REAL-TIME SECURITY APPROACH TO SOFTWARE-DEFINED NETWORKING (SDN)
M.Sc. Thesis

Babatunde Hafis LAWAL

Eskişehir, 2019

REAL-TIME SECURITY APPROACH TO SOFTWARE-DEFINED NETWORKING (SDN)

Babatunde Hafis LAWAL

M.Sc. THESIS

Program in Telecommunication
Supervisor: Assoc. Prof. Dr Nuray AT

Eskisehir
Eskisehir Technical University
Graduate School of Sciences

January 2019

FINAL APPROVAL FOR THESIS

This thesis titled "Real-Time Security Approach to Software-Defined Networking (SDN)" has been prepared and submitted by Babatunde Hafis LAWAL in partial fulfillment of the requirements in "Eskisehir Technical University Directive on Graduate Education and Examination" for the Degree of Master of Science (M.Sc.) in Electrical and Electronics Engineering Department has been examined and approved on 16/01/2019.

Committee Members	**Title - Name Surname**	**Signature**
Member (Advisor)	: Assoc. Prof. Dr. Nuray AT	.....
Member	: Assoc. Prof. Dr. Hakan ŞENEL	.....
Member	: Asst. Prof. Dr. Erol SEKE	.....

Prof. Dr. Ersin YÜCEL

Director of Graduate School of Science

ABSTRACT

REAL-TIME SECURITY APPROACH TO SOFTWARE-DEFINED NETWORKING (SDN)

Babatunde Hafis LAWAL

Department of Electrical and Electronics Engineering

Eskişehir Technical University,

Graduate School of Sciences

January 2019

Advisor: Assoc. Prof. Dr. Nuray AT

The emergence of Software-defined network (SDN) with its ability to enhance programmability in networking and the separation of the control plane from the infrastructural plane has shifted the paradigm in networking. The centralization of the control plane requires robust and real-time security techniques to protect it from any sign of vulnerabilities associated with the network such as distributed denial of service (DDoS) attacks, replay attacks, man-in-the-middle attacks (MITM), etc. In this study, a real-time security approach that can detect and mitigate the security vulnerabilities susceptible to the SDN network is proposed. The proposed method adopts the sFlow technique and the IPSec protocol for security implementation. A secure network architecture was built and implemented on Mininet which runs on a Virtual Machine (VM) to evaluate and analyze the effectiveness of the proposed method. The method not only can detect attacks on the network but does it in real-time which allows it to quickly start a mitigation process to protect the network from being overwhelmed.

Keywords: Software-defined network (SDN), Mininet, sFlow, IPSec, DDoS.

ÖZET

YAZILIM TANIMLI AĞLARDA GERÇEK-ZAMANLI GÜVENLİK YAKLAŞIMI

Babatunde Hafis LAWAL

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Eskişehir Teknik Üniversitesi,

Fen Bilimleri Enstitüsü

Ocak 2019

Danışman: Doç. Dr. Nuray AT

Geleneksel ağların programlanmasını sağlaması, kontrol düzlemini altyapı düzleminden ayırması gibi özelliklerinden dolayı yazılım tanımlı ağlar (Software Defined Network, SDN) geleneksel ağ paradigmasını değiştirmeye devam etmektedir. SDN ağlarında kontrol düzleminin merkezi hale gelmesiyle, dağıtılmış hizmet reddi (distributed denial of service DDoS), ortadaki adam (man-in-the-middle, MITM), tekrar saldırıları gibi diğer ağ güvenlik açıkları için gürbüz ve gerçek-zamanlı güvenlik tedbirlerinin alınmasını zorunlu hale gelmiştir. Bu tezde, SDN ağlarına yöneltilen güvenlik saldırılarını tespit ederek uzaklaştıran gerçek-zamanlı bir güvenlik yaklaşımı önerilmiştir. Önerilen yöntemde güvenlik, sFlow tekniği ve IPSec protokolü kullanılarak sağlanmıştır. Yöntemin etkinliğini değerlendirmek ve analiz etmek için, önerilen güvenli ağ mimarisi sanal makine (virtual machine, VM) üzerinde çalışan Mininet'te gerçeklenmiştir. Önerilen yöntem sadece ağ saldırılarını tespit etmekle kalmayıp bu işlemi gerçek-zamanlı olarak yapmakta ve ağ istila edilmeden bu saldırıları uzaklaştırma işlemini hızla başlatmaktadır.

Anahtar Kelimeler: Yazılım Tanımlı Ağ (SDN), Mininet, sFlow, IPSec, DDoS.

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Assoc. Prof. Dr. Nuray AT for her unwavering support, motivation, enthusiasm, patience, and contributions throughout my study and research. The door to her office was always open whenever I had questions about my research. I could not have imagined having a better advisor and mentor for my study.

I am equally grateful to Dr. Ogundile Olayinka for encouraging and introducing me to the art of research and with whom I wrote my first publication.

Had I forgotten to appreciate the Turkish government and her scholarship board (YTB) for providing this opportunity, I would have been an ingrate. I thank my fellow lab mate, Hussam Kanaan for the stimulating discussions and for all the fun we have had in the last two years. I also appreciate my friend Boubacar Balde for his steadfastness and support.

I am gratefully indebted to my teacher and mentor, Ustaadh Usman Adebayo Giwa, to whom my moral education is credited. May he live long in sound health and abundant wealth.

Finally, I must express my very profound gratitude to my wife, AbdulGaniyu Nimotalai, my parents, brothers, sisters and friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Babatunde Hafıs LAWAL

16/01/2019

STATEMENT OF COMPLIANCE WITH ETHICAL PRINCIPLES AND RULES

I hereby truthfully declare that this thesis is an original work prepared by me; that I have behaved in accordance with the scientific ethical principles and rules throughout the stages of preparation, data collection, analysis, and presentation of my work; that I have cited the sources of all the data and information that could be obtained within the scope of this study, and included these sources in the references section; and that this study has been scanned for plagiarism with "scientific plagiarism detection program" used by Eskisehir Technical University, and that "it does not have any plagiarism" whatsoever. I also declare that, if a case contrary to my declaration is detected in my work at any time, I hereby express my consent to all the ethical and legal consequences that are involved.

Babatunde Hafis LAWAL

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

INDEX OF ABBREVIATIONS AND SYMBOLS

API	: Application Programming Interface
DOS	: Denial of Service
DDoS	: Distributed Denial of Service
ICMP	: Internet Control Message Protocol
IoT	: Internet of Things
IP	: Internet Protocol
IPSec	: Internet Protocol Security
OS	: Operating System
OVS	: Open vSwitch
QOS	: Quality of Service
RAM	: Random-access Memory
SDN	: Software-Defined Networks
sFlow	: Sampled flow
SYN	: Synchronization Message
TCP	: Transport Control Protocol
TLS	: Transport Layer Security
UDP	: User Datagram Protocol
VM	: Virtual Machine

1. INTRODUCTION

The ability to abstract the control functions from the forwarding elements has turned the tide in networking and has led to the emergence of Software-defined networking (SDN). SDN is changing the way IT networking infrastructures are controlled, managed and configured. Network administrators see SDN as a breakthrough for having total remote access and control of the network while network providers and operators believe that SDN will reduce capital expenditure (CAPEX), operational expenditure (OPEX) and energy consumption. Their arguments are all true as SDN allows the programmability of the network, the modification of routing rules or data flows according to real-time necessities, monitoring of network performances from a dashboard and the abstraction of the control plane from the data/infrastructural plane. Other benefits include the implementation of new network services such as quality of service (QoS), enforcement of new policies, access control, traffic engineering, security enhancement, bandwidth management, etc.

SDN solves the issue of proprietary in the traditional network. It has proven to be the required technology to make networking more scalable, dynamic and removes the data plane devices vendor proprietary [1]. SDN abstracts the control functions from the switches or routers as against the traditional network, rendering the data plane devices as merely forwarding agents. The forwarding rules are logically generated by the controller and communicated to the switches through the OpenFlow protocol [2-5].

OpenFlow facilitates the communication processes between the logically centralized software controller and the network forwarding devices, thus, making it possible to implement and deploy SDN technology in current networks [6]. The OpenFlow controller provides a centralized overview of the network with the ability to detect network vulnerabilities and intrusions and implements security policies [7].

1.1. The Problem Statement

Regardless of the intelligence and aptness associated with the SDN, security vulnerabilities remain a bottleneck in its deployment. Since SDN is an active network [6], it suffers from the challenges facing the active network, which is securing the nodes from a malicious injection. Active networks are programmable and allow real-time modification of the network parameters based on necessity, and are prone to threats such as Distributed Denial of Service (DDoS) Attacks, Man in The Middle Attacks (MITM), Replay Attacks, etc.

With little or no advance warning, a DDoS attack can easily overwhelm the computing and communication resources of its victim within a short period of time [8]. DDoS attack has been on the rise ever than before, it aims to exhaust the data and/or control resources of the SDN network and render the network unavailable to eligible users. An MITM attack, on the other hand, causes lack of integrity of the transmitted data, i.e., an unauthorized user may alter or modify the packet in transit [9].

The growth in the rate of connected devices (Internet of Things) has also contributed to the exacerbation of the susceptibility in the network. The controller is one of the major components in the SDN network, if it can be compromised, then the whole network can be brought down. This has called for an urgent defence mechanism that can protect the SDN network efficiently against the threats associated with its deployment and keeping the research community conscious.

1.2. The Objective and Relevance of The Study

In this study, a defence mechanism that could detect vulnerabilities and protect the SDN and its resources was proposed. The proposed method detects and prevents attacks before they could harm or bring down the network. Specifically, it should protect the SDN network against DDoS attack, MITM attacks, etc. The system consists of the sFlow technology and the IPsec protocol for defence strategy implementation. sFlow detects irregularities in packet transmission by sampling fractions of all packets transiting the network. In the case of anomalies in the packets flow, sFlow generates handling rules and communicates it to the controller for flow rule adjustment on the forwarding devices.

IPSec, on the other hand, ensures that only authorized users are allowed on the network through its authentication protocol. It also ensures the integrity of the network by encrypting the transmitted data and encapsulating the whole packets, this will protect the network against attacks such as MITM and replay attacks.

With the growing adoption of the SDN network, it is important to place emphasis on its security. The result of this study will help stakeholders (network operators and administrators) in this field by providing them with a secure means in the deployment of SDN on their network. It will also serve as a guide and/or reference to researchers, academics, and students working in this field. This study provides a clear and concise explanation of the SDN network.

The objective of this study is to set up a secured SDN network by protecting it from the vulnerabilities including the DDoS attacks, MITM, Replay attacks, etc. The setup is done in MININET and the results are analyzed thereafter.

1.3. Thesis Organization

The thesis is organized as follows:

- **Chapter 2** gives a detailed background information on the SDN network, OpenFlow protocol and other basic components of the SDN network. The SDN architecture, how it works, comparison with the traditional networks, benefits and the security challenges associated with it are all given. Some previous works and studies on SDN security are also reviewed.
- In **Chapter 3,** the proposed tools for detecting and mitigating the effects of the vulnerabilities on the SDN network are explained.
- **Chapter 4** describes the test environment used in setting up and evaluating the SDN network and a step by step explanation on creating a prototype of the SDN network. It also emulates a DDoS attack and how the proposed method detects and mitigates its effects with an in-depth analysis of each stage on the SDN network.
- **Chapter 5** summarizes and concludes the study while suggesting some future works that can be done.

2. BACKGROUND ON SOFTWARE-DEFINED NETWORK (SDN)

Software-defined network (SDN) is a networking approach that allows network administrators to programmatically initialize, control, change, and manage network behaviours dynamically via open interfaces such as OpenFlow protocol. Prior to the evolution of SDN, networking devices were closed and proprietary, while the manufacturers of these devices were satisfied, network administrators were frustrated with the complexity associated with the network. A centralized network where configuration could be done in a logical manner without having to configure switches separately is the answer.

