



Securing SDN Data Plane: Investigating the effects of IP Spoofing Attacks on SDN Switches and its Mitigation

Simulation of IP spoofing using Mininet

ANIRUDH SAI MADIRAJU
SHIVAKUMAR YADAV JABBU

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in 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):

ANIRUDH SAI MADIRAJU

E-mail: anmd21@student.bth.se

SHIVAKUMAR YADAV JABBU

E-mail: shja20@student.bth.se

University advisor:

Dragos Ilie

Department of Computer Science

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

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Software-Defined Networking (SDN) represents a network architecture that offers a separate control and data layer, facilitating its rapid deployment and utilization for diverse purposes. However, despite its ease of implementation, SDN is susceptible to numerous security attacks, primarily stemming from its centralized nature. Among these threats, Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks pose the most substantial risks. In the event of a successful attack on the SDN controller, the entire network may suffer significant disruption. Hence, safeguarding the controller becomes crucial to ensure the integrity and availability of the SDN network.

This thesis focuses on examining the IP spoofing attack and its impact on the Data Plane, particularly concerning the metrics of an SDN switch. The investigation centers around attacks that manipulate flow-rules to amplify the number of rules and deplete the resources of a switch within the Data Plane of an SDN network. To conduct the study, a software-defined network architecture was constructed using Mininet, with a Ryu controller employed for managing network operations. Various experiments were carried out to observe the response of the SDN system when subjected to an IP spoofing attack, aiming to identify potential mitigation strategies against such threats.

To simulate the resource exhaustion scenario on the SDN network's Data Plane, we deliberately triggered an escalation in the number of flow-rules installed in the switch. This was achieved by sending packets with spoofed IP addresses, thereby exploiting the switch's limited resources. Specifically, we focused on monitoring the impact on CPU utilization, storage memory, latency, and throughput within the switch. Detailed findings were presented in the form of tables, accompanied by graphical representations to visually illustrate the effects of increasing flow rules on the switches. Furthermore, we explored potential mitigation measures by developing an application that actively monitors the flow rules on the Ryu controller, aiming to detect and counteract such resource-exhausting effects.

Keywords: Software Defined Networking, IP Spoofing, Flooding, DDoS Attacks, Data Plane

Acknowledgments

We give our sincerest thanks to our supervisor Dragos Ilie for his constant guidance and support throughout our thesis and correcting our mistakes along the way. We extend our thanks to our friends and family for providing us with motivation and encouragement to get through tough times and complete our thesis.

List of Tables

2.1	SDN attack planes and mitigation techniques	9
3.1	Differences between SDN and Traditional networks	11
3.2	Requirements and protection of resources in SDN Controller	17
3.3	DDoS attacks causes and mitigation	28
5.1	Latency (in milli seconds) with increasing Number of Flow-rules	39
5.2	Number of bytes in switches with increasing flow rules	40
5.3	CPU usage and memory usage of switch with increasing flow rules	46
5.4	Number of bytes in switches after mitigation	48
5.5	CPU usage and memory usage after mitigation	49
5.6	Latency after mitigation	50

List of Figures

3.1	Fundamentals of SDN	12
3.2	SDN Architecture	14
3.3	SDN Controller and its interfaces	16
3.4	SDN Switch	18
3.5	Structure of flow rule	20
3.6	Structure of OpenFlow Switch	23
3.7	OFPT_PACKET_IN and OFPT_PACKET_OUT	24
3.8	Adding of flowrules by OFPT_FLOW_MOD	25
4.1	Network is unable to connect	34
4.2	Start of Ryu controller	36
4.3	Links and Pingall from Mininet Console	36
4.4	SDN Architecture for TestBed	37
4.5	Flow-rules in SDN Switch 1	38
4.6	Count of flow-rules in Switch 1	38
4.7	Packets of IP spoofing in switch1-eth 1	38
5.1	Throughputs of 3 different switches	44
5.2	Increasing of flowrules in switches with time	45
5.3	Throughputs of 3 different switches after mitigation	52
5.4	Flowrules with increasing time after mitigation	53

List of Abbreviations

API Application Programming Interface

ARP Address Resolution Protocol

CPR Component Replication Resilient

CPU Central Processing Unit

DDoS Distributed Denial of Service

DoS Denial of Service

ICMP Internet Control Message Protocol

IP Internet Protocol

NBI Northbound Interface

NFV Network Function Virtualization

NOS Network Operating System

ONF Open Network Foundation

OS Operating System

OVS Open vSwitch

SBI Southbound Interface

SDN Software Defined Networking

SYN Synchronize

TCP Transmission Control Protocol

UDP User Datagram Protocol

VM Virtual Machine

Contents

Abstract	i
Acknowledgments	ii
1 Introduction	2
1.1 Security Attacks in SDN	2
1.2 Research Gap	3
1.3 Motivation	3
1.4 Aim and Objectives	5
1.5 Research Questions	5
1.6 Outline	5
2 Methodology and Related Work	6
2.1 Methodology	6
2.1.1 Literature Review	6
2.2 Analysis of various papers	7
3 Concepts and Theory	10
3.1 History and evolution of Software Defined Networks	10
3.2 Software Defined Networks and its Fundamental Features	11
3.2.1 Plane separation	11
3.2.2 A simplified device and centralized control	12
3.2.3 Network Function Virtualization	12
3.2.4 Virtualization	13
3.2.5 Openness	13
3.3 SDN architecture	13
3.3.1 SDN Layers	13
3.3.2 SDN Controller	15
3.3.3 SDN switches	17
3.3.4 SDN applications	20
3.4 Openflow	22
3.5 Challenges in Software-Defined Networking	25
3.6 DDoS Attacks on SDN Networks	26
3.6.1 Effects on SDN with DDoS attacks	27
3.6.2 IP spoofing Attacks	27

4	Method	30
4.1	Test Environmet	30
4.2	Mininet emulation tool	30
4.3	Tools and Technologies	31
4.3.1	VirtualBox	31
4.3.2	RyU Controller	32
4.3.3	Open vSwitch Kernel	32
4.3.4	Scapy	32
4.3.5	PuTTY	32
4.3.6	Wireshark	32
4.3.7	Xming	33
4.4	Testbed	33
4.4.1	Requirements	33
4.4.2	Experimental Setup for Experiments	33
5	Results and Analysis	39
5.1	Experiment 1	39
5.1.1	Results and Analysis for Experiment 1	41
5.2	Experiment 2	45
5.2.1	Results and Analysis for Experiment 2	45
5.3	Experiment 3	48
5.3.1	Results and Analysis for Experiment 3	49
6	Discussion	56
7	Conclusions and Future Work	59
7.1	Conclusion	59
7.2	Future Work	60
	References	61
A	Supplemental Information	70

1.1 Security Attacks in SDN

Software-Defined Networking (SDN) is a network architecture that facilitates network virtualization, centralizes network configuration and management, and separates the control plane from the data plane. In this architecture, the data plane comprises of switches and routers responsible for forwarding packets to their intended destinations, while the control plane consists of a centralized control application that interacts with SDN switches using standardized protocols like OpenFlow, which allows for dynamic control over network flows. The control application can be deployed on a server or a virtual machine, providing a programmable interface for centralized control and management of network traffic flows. SDN offers a scalable and flexible network architecture that simplifies network configuration and management, optimizing network performance and efficiency [1].

The Open Networking Foundation (ONF) has defined a three-layer SDN architecture, starting with the Infrastructure layer, also called the Data Layer, which is responsible for forwarding network traffic through routers and switches. In this layer, the SDN controller controls the forwarding plane devices through the southbound interface (SBI), sending flow entries that instruct the infrastructure devices to apply rules to ingress traffic. The Control Plane, the second layer in the SDN architecture, is the central entity that comprises the SDN controller. It has a comprehensive view of the network and makes logical decisions based on network-wide information. The third layer, known as the Application Layer, includes various network applications such as flow monitoring, load balancing, and traffic engineering. These applications interact with the SDN controller through the northbound interface (NBI), which provides a standard interface for network applications to communicate with the controller [2].

The application plane contain the programs that communicate the desired network requirements to the SDN controller via the northbound interface (NBI), and the SDN controller then performs the requested functions by installing flow entries in the forwarding plane. The forwarding plane implements the flow entries received from the controller and enforces these actions at high speed. This approach promotes research and innovation while allowing network administrators to configure, manage, operate, and maintain large-scale networks using computer programs [3]. SDN-based cloud is a new form of cloud-computing environment in which SDN technology is used

to gain control of network architecture and provide networking as a service (NaaS). Since the fundamental role of cloud computing is to deliver on-demand services on various levels, availability is one of the most important security needs [4].

The SDN architecture presents numerous security benefits, such as the ability to transmit traffic analysis and anomaly detection data to the central controller. The controller, with its complete network view, can then analyze and correlate this information, creating new security policies to mitigate potential network attacks.

The centralized nature of the SDN controller makes it a prime target for attackers to launch DDoS attacks, which could result in severe network outages. Moreover, the programmability of the network elements allows attackers to inject malicious code or attack the network through legitimate programming interfaces, leading to trust issues in the SDN architecture.

Additionally, the lack of standardized best practices specific to SDN functions and components presents a challenge for securing the network. This can lead to vulnerabilities in the SDN network, as security policies that worked in traditional networks may not be effective in the SDN environment [5].

1.2 Research Gap

Our thesis focuses on the impact of IP spoofing on the SDN Data Layer that aims to exhaust the CPU utilization and storage of a switch in the Data Plane. Our extensive literature review revealed a lack of research on this topic, which prompted us to design a simulation, using Mininet, to study the behavior of the SDN network when subjected to such attacks. By injecting spoofed IP packets into the switches, we aim to exhaust their storage and CPU resources and investigate the effectiveness of the SDN network in mitigating this type of attack. Our simulation aims to fill the research gap identified in the Literature Review section of our thesis. We believe that our findings will contribute to the advancement of knowledge in the field of SDN security.

1.3 Motivation

Our choice to investigate security issues in SDN was motivated by the observation that full deployment of SDN is still limited in practice. This prompted our interest in exploring the specific security challenges and attacks that can arise in SDN networks.

The various factors that may contribute to security attacks in an SDN networks are:

- **Unauthorized Access** – occurs in application layer, control layer and data layers

- **Data Leakage** – occurs in both control layer and data layer
- **Data Modification** - occurs in application layer, control layer and data layer.
- **Denial of Service** – occurs in both control layer and data layer
- **Incorrect configuration** – occurs in application layer, control layer and data layer

Above factors can result in attacks occurring in different layers of the SDN architecture. According to the paper [6], each of above factors lead to attacks that can potentially occur in the control layer, which is considered to be highly vulnerable to such attacks. Data layer is also vulnerable to attacks but less when compared to the control layer and the research done on security attacks on data layer is less when compared to attacks on control layer, which prompted us to further investigate on attacks that occur in the data layer. There are many methods that can be used to prevent attacks both on data layer and control layer.

Our research interest lies in studying the security vulnerabilities of Software-Defined Networking (SDN), particularly in the context of Denial-of-Service (DoS) attacks on the Data Plane. A DoS attack can be initiated by flooding the SDN switch with a large number of packets, rendering the switch unusable and ultimately disrupting a part of the SDN network. Therefore, our focus in this thesis is investigating the effects of flooding attacks on SDN switches and the potential impact on network performance.

The SDN network is vulnerable to various forms of flooding attacks, including but not limited to UDP flooding, ICMP flooding, and SYN flooding. An ICMP flooding attack, for instance, involves the transmission of a large volume of ICMP echo requests to exhaust the resources of a network device. Given that the switch is an entry point for attackers to launch an attack, it is often the target of flooding attacks in the SDN network [7].

During the ICMP flood, each ICMP echo request triggers an insertion of an entry into the flow tables of SDN data plane devices and if there are a large number of these requests, the flow table gets overloaded rendering the switch inoperable.

Since the flooding attack tries to send a large number of packets to a network device in order to disrupt the working of said network device, monitoring the flow entries and checking if it crosses a limit, where the switch would stop working, is one of the ways how flooding attacks can be mitigated [8].

To ensure network security and stop hostile actions like data breaches, network failures, and unauthorized access to network resources, spoofing attack mitigation in SDN is a crucial responsibility. It is often not possible to totally prevent spoofing attacks, despite the fact that prevention techniques are useful in lowering the risk of such assaults. This is because attackers always develop new and cutting-edge methods to get around security precautions that are deployed in a network.

However, we can reduce the harm that is done by these attacks and make sure that the network is still functioning, by mitigating spoofing attacks in SDN. The use of access control mechanisms that confirm the validity of network communication is one of the mitigation approaches, as is the detection and isolation of spoofed traffic [8].

1.4 Aim and Objectives

The main focus of our thesis is to analyse how the SDN network responds when we increase the flow rules of ICMP packets with spoofed IP addresses and to see how the resources of the SDN switch gets affected.

- **Objective 1:** Determine how flooding of the flow-rules of an SDN switch affects the forwarding traffic in the network.
- **Objective 2:** Determine how storage and CPU utilization of a switch is affected with increasing flow-rules.
- **Objective 3:** Determine how monitoring applications can be used to mitigate the effect of IP Spoofing attacks on SDN switches.

1.5 Research Questions

We have formulated the following research questions which help us to attain the objectives that we have mentioned.

- **RQ1:** How does increasing flow-rules of an SDN switch affect the latency and throughput of switches?
- **RQ2:** How does increasing flow-rules of an SDN switch affect the memory and CPU utilization of the switch?
- **RQ3:** How can we mitigate the effect of IP spoofing attacks on SDN switches?

1.6 Outline

The present thesis is organized into distinct sections, each of which addresses specific aspects of the research topic. In Section 2, a comprehensive literature review was conducted to investigate the existing research on the subject matter, with the aim of identifying research gaps and deficiencies. Section 3 provides an overview of the underlying concepts and technologies relevant to the thesis, including Software-Defined Networking (SDN), Distributed Denial-of-Service (DDoS) attacks, and related tools and technologies. In Section 4, we describe the testbed and experimental setup employed in the study. Section 5 presents the results of the experiments conducted, along with a detailed discussion of the methods employed and the outcomes achieved. Section 6 provides an explanation regarding how the attackers were able to infiltrate the SDN network in our attack scenario and the limitations that our thesis has in its research. Finally, in Section 7, we conclude the thesis with a summary of the findings, a discussion of their implications, and suggestions for future research directions.

Chapter 2

Methodology and Related Work

2.1 Methodology

The methodology used to conduct our thesis consists of Literature review and Experimental Methodology. The Literature review is used to analyse various papers to find research gaps in the topic of IP spoofing attacks in SDN networks and Experiments are used to evaluate the research gaps found in the literature review. This methodology involves various steps:

- Literature Review : Conducting a literature review to identify the research gap for flow rules flooding in Data Plane and effect on network metrics and storage, with increase in flow rules in Data Plane due to DDoS attacks.
- Research Question and Objectives: Formulation of research questions and objectives that address the research gap.
- Experiment Design: Designing an experimental setup to test the research questions, that involves testbed, attack scenario, implementation and monitoring solution for the attack.
- Data Collection: Collection of data through implementation before and after monitoring solution.
- Results and Conclusion: Presenting the collected data with required graphs, analysing the the data collected to draw a conclusion and future work for further research on SDN security.

2.1.1 Literature Review

The Literature review explains about the keywords used for searching state-of-the-art research, Inclusion and Exclusion criteria, Analysis of papers. The goal of the literature evaluation is to determine the state-of-the-art research on security of SDN for various DDoS attacks . Academic articles, industry reports, and other pertinent publications are included in the review.

