



# A Prevention Technique for DDoS Attacks in SDN using Ryu Controller Application

Sudheer Devanaboina  
Yashwanth Adabala

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering with emphasis on Telecommunication Systems. The thesis is equivalent to 20 weeks of full-time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

**Contact Information:**

Author(s):

Sudheer Devanaboina

E-mail: lade21@student.bth.se

Yashwanth Adabala

E-mail: yaad21@student.bth.se

University advisor:

Dr. Dragos Ilie

Department of Computer Science

Faculty of Computing  
Blekinge Institute of Technology  
SE-371 79 Karlskrona, Sweden

Internet : [www.bth.se](http://www.bth.se)  
Phone : +46 455 38 50 00  
Fax : +46 455 38 50 57

---

# Abstract

Software-Defined Networking (SDN) modernizes network control, offering streamlined management. However, its centralized structure makes it more vulnerable to Distributed Denial of Service (DDoS) attacks, posing serious threats to network stability. This thesis explores the development of a DDoS attack prevention technique in SDN environments using the Ryu controller application. The research aims to address the vulnerabilities in SDN, particularly focusing on flooding and Internet Protocol (IP) spoofing attacks, which are a significant threat to network security. The study employs an experimental approach, utilizing tools like Mininet-VM (Virtual Machine), Oracle VM VirtualBox, and hping3 to simulate a virtual SDN environment and conduct DDoS attack scenarios. Key methodologies include packet sniffing and rule-based detection by integrating Snort IDS (Intrusion Detection System), which is critical for identifying and mitigating such attacks. The experiments demonstrate the effectiveness of the proposed prevention technique, highlighting the importance of proper configuration and integration of network security tools in SDN. This work contributes to enhancing the resilience of SDN architectures against DDoS attacks, offering insights into future developments in network security.

**Keywords:** DDoS Attacks, Software Defined Networking, Snort IDS, Network Security

---

## Acknowledgments

We would like to express our gratitude to our supervisor, Dragos Ilie, for the invaluable guidance and support throughout the duration of our thesis work. We are deeply thankful to our families and friends, whose constant encouragement and unwavering belief in us were the driving forces that helped us persevere and successfully complete this journey.

---

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgments</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background and Motivation . . . . .	1
1.2 Aim and Objectives . . . . .	2
1.3 Research Questions . . . . .	2
1.4 Outline . . . . .	3
<b>2 Methodology and Related Work</b>	<b>4</b>
2.1 Methodology . . . . .	4
2.2 Summaries of related papers . . . . .	5
<b>3 Key Technologies</b>	<b>9</b>
3.1 Introduction to SDN . . . . .	9
3.2 Fundamentals of SDN . . . . .	10
3.3 SDN Devices and Protocol . . . . .	12
3.3.1 SDN Controller . . . . .	12
3.3.2 SDN Switch . . . . .	12
3.3.3 OpenFlow Protocol . . . . .	14
3.3.4 Packet flow in SDN . . . . .	15
3.4 Vulnerabilities in SDN . . . . .	17
3.5 DDOS attacks . . . . .	17
3.5.1 TCP SYN Flood attack . . . . .	18
3.5.2 Internet Control Message Protocol (ICMP) Flood . . . . .	19
3.5.3 UDP Flood attack . . . . .	19
3.5.4 IP spoofing attacks . . . . .	20
3.6 Snort IDS . . . . .	20
3.6.1 Types of IDS . . . . .	20
3.6.2 Advanced Features and Performance Analysis of Snort IDS . .	22
<b>4 Method</b>	<b>24</b>
4.1 Research Design . . . . .	24
4.2 Tools and Technologies . . . . .	24
4.2.1 Mininet-VM . . . . .	24
4.2.2 Oracle VM VirtualBox . . . . .	25
4.2.3 PuTTY . . . . .	25

4.2.4	Xming X server . . . . .	25
4.2.5	Ryu Controller . . . . .	26
4.2.6	hping3 . . . . .	26
4.3	Configuring the SDN environment . . . . .	26
4.4	Installation of Snort IDS . . . . .	27
4.4.1	Integration of Snort IDS . . . . .	28
4.5	Configuring Ryu Controller Application . . . . .	29
4.5.1	Formation of mininet network . . . . .	29
4.5.2	Ryu Controller Application . . . . .	31
4.5.3	Configuration of Ryu to mitigate IP spoofing . . . . .	34
4.5.4	Configuration of Ryu to mitigate Flooding attacks . . . . .	36
4.5.5	Configuration of Ryu to mitigate flooding and IP spoofing . . . . .	38
<b>5</b>	<b>Experiments and Results</b>	<b>40</b>
5.1	Experiments . . . . .	40
5.1.1	Experiment 1: Evaluation of Snort3's Detection Capabilities . . . . .	40
5.1.2	Experiment 2: Impact and Mitigation of IP Spoofing Attacks on SDN . . . . .	45
5.1.3	Experiment 3 . . . . .	49
<b>6</b>	<b>Discussion</b>	<b>51</b>
6.1	Introduction . . . . .	51
6.2	Evaluation of Experiments . . . . .	52
6.2.1	Evaluation for Experiment 1 . . . . .	52
6.2.2	Evaluation for Experiment 2 . . . . .	53
6.2.3	Evaluation of Experiment 3 . . . . .	53
6.3	Evaluation of Research questions . . . . .	54
6.4	Limitations . . . . .	55
<b>7</b>	<b>Conclusions and Future Work</b>	<b>56</b>
7.1	Conclusion . . . . .	56
7.2	Future Work . . . . .	57
	<b>References</b>	<b>58</b>

---

# Nomenclature

API	Application Programming Interface
ASICs	Application Specific Integrated Circuits
CAMs	Content-Addressable Memories
CDNi	Content Delivery Network Interconnection
CLI	Command Line Interface
COTS	Commercial Off-The-Shelf
DDoS	Distributed Denial of Service
Dos	Denial of Service
FTE	flow table entries
GUIs	Graphical User Interfaces
HIDS	Host Based Intrusion Detection System
HOC	Half-Open Connection
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IPC	Inter Process Communication
IT	Information Technology
LSDDoS	Low and Slow Distributed Denial of Service
MAC	Media Access Control
MIB	Management Information Base
NBI	Northbound Interface
NICs	Network Interface Cards
NIDS	Network Based Intrusion Detection System

NOS Network Operating System  
ONF Open Networking Foundation  
OS Operating System  
OVS Open vSwitch  
Qos Quality of Service  
RAM Random Access Memory  
SBI Southbound Interface  
SDN Software Defined Networks  
SDRAM Synchronous Dynamic Random Access Memory  
SSH Secure Shell  
SSM Smart Security Mechanism  
TCAMs Ternary Content-Addressable Memories  
TCB Transmission Control Block  
TCP Transfer Control Protocol  
UDP User Datagram Protocol  
URI Uniform Resource Identifier  
VM Virtual Machine  
WBI Westbound Interface  
X Xming X server



---

## List of Figures

3.1	Traditonal Networking vs Software-Defined Networking . . . . .	10
3.2	SDN Architecture . . . . .	11
3.3	OpenFlow Switch . . . . .	13
3.4	Flowchart for packet flow . . . . .	16
3.5	Snort Architecture . . . . .	21
4.1	SDN Architecture using miniedit . . . . .	30
4.2	Mininet console after initiating the network . . . . .	31
4.3	Output in RYU controller . . . . .	33
4.4	Flowrules in SDN Switch . . . . .	33
4.5	Flowchart for IP spoof mitigation configuration . . . . .	35
4.6	Flowchart for flooding mitigation configuration . . . . .	37
5.1	ICMP Flood Alerts . . . . .	41
5.2	ICMP Spoof Alerts . . . . .	42
5.3	TCP Flood Alerts . . . . .	42
5.4	TCP Spoof Alerts . . . . .	43
5.5	UDP Flood Alerts . . . . .	43
5.6	UDP Spoof Alerts . . . . .	44
5.7	A graph which depicts the average time with increase in number of attackers under IP spoofing attack and under mitigation . . . . .	46
5.8	A graph which depicts the percentage of CPU Utilization of Switch under IP spoofing attack . . . . .	46
5.9	A graph which depicts the percentage of Memory Utilization of Switch under IP spoofing attack . . . . .	47
5.10	A graph which depicts the number of flow rules with time increasing under IP spoofing attack . . . . .	48
5.11	A graph which depicts the number of flowrules with time increasing under mitigation . . . . .	48
5.12	A graph which depicts the average time with increase in number of attackers under flooding attack and under mitigation . . . . .	49

---

## List of Tables

5.1	Commands used for different types of attacks . . . . .	41
5.2	Table explaining the command used to implement Snort IDS . . . . .	41

---

## List of Algorithms

1	Algorithm for sdn network in mininet . . . . .	30
2	SimpleSwitch13 Ryu App for Network Management . . . . .	32
3	Configuring Ryu Controller to Block IP Spoofing . . . . .	34
4	Ryu Controller Configuration for Flooding . . . . .	36
5	Configuring Ryu Controller to Block IP Spoofing and Flooding . . . .	39



### 1.1 Background and Motivation

In the world of network technology, Software-Defined Networking (SDN) has emerged as a groundbreaking paradigm, offering a transformative approach to network management and operation. This innovative framework has fundamentally redefined the architecture of networking by decoupling the control and data planes, leading to a more flexible, efficient, and programmable network environment. SDN's centralized control mechanism enables network administrators to dynamically adjust network behaviors to meet changing needs, facilitating improved resource utilization, easier network management, and enhanced user experiences.

However, the very features that make SDN so advantageous also introduce new vulnerabilities, particularly in the context of network security. Among these, Distributed Denial of Service (DDoS) attacks represent a significant threat. DDoS attacks are notoriously known for their ability to flood networks with overwhelming traffic, disrupting services and causing extensive damage. In the context of SDN, these attacks can exploit the centralized nature of the control plane, potentially crippling the network's operational capabilities.

The heightened susceptibility of SDN to DDoS attacks necessitates the development of effective and resilient defense mechanisms. Traditional network security solutions often fall short in addressing the unique challenges posed by the SDN architecture. Therefore, there is a pressing need for innovative strategies specifically tailored to protect SDN environments against the scourge of DDoS attacks. This research endeavors to address this gap by exploring and developing a prevention technique that is not only effective in mitigating DDoS attacks but also congruent with the intrinsic properties of SDN.

This research investigates the specific threats posed by flooding and IP spoofing attacks, which cause service interruptions and manipulate network communication by using false IP addresses. To detect and mitigate both flooding and IP spoofing attacks, the study explores the implementation of Snort IDS (Intrusion Detection System), a widely used open-source tool designed to identify and respond to various types of network threats. The significance of this research is underscored by the increasing reliance on SDN in various critical sectors, including data centers, enterprise networks, and cloud computing services. Ensuring the security and reliability of SDN is paramount, not just for maintaining operational efficiency but also for safeguarding sensitive data and critical infrastructure against the disruptive and potentially catastrophic impacts of DDoS attacks. This study aims to contribute to the

field by presenting a comprehensive approach to fortifying SDN against these threats, thereby enhancing the overall resilience and dependability of network infrastructures in the face of evolving cyber threats.

## 1.2 Aim and Objectives

The purpose of this study is to thoroughly examine DDoS attacks in the context of SDN. By conducting a systematic investigation and experiments, the thesis aims to improve our comprehension of different aspects of DDoS attacks in SDN setups. This includes understanding their different types, assessing their impact, and developing methods to detect and prevent them effectively.

To accomplish the main goal of this research, the following specific objectives are formulated:

1. **To investigate how to detect the occurrence of flooding and IP spoofing attacks.**  
Explore existing detection methods for identifying flooding and IP spoofing attacks in traditional networking environments and adapt and extend these methods to the SDN paradigm, considering the unique characteristics and capabilities of SDN architectures.
2. **To explore different DDoS attack mitigation techniques in SDN.**  
Understand the strengths and weaknesses of current approaches within the unique context of SDN architectures. Based on this comprehensive evaluation, develop and propose a mitigation technique to effectively counteract DDoS threats.
3. **To conduct experimental evaluations and to analyze and interpret findings.**  
Design and implement a controlled experimental setup to simulate various DDoS attack scenarios in an SDN environment and analyze the experimental results while SDN is under attack and mitigation.

## 1.3 Research Questions

The following research questions are formulated to achieve the above objectives:

1. **To what extent does integration of Snort IDS in SDN demonstrate effectiveness in detecting flooding and IP spoofing attacks?**  
To assess the capabilities of the Snort IDS in identifying and mitigating the specific DDoS attack types prevalent in SDN environments, thereby gauging its applicability as an intrusion detection solution within such contexts.
2. **How can Snort IDS be integrated into an SDN environment to configure a mitigation technique for flooding and IP spoofing attacks?**  
To configure an SDN attack detection and prevention technique and test the configured technique in a controlled experimental setup.

### 3. Find the effects of flooding and spoofing attacks on SDN environment while under attack and mitigation

To analyze the impact of flooding and spoofing attacks on SDN environment while under attack and mitigation in a controlled experimental setup.

## 1.4 Outline

The thesis is organized into several chapters, each focusing on distinct aspects of DDoS attacks in SDN.

**Chapter 2: Methodology and Related Work** - This chapter presents a literature review of existing research and methods in the field of SDN, DDoS attacks, and IDS. It highlights various approaches and methodologies used by researchers and practitioners.

**Chapter 3: Key Technologies** - This chapter introduces the basic concepts and technologies related to SDN, including an overview of the SDN architecture, devices, and protocols. It also explores the vulnerabilities in SDN, specifically focusing on DDoS attacks and the role of Intrusion Detection Systems (IDS), with a detailed analysis of Snort IDS.

**Chapter 4: Method** - The methodology of the research is outlined in this chapter. It describes the research design, tools, and technologies utilized in the study, including the configuration of the SDN environment, the installation of Snort IDS, and the setup of the Ryu controller application.

**Chapter 5: Results** - This chapter reports on the experimental results obtained from implementing and testing the DDoS attack prevention techniques in an SDN environment. It includes detailed analysis and discussion of the findings from various experiments.

**Chapter 6: Discussion** - The implications and significance of the experimental findings are discussed in this chapter. It provides an interpretation of the results in the context of SDN security and DDoS attack prevention.

**Chapter 7: Conclusions and Future Work** - The final chapter concludes the thesis with a summary of the findings, their implications for network security, and suggestions for future research directions in the field.

## Chapter 2

---

# Methodology and Related Work

This chapter discusses about the research methodology used and also summarizes related papers in the field of SDN

### 2.1 Methodology

For an overview of the existing research in SDN, DDoS attacks, and mitigation, a literature review is conducted. It aims to provide a comprehensive understanding of the advancements, challenges, and methodologies in the realm of SDN and DDoS mitigation techniques. The review also identifies gaps in current research, offering insights into potential areas for further investigation and development in the field. This process is crucial for grounding the thesis in the existing body of knowledge.

The databases used to collect the research papers are IEEE Explore, Google Scholar, and Science Direct. To achieve a thorough and relevant review, the research papers were selected based on specific search criteria. These included the use of certain search strings such as

1. "Software-Defined Networking AND DDoS Mitigation"
2. "Ryu Controller Application AND DDoS Prevention"
3. "SDN Security Measures AND Distributed Denial of Service"
4. "DDoS Attack Detection in SDN Environments"
5. "Ryu SDN Controller AND Network Security"
6. "DDoS Resilience Techniques in Software-Defined Networks"
7. "Performance Analysis of SDN under DDoS Attacks"
8. "Effective DDoS Handling in SDN with Ryu Controller"

Establishing clear inclusion criteria is crucial for ensuring the relevance and quality of literature in the study. This approach helps in filtering and selecting studies that are directly pertinent to the research objectives, providing a focused and robust foundation for the analysis. The criteria followed for this study is as follows:

1. Papers specifically addressing DDoS attacks in SDN.



2. Research focusing on the use of the Ryu controller for network security or DDoS mitigation.
3. Articles written in English.
4. Recent publications, preferably within the last ten years, to ensure up-to-date information.
5. Literature providing comparative analysis of different DDoS mitigation techniques in SDN environments.

The process involved a careful screening of abstracts to identify papers that directly related to the thesis topic, followed by an in-depth study of these selected papers. This approach ensured that the literature review was not only comprehensive and current but also directly aligned with the research questions and objectives of the thesis.

## 2.2 Summaries of related papers

The paper [8], discusses a DDoS mitigation solution using OpenFlow. OpenFlow monitors traffic flow statistics to detect potential DDoS attacks. It can mirror traffic for suspicious flows to an IDS integrated into the OpenFlow controller, which then analyzes the traffic to identify attack sources. The paper proposes two methods for DDoS attack identification in traffic flows: analyzing packet symmetry and temporarily blocking outgoing traffic to identify persistent sources. These methods are incorporated into custom OpenFlow controller software. The effectiveness of this approach is tested through experiments and simulations.

The paper [21], explores a DDoS mitigation approach using SDN. It introduces a scheme that blocks DDoS attacks from botnets in an SDN environment using standard OpenFlow interfaces. The method emphasizes minimal server involvement, operating through a DDoS blocking application on the SDN controller. Key features include monitoring network traffic, identifying bots, and redirecting legitimate traffic. The system's effectiveness is demonstrated through emulation in Mininet, highlighting its potential in countering sophisticated botnet-based DDoS attacks without statistical anomalies.

The paper [24], presents a comprehensive defense mechanism against DDoS attacks in SDN environments. It highlights the advantages of SDN in combating DDoS attacks, such as centralized control and network programmability. The paper outlines a framework involving various modules like binding, location tracking, packet filtering, and port statistic queries. These modules work together to detect and mitigate DDoS attacks effectively. The paper includes a case study demonstrating the proposed defense mechanism's effectiveness in a simulated SDN environment.

The paper [11], presents a modular architecture for detecting and mitigating network anomalies in SDN environments, leveraging OpenFlow and sFlow protocols. It introduces a method to decouple data collection from the SDN control plane, improving scalability and reducing controller overload. The approach employs an entropy-based algorithm for anomaly detection, and its effectiveness is validated through

experiments with real network traffic. The paper demonstrates how the proposed method effectively mitigates detected anomalies using flow table modifications in an SDN environment.

The paper [23], proposes FL-GUARD, a system for detecting and preventing DDoS attacks in SDN. The system uses dynamic IP address binding to address IP spoofing and employs a C-SVM algorithm for attack detection. It leverages the centralized control of SDN to issue flow tables for blocking attacks at the source port. Experimental results demonstrate the effectiveness of FL-GUARD, highlighting its modular design which facilitates future improvements.

The paper [47], proposes an innovative approach for DDoS attack detection within SDN environments. It introduces a distributed detection mechanism that operates at the edge switches of an SDN network, using an entropy-based model to identify anomalies indicative of DDoS attacks. This method significantly reduces the communication load between switches and the central controller, and is shown to detect attacks effectively and with high accuracy, addressing the challenges of scale and responsiveness in large SDN deployments.

The paper [6], introduces a comprehensive system for detecting and mitigating DDoS attacks in SDN environments. The system, named SD-Anti-DDoS, consists of four modules: attack detection trigger, attack detection, attack traceback, and attack mitigation. It utilizes a neural network-based approach for attack detection and leverages the unique properties of SDN for efficient attack traceback and mitigation. The paper evaluates the system's effectiveness in a testbed environment, demonstrating its capability to rapidly initiate attack detection and accurately trace and block attack sources.

The paper [27], presents a novel approach for defending against DDoS attacks in SDN based Content Delivery Network Interconnection (CDNi) systems. The proposed mechanism integrates OpenFlow table tweaking and a unique marking path map in the ALTO server for enhanced defense. It focuses on the security of content provider servers by using protection switches and a Management Information Base (MIB) in the SDN controller for traffic assessment. This multi-layered defense strategy aims to efficiently secure CDNi networks against DDoS attacks.

