64 has assigned IP (10.0.0.64). The random function script (Randrange (1,256)) is used to generate random source IP addresses in the normal attack pattern.

The topology contains 64 hosts, and the random function source will generate normal traffic towards those 64 hosts (10.0.0.1) to (10.0.0.64). UDP packet is chosen as a packet type to perform the normal traffic with normal payload parameters with interval time set to (0.1) second.

All packets are forwarded to the target address, either carry out a single DDoS attack or multiple DDoS attacks, and the set script to call the script attack (for example, python attack.py x.x.x.x). We used the same mechanism for normal traffic to create the source IP address for attack traffic. The test environment's attack scenario has no payload and set up interval time for (0.025) second.

4.4 WIRESHARK

Wireshark is a widely used analyzer tool for a network protocol that can work with SDN and inspect to the microscopic level [50]. Wireshark brings many advantages such as Deep inspection, running on many platforms, real-time captured data traffic, analyzing packets, and exporting the output analyzed data to XML, CVS, plaint text … etc.

4.5 PERFORMANCE METRICS AND RESULTS

Our suggested detection method performance with Entropy-PCA ML is evaluated using the parameters, precision, error, and accuracy. To calculate these performance measurements, we use a confusion matrix.

4.5.1 Confusion Matrix

A confusion matrix is often used to describe the performance of a classification model with N × N, where N is the number of classes. We split the traffic into two classified groups for normal

and attack. The predicted class instances are represented in each raw line of the matrix, while each column represents the instances of an actual class, as shown in table (4.1).

Table 4.1: Confusion Matrix

		Predicated Class	
		Attack	Normal
Actual Class	Attack	TP	FN
	Normal	FP	TN

where,

TP = True Positive is the number of times the attack traffic was correctly classified.

FN = False Negative is the number of times attack traffic was classified as normal traffic.

TN = True Negative is the number of predictions that were correctly classified.

FP = False Positive is the number of times the normal traffic was classified as attack traffic.

The following performance measurements can be defined from the confusion matrix:

$$Accuracy\ Rate\ = \frac{Number\ of\ correct\ Predications}{Total\ Number\ of\ predications} \tag{4.1}$$

That is,

$$Accuracy\ Rate\ = \frac{TP\ +\ TN}{TP\ +\ FN\ +\ FP\ +\ TN} \tag{4.2}$$

To calculate the error prediction rate, we will follow the following formula:

$$Error\ Rate\ = \frac{Number\ of\ Wrong\ Predications}{Total\ number\ of\ Predications} \tag{4.3}$$

That is,

$$Error\ Rate\ = \frac{FP\ +\ FN}{TP\ +\ FN\ +\ FP\ +\ TN} \tag{4.4}$$

Now, we could calculate the Precision, which is the number of relevant items selected by:

$$Precision\ = \frac{TP}{TP\ +\ FP} \tag{4.5}$$

The results of our proposed method based on the FP, FN, and precision rate will be presented in the following sections.

4.6 SIMULATION SCENARIO AND RESULTS OF OUR PROPOSED METHOD

The experiment was done on a Sony laptop with the specifications that stated earlier. Using Mininet and two depth tree topology has 1 Pox controller, 9 OVS switches, and 64 hosts. All OVS switches operate as OpenFlow switches, as shown in figure (4.1) of our network topology.

Two modules are being added to calculate the Entropy and PCA values, while the Pox controller is modified by adding the two modules for both traffic dummy and attack types.

Normal traffic should be generated before the DDoS attack scenario is shown in order to obtain normal values for entropy and PCA. Normal traffic is generated from 2 hosts with a spoofed source IP address and a Scapy over python script with all specifications mentioned earlier in the traffic data generation section.

