An Entropy-Based Methodology for Detecting and Mitigating DDoS Attacks in SDN Environments to Improve Control Plane Security

Wendwossen Desalegn Gebru

A Thesis Submitted to the Department of Electronics and Communication Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Electronics and Communication Engineering

Office of Graduate Studies

Adama Science and Technology University

May, 2022

Adama, Ethiopia

An Entropy-Based Methodology for Detecting and Mitigating DDoS Attacks in SDN Environments to Improve Control Plane Security

Wendwossen Desalegn Gebru

Advisors:

Major Advisor: **Dr.Javed Shaikh**

Co-advisor: **Mr.Bayissa Taye**

A Thesis Submitted to the Department of Electronics and Communication Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Electronics and Communication Engineering

Office of Graduate Studies

Adama Science and Technology University

May, 2022

Adama, Ethiopia

Declaration

I hereby declare that this Master Thesis entitled "**An Entropy-Based Methodology for Detecting and Mitigating DDoS Attacks in SDN Environments to Improve Control Plane Security**'' is my original work. That is, it has not been submitted for the award of any academic degree, diploma or certificate in any other university. All sources of materials that are used for this thesis have been duly acknowledged through citation.

Name of student	Signature	Date

Recommendation

We, the advisor(s) of this thesis, hereby certify that we have read the revised version of the thesis entitled "**An Entropy-Based Methodology for Detecting and Mitigating DDoS Attacks in SDN Environments to Improve Control Plane Security**'' prepared under our guidance by **Wendwossen Desalegn Gebru** submitted in partial fulfillment of the requirements for the degree of Mater's of Science in Electronics and Communication Engineering. Therefore, We recommend the submission of revised version of the thesis to the department following the applicable procedures.

Major Advisor	Signature	Date
Co-advisor	Signature	Date

Approval Page

We, the advisors of the thesis entitled "**An Entropy-Based Methodology for Detecting and Mitigating DDoS Attacks in SDN Environments to Improve Control Plane Security**'' and developed by **Wendwossen Desalegn Gebru**, hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Major Advisor	Signature	Date
Co-advisor	Signature	Date

Approval of Board of Examiners

We, the undersigned, members of the Board of Examiners of the thesis by **Wendwossen Desalegn Gebru** have read and evaluated the thesis entitled "**An Entropy-Based Methodology for Detecting and Mitigating DDoS Attacks in SDN Environments to Improve Control Plane Security**" and examined the candidate during open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master of Science in Electronics and Communication Engineering.

Chairperson	Signature	Date
Internal Examiner	Signature	Date
External Examiner	Signature	Date

Finally, approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the candidate's Department Graduate Council (DGC) and School Graduate Committee (SGC).

Department Head	Signature	Date
School Dean	Signature	Date
Office of Postgraduate Studies, Dean	Signature	Date

ACKNOWLEDGEMENT

First and foremost, I'd want to thank **God** for his unprecedented blessings and for always being there for me. You have given me the confidence and bravery to believe in myself and improve my skills. I couldn't have done this without your faith in me. My name may be on the cover, but I cannot take credit for all that happened to get this thesis from idea to printing. As a result, I must express my gratitude to the persons mentioned elsewhere here.

I'd want to convey my heartfelt appreciation and gratitude to my advisers, **Dr. Javed Shaikh** and **Mr. Bayissa Taye**. In the department of Electronics and Communication Engineering, School of Electrical Engineering and Computing, Adama Science and Technology University for their unwavering support, mentoring, remark, and recommendations from the beginning to the finish of the thesis study.

I would also want to thank **Kaleab Kassahun,** Software Developer and Network Security Engineer, for his assistance and review contribution to the success of my thesis. His boundless energy, ingenuity, and exceptional abilities have always served as a steady source of inspiration for me. His connection with originalism inspired and fostered my intellectual growth, which will benefit me now and future. I am glad to say that I got the opportunity to work with such an experienced expert like him. He is a wonderful person and one of the greatest trainers I have ever had; I will be eternally grateful to him.

Finally, I'd want to thank my dear friends for their encouragement and support during my job, as well as their important contribution, unending remark, understanding, support, and encouragement in life conditions.

CONTENTS

CHAPTER PAGE

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
CPU	Central Processing Unit
CRT	Commercial Remote Terminal
DDoS	Distributed Denial of Service
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
LTS	Long Term Service
NFV	Network Functions virtualization
NOS	Network Operating System
NVMe	Non-Volatile Memory Express
ONF	Open Networking Foundation
OSI	Open Systems Interconnection
OVS	Open Virtual Switch
PoC	Proof of Concept
SDN	Software Defined Networking
SSD	Solid State Drive
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VM	Virtual Machine
WAN	Wide Area Network

ABSTRACT

As we all know, the architectural framework of software-defined networking (SDN) reduces network managers' work by separating the data plane from the control plane. This makes network deployment easier by providing a programmable interface for application development in areas such as security management, and the centralized logical controller provides greater control over the entire network, which has complete network visibility. This study was done to design a mechanism for implementing a security solution for detecting and mitigating distributed denial of service (DDoS) on the SDN control plane. The proposed approach will be based on an early detection strategy that is aligned with the standard and used by professionals in the field as a guide for implementing such security solutions. This thesis describes an approach that was used to identify and mitigate the risks associated with the OpenFlow protocol and its POX controller. The methodology is validated by performing activities in a controlled simulation scenario using the Mininet tool and the SDN controller. The detection algorithm's results were observed through simulation, then implemented into a network testbed, and the proposed algorithm's results are presented and analyzed. As a result of the methodology's successful use, the problem has been solved by implementing a DDoS attack detection mechanism based on the network's entropy calculation and mitigation by blocking the switch port from which the malicious traffic is generated. The solution is quick and effective against various types of DDoS attacks, including TCP, UDP, and ICMP floods.

Keywords:

DDoS Detection,Entropy Method,ICMP,Mitigation,POX ,SDN

CHAPTER ONE

1. INTRODUCTION

1.1 Background of the Study

Network security innovation is a race between adversaries and security communities to see who can break and secure the network first. Security researchers and practitioners have worked hard and made significant progress in overcoming the threats posed by their adversaries. However, the fast expansion of information and communication technologies, such as mobile devices and cloud computing virtualization, imposes a new network security burden on network administrators. New vulnerabilities, attack mechanisms, and attack channels develop as technology improves. Because of increased industry demand and the rise of cloud services, the complex traditional model, which is so dependent on each provider and has inconsistent policies, has become unscalable and unadopted. As a result, network solution researchers developed a new network technology known as software defined networking (SDN).

SDN is a new network technology that provides a more flexible and scalable network architecture capable of responding quickly to changes in business and end-user needs through simplified network management (Mladenov, 2019). SDN's power has been demonstrated day by day, extending from small local area networks to public cloud architectures. In most cases, SDN demonstrates its great success by providing greater reliability, effectiveness, simplicity, and flexibility at a lower cost (Kandoi & Antikainen, 2015). Nonetheless, despite its numerous advantages, SDN security remains a source of concern among research communities.

SDN is a new networking technology that eliminates the limitations of traditional networks. Traditional network bottlenecks included the complexity of traditional networks, configuration of individual devices using vendor-specific languages, a lack of a global view of the network, and a centralized controlling point (Jose et al., 2021). With the introduction of SDN, a global view of the network became possible, allowing for easier network configuration and management. The separation of the control plane and the data plane is the main feature of SDN. The SDN architecture

is centered on a logically centralized controller that serves as the network controller. The separation of the control plane and the data plane is the main feature of SDN. The SDN architecture is built around a logically centralized controller that serves as the network's brain. The controller functions as the network's operating system. The network becomes programmable through high-level programming languages, easily configurable, and manageable in the SDN architecture (Jose et al., 2021).

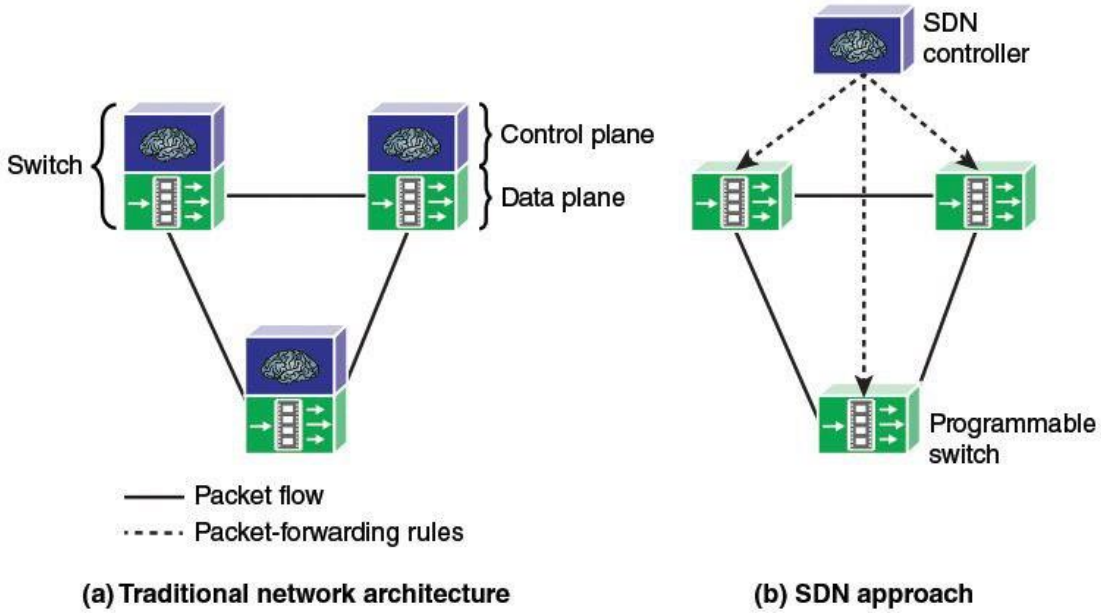

Figure 1.1: Traditional Network vs SDN (Stallings, 2015).

The controller's centralized nature makes it vulnerable to flood attacks, which can disrupt service across the entire network (Deepa et al., 2019). The impact of DDoS attacks on SDN networks is one of the most critical challenges. A DDoS attack on the SDN controller could deplete its processing resources, rendering it inaccessible to legitimate packets, affecting service availability (Thomas & James, 2017). When compared to traditional networks, DDoS attacks will be more effective and cause more damage in SDN networks (Mousavi & St-Hilaire, 2018).

There are currently several research works that propose various methods of detecting and mitigating DDoS attacks in SDN environments, but there is no methodology that serves as a guide for the implementation of these solutions. The Distributed Denial of Service (DDoS) attack has emerged as a significant threat to cyberspace as shown in the figure 1.2. DDoS aims to deplete the victim's resources by preventing

legitimate users from accessing resources, causing financial and reputational harm. Although SDN is a promising solution and the future of networks, it is vulnerable to DDoS attacks.

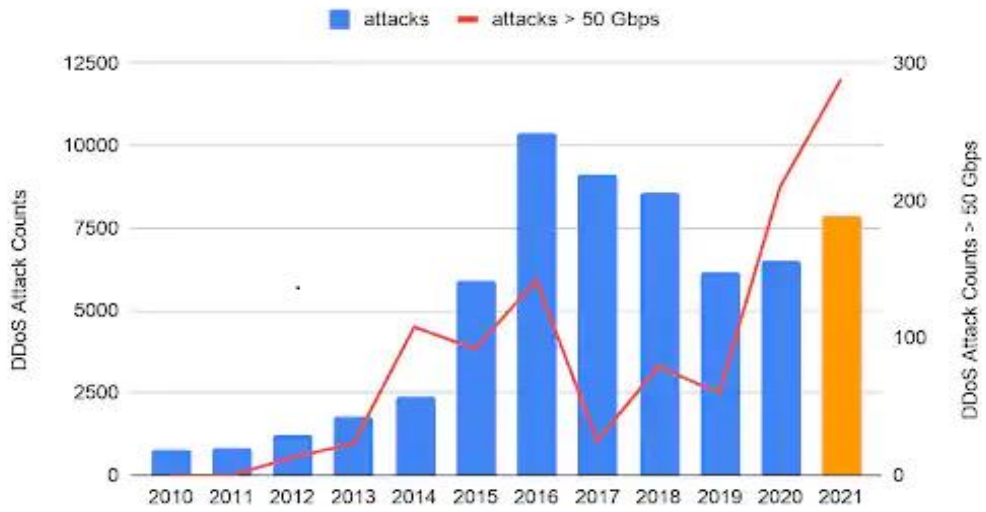

Figure 1.2: Growth Trends for DDoS Attacks and Predictions by Year (Emmons,2021).

DDoS attacks, as the name implies, are distributed in nature and can be launched across the globe by distributed botnets. The distributed nature of the attack, the variable duration pattern of the attack, the variety in the volume of the attack, the use of spoofed IP addresses, and the difficulty in identifying the traffic features are some of the main reasons why DDoS is difficult to detect and address (Jose et al., 2021). DoS can affect both the switches and the controllers in an SDN scenario. The switches in SDN are simple forwarding devices that forward packets based on the rules in the flow tables that the controller inserts. When a switch receives a packet, it compares it to the matching rule in its flow table and decides whether to perform the action defined for that rule. Figure 1.3 depicts an attacker sending a botnet to multiple devices all around the world. Usually, users are unaware that their devices have been infected with a botnet. The botnet armies will then receive a command from the attacker to attack at predetermined times and generate massive amounts of traffic to the victim's SDN application layer, data layer, and control layer server or service. As a result, the legitimate user's service will be disrupted. Because of

modern-day telecommunication network interconnectivity, the magnitude and frequency of DDoS attacks are astounding. Aggressors use botnets to spread quickly.

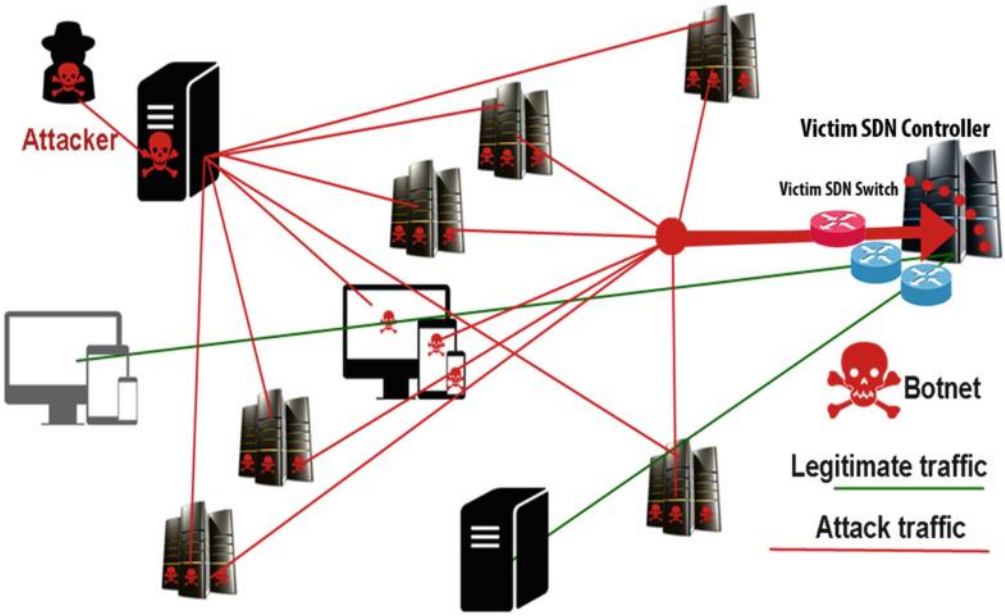

Figure 1.3 : DDoS Attack on SDN Using a Botnet(Haque et al., 2018).

1.2 Motivation

This work would like to begin a chapter in the thesis by encouraging networking and troubleshooting by demonstrating why it's interesting to look at it and what it can offer to better understanding it. However, in order to properly describe everything, this paper need to provide certain network security objects. It is a strange idea to begin with a section describing the security solution fundamentals and then go on to the real deployment motive.

My career journey towards solution development and identifying gaps in industry standards began at my former employer organization, where I worked as a Network and Security engineer, achieving Ethiopian Roads Authority Symantec Endpoint Protection Implementation, Abay Bank S.C. Cisco Nexus data center switch and core switch implementation, Africa Union Commission network expansion, Addis International Bank S.C ASA to FortiGate firewall migration and implementation and Hawassa Industry Park datacenter migration and implementation. The developed methodology will be a specific guide for the implementation of the solutions related to the subject of this thesis, as compared to the standard, which provides general

guidelines in the area of information security management. This thesis work extends to the institutional research segments of development, software application, and cyber security.

1.3 Statement of Problem

Security in a data network is an issue that must be prioritized; otherwise, the organization is vulnerable to plenty of threats that can jeopardize service availability. DDoS attacks are one of the most critical problems in network security nowadays due to the tremendous disruption they can cause to any type of network infrastructure. DDoS attacks primarily target electronic trading sites, blogging sites, and the financial sector. It is also widely acknowledged that the SDN structure is vulnerable to such attacks.

The most dangerous vulnerability in the SDN environment is a DDoS attack. These threats are designed to prevent the controller's functions, such as online services or web applications, from being available. This is because the controller will have all of its resources dedicated to processing a large number of malicious packets, rendering it inaccessible to legitimate traffic. Because the control plane is the weakest link in SDN security, the feasibility of developing a methodology that combines the benefits of proposals for detecting and mitigating flood DDoS attacks aimed at the control plane with the intent of enhancing SDN security is considered.

1.4 General and Specific Objectives

1.4.1 General Objective

The main objective of this work is to improve control plane security by designing and developing an entropy-based methodology for detecting and mitigating DDoS flood attacks in SDN environments.

1.4.2 Specific Objectives

- To look into the existing literature on software-defined networks and their applicability and relevance to this research project.
- To explore the different types of DDoS attacks in SDN, along with their detection and mitigation mechanisms.

- To design and develop an entropy-based methodology for packet generation and detection of DDoS flood attacks in SDN.
- To design a methodology to launch a DDoS attack from a few selected hosts against a specific target and detect and mitigate it.
- To perform a comprehensive analysis of the proposed methodology with its counterparts.

1.5 Significance of the Study

The research study area is a controlled simulation environment in which an SDN will be implemented. Tools will be used to run the tests because it is commonly advisable to employ the standard network virtualization tools for SDN. The quantitative approach was used to address the research problem. The quantitative data obtained from the simulation is interpreted, and the proposed methodology has been validated. This thesis provides relevant concepts from which the following points will be signified.

(a) Traffic generation and detection: We will run the packet generation program from one of the hosts, which will generate random source IP addresses and send them to random destinations. The entropy could then be seen in the controller terminal.

(b) Attack launch and detection: In this case, we launch the attack from a few selected hosts to a single target. The controller's entropy value is going to decrease.

(c) Following that, if the entropy of each set of processes, i.e., packets, falls below a certain threshold, DDoS is detected and those IP addresses are blocked.

1.6 Delimitation and Limitation of the Study

The purpose of this thesis is to construct a methodology for integrating a mechanism for detecting and monitoring DDoS attacks on the SDN control plane. As a promising technology, SDN and its associated security issues lack the necessary methodological proposals for implementation. The standard's structure will be based on the outlined mandatory sections, which are also consistent with the series of steps.

The application of these internationally recognized standards ensures a complete and orderly methodology that can be used in any type of organization.

The methodology will be developed by first identifying the risks and technical vulnerabilities of the SDN controller and then establishing an adequate solution based on the requirements demanded by the solution. The proposed solution will be tested in a controlled simulation environment to collect data, identify and correct flaws in the methodology, and finally validate it.

As a result of the implementation of this methodology, a guide for detecting and mitigating this type of security impact will be obtained as shown in figure 1.4. It will be a guide that, when followed precisely, will assist professionals in technological areas, particularly information security, in improving the security of the SDN control plane, which is the most critical point in this type of architecture.

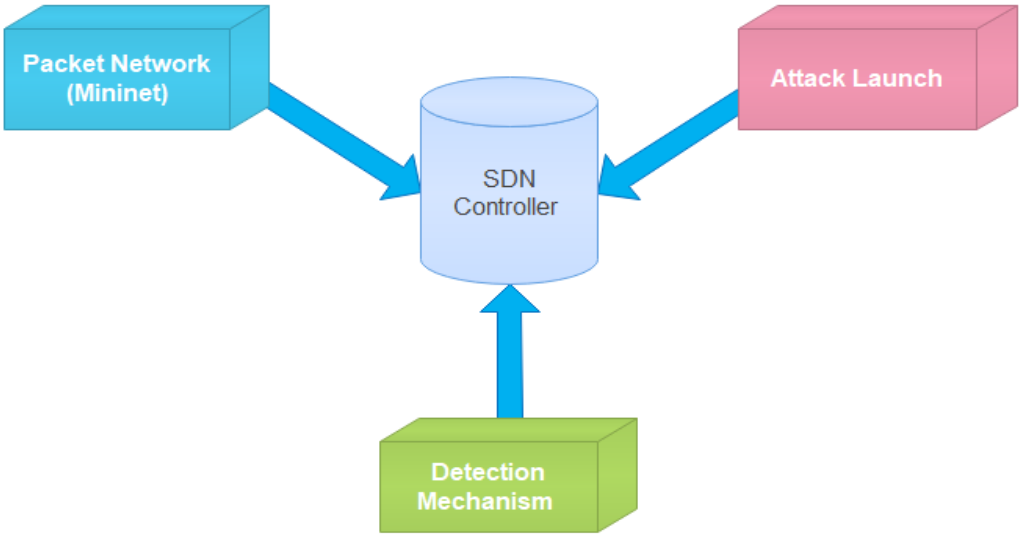

Figure 1.4: System Design Components.

CHAPTER TWO

2. LITERATURE REVIEW

2.1.1 Chapter Overview

In this chapter, a review of previous work on the subject of study is performed, as well as a broad development of the concepts required to support the problem raised.

In addition, we will examine the key building technologies of the SDN architecture and assess the security vulnerabilities of each SDN interface. This analysis is critical in building the necessary security for the SDN controller. The method of attack on the controller is described, as well as how existing security solutions slip down challenges in meeting this security challenge.

2.1 Related Works

SDN technology is increasing Because of the popularity of big data, a programmable controller is required to design new instructions or rules to deal with distinct and unique network traffic flows. Simultaneously, attempts to disrupt online services and breach online systems are becoming more common from adversaries who use a variety of methods to achieve their goals (Masoudi & Ghaffari, 2016).

On the other hand, DDoS attacks with varying traffic rates are the most elusive and destructive threats to computer systems and networks. Several notable studies on DDoS attack detection on SDN networks have been conducted in recent years. Some of the existing approaches also include DDoS attack mitigation. However, when confronted with simultaneous low and high-rate DDoS attacks, most existing detection methods performed poorly. This section examines various relevant efforts concentrating on SDN security against DDoS attacks.

The state of the art of DDoS attacks on SDNs and computing scenarios in the SDNs cloud. A summary of research work on this type of attack is made, as well as open problems to be identified (Dong et al., 2019).

(Kandoi & Antikainen, 2015) emulates attacks and analyzes their effects using the Mininet framework. Furthermore, some mechanisms have been proposed to lessen the impact of these attacks. The settings (timeout value and control plane bandwidth) that must be adjusted according to network requirements to avoid DoS attacks are

also discussed (Kandoi & Antikainen, 2015). In this regard, Kandoi and Antikainen examined the effects of denial-of-service attacks on OpenFlow SDN networks in their article (Kandoi & Antikainen, 2015). Furthermore, some mechanisms have been proposed to lessen the impact of these attacks. Considerations (such as timeout values and control plane bandwidth) that must be adjusted to meet network requirements to avoid DoS attacks are also affected. On the other hand, (Sahoo et al., 2017) proposes a DDoS attack detection mechanism based on General Entropy (GE). The experimental results show that this detection mechanism can detect the attack quickly and with high detection accuracy and a low false-positive rate.

(Mousavi & St-Hilaire, 2018) proposes using SDN's centralized control for attack and introduces a resource-effective and lightweight solution. It demonstrates how DDoS attacks can deplete controller resources and propose a method for detecting such attacks based on the entropy variation of the target IP address (Mousavi & St-Hilaire, 2018) .

Finally, (Dao et al., 2016) in their research, suggests a methodology based on IP filtering to prevent DDoS attacks. The approach discussed in this article analyzes user traffic using the OpenFlow protocol to detect and prevent attacks. When a DDoS attack occurs on a network or server, the only method to limit the damage is to disconnect the network or server from the internet, the main server, or to conceal the source of the DDoS attack. The attacker has access to a large number of dispersed hosts from which the attack may be launched. The spread attack from a large number of servers in the network makes it difficult to identify and stop the source. The most of proposed approaches are based on continuous traffic analysis, which entails controller overhead, resulting in CPU overhead and huge memory depletion.

Control plane DDoS attacks are the main research points of most of the researchers as it is the core of SDN network. Current research work has focused largely on high-rate DDoS attacks on SDN control plane. In the proposed architecture controller is vulnerable to DDoS attack has only detection module with no method to mitigate the attack. Table 2.1 presents a solution to these limits provided in this work.

Table 2.1: DDoS Defense Solutions Based on Information Theory in SDN.

