EVALUATING THE EFFECTIVENESS OF SDN-BASED IOT SECURITY

Masters's Thesis

Yousef Ibrahim Abdalla MOHAMED

Eskişehir 2025

EVALUATING THE EFFECTIVENESS OF SDN-BASED IOT SECURITY

Yousef Ibrahim Abdalla MOHAMED

MASTER'S THESIS

Department of Electrical and Electronics Engineering

Programme in Telecommunications

Supervisor: Prof. Dr. Nuray AT

Eskişehir

Eskişehir Technical University

Institute of Graduate Programs

July 2025

FINAL APPROVAL FOR THESIS

This thesis titled EVALUATING THE EFFECTIVENESS OF SDN-BASED IOT SECURITY has been prepared and submitted by Yousef Ibrahim Abdalla MOHAMED in partial fulfillment of the requirements in “Eskişehir Technical University Directive on Graduate Education and Examination” for the Degree of Master's in Electrical and Electronics Engineering Department has been examined and approved on 04/07/2025.

Committee Members		**Title, Name and Surname**	**Signature**
Member	:	Prof. Dr. Nuray AT	
Member	:	Prof. Dr. Hanife APAYDIN ÖZKAN	
Member	:	Asst. Prof. Dr. İlker ÖZÇELİK	

Prof. Dr. Harun BÖCÜK
Director of the Institute of Graduate Programs

04/07/2025

SUPERVISOR APPROVAL

Masters's student Yousef Ibrahim Abdalla MOHAMED, whom I supervise, has completed this thesis titled EVALUATING THE EFFECTIVENESS OF SDN-BASED IOT SECURITY. According to my inspections, the work is scientifically and ethically appropriate for the student to the thesis defense exam.

Supervisor

Prof. Dr. Nuray AT

ABSTRACT

EVALUATING THE EFFECTIVENESS OF SDN-BASED IOT SECURITY

Yousef Ibrahim Abdalla MOHAMED

Department of Electrical and Electronics Engineering

Programme in Telecommunications

Eskişehir Technical University, Institute of Graduate Programs, July 2025

Supervisor: Prof. Dr. Nuray AT

The exposure of the Internet of Things (IoT) in the last years has strongly affected everything around us in our life by providing innovations like smart farming, smart homes, and remote healthcare services, among others. As the quantity of these devices and their uses increases rapidly, the significant risk posed by the extensive network of interconnected devices calls for effective intrusion detection and prevention methods. In this study, we propose using Software Defined Networks (SDN) in smart homes systems to prevent cyber-attacks like Distributed Denial of server Attacks (DDoS), Man in The Middle Attack (MITM), etc. The proposed method uses the firewall at the SDN controller and Static IP addresses to register the home devices in the controller to prevent any intruder to connect to the network. The network structure was constructed and executed on Mininet, operating on a Virtual Machine (VM), in order to assess and scrutinize the efficiency of the suggested approach. The proposed method can deny DDoS attacks and inform the administrator that there is an attack on the network from an intruder.

Keywords: Internet of Things (IoT), Software Defined Networks (SDN), Distributed Denial of Service Attacks (DDoS), IP address, Mininet.

ÖZET

SDN TABANLI IOT GÜVENLİĞİNİN ETKİNLİĞİNİN DEĞERLENDİRİLMESİ

Yousef Ibrahim Abdalla MOHAMED

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Telekomünikasyon Bilim Dalı

Eskişehir Teknik Üniversitesi, Lisansüstü Eğitim Enstitüsü, Temmuz 2025

Danışman: Prof. Dr. Nuray AT

Son yıllarda Nesnelerin İnterneti'nin (IoT) yaygınlaşması, akıllı tarım, akıllı evler ve uzaktan sağlık hizmetleri gibi yenilikler sağlayarak hayatımızda olan her şeyi güçlü bir şekilde etkilemiştir. Bu cihazların sayısı ve kullanım alanları hızla arttıkça, birbirlerine bağlı cihazlardan oluşan bu büyük ağların oluşturduğu riskler için etkili saldırı tespit ve önleme yöntemlerine ihtiyaç duyulmaktadır. Bu çalışmada, Dağıtık Hizmet Reddi Saldırıları (DDoS), Ortadaki Adam Saldırısı (MITM) gibi siber saldırıları önlemek için akıllı ev sistemlerinde Yazılım Tanımlı Ağların (SDN) kullanılması önerilmiştir. Önerilen sistemde, herhangi bir saldırganın ağa bağlanmasını önlemek için SDN kontrolcüsünde güvenlik duvarı kullanılmış ve ev cihazları Statik IP adresleri kullanılarak kontrolcüye kaydedilmiştir. Önerilen yaklaşımın verimliliğini değerlendirmek üzere ağ yapısı, sanal makine (VM) üzerinde çalışan Mininet üzerinde kurulmuş ve işletilmiştir. Önerilen yöntem DDoS saldırılarını reddedebilmekte ve ağda bir saldırı olduğunu ağ yöneticisine bildirebilmektedir.

Anahtar Sözcükler: Nesnelerin İnterneti (IoT), Yazılım Tanımlı Ağlar (SDN), Dağıtık Hizmet Reddi Saldırıları (DDoS), Statik IP adresi, Mininet.

ACKNOWLEDGEMENTS

I would like to sincerely thank Prof. Dr. Nuray AT for her consistent encouragement, insightful guidance, positive energy, understanding, and meaningful input throughout the course of my studies and research. Her support has been essential for the successful finish of this project, and I am genuinely grateful for the chance to have collaborated under her guidance.

I am truly thankful to the Turkish government and the Presidency for Turks Abroad and Related Communities (YTB) for providing the scholarship that enabled this academic journey. Without their generous assistance, this accomplishment would not have been possible.

Ultimately, I express my deepest gratitude to my parents, whose unwavering love, support, and prayers have been the bedrock of all my achievements. Their confidence in me has been a motivating factor throughout this journey.

Yousef Ibrahim Abdalla MOHAMED

04/07/2025

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 Eskişehir 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.

Yousef Ibrahim Abdalla MOHAMED

CONTENTS

Page

HEADER PAGE……I

FINAL APPROVAL FOR THESIS II

SUPERVISOR APPROVAL III

ABSTRACT IV

ÖZET V

ACKNOWLEDGEMENTS VI

STATEMENT OF COMPLIANCE WITH ETHICAL PRINCIPLES AND RULES VII

CONTENTS VIII

LIST OF TABLES X

LIST OF FIGURES XI

1. INTRODUCTION 1

1.1. Problem Statement 2

1.2. Existing Solutions and Related Work 3

1.3. Research Contributions 4

2. SDN OVERVIEW 5

2.1. SDN Architecture 5

2.1.1. SDN Open Virtual Switches (OVS) 7

2.1.2. OpenFlow protocol 8

2.2. Controller Overview 9

2.3. SDN Benefits 10

2.4. SDN Security Challenges 11

2.4.1. Data plane attacks 12

2.4.2. Control plane attacks 14

2.4.3. Application plane attacks 16

2.5. Traditional Network versus Software Defined Networks 17

3. METHODOLOGY 18

3.1. Oracle VM Virtual Box 18
3.2. Ubuntu 22.04 20
3.3. Mininet 20
3.4. Wireshark 21
3.5. DDoS Attack tool 23
3.6. Floodlight REST API 24
3.7. IPsec 25
4. EMULATION AND PROOF OF CONCEPT IMPLEMENTATION 27
4.1. Network Installation 28
4.1.1. Mininet installation 29
4.1.2. Floodlight installation 30
4.1.3. NAT network configuration 31
4.2. Simulation Test and Results 33
4.2.1. IPsec authentication 33
4.2.2. Static flow rules 36
4.2.3. DDoS attack scenario 39
4.3. Summary 42
5. CONCLUSION 43
REFERENCES 45
APPENDIX A: NETWORK TOPOLOGY
APPENDIX B: FLOW RULES
APPENDIX C: MITIGATION CODE
CURRICULUM VITAE

LIST OF TABLES

Page

Table 3.1. Wireshark color table [http-4] 23

Table 3.2 Flow entry properties [http-5] 24

Table 3.3 IPsec protocols 26

LIST OF FIGURES

Page

Figure 2.1. Traditional networks vs. software-defined networks 5
Figure 2.2. SDN architecture 6
Figure 2.3. OpenFlow message flow 8
Figure 2.4. Flow entries of different OpenFlow versions 9
Figure 2.5. Major security challenges on the SDN layers [29] 11
Figure 2.6. TCP/IP 3-way handshake 13
Figure 2.7. MITM attack scenario 16
Figure 3.1. Oracle VM Virtual Box window (ORACLE, n.d.) 18
Figure 3.2. Wireshark window 22
Figure 3.3. IPsec diagram 26
Figure 4.1. A detailed simulation and assessment framework for the SDN-based IoT security model 28
Figure 4.2. IoT home network topology 29
Figure 4.3. Create virtual machine window 30
Figure 4.4. Floodlight web GUI 31
Figure 4.5. NAT network setup 32
Figure 4.6. Mininet connectivity test topology 32
Figure 4.7. Floodlight controller GUI 33
Figure 4.8. Securing the tunnel from the controller side 34
Figure 4.9. Securing the tunnel from the IoT home system side 35
Figure 4.10. System topology with IPsec protection 36
Figure 4.11. Secured packets between controller and IoT home system 36
Figure 4.12. Applied flow rules on the system 37
Figure 4.13. Proposed system's workflow 38
Figure 4.14. Controller status before the attack 40
Figure 4.15. Attack detection 40
Figure 4.16. Mitigating process 40
Figure 4.17. System topology 41
Figure 4.18. I/O packets graph 41
Figure 4.19. CPU Utilization 42

1. INTRODUCTION

The Internet of Things (IoT) has become a reality that surrounds us and is no longer a future vision as it was before. It has become an integral part of our lives, starting with smart homes that contain all the comfort equipment such as the smart refrigerator, lighting system, home security system, and other equipment that facilitates our lives. We are promised with life of unparalleled convenience, efficiency, and automation. Yet under all these positive factors lies a hidden threat, the majority of IoT systems griddled with security exposures. Numerous IoT devices do not have secure settings, rendering them prime targets for DDoS assaults like those facilitated by the Mirai botnet [1] These security weaknesses can be in a vast range and very harmful. In 2016 a massive DDoS attack was launched at the major DNS provider Dyn. This attack was very harmful and created chaos for many major sites, including Airbnb, Netflix, PayPal, Visa, Amazon, The New York Times, Reddit, and GitHub. This attack was done using malware called Mirai which create botnet out of different IoT devices like smart TVs, cameras, etc. Researchers highlight that IoT systems frequently experience poor authentication, insecure firmware, and an absence of standard protocols, worsening these risks [2]. As we are getting to depend on IoT more and more every day the need of address these security weaknesses also increase. Managing IoT networks has become more difficult due to the increment of it size and the heterogeneity of device, access networks and protocols [3]. To address these threats, Software-Defined Networking (SDN) has arisen as a viable solution, enabling centralized management and flexible policy application in IoT settings [4].

The traditional networks architectures rely on central hardware approach and that what made Software Defined Network (SDN) a revolutionary model that promises to revolutionize the way networks are designed, managed, and operated. By separating the network into control, data, and application planes Software Defined Networking (SDN) addresses the scalability and mobility of traditional network infrastructure [5]. SDN topology provides three layer approach which are application plane, control plane and the data forwarding plane, the control plane is the responsible of making the decision for the whole network [6]. While the data forwarding plane from it name it is responsible for packet forwarding, The rules are implemented at the control plane and the data forwarding plane devices store the rules that guide them on how to handle the packets when they receive them[7]. The control plane is connected to the application layer through the

northbound API and it is connected to the data forwarding plane through the southbound API. One of the most famous southbound API is the OpenFlow. The OpenFlow protocol is used by the SDN controller to manage the traffic and implement the necessary rules as a flow table.

To quickly solve switching around connecting an endless number of devices in IoT, SDN can be one of the best approaches because one of the SDN advantages is that not each device needs individual configurations. It is done through a centralized controller [8].

1.1. Problem Statement

Securing Network connections became crucial with the increase of digital environment but the problem is that these digital environment is threatened more and more by cyberattacks specially Distributed Denial of Service (DDoS) attacks. Most of the significant attacks are SYN, ICMP, and HTTP GET floods which are different types of DDoS attacks. These attacks overwhelm the systems by sending huge number of harmful data making them unreachable for authorized users.

Other cyberattacks can depend on fraud and sneaking but DDoS attacks huge power by extensive data transfers which make them easy to implement yet create countless damages and hard to control. These attacks leads to delay in the performance and can lead to outright failures. The danger of these attacks relays on using the fundamental network protocols, overwhelming server resources or bandwidth by transmitting massive volumes of apparently regular packets.

Although many detection and mitigation strategies have developed over time, real-time identification of flood-based DDoS attacks in SDN continues to be a significant and unsolved issue. Numerous current solutions face problems like elevated false positive rates, sluggish response times, limited scalability, and substantial computational burdens. Moreover, attackers consistently adapt their strategies, frequently imitating legitimate traffic patterns to avoid detection, which complicates the creation of efficient defense measures.

Therefore, there is an immediate demand for strong, flexible, and lightweight detection methods that can function effectively in SDN settings while maintaining a minimal effect on network performance and service availability.

1.2. Existing Solutions and Related Work

With the increasing of Distributed Denial of Service (DDoS) attack's scale and sophistication, researchers started to look at this dilime and made approaches to detect and mitigate these attacks in the Software Defined Networking (SDN) scope and ensuring robust protection mechanisms is crucial for maintaining network resilience.

Multiple detection frameworks have been suggested to tackle this vulnerability. One line of research centers on entropy-based methods, which examine traffic patterns to detect anomalies. In 2023, a study to accurately detect UDP flood attacks within tree-based SDN topologies while maintaining minimal detection delay introduced a hybrid detection model that combines entropy with packet flow characteristics [9].

Some researchers used machine learning to be their leading approach, In 2024 utilizing smart traceback techniques to classify traffic and identify DDoS attacks quickly, even close to the source the FloodKnight system combined Radial Basis Function networks with Particle Swarm Optimization [10]. In 2022, Ensemble ML methods, such as Random Forest and XGBoost, were utilized in ONOS Flood Defender to attain elevated detection precision with low latency, achieving as much as 99.3% in experimental conditions [11].

Another line of research focuses on flood attacks that are specific to particular protocols, like TCP SYN flooding. A lightweight statistical model utilizing entropy, standard deviation, and weighted moving averages has proven capable of effectively identifying SYN flood traffic with reduced false positives [12]. Another approach to detect and block illegitimate SYN handshakes at the edge of the SDN data plane, integrates a TCP proxy with a hashing mechanism was invented[13].

Additional researchers investigated anomaly detection through traffic behavior by utilizing higher-order statistics like skewness and kurtosis. These statistical metrics proved to accurately detect TCP and UDP flood attacks with minimal false positive rates in a sequential heuristic framework tailored for SDN settings [14].

The changing strategies of attackers require flexible models that can manage fluctuating traffic patterns instantly while reducing the effect on network performance and with all these improvements some models still have challenges in identifying complex low-rate flood attacks, or reliance on delicately adjusted thresholds.

1.3. Research Contributions

To address the limitations of existing flood-based DDoS detection systems in Software Defined Networking (SDN), this thesis presents a lightweight, adaptable, and real-time framework for flood detection.

This thesis presents a practical, real-time, and adaptive flood detection system specifically designed for SDN settings. It enhances detection speed, lowers overhead, providing a viable solution for alleviating contemporary DDoS flood attacks in adaptive networks. The implementation in our system are formed using threshold-based, flow-statistics-driven algorithm that is continually monitor the traffic behavior at the controller which gives the advantage over the machine learning models that having high complexity or require wide-ranging training data. Our method enables the system to softly detect flood characteristics like burst packet rates and unusual flow-to-port ratios without depending on extensive models. The suggested framework is integrated directly into the SDN controller, allowing for the prompt prevention of harmful flows at edge switches. Furthermore, the model is deployed and assessed with Mininet and the floodlight controller to imitate realistic DDoS flood attack situations, incorporating SYN flood types. Metrics including detection time, packet loss, and flow rule tally are evaluated. Findings indicate a decrease in controller response duration and enhanced flow management effectiveness.

2. SDN OVERVIEW

The complexity of the traditional network from making it hard to configure the network to suite the needed policies and from reconfiguring the network based on the problems that may occur to the network at any time from loads or faults gave the chance to new technology named Software Defined Network (SDN) to popout and solve this issue by decoupling the control authority of the switches and routers, and gave the ability to program the network as it needed. Figure 2.1 shows the difference between the SDN and traditional networks architecture.