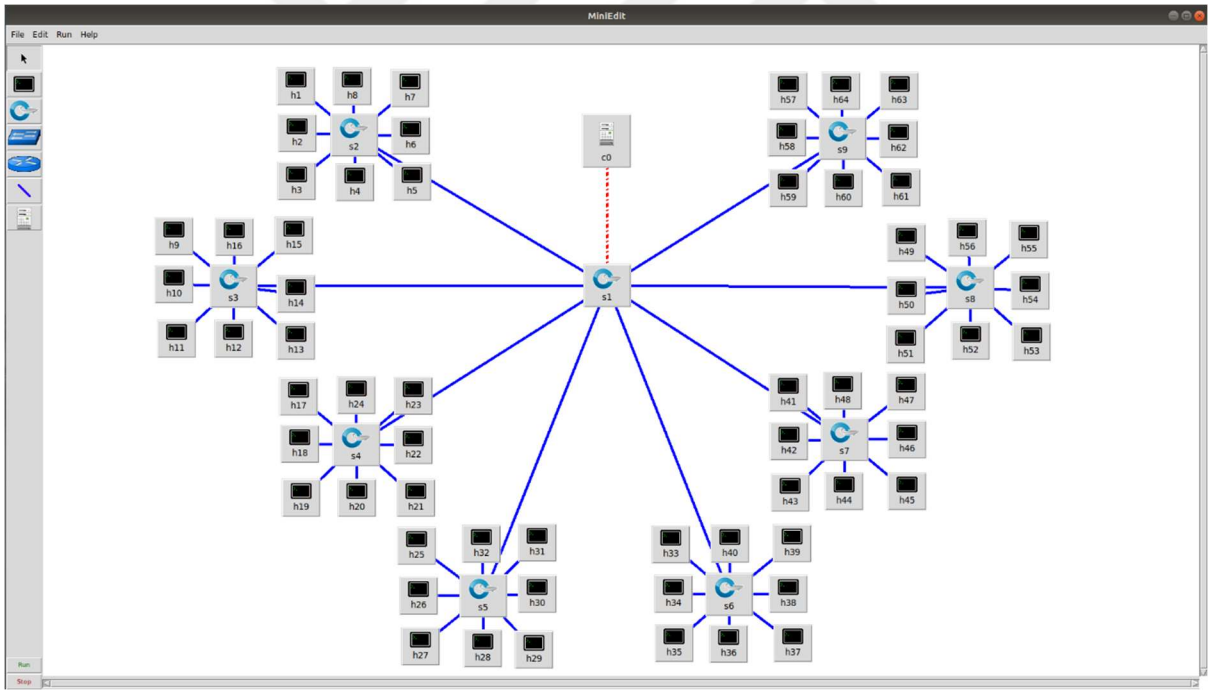

Figure 4.1: Network Topology.

4.6.1 Single DDoS Attack

As we mentioned earlier, DDoS could have two types of attacks. In this section, we start a single victim attack. The same network topology is used with 1 Pox controller, 9 OVS switches, and 64 hosts. We proposed hosts 10 and 20 generate the attack traffic towards host 64 using packet generation tools. Generate UDP traffic as a most common DDoS attack that does not need acknowledgment with no payload to generate packet as fast as 0.025 seconds for interval time.

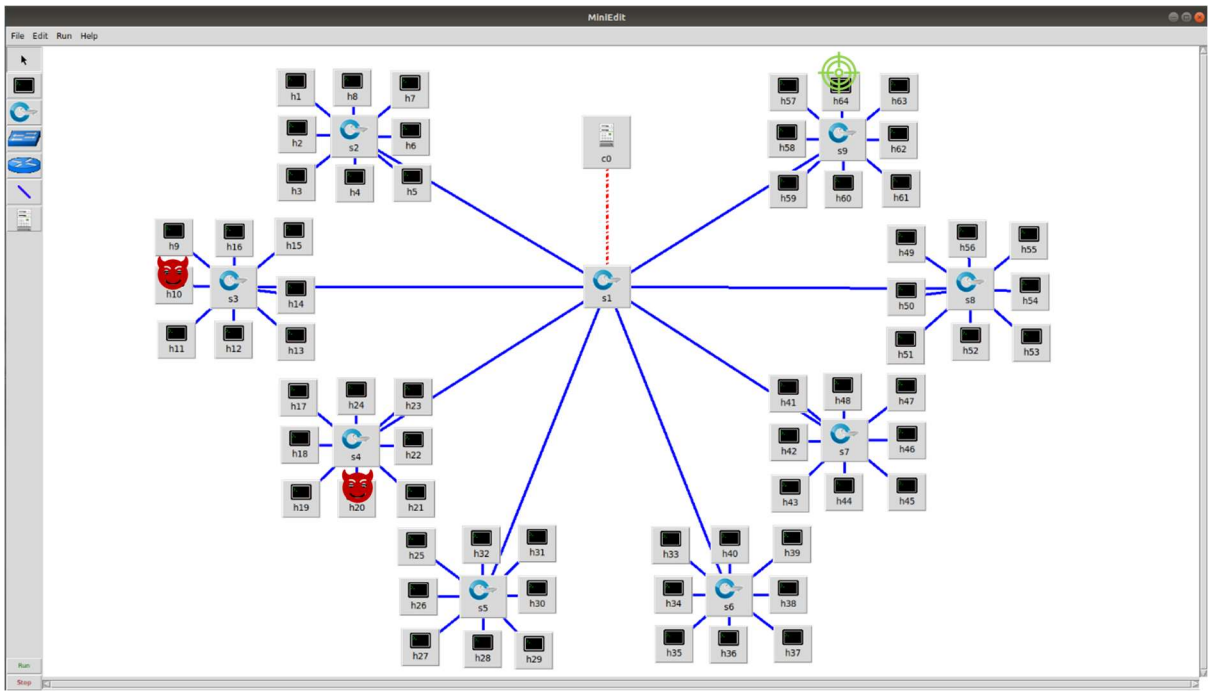

Figure 4.2: Single DDoS Attack Topology.

A simulation is repeated 50 times, each time for more than five minutes, with the two types of attacks are tested, and the details are noted.

First, normal traffic is generated to determine the thresholds of Entropy and PCA; then, after a while, attack traffic is generated from hosts 10 and 20 towards host 64, and the Entropy value is decreased less than the entropy threshold. An alarm of a possible incoming attack is flagged on the screen, and to prevent more damages, The Pox controller was suspended.