The paper [36], focuses on DDoS attack detection and mitigation in SDN environments using machine learning. It introduces a discrete scalable memory-based support vector machine algorithm for detecting DDoS threats and outlines a mitigation architecture. The approach includes preprocessing using Spark standardization, feature extraction using semantic multilinear component analysis, and classification with high accuracy. The effectiveness of this method is demonstrated using the KDD dataset in an SDN environment.

The paper [54], proposes a novel mechanism for cyberattack mitigation in IoT networks using SDN and Network Function Virtualization (NFV). The approach employs virtual IoT honeynets, which are dynamically deployed and managed to distract and analyze attackers, providing enhanced security for IoT systems. The framework integrates with existing SDN and NFV infrastructures, offering a scalable and effective solution for protecting IoT networks against various cyber threats.

The paper [45], examines the implications of cloud computing and SDN on DDoS attack defense mechanisms. It highlights how the integration of cloud computing and SDN introduces new challenges and opportunities for DDoS defense. The paper

proposes a novel DDoS attack mitigation architecture, DaMask, which employs a graphical probabilistic inference model for attack detection and a flexible control structure for rapid and specific attack response. The architecture's effectiveness is validated through simulations, demonstrating its potential in addressing security challenges in modern network paradigms.

The paper [10], introduces OPERETTA, an OpenFlow-based solution for mitigating TCP SYN flood attacks in SDN environments. OPERETTA is implemented in the SDN controller to manage incoming TCP SYN packets and filter out fake connection requests. The solution is designed to work in heterogeneous networks and can be implemented in both centralized and decentralized SDN architectures. The paper demonstrates OPERETTA's effectiveness through simulations, showing improved resilience to TCP SYN flood attacks with reduced CPU and memory consumption.

The paper [48], presents a novel security networking mechanism for defending against DDoS attacks in SDN. This mechanism, called SDSNM, focuses on redefining network architecture to remove or restrict the conditions necessary for DDoS attacks. The paper details the system's design, using OpenFlow and cloud computing technologies, and evaluates its effectiveness through a prototype implementation. The SDSNM system demonstrates a significant improvement in attack detection and mitigation, showcasing its potential for scalable and effective DDoS defense in SDN environments.

The paper [39] focuses on developing an autonomic DDoS mitigation mechanism leveraging the SDN paradigm. It proposes a distributed collaborative framework allowing customers to request DDoS mitigation services from ISPs. This framework enables ISPs to redirect anomalous traffic to security middleboxes while maintaining privacy and legal compliance. The paper's preliminary analysis indicates SDN's promising potential in enabling autonomic mitigation of DDoS attacks and other large-scale threats.

The paper [41], proposes a reference architecture for defending against Low and Slow DDoS (LSDDoS) attacks in Software Defined Infrastructure (SDI) environments. It details two specific architectures: a performance model-based approach and an approach using Commercial Off-The-Shelf (COTS) components. The paper introduces the concept of a "Shark Tank," a monitored environment for analyzing suspicious traffic. This work demonstrates a novel approach to detecting and mitigating LSDDoS attacks by leveraging SDI capabilities.

The paper [51], addresses the vulnerability of SDN-based IoT systems to new-flow attacks, which exhaust network resources by generating a high volume of unmatched packets. It proposes a Smart Security Mechanism (SSM) that leverages standard SDN interfaces for efficient attack detection and mitigation. The mechanism employs a low-cost monitoring method and dynamic access control, significantly enhancing the system's ability to identify and respond to such attacks in SDN-based IoT environments.

The paper [16], proposes an automated DDoS mitigation and traffic management system for SDN environments. It combines hierarchical clustering-based traffic learning, blacklist integration, and dynamic server capacity invocation to effectively defend against packet and bandwidth flooding attacks. The system is tested on a physical SDN testbed, demonstrating its ability to maintain service quality during

DDoS attacks, with varying levels of effectiveness depending on the attack type and complexity.

The paper [46] presents a new architecture for a hybrid honeypot system based on SDN. This architecture aims to improve upon traditional hybrid honeypots by enhancing network topology simulation and attack traffic migration, using the expansibility and controllability of the SDN controller. The proposed system can simulate large, realistic networks to attract attackers and redirect high-level attacks to a high-interaction honeypot for detailed analysis. The paper validates the effectiveness of this system through experiments conducted on the Mininet platform.

The paper [50] addresses the vulnerability of SDN controllers to DoS and DDoS attacks. It introduces FlowRanger, a novel algorithm that prioritizes routing requests based on the likelihood of them being attack attempts. The algorithm classifies requests into multiple buffer queues with varying priorities, thereby enhancing the controller's ability to handle legitimate requests under attack. Simulation results demonstrate FlowRanger's effectiveness in improving request serving rates and reducing the impact of DoS attacks on controllers in SDN environments.

The paper [53] presents OrchSec, a network security architecture integrating network monitoring and SDN control functions. OrchSec utilizes the flexibility and control offered by SDN to develop and deploy security applications effectively. The architecture is designed to address the shortcomings in current security approaches by providing a modular and extensible framework for network monitoring, traffic analysis, and security enforcement, demonstrating its effectiveness through prototype applications and experiments.

This chapter discusses the key technologies relevant to SDN, its fundamental principles, the vulnerabilities inherent in SDN architectures, and the nature of DDoS attacks. It also delves into the functionalities of Snort IDS.

### 3.1 Introduction to SDN

Imagine a world where network management is as dynamic and adaptable as the data it handles - welcome to the revolutionary realm of SDN, where flexibility and control redefine the future of connectivity. SDN represents a paradigm shift in the evolution of networking technology. Emerging in the early 21st century, SDN redefined network management and operations, offering a level of flexibility and control previously unattainable with traditional networking methodologies.

In the early days of computing, networks were simple and limited to single buildings or rooms. They used basic equipment like hubs and repeaters, which had limited control capabilities. As technology and organizations grew, more complex network designs emerged, introducing routers and switches. These devices could intelligently forward data packets based on Internet Protocol (IP) and Media Access Control (MAC) addresses, marking a significant advancement in networking [28].

The rapid growth of the Internet in the late 20th century brought a flood of data traffic, posing new challenges for network management. Traditional networking systems, where control and data functions were intertwined in each device as shown in Figure 3.1, struggled to adapt quickly to changing demands, revealing their limitations. SDN emerged as a groundbreaking solution to these challenges. It separated the control plane (responsible for traffic management decisions) from the data plane (handling actual traffic). This architecture allowed a central controller to dynamically adjust network behavior, providing unprecedented adaptability and efficiency [4].

The roots of SDN date back to the 1980s and 1990s when the concept of central network control and programmable networks was explored. However, SDN didn't find its true purpose until the rise of cloud computing and data-center virtualization. These developments created a perfect use case for SDN and propelled its popularity.

Key milestones in the formalization of SDN included the development of the OpenFlow protocol and the establishment of the Open Networking Foundation (ONF) in 2008. OpenFlow, originating from Stanford University's clean slate project, played a crucial role in standardizing SDN protocols and enabling network experimentation.

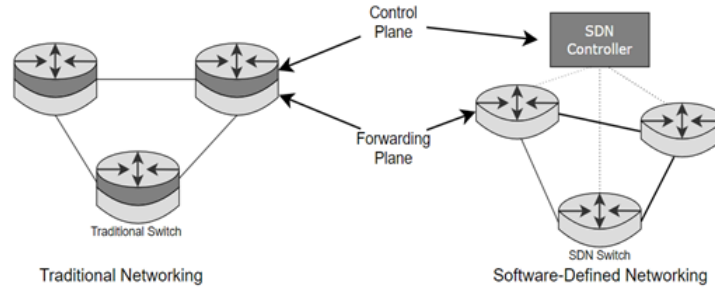


Figure 3.1: Traditional Networking vs Software-Defined Networking

SDN's adoption was further accelerated by its ability to break free from proprietary vendor technologies and protocols. It leveraged commodity IT equipment for core functions, such as firewalls and middle-boxes, making it attractive to network operators due to its flexibility and cost-effectiveness. Additionally, SDN's openness encouraged networking research, enabling data collection and experimentation through open interfaces.

## 3.2 Fundamentals of SDN

SDN is a network architecture concept that aims to make networks more agile and flexible. ONF is a non-profit consortium dedicated to the development, standardization, and commercialization of SDN. As defined in ONF, SDN is the physical separation of the network control plane from the data plane, and the control plane controls several forwarding devices [1].

SDN is a transformative approach to networking that is structured around three key layers: the application plane, the control plane, and the data plane. This layered model optimally reinvents and automates network infrastructure, allowing for greater flexibility and efficiency. Figure 3.2 demonstrates the core SDN architecture.

1. **Data Plane:** The data plane, also known as the infrastructure layer, comprises network devices like switches, routers, and access points. These devices are responsible for transporting user information through the network. In SDN, the traditional fixed functionality of these devices is replaced by a set of instructions from the control plane [14]. This means the same hardware can function as a router, a firewall, or any other network device, depending on the network manager's configuration.
2. **Control Plane:** Central to SDN, the control plane is responsible for the centralized control of the entire information flow within the data plane. It contains policies for data forwarding or diversion, flow tables, and provides a comprehensive view of the network. This is hosted in an SDN controller. The control plane communicates with the data plane through Southbound Application Programming Interfaces (APIs) (such as OpenFlow), allowing the controller to send policies and configurations to data plane devices [31]. The separation of

the control and data planes is crucial for enhancing network programmability and management.

3. **Application Plane:** The application plane is where the development of various applications occurs. These applications facilitate communication and interaction with the entire network architecture, supported by Northbound APIs. This layer gains an abstract perspective of the network, ranging from the distribution of connected devices to the collection of network behavior statistics. The development of high-level applications, often created and implemented by open-source communities, contributes to standardization and adds features like secure encryption and portability.

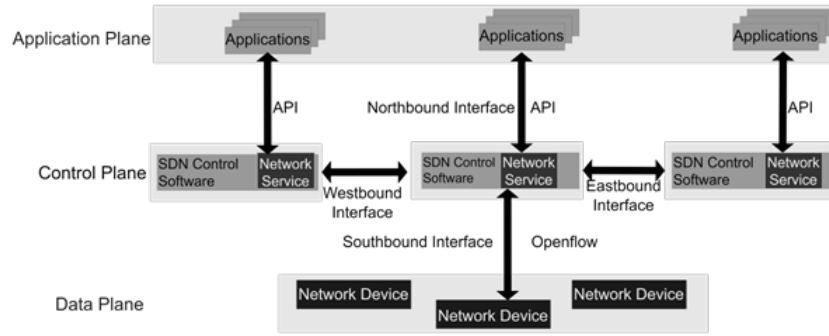


Figure 3.2: SDN Architecture

### SDN Interfaces/APIs:

1. **Southbound Interface (SBI) :** The SBI is an open, vendor-agnostic interface between the control and data planes. It generalizes device functionality to the controller, allowing the controller to communicate flow rules, retrieve flow statistics, and implement application-defined policies [2]. OpenFlow is the most widely accepted industry standard for the SBI, addressing challenges like device heterogeneity and protocol support.
2. **Northbound Interface (NBI) :** The NBI is used by applications in the management plane to interact with the control plane [19]. It offers necessary abstractions for programming language and controller platform independence, similar to operating system (OS) standards. The development of a common NBI is ongoing, with its importance lying in application portability, interoperability, and overall network management.
3. **East/Westbound Interfaces (WBI):** In large-scale networks, multiple SDN controllers may be used, each managing a set of data plane devices. East and WBI facilitate information exchange between these controllers to maintain a global network view. The WBI is used between two SDN controllers, while the Eastbound interface communicates between the SDN control plane and legacy distributed control planes [52].

### 3.3 SDN Devices and Protocol

#### 3.3.1 SDN Controller

The SDN controller, also known as the Network Operating System (NOS), is the cornerstone of any SDN infrastructure. It is pivotal in providing a global view of the network, encompassing data plane SDN devices and connecting these resources with management applications. The controller's role extends to executing flow actions as dictated by application policy among the network devices.

**Topology and Traffic Flow Management:**

The controller's core functionalities revolve around managing network topology and traffic flow. This is achieved through a link discovery module that transmits inquiries and receives packet\_in messages, enabling the controller to construct and maintain an accurate network topology. The topology manager further assists in decision-making processes to find optimal network paths, adhering to quality-of-service (QoS) and security policies [56].

**Statistics Collection and Queue Management:**

Controllers often feature dedicated modules for collecting performance data and managing different incoming and outgoing packet queues. These modules play a crucial role in network performance optimization and efficiency.

**Flow Management:**

A significant aspect of the controller's role is interacting directly with the data plane's flow entries and tables. This interaction primarily occurs through the SBI, enabling precise control and management of network flows [30].

#### 3.3.2 SDN Switch

architecture of SDN, figure 3.2 In the world of SDN, the SDN switch plays a pivotal role in the actual data movement across the network, complementing the strategic control exerted by the SDN controller. The efficacy of an SDN environment is greatly influenced by the capabilities of these switches, particularly their speed in forwarding data and their ability to interact effectively with the SDN controller. This interaction and performance are especially critical in networks supporting delay-sensitive and packet loss-sensitive applications, such as in industrial automation systems, interactive video platforms, and online surgical operations.

SDN switches (shown in the architecture of SDN, Figure 3.2) are broadly categorized into two distinct types, each with unique characteristics and operational paradigms: software switches and hardware switches.

**Software Switches** primarily operate by maintaining their flow tables in Synchronous Dynamic Random Access Memory (SDRAM) [42]. When a packet arrives at a software switch, it is matched against the flow table entries (FTE) processed by the Central Processing Unit (CPU). In cases where no matching FTE is found, the packet is forwarded to the SDN controller. The controller then provides feedback



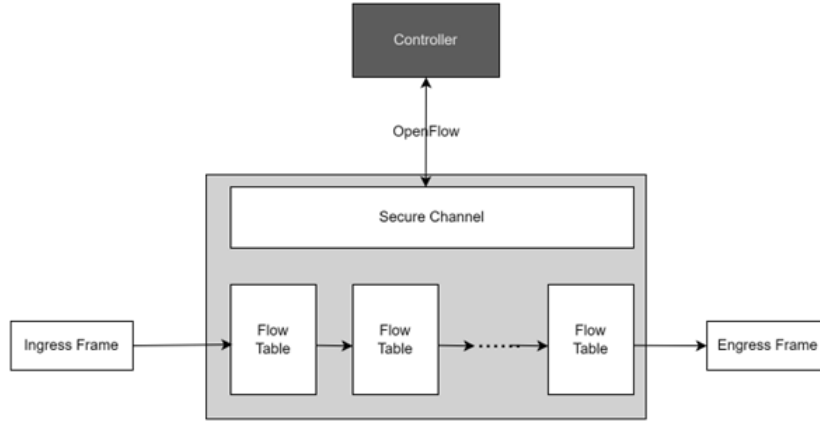


Figure 3.3: OpenFlow Switch

and forwards information to the switch, subsequently updating the software flow table. The packet processing logic in such switches is implemented in software, often optimized through sophisticated software libraries. Examples of software switches include Open vSwitch (OVS), Pantou/OpenWRT, ofsoftswitch13, and Indigo, which typically operate on commodity hardware platforms such as desktops equipped with multiple network interface cards (NICs). Figure 3.3 demonstrates the architecture of OpenFlow Switch.

**Hardware Switches**, in contrast, are characterized by their embedded packet processing functions in specialized hardware. This hardware comprises layer two forwarding tables using Content-Addressable Memories (CAMs), layer three forwarding tables using Ternary Content-Addressable Memories (TCAMs), and Application-Specific Integrated Circuits (ASICs) [12]. The FTEs in hardware switches are stored in CAMs and TCAMs, and the packets are processed by ASICs. Notably, these switches are also equipped with SDRAM and CPU, facilitating the maintenance of flow tables in both TCAM and SDRAM. Hardware switches, such as the Mellanox SN2000 series, NoviFlow NoviSwitch class, HP ProCurve J9451A, and Juniper Juno MX-Series, are capable of line-speed packet matching owing to the efficiency of CAM and TCAM. However, limitations in TCAM capacity often arise due to cost, size, and energy consumption considerations. In contrast, SDRAM, while cheaper and consuming less power, offers more flexibility for implementing complex actions through software [35].

The choice between software and hardware switches in an SDN has profound implications on the overall network performance. Analytical modeling, particularly using queueing theory, provides a framework for characterizing the performance of these switches. Such modeling is instrumental in guiding network engineers through benchmarking switch performance and conducting sensitivity analyses. These models help identify critical performance-influencing factors and support strategic decisions regarding the deployment of hardware versus software switches in specific operational contexts.

### 3.3.3 OpenFlow Protocol

OpenFlow, a critical component within the SDN paradigm, stands as the most widely accepted and deployed open southbound standard for SDN. Its role as the foundational protocol for SDN is underscored by its ability to define general packet forwarding processes, establish forwarding policies, track the forwarding process, and dynamically control it.

#### Fundamental Concepts of OpenFlow

The key abstraction in OpenFlow involves packet processing, flow management, and matching criteria [29]. A packet, comprising a header, payload, and optionally a trailer, is treated as the basic unit for forwarding. The packet header embeds all control information, which is utilized by forwarding devices to identify the packet and decide on processing actions.

Flows represent series of packets following a similar pattern. The Flow Table in OpenFlow contains a list of Flow Entries, each defining a specific pattern of packets and how they should be processed. Flow Entries are prioritized and equipped with counters for tracking packets. Matching is a crucial process in OpenFlow, involving the comparison of incoming packets against the predefined patterns in Flow Entries.

#### Actions and Forwarding in OpenFlow

Actions in OpenFlow can include forwarding packets to a port, modifying packets, or changing their state. Actions are organized into Lists or Sets, where the former can include duplicated actions for cumulative effects, and the latter ensures each action occurs only once. Instructions in Flow Entries describe the specific OpenFlow processing for matched packets, including actions like forwarding or modifying the packet's state [20].

The Forwarding Pipeline in OpenFlow is a series of linked flow tables used in the packet forwarding process. Pipeline processing begins at the first ingress table and proceeds through subsequent tables based on the outcome of matches. The pipeline fields represent values attached to the packet during this process.

#### OpenFlow Communication Mechanisms

The OpenFlow Connection is the primary channel for message exchange between a switch and a controller. It can be implemented using various network transport protocols and identified uniquely through a Connection Uniform Resource Identifier (URI). OpenFlow protocol defines three types of messages: controller-to-switch, asynchronous, and symmetric, each serving different purposes like querying switch status, expressing control commands, or exchanging lightweight information [3].

OpenFlow Controllers manage multiple switches via these connections, and each switch can establish multiple connections with different controllers. These connections ensure robust and reliable management of the switches.

#### OpenFlow Tables and the Forwarding Pipeline

OpenFlow tables are crucial in defining how packets are processed. Each flow entry within these tables comprises match fields, a priority, and instructions. The match

fields include elements like the ingress port, packet headers, and metadata from previous steps. The flow entry is uniquely identified by its match fields and priority, with a special table-miss entry handling packets that do not match any other entries.

The forwarding process involves traversing all flow tables in a pipeline manner, with each step including matching against a flow table and applying associated instructions. The egress tables are configured for specific forwarding actions, and the entire process is guided by the instructions associated with matched entries.

### **Fail-Secure and Fail-Standalone Modes**

OpenFlow also outlines operational modes for switches in case of lost connections to controllers. In the fail-secure mode, the switch continues to operate normally but drops unmatched packets. In contrast, the fail-standalone mode enables the switch to function as a legacy Ethernet switch or router, depending on the configuration.

### **3.3.4 Packet flow in SDN**

This section explains the packet flows in the SDN environment.

#### **Packet Arrival at SDN Switch:**

A data packet enters an SDN switch. This is the point of entry for traffic in an SDN environment. The switch represents the data plane in SDN, responsible for forwarding packets based on instructions from the control plane.