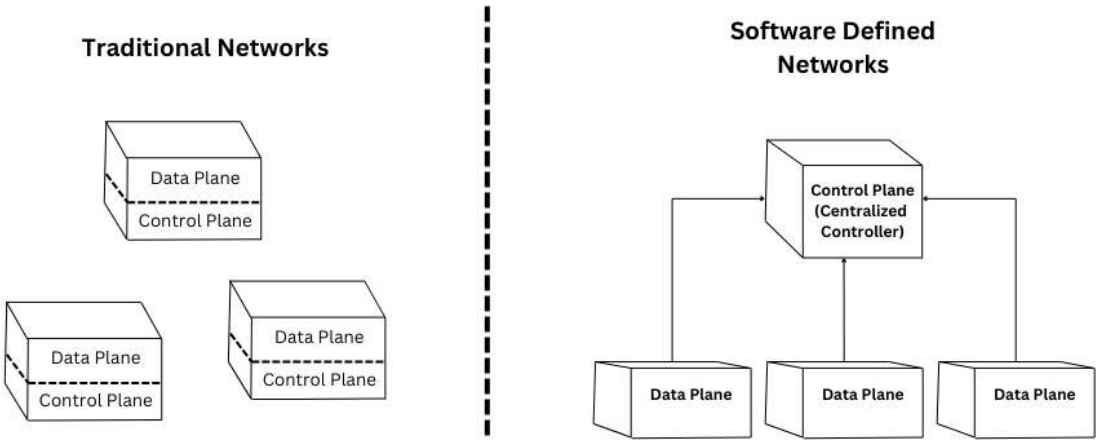

Figure 2.1. *Traditional networks vs. software-defined networks*

2.1. SDN Architecture

Software defined networks gave the spark of hope to change the limitations of the traditional networks by separating the control panel from the underlying switches and routers which makes their job just to forward the traffic and the control authority will be in a centralized controller to manage and modify the network policies [15]. The SDN architecture as shown in Figure 2.1 where it is consisting with three main layer which are the application layer, control plane layer and the data plane layer. Where the application layer communicates with the control layer through the north bound interface and control layer communicate with the data plane layer (Infrastructure layer) through the south bound interface. This division enhances programmability, scalability, and management of the network. [16], [17].

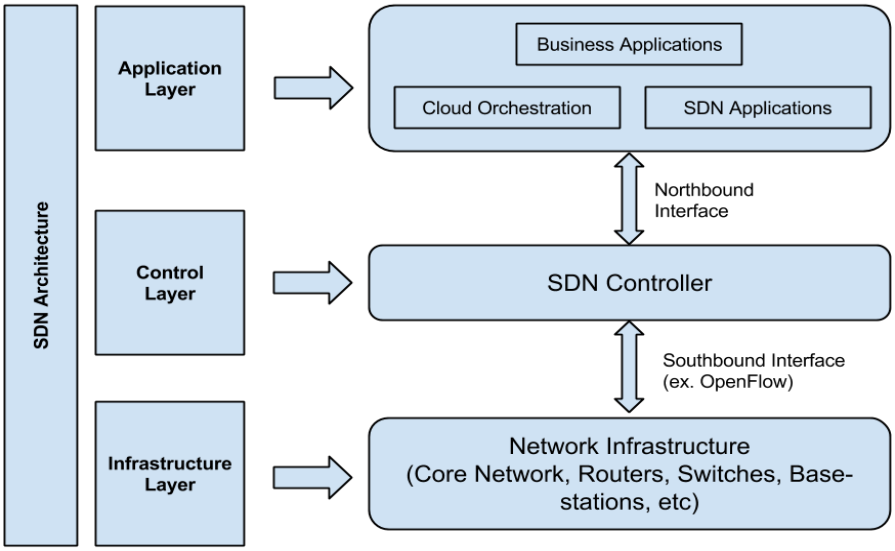

Figure 2.2. *SDN architecture*

a) **Infrastructure Layer** which also known as Data plane layer, here are the devices which responsible for applying the rules which coming from the control layer through the southbound interface from forwarding the packets or drop it according to the installed flow tables, these devices can be either hardware or software.

b) **Southbound Interface** it is the bridge that allow the controller to communicate with the devices in the data plane layer. It is programable interface that defines the set of rules for the forwarding devices in the data plane. OpenFlow continues to be a key protocol in the southbound interface, allowing SDN controllers to efficiently control forwarding devices. It enables flexible policy enforcement and traffic direction according to controller directives [18], [19].

c) **Control Layer** it is the most important part of the Software Defined Network architecture. here where all the security rules and network configurations are saved, the control layer is responsible for obtaining the statistics of the network and modify the rules and flow entries which after that it will be forwarded to the data plane layer devices. The control layer is essential for making routing decisions, implementing security policies, gathering statistics, and refreshing flow tables in the data layer[20], [21]. The controller transferee their instructions into data layer for the network devices to execute the necessary network tasks while the information exchange with upper layer applications through northbound APIs.

d) **Northbound Interface** it allows services in the SDN architecture through middleboxes like load balancers, intrusion detection systems and routing algorithms to be deployed. It connects the application layer with the control layer.

e) **Application Layer** it is the top layer of the SDN architecture, and it is responsible for defining the network behaviour and polices through applications and services. These apps can generate complete functionalities and decisions by responding to alterations in the network. Applications can adjust the network behaviour dynamically when changes are made to the network topology, features, or policies.

Simply when the packet comes through the switch, the switch will check the flow entry which came from the controller and see if the packet will match any of these flow rules. In case of mismatch the switch will send the packet header to the controller through message called Packet_In, then the controller will decide the action related to the packet situation and then will send a Packet_out message back to the switch which will contain the action whether it is forwarding or drop etc.

2.1.1. SDN Open Virtual Switches (OVS)

Open vSwitch (OVS) is a production-quality, multilayer virtual switch designed to enable effective network automation through programmatic extensions while supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). Open VSwitch is flexible as it provides wide range of networking features and protocols also it is known of enhanced performance and the integrity. OpenFlow protocol is supported standard from OVS switches and connects to the SDN controller via the Southbound interface as shown in Figure 2.3.

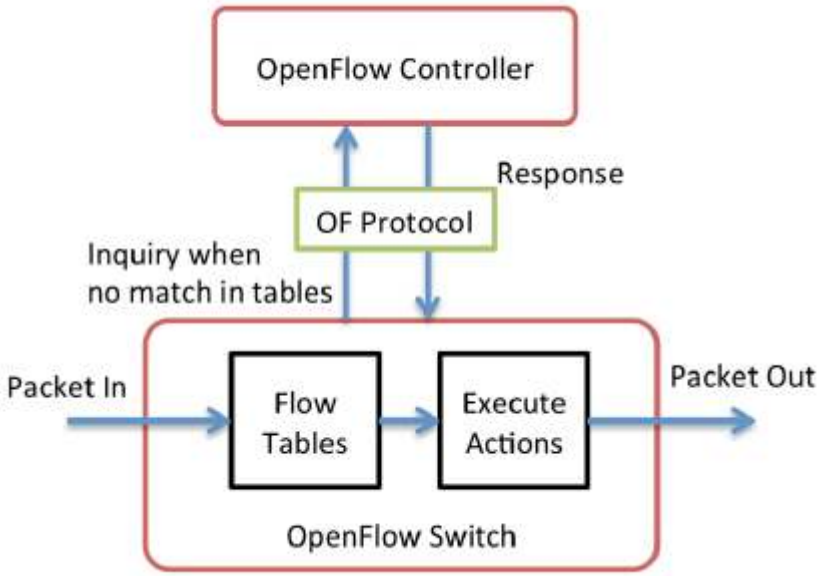

Figure 2.3. *OpenFlow message flow*

2.1.2. OpenFlow protocol

OpenFlow is communication protocol used in the southbound interface, the OpenFlow switch exchanges messages with the controller through a secure channel to receive forwarding entries and report its status. The OpenFlow switch can be classified into two types based on their support for OpenFlow as follows:

- **Dedicated OpenFlow switch:** here the switch cannot perform Layer 2 or Layer 3 forwarding on the traffic it only supports OpenFlow forwarding where it processes all traffic that passes through it in OpenFlow mode.
- **OpenFlow-compatible switch:** It is a commercial switch that supports OpenFlow features such as flow tables and secure channels. It also can perform Layer 2 or Layer 3 forwarding on the traffic.

The communication between the switch and the controller is done through OpenFlow protocol which is usually encrypted using Transport Layer Security (TLS) [22]. the following messages are transmitted over this channel:

- **Controller-to-switch message:** is sent by the controller to the OpenFlow switch to manage or obtain the OpenFlow switch status.
- **Asynchronous message:** is sent by the OpenFlow switch to the controller to update network events or status changes to the controller.

- **Symmetric message:** is sent without solicitation by either the OpenFlow switch or the controller. It is mainly used to set up a connection and detect whether the peer is online.

In traditional networks each switch and router have a routing table which is used to deliver the packets and those tables are largely static so the administrator must update each table individually. In OpenFlow there is something similar called Flow table which contains many rows of flow entries which tell the switch how to handle each packet. This level of flexibility allows each OpenFlow switch to act as a basic firewall as well, Switches can forward packets that do not match any rules to the SDN controller for the controller to inspect and create a new flow rule for it. Figure 2.4 shows the flow entry structure according to each version of OpenFlow.

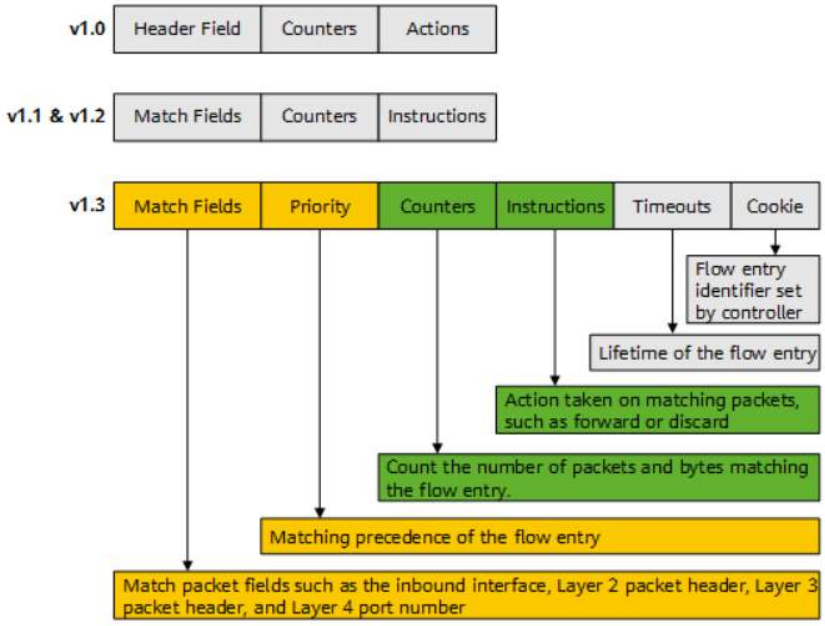

Figure 2.4. *Flow entries of different OpenFlow versions*

2.2.Controller Overview

The SDN network's core is the SDN controller, which has centralized logic, policy database, and a set of rules for traffic flows passing through the forwarding plane. the SDN network to be easily programmed and adjust to changing network conditions in real-time using the SDN controller. As traffic flows through the data plane the SDN controller will establish flows either proactively or reactively. In reactive mode, flows are established upon detection of unknown traffic and as it traverses the network's data plane,

while in the proactive state, the controller will come equipped with predetermined rules prior to the traffic moving through the network.

- **Floodlight** [23]**:** The Floodlight Open SDN Controller is an Apache-licensed, Java-based OpenFlow Controller designed for enterprise use. A group of developers, including several engineers from Big Switch Networks, back it up. Floodlight is coded in Java which allows it to operate inside a Java Virtual Machine (JVM). For my experiment I chose Floodlight to be the controller for my experiment is that Floodlight can be flexible for small and medium networks and it has good Web Graphical User Interface (GUI) which provides all the information about the devices that connected to the controller from switches and links and hosts. The GUI of floodlight controller also provides the networks statistics and the firewall statue. The other thing what made floodlight controller special to me is the list of REST API that the controller provides to get information about the network or add/remove Rules.

2.3. SDN Benefits

Software-Defined Networking (SDN) has emerged as a fundamental technology in diverse networking settings, such as IoT, industrial automation, and vehicular networks, because of its ability to decouple the control and data planes. This architectural distinction facilitates centralized control, flexible resource management, and easier configuration, positioning SDN as an essential enabler for scalable and adaptable networks[24].

The centralized controller model in SDN architecture allows administrators to oversee traffic trends, enforce policies throughout the network, and adjust routing dynamically without interrupting active services. This leads to decreased latency, improved traffic management, and considerably fewer manual setup mistakes [25].

In IoT settings, SDN provides improved visibility and control over device interactions, facilitating the management of the increasing diversity and scale of devices. This is especially crucial in environments such as smart homes and vehicular networks, where conventional static setups are inadequate. SDN allows these settings to adjust dynamically to changes in topology and shifting security requirements [26].

From an economic viewpoint, SDN minimizes hardware reliance and operational expenses by enabling the virtualization and automation of network functions. Businesses gain from enhanced efficiency and decreased deployment time for new services, as

controllers can manage changes across the network without requiring physical modifications [27].

Additionally, the security benefits of SDN are crucial. The centralized structure of SDN facilitates instant threat identification and response, allowing administrators to quickly address anomalies or attacks such as DDoS or MitM. As shown in contemporary applications, policies can be adjusted in real time, and flows can be automatically redirected or halted based on identified threats [28].

2.4. SDN Security Challenges

Security has emerged as a significant issue across various levels, particularly in recently developed network platforms like cloud networks and peer-to-peer networks. Hence, in spite of the numerous benefits, SDNs face several built-in network security hurdles such as scalability, reliability, controller positioning, and latency. The growing threat of security attacks on SDN layers is now a major worry. The lack of best practices in SDN functions, components, and network programmability leads to a rise in security threats such as flow rule consistency, controller vulnerability, legitimacy, malicious applications, and standard northbound and southbound communications potential [29]. The major and most common security challenges on the SDN layers are shown in Figure 2.5.

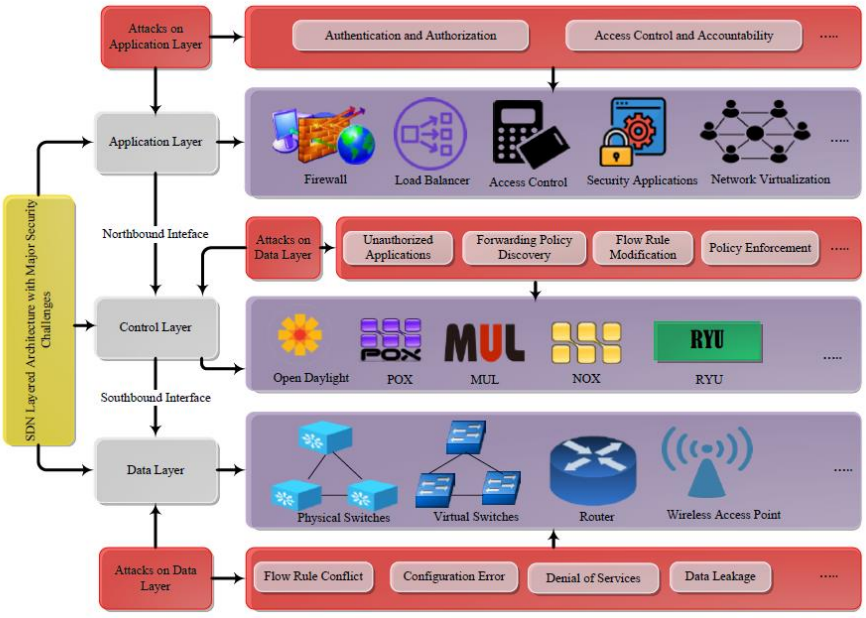

Figure 2.5. *Major security challenges on the SDN layers [29]*

2.4.1. Data plane attacks

Some data plane attacks are given as follows:

- **Fraudulent Flow Rules:** In OpenFlow networks, the OpenFlow controller is responsible for setting up flow rules in the flow tables of the OpenFlow switch. Flow rules can be set up either before a new host sends packets (proactive installation) or when a new host sends its first packet (reactive installation) and with the decision-making function no longer in switches, the primary security issue is identifying true flow rules and distinguishing them from fake or harmful rules [30].
- **Flooding Attacks:** due to the ability to instantly access network services of DDoS and DoS attacks in SDN-based communication systems the frequency of these attacks increased significantly [31]. DDoS attacks are without a doubt the most common type of attacks targeting the control plane in SDN architecture, often originating from the packet forwarding plane when requesting information about unfamiliar traffic. If a network packet does not comply with the flow rules of OpenFlow, it will be sent to the SDN controller by the OpenFlow switch. Specifically, malignant tools may create and transmit fake packet to appear like a legitimate data packet for transmission through an SDN switch and redirect it to the SDN controller. An attacker can render the OpenFlow switch ports inoperative by injecting false links into the SDN topology. Hence, DoS/DDoS represents one of the most damaging and threatening attacks against the SDN controller.
- **TCP-Based Flooding Attacks:** A SYN flood, also known as a half-open attack, is a form of denial-of-service (DDoS) attack designed to render a server inaccessible to genuine traffic by utilizing its entire available resources. Through continually sending SYN packets for initial connection requests, the attacker can flood all open ports on the designated server, resulting in slower or non-existent responses to legitimate traffic from the targeted device. In usual circumstances, the TCP connection goes through three distinct processes to establish a connection as shown in Figure 2.6, SYN flood attacks function by taking advantage of the TCP connection handshake process.