S. No.	Title (Publication-Year)	Problem	Solution	Methodology
1	A Statistical Model for Early Detection of DDoS Attacks on Random Targets in SDN Springer-2021(Shohani et al., 2021)	Early detection methods are not working properly to detect DDoS Attacks.	Author presented a statistical model to detect DDoS attack at early stage through traffic analysis on their own design testbed.	Designed an algorithm which is based on Trapezoidal model.
2	Defense mechanisms against DDoS attack based on entropy in SDN-Cloud using POX controller Springer-2021(Mishra et al., 2021)	DDoS detection in cloud based SDN network is still a challenge.	Entropy based solution is proposed between normal traffic and DDoS traffic with low computation overhead.	Autor designed DDoS detection and mitigation algorithm.
3	A hybrid entropy-based DoS attacks detection system for SDNs: A proposed trust mechanism Journal- 2021(AbdelAzim et al., 2021)	Quick detection of DDoS attack in SDN Network.	Author proposed a hybrid method for DDoS detection and incorporated the information in routing decision.	Entropy based and Kullbackleibler divergence-based solution has been proposed.
4	A cooperative DDoS attack detection scheme based on entropy and ensemble learning in SDN Springer-2021(Yu et al., 2021)	DDoS attack detection with low overhead on controller and detection delay of DDoS attack.	Entropy and ensemble learning based cooperative DDoS attack detection scheme is proposed.	Two detection modules are used in edge switch and other is in controller.

2.2 Overview of Software Defined Networking (SDN)

2.2.1 SDN Architecture

SDN is a network architecture that allows network engineers to dynamically initiate, control, alter, and manage network behavior using open interfaces such as the protocol openflow. SDN is changing the way IT network infrastructures are managed, controlled, and configured (Lawal & Nuray, 2018). The SDN perspective is based on the separation of the control plane and the data plane., with one making data forwarding decisions and the other carrying them out. The non-control plane, like the SDN architecture, is managed by All network flow forwarding choices are made by a centralized controller.

The Open Networking Foundation (ONF) specifies the OpenFlow protocol is used to communicate between the two planes. The SDN architecture is comprised of three planes, as seen in figure 2.1: the application plane, the control plane, and the data plane. In the SDN architecture, the control plane is made up of an SDN controller, which is a separated function from the data plane. The data plane is made up of packet forwarding hardware like routers, switches, and middlebox appliances (Intrusion detection systems, load balancers, and firewalls). The SDN controller communicates with packet forwarding switches via open-source protocols such as OpenFlow. The OpenFlow protocol will dynamically deploy settings and rules to OpenFlow enabled switches on the packet forwarding plane (Kodzai, 2020). According to (Bannour et al., 2017), the SDN application plane includes SDN applications or programs designed to implement network logic and control strategies.

A Northbound API is used by this top-level plane to communicate with the control plane. SDN applications communicate network requirements to the SDN controller, which translates them into Southbound-specific commands and forwarding rules that define the behavior of individual data plane devices. Routing, traffic engineering (TE), firewalls, and load balancing are examples of common SDN applications that run on top of existing controller platforms. The control plane, on the other hand, is considered the primary layer in the SDN architecture; it consists of a centralized software console that controls application and system communications network

devices via open interfaces (Bannour et al., 2017). It also offers centralized management functions such as topology discovery, state synchronization, and device management (Li et al., 2018). It also allows the installation of third-party programs or applications. The control SDN is also known as Network Operating System (NOS), and it already supports network control logic and gives an abstract picture of the global network to the application layer, containing the information needed to establish time-based rules (Bannour et al., 2017).

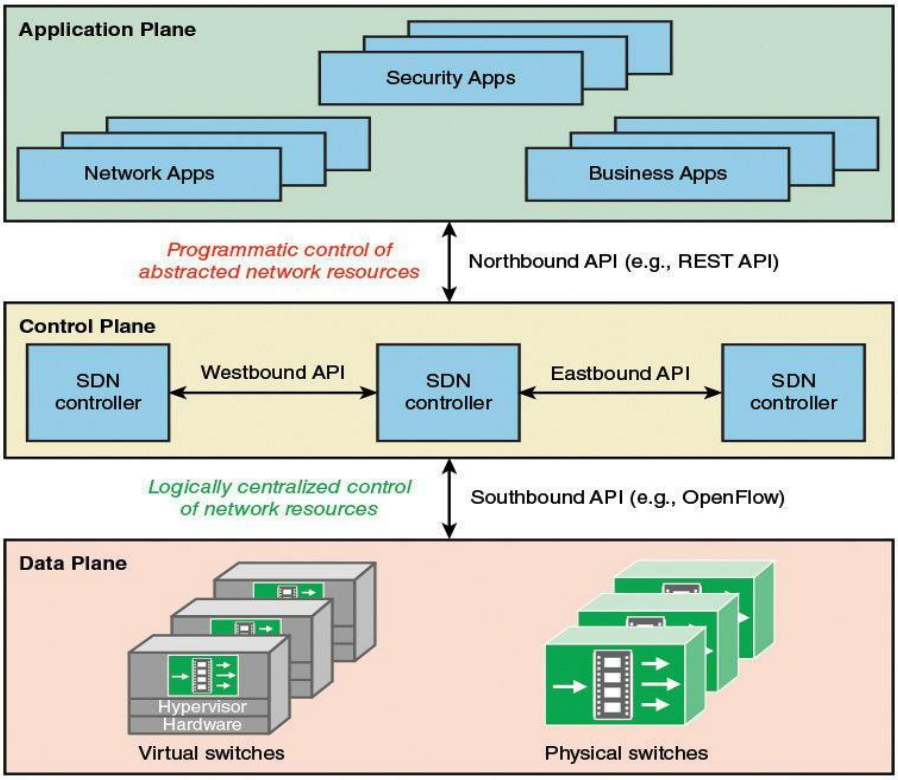

Figure 2.1: SDN Layered Architecture (Stallings, 2015).

(a) **The Control plane** is regarded as the "brain" (Kreutz et al., 2014), It is in control of all decisions and configuration updates supplied to packet forwarding devices in an SDN-based network. The control plane will contain the SDN controller, whose security is a vital component of this study. The SDN controller contains the rules set, logic, and decisions that are forwarded to the data plane via the OpenFlow protocol's southbound API. As a natural outcome, it has evolved into a vital component that governs the entire network (Kreutz et al., 2013).

(b) **Data plane (Infrastructure Layer)** is a group of network devices that provide packet forwarding services on the network. The packet forwarding devices can be hardware or software-based, and they are dynamically configured once the controller sends configuration rules. The configuration rules are broadcast over the SDN controller's southbound interfaces using the OpenFlow protocol. The packet forwarding devices will be routers, switches, or appliances similar to those used in legacy networks, but without embedded control network intelligence. In legacy networks, routers make packet forwarding decisions based primarily on the destination IP address, whereas in SDN, packet forwarding actions are determined by flows of packets between both source and destination devices (Kodzai, 2020).

(c) **The Southbound interface** is an application programmable interface that defines the set of instructions sent to packet forwarding devices via the OpenFlow protocol. They serve as bridges between packet forwarding devices and control devices (Anthony, 2015).

(d) **The Northbound Interface** is the application programmable interface in the architecture that interacts with northbound applications. The interface enables the deployment of middleboxes As services in the SDN architecture, load balancers, intrusion detection systems, and routing algorithms are examples (Anthony, 2015).

2.2.2 SDN Controllers

The SDN controller is the heart (Anthony, 2015)14 of the SDN network, containing centralized logic, a policy database, and all the rules required to carry out predetermined actions on traffic flows passing via the forwarding plane The SDN controller enables the SDN network to be programmable and to dynamically react to changing network circumstances. Several SDN controllers have been launched by various manufacturers, and some are currently being upgraded utilizing various programming languages like as C++, Python, and Java. The programming languages have benefits and drawbacks, with Python-based controllers having issues with multithreading and C++-based controllers having issues with memory management (Hadi et al., 2018). Beacon, RYU, NOX, POX, Floodlight, Maestro, and Open Daylight are examples of SDN controllers. Because of its better documentation and

flexibility, the Python-based Open Daylight project will be used as the SDN controller in the experimental setup for this dissertation (Kreutz et al., 2014). Because it is one of the industry standards that was initially used by prominent companies such as Huawei and Cisco, the controller is the recommended SDN controller. Simply said, as traffic flows on the data plane, the SDN controller will have flows designed either proactively or reactively. The controller will have pre-installed rules in proactive mode before traffic enters the network. Reactive mode flows are implemented when unknown traffic is recognized and goes via the data plane network. (Salman et al., 2016) analyzes numerous SDN controller modules.

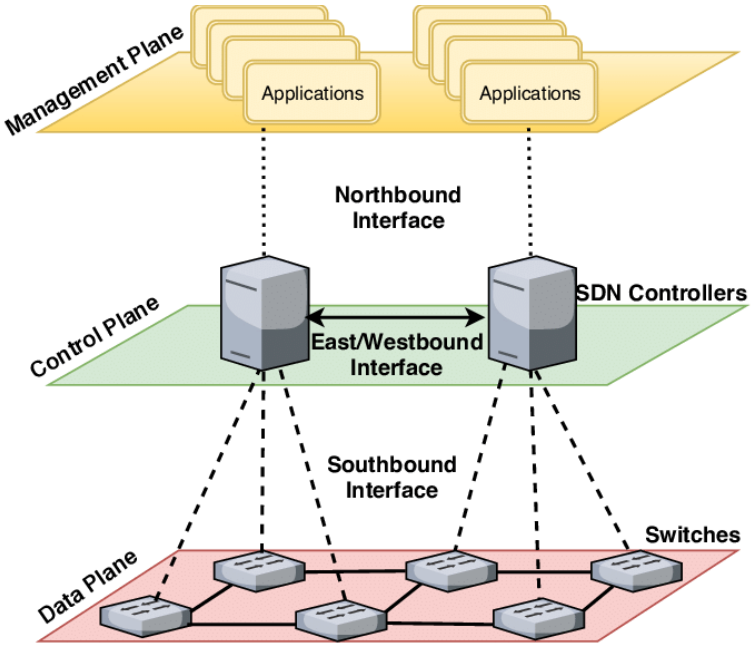

Figure 2.2: SDN Controller Communication Interfaces (Latif et al. (2020).

The routing service module's topology manager and link discoverable node will be in charge of detecting and maintaining the state of the links between the controller and the packet forwarding switches through the OpenFlow protocol. (Salman et al., 2016). SDN controllers may be utilized in large networks in dispersed topologies to increase resilience, capacity, and network system uptime (Kalkan & Zeadally, 2017). By distributing the control function, the distributed solution can be capitalized as a security enhancement that reduces system outage and scaling limitations. The controller performs various tasks such as identifying network

devices and their capabilities, collecting network statistics, and so on. Add-ons such as network monitoring and traffic anomaly detection can be used to extend and improve controller functionality (Centeno et al., 2016). As shown in Figure 2.2, SDN controllers communicate via four interfaces: southbound, northbound, eastbound, and westbound. The controller uses the northbound channel to communicate with the application layer and the southbound channel to communicate with the infrastructure layer. The eastbound and westbound interfaces are used to allow communication between controllers (Haque et al., 2017).

2.2.3 SDN Open Virtual Switches (OVS)

The OVS switches are positioned in the data plane of the SDN architecture and are responsible for Open System Interconnection model (OSI) model IP layer 2 and 3 packet forwarding activities on the user plane. The OpenFlow OVS switch has one or more tables with decision making rules (Salman et al., 2016). Data flows are forwarded based on predefined rules that can be changed as packets travel through the network.

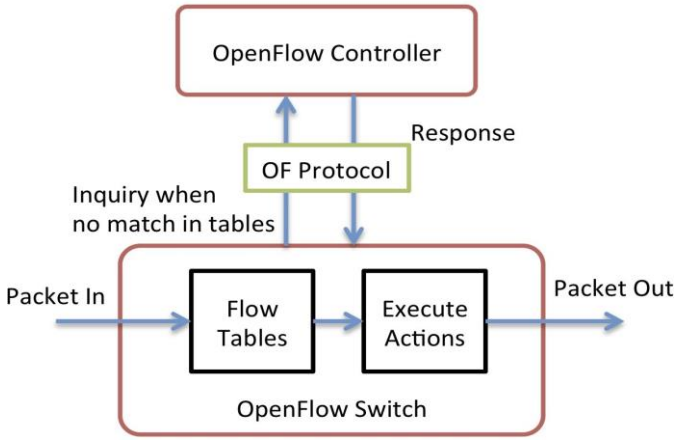

Figure 2.3: OpenFlow message flow (Kodzai, 2020).

As shown in the diagram 2.3, The Southbound interface on OVS switches connects to the SDN controller and supports the standard OpenFlow protocol. OVS switches are important because they provide benefits such as traffic filtering, quality of service, and some amount of security by dividing traffic into virtual local area network tags.

2.2.4 SDN OpenFlow Protocol

The OpenFlow protocol is widely used in the current SDN; it is in charge of secure channel communication between the OpenFlow Switch and the OpenFlow

Controller. The OpenFlow Controller manages a flow table that contains a match field and flows instructions. The OpenFlow Switch manages its flow table, which is supplied by the OpenFlow Controller. OpenFlow defines both the communications protocol between the SDN data plane and the SDN control plane, as well as part of the behavior of the data plane (Ahmad et al., 2015). OpenFlow is a widely accepted and used technology for implementing and deploying SDN technology in today's networks. OpenFlow specifies how a centralized software driver communicates with a network forwarding device, and it provides a unified view of the network. Network policies and services are implemented as OpenFlow applications, which communicate with the control plane via the control plane's Northbound API. The switch and the controller communicate via a Transport Layer Security (TLS) enabled channel. Each switch, according to the specification, must have one or more flow tables that will maintain the flow inputs specified by the controller.

OpenFlow is a key standard communication protocol defined by the Open Network Foundation (ONF) that operates on the OVS switches' southbound interface to the SDN controller (Kodzai, 2020). In the event of new data plane traffic detection, the protocol enables dynamic flow table updating in OVS switches. OpenFlow is a TCP/IP-based transport mechanism that operates on ports 6633 and 6653. The three primary forms of OpenFlow messages are described in (Kodzai, 2020).It may be characterized as controller to switch flow (asynchronous) started by the controller, asynchronous from the switches to the controller, and symmetric messages sent in both directions without solicitation (Kreutz et al., 2014).

The OVS switch's OpenFlow operation entails recursive packet matching in a series of flow tables. Packet matching is performed by predefined priority tags, which determine the actions and rules that will be applied to traffic, such as discarding, blocking, forwarding, or changing (Kreutz et al., 2014). If there is no rule match in the OVS switch, the table that analyzes the flows is called the table miss flow entry (Kodzai, 2020), and it is configurable with options to drop the packet, send it to the controller, or push it to the follow table. The flow tables of the OVS switches are scanned sequentially until the last table with 0 priority is reached, which determines

the last potential action that may be done to that packet. Because of the use of OpenFlow, the controller becomes a primary target for the attacker for the following reasons.

(a) There is no standard for security implementation in the OpenFlow protocol, so product developers use their proprietary methods.
(b) Because SDNs are programmable, they are much more vulnerable to a wide range of malicious attacks and code exploits.
(c) Denial of service and side-channel attacks can be directed at the southbound interface.
(d) SDN configuration errors can be more serious than traditional network errors.
(e) It is also critical to establish trust.

2.2.5 SDN Operation

The most basic SDN operation is to transfer traffic in the network for a packet. Every time a new packet arrives, it checks its flow tables for matching information. If a match is found, the packet will be forwarded directly according to the instructions in the table. If no match is found, the packet is routed to the Controller. If a match is found, the Controller will assign a new rule to the packet based on the instruction field in its flow table. If not, the packet will be dropped shown figure 2.4.

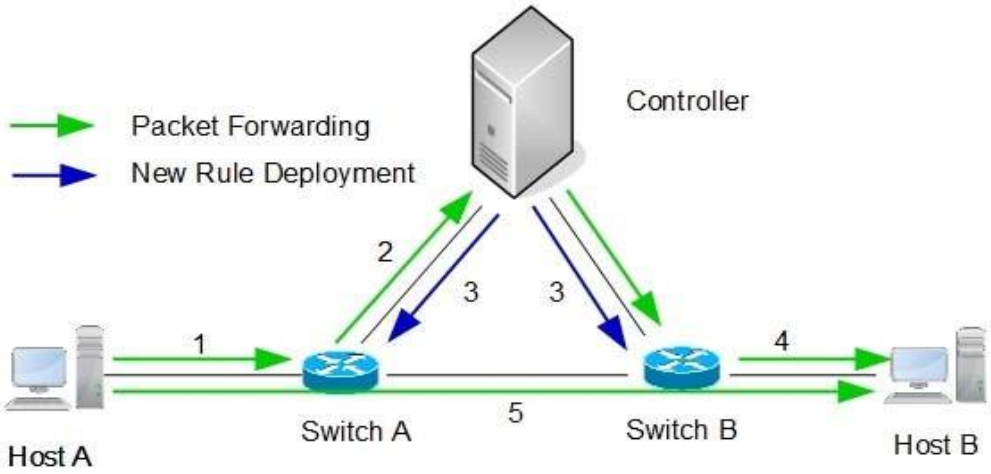

Figure 2.4: Basic SDN Operations(Glăvan et al., 2019).

The new rule is then applied to flow tables in network devices so that the network device knows what to do with similar packets. In terms of basic SDN operation, an attacker can use the new packet mechanism to make the controller unreachable. SDN's basic function is to transfer traffic for a packet across a network. Each new packet to the switch triggers a lookup in the flow table. The flow action will be carried out in order for the match to be successful. Otherwise, the package will be delivered to the driver, who will contact you with further instructions.

2.2.6 POX Controller

There are several types of SDN controllers, including Beacon, RYU, NOX, POX, Floodlight, Maestro, and Open Daylight. SDNs' brain is made up of controllers. The controller performs various tasks, such as identifying network devices and their capabilities, collecting network statistics, and so on. There are currently several open-source driver implementations available, written in various programming languages such as Python, C++, and Java (Kodzai, 2020). POX is an opensource OpenFlow (SDN) controller written in Python. It is used for faster network application development and prototyping can be portrayed in figure 2.5.

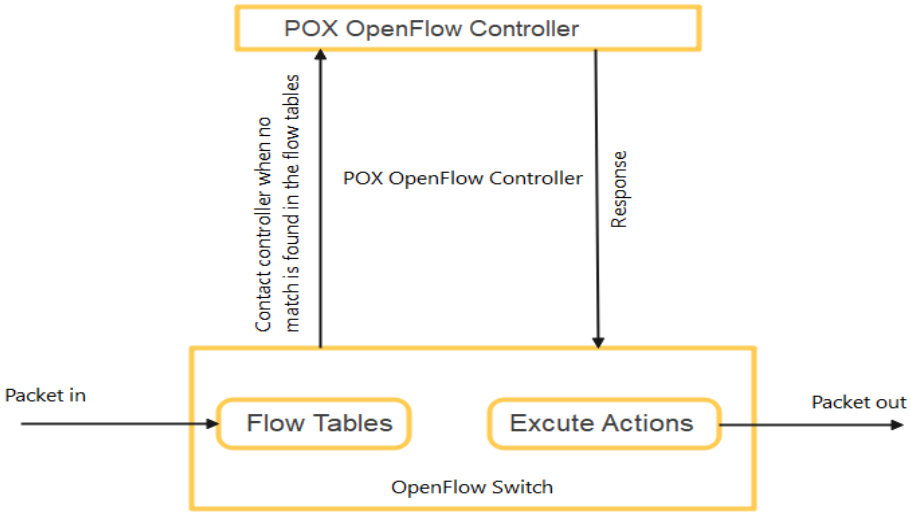

Figure 2.5 : POX Controller Architecture.

The Mininet virtual machine is pre-installed on the POX controller. Using the POX controller, you can transform dumb OpenFlow devices into hubs, switches, load balancers, and firewalls. The POX controller simplifies the execution of OpenFlow and SDN experiments (Kodzai, 2020).

2.3 SDN Security Issues in Brief

When the paper first started looking at security and SDN, it is suggested that two distinct ways to think about security and software-defined networks. This paper will look at security through SDN. What are the new benefits that SDN and security may provide and how will it look at SDN security? Because, once again, the layered approach brought by SDN necessitates that each of these layers inside the SDN model be secure in its own right. The strength of a security system is only as strong as its weakest link. Because, as seen in the previous, an SDN implementation comprises multiple layers. It must underline once more that security is only as strong as its weakest link. SDN can be used to provision and enforce the security plan. Security is an important consideration in all network designs, including legacy networks and SDN-based architectures. The SDN scheme may introduce new challenges not seen in traditional networks.

The list of security challenges to be mitigated is expected to grow as SDN technologies are gradually deployed. Security flaws in SDNs are concentrated in three planes or layers (application, control, and data). Security advantages of SDN are effective monitoring of abnormal traffic and timely dealing with vulnerabilities. SDN and personal, national, and hybrid cloud networks exist here, and web operations teams are distributed. Companies in all industries are at various stages of researching, evaluating, and implementing current system technologies. SDN provides the equipment required for on-premises and cloud network speed, scale, and self-service.

Organizations that want to shift from reactive to preventive system management must use performance monitoring. According to the article, a standard measure is required to see when resources are over or underutilized, and setting lower thresholds on the standard identifies potential problems before they impact the system. Web safety, particularly as it relates to the world's largest system, the internet, has emerged as one of today's most high-profile data security issues. Some educational institutions have already linked their technology resources into a single system, while others are in the process of doing so. The next step for these organizations is to weigh the costs and benefits of establishing a connection between

their personal networks (with their trusted users) and the internet's obscure users and networks.

Regardless of the deployment model, whether classic hybrid or virtual overlay, or the controller agent communication protocol, such as open flow versus vendor specific API's or I2RS SDN, an organization's perimeter security is provided. SDN has altered the definition of the network perimeter. There is no longer a single boundary or single device separating the outside and inside of the Internet of everything concept. In situations where any IP-based device connected to a network may pose a threat, each network element or function within an organization must be secured in its own right. SDN components must provide secure boundaries around services and access; this is referred to as SDN security. Furthermore, the SDN functionality must be protected as if each layer in an implementation had its own perimeter.

2.3.1 Security for the SDN

The goals of security and software-defined networks are as follows when it comes to SDN and security. Instead of acting as an overlay, the SDN design enables security to be embedded into the network. It does not rely on the SDN model. As previously noted, regardless of the model used, security is a concern. Design needs for security can be incorporated.

Centralized policy administration, automatic provisioning, real-time mitigation, and the ability to execute security checks on configurations as they are implemented on the network are all advantages. These are all benefits of SDN, and they are all built in from the ground up and from the start. Another consideration is that many common network attacks will have varying effects on an SDN system. Because in today's networks, it has things like DDoS attacks and well-known attacks, and some of them will influence the network in an SDN network in a somewhat different way. The process for putting them in place will change as well. There is a bit of a learning curve when it comes to knowing how some of these dangers will be applied and arise in an SDN network. In a normal DDoS attack, an attacker will often target numerous devices in the first step of a DoS attack.

By focusing on each device individually, it will develop invalid flows for each of them or perhaps it will try to slow down the CPU on a single device in order to influence the throughput and service provided by that device. In general, it will try to attack as many devices as it can. However, this occurs on a per device basis, and in most circumstances, it is limited to that unique device. Considering a DDoS attack on an SDN network, it can be seen that the attacker only needs to target one device in the network, maybe feeding fake flow information onto that device via the data plane. The modified flow information will then be returned to the controller through SDN agent. If the information forwarded by the agent is faulty in some way, this can cause the controller CPU to spike. Another scenario is that the controller will use the incorrect information to communicate the improper network flows to other agents in the network, harming more than one agent. Even by concentrating on and attacking just one agent, it can be seen how the SDN DoS attack spreads.

The success of this method, however, is reliant on how well the SDN is protected. An SDN is composed of numerous levels and interactions, each of which necessitates a different level of security. Multi-layered security is required for multi-layer, multi-component models. SDN security brings new potential threats and weaknesses, with DoS attack since SDN is a layered design, each layer needs its own security. The easiest method to ensure this is to identify each of the levels and then apply appropriate mitigation requirements and security techniques to aid in layer protection management.

The control of an SDN network is its beating heart. It connects network devices on one end and applications on the other. Any interaction between applications or devices must go through the controller. Controllers employ open flow protocols such as OpenFlow to configure network devices and pick the optimum network path for application traffic. Approaches to controller hardening aim to protect the control from access-based attacks and misconfigurations. Furthermore, any operating system running on the controller must be patched and maintained appropriately. Management security includes always ensuring that when you contact the controller and configure anything on the controller, you utilize secure protocols such as SSH or HTTPS.

Authentication, Authorization, and Accounting (AAA) with role-based access control should be used to monitor and audit any active buddies that are configured on the control-enabled host-based firewall and only allow the required to be indicated. In many cases, the controller can provide individual host-based firewalling to protect the device from unauthorized access and services. There is also security by design in terms of control specific hardening. SDN agents, and other network devices that receive commands from the controller, such as routers, switches, and firewalls. Many strategies that are currently employed in non-SDN installations are used to harden SDN devices. Some of the important ways include implementing routing plane security elements such as authentication as well as routing protocol specific characteristics such as BGP TTL security. Control plane policing is another option for ensuring predictable CPU activity. Control plane and management plane protection are used to adjust and limit device resources, as are data plane security approaches that try to reject undesired traffic as soon as feasible rather than forward it to other network devices.