The first breakthrough came in 2006 when a PhD student from Stanford University, Martin Casado, proposed that there should be a provision in the network security that checks the packets at every crucial point or node within the network. He thereby emphasized the creation of a central security system within the network. The first model he proposed was Secure Architecture for the Networked Enterprise (SANE) which further evolved into Ethane [10] and later OpenFlow. OpenFlow initializes the rules of communication between data plane and a centralized control plane while it enables the entire network to be controlled through written software applications (APIs) [11].

2.1. SDN Layers

The SDN's priority is to decouple the control and data plane from each other and to allow network programmability and automation. To achieve this, SDN divides the network operations into three layers: Infrastructure (Data), Control and Application layers.

The control layer comes under the responsibility of a centralized controller that takes all flow forwarding decisions on the network. The communication between the data and control layers is achieved through the OpenFlow protocol in the southbound interface while the communication between the controller and the application layer is achieved through the representational state transfer application programming interface (REST API) in the northbound interface as specified by the Open Networking Foundation (ONF). Figure 2.1 shows the SDN architecture and how the layers are being separated.

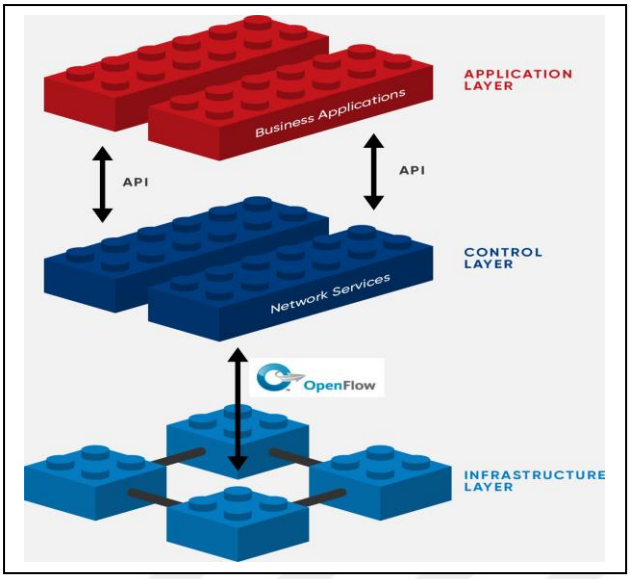

Figure 2. 1 *Software-defined network architecture [12]*

2.1.1. The infrastructure layer

This is sometimes referred to as the data plane or forwarding plane. The basic components of this layer are the network forwarding elements such as switches, routers, OF-switches, other virtual switches, etc. This layer is responsible for forwarding data traffic and the collection of statistics through which the controller can monitor and update the performance of the network [13]. It performs actions such as forwarding, routing and changing packets routes [14].

The control associated with this layer in traditional networking has been abstracted in SDN and transferred to the control plane. The elements in this layer act on the controller-based instructions to update the flow entries in the flow tables. An example of the infrastructure layer can be a group of network switches and routers in the data centres. This layer is the physical layer through which the network can be virtualized via the control layer, and where the SDN controller can manage and control the underlying physical network.

2.1.2. The control layer

The control layer [15, 16] can be logically found just above the infrastructure layer. The intelligence abstracted from the forwarding devices (in traditional networking) resides in this layer. This layer is responsible for communicating with both the infrastructure layer and the layer above it (the application layer). This layer is composed of one or more software controllers which enable programmability and

logical communication with the infrastructure layer through southbound interfaces such as OpenFlow, Ovsdb, Netconf, etc., and communication with the application layer through the northbound interface such as the REST API. It can perform functions such as updating flow entries, implementing firewall rules, switching, routing, fetching network information, obtaining statistics and topology details, etc.

There are many types of controller, some are OpenFlow based controller while others are open source OpenFlow controller alternatives. Some of the controllers are given in the following:

- **NOX [17]:** It was the first OpenFlow controller written in C++ and provides API for Python too. Many research and development projects in the early stage of OpenFlow and SDN were based on NOX.
- **POX [18]:** This is the Python-only version of NOX. It can be considered as a general, open source OpenFlow controller written in Python, and a platform for rapid development and prototyping of network applications. The primary target of POX is research.
- **OpenDayLight (ODL) [19]:** The ODL controller is a highly available, modular, extensible, scalable, and multi-protocol controller infrastructure built for SDN deployment on modern heterogeneous multi-vendor networks. The Service Abstraction Layer (SAL) makes available the needed abstractions to support different southbound protocols such as OpenFlow via plugins.
- **Ryu [20]:** It supports various protocols for managing network devices, such as OpenFlow (1.0, 1.2, 1.3 and Nicira extensions), Netconf, OF-config etc. The goal of Ryu is to develop a high-quality operating system for SDN that can be used in a large production environment. All resources for this controller are freely available under the Apache 2.0 license. Ryu is implemented in Python and its development is truly open.
- **Floodlight [21]:** It is an enterprise-class Open SDN controller, Apache-licensed, Java-based OpenFlow. It is supported by a community of developers including the Big Switch Networks. Floodlight is written in Java and its source code repository is available on GitHub.

- **Maestro:** This is a network operating system for orchestrating network control applications. It provides interfaces for implementing modular network control applications to access and modify the state of the network and coordinate their interactions via multiple protocols including but not limited to OpenFlow.
- **Trema:** This is an OpenFlow controller framework that includes everything needed to create OpenFlow controllers in Ruby and C. Its source package includes basic libraries and functional modules that work as an interface to OpenFlow switches.
- **Beacon:** This is a fast, cross-platform, Java-based controller that supports event-based and threaded operation. It was coded in Java and runs on many platforms, from Android phones to high-end multi-core Linux servers. For example, you can replace your running Learning Switch Net App without disconnecting switches on Beacon.
- **FlowER**: an open-source Erlang based OpenFlow controller. Its purpose is to provide a simplified platform for writing network control software in Erlang [22].

2.1.3. The application layer

This layer [23] is found logically above the control layer. It is open to development and innovation. It communicates with the controller through the northbound interface via REST API's. Applications can be developed based on the information obtained from the controller or based on specific requirements. It can abstract an overview of the network performance and information such as network topology, network statistics, network state from the controller and leverage on this information to provide appropriate guidance to the control layer. Examples of applications that can be developed include, but not limited to, network monitoring, network automation, network management and configuration, network policies and security, network troubleshooting etc.

Some of the SDN programming languages that meet the requirements are Frenetic [24], its successor, Pyretic [25], and Procera [26]. These languages provide

declarative syntax based on functional reactive programming. Due to the nature of functional reactive programming, these languages provide a composable interface for network policies [13].

These applications are developed to proffer end-to-end solutions for both small and large enterprises, data centres, network operators etc. There are many ready-made solutions SDN based applications from different vendors such as Cisco, Brocade, Hp etc.

2.2. Benefits of SDN

SDN implementation assures innovation and new applications. An example of this is the traffic adjustment (traffic engineering) that can be executed on switches by the controller to ensure load balancing and traffic mapping on the network. With its global overview of the network, SDN guarantees power management (energy efficiency), network-wide access control, and home networking. It enhances the decision-making process and efficiency of the network. The introduction of new protocols and mechanisms in handling packets in a network is more relaxed and data plane and control planes can develop impartially without having a major influence on one another.

In addition, the provision for network programmability in SDN ensures seamless communication at all levels, from hardware to software and to end users. Programmability makes applications and the network aware of each other. This enables improved use of resources and creates a potential for new applications. For example, it can be used for revenue generation in which cost plans can be defined based on a level of service provision thereby increasing the CAPEX and OPEX [27]. Applications no more needed to have a full overview of the underlying network structure and the network complexity is hidden from the business applications [28].

Other benefits of SDN include: centralization, traffic programmability and automation, centralized security, effective QoS delivery, management, support for big data and virtualization, agility and network supervision, etc.

2.3. SDN and the 5G Network

The integration of SDN, network functions virtualizations, NFV and software defined radio, SDR facilitates transport network optimization in the 5G network by decoupling the control (C-plane) and the user (U-plane) of the network devices and provides a logically centralized network view and control. This will enhance flexibility and scalability of the network. It will also provide additional functions with the introductions of simple software upgrades while the network is being managed from a centralized point.

In addition, multiplexing gains and reduction in the number of required hardware resources can be realized by sharing pooled hardware resources among multiple functions. The flexibilities enabled by SDN, NFV and SDR assure quick deployment of the network, adaptability and integration of new services. They also reduce the CAPEX, OPEX and ensure energy efficiency as they allow resource sharing and reconfiguration of the network resources which are major properties of the 5G network [29].

2.4. SDN versus Traditional Network

In a traditional network, data plane and control plane reside together in each switch. The performance of the traditional network can be referred to as static and inflexible. It lacks the agility to react to changes in the network. Its control plane is distributed and not logically centralized and are usually embedded in the forwarding devices which are hardware appliances.

The performance of a network node in the traditional network consists of the interaction between the data and control planes. It is the control plane's responsibility to forecast the paths across the network and push them to the data plane to enable data forwarding. Thus, after a flow policy has been made in most cases, any policy changes would involve changing the configuration on each of the devices. In large networks, this might imply that the network operators would require to expend quite a time reconfiguring the devices routinely to adapt to the varying traffic requirements and network conditions and this is not effective for large businesses where the reconfiguration time is far important. The difference between the SDN and the traditional networks is depicted in Figure 2.2.

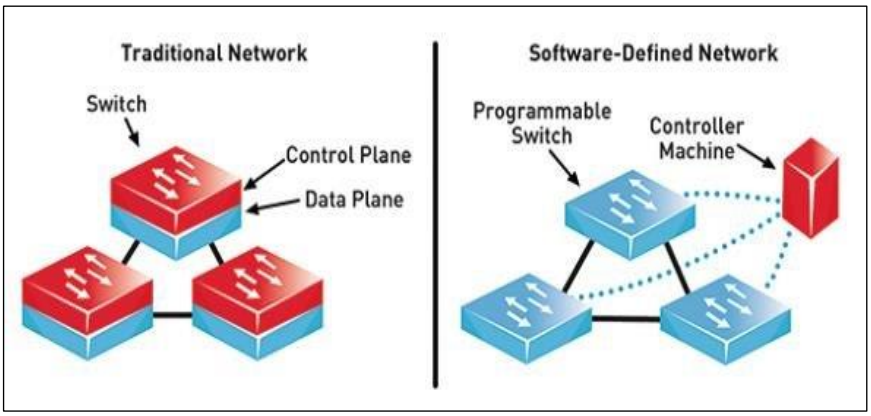

Figure 2. 2. *SDN versus traditional network*

In SDN, the control plane is logically centralized and not distributed among network nodes, as it is done in the traditional network. The controller logically communicates with network nodes to set up the data plane through a southbound SDN protocol [5].

In SDN, the controller has a complete vision of the network and can make appropriate changes according to traffic requirements. This approach significantly simplifies the implementation of some of the network functions. SDN flexibility, agility and virtualization which are achievable through programming makes SDN a better choice for large businesses. Table 2.1 summarizes the differences between the two networks.

Table 2. 1. *Differences between traditional and SDN network*

Traditional Network	**SDN Network**
Each switch has a local brain, i.e., distributed controller	Separated control and data planes. The controller is logically centralized.
Each switch independently updates its local MAC address table	The centralized controller updates the flow table
No centralized control	Centralized control
No centralized visibility	Centralized visibility, update, and reconfiguration.

2.5. OpenFlow Protocol

OpenFlow is an open communication protocol that acts on the second layer of the OSI model and provides access to the routing plane of a switch in the network. OpenFlow makes it easy to track the data packet paths within the network using a software. It was designed for the management of network traffic and allows compatibility among switches and routers of different providers and models. With OpenFlow there is no more need for any hardware configuration and the control can be achieved in a flexible way through programming.

OpenFlow is the first known SDN protocol and was selected as the standard for software-defined networking. It was first developed and tested at Stanford University in 2008 [30]. Switches can be controlled by an independent and centralized controller through OpenFlow protocol. Communication between the switch and the controller is usually done through a channel enabled for Transport Layer Security (TLS) [31]. According to the specification, each switch must include one or more flow table entries to maintain the flow records specified by the controller.