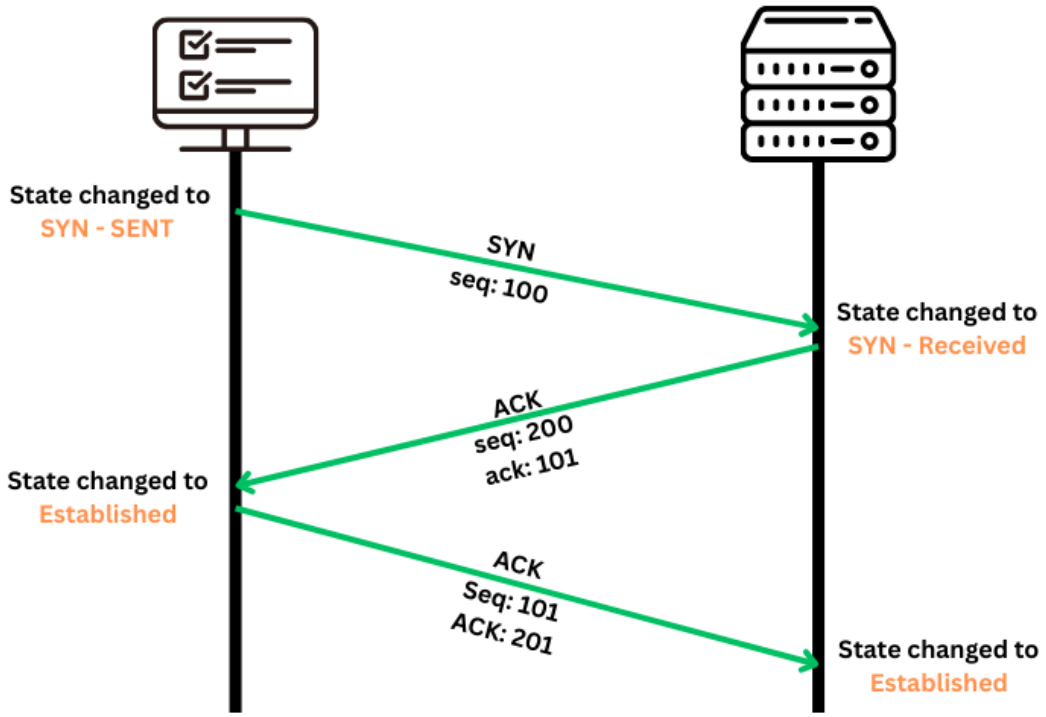

Figure 2.6. *TCP/IP 3-way handshake*

The way that DDoS attack will be released in this case attacker takes advantage of the server's response to the initial SYN packet by sending one or more SYN/ACK packets and waiting for the final handshake step. The attacker will send huge number of SYN packets to the victim then he will for the server to responds to each one of the connection requests and leaves an open port ready to receive the response and while the server is waiting for the final ACK packet which will never come the attacker will keep sending SYN packets which will cause the server to open all available ports and be unable to function normally.

- **ARP Spoofing Attack**: ARP is used to determine the MAC address linked to an IP address. ARP spoofing attacks exploit this procedure by allowing the attacker to send false ARP messages to the LAN and link their MAC address with the IP address of other hosts [32]. To identify ARP spoofing attacks the authors in [33] suggested three approaches this covers techniques for analyzing packets based on signatures, machine learning, and Wireshark.
- **Side Channel Attacks:** Timing information is used in these attacks to uncover crucial data such as flow tables, routes, controller types, and ports, presenting a major danger to communication networks[34]. One of the primary difficulties faced by SDNs is the timing side-channel attacks.

2.4.2. Control plane attacks

Some of the control plane attacks are given in the following.

- **Link Fabrication Attack:** Several SDN controllers establish connections between switches by instructing them to broadcast Link Layer Discovery Protocol (LLDP) frames through all ports. The goal of the Link Fabrication Attack is to trick the SDN controller into thinking there is a direct connection between switches when there isn't. Adjacent switches will detect excessive LLDP traffic and forward it to the controller, enabling it to recognize the presence of a connection between them. This process of link discovery enables automatic detection of network topology and link latencies, which lessens the burden of configuration on administrators and makes it appealing for SDN deployments [35].
- **Persona Hijacking:** The weaknesses in SDN's identifier binding mechanisms gives the advantage to Persona Hijacking encompassing both the IP takeover and flow poisoning phases within the network. Such an adversarial attack greatly increasing their access to network resource by deceiving the SDN infrastructure by treating the attacker as the legitimate owner. Worm-Hole Attack can be caused because of The Persona hijacking, resulting in flow misdirection within the network [36].
- **Reverse Loop:** The SDN network controller's perspective is greatly compromised by reverse loop attacks within the network. The controller is able to determine the presence of a reverse link by analyzing the network's dynamic traits. During the specified timeframe, the inter-switch connections that currently exist are eliminated in these attacks by reversing the links.
- **Topology Freezing:** This type of attacks takes advantages of the weaknesses in the modules that provide topology services in the controllers it also impacts the way the controller views the network and causes the controller's topology view to become frozen, preventing it from updating the dynamic network changes.
- **Network Manipulation:** In the network manipulation attack, the intruder compromises the controller to insert false data into the network and attempt to implement additional attacks across the entire network [37].
- **Traffic Sniffing:** This type of attacks allow the intruder to intercept the network data and by capturing and analyzing the communication interface they can spy on

important information, the intruder can take advantage of unencrypted data to obstruct communication to and from the controller. Implementing complex passwords and secure encryption methods may help mitigate the effects of traffic sniffing in SDN [38].

- **Man-In-The-Middle Attacks (MITM):** An attack happens at the Southbound interface level, where a hacker can be in the middle of communication between the controller layer and infrastructure layer without their knowledge [39]. An Intruder can effortlessly acquire sensitive data by positioning themselves covertly between the user and a trusted system, like a website or application. The individual believes they are only engaging with a reliable website and freely gives up their login details, financial data, or any other sensitive data. There are many types of MITM attacks include:
  - **IP Spoofing:** A hacker changes the IP address of a website, email address, or device to deceive the user into thinking they are communicating with a legitimate source, when in fact they are sharing information with a cybercriminal.
  - **DNS Spoofing:** In order to collect user credentials or other data, spammer develops a counterfeit website that appears familiar to the user and directs them to it.
  - **HTTPS Spoofing:** the users covertly sent to an insecure HTTP site, enabling criminals to monitor activities and pilfer data. When they mistakenly believes a website is using HTTPS, they believe their computer **data is encrypted when sent to the website host.**
  - **Email Hijacking:** Hackers gain unauthorized access to email accounts of banking or credit card companies to monitor transactions and steal data secretly. They could also utilize the email account or a fake email address that is slightly altered from the real one to give incorrect guidance to clients, like transferring funds to a different checking account.
  - **SSL Hijacking:** hackers intercepting user data by taking control of the Secure Sockets Layers (SSL) responsible for encrypting HTTPS connections between users and servers. It is an extension of HTTPS spoofing.

Figure 2.7 shows a real scenario of MITM attack.

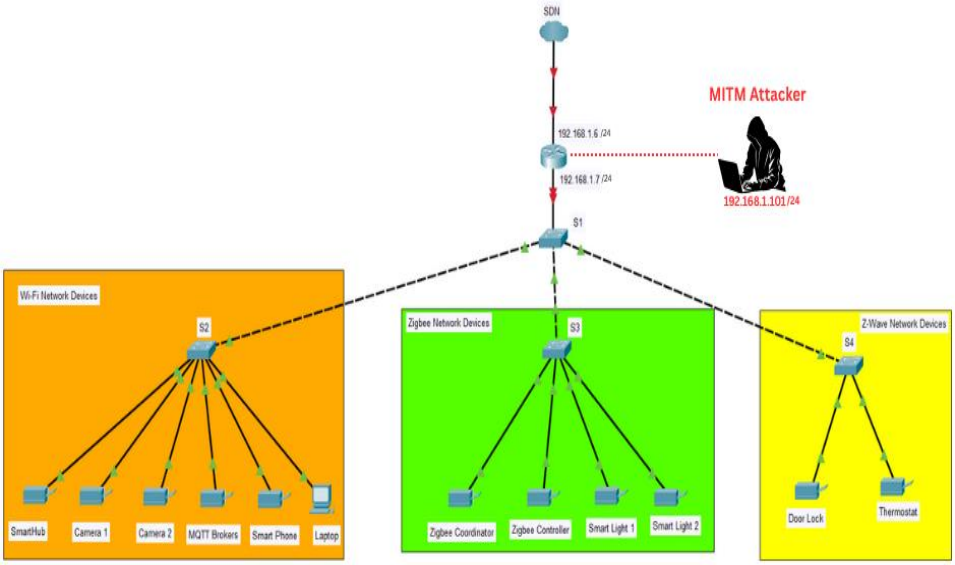

Figure 2.7. *MITM attack scenario*

2.4.3. Application plane attacks

In the following, some applicatin plane attacks are described:

- **App Manipulation: A**pplications are weak to manipulation attacks targeting apps. During these assaults, the hacker could potentially enter a SDN application, create dysfunction, spy on the data being used, and interrupt the services. Moreover, the malicious actor may acquire elevated permissions in order to carry out illicit actions within the SDN applications.
- **API Exploitation:** Attackers can result in unauthorized access to critical data by exploit the API of certain software components connected to SDN systems. This results in API misuse, potentially causing disruption of information flow within the network. It is essential to regularly update the patches of applications running on SDN nodes to prevent exploitation.
- **Brute Force:** Hackers use brute force attacks to try all possible combinations of passwords and usernames in order to find the user credentials. This attack uses a trial and error method to guess the password.

These are the most common attacks that can target the SDN networks and the thesis will cocentrate on DDoS attacks as they are the most popular and harmful types of cyperattacks and can serve as gateways to other types of attacks [http-1].

2.5. Traditional Network versus Software Defined Networks

Traditional networking involves the use of fixed and dedicated hardware devices like routers and switches to manage network traffic unlike the Software Defined Networks which decoupling the control layer from the data layer and makes the devices (routers and switches) just for forwarding date. Traditional networking utilizes specialized hardware devices such as routers and switches to control network traffic, but the problem of traditional networks is that Traditional networks are restricted in how much they can grow because they rely on physical hardware devices, also Traditional networks lack advanced automation features and need substantial manual involvement. Furthermore, traditional networks have a strict, structured layout that is challenging to alter or adjust to evolving business requirements.

3. METHODOLOGY

In this chapter, we describe the practical implementation of our proposed security model within a software-defined networking (SDN) environment. The main focus is on configuring the testbed, setting up the necessary tools and components, and defining the specific flow rules used to control network behavior. The structure of the implementation is outlined step by step, beginning with the simulation environment and proceeding through rule definition and controller configuration. This chapter provides the technical foundation for the experimental analysis presented in Chapter 4.

3.1. Oracle VM Virtual Box

Oracle VM VirtualBox is software for virtualization that can be used on multiple platforms. Users can expand their current computer to support multiple operating systems such as Microsoft Windows, Mac OS X, Linux, and Oracle Solaris simultaneously. Oracle VM VirtualBox enables developers to build multiplatform environments and create applications for container and virtualization technologies all on one machine which makes it easier to deploy to the cloud (ORACLE, 2021). Figure 3.1 shows the GUI of the Oracle VM Virtual Box after it has been installed, it is lightweight and easy to install and use.

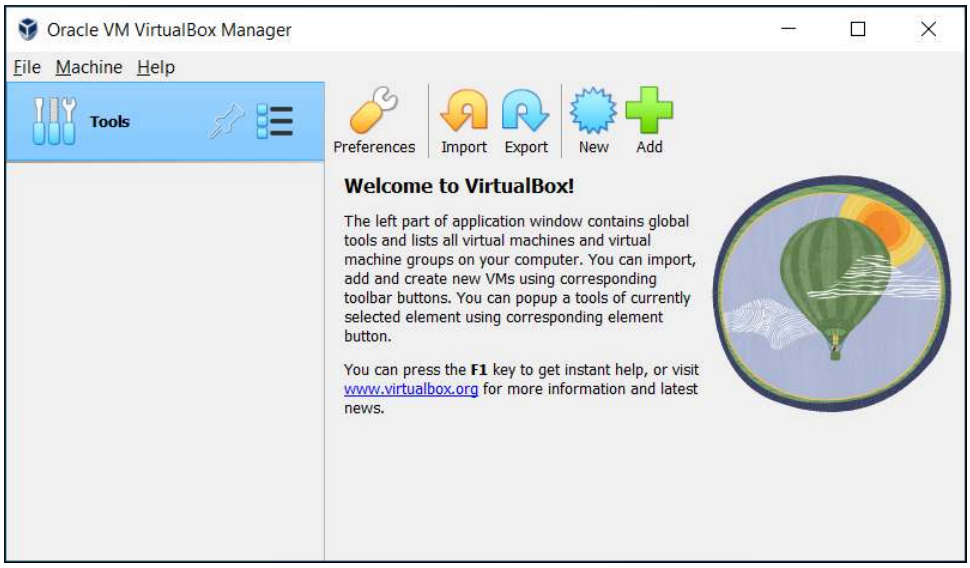

Figure 3.1. *Oracle VM Virtual Box window (ORACLE, n.d.)*

Through this window you can add your Virtual machines the operating system was Microsoft Windows, Mac OS X, Linux. The Oracle VM Virtual Box provides numerous networking options available for various configurations that can be easily set up with just a few clicks which will be mentioned below:

- **NAT:** Network address translation (NAT) establishes a private network connection between the host machine and the chosen VM in this mode. It is a closed network where only the host and the designated VM are allowed to communicate with each other, without any other machines being able to join. The virtual machine has the ability to utilize the host's internet connection while in this network mode.
- **Bridged Adaptor:** Under the Bridged Adaptor setting, each virtual machine is seen as an independent entity on your real network, allowing communication between host machines, other virtual machines, and devices on the network seamlessly.
- **Internal Network:** Establishing an internal network connects several virtual machines (VMs) together. In this state, the VMs are isolated from the host machine and operate on a separate network.
- **Host-only Adapter:** With this option, your virtual machines and the host computer are linked together but isolated from the external network. This offers the utmost level of network security for your VMs but comes with the downside of having somewhat restricted networking capabilities.
- **Generic Driver:** This network mode allows sharing the host machine's generic network interface with the chosen VM, giving the specific VM its own network controller and isolating it from the rest of the physical network.
- **NAT Network:** Like NAT but includes all VirtualBox VMs in the network instead of just one. This indicates that both the host machine and your VMs are on a separate network, enabling them to interact and utilize the host machine's internet connection for internet access.
- **Not Attached:** In this mode, the VM is isolated from the network, resulting in no network or internet access with the host or other virtual machines.

For our case we will use NAT network to connect our virtual machines with each other's and separate it from the host machine network.

3.2. Ubuntu 22.04

Ubuntu is a distribution of Linux that is built upon Debian. Cloud computing, servers, desktops, and Internet of Things (IoT) devices are all compatible with it. It is a Linux distribution operating system that is open-source and can operate in either desktop or server mode. It is simple to install the operating system and users can customize and install relevant packages based on their specific needs. The Ubuntu desktop is used by engineers worldwide as the most popular Linux workstation platform. Ubuntu is eased to use, the navigation through the system is simple as all configuration and application elements can be reached from the main screen, it also have a strong security system it is an open-source platform, is continuously monitored and reviewed by its community members. Consequently, any security weaknesses can be promptly discovered and remedied. Furthermore Ubuntu is a Lightweight Performance system it does not require a lot of resources and runs well on less powerful devices. The standard interface is capable of functioning on under 1 gigabyte of RAM. Additionally, many Ubuntu desktop interfaces are even lighter in weight. For instance, systems with as low as 512 MB of RAM can support ubuntu. When comparing the two, both Windows and macOS demand a significantly larger amount of resources – macOS Big Sur and Windows 11 both require at least 4 GB of RAM to operate [http-2].

3.3. Mininet

Mininet is a tool that simulates a network by generating virtual hosts, switches, controllers, and links. Mininet hosts utilize regular Linux network software, with its switches capable of supporting OpenFlow for customizable routing and Software-Defined Networking[http-3]. It also can be obtained as a package and imported into python. Writing scripts in Python allows us to communicate and control the Mininet nodes as well. Mininet software is selected for its flexibility, scalability, and capability to be customized for various topologies and features. The Characteristics of Mininet is as following:

- It allows for the creation and evaluation of software-defined network applications without the requirement of establishing a physical setup.
- It provides a platform for us to develop and test applications that utilize OpenFlow protocols through a network testbed.
- It can also be utilized without the need for programming.

- It also allows us to incorporate python API, opening up possibilities for building and testing networks.
- It has Command Line Interface that understands topology and OpenFlow, enabling us to troubleshoot or conduct network tests for our application.

3.4. Wireshark

Wireshark is tool for analyzing network packets displays captured packet information with maximum level of detail it is functions as a network packet analyzer. Wireshark serves various purposes, such as resolving performance problems in networks. Cybersecurity experts frequently utilize Wireshark to track connections, examine the contents of suspicious network transactions, and pinpoint surges in network traffic. Figure 3.2 shows the window of Wireshark while capturing packets from the network. Wireshark can do three things:

1. **Packet Capture:** Wireshark monitors a network connection live and captures complete streams of traffic, potentially capturing tens of thousands of packets simultaneously.
2. **Filtering:** Wireshark can cut and analyze all the unorganized real-time data using filters.
3. **Visualization:** A packet sniffer like Wireshark enables users to directly examine the inner workings of a network packet. It also enables you to see complete discussions and network traffic.