Some of the hardening strategies you may wish to apply here include control plane protection, routing protocol security, first hop routing protocol security, and control plane protection. Consider how to manage ICMP traffic by turning off ICMP redirection. In addition, both ICMP unreachable calls and proxies are secure routing methods. The goal of control plane policing is not necessarily to prevent attacks, but rather to ensure that the services that do require network bandwidth and operation are not jeopardized by utilizing pure authentication, route filtering, and managing your resource consumption with routing protocols such as BGP Max prefixes, for example. On the management plane, secure protocols are utilized to manage your infrastructure. As a result, SSH, SCP, HTTPS, and SNMP v3 are no longer supported. Because the fundamental point of safeguarding the data plane is to be able to reject content as quickly as possible before sending it on to other destinations.

To secure any network, the services provided on that network must be specified, as well as the needs based on the network security policy. The idea is to allow only what is required while denying everything else. Closing ports, removing unused services, and managing network access points are simple things to do to reduce

certain potential vulnerability areas. This is also an excellent opportunity to safeguard critical traffic by ensuring bandwidth and prioritizing specific services. Using traffic policing and shaping to reduce the use of new and undiscovered risks is an effective way to do so. Disabling unnecessary ports, protocols, and services, as well as employing an infrastructure access list to identify your necessary flows and only allow those to access the device, are some of the approaches to adopt for network service security. Firewall protection can be used to secure network services, police traffic, and prioritize key flows.

In an SDN deployments, the next layer is applications and APIs. Many network devices operate on closed operating systems with few or no API interfaces to external-to-external applications for data collection, management, or provisioning. To extract and convey information to other applications, well-known protocols such as Syslog, SNMP, and SSL are employed. SDN enables the development of applications with direct access to the controller, allowing them to directly modify network aspects. It is vital to utilize secure coding methods while designing apps to preserve the controller's integrity from unstable and unsecure code.

Threat mitigation solutions should be implemented on and around agents, and threat management protocols should be in place. One of the most effective agent threat management processes is the capacity to identify certain attack types, such as denial of service attacks, spoofing attacks, fragmentation attacks. It is invaluable to be able to connect them to specific attack mitigation measures, which frequently include detecting attack patterns and comprehending multiple attack pathways. Concerns about security in legacy networks must still be addressed in SDN environments.

Static firewalls and middle box appliances will be transferred to SDN architectures and delivered as services on these platforms rather than vendor specific hardware appliances. Increasing the risk of network failure due to a single point of failure by centralizing the control function on the SDN controller. Securing the SDN controller becomes critical to ensuring that the network operates normally. The SDN controller security must be deployed in multiple dimensions and must cover all angles that hackers can use as entry points. The SDN controller's security is reviewed by

examining the primary important attack vectors influencing the SDN architectural integration points, which are outlined below beginning with the separate planes.

2.3.2 Security by the SDN

SDN agents and controllers can be configured to act as a perimeter for a specific device or service. Identity management, threat defense, and content inspection, as well as those that impose compliance and regulatory requirements, can all be applied to connections and forms via security services linked into agent systems. Controllers are allocated to attacks and anomalies in order to enforce network-wide containment and disseminate protection upgrades. After deployment, issues with security policy provisioning are handled, and performance monitoring becomes centralized and automated. When configuring a feature, you can check for correct policy, correct configuration, and correct configuration syntax. Implementing a route, prohibiting a routing protocol recall, and then trying to secure that route protocol is an example of an implausible situation.

It is critical to be able to perform a configuration check to guarantee that the configuration is right before it is passed down to an agent. Because, once again, attacks or service disruptions are not always the result of external users attacking the network and it can also be the result of simple configuration issues. Attacks and abnormalities are reported to controllers, who can implement network wide containment and disseminate security updates. By automatically protecting the agents as soon as it detects an attack, may restrict or eliminate DDoS type attacks and the effects they have on the network.

The most essential aspect of SDN security is that it is embedded into the deployment from the outset. It's no longer an overlay. Security is an essential component of core technology and notice the layered approach to the system in this typical SDN implementation. After that, the containment and mitigation strategy can be transmitted to the SDN layer, and therefore to other agents in the network. When this is completed, will have enterprise-wide protection, with policy updates distributed to all network agents.

2.3.3 Vulnerability of SDN Controller

A secure line is used to communicate with the controller. This secure line makes it difficult to gain full control of the controller. In terms of basic SDN operation, an attacker can use the new packet mechanism to render the controller unreachable. The size of flow tables in controller and network devices is limited due to memory constraints. Attackers flood the network with packets with fake addresses. Because the addresses are unknown, the packet will be routed to the controller. If the number of packets arriving at the controller is high, the controller will devote all of its resources to processing the malicious packets (Haque et al., 2020).

A high-rate malicious packet can completely overwhelm the Controller, rendering it inaccessible to legitimate traffic. This may result in the network's SDN architecture collapsing. Also, it is critical to note that this type of attack takes time to obtain a malicious package. DDoS attackers may employ a periodic attack that occurred at some point in time. This method can be used in conjunction with the detection method to increase the detection time before attackers arrive at their target (Glăvan et al., 2019).

SDN security has become the most well-known topic in recent years. SDN confronts two significant security problems. One strategy is to use SDN to address traditional network security issues. Another goal is to secure SDN and strengthen SDN-enabled infrastructure. SDN is subject to seven key threat vectors in terms of the latter. Among the well-known vulnerabilities, the most devastating attack on the entire network is a distributed denial of service (DDoS) attack on controllers.

2.4 DDoS Attacks in SDN

The DDoS attack is a multi-computer-directed attack that uses "bots" or "zombies," which are networks of computers that are remotely controlled by an attacker to launch massive attacks on a specific target (Sahoo et al., 2017). Its motivation is to deplete network resources so that the service is hampered or stopped, resulting in a lack of service availability. A large number of packets are sent to one or more hosts on the network during the attack. If the source IP addresses of incoming packets are spoofed (which is common), the switch will not find a match in its flow table and will have to forward the packet to the controller. The accumulation of legitimate user

packets and attacker-generated packets can adversely impact the SDN controller's available resources for communication, calculation, and storage, potentially exhausting them completely in the worst-case scenario. Even if there is a backup controller, it must deal with the same problem.

DDoS attacks can be held out on all three planes of the SDN architecture. DDoS attacks are classified into three types based on their potential targets: application-layer DDoS attacks, control layer DDoS attacks, and data layer DDoS attacks. Whatever the target, all of these attacks share the trait of flooding the network with massive amounts of packets, typically Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), or packets User Datagram Protocol (UDP).

DDoS is essentially a flood attack. Many packets are sent to a network device to stop the service or reduce the device's performance. If the incoming source addresses are spoofed, the switch will not find a match and the packet will be forwarded to the controller. The accumulation of DDoS spoofed packets and legitimate packets can force the controller into continuous processing, exhausting them. As a result, the controller is no longer reachable for new legitimate packets. This will bring the controller down, causing the SDN architecture to mess up. The same problem exists for a backup controller. Such attacks can be detected early on by monitoring a few hundred packets and looking for changes in entropy.

The early detection of DDoS attacks prevents the controller from being taken offline. The term 'early' refers to the controller's tolerance level and the amount of traffic handled. As a result, the impact of malicious packet flooding can be mitigated. A portable machine with a fast response time is required. The fast response time allows the controller to regain control by terminating the DDoS attack during the attack period (Cui et al., 2021).The control plane is one of the most appealing targets for DoS and DDoS attacks. Because network resources are visible, the SDN controller can become a single point of failure, exposing the entire network to a security attack.

2.4.1 Application Layer DDoS Attacks

DDoS attacks on the SDN application layer target SDN applications as well as the northbound API. Because the separation between SDN applications is difficult to discern, a DDoS attack targeting the application layer may have an impact on an application other than the attacker's target (Dong et al., 2019). These types of attacks establish full TCP connections with the destination server and then begin flooding it with a large number of HTTP requests with the goal of saturating the available bandwidth with the attacker's illegitimate traffic. Because these are slow and low-level attacks, distinguishing them from legitimate traffic is difficult. As a result, attacks on the SDN application layer are tools that attackers can use to harm victims in the current climate (Bawany & Shamsi, 2016). The main challenge in dealing with this security issue is distinguishing between an attack and a sudden influx of legitimate user packets.

2.4.2 Data Layer DDoS Attacks

Even though they contain administrative, access control, and transmission information, the OpenFlow switch and flow table can be targets for DDoS attacks. The attacker's goal is to disrupt network functionality by gaining unauthorized physical or virtual access to the network. Because the storage capacity of the OpenFlow switch is limited, it is impossible to store all of the flow rules. If the attacker sends a large number of packets from an unknown address in a short period of time, the controller writes new rules for these packets and forwards them to the flow table, which quickly fills up and runs out of space to write a new rule (Dong et al., 2019). As a result, legitimate traffic is no longer transmitted. DDoS attacks can also be directed at the flow cache. When the switch receives a packet from an input port, it searches a flow table for a match. If a match is found, the packet is sent from the cache to the output port. If no match is found, the packet is sent to the controller with the message Packet In. The controller responds with the message OFPT FLOW MOD, which assigns the hard timeout and idle timeout, which define the package's rules and how long the rules will be in effect. When the switch receives the rule from the controller, it processes the packet and stores the rule in the transmission table's cache to process incoming packets directly. During a DDoS attack, a large number of packets forwarded to the switch by malicious nodes are routed to the cache and

await the controller's response, which includes flow rule information (Polat et al., 2020). Packets from attacking nodes fill the switch buffer, preventing legitimate packets from being transmitted.

2.4.3 Control Layer DDoS Attacks

DDoS attacks on the SDN control layer aim to cause network outages by overloading the controller with large amounts of traffic from multiple sources. The controller decides how to forward packets based on the flow policy. When the switch identifies a new network packet on the data plane and no flow rules in the flow table match the current flow information, the entire packet or a portion of the header is sent to the controller to resolve the query. Sending a total packet to the controller can consume a significant amount of data bandwidth when there is a high volume of network traffic. The attacker's first step is to identify the SDN network.

It should be noted that traditional networks typically have a forwarding table that is preconfigured. As a result, there is no additional time required to process and create a flow input for a new incoming packet. In contrast, the controller in SDNs requires some time to create a new flow input for a new incoming packet. Furthermore, the driver takes longer to process the first packet than the subsequent packages. Attackers use this knowledge to determine whether a network is SDN or not, using network scanning tools to see if there is a difference between the response times of the first and subsequent packets. DDoS attacks are without a doubt the most typical sort of SDN attack, primarily targeting the control plane and emerging from the packet forwarding plane when asking the controller for unfamiliar traffic Flooding the controller with flow decision requests attacks the control plane, using controller resources and limiting the controller's capacity to handle valid traffic (Abdullah et al., 2019). Christos Bouras explains the impact of a DDoS attack on 5G networks, as well as how network time protocol (NTP)amplification and malformed packets are employed in DDoS attacks to undertake centralized attacks on both packet forwarders and the SDN controller via the NTP protocol., in (Bouras et al., 2017).

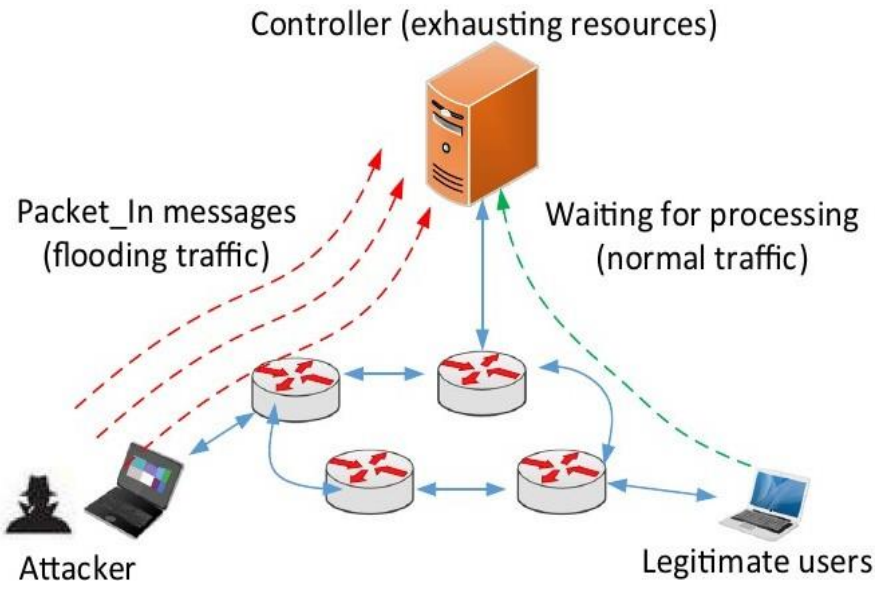

Figure 2.6: DDoS Attack on Controller (Stallings, 2015).

Figure 2.6 depicts how a DDoS attack on the SDN controller is carried out. It is observed when an attacker infects other devices in order to become a part of the infected machines. When the bots are ready to attack, they are given a command to attack for a set period of time. When a data packet does not match the flow table, the OpenFlow switch sends a message to the controller and loads messages packet in exceeding the controller's processing capacity. The controller's resources will be depleted as a result of this type of attack, and legitimate traffic will be restricted access to the controller.

The control layer manages the entire network by controlling all switches. A standard southbound API is used to communicate between the SDN controller and switches (Open Flow). Because controllers may be viewed as a single point of failure for the network, they are a particularly enticing target for DDoS attacks in the SDN architecture. Control plane DDoS assaults can be initiated by hitting the controller, the northbound API, the southbound API, the westbound API, or the eastbound API. DDoS attacks on the control layer are common.

2.5 DDoS Attack and Its Mitigation

DDoS attacks are well-known malicious attempts to exhaust the resources of a computer or network of computers by delivering massive quantities of traffic to them. (Mousavi & St-Hilaire, 2018). The attacker's two main objectives are:

(a) Bandwidth depletion.
(b) Resource exhaustion (Shohani et al., 2021).

DDoS attacks begin with an attacker planting code in compromised PCs known as Botnets. These codes are executed during the attack, and a stream of traffic is directed towards the victim. A more sophisticated attack employs a thin layer of compromised PCs known as handlers to control a larger number of compromised PCs known as zombie hosts. The zombie hosts are in charge of generating attack traffic (Shohani et al., 2021). Using botnets concentrates the attack and keeps the perpetrator hidden behind the scenes. DDoS is one of the most common methods of overburdening and disrupting network service.

Production networks have long been researched, and the many challenges they face are well understood. SDN, on the other hand, is a novel architecture with few articles on the subject. Existing studies look into DDoS and how to apply existing DDoS remedies to SDN. This generally entails treating SDN like a standard production network and ignoring the controller's function. In SDN, switches have no control over incoming packets and waste no time processing them. This also implies that the controller is the entity that takes the most harm during an attack. Figure 2.7 demonstrates the path of a DDoS attack to a host target, with no passing through the controller. The red line represents the adversary path, and POX (Mishra et al., 2021)is the controller. In a DDoS attack, packets are always spoofed or originate from multiple zombie nodes. As a result, packets must pass through the controller to gain access to the target via a flow in the switch table. Not all attacks can be detected or documented, but based on the statistics, they pose an immediate threat to every network.

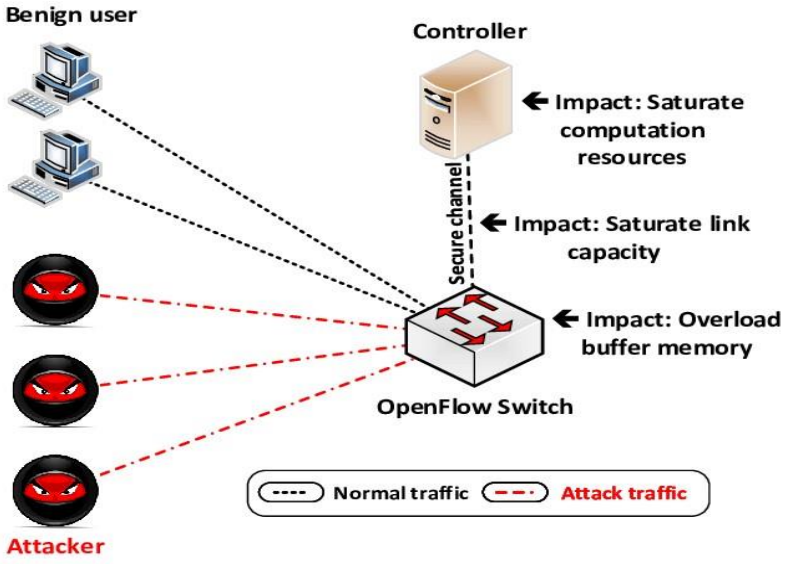

Figure 2.7 : Impact of DDoS Attack (Varghese & Muniyal, 2021).

2.5.1 DDoS Attack Types

To be utilized as zombies, any attack requires access to computers on the victim's network. Hackers use scanning to locate insecure systems in a network. Scanning can be performed at random, using a hit list, on a local subnet, or with an algorithm created by the hacker. As illustrated in Figure 2.8, DDoS attacks are classified based on their attack mode (the manner in which they target) and targeted resource (what they target). This categorization is detailed in and includes focused DDoS assaults on crucial SDN hosting equipment resources such as the CPU and machine memory. By attempting to compromise these resources, the attacker will block other authorized users from accessing critical business applications, resulting in an SDN controller outage and severe revenue loss for operators.

In terms of attack mode, DDoS attacks are conducted utilizing different hacker tools such as man in the middle, IP spoofing, and malicious traffic injection, with the same objective of rendering the SDN controller useless. These attacks will take place either directly or indirectly (Kodzai, 2020) SNMP, DNS, NTP, and ICMP are examples of UDP-based protocols. Other unprotected hosts are sometimes used and exploited as proxies to launch the DDoS attack on the SDN controller. All TCP/IP communications between servers and clients are initiated by a three-way handshake process, it is the basis for all TCP protocol connections The requested port will not

be opened by the server. and will not allow any communication with the client if the three-way handshake is not performed (Hameed & Ahmed Khan, 2018), (AL-Musawi, 2012).

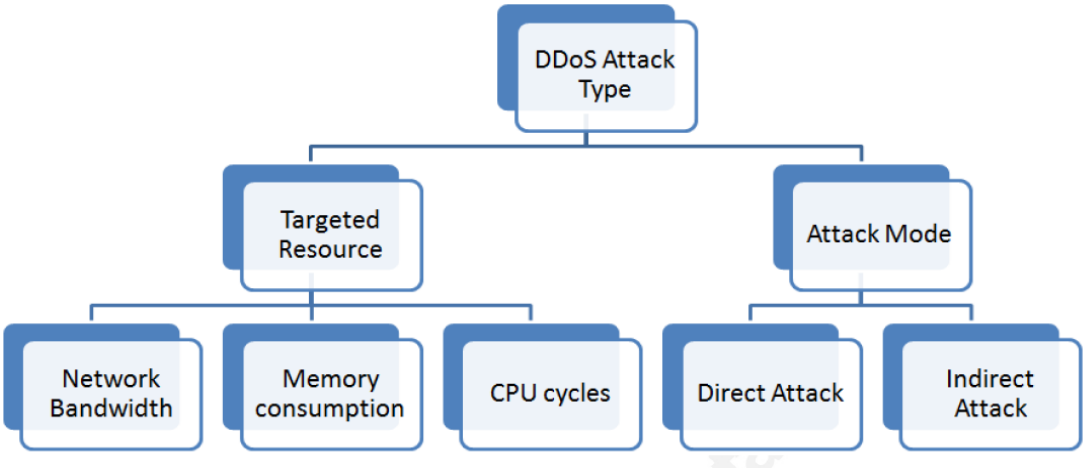

Figure 2.8 :DDoS Attacks Categories (Kodzai, 2020).

Application, host, resource, network, and infrastructure attacks are all types of attacks (Kodzai, 2020).

(a) Application: directing an application to a host in order to prevent legitimate use.
(b) Host: rendering a host inaccessible.
(c) Resource: overburdening a server in order to keep it bound to a constant stream of bogus requests.
(d) Network: sending a large volume of traffic to a network in order to consume all available bandwidth.
(e) Infrastructure: targeting a domain name server in multiple locations at the same time.

In recent years, a few types of DDoS have become increasingly common. The pattern of attacks demonstrates that hackers have been employing specific methods for launching attacks. The server will not open the requested port and will not allow any communication with the client if the three-way handshake is not performed (Hameed & Ahmed Khan, 2018), (AL-Musawi, 2012). DDoS

attacks are carried out by exploiting the three-way handshake procedure, and the attack method is outlined in the following points.

(a) A connection will be requested by the client/user by sending a SYN packet to the server.
(b) The server responds to the client with a SYN-ACK acknowledgement packet.
(c) The client will send an ACK, completing the handshake and establishing the connection (Sakellaropoulou, 2017).

Figure 2.9 summarizes the three-way handshake, as well as how attackers might utilize the flows to launch a breach Hackers and DDoS sources use the 3-way handshake to flood the server with SYN packets. referencing the IP address of a compromised host as the source in the hacking process, which is also known as IP spoofing (Kodzai, 2020).

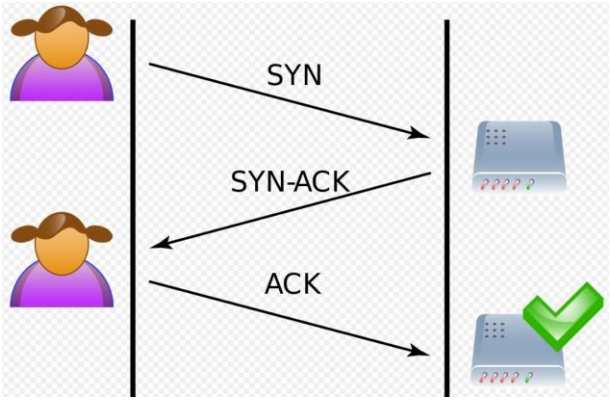

Figure 2.9: TCP/IP 3-way handshake process (Hameed & Ahmed Khan, 2018).

The server responses with SYN-ACK, but the message response never reaches its intended destination, resulting in half-opened sessions that require too many server resources. Due to the influx of these SYN messages, the server becomes overburdened and fails to respond to legitimate traffic due to an excess of incomplete connection requests. DDoS attacks used in testing are the type of targeted resource attack in the context of this dissertation. In the Proof of Concept (PoC), the attacker will target the POX SDN controller in the control plane with the intent of exploiting the controller hosting machine's CPU and memory resources.

Google and Arbor networks have set up a website to track attacks and collect statistics from around the world (Mousavi & St-Hilaire, 2018). The statistics display the type of attack, the duration of the attack, the bandwidth consumed, the most regularly used source and destination ports, the origin of the attack, and the victim's location. The next paragraphs explain numerous sorts of attacks.

(a) UDP Flood attack is one in which a large number of packets are delivered to random ports on the victim's workstation, prompting the machine to look for applications on these ports. The computer must issue Destination Unreachable packets in response to each incoming packet. As the number of incoming packets rises, the delay increases, and the computer finally becomes unreachable.
(b) SYN Flood is a type of attack that uses TCP connection establishment to the victim's system. Several SYN packets are sent to the target, but no ACK is received, forcing the attacker's resources to deplete as it waits for acknowledgement. Up to mid-2021, TCP Flood was the most prevalent identified attack, accounting for 48.7 percent of all attacks (Mousavi & St-Hilaire, 2018).
(c) The DNS Reflection attack includes transmitting faked IP addresses and demanding a response that is significantly bigger than the request, which is subsequently sent to the target. The attacker changes the source IP address to the victim's IP address, directing a significant volume of traffic to the victim. Spoofing packets is a typical DDoS attack method.
(d) The practice of making a high number of requests to a server and overloading it to the point that it cannot reply to valid requests is known as HTTP flooding. With 37.2 percent, this was the second most prevalent sort of attack (Shohani et al., 2021).
(e) Another sort of attack is ICMP Flood, which depletes the victim's resources by delivering a high number of ICMP pings (echo requests), preventing the server from replying (echo replies).

All of these attacks have one thing in common: they all generate a large amount of traffic on the victim's network, depleting its resources. The identification and

mitigation of these high-volume flows in today's traditional networks will aid us in understanding their influence on the SDN architecture.

2.5.2 Techniques for Detecting DDoS

In IP networks, there is a limit to the amount of bandwidth and processing power that can be used to carry traffic. When some network attributes are statistically analyzed, a pattern will emerge for each attribute. The longer the period, the more consistent the pattern. However, this is only true if the network has constant traffic. If there are variances in traffic that are viewed as usual, the numbers will normalize and cannot be considered totally dependable in the long run. a number of methods are used to collect, filter, and process data for anomaly detection. Here are some of the common methods of DDoS detections.

(a) Entropy-based Method

Entropy was used in this study to detect DDoS attacks in SDN. Before introducing it into SDN, it is necessary to examine the methods used in non SDN networks. DDoS detection with entropy requires two components: a threshold and a window size. The time period or the number of packets affect the size of the window. Entropy is calculated inside this timeframe to estimate the uncertainty in the arriving packets. An assault must be detected by a threshold. Depending on the technique, an attack is detected when the estimated entropy exceeds or falls below a threshold.