2.5.1. OpenFlow operations

In a pure OpenFlow switch, the control has been abstracted from the switch and it is only left with forwarding capability. The flow switch has one or more flow tables and a group table (see Section 4.2.3) in it where it performs packet lookups and forwarding. It also has an external channel through which it communicates securely with the external controller via TLS connection using a default TCP port of 6633.

The authentication between the switch and the controller is done mutually by exchanging certificates signed by a site-specific private key. Each switch must have a user-configurable two-way certificate with one certificate for authenticating the controller (controller certificate) and the other for authenticating to the controller (switch certificate). Traffic between the secure communication is not checked against the flow table and if a switch loses contact with the controller due to a TLS session timeout or echo request timeout or any other disconnection, it contacts the backup controller if there is any and if that fails as well, it enters an emergency mode and resets the TCP connection immediately [22].

Figure 2.3 describes the flow algorithm: when a packet arrives at a port for the first time, the switch checks its flow table for matching, if it matches any rule in its entry, it will act upon the instruction, if not, it will forward it to the controller through the packet in message and wait for the controller command (forward or drop) via the packet out message. It then acts upon the instruction and saves the new rule in its flow table for future use.

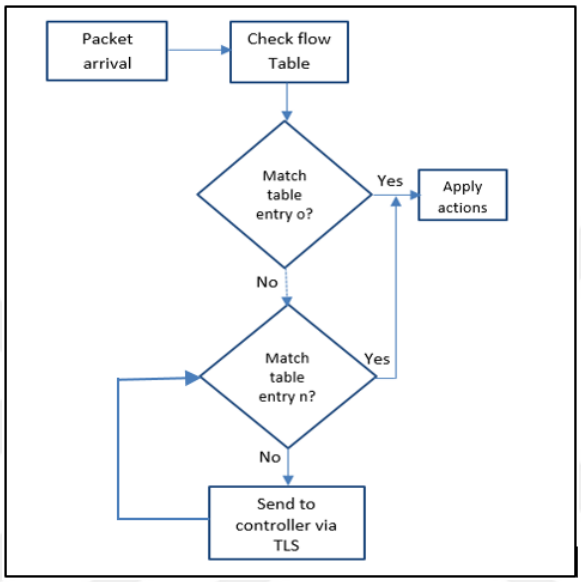

Figure 2. 3. OpenFlow *packet flow algorithm*

2.5.2. Types of OpenFlow protocol messages

There are three different message types (see Figure 2.4) defined by the OpenFlow Protocol and each of them also has sub-categories. They are the Symmetric, Asynchronous and the Controller-to-Switch messages.

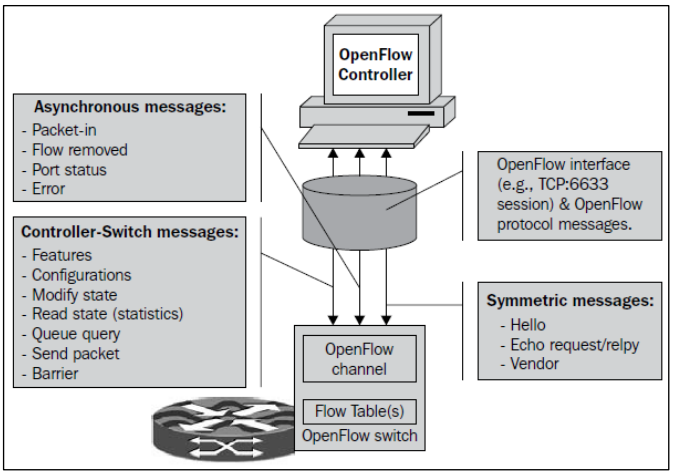

Figure 2. 4 *Types of openFlow protocol messages* [22]

Symmetric Message: This type of message is initiated by either the switch or the controller and sent without request. There are three different types of symmetric messages in the OpenFlow protocol.

Table 2. 2. *The symmetric message*

Hello	This message is exchanged between the switch and the controller upon connection setup.
Echo	This message can come from either the switch or the controller and an echo reply must be returned. It can also indicate latency, bandwidth, and/or liveliness of a controller-switch connection.
Vendor	This will allow additional functionality in future.

Asynchronous (Switch-to-Controller): This type of message is initiated by the switch and used to update the controller of network events and changes to the switch state. Switches send asynchronous messages to the controller to inform about packet arrival, changes in switch state, or an error.

Table 2. 3. *The Asynchronous (switch-to-controller) message*

Hello	Sent after a switch establishes a TCP connection to the controller
EchoRequest![Word8]	The switch requests an echo reply
EchoReply![Word8]	The switch responds to an echo request
Features!SwitchFeatures	The switch reports its features
PacketIn!PacketInfo	The switch sends a packet to the controller
PortStatus!PortStatus	The switch sends port status
FlowRemoved!FlowRemoved	The switch reports that a flow has been removed
StatsReply!StatsReply	The switch reports statistics
Error!SwitchError	The switch reports an error
BarrierReply	The switch responds that a barrier has been processed

Controller-to-Switch: This type of message is initiated by the controller and used to directly manage or inspect the state of the switch. However, it may or may not require a response from the switch.

Table 2. 4. *The controller-to-switch message*

Hello	The controller must send hello before sending any other messages.
EchoRequest![Word8]	The controller requests a switch echo
EchoReply![Word8]	The controller responds to a switch echo request.
FeaturesRequest	The controller requests feature information.
PacketOut!PacketOut	The controller commands switch to send a packet.
FlowMod!FlowMod	The controller modifies a switch flow table.
PortMod!PortMod	The controller configures a switch port.
StatsRequest!StatsRequest	The controller requests statistics.
BarrierRequest	The controller requests a barrier.
SetConfig	The controller sets configuration parameters.
GetQueueConfig !QueueConfigRequest	The controller queries configuration parameters.

2.6. Securing the SDN Network

Securing the SDN network is important since its controller is being centralized. However, for a network to be called a secure network, it must possess three characteristics which are Confidentiality, Integrity and Availability.

Confidentiality: The criterion for this is that information must not be disclosed or exposed to unauthorized subjects such as a person, a system or combined. It ensures that only those with the rights and privileges have access to information.

Integrity: This ensures the trust that can be placed on the transmitted information itself. It affirms that information has not been modified during transmission. Integrity can be categorized into two forms which are:

> ***Source Integrity*:** This guarantees that the data or information are indeed from its intended sender.
>
> ***Data Integrity*:** This ensures that the data or information have not been compromised willfully or accidentally before it reaches its intended recipient. Integrity is related mostly with the issue of MITM attacks.

Availability: This ensures that information or network resources are available and accessible when needed. For example, in the case of a DDoS attack, this means that the resources should be available at a rate which is fast enough for the system to perform its task as intended [32].

2.6.1. DDoS attacks

Distributed Denial of Service (DDoS) attacks take various forms, but they all have a common feature of flooding the network. They tend to saturate or overwhelm the computing or network resources [33]. In SDN, DDoS is referred to as the saturation of the data and/or control plane [34]. The DDoS attack is achieved by sending enormous amounts of packets usually Internet Control Message Protocol (ICMP) messages or a flood of Transmission Control Protocol (TCP) SYN messages or User Datagram Protocol (UDP) packets etc., to the victim.

In SDN, DDoS attacks are launched from various compromised devices to a targeted victim (host) on the network. The request passes through the switch and the switch, in turn, sends a request to the controller for the inclusion of the missing packets in the flow table. If the requests are granted, the controller becomes flooded with more request till its resources are saturated and the network may be unavailable to eligible users or functions at a reduced rate. The network may soon be brought down or out of communication since the objective of the attacker is to

flood or saturate the network resources. Securing the SDN network has been the focus of many researchers with the aim of improving the SDN network.

Figure 2.5 shows a set of systems which are being compromised by an attacker and are used to launch a DDoS attack on a victim.

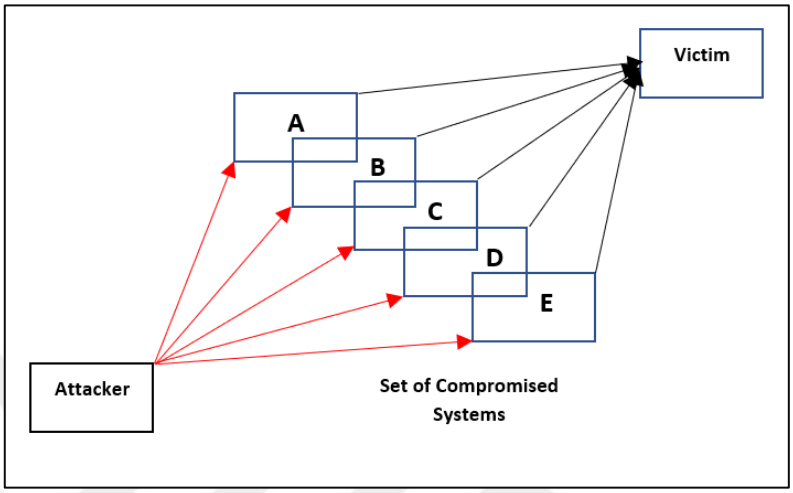

Figure 2. 5. *DDoS attack*

The main goal of this study is to propose a method which can protect the SDN network against DDoS attacks and other security vulnerabilities in real-time by using the sFlow technology embedded with the SDN controller in conjunction with the flow-based IPSec protocol. This study emulates the real SDN network in Mininet by launching a real-time DDoS attack (see Chapter 4) from compromised multiple hosts towards a target in the network with the aim of overwhelming the network resources. The proposed system described in Chapter 3 was deployed to secure the SDN network against this threat.

2.6.3. MITM attacks

A man in the middle (MITM) attack is a form of cyber-attack where an imposter or a malicious actor eavesdrop or listen to a conversation between two parties. He can impersonate both parties and have access to the information that the two parties were/are sharing. The attack can either be from a proxy or the use of malware (malicious software).

It allows a malicious actor to intercept, send and receive data meant for someone else, or not meant to be sent at all, without either of the parties involve

knowing until it is too late [35]. It usually exploits the real-time processing of events, conversations and transfer of data.

MITM attack can have access to the victim via loopholes in any of the following methods; HTTP spoofing, DNS spoofing, IP spoofing, SSL hijacking, e-mail hijacking, browser cookies, Wi-Fi eavesdropping [36]. This is usually considered as an integrity deficiency.

2.7. Review of Available Mitigation Methods for SDN

As SDN has been receiving much attention among network operators, vendors and researchers, security has taken the lead in their study.

Kreutz et al. [37, 38] were the first to provide recommendations on improving SDN security by elaborating on seven different threats against SDN and the possible counter-measures. The threats address each stage of communication between the switches and the controller, and between the controller and the applications, in addition to the interfaces within these stages. They highlight the limitations of the transport layer security (TLS) protocol to secure traffic to/from the controller, and the consequences of hijacking such session. They also raise the issue of trusting networking applications, with the view that a compromised application could jeopardize the security of the whole network.

In [33, 39], R. Kandoi argued that improving configuration parameters may alleviate the harmful effects of DoS attacks. They opined that configuring an optimal idle timeout value and flow aggregation would prevent the switch flow table from being overwhelmed. However, if the attack is launched from multiple sources as in DDoS attacks, the network may still be overwhelmed.

Klöti et al. [4, 40] used the Microsoft STRIDE model and attack trees with the assumption that the controller, network elements, and the control channel between the two are all secure. They identify several possible threats including DoS by overloading the flow table, information disclosure, and tampering the flow table through cache poisoning. Information disclosure might allow a host to deduce the flow states in an OpenFlow switch by measuring the time difference in TCP connection setup. If an OpenFlow switch to which both the client and a TCP server connect implements flow aggregation, a fast TCP setup could imply that an aggregated flow rule has already been installed in the switch, which was possibly

triggered by the communication to the TCP server from other clients connecting to the same OpenFlow switch. While interesting, it is not clear if such a side-channel attack will be effective in a network with many switches. In general, some flow states could be obtained by scanning the whole network to reconstruct the network topology.