Legitimate traffic is defined as traffic with a normal interval time between each packet, a large number of packets. While attack traffic has a shorter time interval through a few bytes and without a payload.

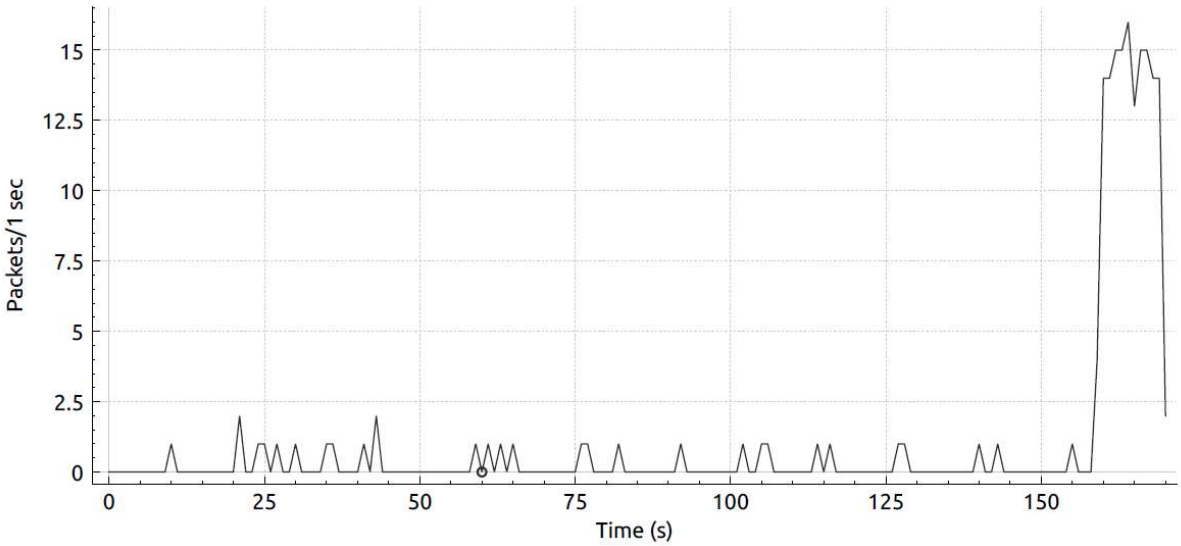

Figure 4.3: Wireshark Output: Attack Traffic on single DDoS Attack host.

As shown above in figure (4.3) from Wireshark that inspect specifically host 64. the normal traffic is generated from 0 to 158 seconds, and hosts 10 and 20 generate attack traffic towards host 64 at 158, and the system flags a possible detection of a single attack at 162.5 seconds and then a further suspend to shut down the controller. The delay in detection time is about 4.5 seconds, which is our 25 experiments' average time.

4.6.2 Multiple DDoS Attack

Multiple DDoS attacks are considering a hard type of attack that targets more than one victim. It is more complicated to distinguish them from normal network traffic because of the variety of targeting victims. PCA algorithm determined the DeltaY subspace threshold, and if this value has become lower than DeltaY, then a suspect of possible multiple attacks is undergoing.