(Wang et al., 2015) propose a method with a 0.1-second window and three threshold levels. The goal of this method is to avoid false positives and false negatives in the network. However, as the authors point out, the method is time consuming and requires more resources. (Aladaileh et al., 2021) propose a faster method of computing entropy by basing the calculation on both packet type and network volume. A time period window is also used in this method. The authors ran several datasets to find a suitable threshold, which is a multiple of the standard deviation of entropy values. The false negatives in this method are higher than in other methods, but the false positives are lower. There is no indication of the percentage of accuracy. There is also no mention of fast computation resources.

Entropy has been used in various approaches to detect DDoS attacks in the network, but to the best of our knowledge, it has not been deployed in SDN. The limits of available resources, as well as the rapid detection of threats, are critical components of any detection task in SDN when passing packets to the controller. In this study, it will use entropy to detect DDoS while keeping the controller's limitations in mind.

(b) Method based on User Traffic Analysis

This method distinguishes frequent users from "abnormal" users (error packets or DDoS) based on the number of packets received. (Dao et al., 2016) work is based on a study conducted on the University of Auckland's data network and a small ISP, which concluded that approximately 90% of frequent users sent fewer than five packets to each destination, while "abnormal" users sent fewer than five packets per connection.

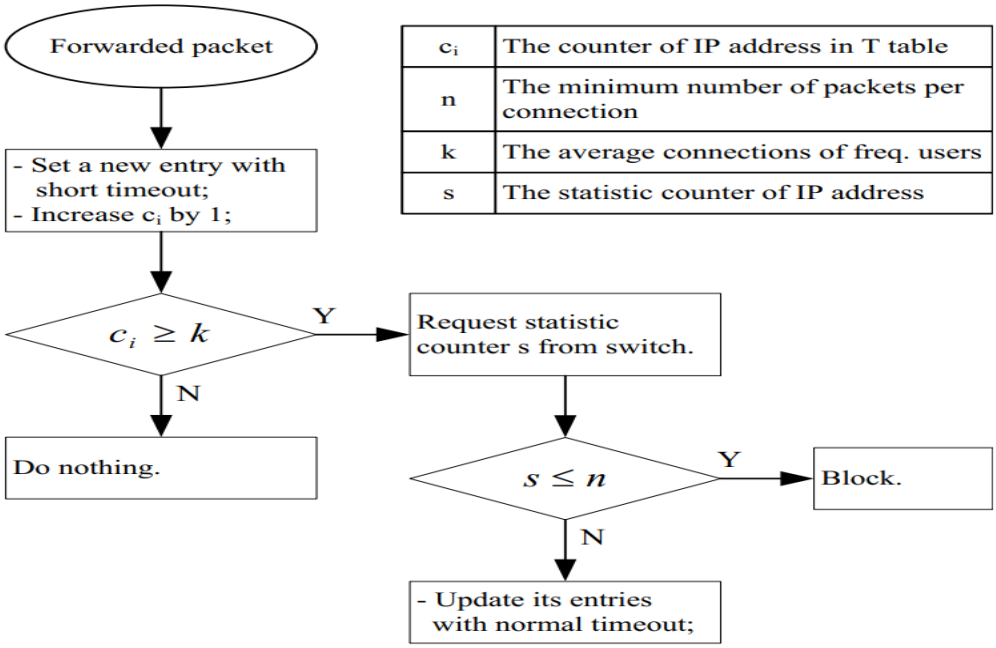

Figure 2.10 : Flowchart of the Method Based on User Traffic Analysis(Dao et al., 2016).

The flowchart for this method is shown in the figure 2.10. A frequent user's minimum number of packets per connection is denoted by n, while the average number of connections corresponds to k. Initially, a T table in the controller is defined to store the source IP addresses of packets c forwarded from the switch. Each

IP address has a ci counter that keeps track of the number of packets received. During the attack, whenever a new packet is forwarded by the switch to the controller, the controller assumes it is from the DDoS attack address. To limit their service life, the controller creates a new specific input with hard timeout and idle timeout values lower than those of normal inputs. The source IP address is then updated in table T for tracking purposes, and its ci counter is increased by one (Dao et al., 2015).

When the value of C_i reaches k, the average number of packet counters is examined. If s is greater than n, it indicates that the source address sent legitimate data connections, indicating that it is a frequent user. As a result, the control resets the hard timeout and idle timeout to their default values. Furthermore, if s is less than n, the traffic is malicious (Dao et al., 2015). The impact of a DDoS attack is reduced because short values are set for the hard timeout and idle timeout of new packets forwarded from the switch to the controller. Malicious source addresses are canceled by block entries after the controller analyzes the behavior of data traffic. This method can deal with a DDoS attack, but it is ineffective when the amount of traffic is high.

(c) Time-based Method

(Dharma et al., 2015) considers the destination address, the time required to achieve high-speed traffic, and a DDoS attack time attack pattern in his work. The time duration of the attack is used to detect it, and the time attack pattern is used to prevent future DDoS attacks. This method, depicted in figure 2.11, assumes that each packet arriving at the controller is a new packet, and thus a valid destination address will be verified. If this address is invalid or unknown on the network, the packet will be routed to the flow collector. A flow collector is a module implemented in the controller that stores an invalid packet and analyzes it statistically. The flow collector notifies the controller if the accumulation of invalid packets increases significantly over a certain period of time. For each network device, the driver creates a new rule that directs the invalid packet to the flow collector.

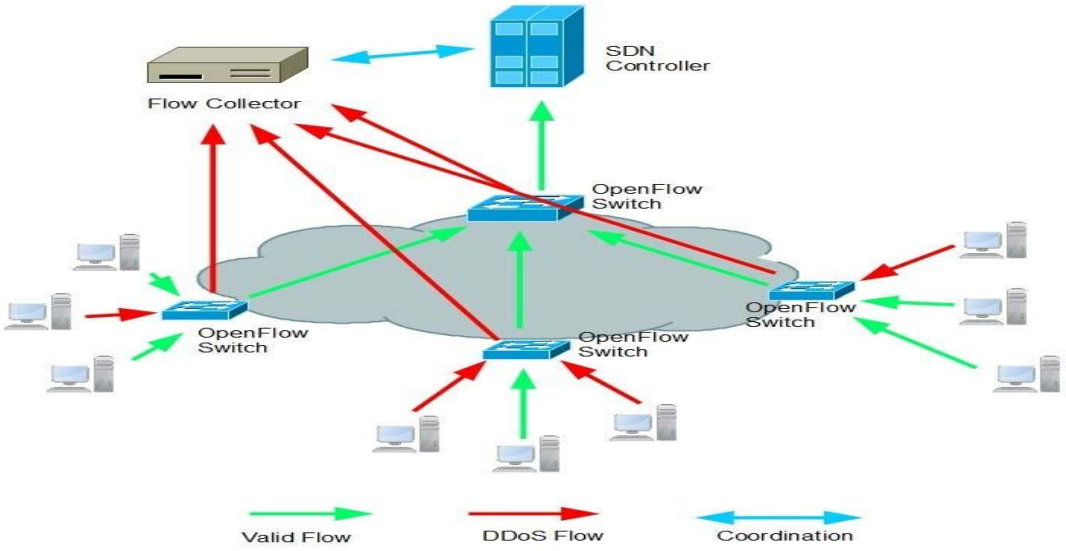

Figure 2.11; Time-Based DDoS Detection Scenario (Dharma et al., 2015).

Table 2.2: Comparison of the Three Suggested Methods.

Criteria /Detection Method	Entropy-Based	User Traffic Analysis	Time-Based
Device where the solution is deployed	Network Controller	Network Controller	Network Controller
Detection Parameter	Destination IP Address	Source IP Address	Destination IP address and time characteristic of the attack
Simulation Scenario	Mininet with nine OpenFlow Switches	Opnet Modeler with an Open-Flow Switch.	Mininet with four OpenFlow Switches.
Additional Driver modules	None	None	Flow Collector
Flexibility in changing detection thresholds	Yes	Yes	Yes
Detection Rate	96%	Not Specified	Not Specified
Effectiveness in a network wide targeted attack	Not Checked	Not Checked	Not Checked
Effectiveness in a multi controller environment	Not Checked	Not Checked	Not Checked

The flow collector is then used to further process the DDoS grouping time pattern at. This pattern will be used to defend against the next DDoS attack. Table 2.2 compares the three methods mentioned above, highlighting the key characteristics of each. The method based on entropy has seen the most development in recent years, as there have been several studies conducted or healed using this technique, and many of its implementations are documented. DDoS attacks, as an ever-present threat, are a real threat to any network, particularly SDN

2.6 DDoS Attack Detection and Mitigation in the SDN Controller

Analyzes various DDoS attacks in an SDN environment based on the severity they impose on SDN entities. This type of analysis will enable researchers to identify the characteristics of severe DDoS attacks and will aid in the development of the best possible DDoS detection scheme. As a result, we examined various existing DDoS attack techniques in the context of the SDN environment. DDoS attackers have become quite sophisticated in recent years, with new types of attacks on traditional networks appearing regularly. Because of SDN's vulnerabilities, it is obvious that it could suffer due to a variety of DDoS attacks.

Despite SDN's centralized control of the entire network to detect and mitigate DDoS attacks, it is still vulnerable to a variety of DDoS attacks that must be addressed. Because control logic is separated from forwarding devices, SDN is vulnerable to a variety of security threats. To secure networks, these threats must be prioritized by researchers and many commercial vendors. Many detection mechanisms have been developed to mitigate DDOS attacks to secure SDN. Based on the possible causes of the threats on the SDN planes, some defense solutions are proposed, which are classified into different categories such as data plane solutions, control plane solutions, switch-controller-based collaborative solutions, collaborative intelligent switches, and integrated solutions (Cui et al., 2021). As well as preventing DDoS attacks.

The controller has been the focus of this security solution. It is the most important component in the structure because it is the driving force of SDN and the operating system. This work is primarily concerned with detecting DDoS threats that endanger the controller, which in turn protects the hosts.

CHAPTER THREE
3. MATERIALS AND METHODS

3.1 Introduction

During a DDoS attack, a high number of packets are sent to a host or a group of hosts on a network. If the source addresses of arriving packets are forged, as is usual, the switch will not detect a match and will have to pass the packet to the controller. By forcing the controller's resources into continuous processing, the accumulation of valid and DDoS faked packets might exhaust them. This renders the controller unavailable for freshly incoming legal packets and may result in the controller failing., resulting in the SDN architecture's weakening. As a result, a quick and effective method of mitigating the attack is required. The randomness of incoming packets is measured here. Entropy, which is used to detect DDoS attacks, is a good measure of randomness.

The aim of the research is to inspire and build an entropy-based technique for packet generation and detection of DDoS flood attacks in SDN, as well as to mitigate DDoS attacks on the SDN controller. To be successful with this implementation, we must examine the requirements for deploying a secure controller as well as the benchmark performance KPIs for normal SDN network operation. We see that security design considerations for SDN-based architectures continue to be critical in the evolution of SDN technology due to security vulnerabilities introduced by centralizing control logic. This will inevitably necessitate a great deal of attention in order to provide stability to the new SDN solution. Hackers continue to target the SDN controller, which necessitates a strong defense against external networks and software-based intrusions.

This chapter will outline the proposed approach for early detection. Due to the controller's limited resources, the attack should be detected within the first few hundred packets. The formulae for entropy, the experimental network topology, and the implementation setups needed for security solutions and computation will be covered. The proposed detection method will then be examined, followed by an assessment of early detection. This section also describes the procedure for dealing

with the issues raised in this study. The following programs were used for this project mainly meanwhile setup details and feature requirements are listed in appendices A1, A2, A3 and A4.

(a) Virtual Box - provides a foundation for the formation of a virtual network.

(b) Mininet - referred to as virtual SDN network topology.

(c) POX is an SDN controller as shown in the proposed model architecture of figure 3.1.

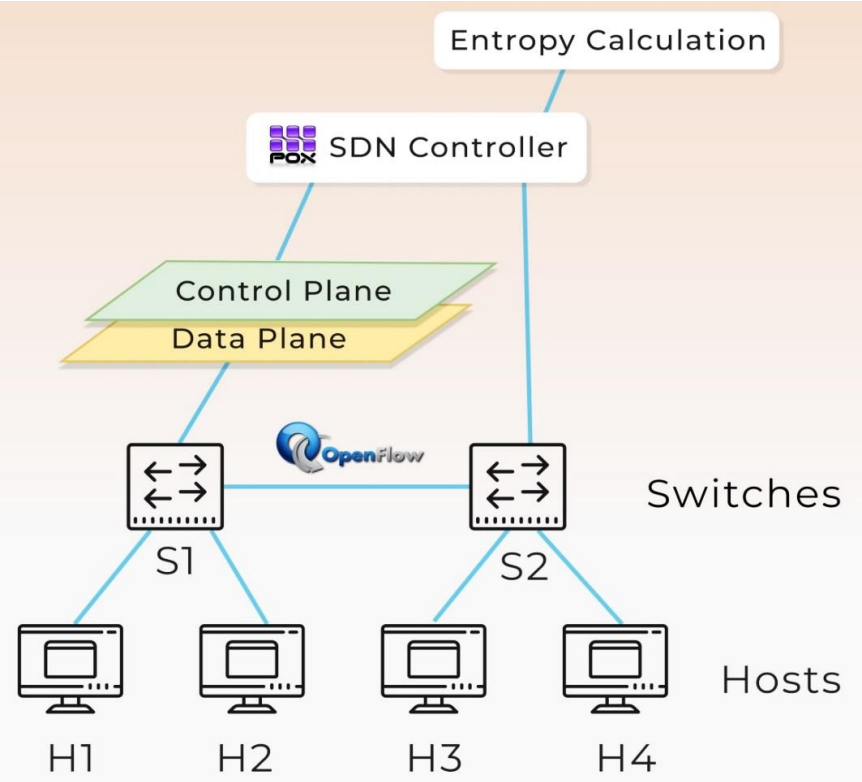

Figure 3.1: Proposed Model Architecture.

3.2 DDoS Detection Using an Entropy-Based Approach

In Chapter 2, we looked at the main components of entropy and how it can be used to detect DDoS. By looking at the formulation and computation in this section. This method examines the level of randomness in the traffic arriving at the controller (each host has an average number of packets it attempts to reach per unit of time). Entropy is a critical concept in information theory. The ability of entropy to measure randomness in a network is the primary reason for using it. The greater the

randomness, the greater the entropy, and vice versa. DDoS detection using entropy requires two components:

(a) A window sizes
(b) A threshold

The size of the window is set by either time or a series of packets. Entropy is calculated inside this timeframe to estimate the uncertainty in the arriving packets. An assault must be detected by a threshold. Depending on the technique, an attack is detected when the estimated entropy exceeds or falls below a threshold. If n is the number of packets in a window and p_i is the probability of each element in the window, then the entropy (H) is calculated using the Shannon Entropy formula of equation (3.3), Any pattern will deviate from this rule and impose a particular order.

A pattern is identified by two factors: Window sizes and threshold The window size, which is determined by a period or the number of packets per window, allows you to quantify incoming traffic. The target parameter is measured in the header field of each window. The threshold is equal to the minimum entropy value measured during normal traffic flow. Entropy is calculated within the window to determine the randomness of the following packets. When the entropy value falls below a certain threshold, it indicates that an attack is underway. Entropy is greatest when all elements have equal probabilities.

When one element appears more frequently than others, the entropy increases. If there is a steady stream of incoming data, it is divided into equal sets known as windows. Each element and its occurrence are counted in the window. When the entropy falls below a certain threshold, we can say that a DDoS attack has occurred. To identify an attack in the controller, check the destination IP address of incoming packets. The controller now contains a method for constructing an incoming packet hash table. When an IP address is entered into the database for the first time, it will be counted as one. If an instance of it exists in the table, its count will be increased. Let W be a data set with n elements and *x* be an event within the set.

$$W = \{(x1, y1), (x2, y2), (x3, y3), \ldots.\} \qquad (3.1)$$

$$p_i = \frac{x_i}{n} \tag{3.2}$$

$$H = -\sum_{i=1}^{n} p_i \log p_i \tag{3.3}$$

The entropy of the window will be calculated after a certain packet. Eq. (3.1) depicts the hash table, where x represents the destination IP address and y represents the number of times it appeared. We use equations (3.1) and (3.2) to estimate the entropy shown in Eq. (3.3), where *W* is the window and pi is the probability of each IP address. Where *x* is the number of events in the set and *n* is the size of the window. We use two factors to calculate entropy for detection in this case.

(a) The destination IP address, and

(b) The number of times it was repeated.

The entropy will be at its highest when each ip address only occurs once. A large number of packets will be directed to a host if an attack is directed at it. These packets will take up the majority of the window, lowering the number of distinct ip addresses in the window and, as a result, entropy. For new packets, entropy will be calculated based on the destination ip address. When there is an attack, the entropy decreases because the ip address of the host or hosts under attack appears more frequently. Because detection takes place within the controller, this work will be customized for the SDN environment.

3.2.1 What is the significance of Entropy?

The source address is always new when packets arrive at the controller. This is why they seek the controller's assistance. Because there is no instance of them in the switch's table, they are passed on to the controller. As demonstrated in Chapter 2, For each new incoming connection, the controller will install a flow in the switch, routing the rest of the incoming packets to the destination without additional processing. As a result, every packet displayed in the controller is completely fresh.

Another known truth about newly arriving packets reaching the controller is that the destination host is in the controller's network. The network is comprised of the

switches and hosts that connect to it. Given that the packet is fresh and its destination is within the network, the quantity of randomness may be defined by computing the entropy based on a window size. The window size is defined as the number of arriving new packets required to compute entropy. When each packet is meant for exactly one host, maximum entropy is achieved. When all packets in a window are bound for the same host, the entropy is at its lowest. SDN is an effective way of detecting DDoS because it can quantify randomness and have minimum and maximum values based on entropy. Whenever a large number of packets attack a single host or a subnet of hosts, the value of entropy can drop.

In the second chapter, we discovered that the two methods required extensive training to detect anomalies. In addition, intrusion detection devices for DDoS detection had to be installed on various network links. These limitations do not apply to entropy.

3.2.2 Window Size

To provide accurate calculations, the window size should be less than or equal to the number of hosts. For this study, it was chosen a window packet of 50. The main reason for selecting 50 is that each host in the network has a limited number of incoming new connections. Once a connection is established in SDN, packets will not pass through the controller unless a new request is made. Additional reason is that each controller is limited to a certain number of switches and hosts. The computation performed for each window is the third reason for selecting this size.

A list of 50 values may be calculated significantly more quickly than a list of 500 values, and an attack can be detected much sooner in a 50-packet window. It is chosen to be 50 because the number of hosts in the test network is less than 100. It is also measured the CPU as well as memory usage tested the entropy with three different window sizes. The threshold is determined by the rate of normal traffic, which is traffic that is not classed as attack traffic. To address queries about the strength of the attack, a simple fraction of attack packets P_a to normal traffic packets P_n equals the rate R as shown in equation (3.4).

$$R = (P_a \div P_n) \times 100\% \quad (3.4)$$

3.3 Functional Requirements for the Proposed Security Solution

This section describes and expands on the specifications and customization options for the tools that will be used in the dissertation's experiments. The details will provide the reader with the requirements for the PoC implementation, as well as the method of setup and configuration that will be used to deliver the proposed solution. The proposed solution will be implemented on the Ubuntu platform using the POX controller. Under a set of specific conditions, the functional requirements define how the proposed solution will work, behave, and deliver a working model. It describes the proposed system features and capabilities of each component of the overall solution in terms of DDoS attack detection and mitigation functionality. The key points of functionality are listed below.

3.3.1 Experiment Environment Setup

The following hardware and software features were used in this experiment on an ASUS laptop. The experimental operating system is based on Ubuntu Linux and runs Ubuntu image files in Oracle VM VirtualBox software. Mininet's version is 2.3.0, and it runs locally on Linux in Appendix A3.

3.3.1.1 Software Systems Requirement

The proposed technology is tested in a simulation environment to ensure its validity. The software to use is Mininet, a widely used tool for simulating SDN architectures. The program runs on an Ubuntu virtual machine with the VirtualBox virtualization software. However, within the software tools used in this project, we have:

- Oracle VM VirtualBox
- Wireshark Packet Analyzer
- Python and VS code
- Secure CRT ® SSH and SFTP tool
- Mininet
- Microsoft Visio 2021
- XD 2022

3.3.1.2 Experimental Setup Server Features

In this work, the most recent version of VirtualBox was used to host a virtual machine with the most recent version of Ubuntu, as this is what has been recommended for running the Mininet simulation software. The information about the virtual machine and its hardware specifications are available in Appendix A4.

3.3.2 Tools and Technologies

This section describes the tools and technologies used to build the test platforms used in the experiments. All of the tools used in the PoC are opensource Linux Ubuntu VirtualBox software. The specific software versions and integration interfaces for all of the tools are further explained, as is the build-up in delivery of a converged secure SDN platform.

3.3.2.1 Packet Generation and Manipulation in Linux Platform

Scapy is in charge of packet generation. It is an extremely effective tool for package generation, scanning, sniffing, attacking, and forgery. Scapy has two parameters such as packet type and packet generation interval. TCP and UDP packets are included in the packet type. UDP packets are used to forge packet source IP addresses. The interval is tailored to the test case. In this experiment, scapy is used to generate network traffic in order to simulate both normal and attack traffic. With scapy, forged IP packets can be generated, masking the identity of the true attacker. Run the scapy program on the host to send packets to the host in the designed network topology, both normal and attack packets (UDP packets and TCP packets).

Scapy is a free and open-source program. Scapy is written in the Python programming language. Scapy is a program that manipulates packets. The Scapy tool forges data packets that come from a source. Scapy decodes and captures data packets. This tool reads packets from pcap files and then compares them to the request and replies. The Scapy tool also performs scanning such as trace-routing and unit tests. You can also use the scapy tool to perform Nmap scanning (Emmons, 2021) .This tool is also very good at a variety of other tasks that most other tools cannot handle, such as sending invalid frames, VLAN hopping+ARP cache poisoning, and VoIP decoding on WEP-protected channels. Because this tool is written in Python, it supports Python 2.7 and Python 3. (3.4 to 3.7). This is a multi-

platform tool that is available for Windows, Linux OSX, and *BSD. The Scapy tool can be used as a shell to interact with network incoming and outgoing traffic (Emmons,2021).

3.3.2.2 POX

It is critical to select an appropriate controller. There are numerous controllers available for researchers and developers at the moment. As shown in Table 3.1, this paper selects some representative controllers and compares their basic information. In this experiment, the Pox controller is used. It is widely used in high speed, light weight experiments. It is intended to be a platform upon which a user-defined controller can be built. Pox is a Python-based opensource controller. Its advantage is that it simplifies the application interface to the controller, allowing it to run in parallel with the controller.

Table 3.1: SDN Controllers.

Controller	Language	OpenFlow	Thread	Released
Ryu	Python	1.0~1.4	single	03/2015
Pox	Python	1.0~2.3	single	10/2013
Nox	C^{++}	1.0 & 1.3	Multiple	02/2014
Beacon	Java	1.0	Multiple	09/2013
Floodlight	Java	1.0	Multiple	11/2012
Opendaylight	Java	1.0 & 1.3	/	03/2015
ONOS	Java	1.0 & 1.3	/	03/2015

Stanford developed the Pox controller, which is based on OpenFlow. Pox is made up of two parts: the kernel and the component. All components communicate with one another through the kernel, which serves as the assembly point for all components. Pox contains an OpenFlow interface for discovering topologies and selecting paths, as well as the ability to customize components to perform specific functions. Pox controller enables rapid development of controller prototype functions and can generate superior performance applications, reducing developer burden and improving development efficiency. Pox is a general-purpose SDN

controller written in Python that supports OpenFlow. It has a high-level SDN API with a representative topology graph and virtualization support.

The Python-based POX SDN controller will be used in the experimental setup for this dissertation because it is regarded as a suitable platform for the implementation of SDN in the academic and research fields. POX is a simple and lightweight design on which several academic studies and prototypes of SDN projects have been carried out. Pox has attracted the attention of many developers and researchers due to its advantages of programming language, clear architecture, good performance, and strong scalability, in addition to the advantages of programming language. For the reasons stated above, the Pox controller was chosen for the simulation experiment in this paper.

3.3.2.3 Mininet

Mininet, an SDN-based network emulation software, is used in the PoC to simulate and create virtual hosts, SDN switches, and a reference controller. Mininet is a vital modeling tool for SDN-based networks that offers information on how the POX controller will react under preset security circumstances. Mininet will connect with the POX controller as a remote controller through a LAN virtual switch generated on the virtualization program VirtualBox, which is installed on an Ubuntu desktop OS virtual machine in the PoC. The external controller is referred to by a remote controller and its associated IP address. Mininet software was chosen because of its flexibility, scalability, and ability to be adjusted for any unique topologies and features. (Kodzai, 2020). It also supports OpenFlow 1.0 or 1.3, an industry accepted standard with multi-vendor controller interoperability, which is essential for constructing SDN-based networks. The Mininet program was created in Python and offers a Python API for building bespoke networks depending on network simulation and prototype needs. (DeCusatis et al., 2016). To analyze the OpenFlow messages used in the connection setup, the software can be easily integrated with packet monitoring software such as Wireshark.