#### **Flow Rule Check:**

The switch checks its Flow Table for a matching rule for the packet's header (source, destination, type of service, etc.). If a Flow Rule Exists: The switch already knows how to handle this packet type based on previous instructions from the controller. It processes the packet accordingly (e.g., forwarding, dropping) and updates local counters for network management (like traffic volume, number of packets, etc.). If No Flow Rule Exists: This indicates a new type of traffic for which the switch has no predefined action.

#### **Interaction with the Controller:**

The switch sends a Packet-In message to the SDN controller, which includes the packet's metadata. The controller, embodying the centralized intelligence of the network, analyzes this information. It might undertake further network-wide analysis or communication, such as sending ARP requests to discover routes or gather network status.

#### **Controller Decides on Actions:**

Based on its comprehensive view of the network, the controller determines the best course of action for the packet. This could involve choosing the most efficient route, applying security rules, or implementing QoS policies. The controller then sends back instructions to the switch, often in the form of new flow rules.

#### **Updating the Switch's Flow Table:**

The switch receives the new flow rules and updates its Flow Table. These rules are

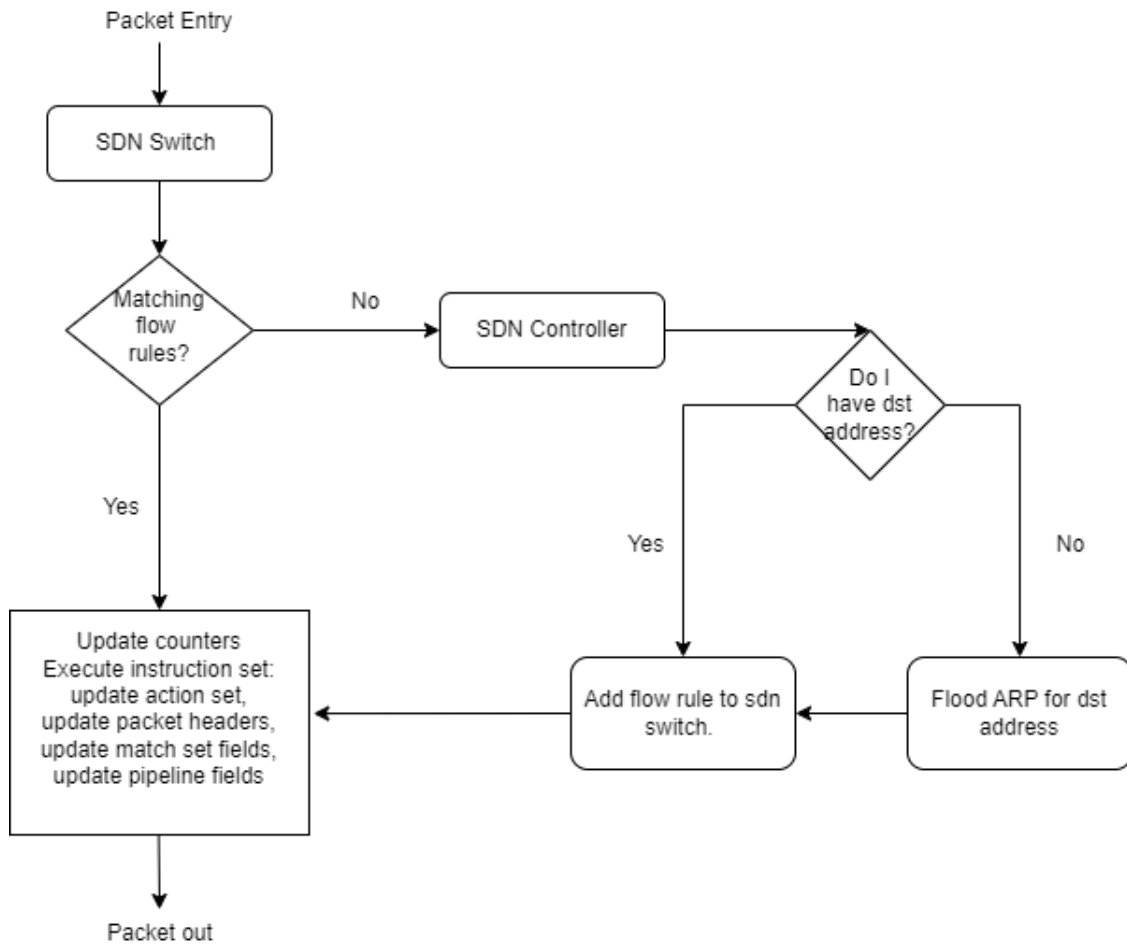


Figure 3.4: Flowchart for packet flow

essentially programming the switch on how to handle this and similar packets in the future. The action might include forwarding the packet to a specific port, modifying the packet, or even dropping it.

### Processing Subsequent Packets:

Once the new rule is in place, similar packets are processed quickly without involving the controller, as the switch now has the necessary instructions. This reflects the SDN's ability to adapt and learn from network traffic, optimizing packet handling over time.

The controller can make real-time decisions based on network conditions, traffic patterns, and policies. It's not just about forwarding packets but managing the network intelligently. Network behavior can be changed through software without altering physical devices, allowing for rapid deployment of new services or changes.

In summary, the flowchart represents a simplified yet comprehensive view of packet processing in an SDN environment, illustrating the principles of centralized control, programmability, and intelligence that are central to SDN's transformative impact on networking.

## 3.4 Vulnerabilities in SDN

SDN introduces unique security risks due to its distinct architecture, specifically the separation and centralization of its control and data planes. This section explores these vulnerabilities, highlighting the potential security threats inherent in SDN frameworks.

**Weak Authentication and Encryption:** SDN controllers, being central to network operations, are susceptible to weak authentication mechanisms and incomplete encryption processes. Weak authentication refers to scenarios where the security measures are insufficient relative to the value of the protected assets. Incomplete encryption, meanwhile, can lead to unauthorized access, as hackers may bypass authentication systems to access sensitive information.

**Information Disclosure:** SDN's architecture can unintentionally disclose sensitive information, leading to unauthorized access and potential control over network traffic. This risk is amplified by the centralized nature of the control plane, which, if compromised, can have widespread implications across the network.

**Controller Vulnerabilities:** The controller's pivotal role in SDN makes it a prime target for attacks. Any compromise in the controller's integrity can lead to manipulated data flows, severely impacting network functionality. Comparisons among different SDN controllers, like Floodlight and OpenDaylight, show varying degrees of vulnerability, emphasizing the need for robust security measures in controller design and implementation.

### Threats Across the layers of SDN:

**Control Plane:** Attacks on the control plane can include network manipulation, where attackers compromise the controller to create false network data, leading to further network attacks.

**Data Plane:** In the data plane, threats like traffic diversion and side channel attacks can redirect or eavesdrop on network traffic, exploiting vulnerabilities in network elements.

**Application Plane:** The application plane is also vulnerable, with threats like app manipulation allowing attackers to access SDN applications and perform unauthorized actions.

### Broad Categories of Threats:

Attacks on communications within the control plane and between controllers and networking devices. There are insufficient security mechanisms to ensure trust between the controller and applications. Potential for faked traffic flows and exploitation of weaknesses in switches. The unique structure of SDN, while offering several advantages in network management, also introduces specific vulnerabilities that require careful consideration and robust security solutions to mitigate. Understanding and addressing these vulnerabilities is crucial for the secure deployment and operation of SDN architectures.

## 3.5 DDOS attacks

Denial of Service (DoS) and DDoS attacks are major concerns for organizations and individuals as they can cause significant disruption to the availability of network

resources. These attacks are designed to overload a network resource, making it unavailable to its intended users [7]. In this thesis, we are going to simulate one of these attacks. It is essential for organizations to be aware of the different types of DoS and DDoS attacks and to have the necessary defenses in place to protect their networks.

A DoS attack is a type of cyber-attack that aims to make a network resource or service unavailable to its intended users. This is typically achieved by overwhelming the target with a flood of traffic, making it difficult or impossible for the resource to handle legitimate requests [26].

A DDoS attack is a variation of a DoS attack in which the traffic is generated from multiple sources, making it more difficult to mitigate. DDoS attacks can be launched from a botnet, which is a group of compromised devices controlled by the attacker. These attacks can cause significant disruption to the availability of network resources and services, resulting in financial losses and reputational damage for the targeted organizations [40].

There are several types of DoS and DDoS attacks, including:

**Volume-based attacks:** These attacks aim to saturate the network bandwidth of the target, such as a flood of traffic from multiple sources.

**Protocol-based attacks:** These attacks exploit weaknesses in the protocols used by a network resource, such as a SYN flood attack that targets the Transfer Control Protocol (TCP) protocol.

**Application-layer attacks:** These attacks target specific vulnerabilities in web applications, such as an HTTP flood attack [43].

In SDN, the attack plane is part of the architecture that is responsible for enforcing the network's security policies and protecting the network from malicious traffic. This can include firewalls, intrusion detection and prevention systems, and traffic filtering mechanisms. In SDN, the attack plane is integrated with the control plane to enable real-time monitoring and response to security threats [18]. The SDN architecture allows for a more flexible and efficient approach to network security, enabling organizations to more effectively detect and respond to DoS and DDoS attacks more.

### 3.5.1 TCP SYN Flood attack

TCP SYN Flood attack is a type of DoS attack that leverages the TCP three-way handshake during the establishment of a TCP connection, with the aim of overwhelming the targeted server's resources and causing it to become unresponsive. Generally, for a TCP connection, the client sends a SYN packet to the open port on a server. Then the server acknowledges the connection by sending a SYN-ACK packet in return to the client and the server stores the information of this connection in a Transmission Control Block (TCB). This process i.e client sending a SYN and receiving a SYN-ACK from the server and the server storing this information in TCB is called Half-Open Connection (HOC). And after this, the client then responds to the server with an ACK packet to establish the connection. If the ACK is not received, after a certain amount of time the HOC will be closed. This process of connecting the client to the server through SYN, SYN-ACK, and ACK is called a three-way handshake [34].

The SYN Flood attack can be done by sending a huge number of SYN packets to the server. The IP addresses of these SYN packets are usually spoofed. As the server does not know about this spoofing, it sends SYN-ACK packets to all the SYN packets it received thereby establishing a huge number of HOCs which in turn creates a huge amount of TCBs. Due to the huge amount of SYN packets, the server's resources are quickly exhausted because of the occupation by useless TCBs. Because of resource exhaustion, a legitimate client cannot connect a TCP connection with the server [9].

### 3.5.2 Internet Control Message Protocol (ICMP) Flood

ICMP [33] is a protocol within the TCP/IP model that employs client/server terminology. All IP-enabled end systems and intermediary devices, such as routers, frequently utilize ICMP servers for network troubleshooting purposes. This protocol is employed to identify and report issues within the network or intermediary devices, including routers, hubs, and switches.

ICMP operates as one of the network layer protocols in the TCP/IP model. Despite belonging to the network layer, its communications are not directly transmitted to the data link layer. Instead, messages are encoded into IP datagrams before being sent to the lowest layer. The protocol field carries a value of one, indicating that the IP data represents an ICMP message category.

Ping flood attacks, also known as ICMP flood attacks, are a common type of cyber threat that uses the ICMP. With its ICMP\_ECHO\_REQUEST packets, colloquially known as "ping" packets, ICMP is essential for network diagnostics and communication, testing the availability of remote hosts. In the context of cybersecurity, attackers may modify these packets to launch DDoS attacks, which attempt to disrupt the normal operation of a target's network.

In a typical ICMP flood attack scenario [25], the attacker deliberately distributes ICMP\_ECHO\_REQUEST packets across the victim's network using broadcast IP addresses. This method ensures that packets are delivered to all machines within the specified network. In response, the machines send ICMP\_ECHO\_REPLY packets back to the victim in response to the forged source address. When orchestrated on a large scale, this surge of traffic can quickly exceed the victim's bandwidth, resulting in service degradation or, in extreme cases, a complete denial of service.

### 3.5.3 UDP Flood attack

The UDP flood attack [49] is a type of DDoS attack in which the attacker sends out a large stream of UDP (User Datagram Protocol) packets from their arsenal of attack resources. In this attack, the attacker can choose to flood a specific or random port on the victim's system with UDP packets. When a system receives a UDP packet, it usually tries to figure out what kind of application is waiting on the destination port. When the system determines that no application is expected, it sends an ICMP (Internet Control Message Protocol) packet in response.

The attacker uses deceptive techniques such as spoofed IP addresses to send these packets repeatedly until the entire bandwidth is consumed, forcing the victim to suspend normal operations. This onslaught of UDP packets drains the victim's resources and impairs its ability to function normally.

### 3.5.4 IP spoofing attacks

IP source address spoofing is a malicious technique where attackers manipulate packets by using false IP source addresses to conceal their identities and launch various attacks, such as DDoS and reflected amplification DDoS [55]. This threat arises due to the lack of validation for the packet's IP source address in Internet packet forwarding, allowing attackers to exploit vulnerabilities. There are three types of IP spoofing scenarios based on the attacker's location:

1. **Host-Based Attack:**

Attackers forge packets with specified or random IP source addresses to shift responsibility to innocent parties. This type includes common attacks like DDoS and reflected amplification DDoS.

2. **Router-Based Attack:**

Attackers exploit vulnerabilities in routers or key routing devices to gain control privileges or modify forwarding functions. This attack is more challenging due to enhanced security measures on these devices by network administrators.

3. **Flow-Based Attack (Man-in-the-Middle Attack):**

Attackers position themselves halfway through the packet flow, taking control of key routing devices like wireless access points. They capture, alter, and replay packets with forged IP source addresses to achieve their malicious goals.

## 3.6 Snort IDS

Snort IDS, an open-source IDS widely used for network security. We discuss the importance of IDS and the features that make Snort a popular choice among security professionals. In today's rapidly evolving cybersecurity landscape, intrusion detection plays a crucial role in safeguarding network infrastructures. IDSs are designed to identify and respond to unauthorized activities and potential threats. Snort IDS is a widely recognized and utilized open-source IDS solution due to its flexibility, extensibility, and powerful detection capabilities [37].

### 3.6.1 Types of IDS

IDSs can be categorized into three main types: network-based IDS (NIDS), host-based IDS (HIDS), and hybrid IDS (HIDS/NIDS). Snort IDS falls under the NIDS category, where it monitors network traffic in real-time to detect and alert administrators about suspicious activities.

NIDS is a type of IDS that monitors network traffic to detect and respond to potential threats and unauthorized activities. NIDS focuses on analyzing network packets and identifying suspicious patterns or anomalies that may indicate an ongoing attack or security breach. Snort IDS, a widely used open-source IDS solution, falls under the NIDS category. Snort IDS operates by capturing packets from the network interface and analyzing their contents in real-time. It employs various techniques to inspect network traffic and identify potential threats. Some of the key features and methodologies employed by Snort NIDS are:



- **Packet Sniffing:** Snort captures packets from the network interface and pre-processes them to ensure accurate analysis. This preprocessing may include tasks like defragmentation, TCP stream reassembly, and IP defragmentation.
- **Rule-Based Detection:** Snort utilizes a rule-based detection engine to match network traffic against a set of predefined rules. These rules, written in the Snort rules language, specify patterns, signatures, or anomalies associated with known attacks or suspicious activities. Administrators can customize and configure these rules based on their specific security requirements.
- **Traffic Analysis:** Snort analyzes the contents of captured packets to extract relevant information and identify potential threats. It examines the header and payload of packets, decodes various network protocols, and applies protocol-specific rules for deeper inspection. By inspecting network traffic at this level, Snort can identify malicious activities or deviations from expected network behavior.
- **Alerting and Logging:** When Snort detects suspicious activity based on the defined rules, it generates alerts and logs the relevant information. These alerts can be customized to specify severity levels and appropriate actions to be taken. By providing real-time alerts and detailed logging, Snort enables administrators to respond promptly to potential threats and investigate security incidents.

The network-based approach of Snort IDS allows it to monitor network traffic at a central point, making it particularly suitable for large-scale networks. By analyzing network packets and employing rule-based detection, Snort can identify a wide range of attacks, including known signatures and anomalous behavior. Its flexibility and extensibility, combined with the active user community, have contributed to the popularity of Snort as a powerful NIDS solution [5].

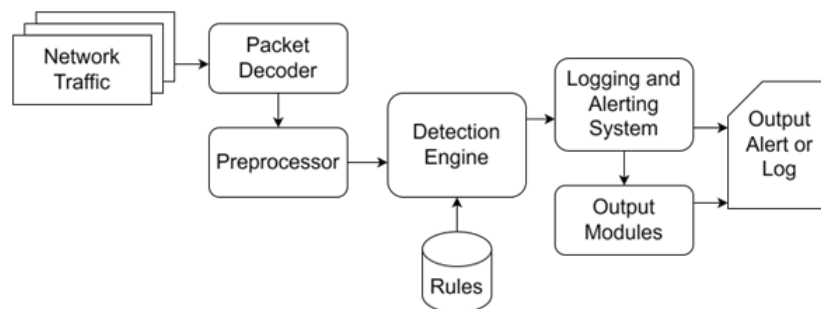


Figure 3.5: Snort Architecture

In Figure 3.5, the Snort Architecture is depicted as a structured workflow designed for network traffic analysis and intrusion detection. The process begins with the 'Network Traffic' entering the system, which is then passed through a 'Packet Decoder'. The decoder's role is to interpret the raw data packets and convert them into a format that can be further processed.

From there, the preprocessed traffic flows into the 'Preprocessor' stage. This module is responsible for performing a variety of functions such as reassembly of

fragmented packet streams, decoding application layer protocols, and performing protocol normalization to prepare the traffic for the next stage.

The core of the architecture is the 'Detection Engine', which receives the preprocessed packet data. Utilizing the set of defined 'Rules', which act as the criteria for detecting suspicious activities, the engine scrutinizes the packet data against these rules to identify any potential threats or intrusions.

In the event of a detection, the information is sent to the 'Logging and Alerting System'. This subsystem is crucial for the notification and recording of incidents. Depending on the configuration, the system can generate alerts in real-time or log the events for later review.

Finally, the 'Output Modules' determine how the alerts or logs are presented or communicated to the network administrator or other security information and event management systems. The output can be customized to suit various formats and protocols as required by the organization's security infrastructure.

Through this structured process, as demonstrated in Figure 3.5, Snort is able to effectively analyze network traffic and alert administrators of any malicious activity, thereby providing a critical layer of security for network environments.

### 3.6.2 Advanced Features and Performance Analysis of Snort IDS

We explore the additional capabilities that enhance Snort's detection capabilities and discuss performance considerations when deploying Snort in large-scale network environments. Furthermore, we analyze the effectiveness and limitations of Snort IDS based on empirical performance evaluations.

#### Advanced Features of Snort IDS

- **Protocol Awareness:** Snort IDS possesses protocol awareness, allowing it to analyze network traffic based on specific protocols. This feature enables deeper inspection and detection of protocol-specific attacks, enhancing the overall effectiveness of the IDS.
- **Flow Analysis:** Snort can analyze traffic flows and detect patterns of suspicious behavior by tracking connections between hosts and monitoring traffic between them. Flow analysis helps identify attacks that span multiple packets and provides a more comprehensive understanding of potential threats.
- **Payload Inspection:** Snort's payload inspection capabilities enable the detection of attacks embedded within packet payloads. It can identify specific patterns, signatures, or anomalies within the data payload, allowing for the detection of sophisticated attacks that may attempt to evade traditional signature-based detection methods.

#### Performance Considerations

- **Hardware Acceleration:** To handle high-speed network traffic, Snort IDS can be enhanced with hardware acceleration techniques. By offloading some processing tasks to specialized hardware, such as NICs with built-in packet

processing capabilities, Snort can achieve higher throughput and minimize the impact on overall system performance.