Keywords used: SDN, Security issues, DoS Attacks, IP spoofing, Flooding Attacks in SDN, DDoS Attacks.

We used these keywords in various databases which include IEEEExplore, ACM Digital Library, Google Scholar and Science Direct and have chosen papers that we felt were relevant to our research work.

2.1.1.1 Inclusion and Exclusion criteria

The inclusion and exclusion criteria determine which studies or articles to include and which articles are excluded.

Inclusion Criterion:

- The paper must be written in English.
- The paper must include the security attacks that can happen in SDN and the how the attack affects the SDN network.
- The paper must be in the year range of 2013-2022.

Exclusion Criterion:

- Paper must not be in a language other than English.
- Papers which were released before the year 2013.
- Papers that have no relevance to our research topic.

2.2 Analysis of various papers

The authors of these papers [1], [9], [6], [10] discuss regarding the various security attacks that can happen in an SDN network, and what part of the system each of them affect and various suggestions for reducing the security attacks that could happen in an SDN network. They also discuss the main problem, as to why SDN networks are not fully deployed as of right now. The papers emphasize on the vulnerabilities in SDN networks, which makes them prone to DoS attacks. The paper [11] discusses regarding the various types of security attacks that can occur in an SDN network and how they affect the SDN network. It also discusses about the defence mechanisms and mitigation for the security attacks.

The paper [12] discusses the impact of DDoS attacks on different types of controller sets like Floodlight, Ryu, POX etc. The authors do a performance analysis of the available controllers and see which of them are likely to be more secure when a DDoS attack occurs. The performance analysis was done with the help of a VM ware Ubuntu machine, Mininet and sFlow-RT. The comparative results were provided in the form of graphs which were generated with the help of Gnuplot.

The paper [13] compares an OpenFlow network and a traditional network with the help of Mininet network emulator. The authors checked the throughput of both OpenFlow and traditional networks and compared which one is better. They concluded that the initial pings took longer in an OpenFlow network since there were no flow table rules that were established, and after the flows were established the performance was similar to traditional networks.

The paper [14] proposes an architecture for anomaly detection and mitigation in SDN environments. The authors used two different protocols for traffic data gathering, which are, native OpenFlow method and s-Flow method in order to check

which allows more scalability. They were able to successfully detect and mitigate the anomalies in the SDN environment and were able to handle large amount of traffic.

The paper [4] describes how DDoS attacks can occur in an SDN based cloud environment and how they disrupt the cloud environment. The authors describe that the main problem that DDoS attacks bring to the cloud environment is the availability aspect of the system. It looks into how the DDoS attacks have evolved over the years and describe the various defence mechanisms that can be used against DDoS attacks.

The authors of the paper [15] discuss about the simulation of a Denial of Service Attack in an SDN network with the help of an INET-based attack simulation framework called SEA++. They used the Attack Specification Language (ASL) and the Attack Simulation Engine (ASE), that is provided by the SEA++ tool. Whatever was present in the Attack description file was converted into an XML file and added to INET during its initialization. This XML file gets parsed and ASE starts its execution and then creates the network with the requirements that are provided. This network is then analyzed and graphs are plotted, which are shown in the paper. These graphs show the comparison between the network when there is no attack and when there is an attack that occurred.

PayLess is a monitoring system that provides a cheaper way to try testing out different types of OpenFlow switches like NOX and Floodlight. This can be used to test out various architectures and see which type of OpenFlow switch can work in a particular architecture [16].

The authors of the paper [17] compare the performance of various types of controllers like Floodlight, POX etc, when they are under DoS attacks and determine which of these controllers are more resilient to DoS attacks. The authors of the paper [18] discuss how SDN can act as a forensic system that helps with attacks that occur in a data center. The SDN elements provide a global view of the data center and help to see exactly where the attack has happened by checking the log of the switches.

The authors of the paper [19] have tested various monitoring systems that were proposed with the help of SEA++ simulation module and have seen which of them was the most efficient monitoring system. The monitoring system is tested by seeing how this system behaves when under attack and is compared with other monitoring systems to see which of the monitoring systems can work more efficiently when under an attack.

The authors of the paper [20] discuss about attacking OpenFlow switches due to the openness of these switches, that is, when a new packet arrives at the switch it gets added into the flow table of the switch and hence if the attacker sends multiple new packets, then the flow tables can be flooded. A Sandbox based Defence mechanism against such attacks has been introduced in the paper and has been proven effective against DoS attacks on the OpenFlow switches.

The authors of the paper [21] describe a Hybrid of Software Defined Networking and Content Delivery Networks to provide services of Content Providers in an easier and efficient way. An SDN controller and some SDN switches are used along with Application Layer Traffic Optimization (ALTO) servers in order to provide easier way

for the Content Providers to broadcast their services, while also providing defense against IP spoofing attacks at the same time.

S.no.	Paper	Plane of Attack	Mitigation technique
1.	Carval Et al [22].	It targets a server.	Detection of DDoS attacks using DataPlane-ML algorithm
2.	Deng Et al [23]	Data Plane	A PacketChecker is deployed in the switches.
3.	Bahaa Et al [20]	Data Plane	A sandbox protection module.
4.	Shang Et al [8]	Control Plane	FLoodDefender module.
5.	Ma Et al. [24]	Control Plane and Data Plane	Using multiple controllers.
6.	Piedrahita Et al [25]	Control Plane	A congestion avoidance system called FLOWFence.
7.	Lim Et al [26]	Control Plane and Data Plane	A scheduling based method.
8.	Abdelsalam Et al [27]	Data Plane	ARP packet monitoring at the ports.
9.	Lim Et al [28]	Application Plane	By adding a Defence mechanism application to the POX controller.
10.	Tri ET al. [29]	Control Plane and Data Plane	None mentioned.
11.	Kumar Et al [30]	Targets Network Services	SAFETY algorithm is proposed.
12.	Ambrosin Et al [31]	Control Plane	LineSwitch module.
13.	Shin Et al. [32]	Control Plane and Data Plane	AVANT-GUARD framework.

Table 2.1: SDN attack planes and mitigation techniques

3.1 History and evolution of Software Defined Networks

Networking technologies have evolved significantly with development of new protocols. The foundation of conventional networking models uses the idea of dispersed control, whereby network nodes independently decided which data to forward based on predefined routing protocols. The inability to manage network traffic dynamically, the difficulty of establishing new network services, and the lack of network visibility and control are disadvantages of using predefined routing protocols.

SDN is a new method of networking, made possible by developments in software and hardware. Since, SDN separates control plane from the data plane, network behavior can be centralized and traffic can be managed dynamically. Compared to conventional networking models, this strategy is more adaptable and scalable, enabling the deployment of new network services and network adaption. The major features and advantages of SDN include enhanced network control, flexibility, and scalability. SDN enables dynamic traffic control, which can enhance network performance and reduce congestion. Additionally, SDN makes it possible to deploy new network services, which facilitates simpler network condition adaptation [33].

- **Network Management:** Network administrators may administer the network more effectively, with fewer mistakes, and with less downtime by separating the control plane and data plane.
- **Scalability:** SDN networks are more scalable than traditional networks since it is simpler to add or delete network devices as necessary thanks to the separation of the control plane and the data plane.
- **Flexibility:** By separating the control and data planes, it is simpler to add new network policies and protocols into place, because changes made to the control plane don't affect the data plane [34].

SDN has a lot of advantages, but it also creates new security difficulties. For instance, the control plane is a single point of failure and a potential target for attacks because it is centralized. Additionally, new attack vectors that weren't present in conventional networking topologies are created by the separation of the control and data planes. Researchers have suggested a number of secure SDN implementation

strategies, such as secure controller design, flow rule verification, and intrusion detection, to overcome these issues [35]. In the table below we give a brief understanding about the differences between traditional networks and SDN networks.

Software Defined Networking (SDN)	Traditional Networks
1) SDN is made up of network components that are made from software.	Traditional networks are made up of network components that are available as hardware.
2) SDN has separate layers in its architecture [36].	Traditional networks do not have separate layers [36].
3) SDN provides a centralized control plane.	Traditional networks do not provide centralized control.
4) SDN networks are more flexible as software is used for its configuration.	Traditional networks are not flexible as only hardware components are used .
5) SDN networks are more scalable since it is simpler to add or delete network devices [34].	Traditional networks are less scalable since the network devices are hardware and they become costly with increase in complexity [34].
6) It is easier to maintain SDN networks since it is software based.	It is tough to maintain traditional networks since hardware needs replacement.

Table 3.1: Differences between SDN and Traditional networks

3.2 Software Defined Networks and its Fundamental Features

3.2.1 Plane separation

In SDN, ‘plane separation’ is defined as division of a network’s control plane and data plane. By separating the two planes, SDN overcomes the difficulty of managing a network, which is a lot more difficult in conventional networks and enables more effective network administration and control.

The SDN control plane is in charge of managing and controlling the network, including traffic routing, network topology, and network policies. Contrarily, the data plane is in charge of sending traffic through the network in accordance with commands from the control plane [33,37]. The image in Figure 3.1 depicts how each layer communicates with each other and the devices used in each layer. Figure 3.1 is derived from [38].

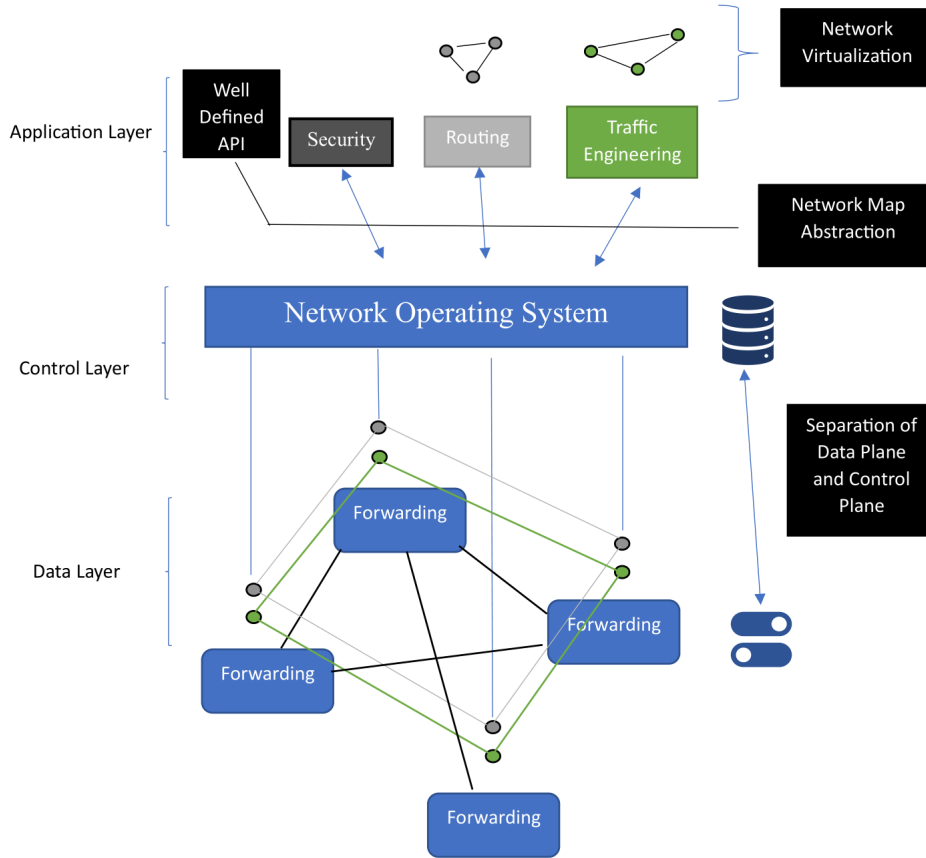


Figure 3.1: Fundamentals of SDN

3.2.2 A simplified device and centralized control

In SDN, the term ‘simplified device’ refers to the simplification of network hardware like switches and routers. Because the control and data planes are separated in the SDN architecture, the network devices, like switches, can be virtualized, making the devices more scalable compared to traditional networks. This can be observed from Figure 3.1 which clearly depicts the separation of control and data planes.

The capability of managing the network from a single location is called ‘centralized control’. The control logic of the network is shifted to a central controller after the control plane and data plane have been split apart. The forwarding rules and policies are configured by this controller through communication with the network devices, giving the network administrator more oversight and control [37].

3.2.3 Network Function Virtualization

The basics of SDN can be derived precisely from the abstractions of distributed state, forwarding, and configuration. The distributed state abstraction gives network programmers a global view of the network. The required forwarding behaviors can be specified by the programmer using the forwarding abstraction without having any knowledge of vendor-specific hardware [39].

The ability to virtualize the network and separate the network service from the

underlying physical network is one of the outcomes of this level of abstraction. "Network Function Virtualization (NFV) is the ability to use virtualization technologies to abstract and virtualize network functions, including routers, firewalls, load balancers, and various other network services. Instead of using specialized hardware equipment, NFV enables network functions to be executed as software instances in a virtual machine (VM), on one or more hosts. In data centers for network nodes and end user premises, NFV gives operators the ability to combine a variety of different types of network equipment into high-volume switches, servers, and storage [40].

3.2.4 Virtualization

Virtualization is a crucial technology for cloud computing, SDN, and NFV. The technology makes it possible to create a scalable, dynamic, and automated programmable virtual network and software-defined network. In integrated cloud platforms like TeleCom clouds, network functions and, virtual network infrastructures are present which require virtualization. The use of multiplexing, aggregation, or emulation to simulate the interface to a physical object is known as virtualization. It creates a virtual object through aggregation out of multiple physical objects, and it creates a virtual object through emulation out of a different kind of physical object [41].

3.2.5 Openness

One of the features of SDN is that its interfaces continue to have open standards with thorough documentation. The defined APIs should provide software with enough control to experiment with and manage different control plane options. The idea is to keep both the northbound and southbound interfaces of the SDN controller open, which enables the research of cutting-edge and novel network operating techniques. This capability can be used by businesses and research institutions to quickly test new theories. Organizations are working together to solve today's network issues, so the rate at which network technology will develop is significantly accelerated. As a result, networks' structure and functionality are improving.

3.3 SDN architecture

Three layers make up the architecture of SDN. We must understand these SDN layers, which include Infrastructure, control and application layer. These need to be thoroughly studied in order to understand how SDN functions. The image in Figure 3.2 has clearly depicted the various layers in the SDN architecture and the components of each layer. The Figure 3.2 has been derived from [42].

3.3.1 SDN Layers

A reference model for SDN was proposed by Open Network Foundation (ONF). This model has three layers: the application layer, the control layer, and the infrastructure layer.

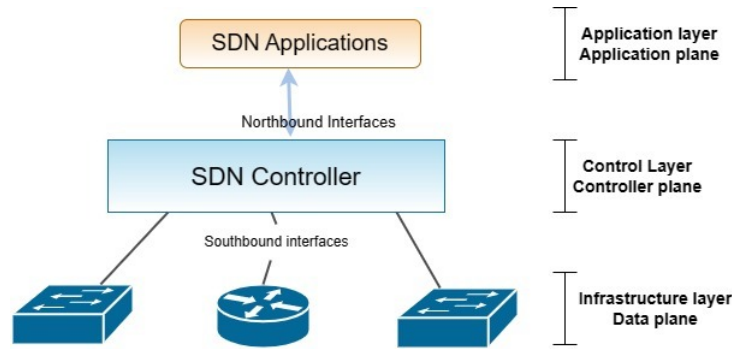


Figure 3.2: SDN Architecture

- **Infrastructure Layer**

Infrastructure layer is a representation of the actual networking hardware and the parts that connect it. The SDN reference model classifies it as the lowest layer in the SDN architecture. SDN architecture needs to support all of the transmission media such as wired, wireless, and optical [43].