3.3.2.4 OpenFlow Switches

OpenFlow switches are critical components of SDN-based networking because they provide data plane routing and traffic forwarding for hosts. The OpenFlow switches,

which were introduced in Chapter 2, are simulated in the Mininet software and connect to the POX remote controller via the OpenFlow protocol. Flow or group tables in OpenFlow switches perform packet lookups and forward traffic based on reactive or preloaded flow actions for the traffic type in question [14]. Installed traffic rules installed by interrogating the POX controller determine the OpenFlow switch forwarding actions. This happens if it comes across a table miss for no matching flows in the switch rules database. The OpenFlow switch sends a PACKET IN message to the controller when it receives a request. The PACKET IN message encapsulates the original host traffic or TCP packet header information, as well as a buffer ID for the stored packet for future reference. The controller will send a PACKET OUT message to the switch with handling instructions as a FLOW-MOD modification, which includes various actions to forward, flood, or drop the packet (Kodzai, 2020). By referencing the buffer IDs of previously installed actions in the flow tables, the OpenFlow switches can construct group tables with appropriate actions to all future packets using this advanced logic.

3.3.2.5 Oracle VM VirtualBox

VirtualBox is a virtualization tool that enables server or desktop virtualization as well as storage virtualization on the same physical machine. Oracle VM VirtualBox allows virtual machine software to execute directly on the host's CPU, but an array of sophisticated methods is utilized to intercept actions that would interfere with your host. Oracle VM VirtualBox intervenes and takes action when a visitor attempts to do something that might be detrimental to your machine or its data. Oracle VM VirtualBox, in particular, simulates a specific virtual environment based on how you built a virtual machine to access a huge quantity of hardware that the guest believes it is accessing When a visitor seeks to access a hard drive, Oracle VM VirtualBox sends the request to the virtual hard disk that you have configured for the virtual machine. This is often an image file on your host. The utility is capable of running different operating systems, including Linux. and old Windows operating systems, allowing for easy testing and experimental trials of software development. The VirtualBox tool dynamically adjusts the allocated Virtual Machine resources, ensuring efficient use of the physical host machine. The tool also has no additional license requirements, making it superior to other virtualization tools.

3.3.2.6 Ubuntu 20.04.3 LTS

Ubuntu 20.04 is a free and open-source Linux distribution that may be installed on a desktop or a server. The operating system is easy to set up and allows the user to customize and install necessary packages based on their needs. Using the VirtualBox tool mentioned in section 3.3.2.5 of the PoC, we will construct a VM for the POX controller and install the Mininet packet forwarder network using Linux. The basic operating system for our programs is Debian Ubuntu version 20.04. This Linux kernel was chosen because it supports all of the SDN protocols as well as all of the software packages needed for the PoC. Ubuntu 20.04 includes support for the Python application (required by POX), the OpenFlow protocol (virtual switches), hosts (security solution), Mininet, and a variety of other opensource protocols. The Ubuntu software also has no licensing requirements and relatively good support, with documentation on the internet and Linux Debian-based forums, but most importantly, it offers long-term services.

3.3.2.7 Wireshark Packet Analyzer

Wireshark is a network protocol analysis tool that may be used in both real-time and offline mode. Wireshark is an opensource tool that runs on a variety of platforms, including Windows, macOS, Linux, and FreeBSD, and provides deep packet inspection and decryption capabilities on a variety of secure protocols, including UDP, TCP, ICMP, SNMPv3, and In the PoC implementation, DDoS communications between the POX controller and the VM generating the DDoS traffic will be analyzed using Wireshark. It is a highly valuable tool in traffic analysis since it can extract OpenFlow messages created between OpenFlow switches and the POX controller. Wireshark will collect all IP packets created by network interface cards, including WIFI, local area network cards, and virtual interfaces, while operating in the background. The Wireshark utility will employ filters to identify the suitable interface for traffic listening and the related parameters (IP address, protocol, or MAC address) of interest in order to extract relevant traffic while preserving disk space.

3.3.2.8 Mininet Python API and Protocols

The Mininet module serves as the core software for the packet forwarding network simulations in the Proof of Concept. The simulations will be carried out by

configuring and executing a python-based script using the emulator software's built-in python API. The PoC will build the packet forwarding networks using custom Python scripts, making it simple to customize parameters like topology OpenFlow version, remote controller IP address, and OpenFlow port for connection to the POX controller. Due to the various permutations in the various test cases, the custom network option will provide easy deployment, scalability, and easy modifications to network configurations.

The southbound protocol that allows communication between the OVS switches and the POX controller is OpenFlow (Kodzai, 2020). OpenFlow enables controller to controller communication by utilizing the OpenFlow channel established between the switch and the controller. OpenFlow is required in the PoC for SDN network setup to POX controller through the use of the entropy calculation hash table. The Open V switch database (OVSDB) is a programmable switch management protocol that is used in SDN network deployments (Kodzai, 2020). In the experiments, the OVSDB is used in the data forwarding network mostly through OpenFlow.

3.4 Methodological Structure

This section explains how to find solutions to the problems raised by the research. The study area is a controlled simulation environment in which an SDN will be implemented. The Mininet tool will be used to run the tests because it is the industry standard for SDN network virtualization. First, it is able to create a Mininet topology with 9 switches and 64 hosts, as proven within the figure 3.2.

A network is attached to an OpenFlow controller. The entropy of visitors associated with the controller is then measured under regular and attack situations. Because it runs on Python, we used the POX controller for this challenge. A Mininet is a network emulator used to create network topology. Packet technology is achieved with the help of Wireshark, in which Wireshark is used for packet generation, sniffing, scanning, packet forging, and attacking. Wireshark is used to generate UDP packets and spoof the packets' source IP address.

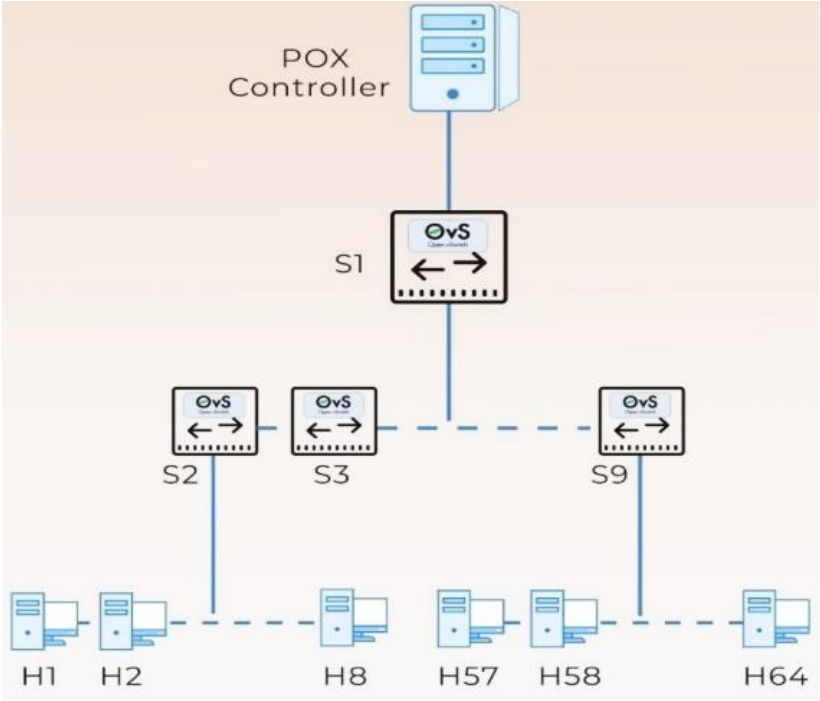

Figure 3.2: Proposed Logical Network Topology.

3.4.1 Design and Type of Research

The quantitative approach was used to address the research problem. The quantitative data obtained from the simulation will be interpreted, and the proposed methodology will be validated.

The following research methods will be used:

(a) Documentary Research: Because everything related to the research environment is investigated to learn more about the subject while also having a broader vision of all the variables that will allow the research and analysis to achieve the proposed goal.

(b) Applied research: Because it seeks to implement or use knowledge from one or more specialized areas to implement it in a practical way to meet specific needs, it provides a solution to social or productive sector problems.

3.4.2 Proposed Flowchart Diagram

Running a lightweight application on the SDN controller helps detect DDoS attempts on a server in an SDN network. The program counts the number of packets flooding into the SDN controller, calculates the packet rate and data packet occurrences based on the destination IP address, TCP port, and source IP address, and determines the entropy of the destination IP address, TCP port, and source IP address (Patel, July 5, 2016) as depicted in the algorithm flow chart of figure 3.3.

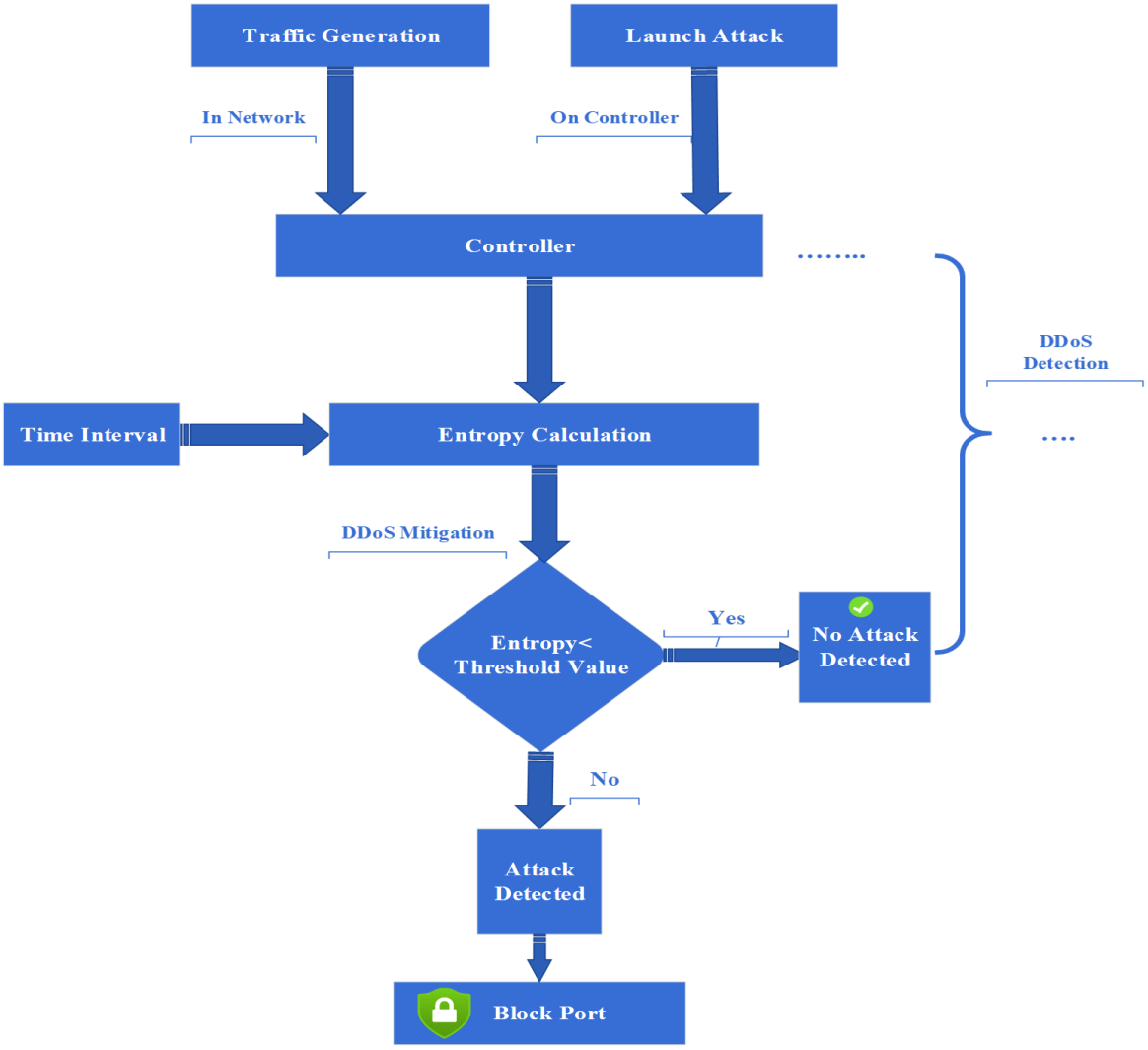

Figure 3.3: DDoS Detection and Mitigation Flowchart .

CHAPTER FOUR

4. RESULTS AND DISCUSSIONS

4.1 Results

4.1.1 POC Implementations

In this chapter, we will go through the tools used in the design process, as well as the integration points for submodules and the deployment of the solution in a virtual environment. The chapter provides an overview of the technologies employed in the control plane infrastructure with the POX controller. In this section, we will discuss the testbed and overall design, which includes the hardware and physical architecture, logical network design, and control to data plane designs.

This section outlines the suggested entropy-based method for mitigating DDoS traffic and the hosts' related targets with the design components with the high-level design and requirements of the discrete components that comprise the prototype creating the security solution under evaluation in this dissertation in the previous chapter. As previously stated, the suggested security solution is based on the SDN controller, which is the primary target of attacks in SDN technology. The suggested technology is implemented in a simulated environment to validate it.

In principle, an OpenFlow controller is linked to a network, and the entropy of traffic associated with the controller is measured under normal and attack conditions. The system module of Python was used in the research to access system-specific parameters and functions.

4.1.2 Logical Network Design for VMs and Physical Layout

This section describes how all of the protocols, tools, and technologies discussed previously are used to deliver a proof-of-concept solution for the dissertation. This section examines how these tools are used to deliver the following key design areas: physical layout design, network logical design, design implementation for DDoS detection and mitigation, and design implementation for interoperability of SDN environment applications.

For the Windows environment, the most recent version of Virtual Box was downloaded. After installing a suitable version of Virtual Box, a Mininet VM image

was downloaded. Mininet can be used to build realistic virtual networks. Mininet allows you to experiment with OpenFlow and Software Defined Networks. Mininet has been used to implement a virtual network topology in order to simulate a real-world environment. Mininet allows for the creation of custom topologies. Because of the power of virtualization, a single system can be made to appear to be a network. The SSH tool is used to access the virtual machine's command line from the physical machine via port 22; its use is not required, but it facilitates the copying of codes and scripts between the two machines.

It includes the physical layout of Ubuntu in VirtualBox, POX controller, SDN Network, and Mininet in the virtualization environment for easy traffic flow interpretation and logic. The specific system requirements and the highest system resources in terms of RAM and CPU used in physical and virtual machines were demonstrated in the previous chapter section 3.3.1.2 and in appendix A4.

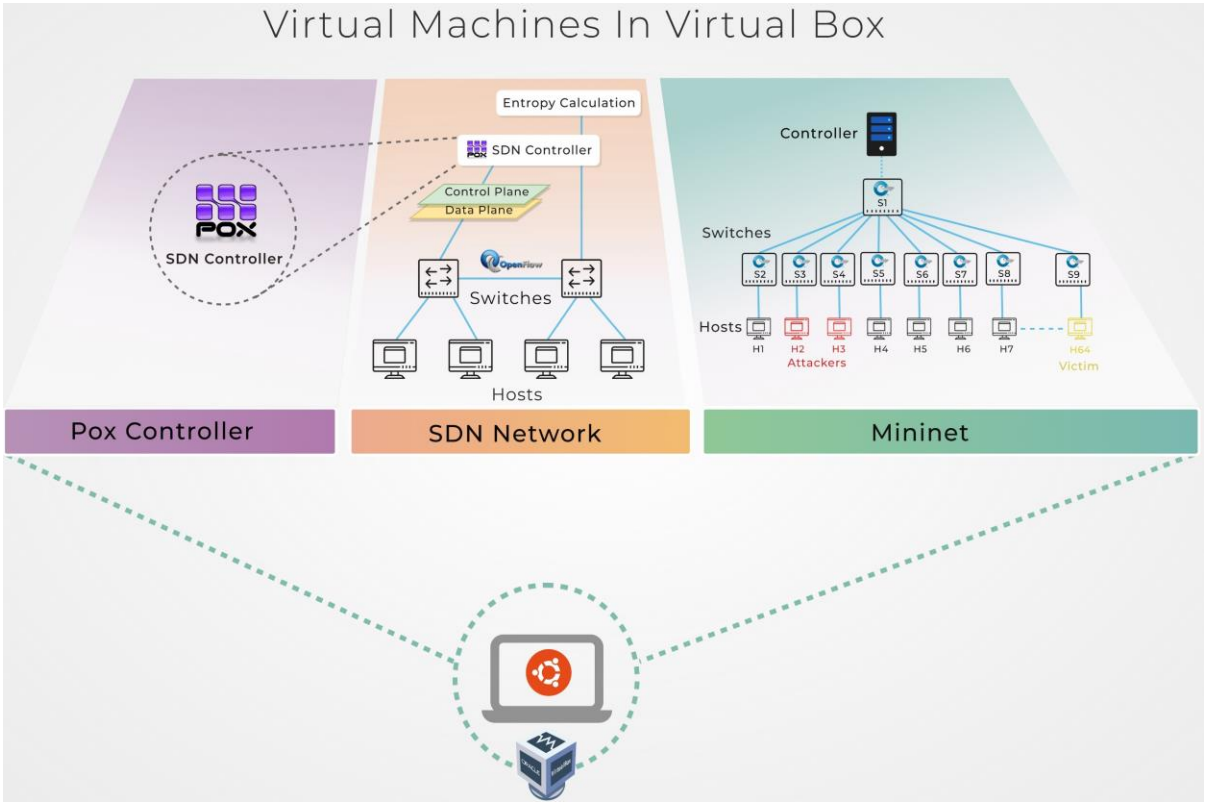

Figure 4.1: Design Setup for Physical and Testbed Layout

4.1.3 Network Topology Modeling

The network topology described in section 3.4, which is a treelike structure, is implemented and used as a reference for the definition of the test scenario This topology was chosen because it is a network structure that is commonly used in data centers and has been used in research related to this work.

It is necessary to set up a network for the test, and the command used to do so is listed below, from within the Mininet environment. By running and entering the following command which is used to generate and create the proposed topology:

$ sudo *mn --switch ovsk --topo tree, depth=2, fanout=8 --controller= remote, ip=192.168.137.16, port=6633.*

This command establishes a network and configures it with controllers, hosts, switches, and links shown in figure 4.2, where a tree-type topology is generated with the following network elements: One controller (c0), nine OVS switches (s1, s2, s3...) and sixty-four hosts (h1, h2, h3...). The respective network links are automatically generated.

```
wendina@VirtualBox:~$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,ip=192.168.137.16
,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 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h
58 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) (s
2, h5) (s2, h6) (s2, h7) (s2, h8) (s3, h9) (s3, h10) (s3, h11) (s3, h12) (s3, h13) (s3, h14) (s3, h15) (s3, h1
6) (s4, h17) (s4, h18) (s4, h19) (s4, h20) (s4, h21) (s4, h22) (s4, h23) (s4, h24) (s5, h25) (s5, h26) (s5, h2
7) (s5, h28) (s5, h29) (s5, h30) (s5, h31) (s5, h32) (s6, h33) (s6, h34) (s6, h35) (s6, h36) (s6, h37) (s6, h3
8) (s6, h39) (s6, h40) (s7, h41) (s7, h42) (s7, h43) (s7, h44) (s7, h45) (s7, h46) (s7, h47) (s7, h48) (s8, h4
9) (s8, h50) (s8, h51) (s8, h52) (s8, h53) (s8, h54) (s8, h55) (s8, h56) (s9, h57) (s9, h58) (s9, h59) (s9, h6
0) (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 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h
58 h59 h60 h61 h62 h63 h64
*** Starting controller
c0
*** Starting 9 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 ...
*** Starting CLI:
mininet>
```

Figure 4.2: Designing the Network Topology.

Mininet can also be used to generate the topology by selecting each of the elements with their respective network links, attackers, and victim hosts. Figure 4.3 depicts

the network topology in Mininet; in this case, a single host has been represented per level 2 switch to facilitate visualization by considering the following table of Mininet launch parameters.

Table 4.1: Mininet Startup Settings.

Parameter Reference	Configuration	Comment
Topology	--topo tree, depth =2	Tree topology
Controller	Controller = remote, ip=192.168.137.16	Remote controller in VM with IP address 192.168.137.16
Southbound Protocol	Openflow13	Connect using OpenFlow version 1.3
Port	Port=6633	Include port address of controller

The open flow switch will be connected to a remote POX controller. The preceding command will generate a topology with 64 hosts linked to 9 switches. Each of the nine switches is built on port 6633 and has an IP address that matches the provided IP address or loopback IP address in the previous statement. Each packet may be examined by the switch, and the source port mapping can be learned. Following that, the port will be assigned the source MAC address. If the packet's destination is already connected with a port, the packet will be forwarded to that port; otherwise, it will be flooded on all ports of the switch. The data path id identifies each open flow switch. The switch can discard packets on a flow, either immediately or completely, based on the controller's commands. Packet dropping can be used to protect against DDoS attacks. POX establishes a methodology for interacting with SDN switches. The OpenFlow protocol was employed in this case. POX must be launched from the command line, and a layer 3 switch has been installed.

As described in section 3.3.2.2, the POX controller was chosen from among the available controllers for this work. The pox driver is installed automatically with the Mininet tool in this project, the same virtual machine as the OpenFlow driver is used, though it is also possible to run it from a different virtual machine. Figure 4.3

illustrates the network components in MiniEdit that are used to represent network nodes.

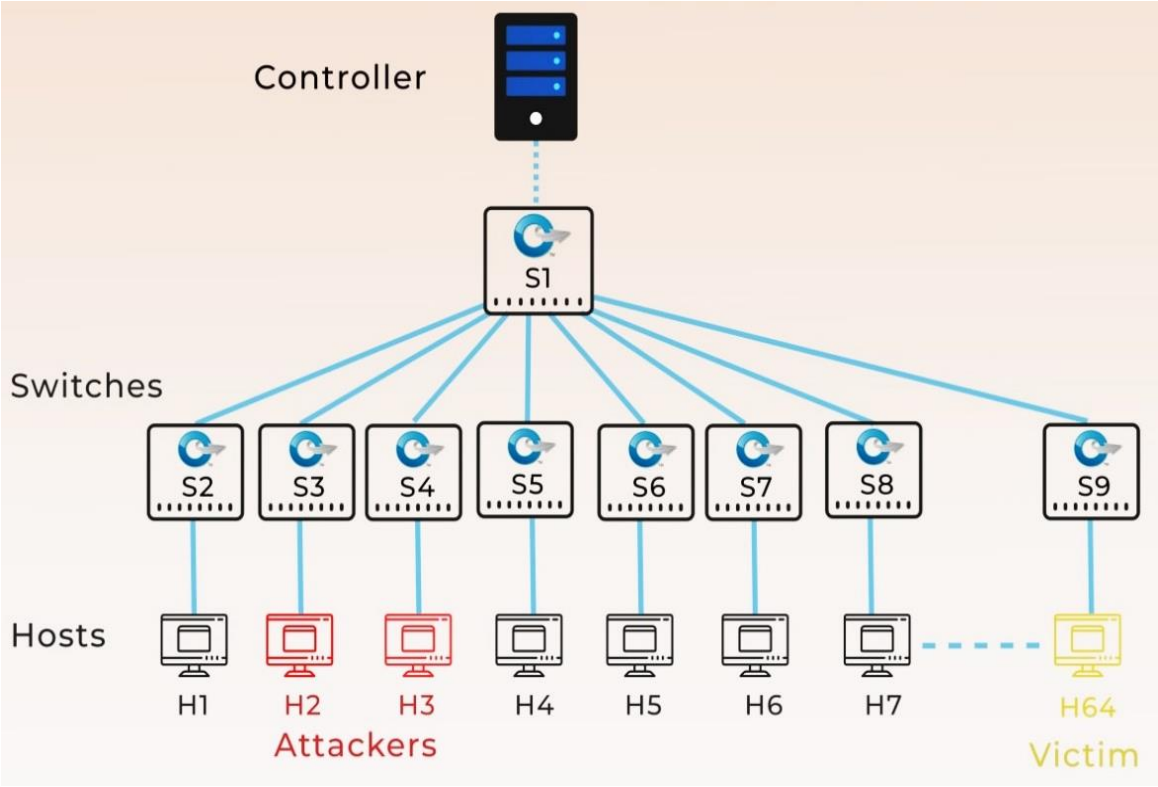

Figure 4.3: MiniEdit Nodes.

The default driver is configured with the IP address 10.1.1.1 and port 6634, so it is necessary to configure it with the IP address of the machine that you want to act as the controller. This is configured by using the command:

wendina@virtualbox: ~$. */pox.py openflow. of_01 --address=192.168.137.16 - - port=6633*

Then it is necessary to activate by the command line to the POX controller with the help of the command:

wendina@virtualbox: ~$ *python3./pox/pox.py forwarding. l3_editing*

Figure 4.4 demonstrates the successful activation of the controller with its respective version, as well as a summary of the switches connected to it. There can be no network convergence if this step is skipped.