- **Distributed Deployment:** For large-scale network environments, Snort IDS can be deployed in a distributed manner. By distributing the workload across multiple sensors, it is possible to handle high traffic volumes effectively. Proper load balancing and coordination between distributed sensors are crucial to maintaining the overall performance and accuracy of the IDS.

This chapter details the implementation process and the tools used in conducting the experiments, highlighting their integral role in achieving the aims of the thesis. It provides an in-depth explanation of the experimental setup, illustrating how each component contributes to the evaluation and/or development of Snort IDS integration and DDoS attack prevention in an SDN environment.

### 4.1 Research Design

This study adopts an experimental approach to evaluate the effectiveness of Snort IDS integration and DDoS attack prevention technique. This approach is chosen for its precision in measuring the system's response under controlled conditions, replicating various DDoS attack scenarios within a simulated SDN environment.

The experimental setup involves configuring a virtual SDN environment using Mininet-VM, hosted on Oracle VM VirtualBox. This environment includes a network topology representative of real-world SDN configurations, managed by the Ryu controller. The integration of Snort IDS with Ryu is a critical component of this setup, designed to detect and mitigate DDoS attacks. The experiments involve simulating various types of DDoS attacks using hping3, allowing us to test the detection capability of Snort IDS and attack prevention capability of Ryu application within the SDN context. This setup not only tests the functionality of individual components but also assesses their interoperability and collective efficiency in a realistic network scenario.

### 4.2 Tools and Technologies

This section details the specific tools and technologies integral to the experimental setup.

#### 4.2.1 Mininet-VM

Mininet [15], is a network emulation tool which is used to evaluate SDN architecture and its designs. It emulates a computer ethernet network by using a computer instead of physical hardware of switches, routers, links, and network cards. This is useful because it allows a user to evaluate both the feasibility and performance of a design before making large-scale hardware investments. Mininet follows an emulation

strategy which relies on concurrent processes running on a single computer rather than real-time equipment. This allows one to assess how a design operates as the size of a network grows.

Mininet is instrumental for the experimental setup, creating a realistic and customizable virtual SDN environment. It allows for the simulation of various network topologies and scenarios, vital for testing the efficacy of our Snort integration and DDoS attack prevention under different network conditions.

### 4.2.2 Oracle VM VirtualBox

Oracle VM VirtualBox [17], is a free and open-source virtualization software for the x86 platform. It acts as a hypervisor or virtual machine (VM) manager, allowing the user to create a virtual environment in which many OS can operate simultaneously. The host OS is the original OS of the user's device on which VirtualBox is installed, whereas the OS or groups of OS that operate on the VM are referred to as guest OS.

VirtualBox is an excellent choice because it supports a wide range of OS as its host, including Windows, Linux, and macOS. By simply halting the system, the application allows the user to control system execution and processes.

Oracle VirtualBox serves as the virtualization platform for Mininet-VM, providing a stable and isolated environment for our SDN simulations. Its flexibility and ease of use are crucial in facilitating the creation and management of different network configurations.

### 4.2.3 PuTTY

PuTTY [44], is a free and open-source terminal emulator, serial console, and network file transfer application. It is primarily used to connect to distant computers or network devices via secure shell (SSH), Telnet, or serial connections. PuTTY is widely used in the IT and system administration community to access and manage remote servers and network equipment. Support for multiple authentication mechanisms (e.g., passwords, SSH keys), session logging, dynamic port forwarding (SSH tunneling), and the ability to define numerous terminal settings are among PuTTY's primary features.

### 4.2.4 Xming X server

Xming X server(X11) or simply X [13], is the name of software used to create graphical user interfaces (GUIs). It is most commonly used in unix systems, where it is practically universal. It uses a client-server approach and is made up of multiple distinct components such as an X server, an X protocol, an Xlib library, and so on. This X protocol, transmitted via Inter Process Communication (IPC) or TCP/IP, allows a running application to be displayed on a machine other than the one from which it was launched.

Xming provides access to an XDMCP (X display manager is a system process in a graphical system X window that allows users to log in from a local computer or a computer network) session running on a remote computer with another X server

and allows users to start, see, and manipulate with remote X applications via SSH tunneling. This is feasible with PuTTY when X11 forwarding is enabled.

PuTTY and Xming X Server are used for remote access and graphical interface support, respectively. PuTTY enables secure access to the Mininet-VM, allowing for command-line interactions with the SDN setup. Xming provides a graphical interface to visualize the network topology and monitor real-time network statistics, which is vital for analyzing the network's behavior during DDoS attack simulations.

### 4.2.5 Ryu Controller

Ryu [38], is a SDN framework built on components. Ryu delivers software components with well-defined APIs that allow developers to easily design new network management and control applications. Ryu supports a variety of network device management protocols, including OpenFlow, Netconf, OF-config, and others. Ryu fully supports OpenFlow 1.0, 1.2, 1.3, 1.4, 1.5, and Nicira Extensions.

The Ryu controller, at the heart of our SDN setup, is responsible for managing network flows and making real-time decisions. Its integration with Snort IDS is key to our strategy, enabling dynamic response to potential DDoS attacks. The Ryu controller is responsible for managing the flow control within the network. For this research, the Ryu controller has been specifically configured to integrate with our DDoS mitigation strategy. Its adaptability and compatibility with various network applications make it an ideal choice for this experiment.

### 4.2.6 hping3

hping3 is a network tool that can send customized ICMP / UDP / TCP packets and display target responses in the same way that ping does with ICMP replies. It supports fragmentation as well as arbitrary packet body and size, and it can be used to transfer files over supported protocols. hping3 can be used to test firewall rules, perform (spoofed) port scanning, test network performance using various protocols, do path MTU discovery, execute traceroute-like actions under various protocols, fingerprint remote OS, audit TCP/IP stacks, and so on.

hping3 is employed to simulate DDoS attacks within our SDN environment. This tool allows for the generation of high volumes of network traffic, mimicking various forms of DDoS attacks. By using hping3, one can effectively test the responsiveness and efficiency of the proposed DDoS attack prevention under different attack scenarios.

## 4.3 Configuring the SDN environment

Configuring the SDN environment was a multi-step process that required careful installation and setup of various software and tools on a laptop with 8GB RAM, internet connectivity, and the following specific configurations and installations:

- ◊ Oracle VM VirtualBox was installed from the official VirtualBox website.

- ◇ Mininet-VM was downloaded from Mininet's GitHub releases and imported into VirtualBox with adjustments made to the VM's system settings to allocate 5160 MB of RAM and 6 processors, maximizing video memory for display.
- ◇ Network settings were configured for NAT with SSH port forwarding from host port 2223 to guest port 22.
- ◇ PuTTY and Xming X Server were installed from PuTTY's official website and a verified source, respectively, to facilitate remote access and graphical interface display.
- ◇ hping3 was installed in Mininet-VM using the command 'sudo apt-get install hping3'.
- ◇ Ryu controller was installed via Git with the commands provided, setting up the necessary SDN controller application.
- ◇ PuTTY was configured with the hostname 'mininet@localhost' and port '2223', with X11 forwarding enabled to connect to Xming X Server.
- ◇ The environment variable 'DISPLAY' was set to the host's IP and display number to establish a connection to the X Server.

After executing these steps, the SDN environment was fully configured, operational, and ready for future configurations and experiments.

## 4.4 Installation of Snort IDS

Snort is installed from the official Snort website. The installation process on Ubuntu involves several steps, beginning with system updates and prerequisite installations:

1. **System Updates:** The process commenced with updating the OS to ensure all packages were current, using the commands:

```
sudo apt-get update && sudo apt-get dist-upgrade -y
```

2. **Setting Timezone:** Correct timezone settings are crucial for accurate timestamping in logs:

```
sudo dpkg-reconfigure tzdata
```

3. **Installing Dependencies:** Necessary libraries and tools were installed to support Snort functionality:

```
sudo apt-get install -y build-essential autotools-dev
libdumbnet-dev liblua5.1-dev libpcap-dev \
zlib1g-dev pkg-config libhwloc-dev cmake liblzma-dev
openssl libssl-dev cpputest libsqlite3-dev \
libtool uuid-dev git autoconf bison flex libcmocka-dev
libnetfilter-queue-dev libunwind-dev \libmnl-dev ethtool libjemalloc-dev
```

4. **Directory Preparation for Snort Files:** A directory was created to store the Snort source files and dependencies:

```
mkdir ~/snort_src
cd ~/snort_src
```

5. **DAQ Installation:** The Data Acquisition library required by Snort was downloaded and installed:

```
wget https://www.snort.org/downloads/snortplus/libdaq-3.0.13.tar.gz
```

6. **Downloading and Installing Snort:** The latest version of Snort was then compiled and installed:

```
https://www.snort.org/downloads/snortplus/snort3-3.1.77.0.tar.gz
```

7. **Verification of Snort Installation:** The installation was verified by checking the version of Snort installed:

```
/usr/local/bin/snort -V
```

8. **Installation of rules:** The command used to the rules installation for the attack detection is as follows:

```
wget https://www.snort.org/downloads/community/snort3-community-rules.tar.gz
```

#### 4.4.1 Integration of Snort IDS

As mentioned in [32], there are two options to integrate Snort IDS in the SDN environment, to put Ryu and Snort IDS on same machine or different machines. But the Ryu documentation [32], suggests to use Snort IDS and Ryu on different machines. So for all the experiments in this thesis, Snort IDS is integrated with the SDN switch in the SDN architecture. Following command is added to the Ryu controller configuraton.



```
socket_config = {'unixsock': False}
```

These following commands launch the Snort IDS on the SDN switch and alerts are generated to the controller.

```
sudo snort -c \usr\local\etc\snort\snort.lua -i s1-eth1 -A unsock -l \tmp  
sudo python pigrelay.py
```

The pigrelay.py script can be cloned from [22]. The script sets up a Unix Domain Socket (SOCKFILE) to listen for Snort alerts. It creates a network socket to connect to a specified controller IP address and port. It continuously listens for Snort alerts on the Unix Domain Socket. When a Snort alert is received, it sends the alert data to the specified controller via the network socket.

## 4.5 Configuring Ryu Controller Application

The Ryu controller application, an essential component of our SDN architecture, was configured to manage network traffic and enforce security policies. It operates as the brain of the SDN, interacting with network devices via OpenFlow protocols to dynamically control data flows based on pre-defined rules.

### 4.5.1 Formation of mininet network

In the Mininet-based network configuration algorithm, the setup begins with the importation of necessary modules. These modules are the building blocks for network emulation, providing tools for creating and managing virtual network components such as nodes, switches, and controllers, as well as for user interaction through the command-line interface (CLI). Once the modules are imported, the main function, myNetwork(), is defined to encapsulate the entire network setup process, ensuring a clean and organized script.

Within the myNetwork() function, a new Mininet object is instantiated without a predefined topology, allowing for a custom network design specified by the user. The network's IP base is set to 10.0.0.0/8, a common private IP address range, providing ample addresses for network expansion. Following the initial setup, the script logs the addition of a remote controller—this is a crucial step, as the controller is responsible for managing the flow of packets across the network. The remote controller is defined with specific attributes such as name, type, communication protocol, and port number, which allows it to communicate with the virtual switches within the emulated network.

The algorithm proceeds with the addition of network components, including OVS kernel switches and host nodes, each assigned unique identifiers and IP addresses. The interconnections between these components are established through bidirectional links, which are essential for the flow of data packets. As each component is added, the script logs the event, providing real-time feedback during the network's construction. After the topology is built and all components are added, the network is initiated. This involves starting the remote controller and associating it with the switches to handle network traffic based on predefined or dynamic rules.

**Algorithm 1** Algorithm for sdn network in mininet

- 1: Import required Mininet modules for network setup and control.
- 2: Define myNetwork for setup; initialize Mininet with specified IP base
- 3: Log and add a named remote controller with protocol and port
- 4: Log switch addition; add two OVSKernelSwitches
- 5: Log host addition; add three hosts with unique IPs
- 6: Log link creation; connect hosts and switches
- 7: Log network start; build with all components
- 8: Link switches to remote controller for management.
- 9: Log setup, launch CLI for network interaction.

Finally, the Mininet CLI is launched, granting the user direct interaction with the emulated network for testing or modification purposes. Once the user exits the CLI, the script gracefully terminates the network and performs cleanup operations, ensuring that no residual processes are left running. The script's entry point sets the logging level to 'info', which controls the verbosity of the output, and calls the `myNetwork()` function to kick-start the network setup, thus completing the algorithm's execution. This methodical approach allows for a controlled and repeatable process for network emulation and testing, crucial for network design and research.

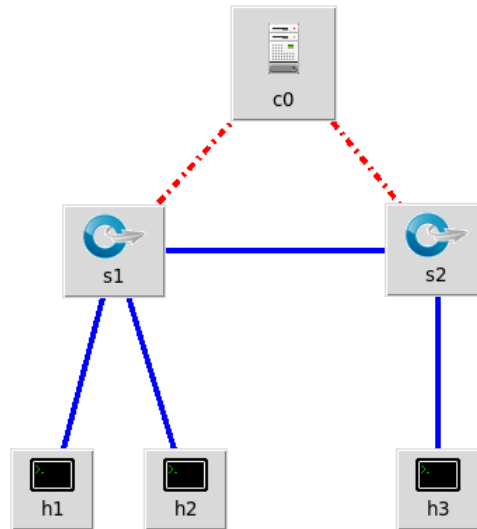
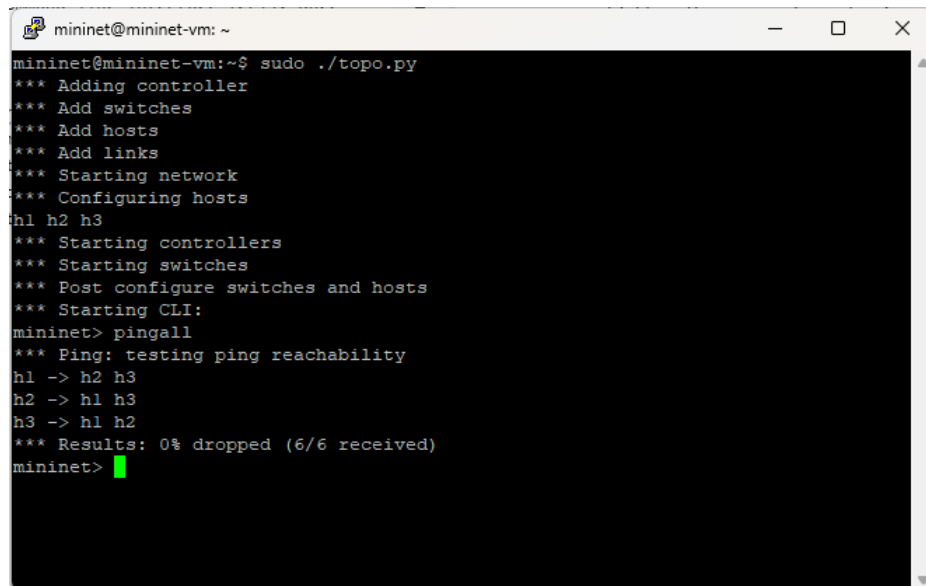


Figure 4.1: SDN Architecture using miniedit

The figure 4.1 displays the creation of basic SDN architecture that can be used for the experiments in Mininet. The network simulation script `topo.py` from 4.2 is created using the `miniedit.py` script.

The figure 4.2 depicts the successful execution of a network simulation script (`topo.py`) in a Mininet environment. The script is initiated with elevated privileges using the `sudo` command. Following this, the Mininet CLI presents a series of informational messages that outline the sequential steps taken to configure the virtual network.



```
mininet@mininet-vm: ~  
mininet@mininet-vm:~$ sudo ./topo.py  
*** Adding controller  
*** Add switches  
*** Add hosts  
*** Add links  
*** Starting network  
*** Configuring hosts  
h1 h2 h3  
*** Starting controllers  
*** Starting switches  
*** Post configure switches and hosts  
*** Starting CLI:  
mininet> pingall  
*** Ping: testing ping reachability  
h1 -> h2 h3  
h2 -> h1 h3  
h3 -> h1 h2  
*** Results: 0% dropped (6/6 received)  
mininet>
```

Figure 4.2: Mininet console after initiating the network

Once the network is started, the CLI provides feedback on the status of the network elements, including hosts, switches, and controllers. This feedback is critical for verification and debugging purposes. For instance, it confirms the successful startup of network components, as indicated by the prompt messages "Starting network", "Starting controllers", and "Starting switches".

The final stage of the simulation involves testing the connectivity between all hosts using the `pingall` command. The console output clearly shows that all hosts (h1, h2, and h3) are able to communicate with one another, as evidenced by the successful ping tests and the report of "0% dropped" packets. This indicates that the network is fully operational, and all hosts are reachable, confirming the correct setup and functionality of the network as intended by the `topo.py` script.

### 4.5.2 Ryu Controller Application

In the outlined algorithm, the process begins with the importation of essential libraries from the Ryu framework. These libraries provide the foundation for application development, event handling, protocol management, and packet processing within the SDN controller. The `SimpleSwitch13` class is then defined to encapsulate the network management logic, inheriting from the `RyuApp` base class and specifying the OpenFlow protocol version 1.3 to ensure compatibility with corresponding network switches.

Initialization of the application includes the creation of a dictionary that maps MAC addresses to ports, enabling the switch to learn the network's topology and effectively route packets. The `switch_features_handler` method configures the initial flow entries for the switch, defining default actions for unmatched packets and typically forwarding them to the controller. For this study, the `idle_time` parameter is considered, which indicates how long the flow can remain inactive without any new packets matching its characteristics. This setup is critical for handling subse-

quent network traffic that does not fit existing flow criteria. The core functionality of

---

**Algorithm 2** SimpleSwitch13 Ryu App for Network Management

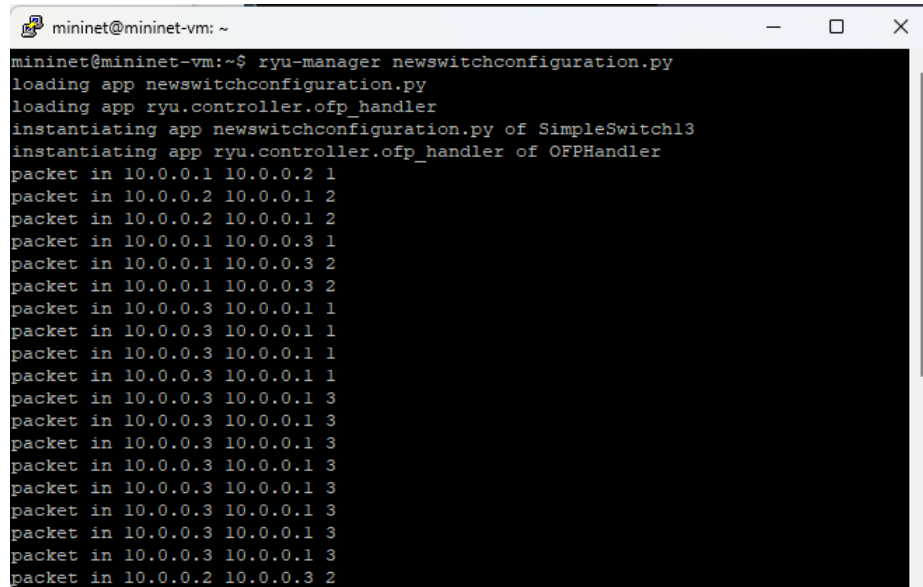
---

- 1: Import Ryu base, event, handler, protocol, and packet libraries.
  - 2: Define SimpleSwitch13 class inheriting from RyuApp.
  - 3: Specify OpenFlow protocol version 1.3.
  - 4: Initialize mac\_to\_port dictionary within the constructor.
  - 5: Define switch\_features\_handler to set up table-miss flow entries.
  - 6: Define add\_flow to add flow entries to the switch.
  - 7: Define packet\_in\_handler to handle incoming packets.
  - 8: Check for truncated packets and print warning if necessary.
  - 9: Parse packet and extract ethernet, IP, and port information.
  - 10: Learn source IP to avoid flooding for subsequent packets.
  - 11: Determine output port based on destination IP.
  - 12: Create action to output packet to determined port.
  - 13: Install flow to switch if destination is known and not a flood action.
  - 14: Check for IP protocol and create match criteria for IP packets.
  - 15: Check buffer ID and send flow modification message to switch.
  - 16: Assemble packet out message and send to switch if needed.
- 

the application is captured within the packet\_in\_handler method, which deals with packets that the switch cannot process without further instructions. This method parses the packets, extracting Ethernet and IP data, and makes decisions based on the source and destination MAC addresses. The controller employs a learning strategy to record the ports associated with source MAC addresses, thus optimizing the flow of traffic by reducing unnecessary packet flooding across the network.