In an SDN switch, there are two logical parts: the data plane and the control plane. According to packet forwarding rules, the data plane performs packet forwarding through its processor. The commands issued by the control layer are the forwarding rules. In order to obtain rules at the switching layer and data-link layer like packet forwarding and link turning, the control plane in switching devices communicates with the controller. To act in the same way in the future under similar circumstances, the rules are stored in memory at the control plane.

According to hardware specifications, SDN switching devices can be divided into two main categories. SDN switches operating on host operating system like Linux are the switches on general hardware [44]. Building SDN switches on vendor-neutral and programmable platform for building networks is provided by open network hardware.

- **Control Layer**

The SDN control layer serves as a link between the Infrastructure layer and the Application layer. This section focuses on two crucial control layer issues, namely rules and policy validation and performance issues with potential control layer solutions.

The logical design of SDN controllers can be broken down into three building blocks: a high-level language, a rule update process, a network status collection and synchronization process.

1. **High Level Language:** The translation of application requirements into packet forwarding rules is one of the controller's primary tasks. To do this task adopting some common configuration languages like Command Line Interface (CLI) is a simple strategy. They are typically insufficient to accommodate dynamic and stateful network status. As a result, it is essential to give SDN applications a high-level language for interacting with controllers.

2. Updating the Rules: SDN controller is in charge of creating packet forwarding rules that describe the policies and installing them into the proper switching devices for use. There are two different definitions of consistency that are discussed in the literature of a controller. One is making use of either just the original rule set or just the updated rule set, this is called strict consistency. The other one is called Eventual Consistency which ensures that once the update procedure is complete, the subsequent packets will eventually use the updated rule set [45].
 3. Collection of Network Status: Controllers gather network status to create a comprehensive picture of the entire network and supply the application layer with the necessary data. Network status collection is done by local traffic statistics like switching devices which store gathered data [46]. The collected data is synchronized eventually with existing data.
- Application Layer
The application layer consists of software programs that run on top of the SDN controller in an SDN architecture and communicate with the network infrastructure via the control plane. These programs are in charge of supplying network services and putting network regulations into effect, in accordance with the needs of a business or a program. Through a northbound interface, SDN applications can easily access a global network view with real-time status. With this knowledge, SDN applications use a high-level language offered by the control layer to implement strategies to control the underlying physical networks [47]. More details about the application layer is discussed in the upcoming sections of our thesis.

The southbound API connects the top tier (control layer) with the bottom tier (infrastructure layer). Such API are used in networks for traffic analysis and monitoring, including OpenFlow, sFlow, and NetFlow [3]. Eastbound and westbound APIs are used for controller connections. Northbound API is used to describe the controller application interface which connects control and application plane. The policies are defined by the management application, and when they are translated into southbound instructions, they are used to program the behavior of the forwarding devices. Applications for network management, load balancing, etc. can be created using the northbound API. The northbound API doesn't have a widely used standard protocol. OpenFlow is the most popular and widely-accepted standard for southbound communication [33].

3.3.2 SDN Controller

Through the logically centralized control provided by a NOS, SDN facilitates network management and reduces the burden of solving networking issues [48]. The NOS can offer services such as device discovery, distribution of network configuration, network state and topology information, and other generic functionality. With NOSs, a developer can define network policies without having to worry about the finer points of how data is distributed among routing elements. For example, by lowering the inherent complexity of developing new network protocols and network applications,

such systems may be able to create a new environment capable of fostering innovation at a faster rate.

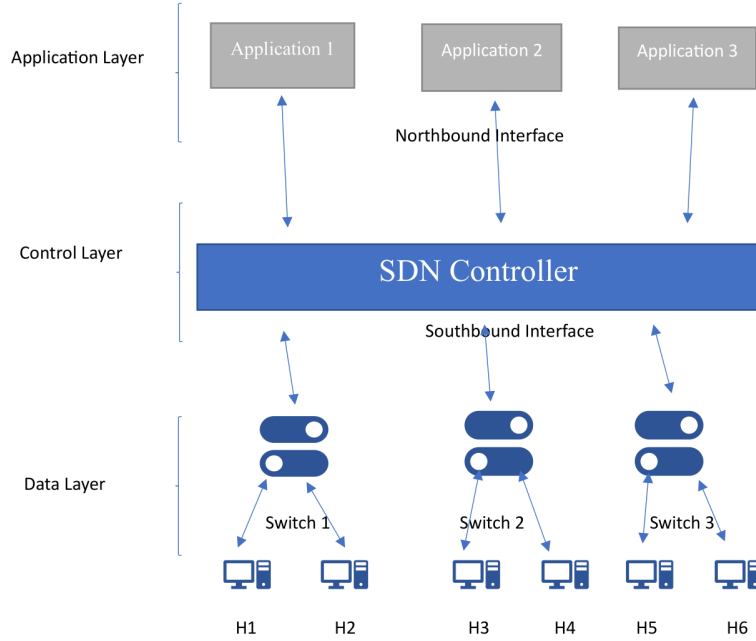


Figure 3.3: SDN Controller and its interfaces

The image in Figure 3.3 depicts how an SDN Controller communicates with the data plane and the application plane of the network. Figure 3.3 has been derived from [49].

- **Centralized and Distributed Control**

A centralized controller is a single entity that oversees all of the network's forwarding devices, which can be observed from Figure 3.3. A small network can be managed by a single controller, naturally, it has a single point of failure and might not be scalable. The management of a network with many data plane elements may not be possible with a single controller.

A distributed NOS can be scaled up to meet the needs of potentially any environment, from small- to large-scale networks, in contrast to a centralized design. A distributed controller could be a physically dispersed group of components or a central cluster of nodes. While the former can be more resilient to various logical and physical failures, the former can provide high throughput for very dense data centers. Fault tolerance is a further characteristic shared by distributed controllers. When a node fails, a neighboring node should take over its functions and equipment. Finally, a distributed controller can increase the control plane's scalability and resilience while minimizing various issues [50].

- **Controller architecture:**

In SDN architecture, the controller is the key element, and where the complex-

ity lies. We use a "divide-and-conquer" approach in this subsection to present a logical controller architecture. The architecture of SDN controller can be mostly discussed with two principles.

1. Object

SDN controller manages two different kinds of objects. One is used for network control, including application layer policies and infrastructure packet forwarding rules. The second concerns network monitoring and is presented as local and global network status. Thus, there are two information flows that are counter-directional in the logical architecture.

2. Interfaces

There are two interfaces on the SDN controller. Transactions with the infrastructure layer are handled by the south-bound interface, which is designated as the controller-infrastructure interface. Transactions with the application layer are handled by the north-bound interface, which is designated as the application-controller interface. This can be observed from Figure 3.1 which clearly shows the northbound and southbound interfaces.

The following tables shows the requirements and resources to be protected for securing controller against various attacks in SDN network [51].

Requirements	Resources to be protected
1.Remote Management in secure manner	1.traffic to and from the SDN Controller
2.Software Updates from trusted sources	2.Software of SDN Controller
3.Interfaces, VMs and required devices are from a trusted source	3.Stored data by Controller
4.Logical and Physical separation of control plane and data plane	4.Configuration and reboot data.
5.logical and Physical separation of control plane and application plane	5.Authentication credentials of Controller
6.Protect Flow tables and configurations between control plane and data plane	6.Northbound, Southbound, East and west interfaces, connections

Table 3.2: Requirements and protection of resources in SDN Controller

3.3.3 SDN switches

Switches in software-defined networking (SDN) are the fundamental component in the separation of the control and data planes, and are a key part of SDN design. SDN switches provide forwarding functionality in the data plane, which rely on the SDN controller for control plane intelligence. The flexible and programmable packet forwarding features of SDN switches, enable network managers to centrally control

the flow of network traffic, using software-defined flow rules. The flow rules specify a lot of things, such as, the source and destination IP addresses, ports, and protocols that determine how traffic is forwarded throughout the network. Network managers can dynamically modify network traffic flows in response to shifting network circumstances or application requirements thanks to the programmability of flow rules [33] [52]. The figure 3.4 depicts how an SDN switch looks like in an SDN network. This image is derived from [53].

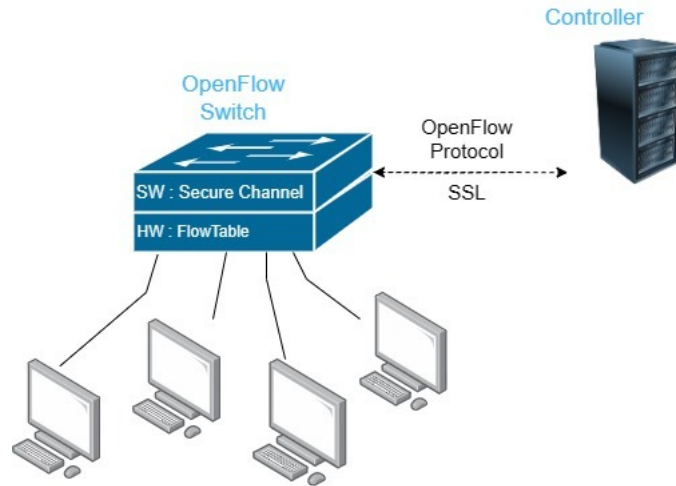


Figure 3.4: SDN Switch

SDN switches support multiple network protocols, offer high-performance packet processing, offer sophisticated packet filtering, and offer forwarding capability. Additionally, they give administrators the capacity to set up and administer network policies and security parameters, enabling them to impose access control and threat prevention systems. Since they are in charge of forwarding data packets within the network, in accordance with the instructions given by the SDN controller, SDN switches play a crucial part in SDN architecture. SDN switches offer a more adaptable and scalable network design, that can change with changing business requirements and network conditions, by separating the control plane and data plane [53].

3.3.3.1 SDN switch features

- **Packet Forwarding:** Forwarding packets from one network device to another is the main responsibility of an SDN switch. SDN switches, in contrast to conventional switches, base their forwarding choices on network policies established by the network administrator using flow tables. These policies are put into practice utilizing flow rules, which define how packets should be treated depending on their header data.
- **Packet Filtering:** SDN switches also have the ability to filter packets based on previously established rules. This enables network managers to restrict access to particular network resources or prohibit particular types of traffic. There are different levels, such as the physical port, the network layer, or the application layer, that can implement packet filtering [54].

- **Traffic Engineering:** SDN switches can also be used for traffic engineering, which entails streamlining network traffic to enhance the effectiveness and performance of the network, to avoid network congestion, which can be accomplished by balancing traffic across several network paths, giving specific categories of traffic priority, or rerouting traffic.
- **Quality of service:** Managing the quality of service (QoS) entails giving network traffic a priority based on its significance or criticality. QoS policies that prioritize traffic depending on various factors, such as, bandwidth, latency, or packet loss, can be implemented using SDN switches. This makes sure that crucial traffic like real-time applications gets precedence over less significant traffic.
- **Network Virtualization:** SDN switches can also offer network virtualization, which enables the construction of several virtual networks on top of a real network infrastructure. This offers greater flexibility and security by allowing network managers to control and isolate various network resources and services.

3.3.3.2 Flow rules in SDN Switches

Flow rules are an important part of software-defined networking (SDN) switches, allowing network managers to centrally govern network traffic flow. Flow rules play a crucial role in directing network traffic by defining various parameters, including source and destination IP addresses, ports, protocols, and more. These rules govern how data is forwarded within the network, ensuring efficient and reliable transmission between different network components. Network managers can dynamically modify network traffic flows in response to shifting network circumstances or application requirements. The SDN controller in an SDN architecture, programs flow rules into the tables of SDN switches. The direction that data packets should be routed over the network is specified by these flow rules. The SDN controller has the ability to alter and update the flow rules in real-time, enabling dynamic traffic management and network optimization [55]. OpenFlow, P4, and Python are just a few of the flow rule programming languages supported by SDN switches. The most extensively adopted and defined protocol for configuring flow rules in SDN switches is OpenFlow. In order to facilitate communication between the SDN controller and the SDN switches, it specifies a set of common messages and commands [56]. The figure 3.5 depicts how packet information is stored in the SDN switches. The figure 3.5 is derived from [57].

Features of Flowrules:

- **Matchfields:** The match fields are used to identify packets that match the flow rules, and the actions define how those packets should be handled. Different packet header fields, including source and destination IP addresses, protocol type, and port numbers, might be included in the match fields. Actions include sending packets to a particular port, discarding packets, changing packet headers, or sending packets to a controller for additional processing.
- **Priorities:** Priorities is a crucial component which govern the order in which flow rules are assessed. Rules with highest priorities are considered before rules

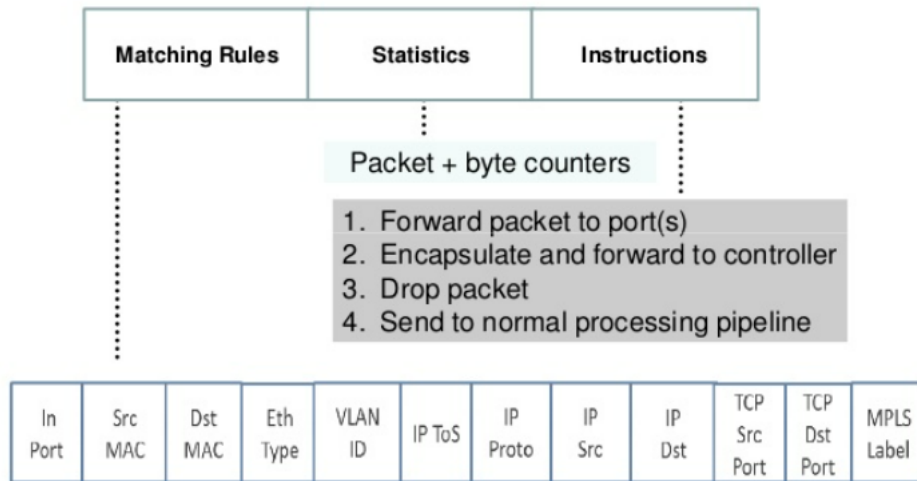


Figure 3.5: Structure of flow rule

with lower priorities in the flow. This enables network managers to enforce particular security standards or prioritize certain sorts of traffic.

- **Counters:** Counters can be used to keep track of the amount of packets and bytes that match a specific rule, they are another crucial component of flow rules. This data can be used to track network usage, spot congestion, and resolve problems within the network.
- **Timeouts:** Timeouts define how long a flow rule should be in effect, they are another crucial component of flow rules. A flow rule that has reached its expiration date is removed from the switch's flow table to make room for new flow rules. By doing this, the network is prevented from becoming overloaded with inactive flow rules [55].

3.3.4 SDN applications

The applications in Software-Defined Networking (SDN) are a crucial part of the entire SDN architecture. The Application layer is in charge of giving software tools to network managers so, they can control network traffic, enforce security rules, and keep track of network performance. The application plane, in this sense, can be thought of as the layer that allows network managers to communicate with the SDN architecture using a range of software programs and tools. These programs are made to offer a variety of functions, from simple network administration to sophisticated security and performance tracking [47] [56].