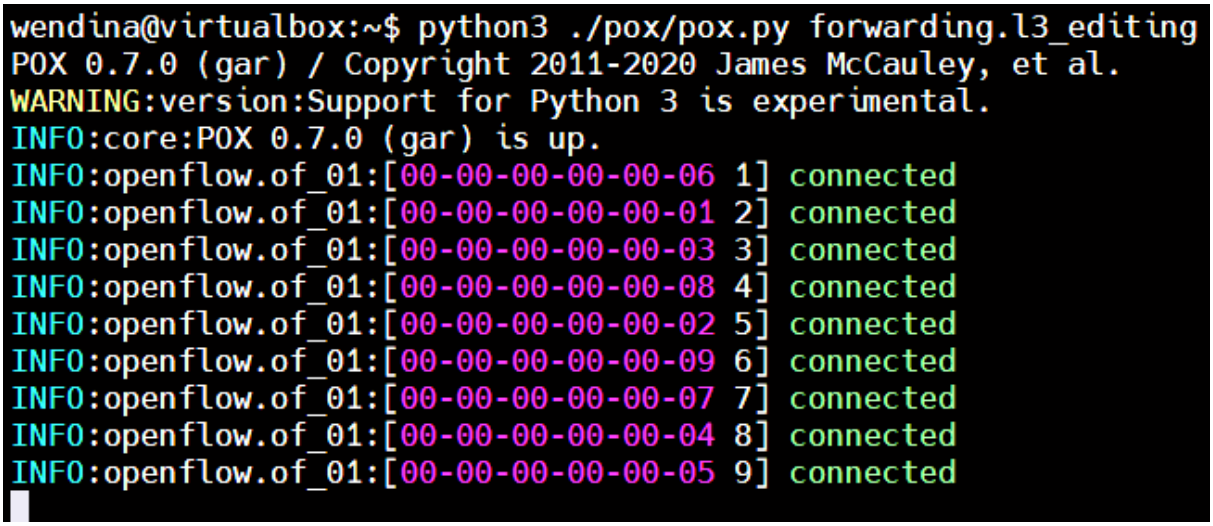

Figure 4.4 : Activating the POX Controller.

Figure 4.5 is the result of the *pingall* command executed on the Mininet command line, it is observed that all packets have been successfully received denoting the convergence of the network. This command is a simple way to check connectivity between hosts on the network, although it is also possible to use the *ping* command from each terminal. Start the Pox controller and design a Mininet topology to check cross host connectivity.

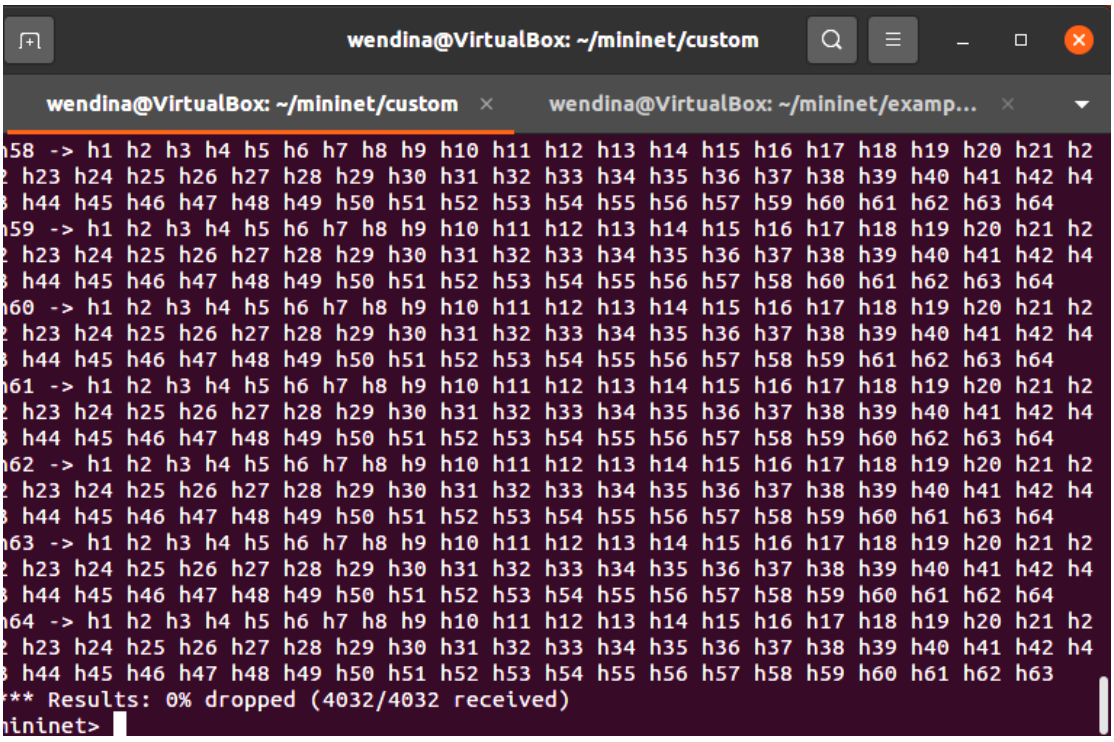

Figure 4.5 : Cross-Host Connectivity Check.

4.2 Validation of Virtualizations and Test Results

The preceding chapter described the different tools, protocols, and technologies necessary to build the proof-of-concept model for the proposed entropy-based network security solution for the POX controller. This section will assess the security solution's efficacy and efficiency against DDoS assaults, as well as the security solution's influence on controller performance. The efficacy of the security solution is determined by determining whether or not the mitigation of DDoS packets from a DDoS producing server was successful. The user must specify the range of IP addresses for the destination IP address. For example, if the user sends the number from 1 to 256. The IP addresses will then be produced in the format 10.0.0.D, where D will be between the start and end values supplied. A function will generate the source IP address at random. The packets will be delivered across the network using the Scapy python module after the source and destination IP addresses have been set. The SDN network was set up in each test, the communication between elements was validated, and the POX performance results were recorded and captured for analysis.

4.2.1 Experimental Evaluation Scenarios for Traffic Generation

Scapy is the program used to produce both legitimate and malicious traffic. This application may be launched from a terminal window as well as by executing scripts. For this project, two scripts have been written: one for normal or legitimate traffic flows and one for attack flows. UDP is the packet type used for both normal and attack communication. Mininet successively allocates IP addresses to hosts beginning with 10.0.0.1. The destination IP address is created based on the range given in the script (10. 0.0.1 - 10.0.0.64) for normal traffic, while the source IP address is generated randomly using the randrange random function. Figures 4.6 and 4.7 show the generation of normal traffic by implementing a script from a host.

4.2.1.1 Launch Traffic and Verification

The inputs '2' and '65' are passed to the generate destination ip function, which produces the destination IP addresses. The launch traffic function, in principle produces random source addresses and sends packets to random destinations

between the hosts. Generally, supply the start and end values from one of the node's command prompts, for example, h1, h64.

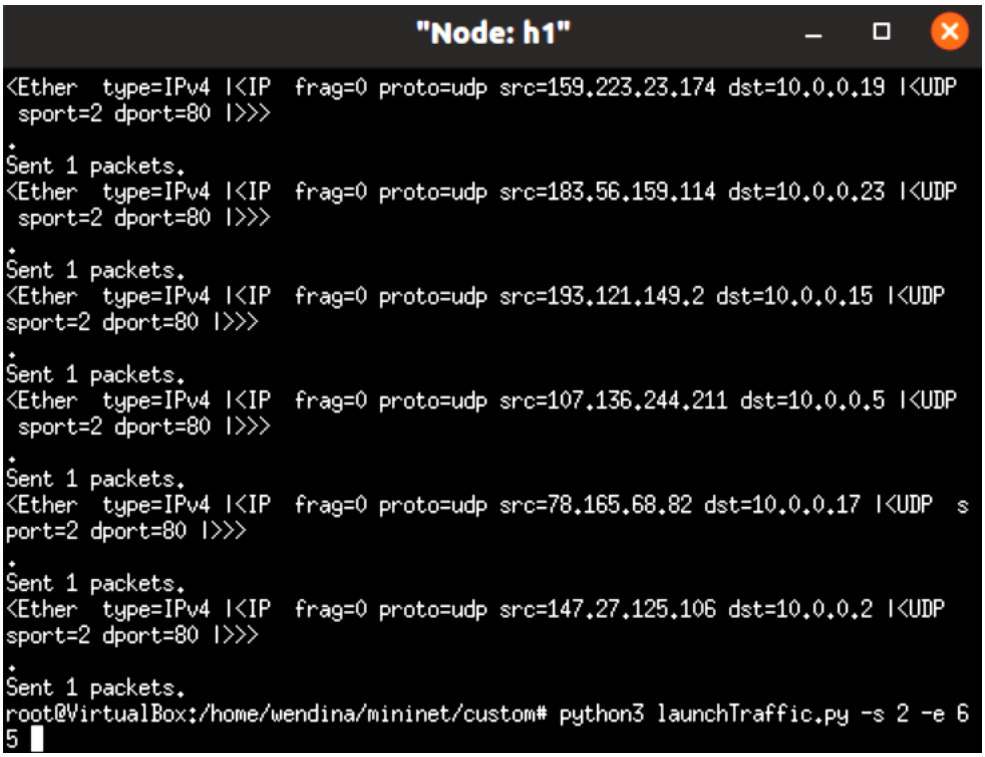

Figure 4.6: Legitimate Traffic Generation for h1.

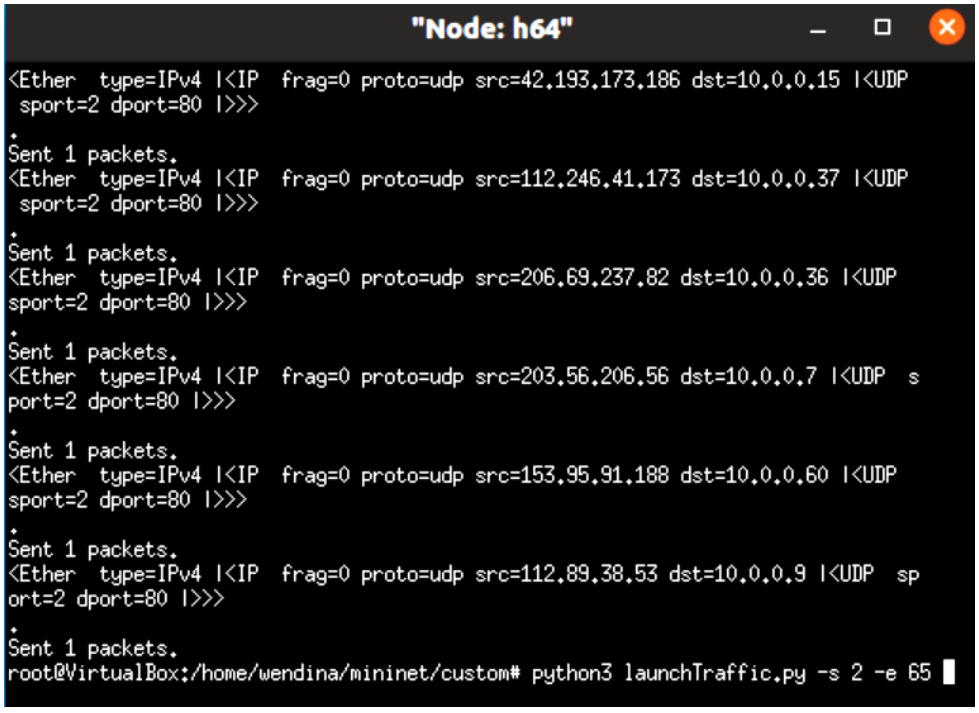

Figure 4.7: Legitimate Traffic Generation for h64.

Wireshark was chosen as a tool for visualizing the packets. Using Wireshark, the loopback interface was monitored prior to any communication on the loopback to achieve this status. Figure 4.8 displays the capturing of legitimate traffic from another network and the data was captured using I/O graphs in Wireshark.

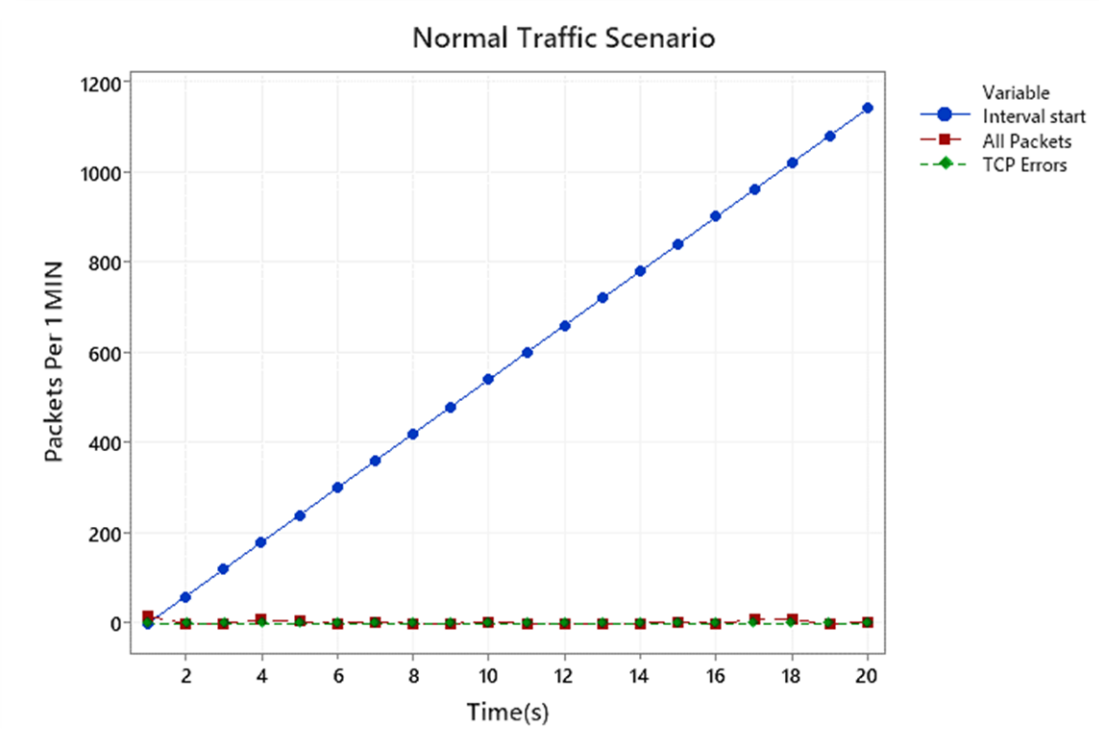

Figure 4.8: Legitimate Traffic Scenario.

4.2.1.2 Launch Attack and Verification

In this case, initiate the attack from a few selected hosts to a single target. The controller's entropy value is now decreasing. In this section, it provides the target IP address from the command prompt of the hosts that are working as a botnet. To carry out the attack on a specific host, the packet generation script must be run using random source ip addresses. Consider the entropy numbers in the pox controller. The number is lower than the previously specified normal traffic thresholds. As a result, the attack may be detected within the first 250 packets of malicious data targeted at a given host in the SDN network. When performing the command, the victim's IP address is supplied, as illustrated in figures 4.9 ,4.10 and 4.11. Repeating the steps in subsection 4.2.1.1 on hosts h1 and h64 while simultaneously providing the following syntax to execute the attack traffic from h1, h2,h3 in xterm windows

respectively to attack on 10.0.0.2 and 10.0.0.64 that attack packets are going to be sent to this destination IP address. The primary purpose of this module is to demonstrate how a malicious user might attempt to attack a network resource.

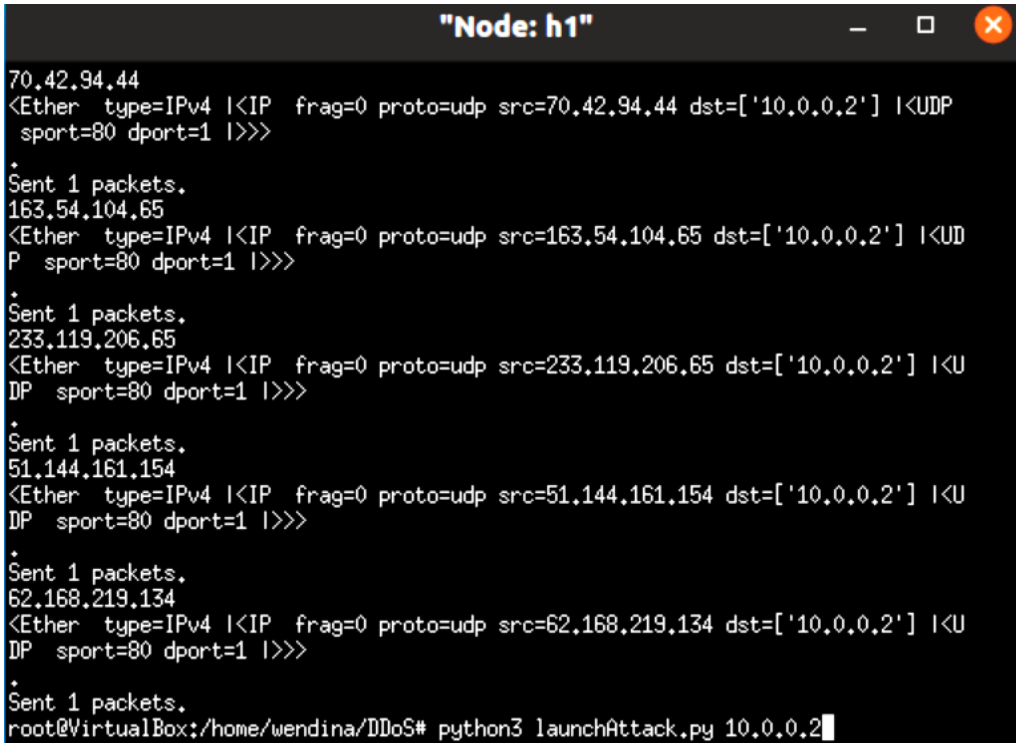

Figure 4.9: Attack Traffic Generation on h1 to 10.0.0.2.

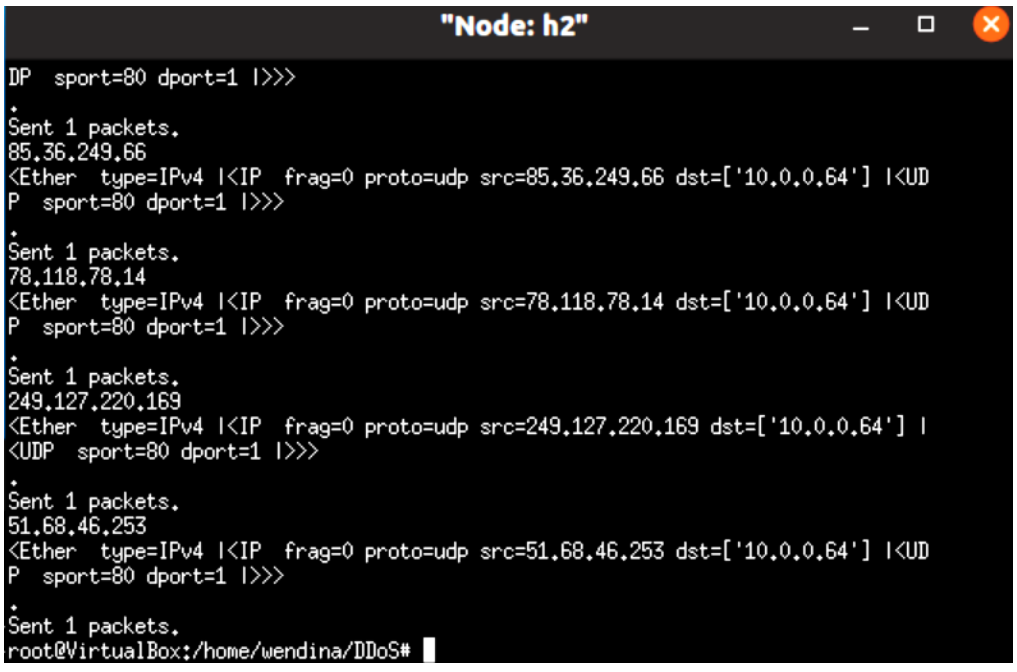

Figure 4.10: Attack Traffic Generation on h2 to 10.0.0.64.

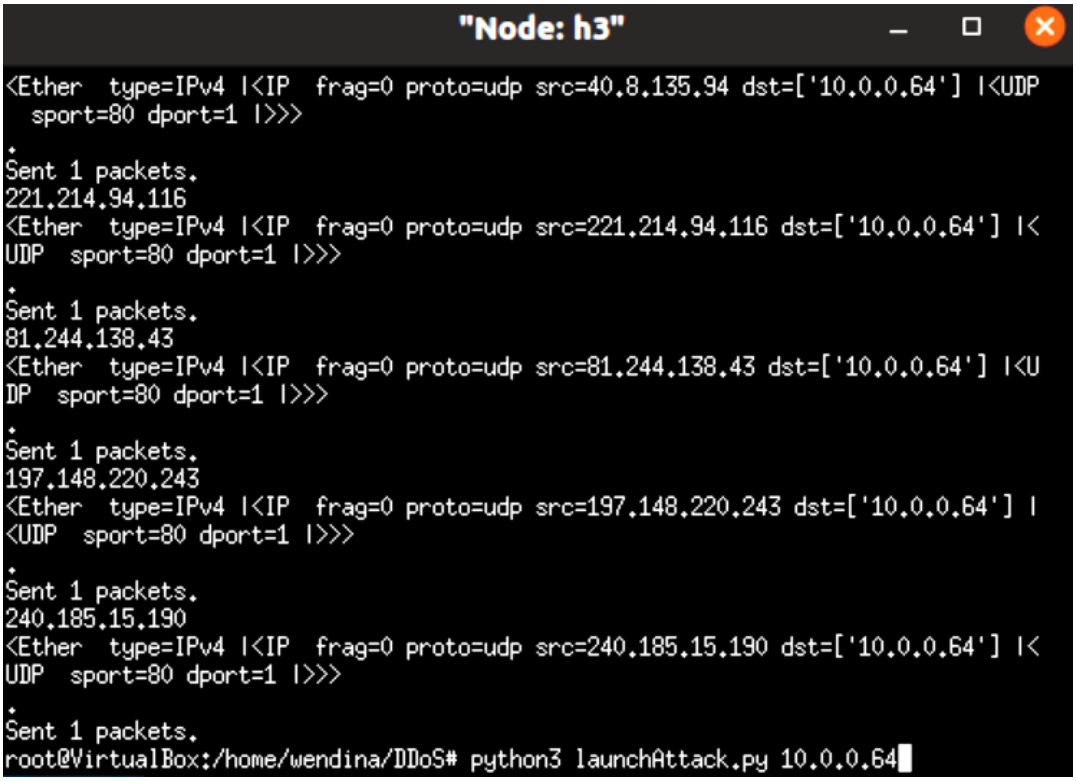

Figure 4.11: Attack Traffic Generation on h3 to 10.0.0.64.

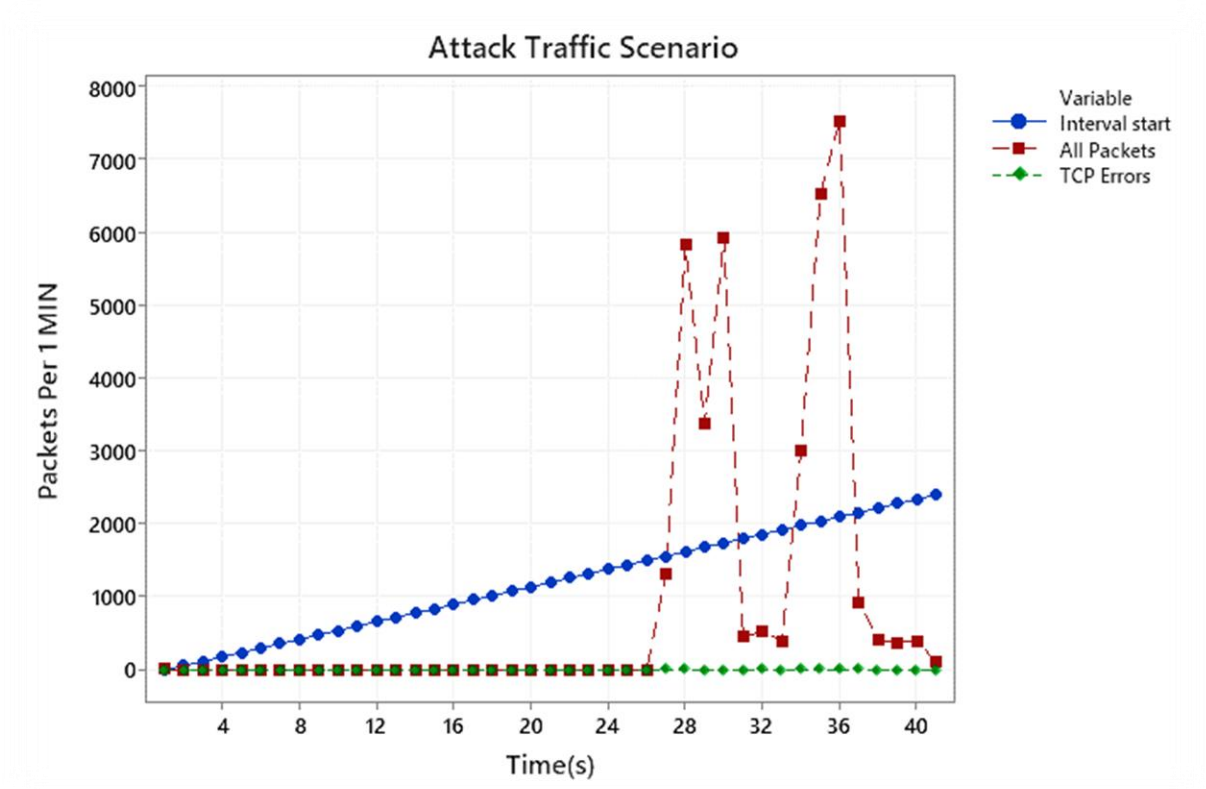

Figure 4.12: Attack Traffic Scenario.