The algorithm then dictates how to handle known destinations by directing traffic through predetermined ports and setting up new flow entries for unknown destinations to prevent future packet-in events for the same flow. For IP packets, match criteria are established based on IP addresses, facilitating the creation of flow rules that enhance network efficiency. When a packet does not match any flow entry and the switch has not buffered it, the controller sends a packet-out message to the appropriate port, thus ensuring continuous traffic flow while the switch updates its flow table with the new rule. This methodical approach enables the Ryu controller to manage network traffic dynamically, adapt to changes, and maintain efficient data transmission throughout the network.

After implementing the algorithm for network management using the Ryu framework, the Figure 4.3 depict the successful operation of the Ryu application. The live output from the Ryu controller indicates that the switchconfiguration.py script is actively processing packets within the emulated network. Each line of the console output presents a packet transaction, clearly showing the source and destination IP addresses, along with the output port number. This level of detail illustrates the controller's ability to make intelligent forwarding decisions in real-time, effectively routing packets between hosts in the network.

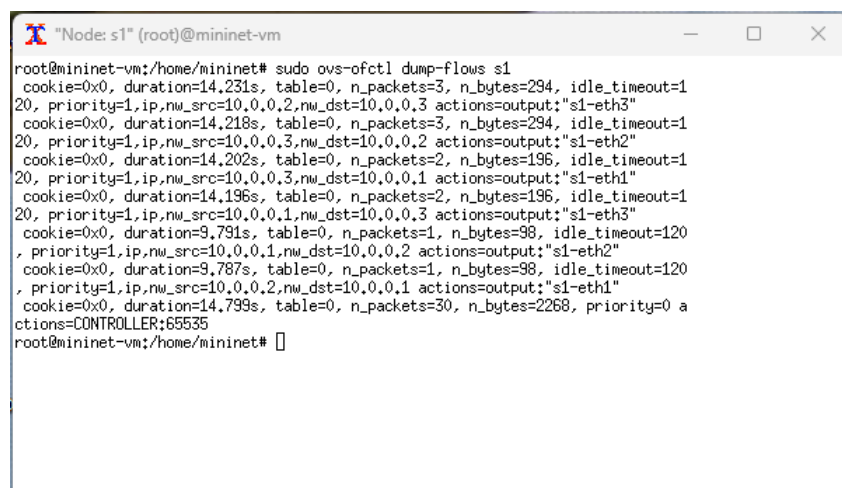


```
mininet@mininet-vm:~$ ryu-manager newswitchconfiguration.py
loading app newswitchconfiguration.py
loading app ryu.controller.ofp_handler
instantiating app newswitchconfiguration.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 10.0.0.1 10.0.0.2 1
packet in 10.0.0.2 10.0.0.1 2
packet in 10.0.0.2 10.0.0.1 2
packet in 10.0.0.1 10.0.0.3 1
packet in 10.0.0.1 10.0.0.3 2
packet in 10.0.0.1 10.0.0.3 2
packet in 10.0.0.3 10.0.0.1 1
packet in 10.0.0.3 10.0.0.1 1
packet in 10.0.0.3 10.0.0.1 1
packet in 10.0.0.3 10.0.0.1 3
packet in 10.0.0.3 10.0.0.1 3
packet in 10.0.0.3 10.0.0.1 3
packet in 10.0.0.3 10.0.0.1 3
packet in 10.0.0.3 10.0.0.1 3
packet in 10.0.0.3 10.0.0.1 3
packet in 10.0.0.3 10.0.0.1 3
packet in 10.0.0.2 10.0.0.3 2
```

Figure 4.3: Output in RYU controller

The figure 4.4 provides a snapshot of the OpenFlow switch's flow table. The `sudo ovs-ofctl dump-flows s1` command has been executed to retrieve the current flow entries on the switch 's1'. The output reveals several flow entries with specific match criteria based on source and destination IP addresses. Each entry also specifies an action, such as forwarding out on a particular switch port, which aligns with the designed behavior of the SimpleSwitch13 application to reduce packet flooding. The presence of these flow entries confirms that the Ryu application is not only receiving and processing packets but also successfully installing flow rules to optimize network traffic.

The detailed information from figure 4.3 and 4.4 provides a comprehensive overview



```
"Node: s1" (root)@mininet-vm
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=14.231s, table=0, n_packets=3, n_bytes=294, idle_timeout=1
20, priority=1, ip,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=output:"s1-eth3"
cookie=0x0, duration=14.218s, table=0, n_packets=3, n_bytes=294, idle_timeout=1
20, priority=1, ip,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=output:"s1-eth2"
cookie=0x0, duration=14.202s, table=0, n_packets=2, n_bytes=196, idle_timeout=1
20, priority=1, ip,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=output:"s1-eth1"
cookie=0x0, duration=14.196s, table=0, n_packets=2, n_bytes=196, idle_timeout=1
20, priority=1, ip,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=output:"s1-eth3"
cookie=0x0, duration=9.791s, table=0, n_packets=1, n_bytes=98, idle_timeout=120
, priority=1, ip,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:"s1-eth2"
cookie=0x0, duration=9.787s, table=0, n_packets=1, n_bytes=98, idle_timeout=120
, priority=1, ip,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:"s1-eth1"
cookie=0x0, duration=14.799s, table=0, n_packets=30, n_bytes=2268, priority=0 a
ctions=CONTROLLER:65535
root@mininet-vm:/home/mininet#
```

Figure 4.4: Flowrules in SDN Switch

of how the Ryu controller interacts with network switches to manage traffic flows. The flow rules installed by the Ryu application contribute to the efficiency of the network by ensuring that packets are routed correctly, thus avoiding the overhead of

processing similar packets multiple times at the controller. This approach not only minimizes latency and maximizes throughput but also demonstrates the controller's capacity to maintain a dynamic and responsive network environment. The successful execution and output signify that the Ryu application is functioning as intended, with the SDN controller maintaining a robust network state through proactive management of the data plane.

### 4.5.3 Configuration of Ryu to mitigate IP spoofing

Ryu controller application is configured to prevent IP spoofing, as shown in Algorithm 1. The basic concept of preventing IP spoofing is explained. As stated in Section 3.3.1, for every first packet received from a new host by the switch, the switch sends a `packet_in` message to ask the controller what action to take on the packet received and the controller sends a message to the switch to have an entry for a flow rule into flow table about which action to take on the packet received from the new host. But, according to the configuration in Algorithm 3, the first packet received from the new host is held by the controller and if a second packet is received, only then a message is sent from the controller to the switch to have an entry for a flow rule into the flow table on what action to take on the packet received from that host. In this way, as spoofed IP flooding sends single packets from random IP addresses, every single packet received from a spoofed IP is dropped.

---

**Algorithm 3** Configuring Ryu Controller to Block IP Spoofing

---

```

1: Initialize the Ryu application and set up data structures
2: Establish a connection to the switch and send a feature request
3: for each packet received do
4:   Extract source IP address
5:   if source IP address is new then
6:     Record the source IP address
7:     if a second packet from the same source IP is received then
8:       Send flow rule to switch for the source IP
9:     else
10:      Drop the packet to prevent IP spoofing
11:    end if
12:  else
13:    Apply action based on existing flow rules
14:  end if
15: end for

```

---

Figure 4.5 extends the process outlined in Section 3.3.4, with additional security measures integrated into the SDN environment to prevent IP spoofing attacks. When a packet arrives at the SDN switch, the integrated Snort IDS evaluates it for potential security threats, including IP spoofing. The Snort system scrutinizes the packet's source IP address to confirm its authenticity. If an IP spoofing attempt is detected, the system triggers a security protocol. Upon detection of IP spoofing, the system generates alerts for the controller. These alerts lead to immediate actions to mitigate the attack. In this configuration, the controller acts according to algorithm

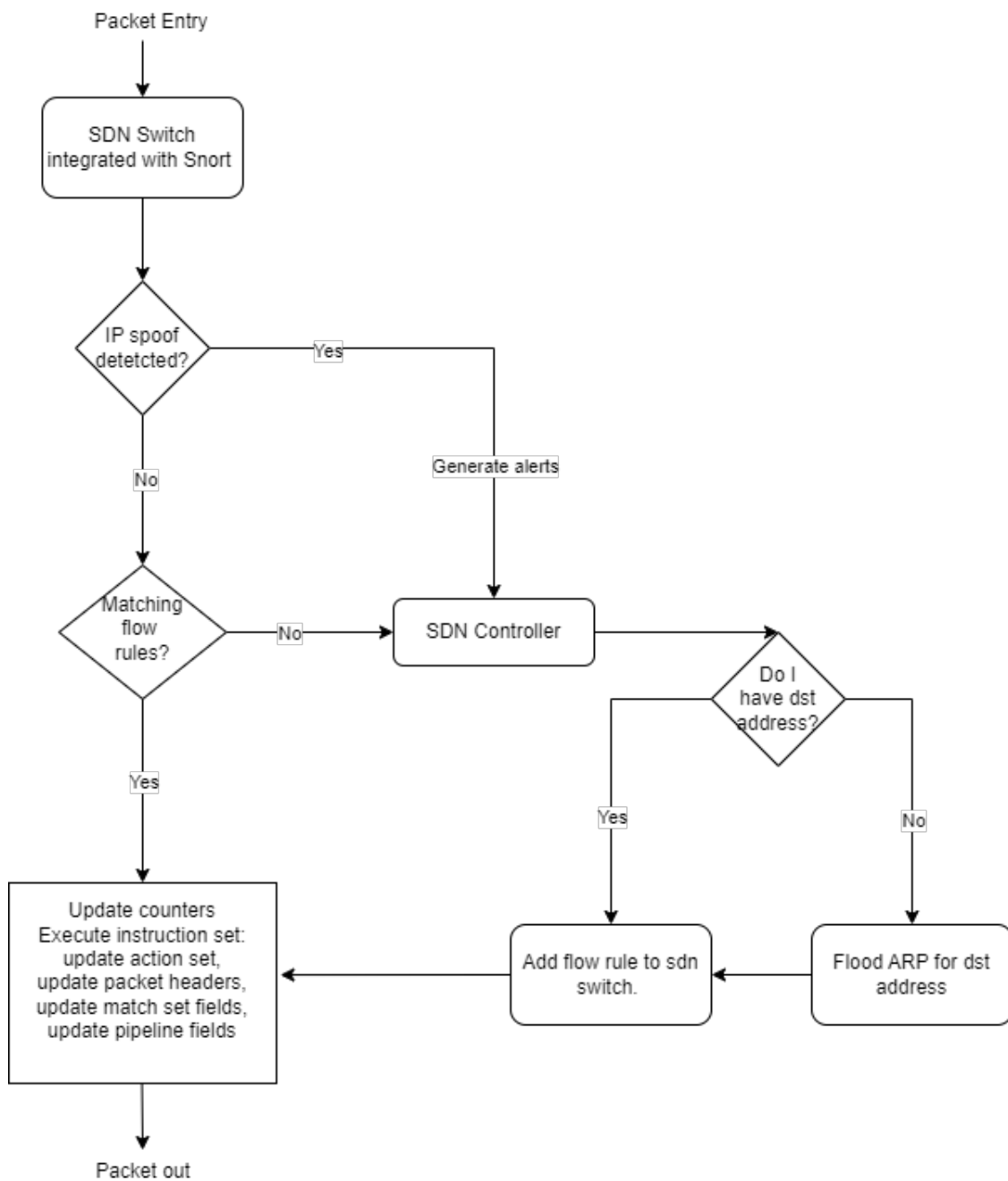


Figure 4.5: Flowchart for IP spoof mitigation configuration

3. If no IP spoofing is detected, the switch proceeds to check its Flow Table for a matching flow rule to determine how to handle the packet. If the switch does not have a predefined action, it refers to the SDN controller. The controller instructs the switch to add a new flow rule to handle the packet. The switch updates its Flow Table with the new instructions from the controller.

#### 4.5.4 Configuration of Ryu to mitigate Flooding attacks

In the proposed Algorithm 4 for Ryu Controller Configuration for Flood Mitigation in an SDN environment, the approach extends the SimpleSwitch13 application to include a monitoring component. This extension involves initializing a dedicated thread for flood detection. The algorithm handles state changes in datapaths by registering or unregistering them based on their state (MAIN\_DISPATCHER or DEAD\_DISPATCHER). Additionally, it periodically requests flow and port statistics from each datapath. In case of flood detection, the algorithm redirects packets matching certain criteria to the controller and modifies the flow entry to prevent forwarding, thereby mitigating the flood risk. This proactive approach to flood mitigation enhances network security by dynamically managing and adjusting to network conditions.

---

**Algorithm 4** Ryu Controller Configuration for Flooding

---

```

1: Extend base SimpleSwitch13 application for monitoring
2: Initialize monitoring thread for flood detection
3: Upon State Change Event:
4: if datapath enters MAIN_DISPATCHER then
5:   Register datapath
6: else if datapath enters DEAD_DISPATCHER then
7:   Unregister datapath
8: end if
9: while true do
10:   Sleep for a predefined monitoring interval
11:   for each datapath do
12:     Send request for flow and port statistics
13:   end for
14:   Sleep for a short interval before next iteration
15: end while
16: Upon Flow Statistics Reply Event:
17: for each flow with priority 1 do
18:   Redirect matching packets to the controller
19:   Modify flow entry to prevent forwarding
20:   Log modification action
21: end for

```

---

Figure 4.6 builds upon the previous section 3.3.4 by incorporating Snort IDS within the SDN switch to prevent flooding attacks. When a packet enters an SDN switch with an integrated Snort IDS, the switch is prepared to handle both standard packet processing and security monitoring for abnormal traffic patterns indicative of a flooding attack. If a flooding attempt is detected, the system triggers a security protocol. Upon detection of flooding, the system generates alerts for the controller. These alerts lead to immediate actions to mitigate the attack. Which includes the



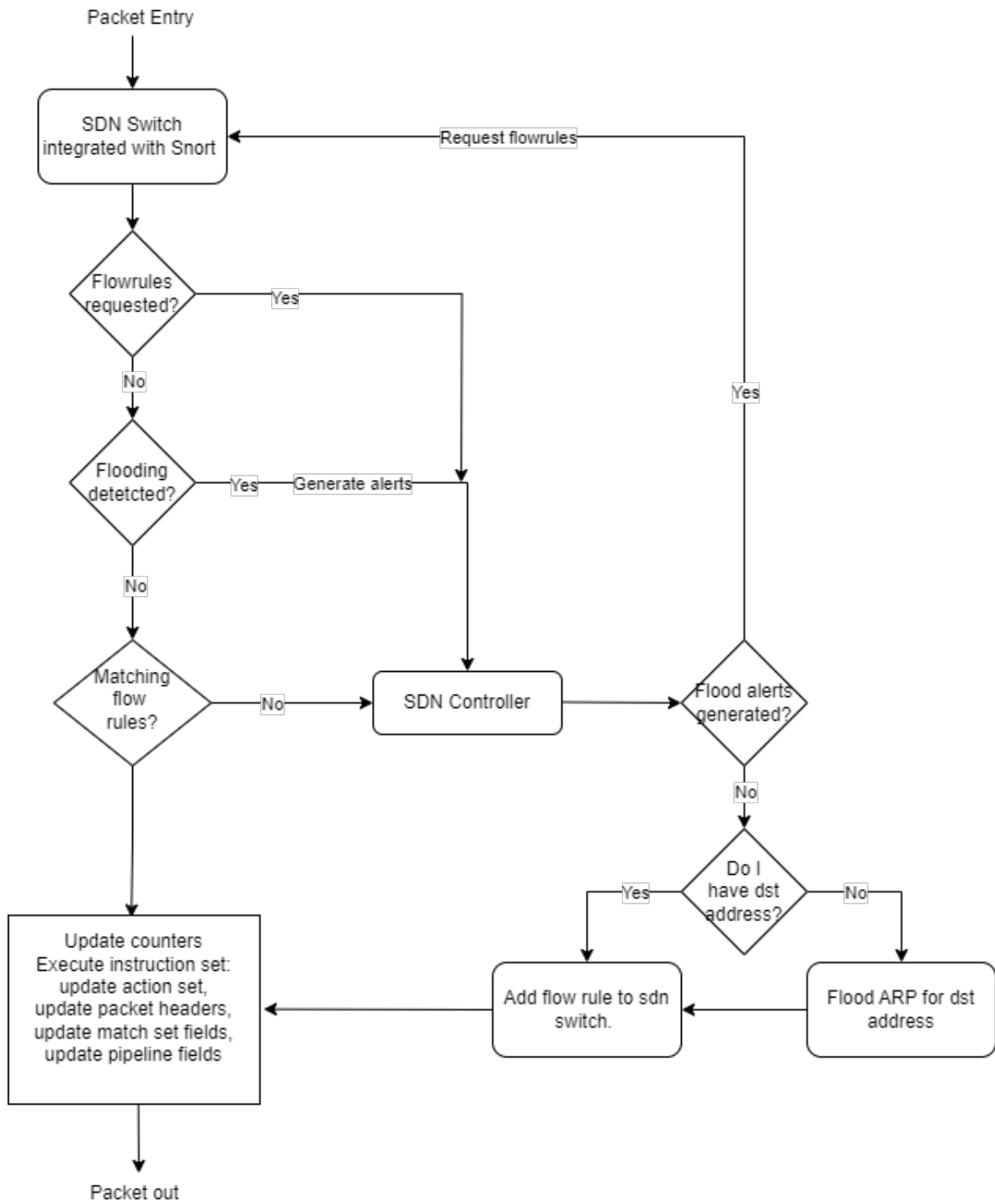


Figure 4.6: Flowchart for flooding mitigation configuration

controller asking for the existing flow rules in the SDN switch. Then, the SDN switch sends the existing flow rules to the controller for further analysis. Then the controller acts according to algorithm 4. If there are no signs of flooding, the switch proceeds to act as mentioned in section 3.3.4

#### 4.5.5 Configuration of Ryu to mitigate flooding and IP spoofing

In this section, the Ryu controller application is configured in such a way that it mitigates flooding and IP spoofing attacks simultaneously. The algorithm 5 explains how the Ryu application is configured. The Ryu controller is initialized, setting up the required data structures that will hold the state and configuration of the network. The controller then establishes a connection with the SDN switch by sending a feature request. This request is designed to retrieve the switch's capabilities and prepare it for subsequent configuration commands.

To address the issue of flooding, the controller initializes a monitoring thread. This thread operates independently of the main packet processing functions, allowing for continuous network monitoring without impacting regular operations. For each incoming packet, the Ryu controller extracts the source IP address to determine its legitimacy. This step is crucial for identifying potential IP spoofing attempts. If the source IP address is not already recorded in the controller's memory, it is considered new and is logged. If a subsequent packet from the same source IP address is received, it implies that the source is a repeated visitor. In this case, the controller sends a flow rule specific to that source IP to the switch. This flow rule instructs the switch on how to handle packets from this source in the future, usually allowing them to pass without further controller intervention. Here, the controller acts in the two configurations mentioned above in Section 4.5.3 and 4.5.4.