The SDN applications can be mostly compared to functions in layer 2 and layer 3 of OSI model. SDN applications can be viewed as the "brain" of the network. Applications send the control logic to data plane that will be converted into instructions and installed in the data plane, directing how the forwarding devices should behave. Traditional network functions like routing, load balancing, and security policy enforcement are carried out by SDN network applications [58].

Even though there are many different use cases, the majority of SDN applications

can be divided into one of these five categories: traffic engineering, mobility and wireless, measurement and monitoring, security and dependability, and data center networking.

- Traffic Engineering

There have been several traffic engineering applications proposed, that include techniques for flow management, fault tolerance, topology update, and traffic characterization, as well as optimization of rule placement, and etc [59] [60]. One of the initial SDN/OpenFlow applications, that was envisioned, was load balancing. For this, various algorithms and methods have been put forth. SDN load balancing makes it simpler to place network services throughout the network [61].

It is possible to achieve infrastructure goals such as latency, performance, and fault tolerance using modern techniques [62].

Avoiding or minimizing the impact of network bottlenecks on the functionality of the provided computing services is one of the key objectives of data center networks. For large-scale service providers, traffic optimization is another intriguing use case, where dynamic scale-out is necessary. The placement of rules can be optimized to increase network efficiency, according to recent research [63].

- Mobility and wireless

It is easier to deploy and manage various wireless network types, including WLANs and cellular networks with SDN-based approaches [64]. The provision of programmable and adaptable wireless network stack layers is one of the first steps toward realizing SDN features in wireless networks [65].

The Odin framework suggests using light virtual access points (LVAPs) as another intriguing method of enhancing wireless networks' management capabilities. It functions with current wireless hardware, unlike OpenRadio, and does not impose any changes to IEEE 802.11 standards [66].

Wireless networks that are extremely dense and heterogeneous have also been a target for SDN. Due to limitations like radio access network bottlenecks, control overhead, and high operational costs, these DenseNets are not without their drawbacks. To work around some of these restrictions, a dynamic two-tier SDN controller hierarchy can be implemented [67].

- Measurement and Monitoring

Solutions for measurement and monitoring can be categorized into two groups. First one is programs that add new functionality to networking services and second one to enhance OpenFlow-based SDN features [68].

The first class of applications include increasing broadband performance visibility [69], whereas, different types of sampling and estimation techniques are typically used in the second class of solutions to reduce the burden on the control plane, when it comes to gathering the data plane statistics [70]. Other initiatives of this second class are call for a strong lighter traffic analysis functions, such as detecting anomaly conditions which may occur due to security attacks.

Different methods for providing real-time, low-latency, and flexible monitoring capabilities to SDN are proposed by other monitoring frameworks, such as OpenSample and PayLess [71].

- Security and dependability

In the context of SDNs, a wide range of security and dependability proposals are emerging. There are essentially two approaches: one uses SDNs to increase network security, and the other increases SDN security [72] [14].

Some applications, like security policy enforcement prevent malicious activity from reaching the network's crucial areas. SDN has been successfully used for a variety of other tasks, including active security and the detection of distributed denial of service (DDoS) flooding attacks. OpenFlow forwarding devices make it simpler to gather various network data quickly, which is very useful for algorithms designed to detect DDoS flooding attacks [73] [74].

To prevent lower priority applications from overwriting rules generated by security applications, early approaches tried to use straightforward techniques like classifying applications and using rule prioritization [75]. There is still much work to be done in order to create reliable and secure SDN infrastructures.

- Data centre Networking

Most of the current IT systems and services, from small businesses to large cloud providers, are reliant on highly scalable and effective data centers. However, these infrastructures continue to present significant computing, storage, and networking challenges. Data centers in networking benefit in solving different problems [76], for instance, through network virtualization and orchestration with computing and storage.

Detecting unusual behaviors in network operation is one potential use of SDN in data centers. It is possible to continuously build signatures for applications, by passively capturing control traffic, and by employing various behavioral models and gathering the necessary data from components involved in the operation of a data center. The signature history can then be used to spot behavioral variations [77].

3.4 Openflow

OpenFlow is an open-source and a key technology in implementing software-defined networking (SDN). For managing networks, SDN seeks to centralize and streamline the management of network devices. A centralized controller uses OpenFlow to govern the forwarding behavior of the network. OpenFlow enables network abstraction through a controller that manages the flow tables and entries of network devices. Applications can communicate with the controller through an API, without needing to directly access the network devices. This allows the controller to handle the details of updating the network devices' flow tables, providing a high level of abstraction [53]. The image in Figure 3.6 shows the internal workings of an SDN switch it is derived from [78].

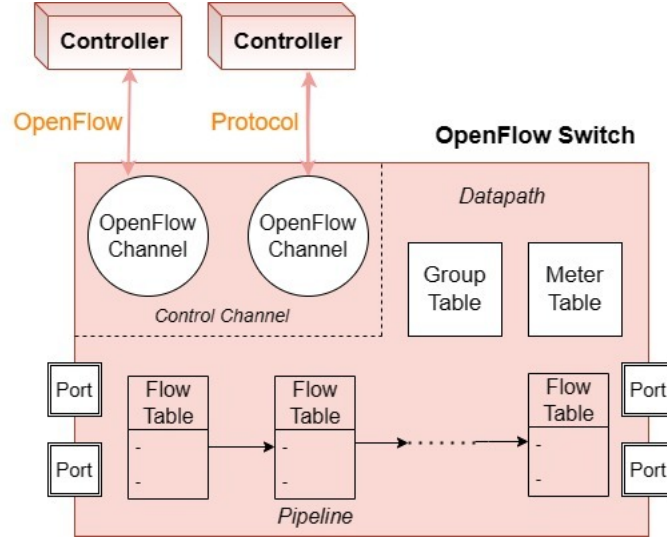


Figure 3.6: Structure of OpenFlow Switch

The Flowtable and the OpenFlow channels are the two essential parts of an OpenFlow switch as shown in figure 3.6. The figure 3.6 is derived from [79]. The set of guidelines that specify how the switch ought to manage incoming packets are specified in the Flowtable. The switch and controller communicate with each other through the OpenFlow interface. This port may be a virtual connection that links the switch and controller or it may be a real port on the switch [54].

The controller and switch communicate using OpenFlow messages. These messages include flow-mod, which the controller uses to add, change, or delete rules from the flow table, and packet-in, which the switch sends to the controller when it receives a packet that doesn't match any rules in the flow table. OpenFlow also includes a number of alternative message types that permit communication between the controller and switch in addition to packet-in and flow-mod messages. For instance, the controller can use the barrier message, to check that the switch has completed processing all previous messages before sending any new messages. Port messages are also sent using the Openflow messages, regarding status of ports, adding of new ports and down link of ports [80] [81].

There are three types of OpenFlow messages:

1. **Controller-to-Switch Messages:** To change the switch's behavior, the controller sends these signals. These messages, as an example, include:
 - **Flow-mod:** The switch's flow table can be modified, added to, or deleted using this message.
 - **Barrier-request:** This message is used to make sure that the switch has processed all previously sent messages.
 - **Queue-get-config-request:** This message is used to get switch configuration details for a certain queue.
2. **Asynchronous Messages:** These messages are sent to controller by the switch when certain events happen. These messages, as an example, include:

- Packet-in: When a packet is received by the switch and none of the rules in its flow table match it, this message is sent.
 - port-status: When a switch port's status changes, such as when a link is brought up or down, this message is issued.
 - Error: When a switch malfunctions, such as when a packet is dropped for lack of resources, this message is sent.
3. Symmetric Messages: These messages are used to test connectivity and latency between the controller and switch, they can be sent in either direction. These messages, as an example, include:
- Echo-request: The controller sends this message to the switch in order to check connectivity and solicit a response.
 - Echo-reply: When the controller issues an echo-request message, the switch responds by sending this message.

Wireshark is used to capture the packets and these packets are analysed. The figure 3.7 shows OpenFlow messages sent by a switch using OFPT_PACKET_IN to the controller for forwarding a packet received from source address 10.0.0.1 to 10.0.0.9.

No.	Time	Source	Destination	Protocol	Length	Info
66825	15.544843727	10.0.0.1	10.0.0.9	ICMP	100	Echo (ping) request id=0x05a2
66826	15.546157959	127.0.0.1	127.0.0.1	OpenFl	208	Type: OFPT_PACKET_IN
66828	15.548153826	127.0.0.1	127.0.0.1	OpenFl	206	Type: OFPT_PACKET_OUT

No.	Time	Source	Destination	Protocol	Length	Info
66826	15.546157959	127.0.0.1	127.0.0.1	OpenFl	208	Type: OFPT_PACKET_IN
66827	15.547926734	127.0.0.1	127.0.0.1	OpenFl	172	Type: OFPT_FLOW_MOD
66828	15.548153826	127.0.0.1	127.0.0.1	OpenFl	206	Type: OFPT_PACKET_OUT

Figure 3.7: OFPT_PACKET_IN and OFPT_PACKET_OUT

The controller sends back a message with OFPT_PACKET_OUT. At the same time, the controller also sends a message with OFPT_FLOW_MOD to the switch to add a flow rule in the SDN switches. The following figure 3.8 shows adding of the flowrules message.

No.	Time	Source	Destination	Protocol	Length	Info
66826	15.546157959	127.0.0.1	127.0.0.1	OpenFL...	268	Type: OFPT_PACKET_IN
66827	15.54722244	127.0.0.1	127.0.0.1	OpenFL...	272	Type: OFPT_PACKET_IN
66828	15.548153826	127.0.0.1	127.0.0.1	OpenFL...	266	Type: OFPT_PACKET_OUT

```

Transaction ID: 3674926293
Cookie: 0x0000000000000000
Cookie mask: 0x0000000000000000
Table ID: 0
Command: OFPT_ADD (0)
Idle timeout: 0
Hard timeout: 0
Priority: 1
Buffer ID: OFP_NO_BUFFER (4294967295)
Out port: 0
Out group: 0
Flags: 0x0000
Pad: 0000
Match
  Type: OFPMT_OXM (1)
  Length: 26
  OXM field
    OXM field
      Class: OFPXM_OPENFLOW_BASIC (0x8000)
      0001 011. = Field: OFPXM_OFB_IPV4_SRC (11)
      ....0 = Has mask: False
      Length: 4
      Value: 10.0.0.1
    OXM field
      Class: OFPXM_OPENFLOW_BASIC (0x8000)
      0001 100. = Field: OFPXM_OFB_IPV4_DST (12)
      ....0 = Has mask: False
      Length: 4
      Value: 10.0.0.9
  Pad: 000000000000
Instruction
  
```

Figure 3.8: Adding of flowrules by OFPT_FLOW_MOD

3.5 Challenges in Software-Defined Networking

The implementation of Software-Defined Networking indeed offers numerous advantages. However, it also presents some challenges at the moment [82]. Below are some of the challenges addressed in the implementation of Software-Defined Networking:

1. **Modeling and Performance of SDN:** In order to deploy SDN/OpenFlow, it is necessary to determine how many controllers are needed and where they should be placed inside a given network architecture. Desired reaction bounds, metric selection, and network topology are just a few of the variables that affect how many and where to place the best controllers. Choosing a proper Network Operating System (NOS) and assessing its performance are equally important. These factors must be taken into account in order to properly address the implementation-related issues of SDN.
2. **Resilience and Recovery of SDN:** The vulnerability of the network is one of the issues with the centralized control strategy advocated by OpenFlow. The total network resilience might be greatly impacted by the controller failing. Although backup controller configuration is possible with OpenFlow, there is no system in place to coordinate these controllers. The CPRecovery component offers a primary-backup technique with two phases: replication and recovery, in order to handle this. While the recovery phase turns the backup controller into the primary controller in a failure state, the replication phase enables frequent updates from the primary to the backup controller.

3. **Security Risks:** Due to the lack of defined certificate formats for data integrity in the OpenFlow specification, the existing Internet architecture has difficulty supporting security features. Advanced authentication and encryption techniques are needed for SDN security. Interoperability problems and vulnerabilities can be introduced through key exchange between controllers. Without mentioning an interoperable variant, the OpenFlow definition suggests plain TCP or optional TLS sessions for secure channels [83]. Different network designs present security issues since the mentioned defenses might not be adequate to thwart assaults on the control and data planes such as eavesdropping, controller impersonation, and various SDN attacks [84].
4. **Integration with legacy networks:** Another major obstacle to the introduction of SDN is integrating with older networks. Because they lack OpenFlow functionality, legacy networks need for careful coordination with new SDN hardware. Although equipment can be upgraded or replaced to accommodate OpenFlow, such network re-engineering might be expensive. Therefore, maintaining the entire functionality and compatibility of the network depends on guaranteeing smooth coordination between SDN hardware and older infrastructure.

3.6 DDoS Attacks on SDN Networks

A Denial of Service (DoS) attack is a type of cyber attack in which the attacker attempts to make a website or network unavailable to its intended users by flooding it with traffic, or sending a large number of requests. In an SDN environment, a DoS attack can be launched by overwhelming the controller or the network devices with an enormous amount of traffic, making them unable to process legitimate requests.

A Distributed Denial of Service (DDoS) attack is a type of DoS attack that is launched from multiple sources simultaneously. The goal of a DDoS attack is to bring down the targeted website or network by flooding it with traffic from multiple sources, making it nearly impossible to defend against. In an SDN environment, a DDoS attack can be launched by using compromised devices to send a large number of requests to the controller or network devices, overwhelming their processing capabilities [85].

SDN architecture can be particularly vulnerable to DDoS attacks because of their centralized control plane. If the controller is overwhelmed by a DDoS attack, it can cause a network-wide outage, making the entire network unavailable to its intended users. To mitigate DDoS attacks, SDN controllers can employ techniques such as traffic filtering and rate limiting to identify and drop malicious traffic.

DoS and DDoS attacks pose a significant threat to SDN environments, and SDN controllers must employ proper security measures to detect and mitigate these attacks. As the number of IoT devices and network-connected devices continues to increase, the risk of DDoS attacks in SDN environments is expected to grow. Therefore, it is crucial to stay up-to-date with the latest security measures to keep SDN networks secure. A DDoS attack in SDN networks usually tends to either flood the network utilization of network elements (such as a controller or the switches), control

plane and flow tables of the switches [86] [87].

3.6.1 Effects on SDN with DDoS attacks

The vulnerability of SDN to DDoS attacks is attributed to its architecture, which comprises three functional layers that are vertically separated. As such, SDN presents a targeted option for DDoS attacks, with each of its layers susceptible to such attacks. These vulnerabilities are succinctly outlined as follows:

- **Buffer Saturation:** When a new packet arrives, the switch matches it with flow table entries. If no match is found, the switch requests the controller to create a new flow rule. While waiting for the rule, the switch buffers part of the packet and sends another part to the controller. If the buffer memory becomes saturated, the switch sends the entire packet to the controller. Buffered packets are either processed by the controller or automatically expired after a certain period via a `packet_out` or `Flow-mod` message [88].
- **Flowtable overflow:** Flowtable overflow is a prevalent vulnerability in SDN switches that can be exploited for DDoS attacks. Malicious actors can flood the switch with an excessive number of requests, causing the flowtable to become saturated quickly and hindering the processing of legitimate traffic. Once the flowtable reaches its limit, the switch's ability to forward packets efficiently is compromised, leading to network congestion and potentially causing a network outage [86].
- **Controller Saturation:** When an attacker inundates a network with fake packets, the switches in the network may send all requests to the controller for processing. As a result, the controller becomes overwhelmed with the processing of fake requests, leading to a significant drain on its processing power and throughput capacity. This can compromise the controller's ability to process legitimate traffic and, in turn, cause network congestion or even network failure.
- **Southbound APIs congestion:** Congestion in the Southbound APIs is a significant challenge for SDN-based networks. It can occur when an OpenFlow switch sends a `packet_in` request to the controller. This `packet_in` message includes the header of the original packet and a buffer ID that points to the stored data part of the packet in the switch's buffer. When the buffer is full the switch will send the entire packet to the controller. An attacker can exploit this vulnerability by sending multiple fake flows to the switch, which can cause congestion in the southbound APIs and make them unavailable to legitimate users [87].