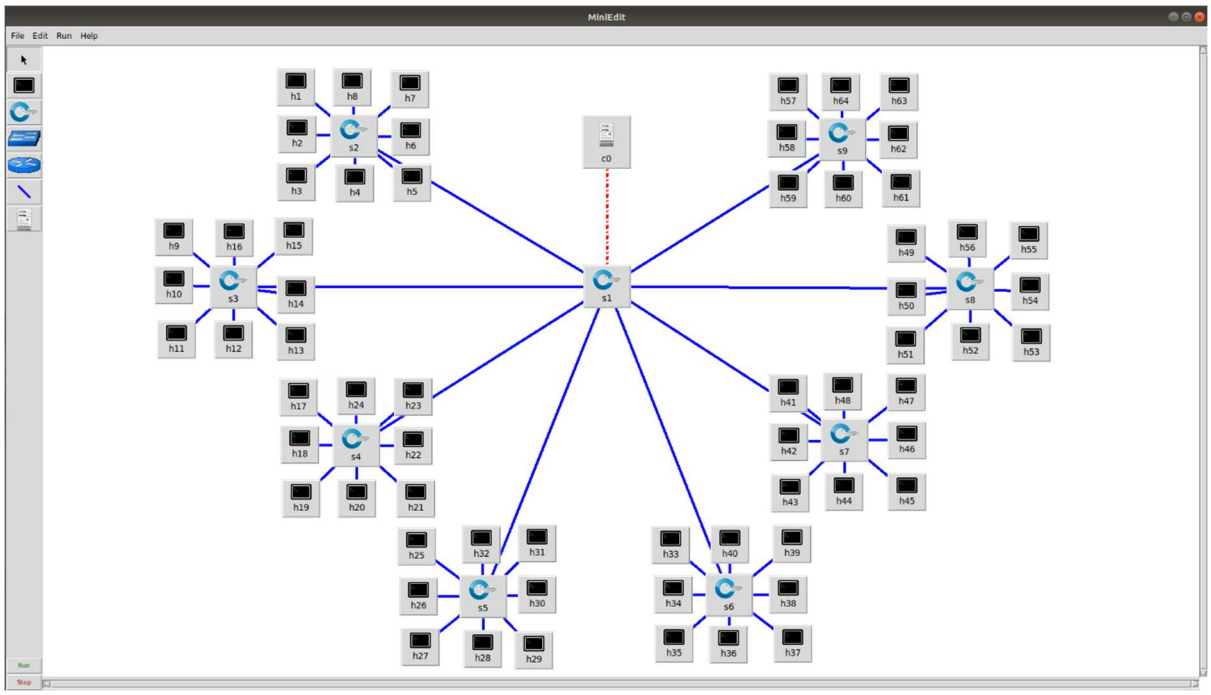

Figure 4.4: Multiple DDoS Attack Topology.

A simulation is started with normal traffic from hosts one and two to generate spoofed traffic to begin collecting and determining the two threshold values for about three minutes. Multiple DDoS attacks that start from hosts (10, 20, 30) and (40) generates towards host (50) and (64) simultaneously, as shown below in Figure (4.5) that inspect by Wireshark on the two hosts victim 64 and 50.

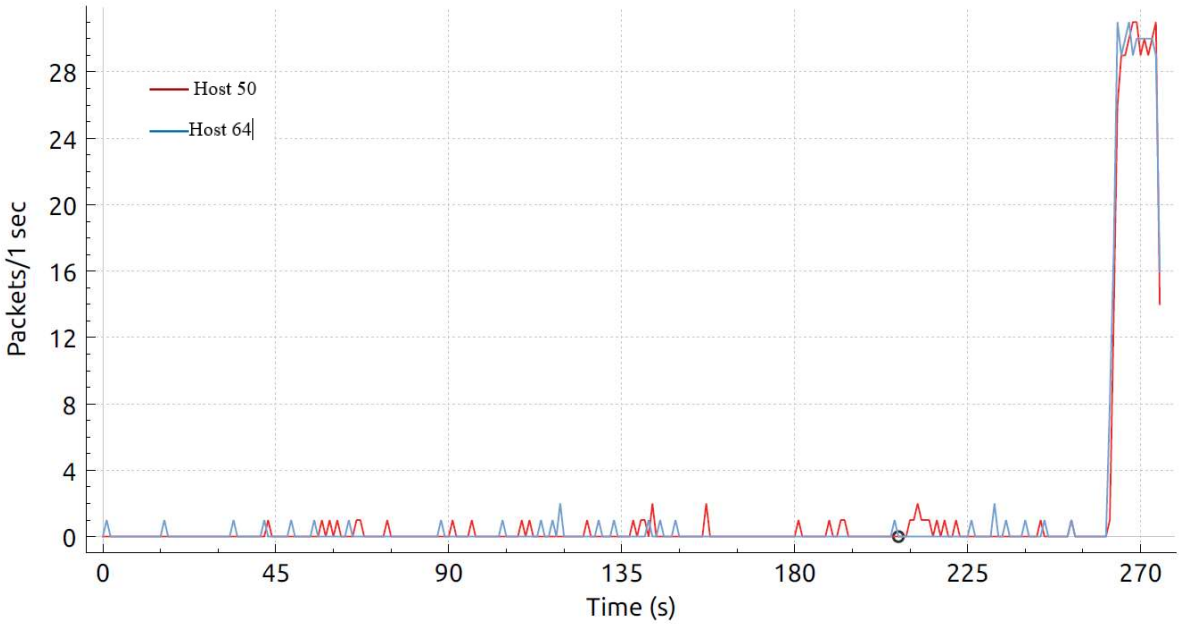

Figure 4.5: Wireshark Output: Attack Traffic on multiple DDoS Attack hosts.

The attacks start at 266 seconds towards the two hosts, and the system flags a possible detection of a single attack at 273 seconds after a violation of the DeltaY threshold. Further action of suspending, such as shutting down the controller, is taken to protect the network and save more time to analyze the traffic.

The delay of detecting multiple DDoS is reasonable longer than a single victim due to these attacks' nature. Multiple DDoS attacks try to hide behind the normal traffic, so in our simulation, the delay of multiple DDoS attacks is about (7) seconds.

4.6.3 Accuracy Rate of Entropy-PCA Method

In our total of fifty experiments, we are collecting all the packets from the Wireshark application. The data set consists of two subsets that are normal and DDoS data. The data are divided into the normal traffic dataset and the test traffic dataset, and the data sets are described in detail in the table (4.2)

As earlier demonstrated in section (4.2.1) of the confusion matrix that (TP) refers to our experiments to attacks traffic categorized as attack whereas legitimate traffic that we generated

on the equation represented as (TN). To get an Error rate or Accuracy rate, we should calculate (FP), which refers to legitimate traffic classified as attack traffic, while (FN) denotes as attack traffic deals as normal legitimate traffic.

We gathered a total of (1.414.671) packets through Wireshark through all our experiments. We analyzed and split them into legitimate and attack traffic.