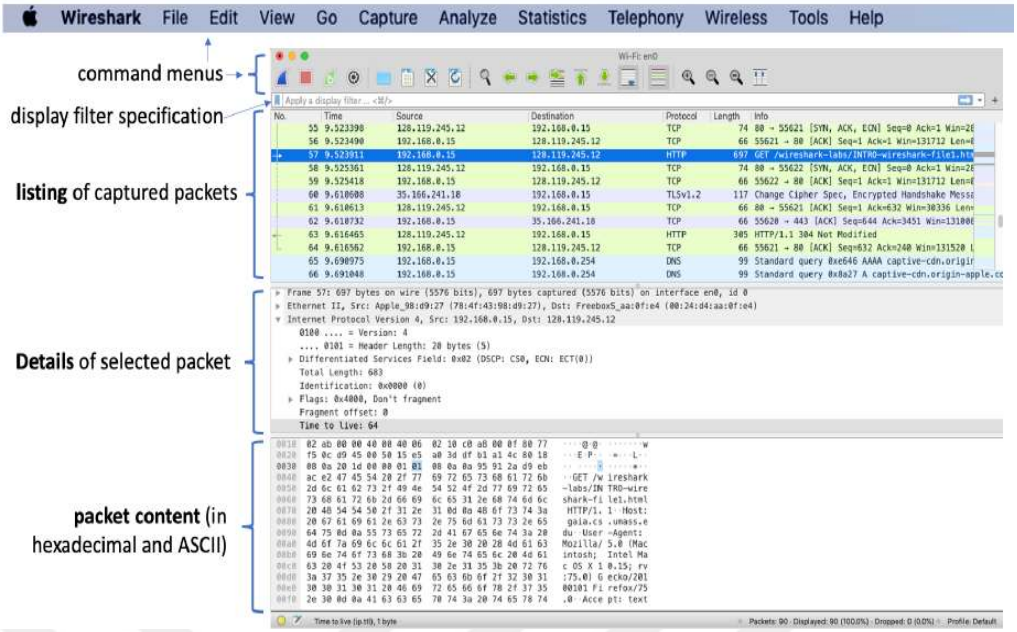

Figure 3.2. *Wireshark window*

As shown in Figure 3.2 Wireshark have five main components:

- **Command Menus:** These are typical drop-down menus positioned at the top of the Wireshark interface. The most important command menus for us are the file menu and the capture menu, The File menu enables you to save captured packet data, open a file with previously captured data, or close the Wireshark application. The Capture menu enables you to start packet capturing.
- **Packet-Listing Window:** It shows a brief description of each captured packet, with the packet number included the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet.
- **Packet-Header Details Window:** gives information about the highlighted packet in the packet-listing window.
- **Packet-Contents Window:** shows all the data from the captured frame, in ASCII and hexadecimal forms.
- **Packet Display Filter Field:** where a protocol name or other details can be typed in to filter the information shown in the packet-listing window.

Wireshark aims to assist users in recognizing packet types through the use of intuitive color coding. The Table 3.1 outlines the standard colors assigned to significant packet types.

Table 3.1. *Wireshark color table [http-4]*

Color in Wireshark	**Packet Type**
Red	TCP
Blue	UDP
Black	Packets with errors
Light green	HTTP traffic
Light yellow	Windows-specific traffic, including Server Message Blocks (SMB) and NetBIOS
Dark Yellow	Routing
Dark Gray	TCP SYN, FIN and ACK traffic

3.5.DDoS Attack tool

DoS and DDoS attacks are efforts to disrupt the normal functioning of a specific server, service, or network by inundating it with a large amount of online traffic. DoS attacks disrupt by sending harmful traffic from one machine, usually a computer. A basic ping flood attack involves sending an excess of ICMP requests to overwhelm a server's processing capacity and hinder its ability to respond effectively while DDoS attack involve the utilization of multiple machines to direct harmful traffic towards a specific target. Frequently, these machines are included in a botnet - a group of computers or other devices that have been infected with malware and are able to be manipulated from a distance by a single attacker. In different situations, numerous attackers come together to launch DDoS attacks by coordinating the traffic from their own computers.

DDoS attack tools can be categorized as following:

- **Low And Slow Attack Tools:** These attack tools are characterized by their use of minimal data and slow operation, as suggested by their name. Created with the intention of maintaining open ports on a specific server by transmitting small data increments through various connections, these utilities exhaust the server's resources, preventing it from sustaining more connections. Unusually, attacks

that are slow and of low intensity can still be successful without the need for a distributed network like a botnet and are often carried out by a single computer.

- **Application Layer (L7) Attack Tools:** These tools focus on layer 7 of the OSI model, which is where Internet-based requests like HTTP take place. By utilizing an HTTP flood attack, a cyber attacker can inundate a target with HTTP GET and POST requests, making it challenging to differentiate the attack traffic from legitimate requests made by genuine users.
- **Protocol And Transport Layer (L3/L4) Attack Tools:** Moving deeper into the protocol stack, these utilities make use of protocols such as UDP to transmit high amounts of data to a specific server, as seen in a UDP flood scenario. Though they may not be very successful on their own, these attacks are usually seen in the form of DDoS attacks, where having more attacking machines amplifies the impact.

In our experiments, we will utilize the hping3 Linux command line tool to create DDoS traffic aimed at the controller.

3.6.Floodlight REST API

The security system for our experiment will be deployed through Floodlight Controller REST API, The REST API enables network administrators and applications to manage the network, access information, and set up network components through code. It enables us to access information on linked switches, obtain specifics on switch ports and statistics. Additionally, the Flow Management feature allows users to Add, change, or remove flow entries on switches and Access active flow entries and statistics. We will be using these features to deploy our network restrictions. Table 3.2 shows some Possible properties of a flow entry that must be taken into account while creating the flow rules.

Table 3.2 *Flow entry properties [44]*

Key	Value	Notes
SWITCH	<switch ID>	ID of the switch (data path) that this rule should be added to xx:xx:xx:xx:xx:xx:xx:xx

Table 3.2 (Continued) *Flow entry properties [http-5]*

NAME	<string>	Name of the flow entry, this is the primary key, it MUST be unique
Actions	<key>=<value>	Allow or deny set of network flows which are match rule
dst-mac Destination	xx:xx:xx:xx:xx:xx	Destination MAC-address
src-mac Destination	xx:xx:xx:xx:xx:xx	Source MAC-address
src-ip	A.B.C.D/M	Source IP-address
dst-ip	A.B.C.D/M	Destination IP address
nw-proto	TCP or UDP or ICMP	Protocol

The rules can be added to the controller either by writing a python code which will add all your rules in one command or Linux “curl” command.

3.7.IPsec

IPsec, short for Internet Protocol Security, is a set of protocols created to protect IP communications by verifying and ciphering every IP packet within a communication session. It functions on the network layer and is utilized for creating Virtual Private Networks (VPNs), guaranteeing safe data transfer across public networks such as the internet. IPsec is one of the original VPN protocols and is still widely used today. It is standardized by the IETF and provides security at the Internet Protocol (IP) layer [40]. IPsec provide for our network confidentially, integrity, authentication and Anti-replay.

IPsec add the Authentication Header (AH) and Encapsulating Security Payload (ESP) protocol to the original IP packet as shown in Figure 3.3. AH [41] secures IP packets by verifying the sender's identity and validating the integrity of the packets. AH guarantees the integrity of data during transmission and authenticates the sender's identity. It utilizes cryptographic hash functions (such as SHA-256) and a common secret key to generate a hash of the packet's content. ESP [http-6] ensures the privacy of IP packets through symmetric encryption. ESP encodes the IP packet's payload to safeguard the information from unauthorized individuals. Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES) are typical encryption algorithms.

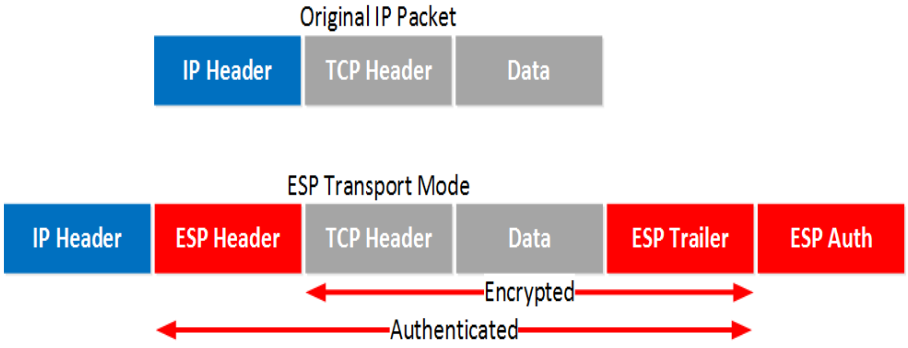

Figure 3.3. *IPsec diagram*

An overview of the protocols that IPsec use to implement it features are shown in Table 3.3.

Table 3.3 *IPsec protocols*

System mode	**Algorithms**
Encryption	DES, 3DES, AES
Authentication	MD5, SHA
Negotiation	DH1, DH2, DH5

Furthermore, IPsec have two modes which are the transport mode and the tunnel mode. The Transport Mode is mainly utilized for direct communication between two individual hosts. It offers security services (such as encryption and authentication) to the content of the IP packet, but not to the source IP header. While the Tunnel Mode is used for network-to-network communications, such as connecting two separate networks over the internet. It encapsulates the entire original IP packet within a new IP packet, providing security to both the payload and the original IP header.

4. EMULATION AND PROOF OF CONCEPT IMPLEMENTATION

In this chapter, we seek to verify the suggested security method an SDN-oriented intrusion detection and mitigation system for smart home IoT settings by means of a well-designed and reproducible simulation. The objective is to replicate a real-life smart home network situation, examine its susceptibility to cyber threats like DDoS attacks, and analyze the system's ability to identify, react to, and alleviate these threats utilizing the software-defined networking approach.

The section is split into three main parts. Initially, we describe the arrangement of the testbed environment, which encompasses the installation and setup of essential software elements including Oracle VM VirtualBox, Ubuntu 22.04, Mininet for network simulation, the Floodlight SDN controller, and additional tools such as Wireshark and hping3. The selection of tools was determined by their alignment with OpenFlow and their capacity to simulate and examine SDN-centric environments with a significant level of control and insight.

Secondly, we create a representative IoT smart home network topology with Mininet, wherein each host represents a typical smart home device (e.g., IP cameras, sensors, smart appliances). The network is connected to the Floodlight controller and secured by flow rules and IPsec security protocols. We emulate various network behaviors and introduce a harmful agent to execute a Distributed Denial of Service (DDoS) attack, replicating actual hazards that smart home networks frequently encounter.

Third, we break the experimental execution into two separate phases: pre-mitigation and post-mitigation. During the initial stage, the network functions without any active firewall measures or IPsec encryption, enabling the simulated attacker to take advantage of vulnerabilities. During the second phase, the mitigation strategies—comprising static flow rules, IPsec tunnel mode, and dynamic threat detection—are employed to showcase their effectiveness in containing attacks and recovering the system.

Figure 4.1 displays the complete architecture and workflow.

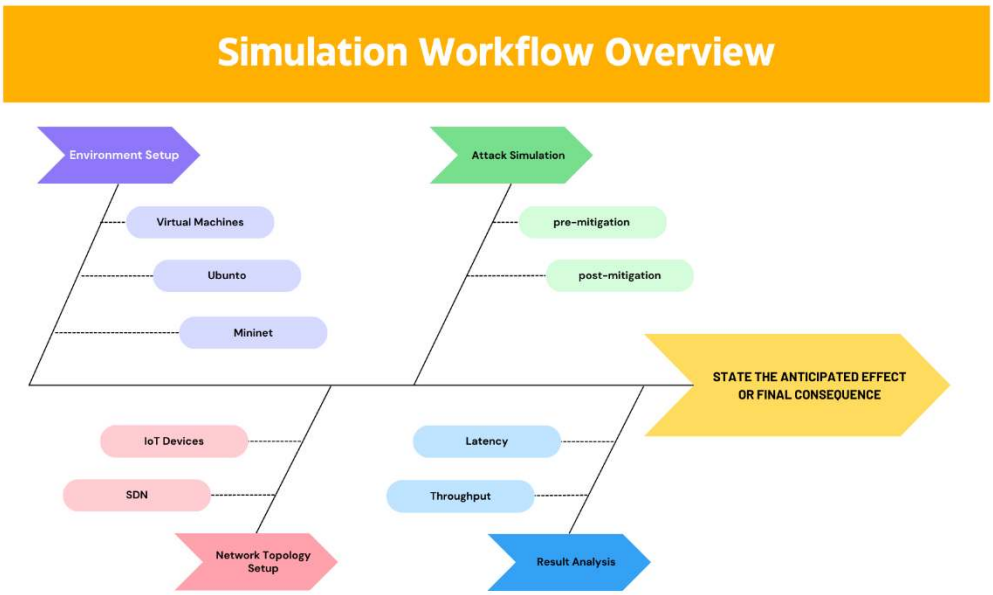

Figure 4.1. *A detailed simulation and assessment framework for the SDN-based IoT security model*

4.1.Network Installation

To effectively replicate a smart home IoT network and assess the security measures suggested in this thesis, it was crucial to create a strong and separate virtual testing environment. This segment describes the comprehensive setup procedure for the experimental testbed, encompassing virtualization configuration, software tool installation, and the connection of components via Mininet and the Floodlight SDN controller.

The emulation setup was created using Oracle VM VirtualBox, chosen for its flexibility and support for various operating systems. Two Ubuntu 22.04 virtual machines were set up: one assigned to operate the Floodlight SDN Controller, and the other tasked with running the Mininet environment that simulates the smart home topology.

These virtual machines were linked together using a NAT Network setup. This configuration created a safe, separated networking environment where simulated devices could interact directly, while still permitting internet access for software installations and updates. The NAT setup guarantees that external traffic does not disrupt the experiment while preserving a realistic network topology. The design is shown in Figure 4.2.

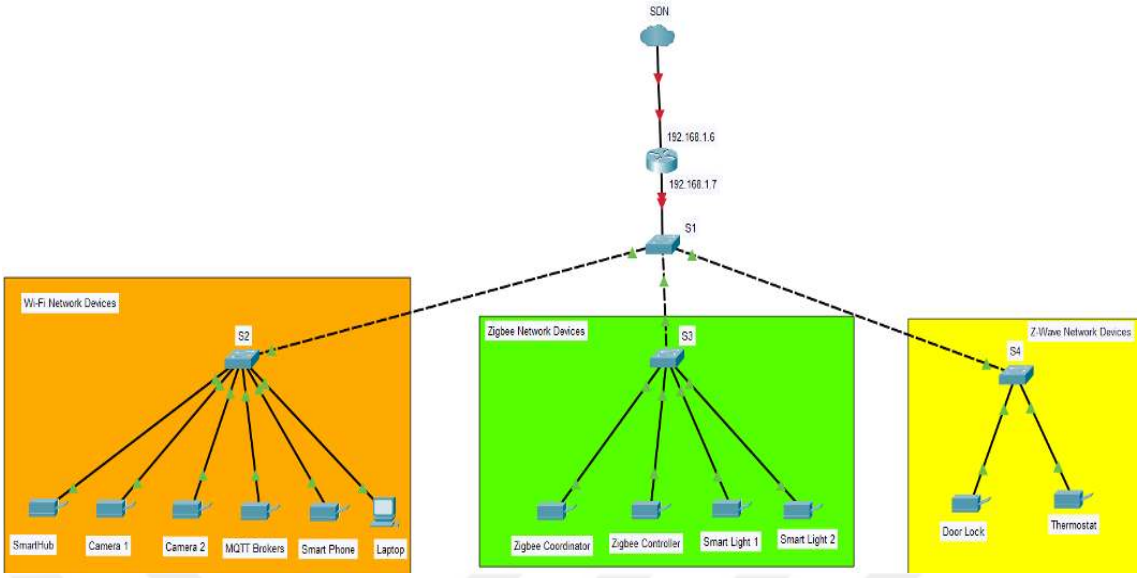

Figure 4.2. *IoT home network topology*

4.1.1. Mininet installation

Mininet was selected as the network emulator due to its built-in support for OpenFlow and its capability to replicate intricate topologies while using minimal system resources. The setup proceeded through the following stages:

1. From the Ubuntu official website download “ubuntu 22.04.04” as iso file.
2. Open Oracle VM Virtual Box and press CTRL+N to add new Virtual Machine.
3. As shown in Figure 4.3 for creating Virtual machine, add the ubuntu 22.04 iso image and then follow the rest of the steps to successfully fix Ubuntu operating system.
4. After Fixing Ubuntu operating system in our Virtual Box now we can download Mininet by opening a new terminal in Ubuntu and following the instructions of Native Installation from Source in the Mininet official website. https://mininet.org/download/

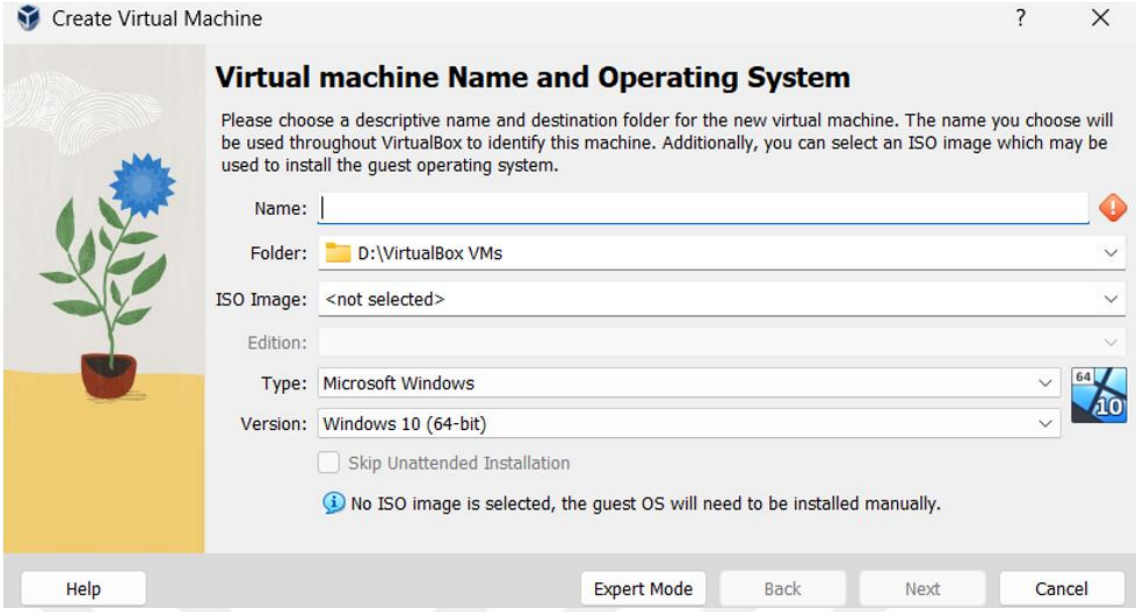

Figure 4.3. *Create virtual machine window*

Once installed, Mininet simulated a smart home featuring twelve IoT devices depicted as Mininet hosts. These hosts were linked to a virtual OpenFlow switches and given Static IP addresses to allow precise flow rule implementation. The topology was created with a Python script (found in appendix A) and is illustrated in Figure 4.2.