---

**Algorithm 5** Configuring Ryu Controller to Block IP Spoofing and Flooding

---

```

1: Initialize the Ryu application and set up data structures
2: Establish a connection to the switch and send a feature request
3: for each packet received do
4:   Extract source IP address
5:   if source IP address is new then
6:     Record the source IP address
7:     if a second packet from the same source IP is received then
8:       Send flow rule to switch for the source IP
9:     else
10:      Drop the packet to prevent IP spoofing
11:    end if
12:  else
13:    Apply action based on existing flow rules
14:  end if
15: end for
16: Initialize monitoring thread for flood detection
17: Upon State Change Event:
18: if datapath enters MAIN_DISPATCHER then
19:   Register datapath
20: else if datapath enters DEAD_DISPATCHER then
21:   Unregister datapath
22: end if
23: while true do
24:   Sleep for a predefined monitoring interval
25:   for each datapath do
26:     Send request for flow and port statistics
27:   end for
28:   Sleep for a short interval before next iteration
29: end while
30: Upon Flow Statistics Reply Event:
31: for each flow with priority 1 do
32:   Redirect matching packets to the controller
33:   Modify flow entry to prevent forwarding
34:   Log modification action
35: end for

```

---

This chapter discusses the results of various experiments conducted to evaluate the effectiveness of Snort IDS, and different configurations of Ryu controller application in an SDN environment under different attack scenarios, such as IP spoofing and flooding attacks. It provides detailed analyses of these experiments, including the methodology, attack simulations, data collection, and the performance of Snort IDS, configured Ryu application in detecting and mitigating these attacks.

## 5.1 Experiments

### 5.1.1 Experiment 1: Evaluation of Snort3's Detection Capabilities

#### 5.1.1.1 Objective

The primary objective of Experiment 1 is to assess the efficacy of Snort3 in identifying and differentiating various forms of network attacks, specifically focusing on flooding and spoofing scenarios.

#### 5.1.1.2 Attack Simulation

A series of controlled network attacks are simulated to test Snort3's detection capabilities. These attacks encompass a range of flooding and spoofing techniques using different protocols.

- **Flooding Attacks:** TCP, UDP, and ICMP flooding attacks are simulated to evaluate Snort3's response to high-volume traffic intended to overwhelm network resources.
- **Spoofing Attacks:** IP address spoofing attacks are executed to test Snort3's ability to identify packets with forged sender addresses, which are commonly used in various cyber attacks.

Table 5.1 in the document provides detailed information on the specific commands used to initiate each type of attack.

Snort IDS is deployed on the 's1-eth0' interface of the network switch with the command:

```
sudo -c /usr/local/etc/snort/snort.lua -i s1-eth0 -a alert_fast
```

Table 5.1: Commands used for different types of attacks

Attack	Command
ICMP flood	hping3 -icmp -flood dst_ip
ICMP spoof	hping3 -icmp -flood -rand-source dst_ip
TCP flood	hping3 -flood -S -p 80 dst_ip
TCP spoof	hping3 -flood -S -rand-source -p 80 dst_ip
UDP flood	hping3 -udp -flood -p 161 dst_ip
UDP spoof	hping3 -udp -flood -rand-source -p 161 dst_ip

Table 5.2: Table explaining the command used to implement Snort IDS

Flag	Description
-c	Specifies the path to the Snort configuration file.
-i	Specifies the interface on which Snort should be implemented
-a	Specifies the location where Snort should log alerts (alert_fast logs alerts on console).

### 5.1.1.3 Results

The results of the experiment are presented in Figures 5.1 to 5.6. Each figure corresponds to a specific type of attack and demonstrates the effectiveness of Snort IDS in detecting these attacks.

```

Node: s1" (root)@mininet-vm
12/28-22:55:36.048332 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048337 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048337 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048681 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048681 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048687 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048687 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048692 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048692 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048697 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048697 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048703 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048703 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048711 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048711 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048719 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048719 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048727 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048727 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048736 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048736 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048742 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048742 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048747 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 10.0.0.1 -> 10.0.0.3
12/28-22:55:36.048747 [**] [1:29456:3] "PROTOCOL-ICMP Flood Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 10.0.0.1 -> 10.0.0.3

```

Figure 5.1: ICMP Flood Alerts

Figure 5.1 shows the result obtained for the ICMP flood attack. It demonstrates that Snort IDS successfully generated alerts for ICMP flood detection. The figure shows two ICMP alerts generated for each ICMP flooding packet that enters the SDN environment.

```

Node: s1" (root)@mininet-vm
12/28-22:58:50.278615 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 115.133.0.142 -> 10.0.0.3
12/28-22:58:50.278615 [**] [1:29456:3] "PROTOCOL-ICMP Spoof Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 115.133.0.142 -> 10.0.0.3
12/28-22:58:50.278629 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 253.244.215.50 -> 10.0.0.3
12/28-22:58:50.278629 [**] [1:29456:3] "PROTOCOL-ICMP Spoof Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 253.244.215.50 -> 10.0.0.3
12/28-22:58:50.278645 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 111.113.33.75 -> 10.0.0.3
12/28-22:58:50.278645 [**] [1:29456:3] "PROTOCOL-ICMP Spoof Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 111.113.33.75 -> 10.0.0.3
12/28-22:58:50.278658 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 218.145.204.140 -> 10.0.0.3
12/28-22:58:50.278658 [**] [1:29456:3] "PROTOCOL-ICMP Spoof Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 218.145.204.140 -> 10.0.0.3
12/28-22:58:50.278672 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 250.94.115.169 -> 10.0.0.3
12/28-22:58:50.278672 [**] [1:29456:3] "PROTOCOL-ICMP Spoof Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 250.94.115.169 -> 10.0.0.3
12/28-22:58:50.278688 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 98.61.202.72 -> 10.0.0.3
12/28-22:58:50.278688 [**] [1:29456:3] "PROTOCOL-ICMP Spoof Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 98.61.202.72 -> 10.0.0.3
12/28-22:58:50.278704 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 168.167.101.17 -> 10.0.0.3
12/28-22:58:50.278704 [**] [1:29456:3] "PROTOCOL-ICMP Spoof Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 168.167.101.17 -> 10.0.0.3
12/28-22:58:50.278718 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 7.180.243.215 -> 10.0.0.3
12/28-22:58:50.278718 [**] [1:29456:3] "PROTOCOL-ICMP Spoof Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 7.180.243.215 -> 10.0.0.3
12/28-22:58:50.278733 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3] {ICMP} 84.14.103.168 -> 10.0.0.3
12/28-22:58:50.278733 [**] [1:29456:3] "PROTOCOL-ICMP Spoof Detected" [**] [Classification: Information Leak] [Priority: 2] {ICMP} 84.14.103.168 -> 10.0.0.3

```

Figure 5.2: ICMP Spoof Alerts

Figure 5.2 shows the result obtained for the ICMP spoofing attack. It demonstrates that Snort IDS successfully generated alerts for ICMP spoofing detection. The figure shows the CLI output for the two ICMP spoof alerts generated for each of the spoofed packets entering the network.

```

Node: s1" (root)@mininet-vm
12/28-23:02:06.456189 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53071 -> 10.0.0.3:80
12/28-23:02:06.456208 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53072 -> 10.0.0.3:80
12/28-23:02:06.456226 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53073 -> 10.0.0.3:80
12/28-23:02:06.456241 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53074 -> 10.0.0.3:80
12/28-23:02:06.456256 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53075 -> 10.0.0.3:80
12/28-23:02:06.456271 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53076 -> 10.0.0.3:80
12/28-23:02:06.456284 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53077 -> 10.0.0.3:80
12/28-23:02:06.456300 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53078 -> 10.0.0.3:80
12/28-23:02:06.456318 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53079 -> 10.0.0.3:80
12/28-23:02:06.456335 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53080 -> 10.0.0.3:80
12/28-23:02:06.456350 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53081 -> 10.0.0.3:80
12/28-23:02:06.456366 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53082 -> 10.0.0.3:80
12/28-23:02:06.456380 [**] [1:40063:5] "PROTOCOL-TCP ACK provocation attempt" [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 10.0.0.1:53083 -> 10.0.0.3:80

```

Figure 5.3: TCP Flood Alerts

Figure 5.3 shows the result obtained for the TCP flood attack. It demonstrates that Snort IDS successfully generated alerts for TCP flood detection. The figure shows each alert generated for each of the TCP flood packet that entered the SDN environment.

Figure 5.4 shows the result obtained for the TCP spoofing attack. It demonstrates that Snort IDS successfully generated alerts for TCP spoofing detection. The figure shows the CLI output for each of the alert generated for each TCP spoofed packet entering the network.

```

Node: s1 (root)@mininet-vm
12/28-23:05:19.904100 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 226.45.122.212:42542 -> 10.0.0.3:80
12/28-23:05:19.906213 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 226.179.195.98:42671 -> 10.0.0.3:80
12/28-23:05:19.906319 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 232.242.251.88:42679 -> 10.0.0.3:80
12/28-23:05:19.907006 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 226.204.70.101:42734 -> 10.0.0.3:80
12/28-23:05:19.907629 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 235.121.113.67:42754 -> 10.0.0.3:80
12/28-23:05:19.907646 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 227.110.142.196:42756 -> 10.0.0.3:80
12/28-23:05:19.907829 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 236.41.209.142:42777 -> 10.0.0.3:80
12/28-23:05:19.912418 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 226.52.18.186:43045 -> 10.0.0.3:80
12/28-23:05:19.912532 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 235.203.100.223:43059 -> 10.0.0.3:80
12/28-23:05:19.912615 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 226.253.165.111:43069 -> 10.0.0.3:80
12/28-23:05:19.912637 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 226.194.62.150:43072 -> 10.0.0.3:80
12/28-23:05:19.913584 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 226.117.236.213:43166 -> 10.0.0.3:80
12/28-23:05:19.915880 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 226.35.151.18:43422 -> 10.0.0.3:80
12/28-23:05:19.915893 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 232.129.251.143:43423 -> 10.0.0.3:80
12/28-23:05:19.915902 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 225.3.147.100:43424 -> 10.0.0.3:80
12/28-23:05:19.916076 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 227.213.121.251:43445 -> 10.0.0.3:80
12/28-23:05:19.916284 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 225.66.141.113:43468 -> 10.0.0.3:80
12/28-23:05:19.917573 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 235.70.4.101:43486 -> 10.0.0.3:80
12/28-23:05:19.917628 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 233.232.55.54:43493 -> 10.0.0.3:80
12/28-23:05:19.917657 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 229.218.117.103:43497 -> 10.0.0.3:80
12/28-23:05:19.917687 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 236.157.126.59:43501 -> 10.0.0.3:80
12/28-23:05:19.917962 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 235.246.225.35:43526 -> 10.0.0.3:80
12/28-23:05:19.918048 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 227.101.72.3:43528 -> 10.0.0.3:80
12/28-23:05:19.918087 [**] [116:410:1] "(ipv4) IPv4 packet from multicast source address" [**] [Priority: 3] {TCP} 227.113.98.117:43532 -> 10.0.0.3:80

```

Figure 5.4: TCP Spoof Alerts

Figure 5.5 shows the result obtained for the UDP flood attack. It demonstrates that Snort IDS successfully generated alerts for UDP flood detection. The figure shows the two alerts generated for each of the UDP flood packet that is entering the SDN environment.

```

Node: s1 (root)@mininet-vm
0.0.1:49931 -> 10.0.0.3:161
12/28-23:08:11.400076 [**] [1:1417:18] "PROTOCOL-UDP FLOOD DETECTED" [**] [Classification: Attempted Information Leak] [Priority: 2] {UDP} 10.0.0.1:49932
-> 10.0.0.3:161
12/28-23:08:11.400076 [**] [1:279:11] "SERVER-OTHER Bay/Nortel Nautica Marlin" [**] [Classification: Attempted Denial of Service] [Priority: 2] {UDP} 10.
0.0.1:49932 -> 10.0.0.3:161
12/28-23:08:11.400082 [**] [1:1417:18] "PROTOCOL-UDP FLOOD DETECTED" [**] [Classification: Attempted Information Leak] [Priority: 2] {UDP} 10.0.0.1:49933
-> 10.0.0.3:161
12/28-23:08:11.400082 [**] [1:279:11] "SERVER-OTHER Bay/Nortel Nautica Marlin" [**] [Classification: Attempted Denial of Service] [Priority: 2] {UDP} 10.
0.0.1:49933 -> 10.0.0.3:161
12/28-23:08:11.400088 [**] [1:1417:18] "PROTOCOL-UDP FLOOD DETECTED" [**] [Classification: Attempted Information Leak] [Priority: 2] {UDP} 10.0.0.1:49934
-> 10.0.0.3:161
12/28-23:08:11.400088 [**] [1:279:11] "SERVER-OTHER Bay/Nortel Nautica Marlin" [**] [Classification: Attempted Denial of Service] [Priority: 2] {UDP} 10.
0.0.1:49934 -> 10.0.0.3:161
12/28-23:08:11.400095 [**] [1:1417:18] "PROTOCOL-UDP FLOOD DETECTED" [**] [Classification: Attempted Information Leak] [Priority: 2] {UDP} 10.0.0.1:49935
-> 10.0.0.3:161
12/28-23:08:11.400095 [**] [1:279:11] "SERVER-OTHER Bay/Nortel Nautica Marlin" [**] [Classification: Attempted Denial of Service] [Priority: 2] {UDP} 10.
0.0.1:49935 -> 10.0.0.3:161
12/28-23:08:11.400100 [**] [1:1417:18] "PROTOCOL-UDP FLOOD DETECTED" [**] [Classification: Attempted Information Leak] [Priority: 2] {UDP} 10.0.0.1:49936
-> 10.0.0.3:161
12/28-23:08:11.400100 [**] [1:279:11] "SERVER-OTHER Bay/Nortel Nautica Marlin" [**] [Classification: Attempted Denial of Service] [Priority: 2] {UDP} 10.
0.0.1:49936 -> 10.0.0.3:161
12/28-23:08:11.400107 [**] [1:1417:18] "PROTOCOL-UDP FLOOD DETECTED" [**] [Classification: Attempted Information Leak] [Priority: 2] {UDP} 10.0.0.1:49937
-> 10.0.0.3:161
12/28-23:08:11.400107 [**] [1:279:11] "SERVER-OTHER Bay/Nortel Nautica Marlin" [**] [Classification: Attempted Denial of Service] [Priority: 2] {UDP} 10.
0.0.1:49937 -> 10.0.0.3:161

```

Figure 5.5: UDP Flood Alerts

Figure 5.6 shows the result obtained for the UDP spoofing attack. It demonstrates that Snort IDS successfully generated alerts for UDP spoofing detection. The figure shows the CLI output for each alert received for each UDP-spoofed packet entering the network.

These results collectively provide insights into the effectiveness of Snort3 in detecting various types of network attacks, supporting the evaluation's objectives.

The results of Experiment 1 demonstrate Snort3's effectiveness in detecting various network attacks, including flooding and spoofing scenarios. These findings are essential in assessing the reliability of Snort3 as a network IDS.



```

Node: s1 (root)@mininet-vm
12/28-23:12:02.091751 ** [116:150:1] "(decode) loopback IP" ** [Priority: 3] {UDP} 127.251.33.68:49216 -> 10.0.0.3:80
12/28-23:12:02.091979 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 228.174.193.68:49234 -> 10.0.0.3:80
12/28-23:12:02.092316 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 232.74.159.4:49262 -> 10.0.0.3:80
12/28-23:12:02.092426 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 227.4.217.127:49271 -> 10.0.0.3:80
12/28-23:12:02.092822 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 228.18.33.21:49287 -> 10.0.0.3:80
12/28-23:12:02.093528 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 236.232.130.31:49306 -> 10.0.0.3:80
12/28-23:12:02.093767 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 227.31.225.20:49321 -> 10.0.0.3:80
12/28-23:12:02.094811 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 225.227.63.147:49382 -> 10.0.0.3:80
12/28-23:12:02.094883 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 225.229.168.177:49388 -> 10.0.0.3:80
12/28-23:12:02.094945 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 236.58.246.167:49392 -> 10.0.0.3:80
12/28-23:12:02.104993 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 230.54.144.136:49399 -> 10.0.0.3:80
12/28-23:12:02.105278 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 238.142.227.1:49427 -> 10.0.0.3:80
12/28-23:12:02.107534 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 227.65.111.65:49472 -> 10.0.0.3:80
12/28-23:12:02.108756 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 234.150.108.192:49581 -> 10.0.0.3:80
12/28-23:12:02.108822 ** [116:150:1] "(decode) loopback IP" ** [Priority: 3] {UDP} 127.16.200.142:49585 -> 10.0.0.3:80
12/28-23:12:02.108830 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 236.100.176.134:49586 -> 10.0.0.3:80
12/28-23:12:02.108877 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 225.162.199.205:49592 -> 10.0.0.3:80
12/28-23:12:02.108934 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 228.3.108.127:49599 -> 10.0.0.3:80
12/28-23:12:02.109603 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 230.124.236.2:49605 -> 10.0.0.3:80
12/28-23:12:02.109804 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 225.122.132.151:49620 -> 10.0.0.3:80
12/28-23:12:02.109910 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 224.63.153.129:49632 -> 10.0.0.3:80
12/28-23:12:02.110081 ** [116:150:1] "(decode) loopback IP" ** [Priority: 3] {UDP} 127.33.250.112:49644 -> 10.0.0.3:80
12/28-23:12:02.110733 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 234.225.174.240:49661 -> 10.0.0.3:80
12/28-23:12:02.110779 ** [116:410:1] "(ipv4) IPv4 packet from multicast source address" ** [Priority: 3] {UDP} 225.93.136.251:49666 -> 10.0.0.3:80

```

Figure 5.6: UDP Spoof Alerts

In the case of flooding attacks, Snort3 successfully detected and generated alerts for TCP, UDP, and ICMP flooding attacks. This indicates its capability to identify high-volume traffic intended to overwhelm network resources, which is a common tactic employed by malicious actors.

Moreover, Snort3 exhibited proficiency in detecting spoofing attacks involving forged sender IP addresses. In both ICMP and TCP spoofing scenarios, the IDS was able to identify packets with spoofed addresses, highlighting its ability to counteract such deceptive tactics commonly employed in cyberattacks.

The results presented in Figures 5.1 to 5.6 confirm Snort3's capability to effectively detect and respond to network attacks, thereby enhancing network security. The system's ability to generate alerts promptly upon detecting suspicious activity provides network administrators with valuable information to respond to potential threats promptly.

Furthermore, it is worth noting that Snort3's performance in this experiment was evaluated under controlled conditions. Real-world network environments may pose additional challenges, and further testing is necessary to assess its performance in diverse and dynamic network settings.

In conclusion, Experiment 1 demonstrates that Snort3 is a robust and reliable network IDS capable of effectively identifying and differentiating various forms of network attacks, including flooding and spoofing attacks. These findings underscore its significance as a valuable tool for enhancing network security and mitigating potential threats. Future research can focus on extending this evaluation to more complex and realistic network scenarios.



## 5.1.2 Experiment 2: Impact and Mitigation of IP Spoofing Attacks on SDN

### 5.1.2.1 Objective

Experiment 2 is designed to analyze the effects of IP spoofing attacks on SDN performance and the efficacy of mitigation technique employed through the integration of Ryu controller applications and Snort IDS.

### 5.1.2.2 Attack Simulation and Mitigation Deployment

IP spoofing attacks were conducted within the established SDN environment to evaluate their impact. After initial observations, these attacks were executed again, this time in conjunction with the Algorithm 3 Ryu controller configuration, to understand the differing effects under this altered setup.