Table 4.2: Total Dataset of our simulations

Type	**The number of**
TP	231.950
FP	3.181
FN	19.786
TN	1.159.754

Applying the Eq. (4.2) to the table above gives accuracy rate detection, and the result is 98.4% for a total of our experiment simulations for both types of attacks. Accordingly, applying equation (4.4) of Error detection Rate is 0.016, which is highly good results considering possibilities to detect both types of attack. It is important to refer here that our method detects as a single attack in some simulation of multiple attacks because the threshold of entropy was violated first before the PCA threshold (DeltaY).

Finally, we find that our dataset's Precision value is 0.97 when applying equation (4.5) on our dataset gathered from the Wireshark.

Table 4.3: Performance Metrics

Performance Metrics	
Accuracy	98.4%
Error Rate	1.6%
Precision	97%

4.7 COMPARISON WITH A RELATED PREVIOUS WORK

This section will compare the performance of our proposed machine learning-based DDoS Detection algorithm previous work such as that presented in [31], and [51]. Figure (4.6) shows a comparison of the Accuracy Rates percentage achieved by the Entropy-PCA method and other methods with a note to be taken into consideration, which in our proposed method, the DDoS attacks include two types of DDoS attacks, while the rest of the methods include attacks on one victim.

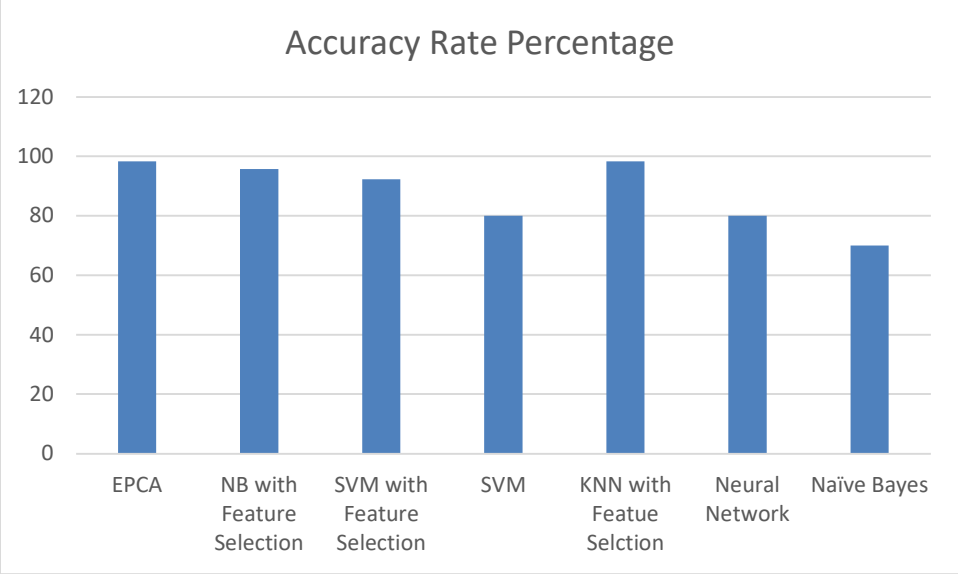

Figure 4.6: Comparison of the accuracy rate of detection methods.

In our Entropy-PCA method, we have achieved a precision rate (98.4%) while NB with a Feature Selection that has six selected features with an embedded filter wrapped (95.7%), SVM has ten selected features (92.28)% whereas the only SVM has (80%) whereas KNN with a Feature

Selection that has six selected features (98.3%) which is slightly lower by 0.1 than our method. The Neural Network and the Naïve Bayes have achieved (80%) and (70%), respectively.

5. CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

A novel, lightweight approach to detect DDoS attacks on SDN architecture is proposed in this thesis. To reveal and detect the DDoS single attack and Multiple attacks, we combined two ML algorithms and attacked them to the Pox controller. We experimented with our solution with two kinds of attacks. The experiment results indicate that our approach is extraordinarily successful with a high degree of precision and a lower error rate.

5.2 FUTURE WORK

As a single threshold may increase FP and FN reports, we need to develop a system to use a dynamic threshold with a range of thresholds that change dynamically depending on the traffic flow's matching role. As a part of detection, we need to move to stage two and find a mitigation technique. Although our work is done through a simple virtual environment, we need to test our work on real equipment and networks.

REFERENCES

[1] F. Group, "Network Innovation through OpenFlow and SDN," *Netw. Innov. through OpenFlow SDN*, 2014.

[2] R. Klöti, V. Kotronis, and P. Smith, "OpenFlow: A security analysis," *Proc. - Int. Conf. Netw. Protoc. ICNP*, 2013.

[3] Z. Hu, M. Wang, X. Yan, Y. Yin, and Z. Luo, "A comprehensive security architecture for SDN," *2015 18th Int. Conf. Intell. Next Gener. Networks, ICIN 2015*, pp. 30–37, 2015.

[4] S. M. Mousavi and M. St-Hilaire, "Early Detection of DDoS Attacks Against Software Defined Network Controllers," *J. Netw. Syst. Manag.*, vol. 26, no. 3, pp. 573–591, 2018.