The following table shows the plane of attack of DDoS attacks, the cause of the attack and the counter measures. The table has been derived from the cited paper [89]

3.6.2 IP spoofing Attacks

DDoS attacks typically rely on a large number of compromised devices, forming a botnet, to flood the target network with traffic. IP spoofing is a crucial technique

DDoS Attack Plane	Causes of Attack	Mitigation of Attack
Control Plane	1. Centralized nature. 2. Limited capacity of flow rule processing 3. Large number of flows.	1. Replicating the controller. 2. Dynamic allotment of controllers. 3. Placing the controller and handling larger flows efficiently
Data Plane	1. Capacity of flow table is limited. 2. Large number of flows in switches. 3. Buffer capacity of a switch is limited.	1. Removing expired rules. 2. Aggregation of Rules in the switches. 3. Increasing Buffer capacity of switches. 4. Decreasing communication delay between controller and switches.

Table 3.3: DDoS attacks causes and mitigation

used by attackers to obfuscate their true identities and make it challenging to trace the source of the attack. By spoofing the source IP addresses in the attack traffic, attackers can make it appear as if the traffic is originating from legitimate sources, thereby evading detection and mitigation efforts [90] [91]. Here are a few common IP spoofing attacks in SDN:

- **Distributed Denial of Service (DDoS) Spoofing:** In a DDoS spoofing attack, multiple compromised devices (often forming a botnet) send a flood of packets to a target network, each packet containing a spoofed source IP address. This makes it difficult for the target network to differentiate legitimate traffic from malicious traffic and can overwhelm network resources
- **Blind Spoofing:** In this type of attack, the attacker sends packets with a spoofed source IP address to a target network without expecting any response. The attacker simply aims to deceive the target network and potentially bypass security measures.
- **Non-Blind Spoofing:** In this attack, the attacker sends packets with a spoofed source IP address and expects a response from the target network. The attacker aims to exploit vulnerabilities in the target network or launch more sophisticated attacks by establishing a communication channel.
- **Man-in-the-middle (MitM) attacks:** The attacker uses a third party's IP address to intercept network traffic between two parties. The attacker can then intercept the conversation or change the information being sent [92].

In this thesis experiment, we employed a distributed spoofing technique combined with blind spoofing. This involved a group of attackers sending spoofed packets with

random source IP, selected randomly to various hosts within the network. Our focus was on a specific scenario where we conducted a flood attack on the SDN switches in the Data Plane using IP spoofed ICMP packets. The purpose was to observe and analyze the impact of such an attack on the network. We assume attackers gains unauthorized access to the SDN controller, they manipulate the network's configuration for launching IP spoofing attacks on the network infrastructure. The objective of our experiment was to assess the effects of flooding switch flow rules on critical switch resources, including CPU utilization, memory usage, network latency, and switch throughput. By inundating the switches with a high volume of spoofed packets, we aimed to simulate a real-world DDoS attack scenario and understand the implications on the SDN infrastructure.

To defend against IP spoofing-based DDoS attacks in SDN, several countermeasures can be employed:

- **Packet Filtering:** SDN controllers can be configured to implement packet filtering mechanisms that drop traffic originating from known spoofed IP addresses or suspicious sources. However, maintaining an up-to-date database of spoofed IP addresses can be challenging due to the dynamic nature of SDN networks.
- **Flow Monitoring and Anomaly Detection:** By monitoring network flows and analyzing traffic patterns, anomalies indicative of IP spoofing attacks can be detected. Machine learning algorithms can be employed to identify abnormal traffic behavior and trigger appropriate mitigation actions.
- **Rate Limiting:** Rate limiting techniques can be applied at the SDN switches to restrict the rate of incoming packets from a particular source. This can help prevent overwhelming the network with excessive traffic from spoofed sources.
- **Source Authentication:** Implementing strong authentication mechanisms, such as digital certificates, can ensure the authenticity of the source IP addresses. This can help mitigate IP spoofing attacks by validating the legitimacy of traffic sources.
- **Collaborative Defense:** Cooperation and information sharing among SDN controllers and network administrators can enhance the effectiveness of DDoS mitigation. Sharing attack indicators, traffic patterns, and mitigation strategies can facilitate a coordinated response to IP spoofing-based DDoS attacks.
- **Traffic Engineering and Load Balancing:** By dynamically rerouting traffic and distributing it across multiple paths, SDN controllers can effectively manage and mitigate the impact of DDoS attacks. Load balancing techniques can ensure that attack traffic is distributed evenly, preventing specific network resources from becoming overwhelmed.

It is crucial to evaluate and combine multiple countermeasures to create a robust defense against IP spoofing-based DDoS attacks in SDN environments. Additionally, continuous monitoring, threat intelligence, and rapid response mechanisms are essential for timely detection and mitigation of such attacks [93].

4.1 Test Environmet

There are various test environments that can be used to evaluate the security of Software-Defined Networking (SDN) platforms. A useful environment for testing particular elements or situations in a controlled environment is a simulation environment. While simulation environments can offer some insights, it's possible that they don't truly reflect how an SDN system behaves in the real world.

On the other hand, although they can be pricey and dangerous to use, real environments provide the most true representation of an SDN system. Actual SDN infrastructure deployment for security testing has the potential to cause unintended affects and disrupt daily operations. In addition, real infrastructure can be expensive to acquire, build up, and maintain for some organizations.

In contrast, emulation tools provide a more practical and secure replacement for actual environments. Security testers can assess the effects of various security attacks on the system using these software-based tools, which simulate the behavior of an SDN system. Emulation tools can be tailored to test a variety of situations and configurations, and they can offer in-depth analysis and feedback that can be used to find vulnerabilities and increase system security as a whole.

Emulation tools, which are precise, safe, and economical, offer a perfect testing environment for security attacks on SDN systems. Emulation tools enable businesses to evaluate SDN systems more quickly and effectively while also guarding against security risks [94].

One of the top open-source network emulators for software-defined networks is Mininet, a widely used program (SDNs). Users can emulate various network devices and protocols, including SDN controllers and switches, and build virtual networks with varying topologies.

The adaptability and customizability of Mininet is one of its main benefits, without specialized hardware, users can quickly build and test various network configurations and run numerous experiments on various SDN scenarios. With the help of Mininet, users can build networks with hundreds or even thousands of devices [95].

4.2 Mininet emulation tool

Mininet is an open-source network emulation tool that can be used to build virtual networks on a single machine. Mininet helps programmers to construct and test intricate network topologies and applications in a safe setting. It creates a realistic

and repeatable environment for network testing and development using lightweight virtualization technology to imitate network devices and traffic. Mininet is a low-cost alternative for network testing and development because it is a free and open-source software program, that can be installed on common hardware. The network namespace function of the Linux kernel is used by Mininet to build compact virtual machines that can be linked together to model intricate network topologies. It may be simply modified to replicate various network environments and scenarios and supports a broad variety of network topologies and configurations. Because Mininet is so adaptable and scalable, programmers can design massive networks with tens of thousands or more nodes [96] [97].

It offers a command-line interface for users to communicate with the network and manage its actions. Mininet is frequently used for research, instruction, and development in both academia and industry. Its widespread use, adaptability, and capacity to faithfully reproduce complicated network environments account for its popularity.

- **Limitations:** Mininet has some drawbacks despite being a helpful tool for network emulation. The inability to mimic big networks is one of its key drawbacks. This is due to the fact that Mininet makes use of a single machine's limited resources, which might be taxed when mimicking expansive networks. Additionally, not all network functions and protocols are supported by Mininet, which can restrict the applications for which it can be used.
- **Advantages:** Mininet has a number of benefits over other network simulation tools. For instance, it is open-source and free, making it available to a variety of people. Additionally, it offers a framework for network emulation that is adaptable and adjustable, enabling users to build complicated topologies and test a range of network applications. Furthermore, Mininet's portable virtual machines make it simple to set up and disassemble network simulations, which can save time and resources.

4.3 Tools and Technologies

4.3.1 VirtualBox

VirtualBox is an open-source software used for virtualizing x86 computer systems. It acts as a hypervisor for creating Virtual Machines (VMs) in which the user can run another Operating System (OS). The Operating System in which VirtualBox application runs is called the host OS and the Operating system that runs inside the VM is called guest OS. VirtualBox can use Windows, Linux or macOS as a host OS. When creating a VM in VirtualBox we can choose CPU, number of cores and the amount of memory of the VM that we want to create [98]. We created a VM with the help of VirtualBox to create a virtual environment to run Mininet and to conduct the experiments and get the results we need.

4.3.2 RyU Controller

Ryu controller is an open, Software-Defined Networking (SDN) Controller designed to increase the agility of the network by making it easy to manage and adapt how traffic is handled. Since in an SDN the controller is considered as a core component that makes decisions in the network, making the controller to easily manage traffic is a big advantage. The Ryu controller provides software components with well defined APIs that make it easier for the developers to create new network management and control applications [99]. We use Ryu controller as it provides great interface to make the controller to deal with flowrules of the switches that we need.

4.3.3 Open vSwitch Kernel

Open vSwitch is an open source, virtual multilayer software switch that can be run in virtual machine environments. It provides access to virtual networking layer with standard control and visibility interfaces, and enables distribution across multiple physical servers. Open vSwitch code is written in C, and provides support for forwarding layer abstraction to different software and hardware platforms. Open vSwitch (OVS) is an important component in modern data center SDNs, where it is used for aggregating all the VMs at the server hypervisor layer [80]. The Open vSwitch (OVS) Kernel is an interface that provides with flexible userspace control and gives the user the ability to control flow-level packet processing of any selected network devices.

4.3.4 Scapy

Scapy is a python program that enables the user to send, sniff and dissect and forge network packets. This allows users to make tools that can probe, scan or attack networks. Scapy is a powerful packet-manipulation tool that is used whenever an attack needs to be simulated and analyzed. It is the best tool available that helps with simulating various types of attacks and thus helping in analyzing various types of attacks [100].

4.3.5 PuTTY

PuTTY is a free implementation tool of SSH and telnet on PCs that run Microsoft Windows, it also includes a xterm terminal emulator. PuTTY is useful in case you want to access an account on a Unix or other multi-user systems from your PC. The primary advantage of PuTTY is that it provides a secure and encrypted way to make SSH connections to the remote system [101].

4.3.6 Wireshark

Wireshark is an application that captures packets from a network connection, such as from your PC to your home office or the Internet. Wireshark is the most often-used packet sniffer in the world. Wireshark has many uses like Troubleshooting networks that have performance issues. Wireshark is used to trace connections, view the contents of suspect network transactions and identify bursts of network traffic [102].

4.3.7 Xming

Xming is an open-source X Window System server for Microsoft Windows. It enables Windows computers use remotely graphical Linux programs on a Linux server. Running graphical programs on distant servers over a network connection is made safe and effective by Xming. System administrators, programmers, and users who require access to Linux machines from Windows desktops can be benefited. Running Linux desktop programs, remote management of Linux servers, and access to graphical tools like Wireshark and GIMP are some frequent uses of Xming [103].

4.4 Testbed

4.4.1 Requirements

This section covers the necessary tools, technologies, tool installations, and tool operation procedures.

- Hardware requirements:
 1. Computer with at least 8 GB of RAM
 2. Virtual Box version 6.1.18
 3. Ethernet adapter
- Software:
 1. Mininet version 2.3.0d5
 2. Putty CLI version 0.74
 3. Xming server version 7.7.0.10
 4. Ryu controller
 5. Scapy packet manipulation tool
- Configuration:
 1. Virtual Box settings: 2 CPUs, 4096 MB RAM, bridged network adapter
 2. Xming: display server running on host machine
 3. Mininet: custom topology with required switches and hosts
 4. Putty: SSH connection to Virtual Box instance with X11 forwarding enabled

4.4.2 Experimental Setup for Experiments

For this thesis we are creating a Mininet topology using python script. We create three switches and 9 hosts (3 hosts attached to each switch) with a Ryu controller. The links between devices is 10 Mbps. The code below is an example which shows how a network in Mininet with a controller can be started. Any required number of switches, hosts can be added and links can be used to connect the devices to the

network. We have added three switches and 9 hosts with a remote controller using python with network.py file.

```
###Python code
from mininet.net import Mininet
from mininet.node import RemoteController, OVSKernelSwitch
from mininet.link import TCLink
# create the network and specify controller and switch.
network = Mininet(controller=RemoteController, switch=OVSKernelSwitch,
link=TCLink)
# add the controller to the network
controller0 = network.addController('controller0', controller=RemoteController,
ip='127.0.0.1', port=6633)
# add the switches
switch1 = network.addSwitch('s1')
#Add the required number of switches
# add the nodes
host1 = network.addHost('h1')
#add required number of hosts
# add the links
network.addLink(swith1, host1, bw=10)
# start the network
network.start()
# start the controller
network.controllers[0].start()
# run the command line interface
network.interact()
# stop the network
network.stop()
```

Run the above code in Mininet VM with follwing command:

```
sudo python3 network.py
```

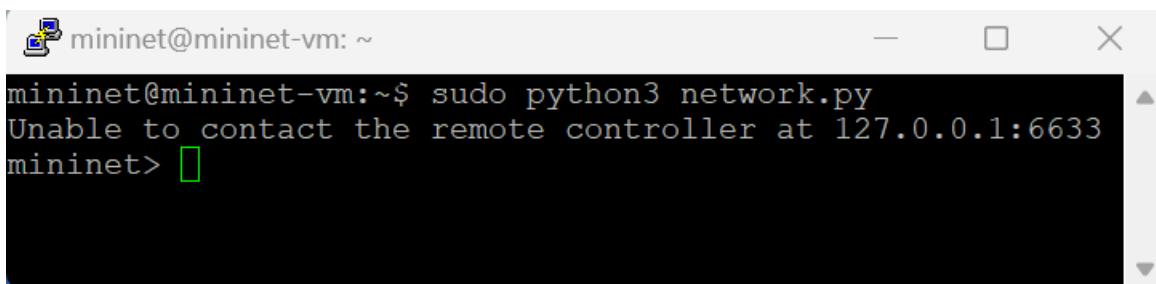


Figure 4.1: Network is unable to connect

We can see from the figure 4.1 that the network is unable to connect, as we did not start the controller yet. There are different types of controllers but we are using Ryu controller for the experiment. Ryu is an open-source SDN controller, with user-friendly interface for controlling network traffic in Mininet. Ryu is lightweight and

has a small footprint in comparison to other SDN controllers, like OpenDaylight and ONOS, making it simpler to deploy and administer. A variety of network protocols are supported by Ryu, which also offers a versatile and configurable interface for setting up network flows and policies. Because of Ryu's modular architecture, developers can simply modify and extend the controller to satisfy certain network requirements [104] [105].