4.1.2. Floodlight installation

Floodlight, a Java-based open-source SDN controller, was chosen due to its compatibility with OpenFlow and its comprehensive REST API, allowing for dynamic management of flow rules and real-time monitoring.

The setup was executed on a distinct Ubuntu 22.04 virtual machine and adhered to these procedures:

1. sudo apt update
2. sudo apt install build-essential ant python2-dev openjdk-8-jdk maven git
3. git clone GitHub - floodlight/floodlight: Floodlight SDN OpenFlow Controller
4. cd floodlight
5. sudo git submodule init
6. sudo git submodule update
7. sudo ant
8. sudo cp libthrift-0.14.1.jar ~/floodlight/lib
9. sudo cp netty-all-4.1.66.Final.jar ~/floodlight/lib/

10. sudo rm libthrift-0.9.0.jar
11. sudo rm netty-all-4.0.31.final
12. sudo gedit build.xml

after these steps the floodlight controller will be installed in our second virtual machine and we can run it by writing the following commands in new terminal:

1. cd floodlight
2. java -jar target/floodlight.jar

Upon a successful startup, the Floodlight Web GUI was reachable at http://localhost:8080/ui/index.html, where all switches, links, and connected hosts were displayed. This visual interface was essential for overseeing flow rules, network metrics, and intrusion notifications, as illustrated in Figure 4.4.

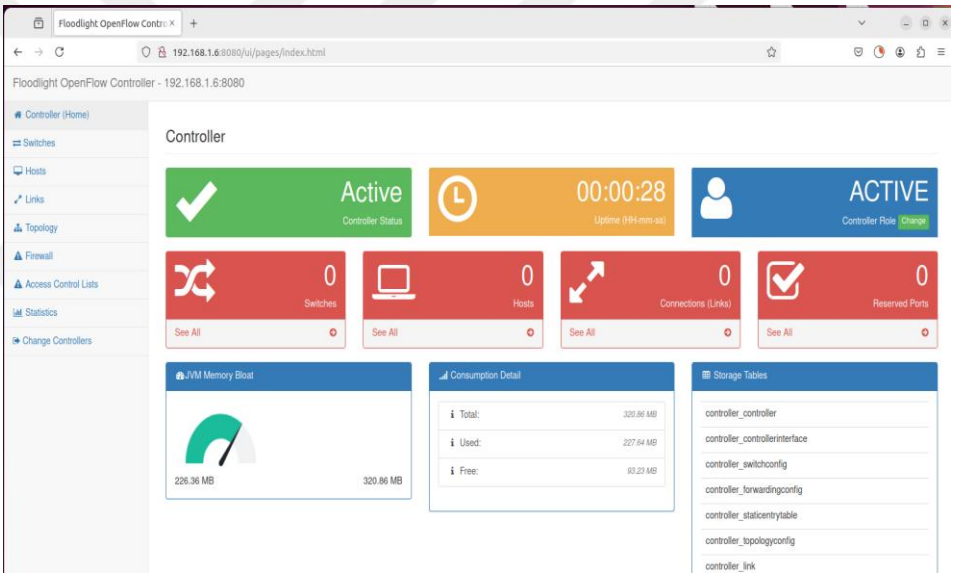

Figure 4.4. *Floodlight web GUI*

4.1.3. NAT network configuration

A custom NAT Network was set up in VirtualBox to facilitate communication between the two VMs while ensuring they remain isolated from the host OS and external devices. This method enabled the Mininet VM to engage directly with the Floodlight controller, simulating a secure LAN setting akin to an actual smart home. The configurations for the NAT network consist of:

Network Title: NAT Network

Subnet: 192.168.1.0/24

DHCP: Turned off (to guarantee Static IP allocation)

Figure 4.5 shows the window of the NAT Network for our experiment.

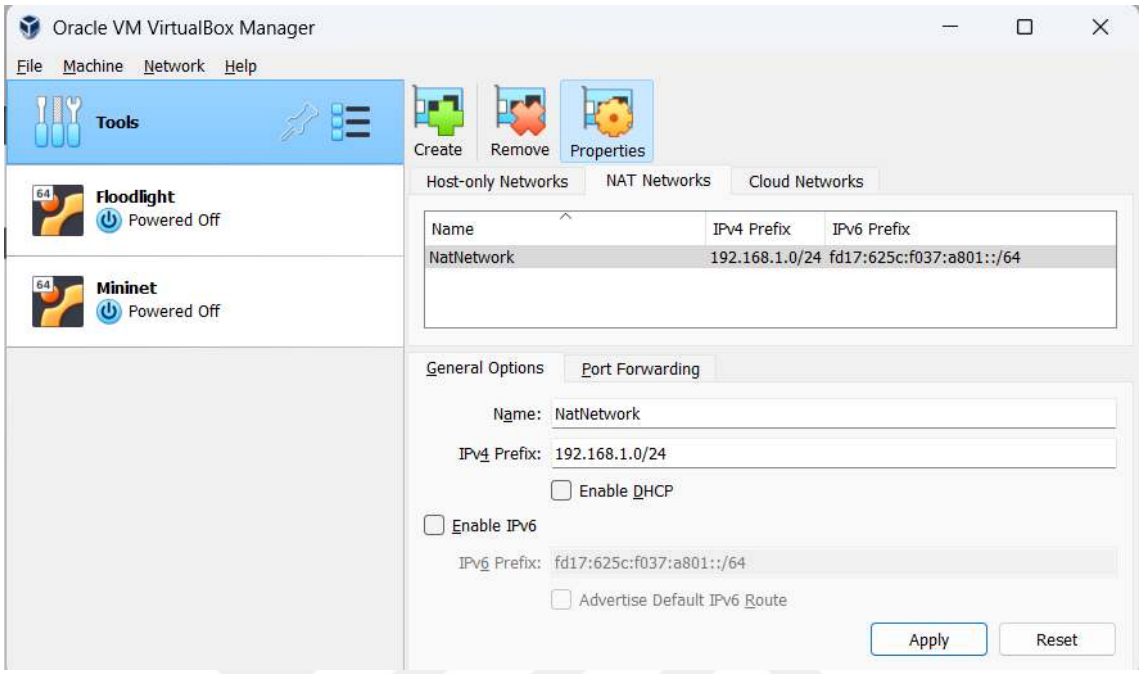

Figure 4.5. *NAT network setup*

To test the connectivity between the two virtual machine we will create a Mininet topology and connect it to our controller and will check the connectivity between the hosts. Figure 4.6 shows that the 3 hosts from our test topology can ping each other successfully without any packet loss, while Figure 4.7 shows the GUI of our topology from the Floodlight controller side.

```
*** Adding controller
*** Adding hosts
*** Adding switch
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Testing connectivity
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

Figure 4.6. *Mininet connectivity test topology*

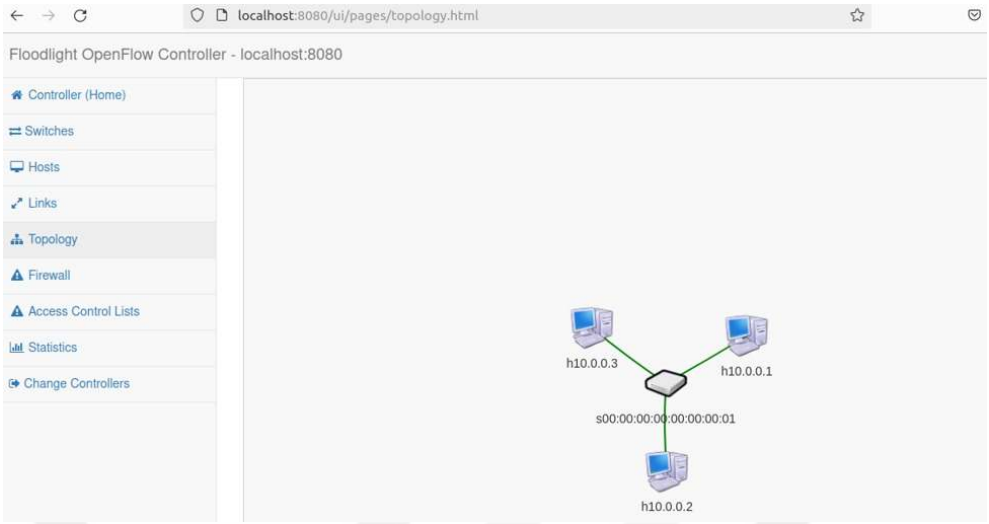

Figure 4.7. *Floodlight controller GUI*

4.2. Simulation Test and Results

The experimental simulation was designed to evaluate the effectiveness and dependability of the suggested SDN-based IoT smart home security framework in real-world attack scenarios. The emphasis was on assessing the system's capability to identify, alleviate, and restore operations following a Distributed Denial of Service (DDoS) attack launched from the internal network. To effectively assess the influence of the security measures, the experiment was carried out in two separate stages:

Phase I – Pre-Mitigation Assessment

Phase II – After-Mitigation Examination (active protection employing fixed flow regulations and IPsec)

Every phase replicates the communication among various IoT devices and features an attack scenario where an unauthorized device (Intruder Host) tries to overwhelm the network by sending TCP SYN packets through the hping3 tool.

4.2.1. IPsec authentication

To provide authentication and security between the IoT home system and the SDN controller, IPsec tunnel mode was implemented. The reason why IPsec was chosen is that can provide encryption for all data which is sent from the switches to the controller and the other way around which helps to defend the network from eavesdrop and MITM attacks. Furthermore, IPsec provides mutual authentication which will deny access for

unauthorized devices and ensure that exchanged date happens only from authorized devices. IPsec connection employs IKEv2 using robust encryption algorithms (AES-256 and SHA-256). PSK authentication is employed on both ends to establish trust and a Dead Peer Detection mechanism is used with a 30-second check interval to ensure the reliability of the connection. To create IPsec protocol two files should be modified from the controller side and the IoT home system side and they are the ipsec.conf and ipsec.secrets file which done as shown in Figure 4.8 and Figure 4.9.

```
Controller side:
•   sudo nano /etc/ipsec.conf
“ config setup
    charondebug=ike 2, knl 2, cfg 2

conn %default
    keyexchange=ikev2
    ike=aes256-sha256-modp1024
    esp=aes256-sha256
    dpdaction=restart
    dpddelay=30s
    rekey=no
    leftauth=psk
    rightauth=psk

conn secure-traffic
    left=192.168.1.6
    leftsubnet=192.168.1.6/32
    right=192.168.1.5
    rightsubnet=192.168.1.5/32
    auto=start “

•   sudo nano /etc/ipsec.secrets
192.168.1.6 192.168.1.5 : PSK "Password123@"
```

Figure 4.8. *Securing the tunnel from the controller side*

```
IoT Home system side:
•   sudo nano /etc/ipsec.conf
" config setup
    charondebug=ike 2, knl 2, cfg 2

conn %default
    keyexchange=ikev2
    ike=aes256-sha256-modp1024
    esp=aes256-sha256
    dpdaction=restart
    dpddelay=30s
    rekey=no
    leftauth=psk
    rightauth=psk

conn secure-traffic
    left=192.168.1.5
    leftsubnet=192.168.1.5/32
    right=192.168.1.5
    rightsubnet=192.168.1.6/32
    auto=start "

•   sudo nano /etc/ipsec.secrets
192.168.1.5 192.168.1.6 : PSK "Password123@"
```

Figure 4.9. *Securing the tunnel from the IoT home system side*

After adjusting the files on both sides, we should restart the IPsec by using the following command “sudo ipsec restart” in order to apply the changes. After the adjustment has been made IPsec Tunnels will be initiated between the end to end connection as shown in Figure 4.10. Furthermore, we can verify that the IPsec Tunnel was successfully established by capturing packets which are sent between the two devices using Wireshark as show in the figure 4.11 and we can see from the figure that the packets have ESP protocol and the data inside it is encrypted.

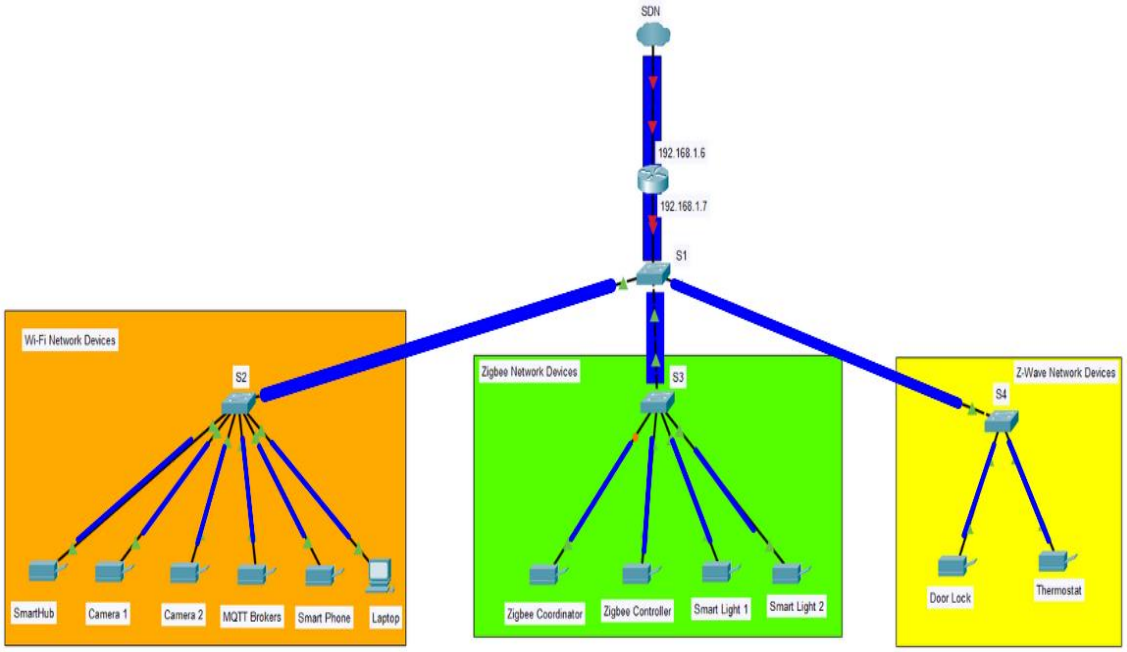

Figure 4.10. *System topology with IPsec protection*

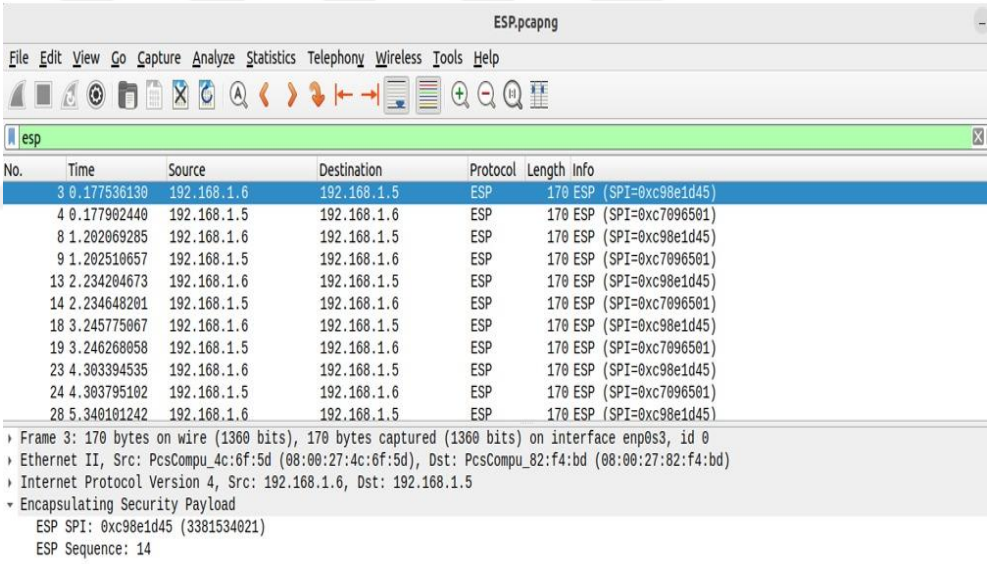

Figure 4.11. *Secured packets between controller and IoT home system*

4.2.2. Static flow rules

For the safety of the system, we restricted the communication between the home devices, and we allowed only the devices which use the same protocols to communicate with each other, we excluded the phone user from these rules to allow the user to control his home through his phone. The applied flow rules are shown in Figure 4.12 where each

device is trying to reach all the home devices but as shown not all devices are able to reach each other.