The performance metrics considered to be analyzed in this experiment are as follows:

- **Average Time:** The average response time taken for 100 requests from legitimate clients will be measured. This measurement will be taken under varying stress conditions, specifically by incrementally increasing the number of attackers from 1 to 10. This approach helps in understanding how network performance is impacted by escalating levels of attack.
- **Resource Utilization of Switch s1:** The CPU and memory utilization of switch s1 will be closely monitored. This data is crucial for assessing the operational efficiency and resource management of the network infrastructure under spoofing attacks.
- **Flow Rules Dynamics:** The changes in the number of flow rules within the SDN environment both during the attack phase and after the implementation of the mitigation strategy are observed. This observation aims to shed light on the network's adaptability and response to security breaches.

### 5.1.2.3 Results

**Average Time Analysis with Increasing Attackers:** The graph from Figure 5.7 depicts the average response time as the number of attackers scales up. The "orange" line shows the environment's performance under attack without any mitigation strategies in place. Here, a clear trend is visible where the average time increases with a little fluctuation as the number of attackers grows. This indicates a direct correlation between the number of attackers and the network's increased latency, demonstrating the detrimental impact of IP spoofing attacks on the SDN's performance.

The "blue" line illustrates the scenario where the mitigation technique is employed against the IP spoofing attack. As observed, the average time initially increases as the number of attackers rises from 1 to 2, suggesting that while mitigation is effective, it does have some impact on processing time. However, from 3 attackers onwards, the time increases only slightly towards 10 attackers when compared with the effect under

attack, indicating that the mitigation measures are scaling effectively to counteract the additional load imposed by the increasing number of attackers.

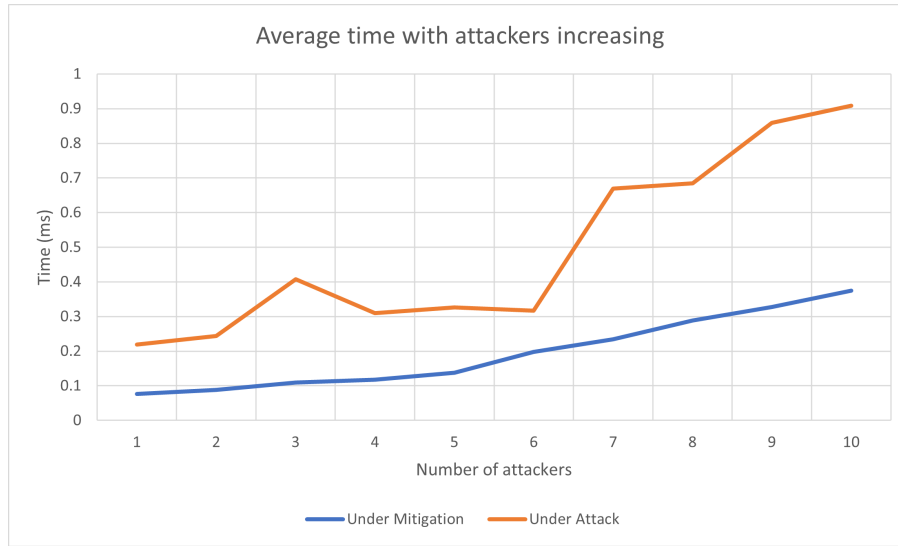


Figure 5.7: A graph which depicts the average time with increase in number of attackers under IP spoofing attack and under mitigation

**CPU and Memory Utilization:** The two graphs from Figures 5.8 and 5.9 depict the CPU and memory utilization of a network device under two conditions: when under an IP spoofing attack, and when mitigation is implemented.

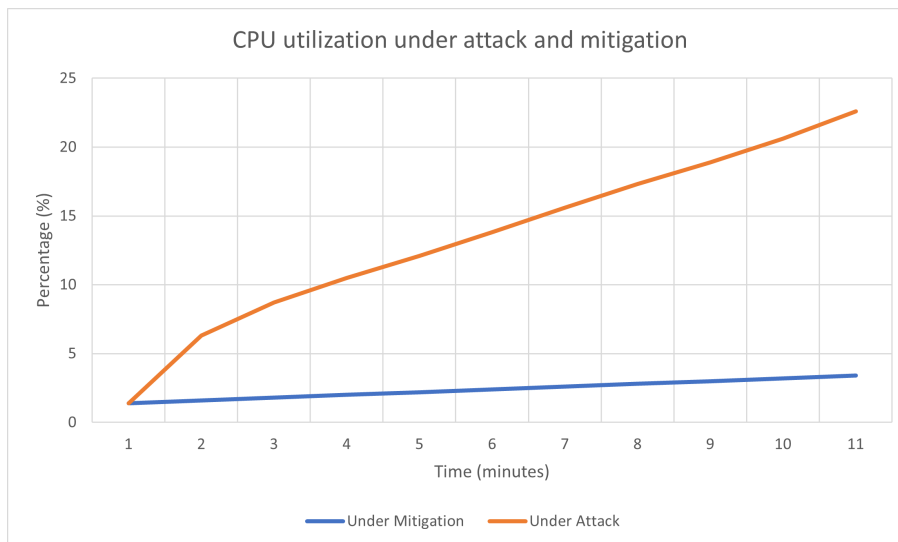


Figure 5.8: A graph which depicts the percentage of CPU Utilization of Switch under IP spoofing attack

In the Figure 5.8, the "orange" line shows CPU utilization under an IP spoofing attack, which increases sharply as time progresses, indicating a substantial computational overhead as the network responds to the attack. The "blue" line displays CPU utilization when the IP spoofing mitigation technique is employed, which is

only slightly increased towards the end, suggesting that the attack's impact on the CPU is minimal when the mitigation technique is employed.

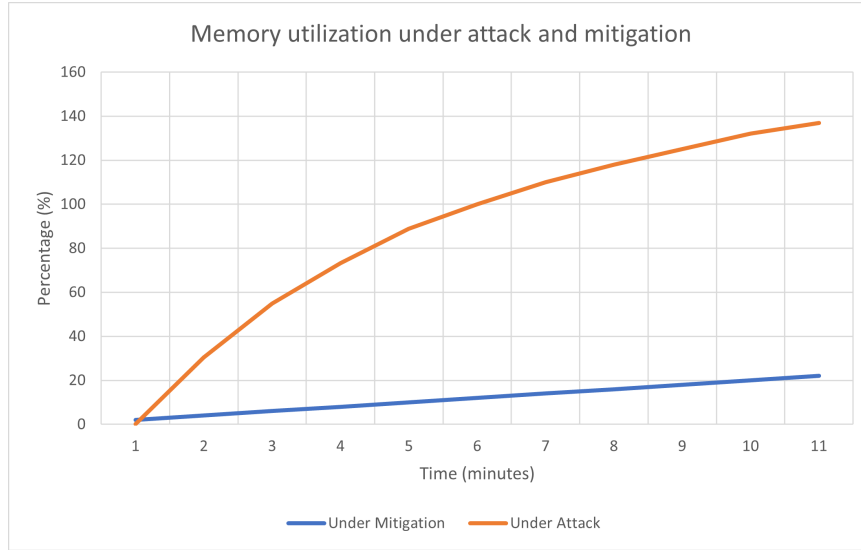


Figure 5.9: A graph which depicts the percentage of Memory Utilization of Switch under IP spoofing attack

In the Figure 5.9, the "orange" line and "blue" line represent memory utilization under IP spoofing attack and under IP spoofing mitigation technique, respectively. In Figure 5.9, it can be observed that the orange line is increased to more than 100 percent, i.e., 138, to the end of the graph because the switch is overwhelmed with so many flow rules when under attack. The memory reached more than 100 percent, that is, until 140 because, as the SDN environment is implemented in VirtualBox, the memory is dynamically allocated, which is why it was possible to reach more than 100 percent. Memory usage appears unaffected by the mitigation process; it only goes up to 20 percent of the memory assigned. It can be observed from the blue line that it gradually increased to 20 percent as time progressed when the SDN environment was employed with the IP spoofing mitigation technique.

**Flow Rules Analysis:** The two graphs in Figures 5.10 and 5.11, present the number of flow rules added to switch in an SDN environment under two conditions: during an IP spoofing attack and when IP spoofing mitigation strategy is applied.

The first graph, Figure 5.10, shows a linear and continuous increase in the number of flow rules over 10 minutes. This trend suggests that as the attack continues, the system is constantly generating new flow rules to handle the traffic. This could potentially lead to network performance issues due to the processing overhead of managing a large number of flow rules.

The second graph, Figure 5.11, depicts a very different pattern. Here, there is a sharp rise in the number of flow rules within the first two minutes, reaching a peak and then sharply declining almost back to baseline levels by the four-minute mark. This spike implies the network's immediate response to the attack, where mitigation mechanisms are activated. The subsequent drop indicates that the network is able to stabilize by effectively neutralizing the threat and then reducing the number of

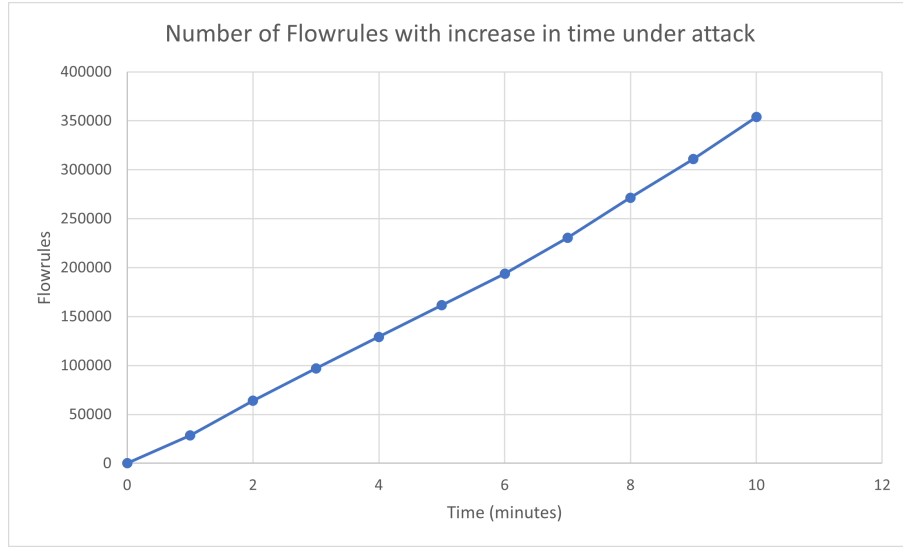


Figure 5.10: A graph which depicts the number of flow rules with time increasing under IP spoofing attack

flow rules to a normal operating level.

The number of flow rules increases linearly with time due to same number of packets being sent by the attacker periodically throughout the experiment, reflecting the SDN controller's attempt to manage the growing number of traffic flows, which could be a result of the attack's complexity and the controller's efforts to maintain network order.

**Flow Rules Variation with Mitigation (Figure 5.11):** An initial surge in the number of flow rules is observed, which quickly peaks and then diminishes. This pattern represents the rapid deployment of mitigation technique followed by a stabilization of the network, suggesting an effective mitigation response.

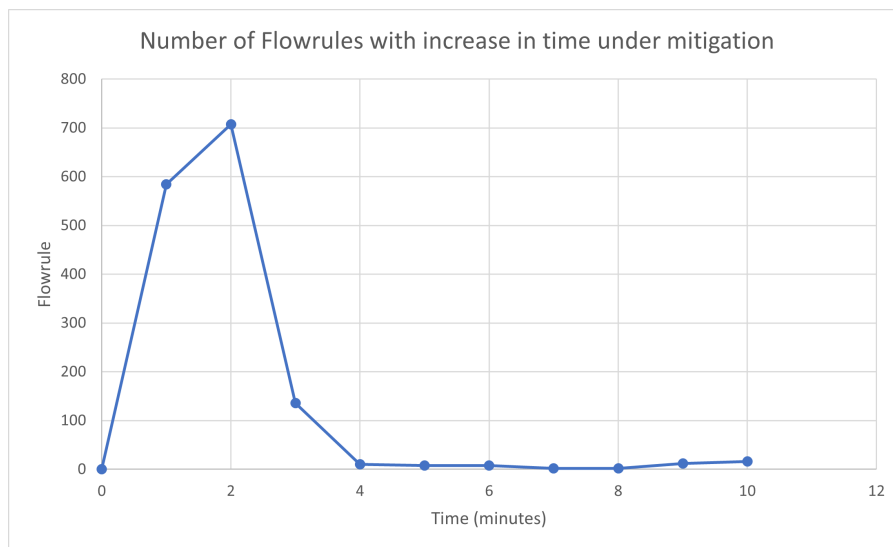


Figure 5.11: A graph which depicts the number of flowrules with time increasing under mitigation

There was a constant increase until the first minute, and that increase dropped between the first and second minute comparatively because the mitigation technique was implemented and it took some time to remove the already existing flow rules considering the `idle_time` parameter which is 120 seconds considered for our Ryu controller configuration.

### 5.1.3 Experiment 3

#### 5.1.3.1 Objective

The primary goal of Experiment 3 is to analyze the effect of flooding attacks on SDN performance and the efficacy of the mitigation technique employed through the integration of the Ryu controller and Snort IDS.

#### 5.1.3.2 Attack simulation

Flooding attacks were conducted within the established SDN environment to evaluate their impact. After initial observations, these attacks were executed again, this time in conjunction with the Algorithm 4 Ryu controller configuration, to understand the differing effects under this altered setup.

#### 5.1.3.3 Results

The graph provided, Figure 5.12, illustrates the response times of an SDN environment subjected to a flooding attack, comparing periods with and without mitigation efforts.

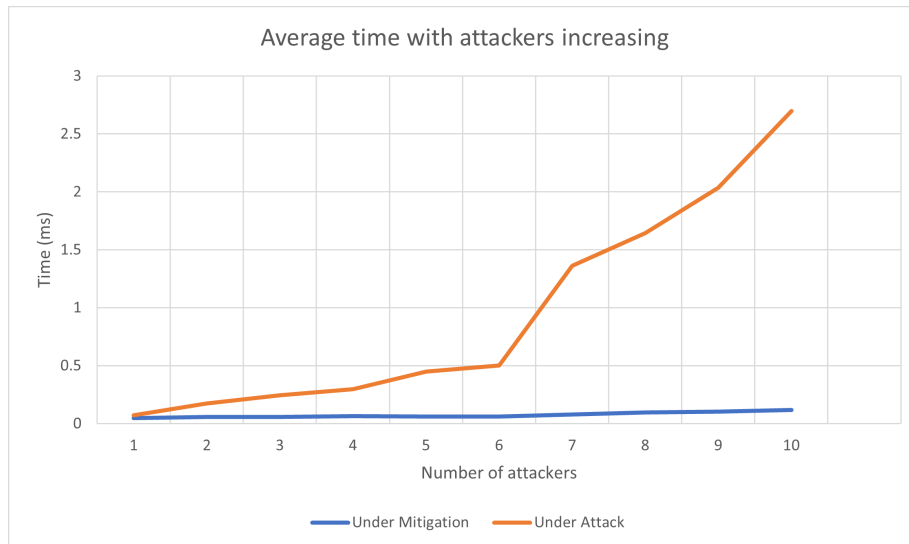


Figure 5.12: A graph which depicts the average time with increase in number of attackers under flooding attack and under mitigation

The "orange line" represents the average response time of the network while it is under a flooding attack. There is a clear upward trend as the number of attackers increases, with the response time escalating significantly. This pattern indicates that

the network is increasingly strained as more malicious requests flood the system, leading to a degradation of performance for legitimate traffic.

The "blue line" shows the average response time when mitigation strategies are implemented. Despite the increasing number of attackers, the response time remains consistently low, with only a slight, almost negligible, upward trend. This consistency demonstrates the effectiveness of the mitigation techniques in place, which manage to keep the network's response time stable and within acceptable limits, even as the attack intensifies.

### 6.1 Introduction

In this section, it is explored why SDN is particularly susceptible to DDoS attacks, a focus stemming from SDN’s centralized control architecture. The literature review highlights both the innovative potential and the security challenges of SDN, leading to a comprehensive experimental investigation. The experiments specifically address how various DDoS attacks, such as TCP, UDP, and ICMP spoof and flood, significantly impact SDN. These findings not only elucidate the process of packet handling under attack conditions but also offer insights into effective mitigation strategies, enhancing our understanding of SDN’s security landscape.

The research is focused exclusively on spoofing and flooding attacks (ICMP, TCP, UDP) on SDN. This selection was based on the observation that other forms of attacks have a relatively minimal impact on SDN architectures. Spoofing and flooding attacks are particularly pertinent due to their ability to exploit the centralized nature of SDN, overwhelming the control plane and disrupting network operations. This targeted approach in our study allows for a deeper, more relevant analysis of the most critical vulnerabilities facing SDNs today, providing valuable insights for developing robust defense mechanisms. The integration of Snort IDS in SDN environments offers a strategic advantage, particularly in the context of research focused on mitigating spoofing and flooding attacks.

Snort’s exceptional protocol analysis and content-matching capabilities align seamlessly with the need to address critical vulnerabilities in SDNs. Its customizability and open-source nature allow for tailored defense mechanisms against rapidly evolving network threats, including those prevalent in SDN architectures. Snort’s efficiency in processing large volumes of traffic with minimal latency is vital for maintaining the performance and integrity of SDNs, emphasizing its suitability for real-time intrusion detection and immediate response in such dynamic network environments.

The selection of Mininet as the primary tool for SDN simulation in our research is grounded in several key reasons, especially when compared to our experience with OMNeT++. While we initially considered OMNeT++ for its comprehensive simulation capabilities, we encountered significant challenges in successfully implementing SDN within it. The primary issue was the reliance on third-party resources for SDN integration, which proved to be unreliable in OMNeT++. This led us to pivot to-

wards Mininet. Mininet is selected as it stands out for its specialized focus on SDN. It offers a more straightforward and reliable platform for creating realistic SDN environments, allowing us to simulate a network on a single machine with minimal resource utilization. This ease of use, combined with its robust support for SDN technologies like OpenFlow, makes Mininet an ideal choice for conducting detailed and effective SDN research.

The selection of the Ryu controller for our SDN experiments was primarily driven by its strong OpenFlow compatibility, essential for effective network flow management in SDN environments. Ryu's modular design offers unparalleled flexibility and customizability, allowing us to tailor the controller to our specific experimental needs. Its user-friendly Python-based framework facilitates ease of development, making it accessible even to those with limited programming expertise.

For our experimentation, we utilized a simple SDN architecture within a Mininet virtual machine on VirtualBox, constrained by limited resources. This setup was chosen as it realistically reflects the challenges of setting up extensive network simulations with numerous hosts and switches in a resource-constrained environment. This approach allows us to focus on the core aspects of SDN vulnerabilities to spoofing and flooding attacks while acknowledging the limitations imposed by our experimental setup.

## 6.2 Evaluation of Experiments

### 6.2.1 Evaluation for Experiment 1

Experiment 1, focusing on the evaluation of Snort3's detection capabilities, directly aligns with the first objective and partially with the second objective of this research. It contributes to the investigation of detection methods for flooding and IP spoofing attacks in an SDN environment and explores the integration of Snort IDS as a mitigation technique.

The experiment used controlled simulations of various flooding (TCP, UDP, ICMP) and spoofing attacks. This approach allowed for a thorough assessment of Snort3's ability to identify and differentiate between these types of attacks. Snort3 demonstrated its effectiveness by successfully generating alerts for each type of simulated attack. This indicates its proficiency in recognizing high-volume traffic and forged sender addresses characteristic of flooding and spoofing attacks.

The deployment of Snort IDS on the 's1-eth0' interface showcases the practical aspect of integrating intrusion detection systems within an SDN setup. The successful detection of all simulated attacks underlines the potential of Snort IDS to be a reliable component in an SDN environment for mitigating DDoS attacks.