Scott-Hayward et al. [41] surveyed the literature for potential control and data plane attacks and mitigation techniques, listing security issues discussed so far. The authors categorize research in SDN security as Analysis, Enhancement, or Solution, providing information as to which bound (north or south) is the research conducted on, and whether protocols other than OpenFlow are analyzed. A set of security threats, such as unauthorized access and data leakage, are enumerated and mapped against each of the five SDN layers and interfaces.

Dridi et al. [42] proposed an SDN-Guard for mitigating DoS attack on the SDN network. Their proposition uses an SDN application which communicates directly with the SDN controller. The application will be responsible for flow management, rule aggregation, and monitoring (collecting statistics about flows) module to ensure a secure network.

Ertaul et al. showed in [43] how hping3 tool can be used to perform a DDoS attack by sending a flood of TCP, UDP or ICMP packets on the SDN network. They noticed that packets were sent in a clear form in the SDN network and that it can encourage spoofing of the data packets flowing through the OpenFlow communication channel. They proposed a security model which uses Internet Protocol Security (IPSec) with the integration of Advanced Encryption Standard (AES) encryption and TLS for the controller-switch communication on the SDN networks.

In [44], Kaur et al. applied an OpenFlow based firewall application using the open source POX controller based on python programming to secure the network. However, this design has two drawbacks: one is the firewall overload on the controller and the other one is the use of a memoryless (stateless) firewall. It should be noted that stateful firewalls are capable of storing information about past events which can speed up decision making process on the network against future threats.

In [8], Haining Wang et al. set up a mechanism in a leaf router which connects the host to the internet. This method consumes time and network resources in the detection of flood attacks as the detection mechanism can only be applied to the leaf router.

Bing Wang in [45] presented a graph model-based attack detection and mitigation system that stores (or, records) known traffic patterns as a relational graph between patterns and their labels to distinguish between normal and abnormal traffic. The prevention technique used in that study was based on blocking the attack source IP and sending all packets to the controller via a path based on the graph model. The IPSec protocol and its applications were studied in [32, 46]. In [43], security issues of SDN network were discussed and a security measure using IPSec was proposed to protect the traffic between hosts.

The required software (Mininet) to emulate an SDN network prototype is discussed in [47, 48]. Besides, [9, 49] gives further explanation on the basic components of the SDN network such as the OpenFlow switch and the Floodlight controller used in the experimental setup.

Unfortunately, none of the above-mentioned methods were able to secure the SDN network in real-time. Motivated by the limitations of the existing works in the literature, an approach that can detect and prevent attacks in real-time and ensures the integrity of the traffic transiting the network is proposed in this thesis.

3. SECURITY TECHNIQUE

This chapter elaborates on the adopted security techniques used in the detection and mitigation of security vulnerabilities in the SDN network. The proposed system consists of the sFlow technology and the IPSec protocol for defence strategy implementation. Network irregularities in packet transmission are detected through sFlow by sampling fractions of all packets transiting the network.

In the case of anomalies in the packets flow, sFlow generates handling rules and communicates with the controller for flow rule adjustment on the forwarding devices. IPSec, on the other hand, ensures that only authorized users are allowed on the network through its authentication protocol. It also ensures the integrity of the network by encrypting the transmitted data and encapsulating the whole packets, this will protect the network against attacks such as MITM and replay attacks.

3.1. sFlow Technology

sFlow is an open source statistical time-based and real-time traffic sampling technology used for monitoring traffic in data networks at high speed. This is achieved by collecting fractions of all packets transiting the network and forwarding them to a collector for analysis. The analyzer converts the traffic into useful metrics which are accessible through the REST API and communicates handling rules to the controller if it detects unusual traffic pattern in the network [50]. The sFlow system architecture can be seen in Figure 3.1.

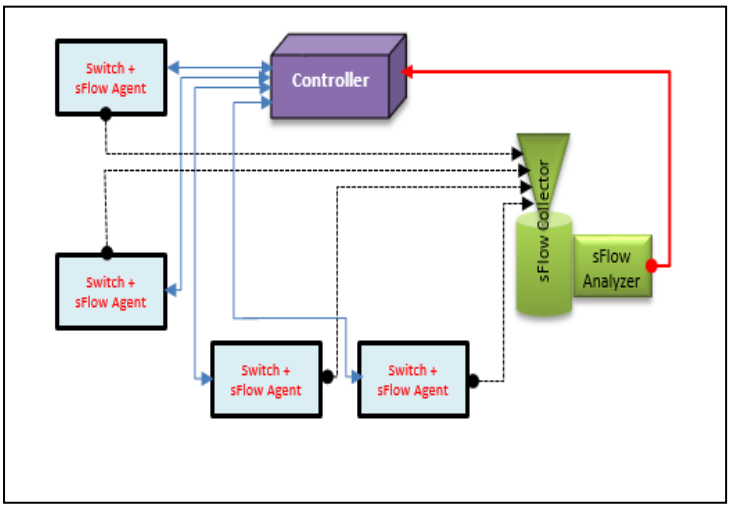

Figure 3. 1. *sFlow system architecture*

sFlow system architecture consists of:

• **The sFlow Agent**: This can either be embedded in a switch or router, or in a standalone probe. It takes samples from packets transiting the network and sends them as datagram to a collector.

• **The sFlow Collector**: The collector serves as a server where sFlow datagrams are collected and stored.

• **The sFlow Analyzer**: It analyses the received/stored datagrams and provides a real-time or historical overview of the network traffic flow. In-depth information and metrics about the network parameters can be obtained from the analyzer and can be used to detect and generate rules to curb irregularities such as DDoS attacks, virus attacks etc., on the network [51].

The sFlow agents use the time-based sampling of the interface counters and the statistical sampling of switched packets. It combines interface counters and flow samples into a sFlow datagram and sends them to the sFlow collector via the UDP protocol. sFlow document is defined in RFC 3176 [52].

Some of the benefits of sFlow can be summarized as follows:

Deployment: sFlow can be easily deployed on an existing network with a simple configuration.

Scalability: It can monitor speeds up to 10Gbps and tens of thousands of flows over several ports. sFlow packet overhead is less than 0.02% for a 10Gb Ethernet link.

Real-Time: sFlow statistics are designed to reflect simultaneously as the traffic streams on the network.

Flexibility: sFlow presents a real-time network overview of the traffic flow, detailed insight into the network patterns, link utilization, and network performance.

Other benefits and applications include real-time congestion management, route profiling, detection, diagnosing, network audit and source tracker etc.

3.2. sFlow Technique for Detecting and Mitigating DDoS Attacks

For every *N* packet transiting the network, an average of one of these packets is sent to the sFlow collector for analysis. These samples are collected from all the

switches on the network by the sFlow agents which are pre-installed on them. Sampling rate and polling intervals are assigned on sFlow to collect the samples periodically.

Sampling rate (S): The number of packets that should be sent for sampling from packets transiting the network.

Polling Intervals (I): This is the time difference between successive sample collections. For example, if S = 10 and I = 5, it means, at every interval of 5 seconds, one out of 10 packets transiting the network will be forwarded to the collector for sampling. Details about the sampling rate and the polling interval used in this study can be found in Section 4.3.

A threshold value, *T*, can also be set to ensure that the number of packets per second, *P*, flowing through the switch/es is below this threshold value. This is done by starting the mitigation script usually written as a REST API in Javascript, Python etc.

The threshold value *T* can be determined by monitoring the rate of normal traffic on the network and the threshold can be set at 10 times the peak value of the expected traffic. Also, a conservative approach (to eliminate false positives) is to determine the level of traffic that would saturate WAN bandwidth or disrupt services and set a threshold to trigger at that level of traffic. Another method is to set the threshold *T* at a value which represents a fraction of the total network bandwidth, e.g., 10%, 20% of the total bandwidth. In this study, *T* is set at 25% of the total network bandwidth.

If the traffic surpasses the threshold, *T*, the sFlow management system triggers an alarm and generates traffic handling rule as defined in the script which will be communicated to the controller. The controller, on the other hand, sends an update on the new forwarding rules (to drop packets from the infected port/s) to the switch/es through OpenFlow protocol. The effect of the attack will soon be minimized and with a further update from the analyzer and the controller, the attack will be eliminated, and the network will be protected.

3.3. sFlow Sampling Algorithm

Figure 3.2 illustrates the sFlow sampling algorithm. When a new packet is transmitted on the network, sFlow agent counts the number of packets and check if the packet is an exempted (symmetric message) one which it should ignore or a sample to be collected. If it is an exempted packet it will wait for the next batch (as defined by **I**) of samples. If not, it will take a quantity (as specified by **S**) and the packet is assigned a source and destination address before being sent to its destination. This is repeated for every polling interval.

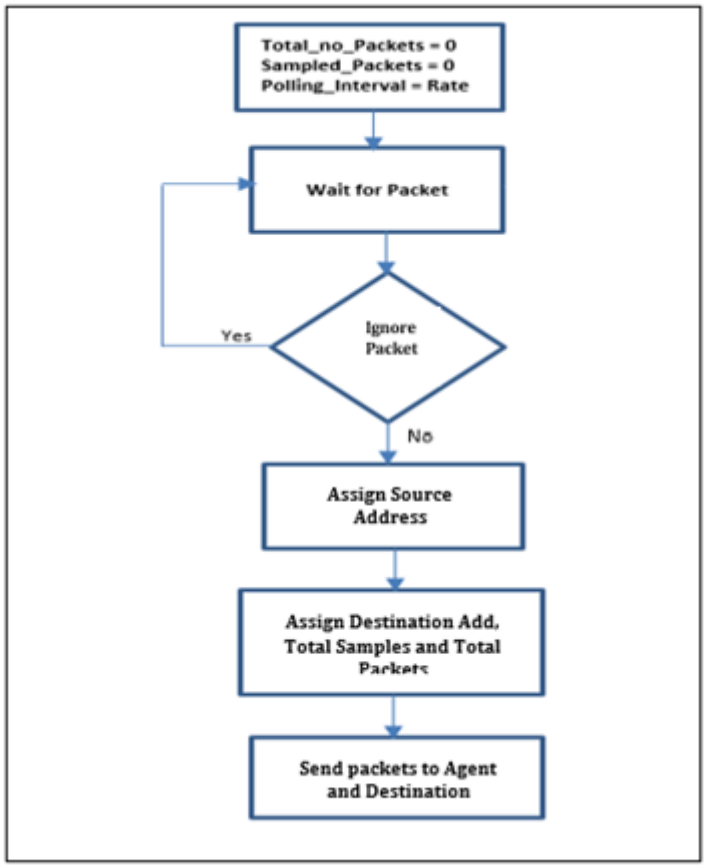

Figure 3. 2. *sFlow sampling algorithm*

The process involved in the detection and mitigation of an attack on the network can be summarized in eight steps as shown in Figure 3.3.

The first step is to categorize the traffic according to their address group. This will enable the identification of both internal and external addresses. The second step is to define flows by naming the packet attributes that are used to group them such as flow keys, ipsource, ipdestination, frames, filter (internal or external), etc.