```
mininet> pingall
*** Ping: testing ping reachability
smartHub -> camera1 camera2 zcoor stlight1 slight2 X thermostat mqttBroker userP
hone userLaptop
camera1 -> smartHub camera2 X X X X X mqttBroker userPhone userLaptop
camera2 -> smartHub camera1 X X X X X mqttBroker userPhone userLaptop
zcoor -> smartHub X X stlight1 slight2 X X X userPhone X
stlight1 -> smartHub X X zcoor slight2 X X X userPhone X
slight2 -> smartHub X X zcoor stlight1 X X X userPhone X
doorLock -> X X X X X X thermostat X userPhone X
thermostat -> smartHub X X X X X doorLock X userPhone X
mqttBroker -> smartHub camera1 camera2 X X X X X userPhone userLaptop
userPhone -> smartHub camera1 camera2 zcoor stlight1 slight2 doorLock thermostat
 mqttBroker userLaptop
userLaptop -> smartHub camera1 camera2 X X X X X mqttBroker userPhone
*** Results: 49% dropped (56/110 received)
```

Figure 4.12. *Applied flow rules on the system*

Figure 4.13 depicts the decision-making process employed in the SDN controller to implement protocol-based communication guidelines among IoT devices in the smart home network. This system is a component of the static flow rule setup implemented via the Floodlight REST API and aims to minimize the lateral spread of threats throughout the internal network.

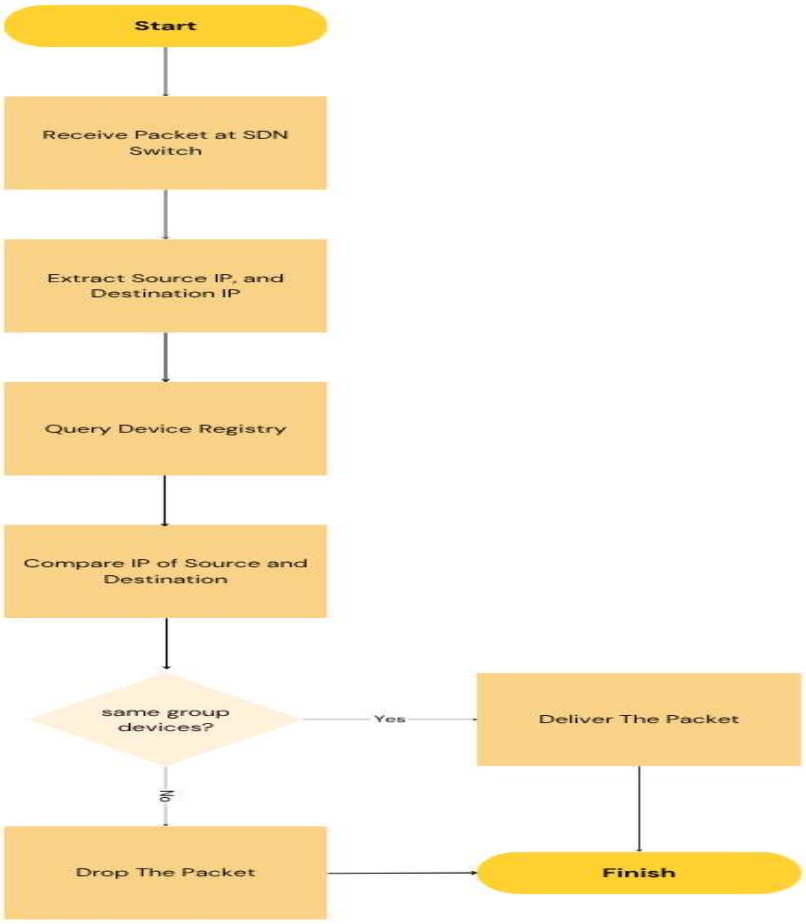

Figure 4.13. *Proposed system's workflow*

The system functions in the following manner:

The process initiates when a packet arrives at a switch that supports OpenFlow in an SDN environment.

Capture Packet at SDN Switch: The switch intercepts the packet and sends it to the SDN controller for additional examination (if there is no corresponding flow rule).

Obtain Source and Destination IP: The controller analyzes the packet headers to retrieve the IP addresses of the source and destination devices.

Query Device Registry: By utilizing a predefined device registry (a mapping of device IP addresses to their permitted communication protocols), the controller obtains the anticipated protocol details for both devices.

Compare Protocols of Source and Destination: The controller assesses if the communication protocols utilized by the source and destination devices are compatible. This ensures that only similar categories of devices (e.g., MQTT-compatible sensors or RTSP-supporting cameras) are able to exchange packets.

Decision Point – Same group devices?

Yes → When the protocol types are identical, the controller directs the switch to transmit the packet, and a flow rule might be established to permit similar traffic in the future.

No → If the protocols are incompatible, the packet is discarded, stopping any unauthorized or irregular communication. This is particularly useful in thwarting lateral attacks where compromised devices attempt to scan or link to others through unauthorized protocols.

Record & Conclude: No matter the result, the occurrence might be documented for auditing or anomaly detection reasons, and the process finishes.

This reasoning guarantees a policy-driven segmentation of the smart home network, permitting communication solely between devices within the same protocol group. It greatly minimizes the attack surface and enhances overall network cleanliness.

The flow rules can be applied through REST API of the Floodlight Controller through the CURL command, the Rules is available in appendix (B)

4.2.3. DDoS attack scenario

The attack scenario has been launched from Camera2 and the MQTT Broker devices at the same time assuming that the intruder managed to get access through these devices and aimed at the Phone user as it is the most important device in the system. In Figure 4.14 shows the controller statues which shows that the system is working normally and there is no threat coming from any switch, then the attack has been launched by using Xterm command in Mininet to interact with Camera2 and the MQTT Broker devices, and then we launched the attack through “hping3 –flood -S -p 80 10.0.0.200” in the two devices to launch dual attack. While the attack was starting, we checked the connectivity of the other hosts and found out that all the hosts were unreachable because the controller is overwhelmed from the attack. Figure 4.15 shows that a threat has been detected from the Wi-Fi switch and the controller is trying to mitigate the threat.

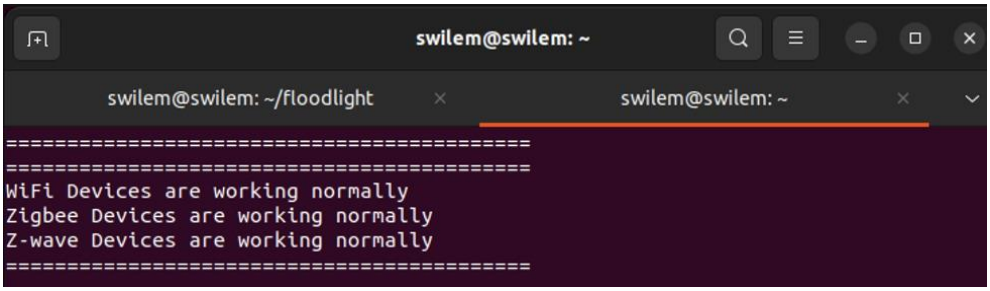

Figure 4.14. *Controller status before the attack*

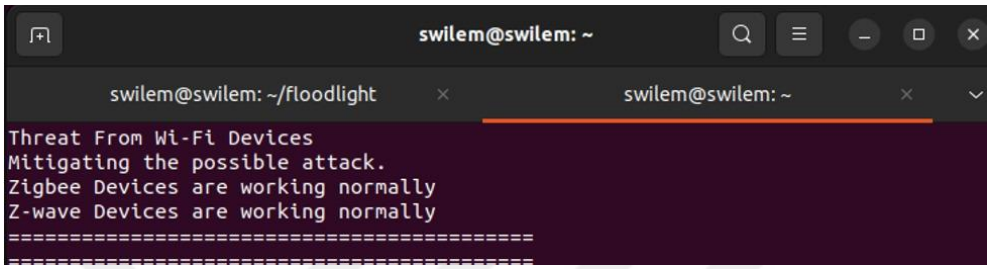

Figure 4.15. *Attack detection*

After the attack is being detected the system will work on mitigating the attack and after the attack is mitigated the system will show the IP address of the devices that launched the attack and will deny any further packet coming from these devices and allow the rest of the devices to work normally as shown in Figure 4.16.

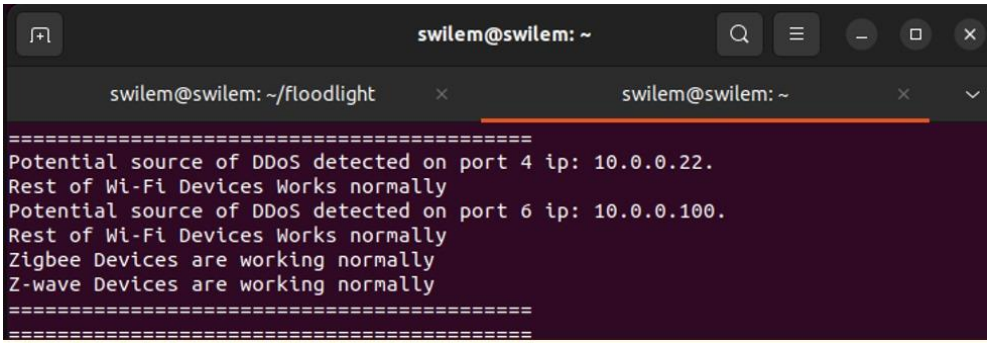

Figure 4.16. *Mitigating process*

The Mitigation prosses has been done using python script in appendix C and the mitigation steps as follows:

Step1: disable all the active connections from the SDN controller to avoid overwhelming.

Step2: Find the port that sent the huge number of packets and keep it disable but re-enable the connection to the other ports.

Step3: check the switch that is connected to the defrauded port and check the devices which are connected to this switch to find the devices which are causing the attack.

Step4: disable the devices that causes the attack and re-enable the switch to communicate normally with the controller.

Figure 4.17 shows the network topology after the mitigation has been done.

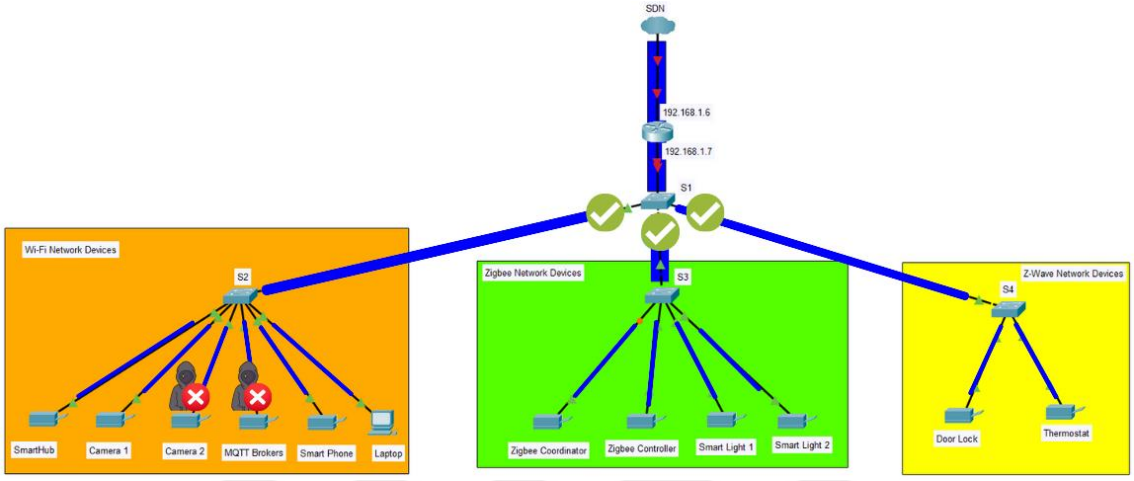

Figure 4.17. *System topology*

The system took approximately 119 seconds to mitigate the attack as shown in Figure 4.18 where the packets were captured from the phone user side which showed that the attack has been successfully stopped and the system back to normal.

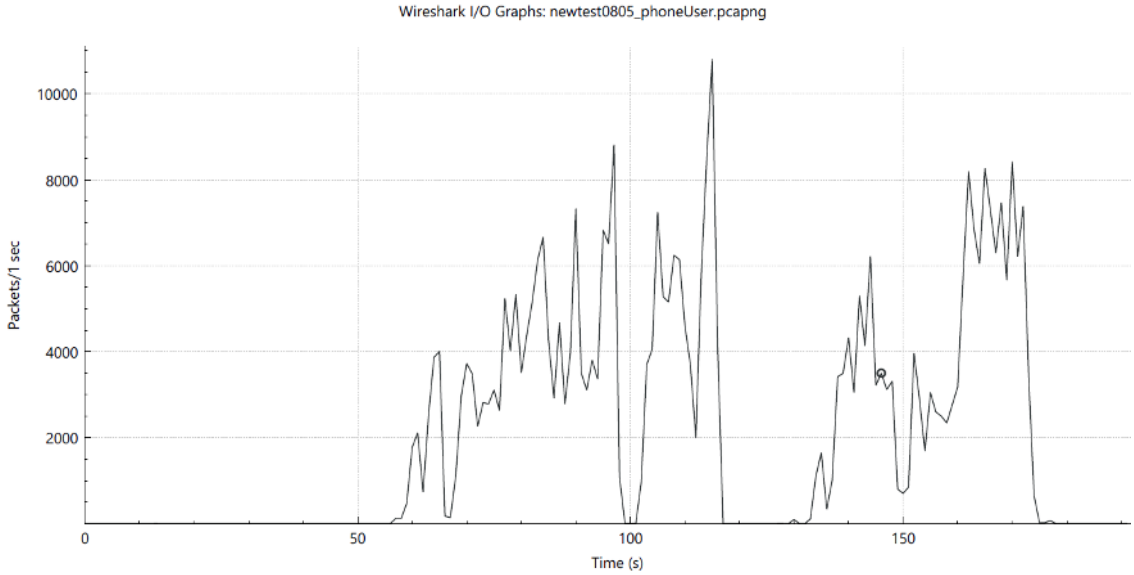

Figure 4.18. *I/O packets graph*

In Figure 4.19 shows the CPU performance of the controller The CPU utilization graph shows clear stages in system activity throughout the simulation. Throughout the majority

of the observed timeframe, CPU usage stays fairly low and consistent, varying between 20% and 50%. Nonetheless, there are numerous significant peaks, particularly:

- At approximately 17:46:52, the CPU usage reaches nearly 300%, probably indicating the start of a DDoS attack or a traffic spike caused by the attacker.
- Minor yet notable spikes are also observed at 17:46:15, 17:47:22, and 17:47:55, suggesting possible burst patterns or sporadic mitigation efforts (e.g., flow rule modifications, IPsec tunnel verifications).
- These peaks are closely related to the expected load generated in the pre- and post-mitigation stages of the experiment. The SDN controller seems to manage these spikes effectively, as CPU usage reliably reverts to normal following each occurrence.

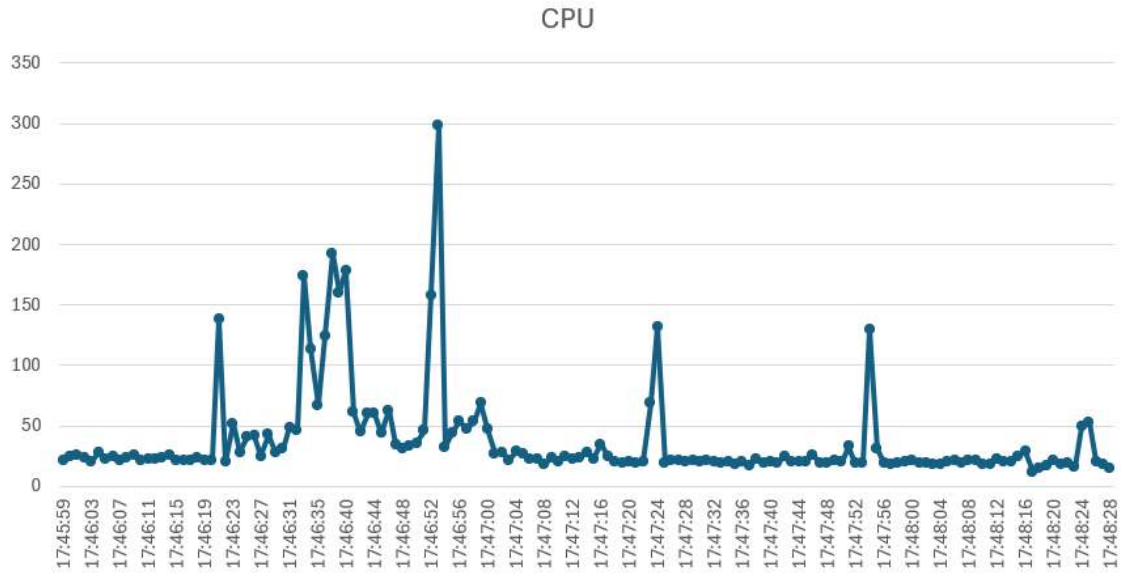

Figure 4.19. *CPU Utilization*

4.3. Summary

In this chapter we have started by setting up the network and prepared our two virtual machines by installing all the required applications and we created our NAT network. Furthermore, installed the flow rules to restrict the communication between the devices and activated IPsec protocol between the controller and the home system and lastly we launched DDoS attack and tested the efficiency of the system in mitigating the attack.

5. CONCLUSION

This study aimed to tackle a significant issue in contemporary networked systems: the requirement for adaptive, smart, and automated protections against cyber threats, especially within software-defined networking (SDN) contexts. This thesis has shown a feasible and efficient method for network protection by integrating a comprehensive understanding of SDN architecture with practical attack scenarios and enforcement of flow rules at the controller level.

The essence of this study focused on devising and executing a system to track, assess, and manage packet transmission according to the compatibility of protocol types among communicating devices. By employing a meticulously designed series of OpenFlow rules alongside a programmable controller, we developed a system that can detect and address traffic that breaches established policy compatibility standards. This proactive strategy improves network security by stopping potential attack pathways before they can spread through the system.

The results of the simulation confirmed the theory supporting this method. When faced with harmful traffic—like unusual UDP flooding from an attacking node—the controller effectively filtered and discarded unauthorized packets, reducing their effect on the system. The gathered performance metrics, such as CPU utilization, memory usage, and disk I/O, distinctly demonstrated the system's activity throughout various phases of the simulation. Peaks in resource consumption aligned with attack occurrences, and the swift return to normal afterwards showcased the success and efficacy of the mitigation approach. These outcomes emphasize the feasibility of implementing such solutions in actual SDN applications.

Additionally, this thesis enhances the field by offering a workflow-oriented approach that network engineers and researchers can utilize to establish and implement security policies within SDN infrastructures. The flexibility and modularity of this method also allow it to be adapted for various protocol-based validation rules, broadening the possible applications beyond the examined scenario.

In summary, this study represents a significant advancement in automated security management based on SDN. The effective execution and evaluation of a protocol-checking rule enforcement system show that SDN's programmability can be utilized not just for dynamic routing and load balancing but also for strong and intelligent threat

prevention. This paves the way for future research in domains like AI-based rule generation, decentralized policy synchronization, and enhanced anomaly detection within SDN settings.

REFERENCES

[1] P. Baniya, A. Agrawal, K. Abid, J. Nath, B. K. Chaudhary, and B. Kunwar, "The Internet of Things: Security Challenges and Opportunities," in *2024 3rd International conference on Power Electronics and IoT Applications in Renewable Energy and its Control (PARC)*, 2024, pp. 153–158. doi: 10.1109/PARC59193.2024.10486356.

[2] A. A. Ali, O. Al-Blooshi, R. A. Ali, and H. A. Hamadi, "Securing the Internet of Things (IoT) Application: Best Practices and Challenges in IoT Software," in *2024 Advances in Science and Engineering Technology International Conferences (ASET)*, 2024, pp. 1–7. doi: 10.1109/ASET60340.2024.10708648.

[3] I. Bedhief, M. Kassar, and T. Aguili, *SDN-based architecture challenging the IoT heterogeneity*. 2016. doi: 10.1109/SCNS.2016.7870558.

[4] M. Kumar, U. Yadav, P. Raghav, and J. D. Kumar, "Securing the Internet of Things (IoT): Current Landscape, Obstacles, and Future Strategies," *CONFERENCE PROCEEDING*, 2024, [Online]. Available: https://api.semanticscholar.org/CorpusID:270043883 (Access date: 11.11.2024)

[5] M. SafaeiSisakht, C.-H. Hsu, P.-Y. Hsu, and M.-Y. Chen, *An Intelligent Two-Phase Automated Architecture for Securing SDN-Based IoT Infrastructure*. 2023. doi: 10.1109/ICEIB57887.2023.10170386.

[6] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in *2014 27th Biennial Symposium on Communications (QBSC)*, 2014, pp. 71–75. doi: 10.1109/QBSC.2014.6841187.

[7] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements," *IEEE Access*, vol. 5, pp. 1872–1899, 2017, doi: 10.1109/ACCESS.2017.2666200.

[8] A. H. Shamsan and A. R. Faridi, "SDN-Assisted IoT Architecture: A Review," in *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, 2018, pp. 1–7. doi: 10.1109/CCAA.2018.8777339.

[9] F. Ahmad, M. Saleem, and U. ur Rehman, "Detection and Mitigation of Malicious DDoS Floods in Software Defined Networks," *Journal of Electrical Electronics Engineering*, no. 4, June 2023, [Online]. Available: https://api.semanticscholar.org/CorpusID:265201423 (Access date: 15.11.2024)

[10] N. Dayal and S. Srivastava, "FloodKnight: an intelligent DDoS defense scheme to combat attacks near attack entry points," *J. Comput. Virol. Hacking Tech.*, vol. 20, pp. 819–839, 2024, [Online]. Available: https://api.semanticscholar.org/CorpusID:271732203 (Access date: 15.11.2024)

[11] N. Aslam, S. Srivastava, and M. M. Gore, "ONOS Flood Defender: An Intelligent Approach to Mitigate DDoS Attack in SDN," *Transactions on Emerging*

Telecommunications Technologies, vol. 33, no. 9, p. e4534, 2022, doi: https://doi.org/10.1002/ett.4534. (Access date: 15.11.2024)

[12] S. Batool *et al.*, "Lightweight Statistical Approach towards TCP SYN Flood DDoS Attack Detection and Mitigation in SDN Environment," *Secur. Commun. Networks*, vol. 2022, pp. 2593672:1-2593672:14, 2022, [Online]. Available: https://api.semanticscholar.org/CorpusID:246316124 (Access date: 15.11.2024)

[13] S. Kalash, N. Makarem, L. Issa, A. Tajeddine, and N. Abbas, "Detection and Prevention of TCP DoS/DDoS Attacks in Software Defined Network," *2023 IEEE 4th International Multidisciplinary Conference on Engineering Technology (IMCET)*, pp. 50–55, 2023, [Online]. Available: https://api.semanticscholar.org/CorpusID:266601757 (Access date: 15.11.2024)

[14] T. Shaikh, "Sequential Heuristic Model for DDOS Attack Detection in Software-Defined Network," *2023 20th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pp. 614–620, 2023, [Online]. Available: https://api.semanticscholar.org/CorpusID:273432210 (Access date: 18.11.2024)

[15] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013, doi: 10.1109/MCOM.2013.6461195.

[16] S. Bhaskaran and S. M, "Study on Networking Models of SDN using Mininet and MiniEdit," in *2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*, 2024, pp. 1159–1164. doi: 10.1109/IDCIoT59759.2024.10467218.

[17] W. Hu, Y. Liu, P. Yu, X. Li, and X. Dai, "Design of Road Traffic Management and Control Network Architecture Based on SDN: An Innovative Approach," in *2024 International Conference on Cyber-Physical Social Intelligence (ICCSI)*, 2024, pp. 1–6. doi: 10.1109/ICCSI62669.2024.10799454.

[18] M. H. Rempola, A. Smith, Y. Li, and L. Du, "Securing SDN Communication through Quantum Key Distribution," in *2024 IEEE Transportation Electrification Conference and Expo (ITEC)*, 2024, pp. 1–5. doi: 10.1109/ITEC60657.2024.10598919.

[19] J. Bian, Y. Chen, Z. Song, W. Jiao, and Y. Chi, "Research on Cloud Computing Network Security Design Based on OpenFlow," in *2024 IEEE 7th International Conference on Information Systems and Computer Aided Education (ICISCAE)*, 2024, pp. 707–714. doi: 10.1109/ICISCAE62304.2024.10761237.

[20] A. Charoenphol and M. J. Reed, "Link Fabrication Attack Detection for Software Defined Networks," in *2024 International Conference on Intelligent Computing, Communication, Networking and Services (ICCNS)*, 2024, pp. 1–8. doi: 10.1109/ICCNS62192.2024.10776339.

[21] P. Chatterjee and D. B. Rawat, "Security Enhanced Framework for Network Access Control in Distributed Software-Defined Networks," in *2024 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2024, pp. 1816–1821. doi: 10.1109/ICCWorkshops59551.2024.10615557.

[22] T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” 2008.

[23] H. Akçay and D. Yiltas, “Web-Based User Interface for the Floodlight SDN Controller,” *International Journal of Advanced Networking and Applications*, vol. 8, pp. 3175–3180, Apr. 2017.

[24] T. Darwish, T. Alhaj, and F. Elhaj, “Controller placement in software defined emerging networks: a review and future directions,” *Telecommun. Syst.*, vol. 88, p. 18, 2025, doi: 10.1007/s11235-024-01252-0.

[25] A. Trisnandar and M. T. Kurniawan, “A Survey on Techniques Used to Improve Network Performance and Flexibility in Software Defined Network,” in *2024 8th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, 2024, pp. 603–608. doi: 10.1109/ICITISEE63424.2024.10729893.

[26] D. Khope, “SOFTWARE DEFINE NETWORK AND IOT,” *Gurukul International Multidisciplinary Research Journal*, p., 2024, doi: 10.69758/gimrj2406i8v12p119.

[27] J. Agujiobi, “Software Defined Netconomics,” *International Journal of Science and Research Archive*, p., 2024, doi: 10.30574/ijsra.2024.11.1.0203.

[28] I. Mahar, L. Wu, Z. Maher, and G. Rahu, “A Comprehensive Survey of Software Defined Networking and its Security Threats,” *2024 IEEE 1st Karachi Section Humanitarian Technology Conference (KHI-HTC)*, pp. 1–5, 2024, doi: 10.1109/KHI-HTC60760.2024.10482096.

[29] S. Farooq, S. Riaz, and A. Alvi, “Security and Privacy Issues in Software-Defined Networking (SDN): A Systematic Literature Review,” *Electronics (Basel)*, vol. 12, p. 3077, Jul. 2023, doi: 10.3390/electronics12143077.

[30] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, “Security in Software Defined Networks: A Survey,” *IEEE Communications Surveys & Tutorials*, vol. 17, p. 1, Jan. 2015, doi: 10.1109/COMST.2015.2474118.

[31] Q. Yan, F. R. Yu, Q. Gong, and J. Li, “Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 602–622, 2016, doi: 10.1109/COMST.2015.2487361.

[32] J. Xia, Z. Cai, G. Hu, and M. Xu, “An Active Defense Solution for ARP Spoofing in OpenFlow Network,” *Chinese Journal of Electronics*, vol. 28, pp. 172–178, Jan. 2019, doi: 10.1049/cje.2017.12.002.

[33] N. Ahuja, G. Singal, D. Mukhopadhyay, and A. Nehra, “Ascertain the efficient machine learning approach to detect different ARP attacks,” *Computers & Electrical Engineering*, vol. 99, p. 107757, Apr. 2022, doi: 10.1016/j.compeleceng.2022.107757.

[34] F. Shoaib, Y.-W. Chow, E. Vlahu-Gjorgievska, and C. Nguyen, “Mitigating Timing Side-Channel Attacks in Software-Defined Networks: Detection and Response,” *Telecom*, vol. 4, no. 4, pp. 877–900, 2023, doi: 10.3390/telecom4040038.

[35] D. Smyth, S. McSweeney, D. O'Shea, and V. Cionca, "Detecting Link Fabrication Attacks in Software-Defined Networks," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, 2017, pp. 1–8. doi: 10.1109/ICCCN.2017.8038435.

[36] J. Hua, Z. Zhou, and S. Zhong, "Flow Misleading: Worm-Hole Attack in Software-Defined Networking via Building In-Band Covert Channel," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1029–1043, 2021, doi: 10.1109/TIFS.2020.3013093.

[37] A. Pradhan and R. Mathew, "Solutions to Vulnerabilities and Threats in Software Defined Networking (SDN)," *Procedia Comput Sci*, vol. 171, pp. 2581–2589, Jan. 2020, doi: 10.1016/j.procs.2020.04.280.

[38] A. Alhaj and N. Dutta, "Analysis of Security Attacks in SDN Network: A Comprehensive Survey," 2022, pp. 27–37. doi: 10.1007/978-981-16-4244-9_3.

[39] A. Sebbar, M. Boulmalf, M. D. E.-C. El Kettani, and Y. Baddi, "Detection MITM Attack in Multi-SDN Controller," in *2018 IEEE 5th International Congress on Information Science and Technology (CiSt)*, 2018, pp. 583–587. doi: 10.1109/CIST.2018.8596479.

[40] F. Hauser, M. Häberle, M. Schmidt, and M. Menth, "P4-IPsec: Site-to-Site and Host-to-Site VPN with IPsec in P4-Based SDN," *IEEE Access*, vol. 8, p. 1, Jan. 2020, doi: 10.1109/ACCESS.2020.3012738.

[41] K.-K. R. Choo, R. Chaudhary, A. Jindal, G. Aujla, S. Aggarwal, and N. Kumar, "BEST: Blockchain-based Secure Energy Trading in SDN-enabled Intelligent Transportation System," *Comput Secur*, May 2019, doi: 10.1016/j.cose.2019.05.006.

http-1: https://api.semanticscholar.org/CorpusID:195837615 (Access date: 02.03.2025)

http-2: https://docs.openvswitch.org/_/downloads/en/latest/pdf/ (Access date: 05.03.2025)

http-3:https://www.opennetworking.org/wpcontent/uploads/2014/10/openflow-spec-v1.4.0.pdf. (Access date: 05.03.2025)

http-4:https://www.comptia.org/content/articles/what-is-wireshark-and-how-to-use-it. (Access date: 05.03.2025)

http-5:
https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343539/Floodlight+REST+API. (Access date: 05.03.2025)

http-6:https://deliveredsocial.com/software-defined-networking-sdns-benefits-challenges-applications/ (Access date: 10.03.2025)

APPENDIX A: NETWORK TOPOLOGY