The capturing of attack traffic from another host and the data generated was interpreted using I/O graphs in Wireshark in similar manner to legitimate traffic which was discussed previously. The graph above depicts network traffic over time. Only one host was utilized to produce attack traffic over here. The amplitude on the y-axis would have been greater if more than one host had been used and the attack TCP packets directed towards the host with address 10.0.0.64. The source addresses are random and correspond to the ranges set in the script.

4.3 Deploy the Solution in the Test Environment

The simulation of the network along with the DDoS attacks was detailed in sections 4.1 and 4.2. The chosen solution requires only the controller because the detection and mitigation of DDoS attacks is done by programming it. The detection of DDoS threats using the Entropy value before and after the DDoS attack may be evaluated. The detection module is responsible for calculating the entropy of the network, which is greater than one because there is only normal traffic (see Figure 4.13) and takes values less than 0.5 when there is an attack in progress (see Figure 4.14).

```
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.19650210303
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.23048150312
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.2644609032
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.29844030329
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.33241970338
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.36639910346
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.43971002844
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.47368942853
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.50766882861
INFO:forwarding.detection:{IPAddr('10.0.0.36'): 1, IPAddr('10.0.0.44'): 1, IPAddr('10.0.0.52'): 1
, IPAddr('10.0.0.64'): 1, IPAddr('10.0.0.5'): 1, IPAddr('10.0.0.6'): 2, IPAddr('10.0.0.14'): 1, I
PAddr('10.0.0.25'): 1, IPAddr('10.0.0.3'): 1, IPAddr('10.0.0.50'): 1, IPAddr('10.0.0.62'): 1, IPA
ddr('10.0.0.8'): 3, IPAddr('10.0.0.11'): 2, IPAddr('10.0.0.51'): 3, IPAddr('10.0.0.59'): 3, IPAdd
r('10.0.0.43'): 1, IPAddr('10.0.0.32'): 1, IPAddr('10.0.0.40'): 1, IPAddr('10.0.0.48'): 1, IPAddr
('10.0.0.60'): 1, IPAddr('10.0.0.54'): 1, IPAddr('10.0.0.2'): 2, IPAddr('10.0.0.10'): 1, IPAddr('
10.0.0.18'): 2, IPAddr('10.0.0.21'): 3, IPAddr('10.0.0.61'): 1, IPAddr('10.0.0.55'): 1, IPAddr('1
0.0.0.34'): 1, IPAddr('10.0.0.15'): 1, IPAddr('10.0.0.46'): 1, IPAddr('10.0.0.30'): 1, IPAddr('10
.0.0.58'): 1, IPAddr('10.0.0.20'): 1, IPAddr('10.0.0.7'): 3, IPAddr('10.0.0.41'): 1, IPAddr('10.0
.0.22'): 1}

***** Entropy Value =  1.50766882861 *****
```

Figure 4.13: Entropy with Normal Traffic.

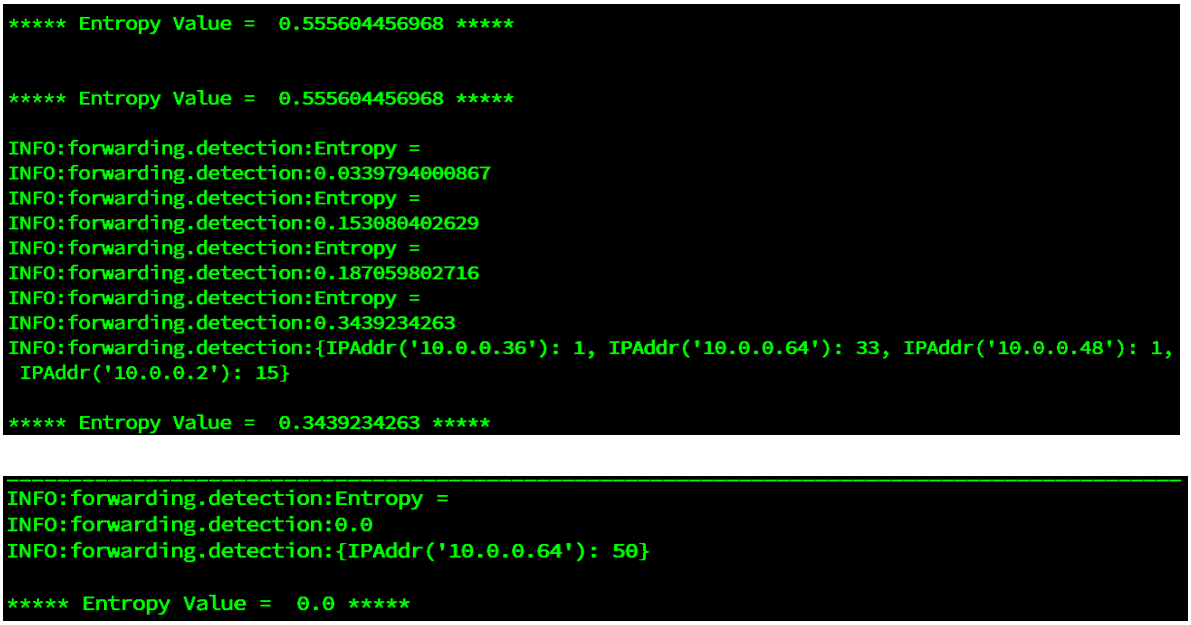

Figure 4.14: Entropy with an Ongoing DDoS Attack.

Analyzing the POX controller's entropy levels. The number falls below the threshold for normal traffic (0.5 in this case). As a consequence, it may identify an attack within the first 250 packets of malicious traffic detected by a host in the SDN network.

4.3.1 DDoS Attacks Mitigation Functionality Verification and Measurements

This section analyzes one of the outcomes of the following sequential phases such as risk identification, planning, mechanism selection, testing, implementation monitoring, and improvement. The main goal is to carry out the installation of the chosen solution into the organization's infrastructure. Thus, technical compliance with the detection and mitigation of DDoS attacks in SDN environments based on the entropy standard is believed to enhance control plane security. The approach of this work, because it was specified in the scope that only a controlled simulation scenario would be employed. In a real case, the implementation would be done in the SDN controller by developing a detection module and altering the POX controller's l3 learning.py file in the same way that it was done in the test scenario. It was created based on what was done in the simulation scenario, explaining the configurations done on the SDN controller that might be used in a real controller.

Figure 4.15 shows the detection and mitigation of the attack on the SDN controller, this scenario corresponds to an attack on host 64 from hosts 2 and 3, while host1 generates legitimate traffic. When the entropy value decreases from the set limit (0.5), the controller displays a message indicating that an attack is in progress and proceeds to block the switch port from which the attack comes, in this case port 1 of the switch 2.

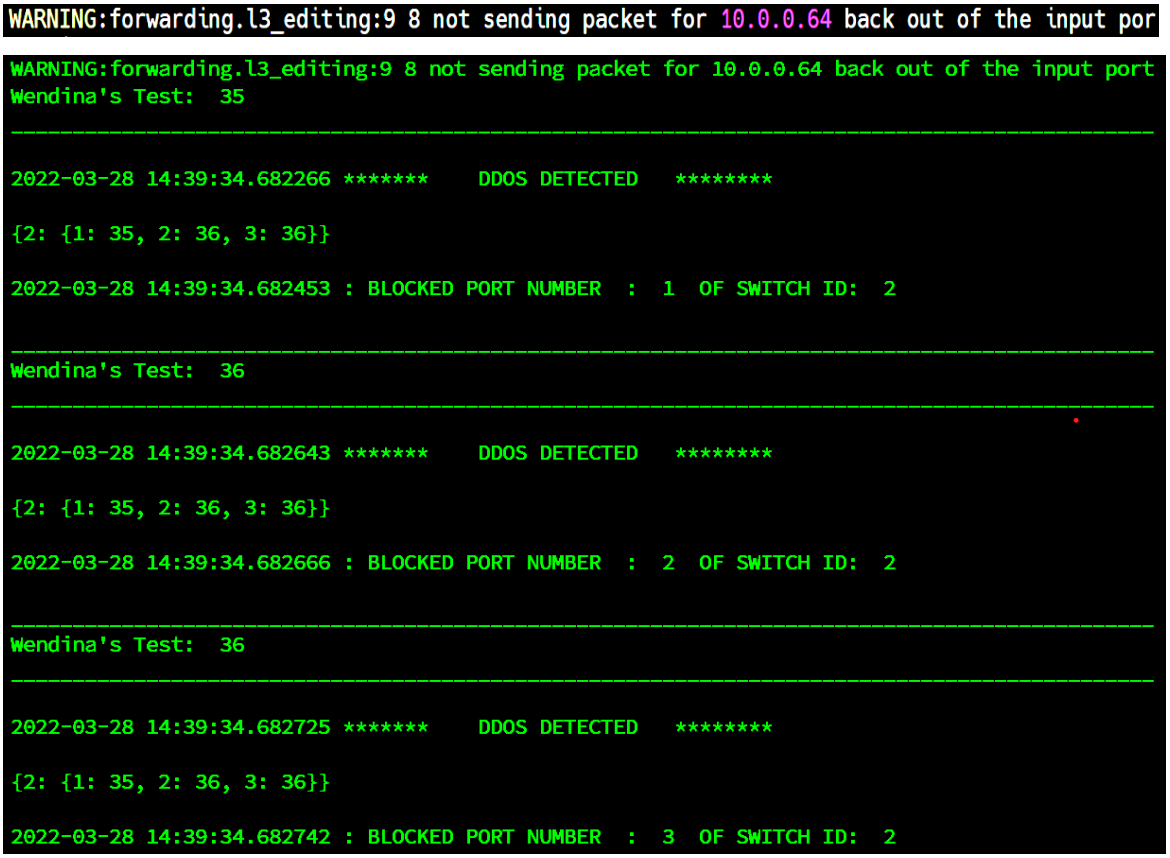

Figure 4.15 :DDoS Attack Detection and Mitigation.

It can be seen that port number 2 of switch 2 has been blocked to prevent further harm from being caused by the attack. Looking through the dictionary, you'll note that port 1 of switch 2 has been activated 35 times. Switch 2's similar port 3 has been activated 36 times. These are direct signs of an attack, and the application will block these ports if it detects that the network is under attack.

Analysis of DDoS mitigation impact on the POX controller and attack traffic with prevention using proposed algorithm that detects the attack network and can be

compared with its equivalents resulted as shown in the following figures of 4.16 and 4.17 of entropy algorithm and association algorithm respectively.

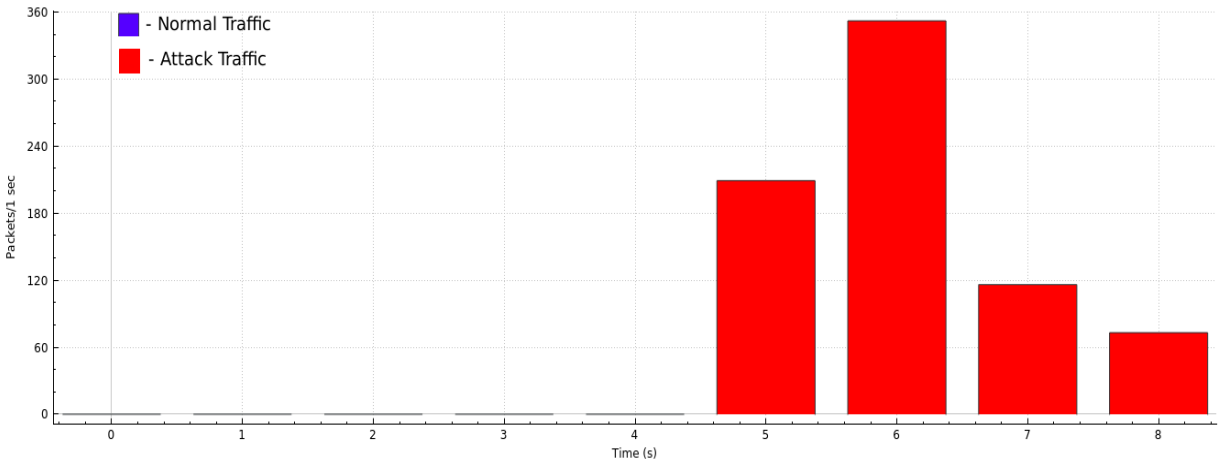

Figure 4.16: Attack Traffic with Mitigation, Entropy Algorithm.

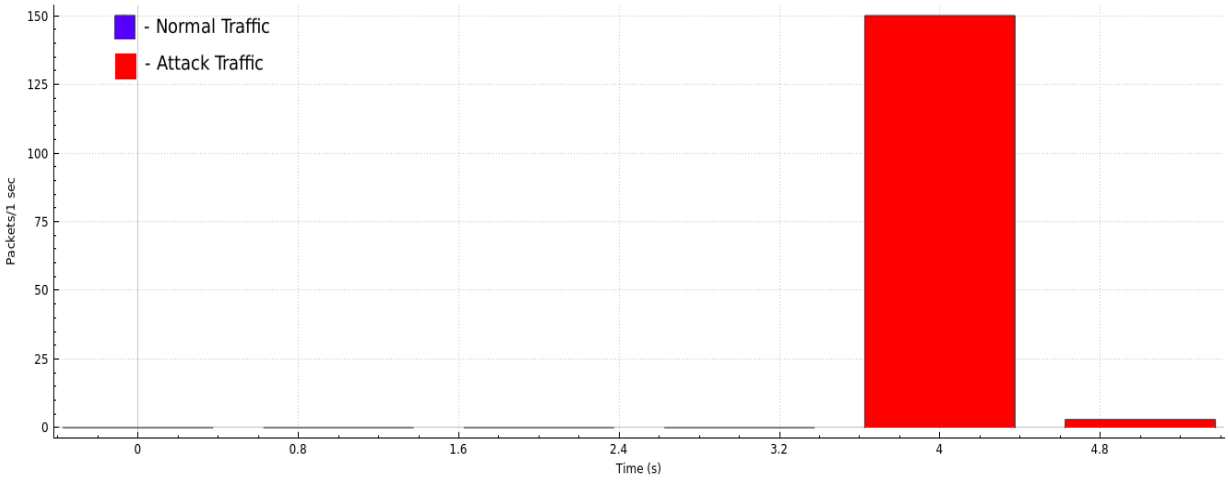

Figure 4.17: Attack Traffic with Mitigation, Association Algorithm.

Figure 4.18 shows the measurement of entropy in the scenario posed during the test. Initially without current traffic in the network the entropy had the value of 1, when generating legitimate traffic, the entropy amounted to 1.5. At the time of generating the DDoS attack towards host 64 the value of the entropy dropped to 0.4. The graph shows that from the time the attack is generated (4.5s) until it is detected (6.5s) approximately two seconds pass, this being an early detection. it is clear that the

attack traffic with the mitigation approach applied in this research can achieve a greater entropy decrease rate and has the function of quick detection.

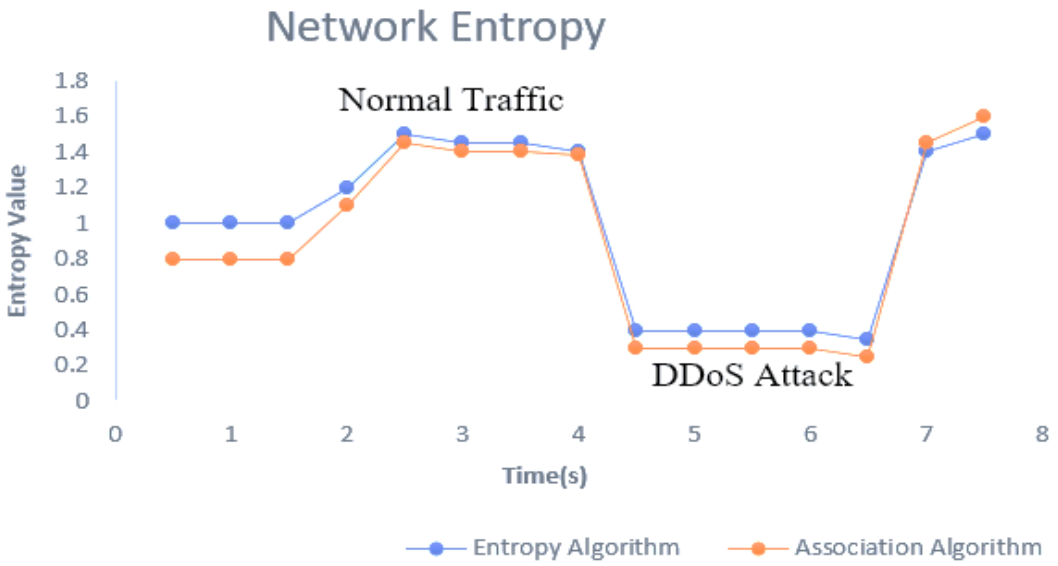

Figure 4.18 : Entropy Variations and Measurement.

When one host receives much more packets than the other, the entropy value decreases. As a result, the entropy falls below the prior limit of one. The decrease in entropy value shows inconsistency in network traffic. This dictionary is generated only when the entropy value is less than 0.5. The essential values in our Mininet topology correlate to the switches. The keys in the value dictionaries correspond to the needed switch's ports. The final value is the number of times the ports have been activated. This is another reliable sign of unexpected network behavior that could indicate the existence of a DDoS attack. It was designed to track detection time since it is the factor that influences whether detection and mitigation are carried out early in the attack.

4.3.2 Methods and Parameters for Monitoring the Attack

If this were a real-world situation, the monitoring would be carried out within the time frame given in the planning, observing the performance of the developed solution against real-world attacks on the controller. Because this was a controller test scenario, the monitoring was done by producing various forms of TCP, UDP, and ICMP attack traffic.

Reviewing the logs generated in the controller interface produced the detection time. The network administrator was described as the person in charge of doing the monitoring and interpreting the data. Monitoring results against different TCP, UDP, and ICMP flood DDoS attacks were recorded. TCP protocol synchronization request packets (SYN) with random source IP addresses and a specified destination (victim) IP address were generated for the TCP Flood attack, also known as SYN Flood. A Wireshark capture of the packets generated for this attack is shown in figure 4.19.

No.	Time	Source	Destination	Protocol	Length	Info
798	38.148825063	72.167.139.21	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
799	38.180567964	13.117.90.128	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
800	38.215500797	114.85.113.143	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
801	38.251180729	109.138.220.113	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
802	38.283002108	99.233.85.32	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
803	38.312817793	247.102.172.141	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
804	38.343965763	67.135.204.24	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
805	38.372318285	150.213.114.199	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
806	38.401875790	34.224.66.135	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
807	38.432022720	84.21.62.188	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
808	38.460469362	182.6.177.166	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
809	38.494801176	160.164.105.193	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
810	38.527416739	178.131.191.3	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
811	38.574467185	162.169.155.18	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
812	38.605612586	51.235.149.81	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
813	38.636257175	80.231.131.157	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
814	38.667898869	176.216.187.193	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
815	38.696431697	217.13.101.180	10.0.0.64	TCP	54	2 → 80 [SYN] Seq
816	38.726104888	119.229.236.39	10.0.0.64	TCP	54	2 → 80 [SYN] Seq

Figure 4.19: TCP Flood Attack.

No.	Time	Source	Destination	Protocol	Length	Info
2293	162.941497807	183.180.130.218	10.0.0.64	UDP	42	53 → 53 L
2294	162.971582798	82.207.134.79	10.0.0.64	UDP	42	53 → 53 L
2295	163.002777052	141.92.82.36	10.0.0.64	UDP	42	53 → 53 L
2296	163.033778409	84.3.42.86	10.0.0.64	UDP	42	53 → 53 L
2297	163.064299358	153.243.108.129	10.0.0.64	UDP	42	53 → 53 L
2298	163.098929539	180.194.11.36	10.0.0.64	UDP	42	53 → 53 L
2299	163.128455258	92.67.159.113	10.0.0.64	UDP	42	53 → 53 L
2300	163.159855541	233.14.254.64	10.0.0.64	UDP	42	53 → 53 L
2301	163.191294437	129.17.84.221	10.0.0.64	UDP	42	53 → 53 L
2302	163.222926918	128.151.220.153	10.0.0.64	UDP	42	53 → 53 L
2303	163.251258935	120.112.150.175	10.0.0.64	UDP	42	53 → 53 L
2304	163.282579771	6.144.168.173	10.0.0.64	UDP	42	53 → 53 L
2305	163.314115539	101.144.61.132	10.0.0.64	UDP	42	53 → 53 L
2306	163.344757167	20.212.10.121	10.0.0.64	UDP	42	53 → 53 L
2307	163.383168217	130.213.63.61	10.0.0.64	UDP	42	53 → 53 L
2308	163.411537072	20.155.46.190	10.0.0.64	UDP	42	53 → 53 L
2309	163.442778110	250.75.174.246	10.0.0.64	UDP	42	53 → 53 L
2310	163.471178632	221.227.95.54	10.0.0.64	UDP	42	53 → 53 L
2311	163.500677685	163.95.194.252	10.0.0.64	UDP	42	53 → 53 L

Figure 4.20: UDP Flood Attack.

The UDP flood attack was created in the same way as the TCP Flood attack was, except that the UDP protocol was utilized and tested with different ports of destination that correspond to typical services that handle this protocol, such as port

53/UDP (DNS). Figure 4.20 demonstrates a Wireshark capture of the packets transmitted in this attack.

No.	Time	Source	Destination	Protocol	Length	Info
3158	213.347800863	131.184.98.155	10.0.0.64	ICMP	42	Echo (ping) request id=
3159	213.376523621	156.25.62.214	10.0.0.64	ICMP	42	Echo (ping) request id=
3160	213.427846024	245.139.1.70	10.0.0.64	ICMP	42	Echo (ping) request id=
3161	213.456659048	191.87.115.160	10.0.0.64	ICMP	42	Echo (ping) request id=
3162	213.495504959	31.19.227.255	10.0.0.64	ICMP	42	Echo (ping) request id=
3163	213.527504432	123.107.242.238	10.0.0.64	ICMP	42	Echo (ping) request id=
3164	213.557278202	27.70.199.91	10.0.0.64	ICMP	42	Echo (ping) request id=
3165	213.589692285	167.203.56.243	10.0.0.64	ICMP	42	Echo (ping) request id=
3166	213.622753353	126.28.81.23	10.0.0.64	ICMP	42	Echo (ping) request id=
3167	213.652716765	15.65.117.44	10.0.0.64	ICMP	42	Echo (ping) request id=
3168	213.683801724	128.12.232.86	10.0.0.64	ICMP	42	Echo (ping) request id=
3169	213.723062401	218.247.143.181	10.0.0.64	ICMP	42	Echo (ping) request id=
3170	213.757689331	52.25.73.62	10.0.0.64	ICMP	42	Echo (ping) request id=
3171	213.790915129	40.70.81.28	10.0.0.64	ICMP	42	Echo (ping) request id=
3172	213.868447264	81.189.176.231	10.0.0.64	ICMP	42	Echo (ping) request id=
3173	213.905246615	201.193.212.83	10.0.0.64	ICMP	42	Echo (ping) request id=
3174	213.936934550	155.210.236.67	10.0.0.64	ICMP	42	Echo (ping) request id=
3175	213.973817691	87.37.64.194	10.0.0.64	ICMP	42	Echo (ping) request id=
3176	214.004911812	235.66.119.90	10.0.0.64	ICMP	42	Echo (ping) request id=

Figure 4.21 : ICMP Flood Attack.

For the ICMP Flood attack, Echo request packets were generated, which use the ICMP protocol. Wireshark's capture of the packets generated in this attack is shown in figure 4.21.

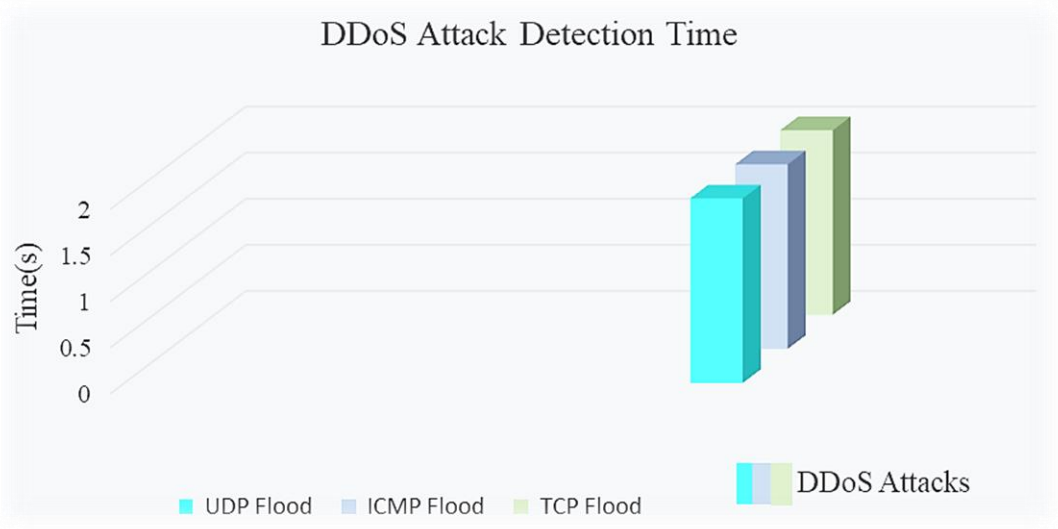

Figure 4.22 : Detection Time of DDoS Attacks (TCP, UDP and ICMP Flood).