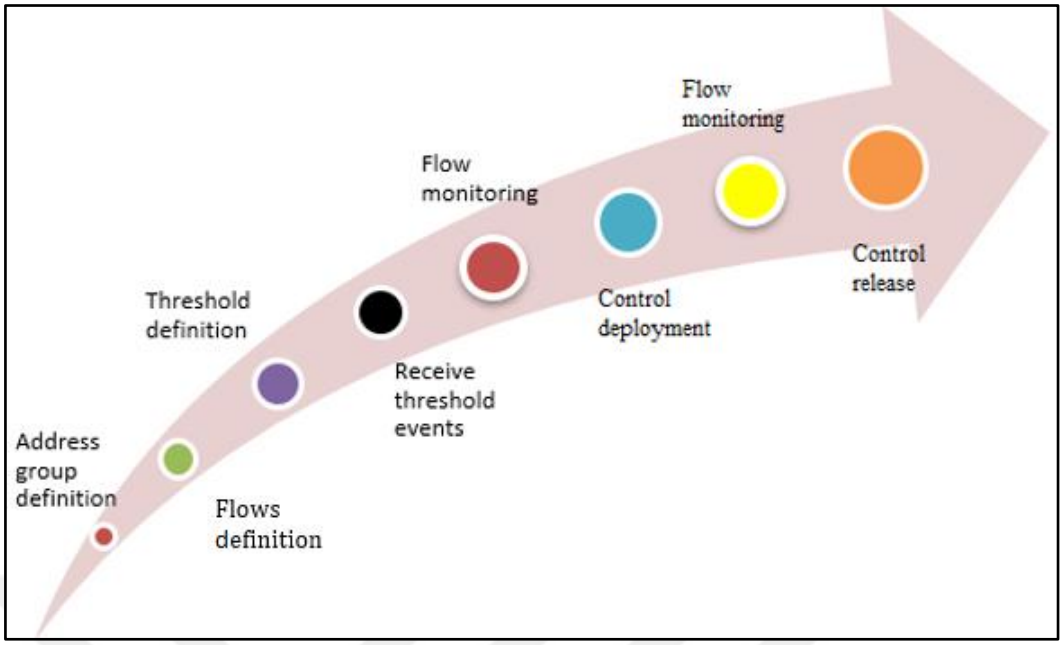

Figure 3. 3. *DDoS attack detection and mitigation process.*

The next step is to define the threshold value on incoming or outgoing traffic flow, it is usually defined in packets per second, e.g., 1,500 packets per second. After this, the threshold events are received using an application poll for events (long polling). It receives the asynchronous notification for new events. The following command requests for the event activities of up to 30 seconds.

```
curl "http://localhost:8008/events/json?eventID=4&timeout=30"
```

The result may look like the following:

```
[{ "agent": "10.0.0.8",
 "dataSource": "4",
 "eventID": 5,
 "metric": "incoming",
 "threshold": 1500,
 "thresholdID": "incoming",
 "timestamp": 1357169369479,
 "value": 2031.149418835524
}]
```

The above event shows that the incoming traffic flow is generating 2031 packets per second. The next step is to monitor the flow using the information retrieved from the event such as metric, data source, agents, etc. The following command identifies the specific flow that exceeded the threshold:

```
curl http://localhost:8008/metric/10.0.0.8/4.incoming/json
```

The result of this command is as follows and it indicates that the flow that exceeded the threshold is coming from an external address 192.168.56.101 to an internal address 10.0.0.1


```
[{ "agent": "10.0.0.8",
 "dataSource": "4",
 "metricName": "incoming",
 "metricValue": 1582.93965044338071,
 "topKeys": [
 { "key": "192.168.56.101,10.0.0.1",
 "updateTime": 1357169662500,
 "value": 1582.93965044338071
}]
```


The next step is to deploy control by dropping traffic flow from the identified address where the surge is coming from. sFlow continues to monitor the status of the network to ensure that that the control has been applied and that the threats have eased. After some time, the deployed control is stopped and the flow table entries blocking the attacks are released.

The full script of this process can be found in Appendix A.

3.4. Internet Protocol Security (IPSec)

Internet Protocol Security (IPSec) is a set of protocols and algorithms that are used to ensure a secure and private communication at the network layer in a public network using cryptographic keys. It is usually set up on the switches or routers and allows transmitted packets to be authenticated, encrypted and encapsulated. In addition, it negotiates by checking the configuration between/among the gateways in the network. If the keys are different during negotiation, the network will not be established [9].

IPSec ensures authentication between the hosts on the network and protects traffic between the gateways or between the gateway and a host. It also supports network-level data integrity, authentication and confidentiality. It protects against denial of service attacks, replay attacks and man in the middle attacks, etc.

3.5. Flow-Based IPSec

In a bid to enhance SDN security, there is a need to introduce IPSec into the network, but, the traditional IPSec is too rigid to be implemented on the SDN

network because of its manual configuration requirements. The flow-based IPSec [10] protocol is an SDN controlled security protocol, which functions as the traditional IPSec with support for SDN services. It supports and creates a secure path for both host-to-gateway and gateway to gateway [11] transmission. Unlike the traditional IPSec, the flow-based IPSec can be programmatically built on the application layer and communicated to the controller through a REST protocol. The controller can then translate the security requirements and implements them on the forwarding devices.

The flow-based IPSec is a controller-based data protection services which allow the protection of data traffic based on a defined security policy. The controller can establish and manage the IPsec security associations by generating the required parameters for key negotiations. Some of the major algorithms IPSec uses for securing the network are shown in Table 3.1.

Table 3. 1. *IPSec key algorithm*

Security Mode	Algorithm
Negotiation	DH group
Authentication	MD5, SHA1
Encryption	DES, 3DES, AES

The security rules do not need to be run on the gateways directly, it is automated by the controller and once it is implemented it does not need to be repeated. This reduces the overload and ensures that the implementation is lighter.

3.6. IPSec Protocol Schemes and Modes

IPSec uses two protocol scheme which is Authentication Header AH [12] and Encapsulation Security Payload ESP [13]. In addition to authentication and data integrity that they both provide, ESP also provides confidentiality on the transmitted data by encapsulating the data payload and/or IP header. The level of encryption will, however, depend on the adopted mode, which can either be IPSec Tunneling or IPSec Transport Mode.

The IPSec Transport Mode encrypts only the data payload while the IP header remains open but IPSec Tunnel mode encrypts both the data payload and the IP header. The tunnel mode wraps the whole encrypted traffic with a new IP header before transmission. Table 3.2 and Figure 3.3 give full details on the IPSec protocols, and modes and how they protect the network and the traffic passing through it.

Table 3. 2. *IPSec protocols and modes*

Protocol / Mode	AH	ESP
Transport	AH Transport Mode Packet	ESP Transport Mode Packet
Tunnel	AH Tunnel Mode Packet	ESP Tunnel Mode Packet

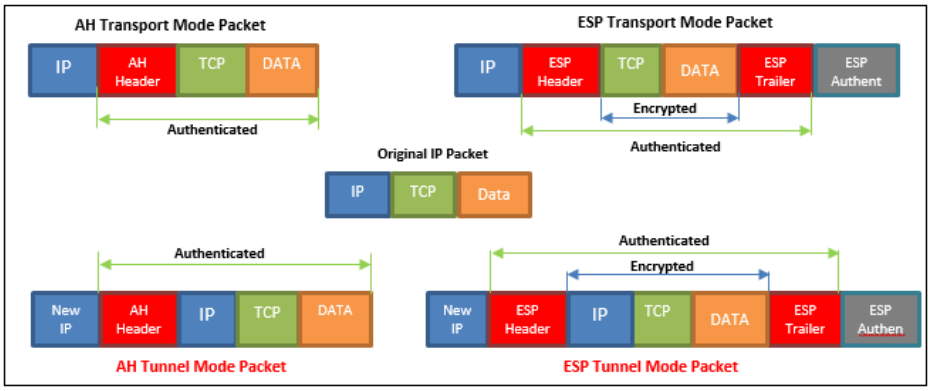

Figure 3. 4. *IPSec modes*

Since SDN allows the programmability of the control layer, both schemes and modes can be deployed and automated by the controller based on the requirement and necessity in the network. This will allow SDN maximize the features associated with each of the protocols. The network administrator can define the rules via the application layer in the SDN network according to the importance attached to the traffic and the type of the network (e.g., Host to Gateway or Gateway to Gateway). Transport mode is good for Host to Gateway while Tunnel mode is more secure for Gateway to Gateway transmission.

4. TEST EMULATION AND RESULTS

This chapter consists of four sections:

- SDN network setup and emulation.
- Attacking the network (infecting the network with some security vulnerabilities).
- Detecting and mitigating the attack.
- Emulation result and evaluation.

4.1. SDN Network Setup

To create or set up an SDN network requires an emulator or a simulator. They are both testbeds for creating prototypes of the SDN network. An emulator replicates or recreates the behaviour of a system. It can replace the original for real use and it is usually used when the importance is about what a system does and not how it does it. A simulator on the other hand focuses on modelling the components of a system. It is usually used when the preference is about how a system does something. In SDN, the aim is usually the re-creation or replication of a real-life network prototype before real deployment.

An example of an SDN simulator is Estinet [47, 53, 54] while an example of an SDN emulator is Mininet [55, 56]. Mininet is the most well-known software tool for SDN network research as noted by the Open Network Summit (ONS) conference in 2013 [35]. Mininet is adopted as the testbed for setting up the SDN network in this study.

4.1.1. Mininet

Mininet is an emulator software tool, which allows an SDN experiment to be emulated on a single computer. It uses lightweight process-based virtualization (Linux network namespaces and Linux container architecture) to run many hosts and switches on a single OS kernel. It can create kernel or user-space OpenFlow switches, controllers and hosts to communicate over the emulated network and connects switches and hosts using virtual Ethernet pairs. It simplifies the initial development, debugging, testing, and deployment process. New network applications can first be developed and tested on an emulator before moving into the actual operational infrastructure [22].

Mininet is an open source emulator software which uses real devices that run real applications on a virtualized hardware (e.g., Oracle VM VirtualBox) on a single computer [47]. This is like performing experiments on the real network at a lower cost, however, it may be slower than the real network.

It has the capabilities that enable researchers or network programmers to create SDN prototype in a simple manner and to interact, customize and share it. It also provides a smooth path for running the prototype on hardware [48].

Mininet enables complex topology testing, without the need to wire up a physical network. It includes a command-line interface (CLI) that is topology-aware and OpenFlow-aware, for debugging or running network-wide tests. Mininet can be started out of the box without any programming, but it also provides a straightforward and extensible Python API for network creation and experimentation.

Rather than being a simulation tool, Mininet is an emulation environment, which runs real, unmodified code, including application code, OS kernel code, and control plane code (both OpenFlow controller code and OpenvSwitch code). It is easy to install and is available as a pre-packaged **virtual machine** (**VM**) image that runs on VMware or VirtualBox for Mac/Windows/Linux with OpenFlow tools already installed.

4.1.2. Network setup in Mininet

The system specification used in setting up the SDN network in this study is given below:

A Casper PC with **Processor** Intel (R) Core (TM) i5 CPU @ 2.50GHz

RAM: 8.00 GB (7.5GB usable)

Host operating system: Windows 10 (64-bit)

Guest Operating System: Ubuntu 32bits with 1024MB of RAM

Virtual Machine: Oracle VM Virtual Box version 5.2.6

Emulator: MININET version 2.2

Controller: Floodlight (see Section 2.1.2).

4.1.2.1. Installing Mininet

Mininet can be downloaded from http://mininet.org/download. It is a compressed archive containing two files. After download, the file is decompressed and the mininet ovf (open virtualization format) file is imported into the virtual machine (VirtualBox).

To launch Mininet, the VirtualBox manager on the host system is started and the necessary settings are done on Mininet. Figure 4.1 shows Mininet in the VirtualBox ready for launch.

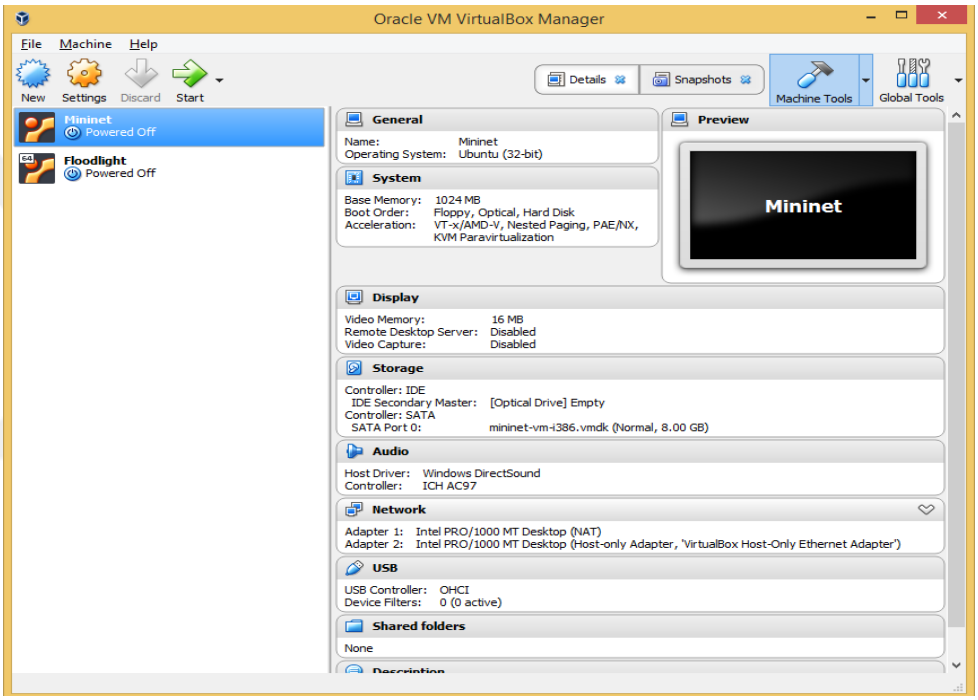

Figure 4. 1. *Mininet imported to VirtualBox*

After launching Mininet, a topology (ring, tree, linear etc.) can be created to emulate a real network environment. This can be achieved by running essential commands on Mininet. In this study, the tree topology was adopted and the simple steps for creating the network are given below.