```
from mininet.net import Mininet
from mininet.node import RemoteController, OVSKernelSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink

def createSmartHomeWithProtocols():
    net = Mininet(controller=RemoteController, link=TCLink, switch=OVSKernelSwitch)

    info('*** Adding controller\n')
    c0 = net.addController('c0', controller=RemoteController, ip='192.168.1.6', port=6653)

    info('*** Adding switches\n')
    s1 = net.addSwitch('s1')  # Main switch
    s2 = net.addSwitch('s2')  # Wi-Fi network
    s3 = net.addSwitch('s3')  # Zigbee network
    s4 = net.addSwitch('s4')  # Z-Wave network

    info('*** Adding hosts (IoT devices)\n')
    # Wi-Fi devices
    smartHub = net.addHost('smartHub', ip='10.0.0.10')
    camera1 = net.addHost('camera1', ip='10.0.0.21')
    camera2 = net.addHost('camera2', ip='10.0.0.22')

    # Zigbee devices
    zcoor = net.addHost('zcoor', ip='10.0.0.31')
    slight1 = net.addHost('stlight1', ip='10.0.0.32')
    slight2 = net.addHost('slight2', ip='10.0.0.33')

    # Z-Wave devices
```

```
doorLock = net.addHost('doorLock', ip='10.0.0.42')
thermostat = net.addHost('thermostat', ip='10.0.0.43')



# MQTT broker
mqttBroker = net.addHost('mqttBroker', ip='10.0.0.100')

# User devices
userPhone = net.addHost('userPhone', ip='10.0.0.200')
userLaptop = net.addHost('userLaptop', ip='10.0.0.201')

info('*** Adding links\n')
# Connect switches
net.addLink(s1, s2, cls=TCLink, bw=100)
net.addLink(s1, s3, cls=TCLink, bw=100)
net.addLink(s1, s4, cls=TCLink, bw=100)

# Connect Wi-Fi devices

net.addLink(s2, smartHub, cls=TCLink, bw=50)
net.addLink(s2, camera1, cls=TCLink, bw=20)
net.addLink(s2, camera2, cls=TCLink, bw=20)
net.addLink(s2, mqttBroker, cls=TCLink, bw=50)

# Connect Zigbee devices
net.addLink(s3, zcoor, cls=TCLink, bw=10)
net.addLink(s3, slight1, cls=TCLink, bw=5)
net.addLink(s3, slight2, cls=TCLink, bw=5)

# Connect Z-Wave devices
net.addLink(s4, doorLock, cls=TCLink, bw=5)
```

```
    net.addLink(s4, thermostat, cls=TCLink, bw=5)

    # Connect user devices
    net.addLink(s2, userPhone, cls=TCLink, bw=50)
    net.addLink(s2, userLaptop, cls=TCLink, bw=100)

    info('*** Starting network\n')
    net.build()
    c0.start()
    s1.start([c0])
    s2.start([c0])
    s3.start([c0])
    s4.start([c0])

    info('*** Running CLI\n')
    CLI(net)

    info('*** Stopping network')
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    createSmartHomeWithProtocols()
```

APPENDIX B: FLOW RULES