[5] S. Y. Mehr and B. Ramamurthy, "An SVM based DDoS attack detection method for Ryu SDN controller," *Conex. 2019 Companion - Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol. Part Conex. 2019*, pp. 72–73, 2019.

[6] A. R. Abdou, P. C. Van Oorschot, and T. Wan, "Comparative analysis of control plane security of SDN and conventional networks," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 4, pp. 3542–3559, 2018.

[7] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," *HotSDN 2013 - Proc. 2013 ACM SIGCOMM Work. Hot Top. Softw. Defin. Netw.*, pp. 165–166, 2013.

[8] N. Patani and R. Patel, "A Mechanism for Prevention of Flooding based DDoS Attack," *Int. J. Comput. Intell. Res.*, vol. 13, no. 1, pp. 101–111, 2017.

[9] Z. Yao and Z. Yan, "Security in software-defined-networking: A survey," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10066 LNCS, no. 4, pp. 319–322, 2016.

[10] K. Kalkan, G. Gur, and F. Alagoz, "SDNScore: A statistical defense mechanism against DDoS attacks in SDN environment," *Proc. - IEEE Symp. Comput. Commun.*, no. September, pp. 669–675, 2017.

[11] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDOS) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 602–622, 2016.

[12] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Y. Yau, and J. Wu, "Real-time DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 7, pp. 1838–1853, 2018.

[13] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS '08*, 2008, pp. 1–9.

[14] C. Dillon and M. Berkelaar, "OpenFlow (D)DoS Mitigation," 2014.

[15] R. Kandoi and M. Antikainen, "Denial-of-service attacks in OpenFlow SDN networks," *Proc. 2015 IFIP/IEEE Int. Symp. Integr. Netw. Manag. IM 2015*, pp. 1322–1326, 2015.

[16] M. Nugraha, I. Paramita, A. Musa, D. Choi, and B. Cho, "Utilizing OpenFlow and sFlow to Detect and Mitigate SYN Flooding Attack," *J. Korea Multimed. Soc.*, vol. 17, no. 8, pp. 988–994, 2014.

[17] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "DDoS attack protection in the era of cloud computing and Software-Defined Networking," *Comput. Networks*, vol. 81, pp. 308–319, 2015.

[18] S. Shin, L. Xu, S. Hong, and G. Gu, "Enhancing Network Security through Software Defined Networking (SDN)," in *2016 25th International Conference on Computer Communications and Networks, ICCCN 2016*, 2016.

[19] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDOS) flooding attacks," *IEEE Commun. Surv. Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[20] K. Raghunath and P. Krishnan, "Towards A Secure SDN Architecture," *2018 9th Int.*

Conf. Comput. Commun. Netw. Technol. ICCCNT 2018, pp. 1–7, 2018.

[21] N. N. Dao, J. Park, M. Park, and S. Cho, "A feasible method to combat against DDoS attack in SDN network," *Int. Conf. Inf. Netw.*, vol. 2015-Janua, pp. 309–311, 2015.

[22] P. Khuphiran, P. Leelaprute, P. Uthayopas, K. Ichikawa, and W. Watanakeesuntorn, "Performance comparison of machine learning models for DDoS attacks detection," *2018 22nd Int. Comput. Sci. Eng. Conf. ICSEC 2018*, pp. 1–4, 2018.

[23] Q. Niyaz, W. Sun, and A. Y. Javaid, "A Deep Learning Based DDoS Detection System in Software-Defined Networking (SDN)," *ICST Trans. Secur. Saf.*, vol. 4, no. 12, p. 153515, 2017.

[24] F. Bensalah, N. El Kamoun, and M. A. El Houssaini, "Inline detection of denial of service attacks in software defined networking using the hotelling chart," *Procedia Comput. Sci.*, vol. 160, pp. 785–790, 2019.

[25] P. Xiao, Z. Li, H. Qi, W. Qu, and H. Yu, "An efficient DDoS detection with bloom filter in SDN," *Proc. - 15th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. 10th IEEE Int. Conf. Big Data Sci. Eng. 14th IEEE Int. Symp. Parallel Distrib. Proce*, pp. 1–6, 2016.

[26] Z. Chen, F. Jiang, Y. Cheng, X. Gu, W. Liu, and J. Peng, "XGBoost Classifier for DDoS Attack Detection and Analysis in SDN-Based Cloud," *Proc. - 2018 IEEE Int. Conf. Big Data Smart Comput. BigComp 2018*, pp. 251–256, 2018.

[27] A. A. Bahashwan, M. Anbar, and S. M. Hanshi, *Overview of IPv6 Based DDoS and DoS Attacks Detection Mechanisms*, vol. 1132 CCIS. 2020.

[28] K. Kalkan, L. Altay, G. Gür, and F. Alagöz, "JESS: Joint Entropy-Based DDoS Defense Scheme in SDN," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2358–2372, 2018.

[29] N. Z. Bawany, J. A. Shamsi, and K. Salah, "DDoS Attack Detection and Mitigation Using SDN: Methods, Practices, and Solutions," *Arab. J. Sci. Eng.*, vol. 42, no. 2, pp. 425–441, 2017.