After successfully installing Ryu and connecting the Ryu controller to the network, we can now proceed and connect it to the network. Whenever switches get connected with the controller, the controller sends feature request message to the switch after establishing the session with SDN switches. Then the switch responds with a reply message to feature request message from controller. Then Ryu controller sends config request messages to the switches to set configuration parameters [106]. Ryu-application code that sets config messages and respond to the feature request messages is as follows.

```
#sample code for ryu switch
class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
    ##Code to handle the Switch feature respomd messages
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        match = parser.OFPMatch()
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                           ofproto.OFPCML_NO_BUFFER)]
        mod = parser.OFPFlowMod(datapath=datapath, match=match
                                instructions=inst )
        datapath.send_msg(mod)
```

We also needed to add additional code that can handle OpenFlow packets, that request OpenFlow rules from the SDN switches.

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath_travel = msg.datapath
    of_proto = datapath.ofproto
    ofproto_parser = datapath_travel.ofproto_parser
    in_port = msg.match['in_port']
    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]
```

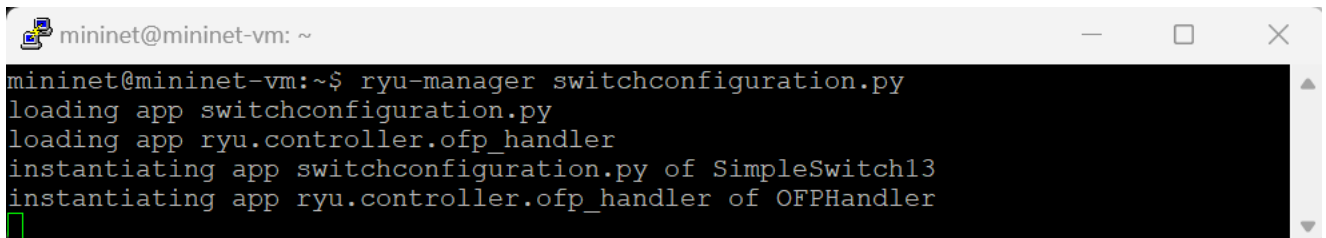
```

destination = eth.dst
source = eth.src
dpid = datapath_travel.id
self.mac_to_port.setdefault(dpid, {})
actions = [parser.OFPACTIONOutput(out_port)]
out = parser.OFPPacketOut(datapath=datapath_travel
                          in_port=in_port, actions=actions, data=data)
datapath_travel.send_msg(out)

```

After adding the code, we need to include all the necessary packages in the Ryu application and run using following command in the new terminal.

```
ryu-manager switchconfiguration.py
```



```

mininet@mininet-vm: ~
mininet@mininet-vm:~$ ryu-manager switchconfiguration.py
loading app switchconfiguration.py
loading app ryu.controller.ofp_handler
instantiating app switchconfiguration.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler

```

Figure 4.2: Start of Ryu controller

After starting the Ryu controller now we can verify using pingall as shown in figure 4.3. We can see links connected using "links" command and all hosts using "dump" command in Mininet. We can see all available options in Mininet console by running 'help' in Mininet console. Network emulator 'xterm' can be used to launch virtual terminals for the various network nodes that are created within the Mininet network emulation environment. We can use each nodes virtual terminals for performing tasks. They can be used to interact with the node as if they were connected to a physical terminal. They can run commands, configure network interfaces, and troubleshoot network issues.

<pre> mininet> links s1-eth1<->h1-eth0 (OK OK) s1-eth2<->h2-eth0 (OK OK) s1-eth3<->h3-eth0 (OK OK) s2-eth1<->h4-eth0 (OK OK) s2-eth2<->h5-eth0 (OK OK) s2-eth3<->h6-eth0 (OK OK) s3-eth1<->h7-eth0 (OK OK) s3-eth2<->h8-eth0 (OK OK) s3-eth3<->h9-eth0 (OK OK) s1-eth4<->s2-eth4 (OK OK) s2-eth5<->s3-eth4 (OK OK) mininet> </pre>	<pre> mininet> pingall *** Ping: testing ping reachability h1 -> h2 h3 h4 h5 h6 h7 h8 h9 h2 -> h1 h3 h4 h5 h6 h7 h8 h9 h3 -> h1 h2 h4 h5 h6 h7 h8 h9 h4 -> h1 h2 h3 h5 h6 h7 h8 h9 h5 -> h1 h2 h3 h4 h6 h7 h8 h9 h6 -> h1 h2 h3 h4 h5 h7 h8 h9 h7 -> h1 h2 h3 h4 h5 h6 h8 h9 h8 -> h1 h2 h3 h4 h5 h6 h7 h9 h9 -> h1 h2 h3 h4 h5 h6 h7 h8 *** Results: 0% dropped (72/72 received) mininet> </pre>
--	--

Figure 4.3: Links and Pingall from Mininet Console

For the experiment we choose attackers named host2 to host 8 as shown in figure 4.4. The attacker hosts constantly send packets to other attacker hosts in the network

with different source IP address. When the packets reach the SDN switches from attackers, the SDN switch checks for flow rules that match with the packets. If no flow rules match, then the SDN switch sends the packets to the controller for modifying the flow rules. The SDN controller sends a Flow-mod message with new source IP and destination IP addresses, which adds new flow rules to the SDN switches.

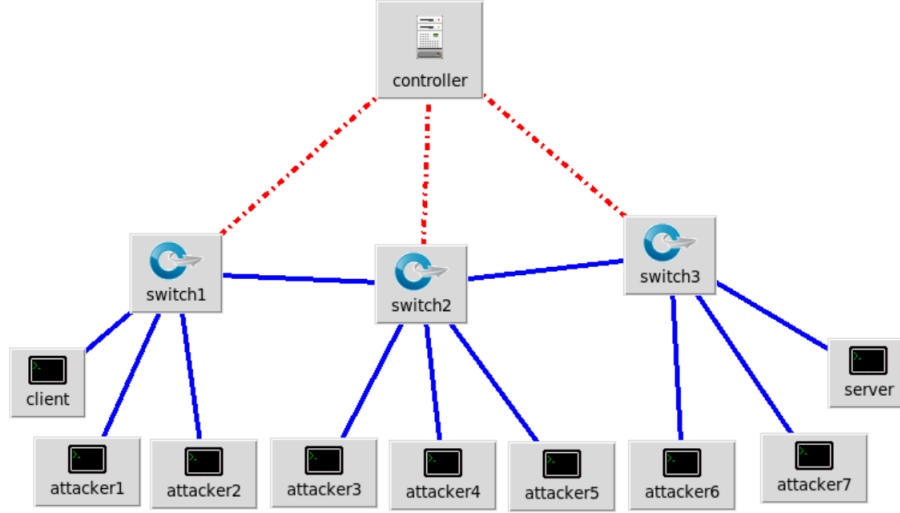


Figure 4.4: SDN Architecture for TestBed

The attack is initiated by running a python script `ipflood.py`. The python script is added to the destination address of attacker2 to attacker9 with random source IP addresses, the packets dispatch from each attacker until the script is stopped. We run the python script in each attacker from Mininet console using the following command:

```
for i in {2..9};do attacker$i python3 ipflood.py & done
```

After initiating the attack we can see the controller constantly adding flow rules to the SDN switches for every new packet, thus constantly increasing the number of flow-rules. The flow-rules can be seen using the following command in the switch1 terminal. Figure 4.5 shows flow-rules with random source IP addresses.

```
sudo ovs-ofctl dump-flows s1
```

We can count number flow-rules in switches. Figure 4.6 shows count of total flow-rules in switches. Wireshark is used to analyse packets and activity of packets. The packets of OpenFlow between switches and controller can be easily filtered and analysed using wireshark. The packets after initiation of the IP Spoofing attack can be seen in figure 4.7. The OpenFlow messages can be seen in figures 3.7,3.8 taken from Wireshark.

```

cookie=0x0, duration=0.242s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=207.230.232.162,nw_dst=10.0.0.8 actions=output:"s1-eth4"
cookie=0x0, duration=0.242s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=34.101.90.202,nw_dst=10.0.0.2 actions=output:"s1-eth2"
cookie=0x0, duration=0.241s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=152.172.139.156,nw_dst=10.0.0.2 actions=output:"s1-eth2"
cookie=0x0, duration=0.241s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=242.69.248.48,nw_dst=10.0.0.6 actions=output:"s1-eth4"
cookie=0x0, duration=0.241s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=100.63.215.46,nw_dst=10.0.0.4 actions=output:"s1-eth4"
cookie=0x0, duration=0.240s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=244.177.182.5,nw_dst=10.0.0.8 actions=output:"s1-eth4"
cookie=0x0, duration=0.240s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=68.165.196.101,nw_dst=10.0.0.2 actions=output:"s1-eth2"
cookie=0x0, duration=0.240s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=73.67.121.51,nw_dst=10.0.0.4 actions=output:"s1-eth4"
cookie=0x0, duration=0.239s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=242.178.146.107,nw_dst=10.0.0.4 actions=output:"s1-eth4"
cookie=0x0, duration=0.239s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=107.75.76.180,nw_dst=10.0.0.2 actions=output:"s1-eth2"
cookie=0x0, duration=0.239s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=192.44.167.144,nw_dst=10.0.0.2 actions=output:"s1-eth2"
cookie=0x0, duration=0.239s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=0.170.81.71,nw_dst=10.0.0.3 actions=output:"s1-eth3"
cookie=0x0, duration=0.239s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=33.95.145.222,nw_dst=10.0.0.3 actions=output:"s1-eth3"
cookie=0x0, duration=0.239s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=91.176.178.11,nw_dst=10.0.0.2 actions=output:"s1-eth2"
cookie=0x0, duration=0.239s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=139.254.47.21,nw_dst=10.0.0.3 actions=output:"s1-eth3"
cookie=0x0, duration=0.238s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=42.238.125.249,nw_dst=10.0.0.3 actions=output:"s1-eth3"
cookie=0x0, duration=0.238s, table=0, n_packets=0, n_bytes=0, idle_timeout=100, priority=1,ip,nw_src=157.64.128.46,nw_dst=10.0.0.3 actions=output:"s1-eth3"
cookie=0x0, duration=363.455s, table=0, n_packets=18571, n_bytes=782334, priority=0 actions=CONTROLLER:65535
root@mininet-vm:/home/mininet#

```

Figure 4.5: Flow-rules in SDN Switch 1

```

"Node: s1" (root)@mininet-vm
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1 | wc -l
2
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1 | wc -l
5757
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1 | wc -l
6752
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1 | wc -l
10405
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1 | wc -l
15117
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1 | wc -l
27255
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1 | wc -l
30876
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1 | wc -l
32237
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1 | wc -l
34071
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1 | wc -l
34903
root@mininet-vm:/home/mininet# sudo ovs-ofctl dump-flows s1 | wc -l
35200
root@mininet-vm:/home/mininet#

```

Figure 4.6: Count of flow-rules in Switch 1

No.	Time	Source	Destination	Protocol	Length	Info
64	3.853224276	198.156.1.61	10.0.0.8	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
65	3.899972136	16.193.262.253	10.0.0.4	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
66	3.983292130	226.8.30.109	10.0.0.3	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
67	4.115022391	111.160.211.129	10.0.0.4	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
68	4.320879490	169.31.237.89	10.0.0.7	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
69	4.362193120	214.149.129.214	10.0.0.9	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
70	4.474397138	236.111.58.7	10.0.0.9	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
71	4.543213371	228.163.213.99	10.0.0.5	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
72	4.632905928	213.159.22.71	10.0.0.5	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
73	4.761993835	246.221.297.39	10.0.0.8	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
74	4.875165999	211.227.173.140	10.0.0.4	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
75	4.967063577	147.182.37.178	10.0.0.9	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
76	5.068554342	11.181.8.89	10.0.0.2	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
77	5.152490116	172.195.142.241	10.0.0.6	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
78	5.302315723	171.35.74.27	10.0.0.5	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
79	5.356622706	30.36.25.30	10.0.0.7	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
80	5.615576167	229.8.4.199	10.0.0.5	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
81	5.667557430	43.155.85.19	10.0.0.8	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
82	5.739527293	46.176.92.111	10.0.0.5	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
83	5.869168970	57.157.237.103	10.0.0.3	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
84	5.927288823	232.188.100.24	10.0.0.6	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
85	6.007492190	58.25.113.246	10.0.0.3	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
86	6.062739891	98.68.154.65	10.0.0.4	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
87	6.156395240	154.44.149.222	10.0.0.9	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
88	6.228379418	122.85.231.21	10.0.0.8	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
89	6.276417048	48.159.142.241	10.0.0.6	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
90	6.331289985	81.146.116.109	10.0.0.6	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
91	6.405597671	111.82.189.31	10.0.0.9	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
92	6.478738107	16.146.19.51	10.0.0.8	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
93	6.547458232	100.197.126.147	10.0.0.5	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
94	6.607145314	156.156.251.202	10.0.0.9	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
95	6.653240023	95.58.248.58	10.0.0.3	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
96	6.715395999	30.192.142.127	10.0.0.3	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
97	6.756701713	231.184.206.207	10.0.0.8	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...
98	6.800000000	192.168.1.100	10.0.0.8	ICMP	42	Echo (ping) request 1d-0x0000, seq=0/0, ttl=64 (no response ...

Figure 4.7: Packets of IP spoofing in switch1-eth 1

5.1 Experiment 1

In Experiment 1, the latency calculations were performed by measuring the ping between the client and the server, while simultaneously increasing the number of flow rules in the switches after initiating an IP spoofing attack. The attack was initiated by executing the "flood.py" file in the attacker hosts of SDN network as shown in Figure 4.4.

To collect latency data, 1000 ICMP packets were sent between the client and server, and the round trip time values for minimum, average, maximum, and deviation were recorded. However, calculating the latency at each switch proved to be challenging due to the use of loopback in Mininet for switches with controllers. Instead, the focus was on collecting the flow rules present in all switches, as the packets from the client had to traverse through switch1, switch2, and switch3 before reaching the server. The average number of flow rules in all switches was recorded along with the corresponding ping values, which were stored in a separate file.

rtt / flow	min	avg	max	mdev
500	0.081	0.610	48.131	4.776
1000	0.096	0.650	53.452	5.307
2000	0.098	0.694	59.760	5.826
5008	0.090	0.762	61.647	6.232
10014	0.091	0.787	69.787	6.435
20384	0.092	0.820	77.885	6.645
30032	0.094	0.859	86.983	7.571
40000	0.091	0.934	92.906	8.264
50000	0.097	0.984	98.000	9.472
60300	0.095	1.173	104.080	10.343
70017	0.102	1.345	110.279	11.096
90001	0.112	1.579	122.722	12.739
110000	0.102	1.631	127.686	13.126
130000	0.101	1.756	133.325	13.793
150069	0.092	1.832	141.768	14.332

Table 5.1: Latency (in milli seconds) with increasing Number of Flow-rules

Table 5.1 presents the collected data, displaying the latency values obtained as the

number of flow rules increased in SDN switches. The table provides the average number of flow rules in SDN switches, the minimum value of ping latency, the average value of ping latency, the maximum value of ping latency, and the standard deviation of ping latency.

The throughput of every SDN switch is collected at each minute with increasing of flow rules. To collect the number of bytes in the switches, an application is developed in Ryu controller that can monitor and send requests for flow rules to switches every minute. For collecting throughput there is a little change in architecture of SDN in figure 4.4. All hosts in the architecture are used as attackers and they send spoofed packets to all other hosts randomly.