```
curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "deny-camera1-to-zcont",
  "cookie": "0",
  "priority": "32768",
  "eth_type": "0x0800",
  "ipv4_src": "10.0.0.21/32",
  "ipv4_dst": "10.0.0.42/32",
  "actions": "drop"
}' http://192.168.1.6:8080/wm/staticflowpusher/json

curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "deny-camera1-to-thermostat",
  "cookie": "0",
  "priority": "32768",
  "eth_type": "0x0800",
  "ipv4_src": "10.0.0.21/32",
  "ipv4_dst": "10.0.0.43/32",
  "actions": "drop"
}' http://192.168.1.6:8080/wm/staticflowpusher/json


curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "deny-camera2-to-zcont",
  "cookie": "0",
  "priority": "32768",
  "eth_type": "0x0800",
  "ipv4_src": "10.0.0.22/32",
  "ipv4_dst": "10.0.0.42/32",
  "actions": "drop"
```

```
}' http://192.168.1.6:8080/wm/staticflowpusher/json

curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "deny-camera2-to-thermostat",
  "cookie": "0",
  "priority": "32768",
  "eth_type": "0x0800",
  "ipv4_src": "10.0.0.22/32",
  "ipv4_dst": "10.0.0.43/32",
  "actions": "drop"
}' http://192.168.1.6:8080/wm/staticflowpusher/json



curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "deny-BLE-Device-to-zcont",
  "cookie": "0",
  "priority": "32768",
  "eth_type": "0x0800",
  "ipv4_src": "10.0.0.60/32",
  "ipv4_dst": "10.0.0.42/32",
  "actions": "drop"
}' http://192.168.1.6:8080/wm/staticflowpusher/json


curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "deny-BLE-Device-to-thermostat",
  "cookie": "0",
  "priority": "32768",
  "eth_type": "0x0800",
```

```
  "ipv4_src": "10.0.0.60/32",
  "ipv4_dst": "10.0.0.43/32",
  "actions": "drop"
}' http://192.168.1.6:8080/wm/staticflowpusher/json


curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "deny-mqttBroker-to-thermostat",
  "cookie": "0",
  "priority": "32768",
  "eth_type": "0x0800",
  "ipv4_src": "10.0.0.100/32",
  "ipv4_dst": "10.0.0.43/32",
  "actions": "drop"
}' http://192.168.1.6:8080/wm/staticflowpusher/json


curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "deny-mqttBroker-to-zcont",
  "cookie": "0",
  "priority": "32768",
  "eth_type": "0x0800",
  "ipv4_src": "10.0.0.100/32",
  "ipv4_dst": "10.0.0.42/32",
  "actions": "drop"
}' http://192.168.1.6:8080/wm/staticflowpusher/json


curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
```

```
  "name": "deny-laptop-to-thermostat",
  "cookie": "0",
  "priority": "32768",
  "eth_type": "0x0800",
  "ipv4_src": "10.0.0.201/32",
  "ipv4_dst": "10.0.0.43/32",
  "actions": "drop"
}' http://192.168.1.6:8080/wm/staticflowpusher/json


curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "deny-laptop-to-zcont",
  "cookie": "0",
  "priority": "32768",
  "eth_type": "0x0800",
  "ipv4_src": "10.0.0.201/32",
  "ipv4_dst": "10.0.0.42/32",
  "actions": "drop"
}' http://192.168.1.6:8080/wm/staticflowpusher/json

curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "deny-all-from-locker",
  "cookie": "0",
  "priority": "3000",
  "eth_type": "0x0800",
  "ipv4_src": "0.0.0.0/0",
  "ipv4_dst": "10.0.0.42/32",
  "actions": "drop"
}' http://192.168.1.6:8080/wm/staticflowpusher/json
```

```
curl -X POST -d '{
  "switch": "00:00:00:00:00:00:00:01",
  "name": "allow-phone-to-locker",
  "cookie": "0",
  "priority": "4000",
  "eth_type": "0x0800",
  "ipv4_src": "10.0.0.200/32",
  "ipv4_dst": "10.0.0.42/32",
  "actions": "output=normal"
}' http://192.168.1.6:8080/wm/staticflowpusher/json
```

APPENDIX C: MITIGATION CODE

```
import requests
import json
import time

def add_flow_rule(data):
    url = "http://192.168.1.6:8080/wm/staticflowpusher/json"
    headers = {'Content-Type': 'application/json'}
    response = requests.post(url, data=json.dumps(data), headers=headers)

def delete_flow_rule(payload):
    url = "http://192.168.1.6:8080/wm/staticflowpusher/json"
    headers = {'Content-Type': 'application/json'}
    response = requests.delete(url, headers=headers, data=json.dumps(payload))



def get_switch2_ports(controller_ip, switch_2):
    url = f"http://{controller_ip}:8080/wm/core/switch/{switch_2}/port/json"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        if "port_reply" in data and data["port_reply"]:
            port_data = {}
            for port in data["port_reply"][0]["port"]:
                port_number = port["port_number"]
                receive_packets = int(port["receive_packets"])
                port_data[port_number] = receive_packets
            return port_data
        else:
            print(f"Threat From Wi-Fi Devices")



            data = {
```

```
                    "switch": "00:00:00:00:00:00:00:02",
                    "name": "deny-all-from-s2",
                    "cookie": "0",
                    "priority": "4000",
                    "eth_type": "0x0800",
                    "ipv4_src": "0.0.0.0/0",
                    "ipv4_dst": "0.0.0.0/0",
                    "actions": "drop"
                    }
        add_flow_rule(data)

        return None
    else:
        print(f"Failed to retrieve port information for switch {switch_id}. Error: {response.text}")
        return None

def get_switch3_ports(controller_ip, switch_3):
    url = f"http://{controller_ip}:8080/wm/core/switch/{switch_3}/port/json"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        if "port_reply" in data and data["port_reply"]:
            port_data = {}
            for port in data["port_reply"][0]["port"]:
                port_number = port["port_number"]
                receive_packets = int(port["receive_packets"])
                port_data[port_number] = receive_packets
            return port_data
        else:
            print(f"Threat From ZigBee Devices")
```

```
        data = {
                    "switch": "00:00:00:00:00:00:00:03",
                    "name": "deny-all-from-s3",
                    "cookie": "0",
                    "priority": "4000",
                    "eth_type": "0x0800",
                    "ipv4_src": "0.0.0.0/0",
                    "ipv4_dst": "0.0.0.0/0",
                    "actions": "drop"
                    }
        add_flow_rule(data)

        return None
    else:
        print(f"Failed to retrieve port information for switch {switch_id}. Error: {response.text}")
        return None

def get_switch4_ports(controller_ip, switch_4):
    url = f"http://{controller_ip}:8080/wm/core/switch/{switch_4}/port/json"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        if "port_reply" in data and data["port_reply"]:
            port_data = {}
            for port in data["port_reply"][0]["port"]:
                port_number = port["port_number"]
                receive_packets = int(port["receive_packets"])
                port_data[port_number] = receive_packets
            return port_data
        else:
            print(f"Threat From Z-Wave Devices")
```

```
            data = {
                        "switch": "00:00:00:00:00:00:00:04",
                        "name": "deny-all-from-s4",
                        "cookie": "0",
                        "priority": "4000",
                        "eth_type": "0x0800",
                        "ipv4_src": "0.0.0.0/0",
                        "ipv4_dst": "0.0.0.0/0",
                        "actions": "drop"
                        }
            add_flow_rule(data)

            return None
    else:
        print(f"Failed to retrieve port information for switch {switch_id}. Error: {response.text}")
        return None

def check_ddos_s2(controller_ip, switch_2, threshold):
    port_data = get_switch2_ports(controller_ip, switch_2)
    if port_data is not None:
        ddos_detected = False
        for port, packets in port_data.items():
            if port == "local":
                continue

            if packets >= threshold:
                ddos_detected = True

                # Check connected switches
                if port == "1":
```

```
        print(f"Potential source of DDoS detected on port {port}.")

    elif port == "2":

        data = {
                "switch": "00:00:00:00:00:00:00:02",
                "name": "deny-all-from-smartHub",
                "cookie": "0",
                "priority": "3000",
                "eth_type": "0x0800",
                "ipv4_src": "10.0.0.10/32",
                "ipv4_dst": "0.0.0.0/0",
                "actions": "drop"
                }
        add_flow_rule(data)
        print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.10.")
        # Perform the DELETE request
        payload = {"name": "deny-all-from-s2"}
        delete_flow_rule(payload)
        print(f"Rest of Wi-Fi Devices Works normally ")

    elif port == "3":

        data = {
                "switch": "00:00:00:00:00:00:00:02",
                "name": "deny-all-from-Camera1",
                "cookie": "0",
                "priority": "3000",
                "eth_type": "0x0800",
                "ipv4_src": "10.0.0.21/32",
                "ipv4_dst": "0.0.0.0/0",
                "actions": "drop"
                }
```

```
    add_flow_rule(data)
    print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.21.")
    # Perform the DELETE request
    payload = {"name": "deny-all-from-s2"}
    delete_flow_rule(payload)
    print(f"Rest of Wi-Fi Devices Works normally ")

elif port == "4":

    data = {
            "switch": "00:00:00:00:00:00:00:02",
            "name": "deny-all-from-Camera2",
            "cookie": "0",
            "priority": "3000",
            "eth_type": "0x0800",
            "ipv4_src": "10.0.0.22/32",
            "ipv4_dst": "0.0.0.0/0",
            "actions": "drop"
            }
    add_flow_rule(data)
    print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.22.")
    # Perform the DELETE request
    payload = {"name": "deny-all-from-s2"}
    delete_flow_rule(payload)
    print(f"Rest of Wi-Fi Devices Works normally ")

elif port == "5":

    data = {
            "switch": "00:00:00:00:00:00:00:02",
            "name": "deny-all-from-BLE-Device",
            "cookie": "0",
            "priority": "3000",
```

```
            "eth_type": "0x0800",
            "ipv4_src": "10.0.0.60/32",
            "ipv4_dst": "0.0.0.0/0",
            "actions": "drop"
            }
    add_flow_rule( data)
    print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.60.")

    # Perform the DELETE request
    payload = {"name": "deny-all-from-s2"}
    delete_flow_rule(payload)
    print(f"Rest of Wi-Fi Devices Works normally ")

elif port == "6":

    data = {
            "switch": "00:00:00:00:00:00:00:02",
            "name": "deny-all-from-MQTT-Broker",
            "cookie": "0",
            "priority": "3000",
            "eth_type": "0x0800",
            "ipv4_src": "10.0.0.100/32",
            "ipv4_dst": "0.0.0.0/0",
            "actions": "drop"
            }
    add_flow_rule(data)
    print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.100.")
    # Perform the DELETE request
    payload = {"name": "deny-all-from-s2"}
    delete_flow_rule(payload)
    print(f"Rest of Wi-Fi Devices Works normally ")

elif port == "7":
```

```
    data = {
            "switch": "00:00:00:00:00:00:00:02",
            "name": "deny-all-from-Mobil-User",
            "cookie": "0",
            "priority": "3000",
            "eth_type": "0x0800",
            "ipv4_src": "10.0.0.200/32",
            "ipv4_dst": "0.0.0.0/0",
            "actions": "drop"
            }
    add_flow_rule(data)
    print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.200.")
    # Perform the DELETE request
    payload = {"name": "deny-all-from-s2"}
    delete_flow_rule(payload)
    print(f"Rest of Wi-Fi Devices Works normally ")

elif port == "8":

    data = {
            "switch": "00:00:00:00:00:00:00:02",
            "name": "deny-all-from-Laptop-user",
            "cookie": "0",
            "priority": "3000",
            "eth_type": "0x0800",
            "ipv4_src": "10.0.0.201/32",
            "ipv4_dst": "0.0.0.0/0",
            "actions": "drop"
            }
    add_flow_rule( data)
    print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.201.")
    # Perform the DELETE request
```

```
                payload = {"name": "deny-all-from-s2"}
                delete_flow_rule(payload)
                print(f"Rest of Wi-Fi Devices Works normally ")

        if not ddos_detected:
            print("WiFi Devices are working normally")

    else:
        print("Mitigating the possible attack.")

def check_ddos_s3(controller_ip, switch_3, threshold):
    port_data = get_switch3_ports(controller_ip, switch_3)
    if port_data is not None:
        ddos_detected = False
        for port, packets in port_data.items():
            if port == "local":
                continue

            if packets >= threshold:
                ddos_detected = True

                # Check connected switches
                if port == "1":

                    print(f"Potential source of DDoS detected on port {port}.")

                elif port == "2":

                    data = {
                            "switch": "00:00:00:00:00:00:00:03",
                            "name": "deny-all-from-Zcordinate",
                            "cookie": "0",
                            "priority": "3000",
```

```
            "eth_type": "0x0800",
            "ipv4_src": "10.0.0.31/32",
            "ipv4_dst": "0.0.0.0/0",
            "actions": "drop"
            }
        add_flow_rule(data)
        print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.31.")
        # Perform the DELETE request
        payload = {"name": "deny-all-from-s3"}
        delete_flow_rule(payload)

    elif port == "3":

        data = {
            "switch": "00:00:00:00:00:00:00:03",
            "name": "deny-all-from-Smart-Light1",
            "cookie": "0",
            "priority": "3000",
            "eth_type": "0x0800",
            "ipv4_src": "10.0.0.32/32",
            "ipv4_dst": "0.0.0.0/0",
            "actions": "drop"
            }
        add_flow_rule(data)
        print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.32.")
        # Perform the DELETE request
        payload = {"name": "deny-all-from-s3"}
        delete_flow_rule(payload)

    elif port == "4":

        data = {
            "switch": "00:00:00:00:00:00:00:03",
```

```
                    "name": "deny-all-from-Smart-Light2",
                    "cookie": "0",
                    "priority": "3000",
                    "eth_type": "0x0800",
                    "ipv4_src": "10.0.0.33/32",
                    "ipv4_dst": "0.0.0.0/0",
                    "actions": "drop"
                    }
                add_flow_rule(data)
                print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.33.")
                # Perform the DELETE request
                payload = {"name": "deny-all-from-s3"}
                delete_flow_rule(payload)

        if not ddos_detected:
            print("Zigbee Devices are working normally")

    else:
        print("Mitigating the possible attack.")

def check_ddos_s4(controller_ip, switch_4, threshold):
    port_data = get_switch4_ports(controller_ip, switch_4)
    if port_data is not None:
        ddos_detected = False
        for port, packets in port_data.items():
            if port == "local":
                continue

            if packets >= threshold:
                ddos_detected = True

                # Check connected switches
                if port == "1":
```

```
        print(f"Potential source of DDoS detected on port {port}.")

    elif port == "2":

        data = {
                "switch": "00:00:00:00:00:00:00:04",
                "name": "deny-all-from-Zcontroller",
                "cookie": "0",
                "priority": "3000",
                "eth_type": "0x0800",
                "ipv4_src": "10.0.0.41/32",
                "ipv4_dst": "0.0.0.0/0",
                "actions": "drop"
                }
        add_flow_rule(data)
        print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.51.")
        # Perform the DELETE request
        payload = {"name": "deny-all-from-s4"}
        delete_flow_rule(payload)

    elif port == "3":

        data = {
                "switch": "00:00:00:00:00:00:00:04",
                "name": "deny-all-from-Door-Lock",
                "cookie": "0",
                "priority": "3000",
                "eth_type": "0x0800",
                "ipv4_src": "10.0.0.42/32",
                "ipv4_dst": "0.0.0.0/0",
                "actions": "drop"
                }
```

```
                add_flow_rule(data)
                print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.42.")
                # Perform the DELETE request
                payload = {"name": "deny-all-from-s4"}
                delete_flow_rule(payload)

            elif port == "4":

                data = {
                        "switch": "00:00:00:00:00:00:00:04",
                        "name": "deny-all-from-Thermostat",
                        "cookie": "0",
                        "priority": "3000",
                        "eth_type": "0x0800",
                        "ipv4_src": "10.0.0.43/32",
                        "ipv4_dst": "0.0.0.0/0",
                        "actions": "drop"
                        }
                add_flow_rule(data)
                print(f"Potential source of DDoS detected on port {port} ip: 10.0.0.43.")
                # Perform the DELETE request
                payload = {"name": "deny-all-from-43"}
                delete_flow_rule(payload)

        if not ddos_detected:
            print("Z-wave Devices are working normally")

    else:
        print("Mitigating the possible attack.")

if __name__ == "__main__":

    threshold = 700000  # Threshold of received packets
```

```
controller_ip = "192.168.1.6"
switch_2 = "00:00:00:00:00:00:00:02"
switch_3  = "00:00:00:00:00:00:00:03"
switch_4  = "00:00:00:00:00:00:00:04"


while True:
    print("========================================")
    check_ddos_s2(controller_ip, switch_2, threshold)
    check_ddos_s3(controller_ip, switch_3, threshold)
    check_ddos_s4(controller_ip, switch_4, threshold)
    print("========================================")
    # Wait for a certain interval before checking again
    time.sleep(10)
```

CURRICULUM VITAE

ORCID ID: 0009-0007-1127-1745

Name Surname : Yousef I. A. Mohamed

Education and Professional Background:

- 2016-2020, Eastern Mediterranean University, Bachelor of Science | Electrical and Electronic Engineering (ABET Accredited)
- 2020-2021, Sales Engineer, SwilemGroup Company
- Apr. 2021–Sep. 2021, Radio Frequency Engineer, Huawei Telecommunication Company
- 2022-2025, Eskişehir Technical University, Master of Science in Electrical and Electronics Engineering, Telecommunications Programme