The results confirm the capability of Snort3 to operate effectively in an SDN environment. It detects diverse attack scenarios, thus addressing a significant concern in SDN security. The accurate detection of flooding attacks, which involve overwhelming network resources with high-volume traffic, indicates Snort3's robustness



in managing and analyzing large-scale data flows, a critical requirement in SDN. The ability to identify spoofed packets reinforces the efficacy of Snort3 in handling more sophisticated forms of cyber threats, where attackers mask their identity.

These findings are integral to understanding the practical application and reliability of Snort3 as a network IDS within the SDN framework. It provides a foundation for developing more advanced security measures and protocols in SDN.

### 6.2.2 Evaluation for Experiment 2

Experiment 2's focus on analyzing the impact and mitigation of IP spoofing attacks on SDN directly contributes to the second and third objectives of the study. It not only explores DDoS attack mitigation technique in SDN but also conducts a thorough experimental evaluation to analyze and interpret the findings.

The experiment involved conducting IP spoofing attacks within an SDN setup and employing a combination of Ryu controller applications and Snort IDS for mitigation. This provided a practical scenario to evaluate the effectiveness of these techniques. The use of Algorithm 3 Ryu controller configuration for mitigation demonstrated how specific strategies could be integrated into SDN for handling spoofing attacks, revealing both the strengths and weaknesses of these approaches. The experiment focused on various metrics like average response time, resource utilization of OpenFlow Switch, and flow rules dynamics. These metrics were critical in understanding how the SDN environment responds to attacks and mitigation efforts.

The increase in average response time, CPU, and memory utilization under attack conditions (as shown in Figures 5.7, 5.8, and 5.9) indicates the detrimental impact of IP spoofing attacks on network performance. The less effect on average response time, CPU, and memory utilization under mitigation (as seen in the respective figures) demonstrates the effectiveness of the employed mitigation strategy. This suggests that while mitigation impacts processing time, it is capable of scaling effectively to handle increased attack loads.

The variation in the number of flow rules under attack and mitigation conditions (Figures 5.10 and 5.11) provides insights into the SDN controller's adaptability. The initial surge in flow rules followed by a rapid decline under mitigation suggests an effective and efficient network response to neutralize threats.

### 6.2.3 Evaluation of Experiment 3

Experiment 3's focus on analyzing the effect of flooding attacks on SDN performance and the efficacy of mitigation techniques aligns with the study's first and second objectives. It also contributes to the third objective by providing experimental evaluations and analyses.

The experiment involved simulating flooding attacks within the SDN environment. This approach was essential for investigating the detection capabilities of the network in the face of high-volume traffic. The results from the experiment, particularly the escalated response times under attack, underscore the necessity and

challenge of detecting flooding attacks in SDN. It highlights the importance of effective detection methods in maintaining network integrity. The integration of the Ryu controller (Algorithm 4) and Snort IDS in the experimental setup provided a practical framework to assess the mitigation strategies against flooding attacks. The consistently low response times during mitigation, despite the increasing number of attackers, demonstrate the robustness of the techniques employed in safeguarding the SDN environment.

The significant increase in response time under flooding attack conditions (as shown in Figure 5.12) clearly illustrates the vulnerability of SDN environments to such high-volume traffic scenarios. The stable and low response times during mitigation, even with a rising number of attackers, are indicative of the effectiveness of the mitigation strategies. This demonstrates the potential of the Ryu controller and Snort IDS in maintaining network performance under stress.

The results show that the implemented mitigation techniques are practically effective. This is crucial for real-world applications in SDN environments where the predictability of the response under attack is vital. The experiment provides valuable insights into the resilience of SDN architectures against flooding attacks. The ability to maintain consistent performance in attack scenarios is key to the reliability and trustworthiness of SDN systems.

### 6.3 Evaluation of Research questions

After extensive research and experimentation, this study successfully achieved its aim, which was to comprehensively examine the DDoS attacks and to develop effective methods for detecting and preventing such attacks, addressing the research questions formulated for this study.

For RQ1, "To what extent does integration of Snort IDS in SDN demonstrate effectiveness in detecting flooding and IP spoofing attacks?", experiment 1 was conducted to thoroughly examine the Snort IDS's effectiveness in detecting ICMP, TCP, UDP flooding and spoofing attacks. The results of Experiment 1 demonstrate Snort3's effectiveness in detecting all the flooding and spoofing scenarios. These findings are essential in assessing the reliability of Snort3 as a network IDS.

For RQ2, "How can Snort IDS be integrated into an SDN environment to configure a mitigation technique for flooding and IP spoofing attacks?", the study started exploring various mitigation techniques that already exist and found that there is scope to develop a simple controller configuration. Then, the Ryu controller application is chosen to be configured for this study. Firstly, the Ryu controller application is configured in such a way that it mitigates IP spoofing attacks, then flooding attacks, and later both attacks.

For RQ3, "Find the effects of flooding and spoofing attacks on SDN environment while under attack and mitigation", experiments 2 and 3 are conducted to examine the effects of flooding and spoofing attacks and also tested when the SDN environ-

ment is employed with mitigation techniques developed for RQ2. The mitigation techniques were successful in mitigating flooding and spoofing scenarios.

## 6.4 Limitations

While the thesis on SDN DDoS attack mitigation, focusing on IP spoofing and flooding attacks, provides valuable insights, it's essential to acknowledge and address potential limitations to ensure a balanced interpretation of the findings.

1. Findings are specific to the experimental setup and may not be directly applicable to all SDN environments or network configurations.
2. In this thesis, hping3 was employed to generate attack packets. However, it is important to acknowledge that if an alternative network tool is chosen by an attacker to flood the attack packets, there could be potential limitations to the findings of this study. This is because we only used alerts which detect only the type of payload that an hping3 command sends. More alerts should be included to detect various attack packets for overcoming this limitation.
3. The proposed method of waiting for a response from the same IP to determine legitimacy introduces a limitation in terms of user experience. Requiring a second request for verification may create a suboptimal user experience, causing inconvenience and delay for legitimate users who may not expect or tolerate additional steps in their interactions with the network services. This limitation underscores the importance of balancing security measures with user convenience and the potential impact on overall satisfaction with the service.
4. A potential limitation arises from the dependency on a specific SDN switch type or model in the thesis. If there is a change in the SDN switch used, the effectiveness and compatibility of the proposed DDoS mitigation technique may be compromised.

### 7.1 Conclusion

This research embarked on a comprehensive journey to explore the vulnerabilities of Software-Defined Networking (SDN) to Distributed Denial of Service (DDoS) attacks and to develop effective mitigation strategies. The significance of this work is anchored in the pivotal role of SDN in modern network infrastructures, including data centers, enterprise networks, and cloud services. The susceptibility of SDN to DDoS attacks, particularly flooding and IP spoofing, posed a formidable challenge that this study aimed to address.

Through a series of meticulously designed experiments, the research evaluated the effectiveness of the Snort Intrusion Detection System (IDS) in detecting various forms of DDoS attacks. The results demonstrated Snort's robust capabilities in identifying and alerting the controller about potential security breaches. This was particularly evident in its success in detecting flooding and IP spoofing attacks across different protocols, highlighting its adaptability to SDN environments.

Moreover, the study delved into the performance impacts of these attacks on SDN. The findings revealed that DDoS attacks could significantly degrade network performance, evidenced by increased response times and resource utilization. However, the integration of the Ryu controller with Snort IDS presented a promising mitigation technique. This approach effectively stabilized the network under attack conditions, as seen in the controlled CPU utilization and efficient management of flow rules.

A critical insight from this research is the importance of tailored security solutions in SDN environments. While traditional security measures have their place, the unique architecture of SDN requires specialized strategies. The combined use of Ryu controller applications and Snort IDS in this study exemplifies such approach, offering a significant advancement in protecting SDN against DDoS attacks.

In conclusion, this research makes a substantial contribution to the field of network security, particularly in the context of SDN. Placing Snort IDS in the SDN switch's data plane allows for the early detection of attacks, enabling swift mitigation as alerts are generated to the controller. This approach not only underscores the heightened risks associated with the centralized control feature of SDN but also provides a practical solution to mitigate these risks safely on the data plane, preventing the propagation of attacks deeper into the network. The success of this proposed

mitigation technique holds immense potential for enhancing the resilience and reliability of SDNs, showcasing the significance of addressing security concerns at their inception within the network infrastructure.

## 7.2 Future Work

The future research building on the findings of this study can extend the scope to address more complex network environments, which is important for the ever-evolving landscape of cyber threats. This involves exploring the applicability of the developed mitigation strategy in more complex network architectures. As cyber threats continue to evolve, research must keep pace by developing advanced detection and mitigation techniques which include focusing on DDoS attack vectors that utilize AI and machine learning, adapting to encrypted traffic. As this study examined the effect of DDoS attacks in SDN in the data plane, the future research can be focused on finding the effect of the attacks on control plane. Testing and comparing the performance of various SDN controllers beyond Ryu in DDoS mitigation would provide a broader understanding of their strengths and weaknesses.

---

## References

- [1] “Software-defined networking (sdn) definition - open networking foundation,” <https://opennetworking.org/sdn-definition/>, (Accessed on 03/24/2023).
- [2] S. Ahmad and A. H. Mir, “Sdn interfaces: protocols, taxonomy and challenges,” *Int. J. Wirel. Microwave Technol. (IJWMT)*, vol. 12, no. 2, pp. 11–32, 2022.
- [3] M. Alsaeedi, M. M. Mohamad, and A. A. Al-Roubaiey, “Toward adaptive and scalable openflow-sdn flow control: A survey,” *IEEE Access*, vol. 7, pp. 107 346–107 379, 2019.
- [4] N. Anerousis, P. Chemouil, A. A. Lazar, N. Mihai, and S. B. Weinstein, “The origin and evolution of open programmable networks and sdn,” *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1956–1971, 2021.
- [5] B. Caswell, J. Beale, and A. Baker, *Snort intrusion detection and prevention toolkit*. Syngress, 2007.
- [6] Y. Cui, L. Yan, S. Li, H. Xing, W. Pan, J. Zhu, and X. Zheng, “Sd-anti-ddos: Fast and efficient ddos defense in software-defined networks,” *Journal of Network and Computer Applications*, vol. 68, pp. 65–79, 2016.
- [7] M. C. Dacier, H. König, R. Cwalinski, F. Kargl, and S. Dietrich, “Security challenges and opportunities of software-defined networking,” *IEEE Security & Privacy*, vol. 15, no. 2, pp. 96–100, 2017.
- [8] C. Dillon and M. Berkelaar, “Openflow (d) dos mitigation,” *Technical Report (Feb. 2014)*, 2014.
- [9] W. Eddy, “Tcp syn flooding attacks and common mitigations,” Tech. Rep., 2007.
- [10] S. Fichera, L. Galluccio, S. C. Grancagnolo, G. Morabito, and S. Palazzo, “Op-eretta: An openflow-based remedy to mitigate tcp synflood attacks against web servers,” *Computer Networks*, vol. 92, pp. 89–100, 2015.
- [11] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, “Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments,” *Computer Networks*, vol. 62, pp. 122–136, 2014.
- [12] P. Goransson, C. Black, and T. Culver, *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.
- [13] M. Granát, “Mobile robot control and guidance using computer vision,” [https://theses.cz/id/dmmadw/Bachelor\\_thesis\\_Granat\\_Archive.pdf](https://theses.cz/id/dmmadw/Bachelor_thesis_Granat_Archive.pdf), 2018.

- [14] M. I. Hamed, B. M. ElHalawany, M. M. Fouda, and A. S. T. Eldien, "A novel approach for resource utilization and management in sdn," in *2017 13th International Computer Engineering Conference (ICENCO)*. IEEE, 2017, pp. 337–342.
- [15] B. Hardin, D. Comer, and A. Rastegarnia, "On the unreliability of network simulation results from mininet and iperf," *International Journal of Future Computer and Communication*, vol. 12, no. 1, 2023.
- [16] A. Kalliola, K. Lee, H. Lee, and T. Aura, "Flooding ddos mitigation and traffic management with software defined networking," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*. IEEE, 2015, pp. 248–254.
- [17] R. Khan, N. AlHarbi, G. AlGhamdi, and L. Berriche, "Virtualization software security: Oracle vm virtualbox," in *2022 Fifth International Conference of Women in Data Science at Prince Sultan University (WiDS PSU)*. IEEE, 2022, pp. 58–60.
- [18] K. Kirkpatrick, "Software-defined networking," *Communications of the ACM*, vol. 56, no. 9, pp. 16–19, 2013.
- [19] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *Journal of Network and Computer Applications*, vol. 156, p. 102563, 2020.
- [20] Y. Li, D. Zhang, J. Taheri, and K. Li, "Sdn components and openflow," *Big Data Softw. Defin. Networks*, vol. 12, pp. 49–67, 2018.
- [21] S. Lim, J. Ha, H. Kim, Y. Kim, and S. Yang, "A sdn-oriented ddos blocking scheme for botnet-based attacks," in *2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2014, pp. 63–68.
- [22] J. Lin. (2023) Pigrelay: A python script for xyz. Accessed on 09/12/23. [Online]. Available: <https://github.com/John-Lin/pigrelay/blob/master/pigrelay.py>
- [23] J. Liu, Y. Lai, and S. Zhang, "Fl-guard: A detection and defense system for ddos attack in sdn," in *Proceedings of the 2017 international conference on cryptography, security and privacy*, 2017, pp. 107–111.
- [24] S. Luo, J. Wu, J. Li, and B. Pei, "A defense mechanism for distributed denial of service attack in software-defined networks," in *2015 Ninth International Conference on Frontier of Computer Science and Technology*. IEEE, 2015, pp. 325–329.
- [25] T. Mahjabin, Y. Xiao, G. Sun, and W. Jiang, "A survey of distributed denial-of-service attack, prevention, and mitigation techniques," *International Journal of Distributed Sensor Networks*, vol. 13, no. 12, p. 1550147717741463, 2017.
- [26] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," in *2015 International Conference on Computing, Networking and Communications (ICNC)*, 2015, pp. 77–81.
- [27] N. I. Mowla, I. Doh, and K. Chae, "Multi-defense mechanism against ddos in sdn based cdni," in *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, 2014, pp. 447–451.

- [28] T. Muhammad, “Revolutionizing network control: Exploring the landscape of software-defined networking (sdn),” *International Journal Of Computer Science And Technology*, vol. 3, no. 1, pp. 36–68, 2019.
- [29] *OpenFlow Specification. Version 1.5.1 (Wire Protocol 0x06)*, Open Networking Foundation, 2015, accessed: [Insert date of access].
- [30] M. Paliwal, D. Shrimankar, and O. Tembhurne, “Controllers in sdn: A review report,” *IEEE access*, vol. 6, pp. 36 256–36 270, 2018.
- [31] M. Á. B. Pérez, N. Y. S. Losada, E. R. Sánchez, G. M. Gaona *et al.*, “State of the art in software defined networking (sdn),” *Visión electrónica*, vol. 13, no. 1, pp. 178–194, 2019.
- [32] R. Project. (2023) Integrating ryu with snort. Accessed on 09/09/23. [Online]. Available: [https://ryu.readthedocs.io/en/latest/snort\\_integrate.html](https://ryu.readthedocs.io/en/latest/snort_integrate.html)
- [33] K. C. Purohit, M. Anand Kumar, A. Saxena, and A. Mittal, “The impact of icmp attacks in software-defined network environments,” in *International Conference on Computational Intelligence and Data Engineering*. Springer, 2022, pp. 319–333.
- [34] B. Ramkumar and T. Subbulakshmi, “Tcp syn flood attack detection and prevention system using adaptive thresholding method,” in *ITM Web of Conferences*, vol. 37. EDP Sciences, 2021, p. 01016.
- [35] D. B. Rawat and S. R. Reddy, “Software defined networking architecture, security and energy efficiency: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 325–346, 2016.
- [36] M. Revathi, V. Ramalingam, and B. Amutha, “A machine learning based detection and mitigation of the ddos attack by using sdn controller framework,” *Wireless Personal Communications*, pp. 1–25, 2021.
- [37] M. Roesch *et al.*, “Snort: Lightweight intrusion detection for networks.” in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.
- [38] Ryu Project. (2023) Ryu sdn framework. [Online]. Available: <https://ryu-sdn.org/>
- [39] R. Sahay, G. Blanc, Z. Zhang, and H. Debar, “Towards autonomic ddos mitigation using software defined networking,” in *SENT 2015: NDSS workshop on security of emerging networking technologies*. Internet society, 2015.
- [40] A. N. H. D. Sai, B. H. Tilak, N. S. Sanjith, P. Suhas, and R. Sanjeetha, “Detection and mitigation of low and slow ddos attack in an sdn environment,” in *2022 International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics ( DISCOVER)*, 2022, pp. 106–111.
- [41] M. Shtern, R. Sandel, M. Litoiu, C. Bachalo, and V. Theodorou, “Towards mitigation of low and slow application ddos attacks,” in *2014 IEEE international conference on cloud engineering*. IEEE, 2014, pp. 604–609.
- [42] D. Singh, B. Ng, Y.-C. Lai, Y.-D. Lin, and W. K. Seah, “Modelling software-defined networking: Software and hardware switches,” *Journal of Network and Computer Applications*, vol. 122, pp. 24–36, 2018.



- [43] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and R. Buyya, "Ddos attacks in cloud computing: Issues, taxonomy, and future directions," *Computer Communications*, vol. 107, pp. 30–48, 2017.
- [44] P. Team. (2023) Putty - a free ssh and telnet client. [Online]. Available: <https://www.putty.org/>
- [45] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "Ddos attack protection in the era of cloud computing and software-defined networking," *Computer Networks*, vol. 81, pp. 308–319, 2015.
- [46] H. Wang and B. Wu, "Sdn-based hybrid honeypot for attack capture," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, 2019, pp. 1602–1606.
- [47] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed ddos detection mechanism in software-defined networking," in *2015 IEEE Trustcom/Big-DataSE/ISPA*, vol. 1. IEEE, 2015, pp. 310–317.
- [48] X. Wang, M. Chen, and C. Xing, "Sdsnm: A software-defined security networking mechanism to defend against ddos attacks," in *2015 ninth international conference on frontier of computer science and technology*. IEEE, 2015, pp. 115–121.
- [49] Y. Wang, J. Ding, T. Zhang, Y. Xiao, and X. Hei, "From replay to regeneration: Recovery of udp flood network attack scenario based on sdn," *Mathematics*, vol. 11, no. 8, p. 1897, 2023.
- [50] L. Wei and C. Fung, "Flowranger: A request prioritizing algorithm for controller dos attacks in software defined networks," in *2015 IEEE International Conference on Communications (ICC)*. IEEE, 2015, pp. 5254–5259.
- [51] T. Xu, D. Gao, P. Dong, H. Zhang, C. H. Foh, and H.-C. Chao, "Defending against new-flow attack in sdn-based internet of things," *IEEE Access*, vol. 5, pp. 3431–3443, 2017.
- [52] Y. Yu, X. Li, X. Leng, L. Song, K. Bu, Y. Chen, J. Yang, L. Zhang, K. Cheng, and X. Xiao, "Fault management in software-defined networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 349–392, 2018.
- [53] A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou, "Orchsec: An orchestrator-based architecture for enhancing network-security using network monitoring and sdn control functions," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–9.
- [54] A. M. Zarca, J. B. Bernabe, A. Skarmeta, and J. M. A. Calero, "Virtual iot honeynets to mitigate cyberattacks in sdn/nfv-enabled iot networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1262–1277, 2020.
- [55] C. Zhang, G. Hu, G. Chen, A. K. Sangaiah, P. Zhang, X. Yan, and W. Jiang, "Towards a sdn-based integrated architecture for mitigating ip spoofing attack," *IEEE Access*, vol. 6, pp. 22 764–22 777, 2017.

- [56] L. Zhu, M. M. Karim, K. Sharif, C. Xu, F. Li, X. Du, and M. Guizani, “Sdn controllers: A comprehensive analysis and performance evaluation study,” *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–40, 2020.