The above figure 4.22 indicates that the detection of DDoS attacks using the implemented mechanism is independent of the packet type, since the same is obtained detection time (2s) for all three types of simulated attacks. This is because the mechanism used is based on the destination IP address to measure the entropy of

the network. SYN Flood is one of the most common DDoS attacks are observed and it Occurs when an attacker sends a succession of Transmission Control Protocol (TCP) synchronization requests (SYN) to the target. A UDP flood attack is similar to a SYN Flood in that a botnet is used to send a large amount of traffic to the destination server.

It was concluded that the solution meets the objectives set out in the planning, however, a disagreement was found regarding the excessive number of messages in the controller terminal that prevents the network administrator from monitoring the events reported by the controller. According to the results of the monitoring, it is possible to interpret that the DDoS attack detection and mitigation mechanism implemented in the SDN controller is effective for TCP flood packet types, UDP flood, and ICMP flood are that detection is independent of the type of attack packet.

Table 4.2: Comparative Analysis of Methods

	Entropy Method			Association Method
Attack traffic To Normal Traffic Ratio	Total Packets	Detection and Mitigation Time(s)	Total Packets	Detection and Mitigation Time(s)
25%	602	4	36	2
50%	548	6	77	2
75%	615	8	70	6
100%	750	4	122	2

When an analysis of the associated approach to its counterparts is considered and knowing that OpenFlow entries are used to mitigate the attack. When the experimental results are compared to the results from the previous experiment, it is clear that the attack traffic with the preventive approach applied in this research can achieve a greater entropy decrease rate and has the function of quick detection. Table 4.2 lists the comparison between the proposed association method and its equivalent entropy method.

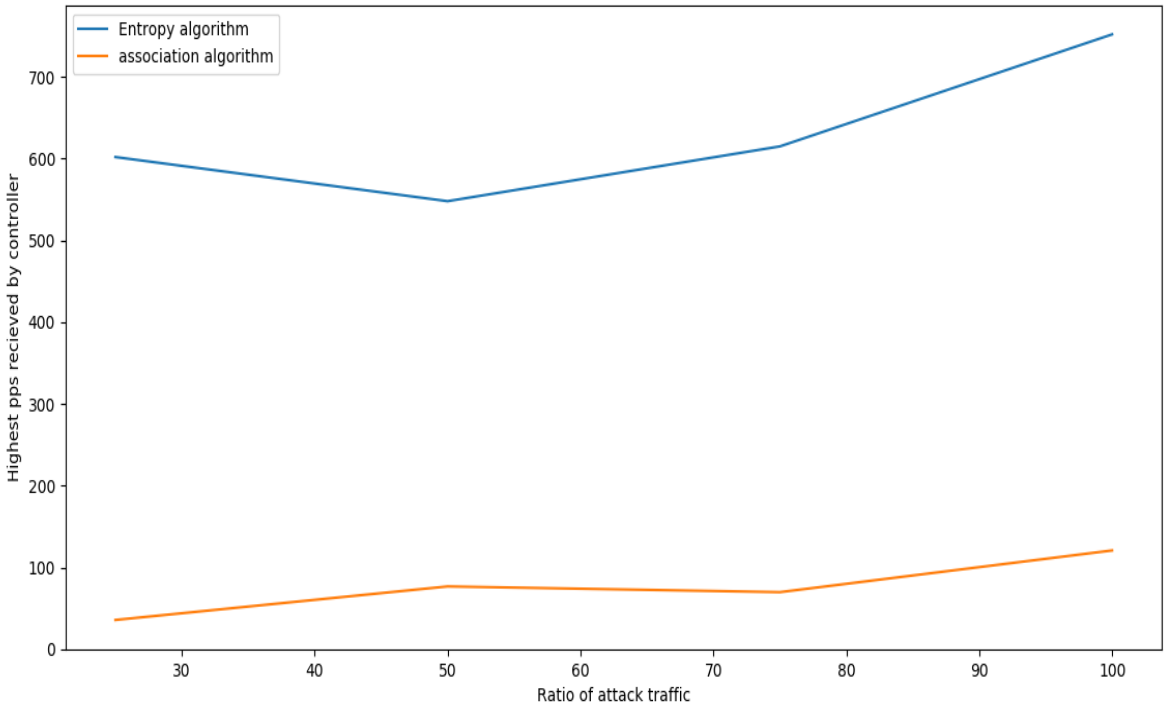

Figure 4.23: Packets Received and the Ratio of Attack Traffic.

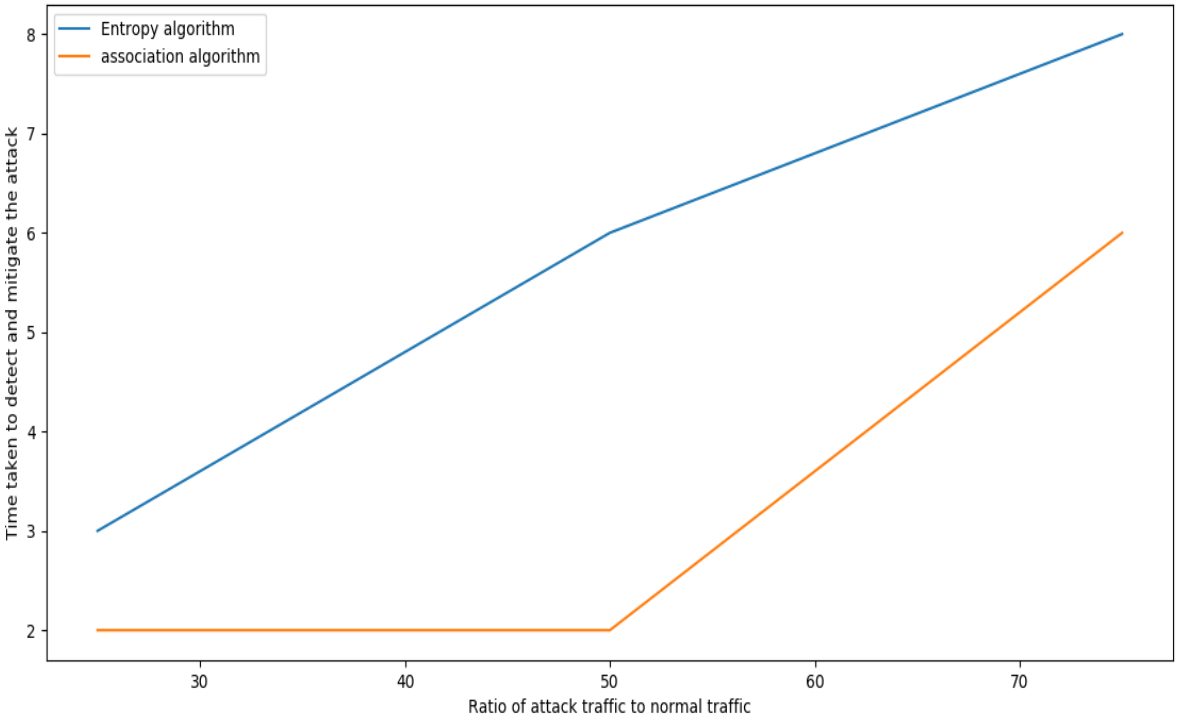

Figure 4.24: Time Comparison to Detect and Mitigate the Attack.

The test results of figure 4.23 and 4.24 above demonstrate that the association entropy has a decent attack detection capability and can successfully capitalize the interconnection of information entropy and record energy entropy. It has not only a faster attack detection performance, although it has a high entropy packet loss rate, which is much better than other approaches, indicating that the methodology can identify DDoS attacks in simulated network circumstances.

4.3.3 Duration of System Implementation

The implemented method is effective for TCP, UDP and ICMP flood DDoS attacks and meets the stated objectives. There is no set time for implementing each process or the approach itself, as this varies based on the company. As a result, the implementation time of the information security management system with this standard can be taken as a reference to avoid the implementation becoming outdated due to various factors such as continuous changes in risks, changes in the organization's priorities in terms of total protection, the appearance of new threats, and so on. The organization should estimate the deployment time primarily based on the availability of resources, for example, consider whether the individuals are responsible for the application will be dedicated exclusively to that task or assigned other tasks.

Duties of greater or lesser priority that would not allow you to complete the application in a short period of time. It is also crucial to take into account the degree of expertise of the staff in charge of the subject in issue, as it could be the case that an additional term of learning and training may be required. In terms of hardware costs, the solution does not involve the installation of any more devices because it is implemented on the same SDN controller. In this context software expenses are created because POX is a free license controller that allows you to make the required adjustments. The solution is established for a production SDN network; if this infrastructure is not available, the expenses associated with the hardware to deploy the SDN network must be addressed.

4.4 Discussions

- The implemented security solution has been validated in this research by simulating Mininet and generating different packet flood attacks, which has contributed to a high level of effectiveness by selecting the appropriate entropy threshold to detect a DDoS attack. After applying this security solution in the simulation scenario, it coincides with the results of previous research allowing early detection for DDoS attacks and is effective with any type of packet (TCP, UDP and ICMP).
- In the present work, controlled attacks were carried out for the monitoring process, because it is a simulated environment and it is not possible to have real attacks. The application of this process in a real-world environment does not require the generation of attacks by the organization, but rather the monitoring of its performance in the face of common events that occur in the network during a time period specified in the planning. This would provide in more accurate results for the solution's performance reporting.
- The implemented system manages to achieve the stated objectives; however, it makes it difficult for the network administrator to represent the point at which the attack occurred and was mitigated, because this alert is lost in the controller terminal due to an excessive number of messages indicating the network's current entropy. This also prevents further incidences in the controller from being discovered, making it impossible to monitor such devices and, as a result, of the network.
- The non-conformity identified in the monitoring process was corrected by removing the lines of code that print the logs of information relevant to the network's entropy value, leaving just the alert of detection and mitigation of the attack. On the detecting module and the POX controller module, corrective steps were taken. While confirming that a DDoS attack is ongoing, print information. Even when there is no traffic, the command line reports the network's entropy value all the time.
- The solution was successful for the scenario proposed with the POX controller; however, there was little information found for other controllers.

5. CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The SDN design decouples the control plane from the data plane, making network reconfiguration and centralized management easier. However, due to the centralized nature of SDN, the controller becomes a single point of failure that may be targeted by DDoS attacks, exposing the entire network. A conceptual methodology for safeguarding SDN networks, as well as a new technique for detecting and mitigating DDoS attacks, are discussed in this research. The execution of a DDoS attack directed at the SDN controller is rather simple and clear in that no unique packet type is required for the attack, since any of the UDP, TCP, or ICMP protocols may be used to flood the controller and analyze its resources. The suggested technique computes the number of DST_IPs associated with a single SRC_IP as well as the number of SRC_IPs associated with a single MAC_ADDR. Then it compares the computed values to the threshold values and adds the necessary flow entries.

The successful application of each fabric of the methodology resulted in the implementation of a solution based on network entropy calculation, capable of detecting UDP flood, TCP flood, and ICMP flood attacks early on and mitigating them by blocking the switch port from which the malicious traffic originates. Implementing the controller module solution saves money since the organization does not need to purchase additional software or hardware because the solution is based on open source.

The implementation of the selected method is not necessarily complex, and making more advanced adjustments requires knowledge of the Python programming language even if you have basic understanding of how to use the POX controller. Network entropy is a value that may be used to detect anomalies in network traffic, in this example recognizing that a host is receiving an unusually large number of packets in comparison to the other hosts on the network. The solution achieves the objective of detecting and mitigating DDoS attacks on the network, however, the action of blocking the switch port connected to the host that performs the attack has the disadvantage that does not allow such a host to send or receive legitimate traffic.

As an improvement for this problem, Machine learning techniques could be implemented to filter the packets, so that the host is not blocked in its entirely, only the malicious traffic it generates.

The simulation results showed that the proposed approach is successful for detecting the attack in a shorter period of time and with a low number of packets. Using flow entries, the attack is successfully mitigated after discovery. Using the POX controller has the benefit of previously having been thoroughly evaluated due to its open distribution, therefore there is a source of knowledge available on the solutions that can be developed with it, as opposed to other controllers whose information is rare.

5.2 Recommendations

It's advisable to make no configurations on the POX controller module l3 learning because if it has an error, the client will not start; instead, make a duplicate of the file and build the configurations on this and choose a method with extensive theoretical and, more importantly, practical documentation to ensure that all necessary information is available at the time of deployment.

It is recommended that a reasonable time be set aside to complete each subprocess, based on the availability of professionals and the resources needed to carry out the optimization method.

The work discussed is confined to the network layer, that is, it only evaluates Internet Protocol as a parameter for attack detection. In the future, this study might be expanded to include features from other layers, such as MAC addresses in the data link layer.

In addition, a load balancing module may also be built on a pool of controllers to further redistribute risk and efficiently manage traffic to the controller. It is also recommended to implement the GNS3 tool. Complex network topologies may be built with this tool. The simulation of multiple platforms is simple. The virtual network can be linked to the actual world.

Acknowledgments: This research thesis was funded by Adama Science and Technology University under the grant number of ASTU/SM-R/510/22, Adama, Ethiopia.

6. REFERENCES

AbdelAzim, N. M., Fahmy, S. F., Sobh, M. A., & Eldin, A. M. B. (2021). A hybrid entropy-based DoS attacks detection system for software defined networks (SDN): A proposed trust mechanism. Egyptian Informatics Journal, 22(1), 85-90.

Abdullah, M. Z., Al-awad, N. A., & Hussein, F. W. (2019). Evaluating and Comparing the Performance of Using Multiple Controllers in Software Defined Networks. International Journal of Modern Education & Computer Science, 11(8).

AL-Musawi, B. Q. M. (2012). Mitigating DoS/DDoS attacks using iptables. International Journal of Engineering & Technology, 12(3), 101-111.

Aladaileh, M., Anbar, M., Hasbullah, I. H., Sanjalawe, Y. K., & Chong, Y.-W. (2021). Entropy-based approach to detect DDoS attacks on software defined networking controller. CMC-COMPUTERS MATERIALS & CONTINUA, 69(1), 373-391.

Anthony, L. (2015). Security Risks in SDN and Other New Software Issues. RSA Conference. Frost and Sullivan,

Bannour, F., Souihi, S., & Mellouk, A. (2017). Distributed SDN control: Survey, taxonomy, and challenges. IEEE Communications Surveys & Tutorials, 20(1), 333-354.

Bouras, C., Kollia, A., & Papazois, A. (2017). Teaching network security in mobile 5G using ONOS SDN controller. 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN),

Cui, Y., Qian, Q., Guo, C., Shen, G., Tian, Y., Xing, H., & Yan, L. (2021). Towards DDoS detection mechanisms in software-defined networking. Journal of Network and computer Applications, 103156.

Dao, N.-N., Kim, J., Park, M., & Cho, S. (2016). Adaptive suspicious prevention for defending DoS attacks in SDN-based convergent networks. PloS one, 11(8), e0160375.

DeCusatis, C., Carranza, A., & Delgado-Caceres, J. (2016). Modeling Software Defined Networks using Mininet. Proc. 2nd Int. Conf. Comput. Inf. Sci. Technol. Ottawa, Canada,

Deepa, V., Sudar, K. M., & Deepalakshmi, P. (2019). Design of ensemble learning methods for DDoS detection in SDN environment. 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN),

Dharma, N. G., Muthohar, M. F., Prayuda, J. A., Priagung, K., & Choi, D. (2015). Time-based DDoS detection and mitigation for SDN controller. 2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS),

Dong, S., Abbas, K., & Jain, R. (2019). A survey on distributed denial of service (DDoS) attacks in SDN and cloud computing environments. IEEE Access, 7, 80813-80828.

Emmons, m. a. (2021). ScapyPacket Manipulation in Kali Linux. geeksforgeeks.org. Retrieved 15 Apr, 2022 from https://www.geeksforgeeks.org/scapy-packet-manipulation-in-kali-linux/

Glăvan, D., Răcuciu, C., Moinescu, R., & Antonie, N.-F. (2019). Detecting the DDoS attack for SDN Controller. Scientific Bulletin" Mircea cel Batran" Naval Academy, 22(1), 1-8.

Hadi, F., Imran, M., Durad, M. H., & Waris, M. (2018). A simple security policy enforcement system for an institution using SDN controller. 2018 15th International Bhurban Conference on Applied Sciences and Technology (IBCAST),

Hameed, S., & Ahmed Khan, H. (2018). SDN based collaborative scheme for mitigation of DDoS attacks. Future Internet, 10(3), 23.

Haque, M. R., Tan, S. C., Lee, C. K., Yusoff, Z., Ali, S., Kaspin, I., & Ziri, S. R. (2018). Analysis of DDoS attack-aware software-defined networking controller placement in Malaysia. In Recent Trends in Computer Applications (pp. 175-188). Springer.

Haque, M. R., Tan, S. C., Yusoff, Z., Nisar, K., Lee, C. K., Kaspin, R., Chowdhry, B. S., Ali, S., & Memon, S. (2020). A Novel DDoS Attack-aware Smart Backup Controller Placement in SDN Design. Annals of Emerging Technologies in Computing (AETiC), Print ISSN, 2516-0281.

Jose, A. S., Nair, L. R., & Paul, V. (2021). Towards Detecting Flooding DDOS Attacks Over Software Defined Networks Using Machine Learning Techniques.REVISTAGEINTEC-GESTAOINOVETECNOLOGIAS, 11(4), 3837-3865.

Kalkan, K., & Zeadally, S. (2017). Securing internet of things with software defined networking. IEEE Communications Magazine, 56(9), 186-192.

Kandoi, R., & Antikainen, M. (2015). Denial-of-service attacks in OpenFlow SDN networks. 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM),

Kodzai, C. (2020). Impact of network security on SDN controller performance University of Cape Town].

Kreutz, D., Ramos, F. M., & Verissimo, P. (2013). Towards secure and dependable software-defined networks. Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking,

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2014). Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 103(1), 14-76.

Lawal, B. H., & Nuray, A. (2018). Real-time detection and mitigation of distributed denial of service (DDoS) attacks in software defined networking (SDN). 2018 26th Signal Processing and Communications Applications Conference (SIU),

Masoudi, R., & Ghaffari, A. (2016). Software defined networks: A survey. Journal of Network and computer Applications, 67, 1-25.

Mishra, A., Gupta, N., & Gupta, B. (2021). Defense mechanisms against DDoS attack based on entropy in SDN-cloud using POX controller. Telecommunication systems, 77(1), 47-62.

Mladenov, B. (2019). Studying the DDoS attack effect over SDN controller southbound channel. 2019 X National Conference with International Participation (ELECTRONICA),

Mousavi, S. M., & St-Hilaire, M. (2018). Early detection of DDoS attacks against software defined network controllers. Journal of Network and Systems Management, 26(3), 573-591.

Patel, P. (July 5, 2016). Implementing software defined network (SDN) based firewall. Nirma University. Retrieved April 19, 2022 from https://www.opensourceforu.com/2016/07/implementing-a-software-defined-network-sdn-based-firewall/

Sahoo, K. S., Sarkar, A., Mishra, S. K., Sahoo, B., Puthal, D., Obaidat, M. S., & Sadun, B. (2017). Metaheuristic solutions for solving controller placement problem in SDN-based WAN architecture. ICETE 2017-Proceedings of the 14th International Joint Conference on e-Business and Telecommunications,

Sakellaropoulou, D. (2017). A qualitative study of sdn controllers. Athens University of Economics and Business.

Salman, O., Elhajj, I. H., Kayssi, A., & Chehab, A. (2016). SDN controllers: A comparative study. 2016 18th mediterranean electrotechnical conference (MELECON),

Shohani, R. B., Mostafavi, S., & Hakami, V. (2021). A statistical model for early detection of DDoS attacks on random targets in SDN. Wireless Personal Communications, 120(1), 379-400.

Stallings, W. (2015). Foundations of modern networking: SDN, NFV, QoE, IoT, and Cloud. Addison-Wesley Professional.

Thomas, R. M., & James, D. (2017). DDOS detection and denial using third party application in SDN. 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS),

Wang, R., Jia, Z., & Ju, L. (2015). An entropy-based distributed DDoS detection mechanism in software-defined networking. 2015 IEEE Trustcom/BigDataSE/ISPA,

Yu, S., Zhang, J., Liu, J., Zhang, X., Li, Y., & Xu, T. (2021). A cooperative DDoS attack detection scheme based on entropy and ensemble learning in SDN. EURASIP Journal on Wireless Communications and Networking, 2021(1), 1-21.

7. APPENDICES

7.1 Appendix A: Experiment

A1. Set up & Configuration

This section includes and provides further information on the specifications and customization settings of the tools used in the dissertation's experiments. The specific details will give the reader with the installation processes, setup technique, and configurations that were used to deliver the proposed solution.

A2. Prerequisites for setting up an Environment

a. Virtual Box

Simply download the most recent version of Virtual Box for your operating system from the link: https://www.virtualbox.org/wiki/Downloads

b. Mininet

Mininet the packet forwarder emulation software is installed on Ubuntu server. The installation steps for the software are as shown below from Mininet website. Source of the packages is extracted from http://mininet.org/download/. The installation option that was used is the native installation from source option.

c. Scapy on Mininet

To install scapy manually type in the following command:

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

d. Python on Mininet

If python is not installed on Mininet, type in the following command to install it manually: $ sudo apt-get install python3

e. SDN Controller setup using POX

Move to the pox directory in the Virtual Box terminal running Mininet by entering the following command: $ cd pox

In the SSH window and terminal of Virtual Box enter the following command to run the pox controller: $ python3./pox.py forwarding. l3_editing

A3. Hardware Features Requirement

Table 7.1: Hardware Features and Roles.

Features	Detail
Processor	Intel® Core™ i7-8565U CPU @ 1.80GHz 1.99GHz
RAM	12GB
Hard disk	1.5TB NV Me Solid State Drive (SSD)
Operating System	Windows® 11 and Ubuntu 20.04 LTS
Wireless Network	Intel ® Dual Band Wireless-AC 8265

A4. Experimental Setup Server Features

Table 7.2: Virtual Machine Specifications and Roles.

Specifications	Detail
Virtual Machine Name	Mininet + POX
Platform	Ubuntu
Version	Ubuntu (64-bit) 20.04 LTS
RAM	4096MB
Virtual Hard disk size	40GB
Network Configuration	Bridged Adaptor
CPU	3 Cores

7.2 Appendix B: Source Code Listing

B1. Launch Traffic Code

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


def sourceIPgen():
    not_valid = [10,127,254,1,2,169,172,192]

    first = randrange(1,256)

    while first in not_valid:
        first = randrange(1,256)

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

    return ip


def gendest(start, end):

    first = 10
    second =0; third =0;
    ip = ".".join([str(first),str(second),str(third),str(randrange(start,end))])
   # print start
   # print end
    return ip

#if __name__ == '__main__':
  #main()

def main(argv):
   # global start
   # global end
    print (argv)
    try:
        opts, args = getopt.getopt(sys.argv[1:],'s:e:',['start=','end='])
        opts, args = getopt.getopt(sys.argv[1:],'s:e:',['start=','end='])
    except getopt.GetoptError:
        sys.exit(2)
    for opt, arg in opts:
        if opt =='-s':
            start = int(arg)
        elif opt =='-e':
            end = int(arg)
    if start == '':
        sys.exit()
    if end == '':
        sys.exit()


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

    for i in range(1000):
        packets = Ether()/IP(dst=gendest(start, end),src=sourceIPgen())/UDP(dport=80,sport=2)
        print(repr(packets))

        sendp( packets,iface=interface.rstrip(),inter=0.1)

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

B2. Launch Attack Code

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

#this function generates random IP addresses


# these values are not valid for first octet of IP address

  not_valid = [10,127,254,255,1,2,169,172,192]

  first = randrange(1,256)


  while first in not_valid:
    first = randrange(1,256)
    print (first)
  ip = ".".join([str(first),str(randrange(1,256)), str(randrange(1,256)),str(randrange(1,256))])
  print (ip)
  return ip


def main():
  for i in range (1,5):
    mymain()
    time.sleep (10)
#send the generated IPs
def mymain():

#getting the ip address to send attack packets
  dstIP = sys.argv[1:]
  print (dstIP)
  src_port = 80
  dst_port = 1

# open interface eth0 to send packets
  interface = popen('ifconfig | awk \'/eth0/ {print $1}\'').read()

  for i in range(0,500):
# form the packet
    packets = Ether()/IP(dst=dstIP,src=sourceIPgen())/UDP(dport=dst_port,sport=src_port)
    print(repr(packets))

# send packet with the defined interval (seconds)
    sendp( packets,iface=interface.rstrip(),inter=0.025)

#main


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