[30] N. Ahuja and G. Singal, "DDOS Attack Detection Prevention in SDN using OpenFlow

Statistics," *Proc. 2019 IEEE 9th Int. Conf. Adv. Comput. IACC 2019*, pp. 147–152, 2019.

[31] H. Polat, O. Polat, and A. Cetin, "Detecting DDoS attacks in software-defined networks through feature selection methods and machine learning models," *Sustain.*, vol. 12, no. 3, 2020.

[32] D. Hu, P. Hong, and Y. Chen, "FADM: DDoS Flooding Attack Detection and Mitigation System in Software-Defined Networking," *2017 IEEE Glob. Commun. Conf. GLOBECOM 2017 - Proc.*, vol. 2018-Janua, pp. 1–7, 2017.

[33] P. Kumar, M. Tripathi, A. Nehra, M. Conti, and C. Lal, "SAFETY: Early Detection and Mitigation of TCP SYN Flood Utilizing Entropy in SDN," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, no. 4, pp. 1545–1559, 2018.

[34] N. Meti, D. G. Narayan, and V. P. Baligar, "Detection of distributed denial of service attacks using machine learning algorithms in software defined networks," *2017 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2017*, vol. 2017-Janua, pp. 1366–1371, 2017.

[35] A. Ahalawat, S. S. Dash, A. Panda, and K. S. Babu, "Entropy Based DDoS Detection and Mitigation in OpenFlow Enabled SDN," *Proc. - Int. Conf. Vis. Towar. Emerg. Trends Commun. Networking, ViTECoN 2019*, pp. 1–5, 2019.

[36] P. Kamboj, M. C. Trivedi, V. K. Yadav, and V. K. Singh, "Detection techniques of DDoS attacks: A survey," *2017 4th IEEE Uttar Pradesh Sect. Int. Conf. Electr. Comput. Electron. UPCON 2017*, vol. 2018-Janua, pp. 675–679, 2017.

[37] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed DDoS detection mechanism in software-defined networking," *Proc. - 14th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. Trust. 2015*, vol. 1, pp. 310–317, 2015.

[38] H. Yao, Y. Liu, and C. Fang, "An abnormal network traffic detection algorithm based on big data analysis," *Int. J. Comput. Commun. Control*, vol. 11, no. 4, pp. 567–579, 2016.

[39] H. Sun, Y. Zhaung, and H. J. Chao, "A principal components analysis-based robust DDoS defense system," *IEEE Int. Conf. Commun.*, pp. 1663–1669, 2008.

[40] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft, "Structural analysis of network traffic flows," *Perform. Eval. Rev.*, vol. 32, no. 1, pp. 61–72, 2004.

[41] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," *Comput. Commun. Rev.*, vol. 35, no. 4, pp. 217–228, 2005.

[42] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," p. 219, 2004.

[43] S. Saharan and V. Gupta, "Prevention and Mitigation of DNS based DDoS attacks in SDN Environment," *2019 11th Int. Conf. Commun. Syst. Networks, COMSNETS 2019*, vol. 2061, pp. 571–573, 2019.

[44] M. Parashar, A. Poonia, and K. Satish, "A Survey of Attacks and their Mitigations in Software Defined Networks," *2019 10th Int. Conf. Comput. Commun. Netw. Technol. ICCCNT 2019*, pp. 1–8, 2019.

[45] P. Biondi, "Scapy documentation (!)."

[46] "Mininet Overview - Mininet." [Online]. Available: http://mininet.org/overview/. [Accessed: 01-Nov-2020].

[47] "XTERM – Terminal emulator for the X Window System." [Online]. Available: https://invisible-island.net/xterm/. [Accessed: 01-Nov-2020].

[48] "Welcome to Scapy's documentation! — Scapy 2.4.4. documentation." [Online]. Available: https://scapy.readthedocs.io/en/latest/. [Accessed: 01-Nov-2020].

[49] "PyCharm: the Python IDE for Professional Developers by JetBrains." [Online]. Available: https://www.jetbrains.com/pycharm/. [Accessed: 01-Nov-2020].

[50] "Wireshark · Go Deep." [Online]. Available: https://www.wireshark.org/. [Accessed: 01-Nov-2020].

[51] R. T. Kokila, S. Thamarai Selvi, and K. Govindarajan, "DDoS detection and analysis in SDN-based environment using support vector machine classifier," *6th Int. Conf. Adv.*

Comput. ICoAC 2014, pp. 205–210, 2015.

APPENDIX A

[CODES]

We will paste here the most relevant codes that we used in our simulation.