Step 1: Launch Mininet

Step 2: Enter the username and the password, both are "mininet".

Step 3: Run the following command to create the network topology. The command should be run with root privileges using *sudo* command

```
sudo mn --controller=remote,ip=127.0.0.1,port=6653 --topo=tree,3
```

The command in Step 3 above setups the SDN network topology on Mininet by creating virtual switches, hosts and links. OpenVSwitch (OVS) was used as the implementation for OpenFlow switches [49, 57] and floodlight as the controller. Figure 4.2 gives the description of the emulated network on the Mininet interface as described above.

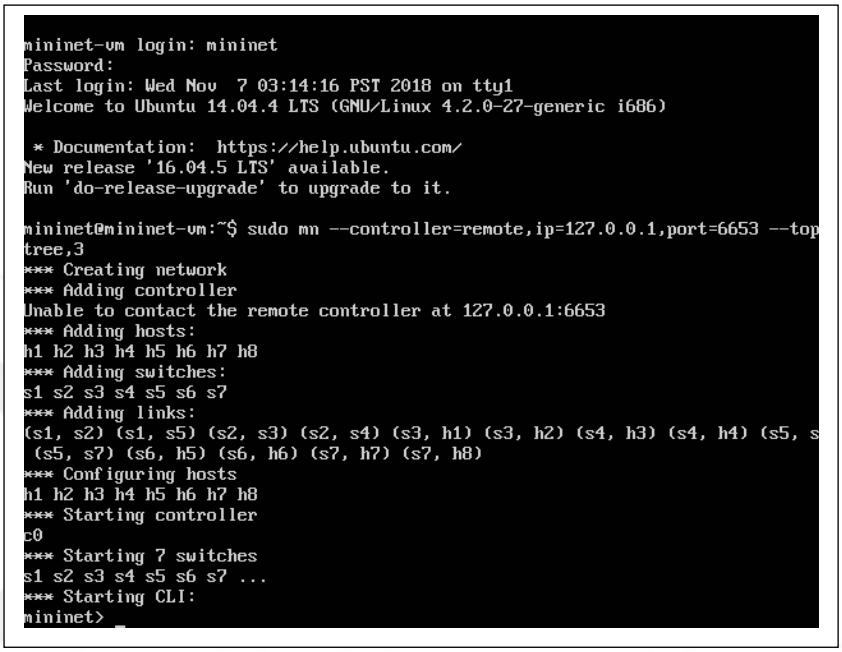

Figure 4. 2. *Emulated SDN network topology*

However, Mininet does not save topology and therefore, the process needs to be repeated whenever the experiment is to be carried out. To prevent this, a graphical user interface (GUI) called MiniEdit was created for Mininet.

4.1.2.2. MiniEdit

MiniEdit is a GUI editor for Mininet and it is embedded in Mininet. It is created to demonstrate the extension of Mininet. MiniEdit can build and configure topologies including their network elements. In addition, it can save and run the topology for emulation.

To start MiniEdit, Mininet must be running on the VirtualBox and the following command is executed to call or open the MiniEdit GUI.

```
sudo ~/mininet/examples/miniedit.py
```

Before running the above command in Mininet, the Xming server and PuTTY must have been started.

Xming is an open source X-Windows terminal emulator server that runs on the Windows operating system. Xming allows GUI programs from Linux to be displayed on a Window's running machine. Xming performs no function on a windows OS unless it is used along with the windows program PuTTY (Figure 4.3) with the X11 forwarding enabled on it.

PuTTY is a Windows terminal program that enables connection to a Linux/UNIX server to allow CLI work to be done on a Linux/UNIX server from Windows PCs.

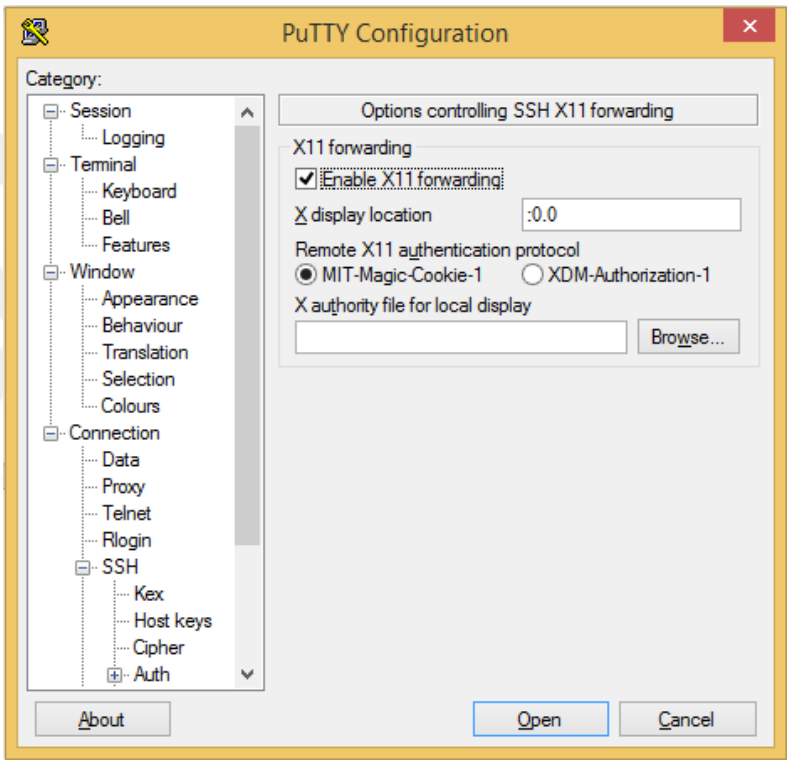

Figure 4. 3. *PuTTY with X11 enabled*

PuTTY establishes a terminal session with the Linux server while Xming allows Linux GUIs to be displayed on the Windows system.

4.1.2.3. Customized emulated network topology

A customize topology can be created using MiniEdit or python API, an example of this is given below. MiniEdit generated the emulated network used in this study as described in Section 4.1.2.1. This is shown in Figure 4.4.

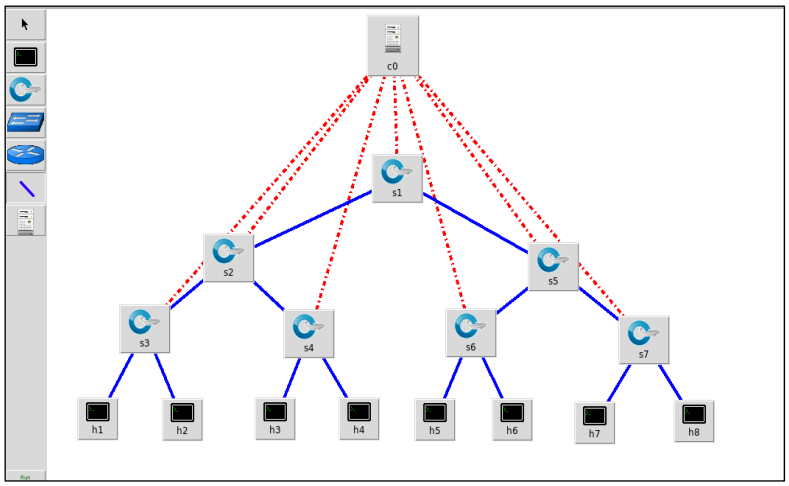

Figure 4. 4. *Emulated network In MiniEdit*

MiniEdit can run or stop the emulation of the created topology in Mininet and it can save the topology for future use. It can also configure the network parameter for each of the network elements (controller, switch or host).

The emulated network is a tree topology consisting of one controller, seven OpenVSwitches and eight hosts. Each switch is connected (logically) to the controller (c0) and some switches are connected to other switches. However, the switches are configured to avoid redundancy.

The emulated network described above can also be created using the python API (see Appendix B).

4.2. Emulating an Attack on the Network

To test the efficiency of the proposed method described in Chapter 3, there is a need to infect the network with flood attacks, this will help to ascertain the impact of the security threats and the effectiveness of the adopted mitigation technique. This will be done in different phases, the SDN network will be attacked with ICMP flood attacks. The ICMP flood attack can be generated using the "*ping -f*" command followed by the address of the victim. Details can be found in Section 4.4.

4.3. Detection and Attack Mitigation

The proposed architecture for securing the network employs the sFlow technology and the IPSec protocol to detect, mitigate and protect against threats in the SDN network. IPSec encapsulates the links between the hosts and the switches by creating a tunnel path for packets transmission. All packets in transit are

protected from unauthorized users from tampering or modification, this can be seen in Figure 4.5. This will enhance the network security by protecting against threats such as MITM and replay attacks while sFlow detects and protects the network against DDoS attacks.

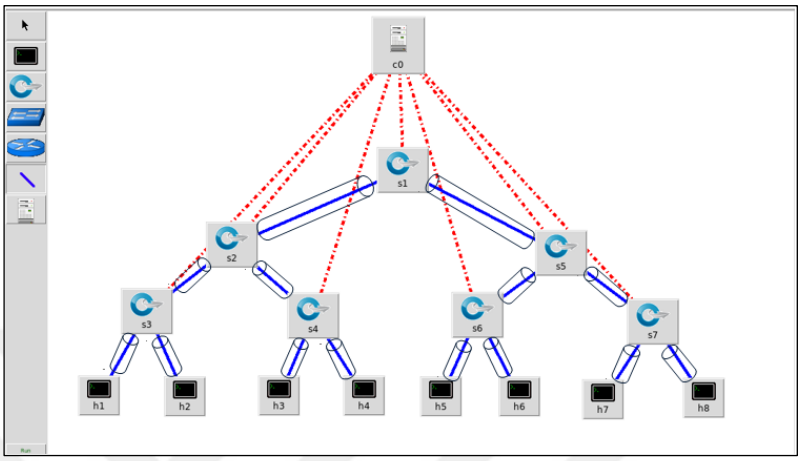

Figure 4. 5. *A secure SDN architecture*

The sFlow agents (see Section 3.1) will be installed on each of the switches and packet samples will be collected and sent for analysis and these are monitored in real-time. In this setup, there are seven switches, therefore, the agents will be installed on each of them i.e., switch S1 to S7. The following command does this installation.

```
sudo ovs-vsctl -- --id=@sflow create sflow agent=eth0 target =\"127.0.
0.1:6343\" sampling=10 polling=5 -- -- set bridge s1 sflow=@sflow
```

The SDN controller (floodlight) and sFlow are built outside of Mininet but all are built on a single virtual machine, this is depicted in Figure 4.6.