```
def _monitor(self):
    for datapath in self.datapaths():
        ofproto_datapath = datapath.ofproto
        parser_datapath = datapath.ofproto_parser
        req = parser_datapath.OFPFlowStatsRequest(datapath)
        datapath.send_msg(req)
```

FRules1	N_Bytes1	FRules2	N_Bytes2	FRules3	N_Bytes3
23638	1257690	33957	1698942	23702	1262982
43354	2167166	61993	2909088	43458	2184364
61457	3031080	89227	4082160	62647	3066076
81525	3865782	112916	5180976	83757	3907788
101877	4641738	131807	6207070	104356	4679970
115142	5352452	141794	7121633	118126	5408924
125834	6006210	152037	8003742	128289	6097387
133422	6466324	160626	8745903	136284	6644093
138212	6876288	168224	9294984	141620	7112090
142539	7241691	174515	9696352	145956	7526657
146723	7566073	179010	9973310	150764	7893729
150321	7858241	182088	10136896	157912	8192606
152949	8019823	181867	10245458	159509	8417260

Table 5.2: Number of bytes in switches with increasing flow rules

To collect information of flow rules sent by switches, and to collect the number of bytes, another function is added in the Ryu application. The Throughput.csv file is used to collect information about switch, both the number of flow rules and total number of bytes can be collected.

```
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, [MAIN_DISPATCHER])
def flow_stats_reply_handler(self, ev):
    self.switch_id = ev.msg.datapath.id
    self.flowswitch[self.switch_id-1][1] +=len(ev.msg.body)
    for flow in ev.msg.body:
        if flow.priority ==0:
            self.flowswitch[self.switch_id-1][2] +=flow.byte_count
```



```

else:
    self.flowswitch[self.switch_id-1][2] +=flow.byte_count+100

```

The collected values are written into a table and later the throughput is calculated in BPS.

```

throughput = (difference in totalbytes for 60 second )*8/60

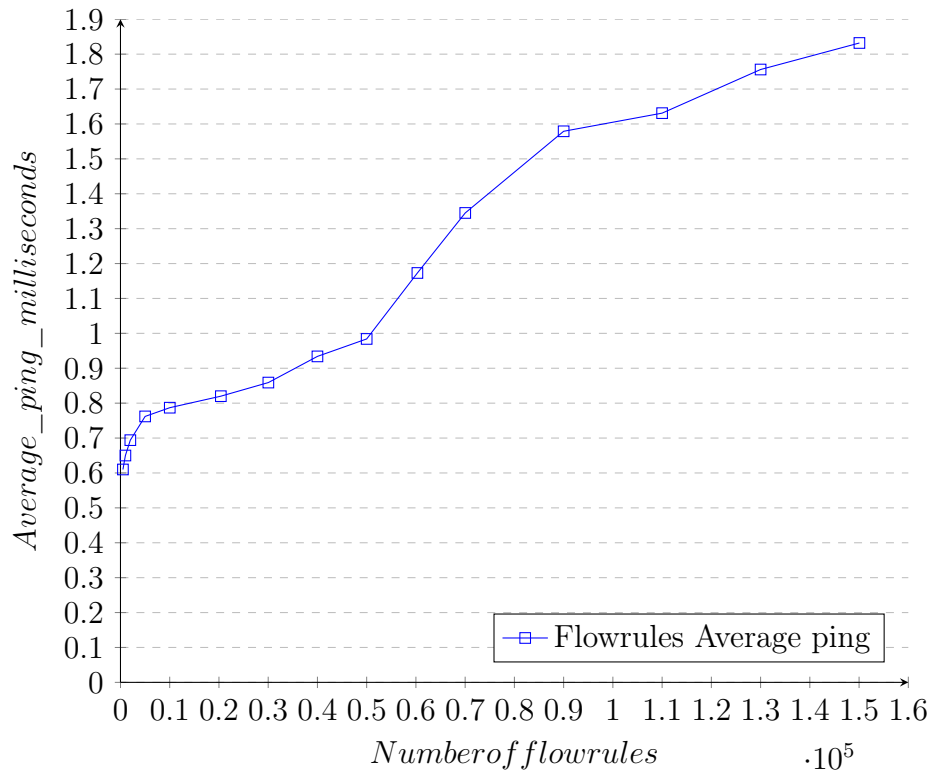
```

Table 5.2 illustrates the number of flow rules and corresponding throughput calculated for each switch over a one-minute interval.

5.1.1 Results and Analysis for Experiment 1

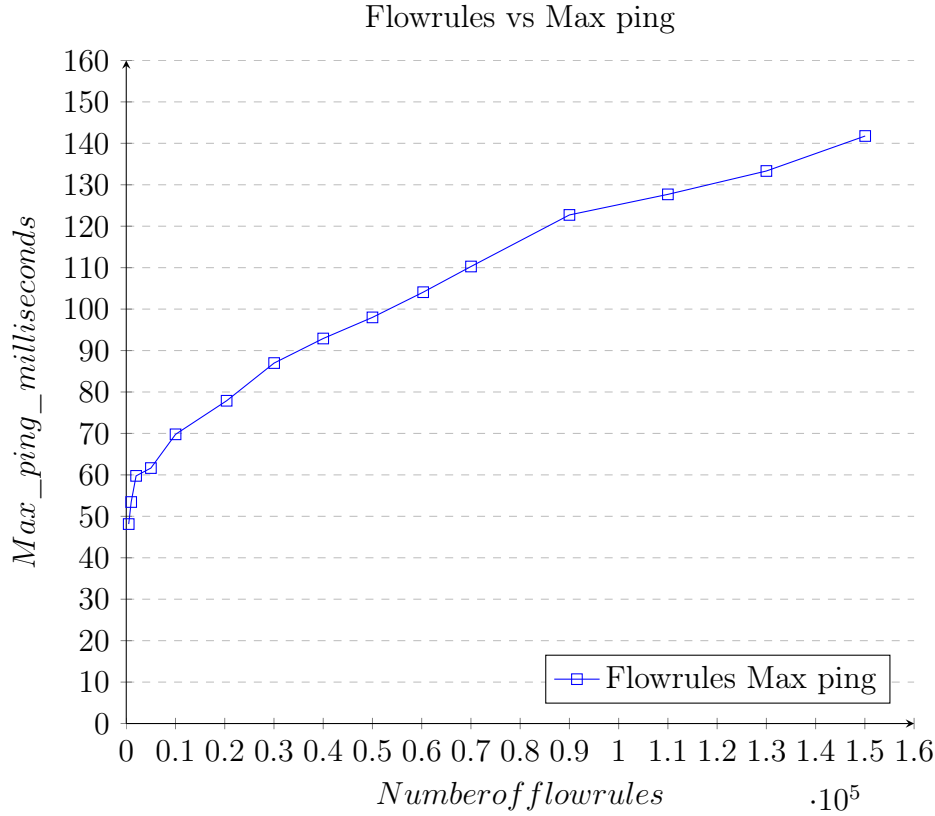
After conducting Experiment 1, various graphs were generated using the collected data. These graphs are divided into four categories, with the first three focusing on latency and the remaining one on throughput. Specifically, the graphs were plotted based on the tables depicting the relationship between flow rules and different latency measurements. The first graph illustrates the correlation between the number of flow rules and the average latency of ping. The second graph depicts the relationship between flow rules and the maximum latency observed in pings, while the third graph shows the correlation between flow rules and the standard deviation of latency.

Flowrules vs Average ping



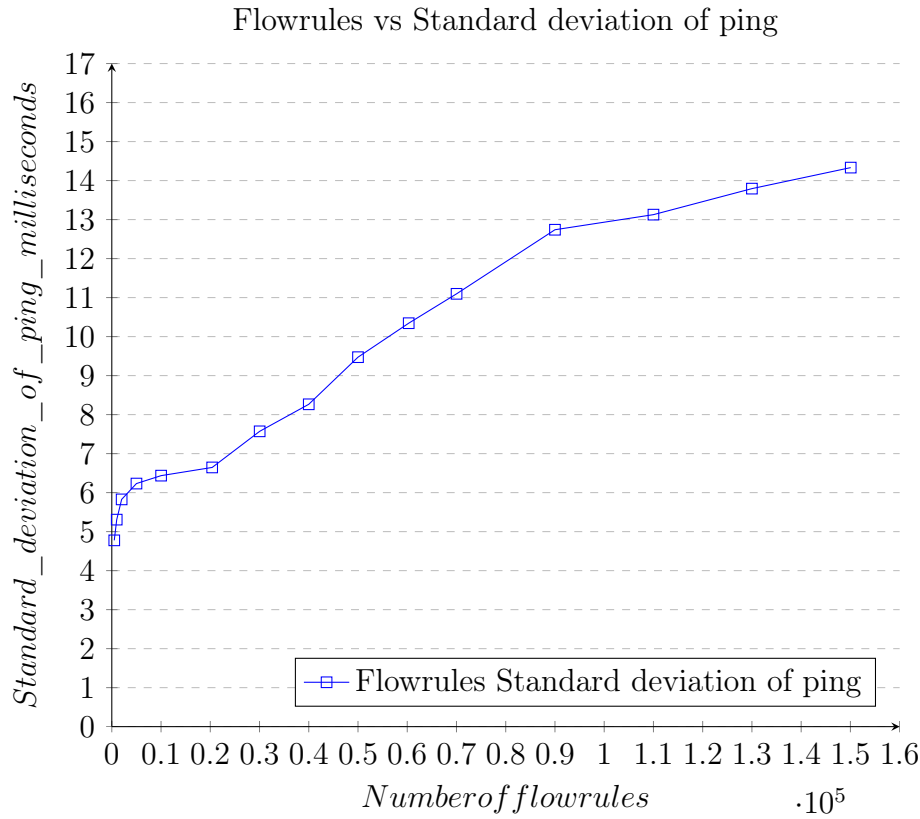
Based on the graph "Flowrules vs Average ping", the average ping is affected as the number of flow rules increases in the initial stages (500 to 10,000 flow rules), the average ping shows a gradual increase. The average ping starts at 0.610 ms for 500 flow rules and gradually increases to 0.787 ms for 10,014 flow rules. This indicates that as the number of flow rules increases, there is a slight impact on the average ping, but the effect is not significant.

As the number of flow rules continues to increase (20,000 to 150,069 flow rules), the average ping shows a more noticeable increase. From 20,384 flow rules to 150,069 flow rules, the average ping increases from 0.820 ms to 1.832 ms. This suggests that as the number of flow rules reaches higher levels, there is a more substantial impact on the average ping values.



Based on the graph "Flowrules vs Max ping", the maximum ping is affected as the number of flow rules increases in the initial stages (500 to 10,000 flow rules), the maximum ping shows a gradual increase. The maximum ping starts at 48.131 ms for 500 flow rules and gradually increases to 69.787 ms for 10,014 flow rules. This indicates that as the number of flow rules increases, there is a slight impact on the maximum ping, but the effect is not significant.

As the number of flow rules continues to increase (20,000 to 150,069 flow rules), the maximum ping shows a more noticeable increase. From 20,384 flow rules to 150,069 flow rules, the maximum ping time increases from 77.885 ms to 141.768 ms. This suggests that as the number of flow rules reaches higher levels, there is a more substantial impact on the maximum ping values.



Based on the graph "Flowrules vs Standard deviation of ping", the standard deviation of ping is affected as the number of flow rules increases in the initial stages (500 to 10,000 flow rules), the standard deviation of ping values show a gradual increase. The standard deviation starts at 4.776 ms for 500 flow rules and gradually increases to 6.435 ms for 10,014 flow rules. This indicates that as the number of flow rules increases, there is a slight impact on the standard deviation of ping, but the effect is not significant.

As the number of flow rules continues to increase (20,000 to 150,069 flow rules), the standard deviation of ping values show a more noticeable increase. From 20,384 flow rules to 150,069 flow rules, the standard deviation increases from 6.645 ms to 14.332 ms. This suggests that as the number of flow rules reaches higher levels, there is a more substantial impact on the standard deviation of ping.

The graphs above clearly demonstrate that as the number of flow rules increases in SDN switches, there is a consistent and adverse impact on response times, resulting in higher latency. The standard deviation also increases, indicating a greater variation in latency measurements. These findings highlight the importance of effectively managing flow rules to maintain satisfactory network performance and minimize latency-related issues.

The analysis of the figures generated Figure 5.1 and Figure 5.2 from table 5.2 reveals that as the number of flow rules increases in SDN switches, the throughput in these switches decreases. For example, in switch2 at 60 seconds time we can see 76000 bps as the throughput and at 120 seconds it decreased to 52000 bps and at 780 seconds it is now 2000 bps, similarly we can see the other switches following the decreasing trend, this decrease in throughput is observed over time, indicating that

the switches are unable to handle the increasing load imposed by the attack traffic. As time progresses, the number of flow rules increases, contributing to the reduction in throughput in SDN switches. This observation is supported by Figure 5.1, which demonstrates the gradual decrease in throughput over time. For example, in switch2 at 60 seconds time we can see 38000 is the number of flowrules and at 120 seconds it increased to 60000 and at 780 seconds it is now 18000 flowrules, we can also observe that the increase in the number of flowrules is a lot less with increasing time, similarly we can see the other switches following the increasing trend. Additionally, the analysis of Figure 5.1 and Figure 5.2 reveals that switch 2 exhibits higher throughput and a larger number of flow rules compared to switch 1 and switch 3. This difference in performance can be attributed to the architectural configuration depicted in Figure 4.4. Switch 2 acts as a bridge between switch 1 and switch 3, necessitating all forwarding traffic to pass through switch 2. Consequently, switch 2 experiences a higher volume of traffic, leading to increased throughput and a larger number of flow rules.

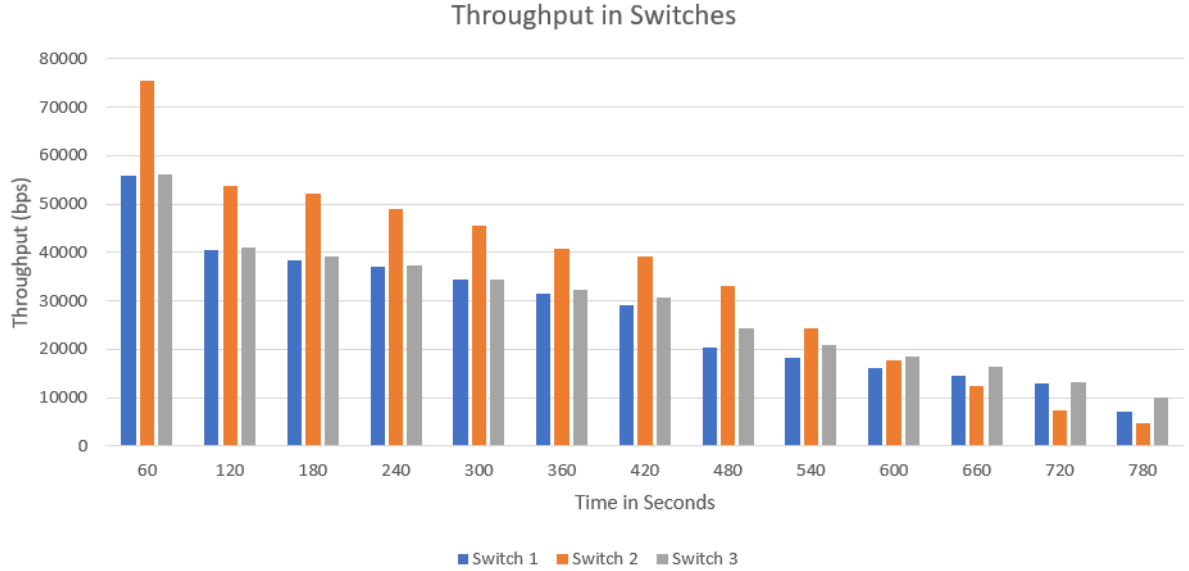


Figure 5.1: Throughputs of 3 different switches

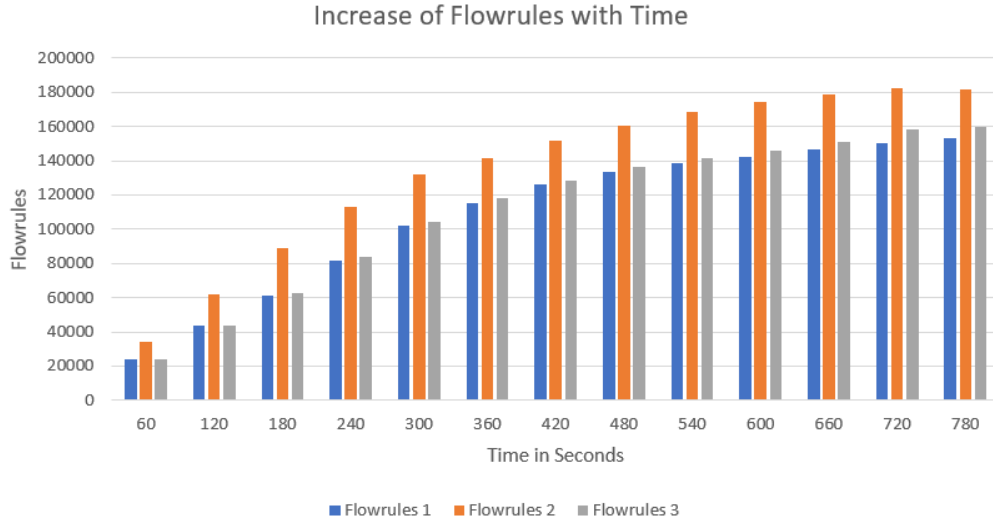


Figure 5.2: Increasing of flowrules in switches with time

5.2 Experiment 2

To collect storage and CPU utilization of switches with increasing flow rules, the concept of storage is used. Mininet uses storage and CPU of the virtual machine for running the virtual nodes that are created by Mininet. Switches in Mininet use the Open-vSwitch package for storing and processing various functions of SDN switches. The total percentage of memory and CPU utilized by Open-vSwitch package is collected with increasing of flow rules using "ps aux". The process' pid is collected after running the command "ps aux | grep "pid" in Mininet. The total flow-rules with storage percentage and CPU utilization is collected into a separate file named "storageandcpu".

5.2.1 Results and Analysis for Experiment 2

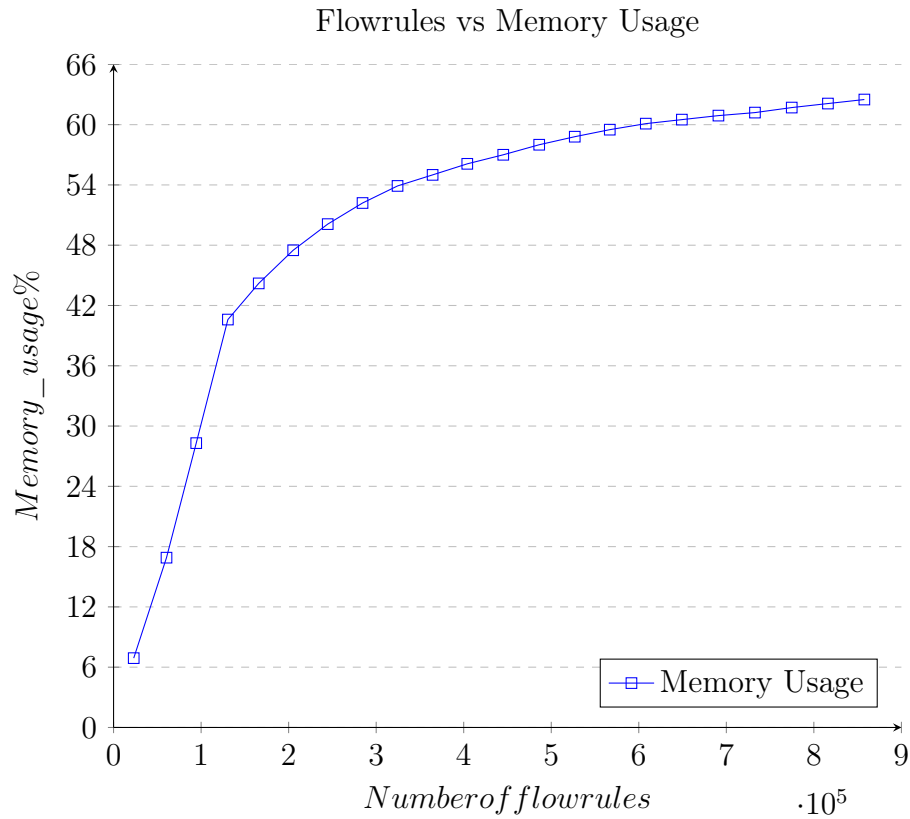
The analysis of the generated graphs "Flowrules vs CPU Usage" and "Flowrules vs Memory Usage" from Table 5.3 provides valuable insights into the impact of increasing flow rules on CPU and memory usage in the switches.

The memory usage graph exhibits a sharp initial increase in memory usage values as the number of flow rules increases. However, as the number of flow rules continues to increase, the rate of memory usage growth slows down. This observation suggests that at the beginning, the VM assigned a certain amount of memory to the process, but as the process requires more memory to accommodate the growing number of flow rules, the VM tries to allocate additional memory from its available resources. It is important to note that the memory usage increase is more pronounced compared to the CPU usage increase.

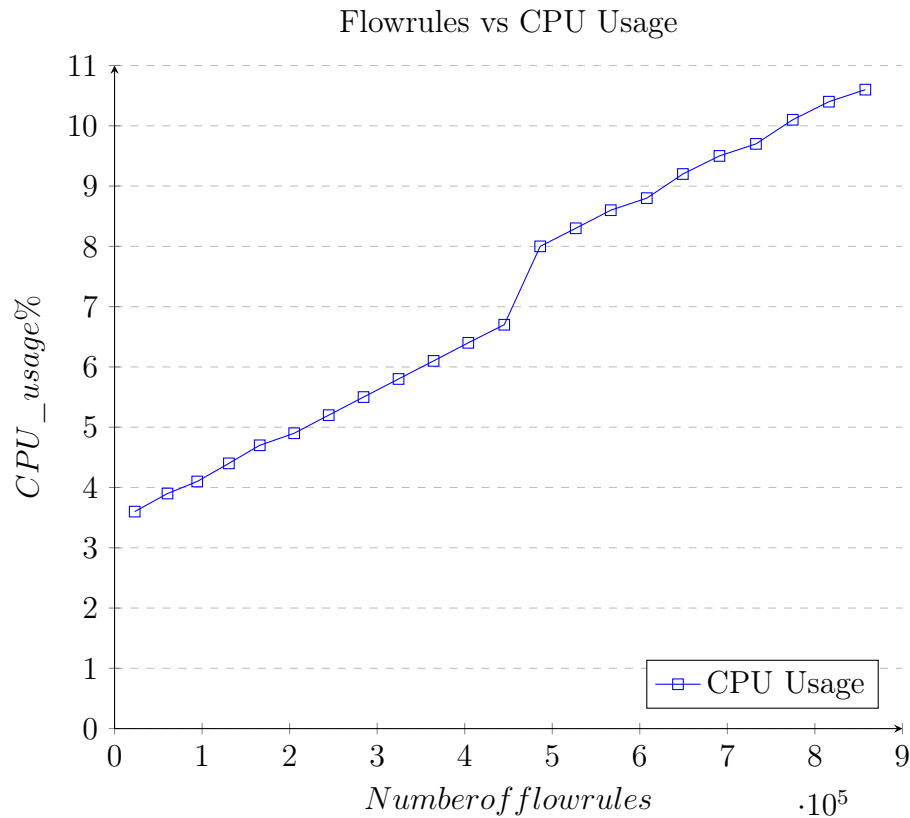
On the other hand, the CPU usage graph demonstrates a gradual increase in CPU usage as the number of flow rules increases. However, the increase in CPU usage is not significant or threatening. This indicates that the IP spoofing attack does not have a notable impact on the CPU usage of the switches.

Total_flowrules	% memory usage	% cpu usage
23018	6.9	3.6
60465	16.9	3.9
94416	28.3	4.1
130596	40.6	4.4
165843	44.2	4.7
205190	47.5	4.9
244659	50.1	5.2
284425	52.2	5.5
324406	53.9	5.8
364476	55.0	6.1
404065	56.1	6.4
445160	57.0	6.7
486231	58.0	8.0
526868	58.8	8.3
566880	59.5	8.6
608016	60.1	8.8
649224	60.5	9.2
690987	60.9	9.5
732727	61.2	9.7
774742	61.7	10.1
816086	62.1	10.4
857695	62.5	10.6

Table 5.3: CPU usage and memory usage of switch with increasing flow rules



In summary, the analysis reveals that increasing the total number of flow rules in the switches results in higher memory usage, with a relatively more rapid initial increase. The CPU usage, while showing a slight increase, does not reach concerning levels. These findings suggest that IP spoofing has a greater impact on memory usage, potentially exhausting the available memory resources on the switches. However, the CPU utilization remains within acceptable limits and is not significantly affected by the attack.



5.3 Experiment 3

To mitigate the effect on SDN switches, we are trying to monitor the flow rules in the SDN switches using a new application that we have written in the Ryu controller. The application monitors the flow rules for every sixty seconds and removes the flow-rules, if number of flow-rules in switches is greater than 1000, time of duration of flow rule is greater than 10 seconds and number of packets in flow rules is less than 2. The values above have decided based on the size of network but this values for monitoring can be changed depending on the size of network, services provided by cloud and on the basis of protocols. This application continuously monitors and removes flow rules that are injected by IP spoofing attack and maintains length of flow rules to a limited amount.