A.1 NORMAL TRAFFIC

```
from scapy.all import sendp, IP, UDP, Ether, TCP
from random import randrange

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
```

Figure A.1: Generate Source IP.

```
def generateDestinationIP(start, end):
    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
```

Figure A.2: Generate Random Destination 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)

if __name__ == '__main__':
  main(sys.argv)
```

Figure A.3: Generate Normal Traffic.

A.2 ATTACK TRAFFIC

```
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)

if __name__=="__main__":
  main()
```

Figure A.4: Lunch an Attack Towards a Specific Host.

A.3 ENTROPY VALUE

```
import math
from pox.core import core

log = core.getLogger()

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

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

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

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

          self.destEntropy.append(sum(entropyList))

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

      def __init__(self):
      pass
```

Figure A.5: Calculate Entropy Value.

A.4 PCA VALUE

```
intercept = 0
slope = 0

sqDistSum = 0
sdDeviation = 0
meanYDist = 0
yDist =0

rmsSqSum = 0
rms = 0

def calcMean(self,s,d):
    if(self.count!=0):
       self.meanSrc=(self.meanSrc*(self.count-1)+s)/self.count
       self.meanDst=(self.meanDst*(self.count-1)+d)/self.count

def calcSqDeviation(self) :

    if(self.count !=0 ) :
  self.meanYDist=(self.meanYDist*(self.count-1)+self.yDist)/self.count

self.sqDistSum += pow((self.yDist-self.meanYDist),2)
if(self.count-1 != 0):
  self.sdDeviation=math.sqrt(self.sqDistSum/(self.count-1))

def calcRms(self):
    self.rmsSqSum += pow(self.yDist,2)
    self.rms =math.sqrt(self.rmsSqSum/self.count)

def calcYDistance(self,srcIpNum,dstIpNum) :
    temp = self.slope*srcIpNum + self.intercept
    self.yDist = (dstIpNum-temp)
```

Figure A.6: Calculate initial PCA Value.

```
    def getsdDeviation(self) :
        self.calcSqDeviation()
        return self.sdDeviation

    def getYDist(self):
        return self.yDist

    def getRms(self):
        self.calcRms()
        return self.rms

    def collectStats(self, srcIp, dstIp):
        self.count +=1
        if(self.srcDict.has_key(str(srcIp))==1):
          srcIpNum=self.srcDict[str(srcIp)]
        else :
          self.srcDict[str(srcIp)]=len(self.srcDict) + 1
          srcIpNum=self.srcDict[str(srcIp)]
        dstIpNum=ipToNum(dstIp)

    self.srcIpList.append(srcIpNum)
    self.dstIpList.append(dstIpNum)
    d={'x' : self.srcIpList,'y' : self.dstIpList}
    data=pd.DataFrame(data=d)

        lm = smf.ols(formula = 'y ~ x', data = data).fit()

    self.intercept= lm.params.Intercept
    self.slope= lm.params.x

    self.calcYDistance(srcIpNum,dstIpNum)

    def __init__(self):
    pass
```

Figure A.7: Calculate PCA Value.

A.5 POX CONTROLLER MODIFICATION

It is a POX controller with a modification script to detect DDoS attacks through our method Entropy-PCA. First, we call some important functions and reset values as an initial code.

```
import os
import datetime
import pox
import itertools
import time
import pox.openflow.libopenflow_01 as of
from pox.core import core
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
from pox.lib.revent import *

from .detectionUsingPCA import PCA
from .detectionUsingEntropy import Entropy
import time

# reset counter entropy
diction = {}
ent_obj = Entropy()
set_Timer = False
defendDDOS=False
#
pca_obj = PCA()

initialCount = 0
```

Figure A.8: Import Function and Reset Values.

```
    global ddosPCACount
    global initialCount
    global ddosStart
    global startPCA
    global endPCA
###################### Entropy detection
    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
```

Figure A.9: Import Calculated Entropy Value.

```
    def _timer_func ():
      global diction
      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                              Single DDOS DETECTED                                  \n"
              print "\n",str(diction)
              print "\n",datetime.datetime.now(),": BLOCKED PORT NUMBER  : ", str(i), " OF SWITCH ID: ", str(k)
              print "\n_____________________________________________________________________________"

              dpid = k
              msg = of.ofp_packet_out(in_port=i)
              core.openflow.sendToDPID(dpid,msg)
      diction={}

#diction mean a count of how many times the value is under the threshold
 #k=switch ID number
 #i-POrt ID
```

Figure A.10: Detect Single DDoS attack.

```
################## Entrtop Works when value less than 1
      if ent_obj.value <1:
        preventing()
        if timerSet is not True:
         Timer(1, _timer_func, recurring=True)

      else:
        timerSet=False


############## PCA works now
      initialCount = initialCount + 1
      if(initialCount>5):
        if(-1 < pca_obj.getYDist() < 1 and ddosStart ==False) :
            ddosStart=True
            ddosPCACount = 0
            startPCA =time.time()
        elif(-5 < pca_obj.getYDist() < 5 and ddosStart ==True):
            endPCA=time.time()
            ddosPCACount = ddosPCACount +1
            if(ddosPCACount > 8 and (endPCA - startPCA)<2 ):
                    print "\n_____________________________________________________________________________________________"
                    print "\n                                Mutiple DDOS DETECTED                                    \n"
                    print "\n",datetime.datetime.now(),": BLOCKED PORT NUMBER  : ", event.connection.dpid , " OF SWITCH ID: ", event.port
                    print "\n_____________________________________________________________________________________________"

        else :
            ddosStart=False
            ddosPCACount = 0


      self._send_lost_buffers(dpid, packet.next.srcip, packet.src, inport)
```

Figure A.11: Detect Multiple DDoS attacks.