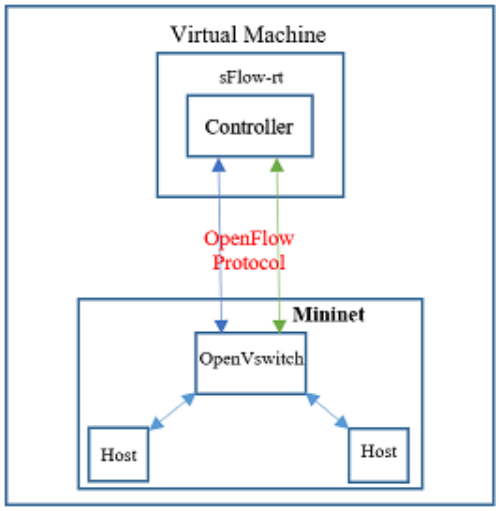

Figure 4. 6. *Proposed system architecture*

This is the proposed SDN architecture used in the detection and mitigation of the effects of the DDoS attack on the network.

The sFlow datagrams from the network devices are continuously sent to the sFlow analyzer which converts them into useful metrics and is accessible through a REST API. Anomalies are detected through the REST API by detecting changes in network traffic. sFlow generates rules through the API for mitigation which are sent to the controller. The controller then reconfigures the network elements and their forwarding rules and behaviours through the OpenFlow protocol which is responsible for carrying information between the control plane and forwarding plane.

4.4. Result and Evaluation

After setting up the network on Mininet and installing sFlow agents on each of the switches as explained in Section 4.3, the tests were carried out in three stages and sFlow collected samples for analysis. Host (h1) was set as the victim while other hosts (h2 - h8) were compromised to attack host (h1) by generating flood attacks and sending them to it.

Stage 1: Host (h1) was pinged from other hosts on the network when there was no attack on it and it was proved reachable. The graph in Figure 4.7 shows that the traffic rate is less than 2000pps while there was no attack on the host (h1).

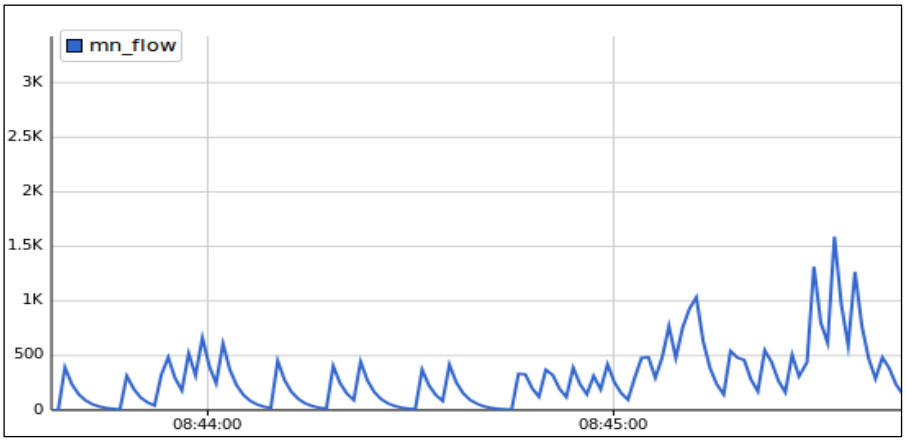

Figure 4. 7. *Testing for reachability on host h1*

Stage 2: ICMP flood attacks were launched from other hosts on the network to host (h1). This generated a huge amount of traffic that could overwhelm the network resources. The continuous request was sent to the controller for the

inclusion of missing packets in the flow tables. Shortly, the controller became so busy that it could not respond to requests from other hosts on the network and it started dropping packets from eligible users or hosts. This event was captured using Wireshark as displayed in Figure 4.8.

Source	Destination	Protocol	Length	Info
10.0.2.15	212.175.41.102	ICMP	136	Destination unreachable (Port unreachable)
10.0.2.15	193.140.21.38	ICMP	136	Destination unreachable (Port unreachable)
10.0.2.15	212.175.41.103	ICMP	136	Destination unreachable (Port unreachable)
10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x1970, seq=1/256, ttl=64 (no response found!)
10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x1970, seq=2/512, ttl=64 (no response found!)
10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x1970, seq=3/768, ttl=64 (reply in 4768)
10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0x1970, seq=3/768, ttl=64 (request in 4767)
10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x1970, seq=3/768, ttl=64 (no response found!)
10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x1970, seq=3/768, ttl=64 (no response found!)
10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x1970, seq=3/768, ttl=64 (no response found!)
10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x1970, seq=3/768, ttl=64 (no response found!)
10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x1970, seq=3/768, ttl=64 (no response found!)
10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x1970, seq=3/768, ttl=64 (no response found!)

Figure 4. 8. *Host h1 unreachable after attack*

However, sFlow was not initiated to detect or mitigate the surge in the traffic transiting the network in this stage. This was done to determine the effect of the attack on the network. Figure 4.9 shows the effect of the ICMP flood attack on the network. The volume of the traffic soared to approximately 2M pps.

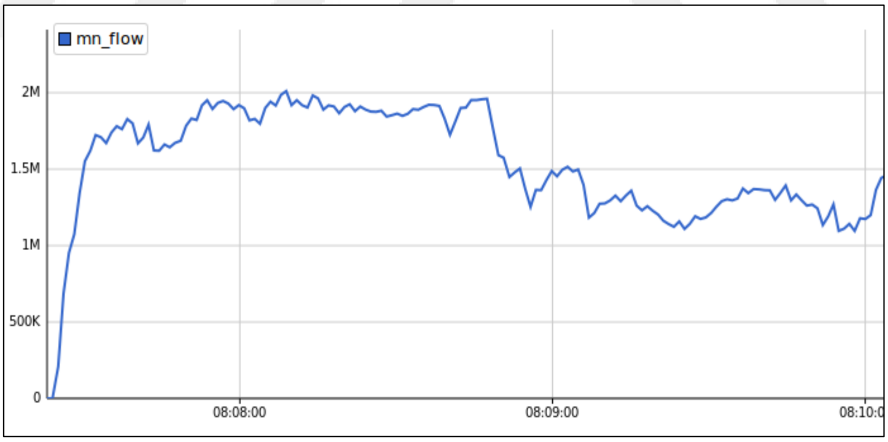

Figure 4. 9. *ICMP flood attack*

Stage 3: At this stage, sFlow was launched to detect and mitigate the effect of the attack on the network. A threshold of 700,000 pps was set on sFlow at a sampling rate ***S*** of 10 (1 in every 10 packets should be sent for analysis) and a polling interval ***I*** of 5 (samples should be taken every 5 seconds). It can be seen from Figure 4.10 that after a sharp rise was discovered by the analyzer, with the traffic surging to almost 1.7M pps, sFlow quickly generates the handling rule for the controller and it informs the switch/es to update its/their flow table/s; then packets were dropped

from infected ports and the attack was mitigated within a period of ~10-15 seconds of implementation and the traffic went below the threshold line.

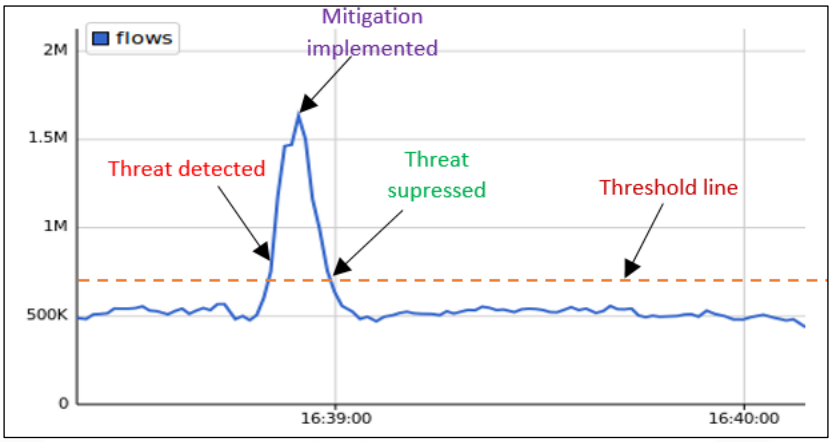

Figure 4. 10. *Attack detection and mitigation*

5. CONCLUSION AND RECOMMENDATION.

5.1. Conclusion

This study was aimed to present how the SDN network security can be enhanced by implementing sFlow technology combined with IPSec protocol. The sFlow technology is a real-time analyzer that can be embedded with the controller and its packet overhead consumption is negligible about 0.02% of a 10 Gb Ethernet link yielding a very flexible and light solution. Its response detection and control time is approximately 15 secs which makes it a probable solution to mitigate the threats posed by DDoS attacks on the SDN network resources.

The flow-based IPSec protocol which is a special form of the Virtual Private Network (VPN) traditional IPSec can also be integrated into the network to ensure confidentiality, integrity and authenticity on the network. The IPSec protocol establishes a tunnel for the packet in transit protecting them against MITM attacks, IP spoofing, etc.

Comparing with the other methods discussed in Chapter 2 of this manuscript, the proposed method outperforms them as it detects threats on the network in real-time and helps the controller to take actions and decide on dropping packets from infected ports while ensuring that the network resources are not subdued or overwhelmed.

Most of the methods in the literature have large packet overheads that consume a notable fraction of the overall bandwidth of the controller. In contrast, the method used in this work does not install any extra device on the controller. It only generates rules to be implemented by the controller and communicates this through the existing northbound interface. This ensures easy deployment and scalability on the network.

The implemented security measure protects the network against threats and ensures the availability of the network to authorized users. In addition, it prevents both internal and/or external attacks and abuses such as DDoS attacks that could be generated by eligible or unauthorized users on the network. With the measures taken, confidentiality, the integrity of the data, and availability of the network resources are achieved.

5.2. Future Work

The future of networking will depend on SDN, with automation being the main interest of the network operators, providers and administrators. For example, Google has recently launched the fourth pillar of its SDN strategy (Espresso) with the purpose of bringing SDN to the public internet [58]. Also, large data centres across the globe have increased the deployment of SDN on their network. However, security is remaining a major topic of concentration in SDN as it usually does in networking.

As a continuation of this study, the major recommendation is to apply a robust machine learning approach that can distinguish a flash crowd (surge in legitimate traffic) from a DDoS attack on the SDN network. The approach should be able to learn and use the legitimate traffic arrival pattern, flow correlation coefficient, packet information distance and other probability metrics in the network to differentiate between flash crowd from attacks.

In addition, the quest for a more secure network usually has its share on the quality of service (QoS) on the network, with most security methods having an overhead that impedes the QoS. Therefore, while deploying or proposing a novel method for the enhancement of SDN security in future, the QoS should be considered and not be traded-off.

Also, despite the attractive benefits and advantages of SDN over existing network solutions which include its ability to respond to the demand of the 5G network for personalized services including IoT, more attention needs to be given to its performance, scalability, interoperability and security of the network.

REFERENCES

[1] M. Vizv'ary, "Mitigation of DDoS Attacks in Software Defined Networks," *Advisor: doc. RNDr. V 'aclav Raacansk'y, CSc*.

[2] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," (in English), *Acm Sigcomm Computer Communication Review*, vol. 38, no. 2, pp. 69-74, Apr 2008.

[3] Y. Li, D. Zhang, J. Taheri, and K. Li, "SDN components and OpenFlow."

[4] R. Kloti, V. Kotronis, and P. Smith, "Openflow: A security analysis," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, 2013, pp. 1-6: IEEE.

[5] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-defined networking (SDN): Layers and architecture terminology," 2070-1721, 2015.

[6] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2317-2346, 2015.

[7] S. J. Vaughan-Nichols, "OpenFlow: The next generation of the network?," *Computer*, no. 8, pp. 13-15, 2011.

[8] C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks*, vol. 44, no. 5, pp. 643-666, 2004.

[9] B. H. Lawal and N. At, "Improving Software Defined Network Security via sFLow and IPSec Protocol," *Anadolu Üniversitesi Bilim Ve Teknoloji Dergisi A-Uygulamalı Bilimler ve Mühendislik,* pp. 1-1, 2018.

[10] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, 2007, vol. 37, no. 4, pp. 1-12: ACM.

[11] A. L. V. Caraguay, L. I. B. Lopez, and L. J. G. Villalba, "Evolution and challenges of software defined networking," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, 2013, pp. 1-7: IEEE.

[12] h. w. o. o. s. Open Networking Foundation. (2017, December 18). Available: https://www.opennetworking.org/sdndefinition

[13] W. Braun and M. Menth, "Software-defined networking using OpenFlow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302-336, 2014.

[14] E. Haleplidis, "Overview of RFC7426: SDN Layers and Architecture Terminology."

[15] P. A. Porras, S. Cheung, M. W. Fong, K. Skinner, and V. Yegneswaran, "Securing the Software Defined Network Control Layer," in *NDSS*, 2015.

[16] Y. Jimenez, C. Cervello-Pastor, and A. J. Garcia, "On the controller placement for designing a distributed SDN control layer," in *Networking Conference, 2014 IFIP*, 2014, pp. 1-9: IEEE.

[17] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the Datacenter," in *HotNets*, 2009.

[18] S. Kaur, J. Singh, and N. S. Ghumman, "Network programmability using POX controller," in *ICCCS International Conference on Communication, Computing & Systems, IEEE*, 2014, vol. 138.

[19] D. Suh, S. Jang, S. Han, S. Pack, T. Kim, and J. Kwak, "On performance of OpenDaylight clustering," in *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*, 2016, pp. 407-410: IEEE.

[20] R. K. Arbettu, R. Khondoker, K. Bayarou, and F. Weber, "Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers," in *Telecommunications Network Strategy and Planning Symposium (Networks), 2016 17th International*, 2016, pp. 37-44: IEEE.

[21] H. Akcay and D. Yiltas-Kaplan, "Web-Based User Interface for the Floodlight SDN Controller," *Int. J. Advanced Networking and Applications,* vol. 8, no. 05, pp. 3175-3180, 2017.

[22] S. Azodolmolky, "Software Defined Networking with OpenFlow," *Packt Publishing Ltd.*, Book no. ISBN 978-1-84969-872-6, p. 152, Oct 25 2013.

[23] T. Zou, H. Xie, and H. Yin, "Supporting software defined networking with application layer traffic optimization," ed: Google Patents, 2016.

[24] N. Foster *et al.*, "Frenetic: A network programming language," *ACM Sigplan Notices,* vol. 46, no. 9, pp. 279-291, 2011.

[25] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing Software Defined Networks," in *NSDI*, 2013, vol. 13, pp. 1-13.

[26] A. Voellmy, H. Kim, and N. Feamster, "Procera: a language for high-level reactive network control," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 43-48: ACM.

[27] S. Sezer *et al.*, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Communications Magazine,* vol. 51, no. 7, pp. 36-43, 2013.

[28] M. Vahlenkamp, "Design and Implementation of a Software-Defined Approach to Information Centric Networking," *Hamburg University of Applied Science, Germany,* 2013.

[29] P. K. Agyapong, M. Iwamura, D. Staehle, W. Kiess, and A. Benjebbour, "Design considerations for a 5G network architecture," *IEEE Communications Magazine,* vol. 52, no. 11, pp. 65-75, 2014.

[30] L. R. Prete, C. M. Schweitzer, A. A. Shinoda, and R. L. S. de Oliveira, "Simulation in an SDN network scenario using the POX Controller," in *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, 2014, pp. 1-6: IEEE.

[31] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol version 1.2," 2070-1721, 2008.

[32] O. Ogundile, B. Lawal, and O. Osanaiye, "A Secured Voice over Internet Protocol (VoIP) Setup Using MiniSipServer," *International Journal of Scientific and Engineering Research*.

[33] R. Kandoi and M. Antikainen, "Denial-of-service attacks in OpenFlow SDN networks," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, 2015, pp. 1322-1326: IEEE.

[34] A. Hussein, I. H. Elhajj, A. Chehab, and A. Kayssi, "SDN security plane: An architecture for resilient security services," in *Cloud Engineering Workshop (IC2EW), 2016 IEEE International Conference on*, 2016, pp. 54-59: IEEE.

[35] N. DuPaul. (2018, November 29). *MAN IN THE MIDDLE (MITM) ATTACK*.

[36] https://us.norton.com/internetsecurity-wifi-what-is-a-man-in-the-middle-attack.html. (2018, November).

[37] P. Ahmad, S. Jacob, and R. Khondoker, "Security Analysis of SDN Applications for Big Data," in *SDN and NFV Security*: Springer, 2018, pp. 39-55.

[38] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 55-60: ACM.

[39] M. Şimşek and A. Şentürk, "Fast and lightweight detection and filtering method for low-rate TCP targeted distributed denial of service (LDDoS) attacks," *International Journal of Communication Systems*, p. e3823, 2018.

[40] A. R. Abdou, P. C. van Oorschot, and T. Wan, "Comparative Analysis of Control Plane Security of SDN and Conventional Networks," *IEEE Communications Surveys & Tutorials*, 2018.

[41] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623-654, 2016.

[42] L. Dridi and M. F. Zhani, "SDN-guard: Dos attacks mitigation in SDN networks," in *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, 2016, pp. 212-217: IEEE.

[43] L. Ertaul and K. Venkatachalam, "Security of Software Defined Networks (SDN)," in *Conf. Wireless Networks*, 2017.

[44] K. Kaur, K. Kumar, J. Singh, and N. S. Ghumman, "Programmable firewall using software defined networking," in *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on*, 2015, pp. 2125-2129: IEEE.

[45] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "DDoS attack protection in the era of cloud computing and software-defined networking," *Computer Networks*, vol. 81, pp. 308-319, 2015.

[46] S. Frankel and S. Krishnan, "Ip security (ipsec) and internet key exchange (ike) document roadmap," 2070-1721, 2011.

[47] S.-Y. Wang, "Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet," in *Computers and Communication (ISCC), 2014 IEEE Symposium on*, 2014, pp. 1-6: IEEE.

[48] F. Keti and S. Askar, "Emulation of software defined networks using mininet in different simulation environments," in *Proceedings of the 2015 6th International Conference on Intelligent Systems, Modelling and Simulation. IEEE Computer Society*, 2015.

[49] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, p. 19: ACM.

[50] B. H. Lawal and N. At, "Real-time detection and mitigation of distributed denial of service (DDoS) attacks in software defined networking (SDN)," in *2018 26th*

Signal Processing and Communications Applications Conference (SIU), 2018, pp. 1-4: IEEE.

[51] blog.sflow.com, "controlling-large-flows-with-openflow," ed, 2018.

[52] P. P. a. M. Lavine. (July 2004, February 8). *sFlow Version 5*.

[53] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, "EstiNet openflow network simulator and emulator," *IEEE Communications Magazine*, vol. 51, no. 9, pp. 110-117, 2013.

[54] S. Y. Wang and H. Kung, "A simple methodology for constructing extensible and high-fidelity TCP/IP network simulators," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 1999, vol. 3, pp. 1134-1143: IEEE.

[55] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2014, pp. 1-6: IEEE.

[56] J. Yan and D. Jin, "Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015, p. 27: ACM.

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

[58] L. Hardesty. (2017, 11/28/2018). *Google Brings SDN to the Public Internet*.

Appendix A

Detection and Mitigation Script

```
import requests
import json
groups = {
  'external': ['0.0.0.0/0'],
  'internal': ['10.0.0.0/1']
}
flows = dict(keys='ipsource,ipdestination', value='frames', filter='so
urcegroup= external&destinationgroup=internal')
threshold = {
  'metric': 'incoming',
  'value': 700000
}
target = 'http://localhost:8008'
r = requests.put(target + '/group/json', data = json.dumps(groups))
r = requests.put(target + '/flow/incoming/json', data = json.dumps(flo
ws))
r = requests.put(target + '/threshold/incoming/json', data = json.dump
s(threshold))
eventurl = target + '/events/json?maxEvents=10&timeout=60'
eventID = -1
while 1 == 1:
  r = requests.get(eventurl + '&eventID=' + str(eventID))
if r.status_code != 200: break
events = r.json()
if len(events) == 0: continue
eventID = events[0]["eventID"]
for e in events:
  if 'incoming' == e['metric']:
  r = requests.get(target + '/metric/' + e['agent'] + '/' + e['dataSou
rce'] + '.' + e['metric'] + '/json')
metric = r.json()
if len(metric) > 0:
  print metric[0]["topKeys"][0]["key"]
```

After running the script, the following output was generated in few se conds detecting large flow on the network and its source address: 192. 168.56.101,10.0.0.2,10.0.0.3,10.0.0.4,10.0.0.5,10.0.0.6,10.0.0.7,10.0. 0.8

Appendix B

Customized topology

```
#Mininet customized topology
from mininet.topo import Topo
    #import module

class Test(Topo):
   # create a class
            def _init_(self):
            # define the constructor"
            #initialized topology
            Topo._init_(self)

            #Add switches
            s1 = self.addSwitch("s1")
            s2 = self.addSwitch("s2")
            s3 = self.addSwitch("s3")
            s4 = self.addSwitch("s4")
            s5 = self.addSwitch("s5")
            s6 = self.addSwitch("s6")
            s7 = self.addSwitch("s7")

            #Add hosts
            h1 = self.addHost("h1")
            h2 = self.addHost("h2")
            h3 = self.addHost("h3")
            h4 = self.addHost("h4")
            h5 = self.addHost("h5")
            h6 = self.addHost("h6")
            h7 = self.addHost("h7")
            s8 = self.addHost("h8")

            #Add links
            self.addLink(s1,s2)
            self.addLink(s1,s5)
            self.addLink(s2,s3)
            self.addLink(s2,s4)
            self.addLink(s3,h1)
            self.addLink(s3,h2)
            self.addLink(s4,h3)
            self.addLink(s4,h4)
            self.addLink(s5,s6)
            self.addLink(s5,s7)
            self.addLink(s6,h5)
            self.addLink(s6,h6)
            self.addLink(s7,h7)
            self.addLink(s7,h8)

topos = {'Test': (lambda: test()
```

RESUME

Name–Surname	: Babatunde Hafis LAWAL
Language	: English and Turkish
Birth Place and Year	: Lagos,Nigeria/1986
Email	: lawal.hafisbabatunde@gmail.com

EDUCATION

2016 – 2019 Eskisehir Technical University, Graduate School of Science, Electrical and Electronics Engineering, Telecommunication.

2009 – 2013 Tai Solarin University of Education, Nigeria. Department of Physics and Telecommunication.

WORK EXPERIENCE

2014 - 2015 Laplace Technologies Ltd, Nigeria. Radio Network Optimization Engineer.

2013 - 2014 Ministry of Information, Enugu, Nigeria. Public Research & Statistics (PRS) Officer.

2011 - 2012 Tai Solarin University Staff Secondary School, TASUESS, Mathematics Teacher (Teacher trainee).

2010 – 2011 Muhri International Television, MITV, Masters Control room operator (Internship).

CONFERENCE PRESENTATION

- Lawal B. H, At N. (2018). Real-Time Detection and Mitigation of Distributed Denial of Service (DDoS) Attacks in Software Defined Networking (SDN), Izmir, Turkey. IEEE Xplore. 26th IEEE Signal Processing and Communications Applications Conference, 10.1109/SIU.2018.8404674, page 1 – 4.

JOURNAL PUBLICATIONS

- Lawal B. H, At N. (2018). Improving Software Defined Network Security Via sFlow and IPSec Protocol, Turkey. Anadolu University Journal of Science and Technology A- Applied Sciences and Engineering, Turkey, 10.18038/aubtda.421939, page 1-10.
- Ogundile O.O, Lawal B.H, Osanaiye O.A (2012). A Secured Voice over Internet Protocol (VoIP) setup Using MiniSipServer, International Journal of Scientific and Engineering Research, Volume 3, Issue 11, ISSN 2229-5518, Page 1086-1093.

AWARDS

2016 Graduate Studies, Turkish Government Scholarship, Turkey.

2014 Best Graduating Student in the College of Science and Information Technology (COSIT), Tai Solarin University of Education, Nigeria.

2014 Best Graduating Student in the Department of Physics and Telecommunication, Tai Solarin University of Education, Nigeria.

2012 Most Brilliant (Male) Student in the Department of Physics and Telecommunication, TITSA, TASUED Chapter, Nigeria.