```
if ((len(ev.msg.body) >= 1000 )&& (flow.duration_sec > 10)):
    if flow.packet_count < 2
        mod = parser.OFPFlowMod(datapath = ev.msg.datapath,
                                table_id = flow.table_id, command=ofproto.OFPFC_DELETE_STRICT,
                                out_port=ofproto.OFPP_ANY, out_group=ofproto.OFPG_ANY,
                                match=flow.match, instructions=instruction)
        ev.msg.datapath.send_msg(mod)
self.logger.info("flow rule deletd")
```

After running the mitigation application, which is on the Ryu controller, the latency, throughput, storage and CPU utilization of switches are calculated.

Collection of throughput, latency, storage and CPU utilization is done in a similar fashion as mentioned in Experiment 1 and Experiment 2, and we have now tabulated these values in tables 5.4, 5.5 and 5.6. This data is used to obtain the graphs that are depicted below.

FRules1	N_Bytes1	FRules2	N_Bytes2	FRules3	N_Bytes3
5599	376354	7533	428812	4908	273664
8194	694000	11716	879892	8199	590134
8130	1012906	11586	1331434	8155	911644
8246	1330216	11612	1776802	8035	1220764
7779	1634296	11026	2204992	7730	1523836
8072	1946314	11555	2648302	8064	1834930
8103	2264422	11517	3097576	7975	2149090
8047	2574256	11482	3537232	8070	2459344
7887	2884342	11312	3977224	7969	2772202
8058	3197746	11549	4422466	8072	3084514
7909	3506404	11366	4862920	7885	3392080
8116	3818380	11553	5305054	8153	3707206

Table 5.4: Number of bytes in switches after mitigation

Table 5.5 displays the recorded values for memory and CPU utilization following the implementation of the mitigation technique aimed at reducing flow-rule flooding

within the switches of our SDN network. As a result of the implemented mitigation technique, the flow rule count was kept in check, ensuring that it remained within a certain limit and was consistent. Additionally, the table indicates that the values for both memory and CPU utilization were stable and did not exhibit significant increases. Similarly, Table 5.4 shows the number of bytes recorded in the switches after mitigation. It can be observed that the total count of flow rules in both tables remained below 10,000 at all times and were curtailed if the threshold was crossed. These results unequivocally demonstrate the effectiveness of the mitigation technique in preventing flow-rule overflow and subsequent switch malfunction.

Total_flowrules	% memory usage	%cpu usage
6	0.4	0.5
4543	2.2	1.5
9369	3.4	1.9
9290	3.2	1.9
9297	3.4	1.9
8845	3.1	1.9
9230	3.2	1.9
9198	3.1	1.9
9199	3.1	1.9
9053	3.1	1.9
9226	3.2	1.9
9053	3.2	1.9
9274	3.4	1.9

Table 5.5: CPU usage and memory usage after mitigation

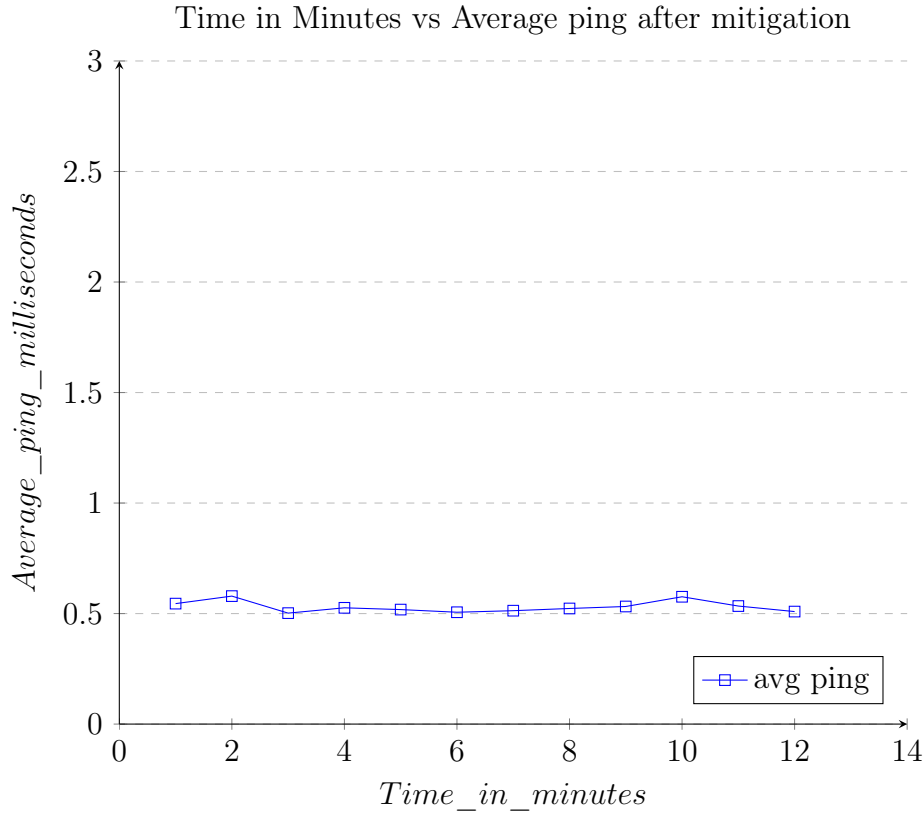
The data presented in Table 5.6 displays the latency measurements following the implementation of the mitigation technique. The recorded values indicate a consistent and stable state, signifying that the switches in the SDN system are now more efficient in responding to pings and establishing connections in a faster manner, as compared to the latency values obtained prior to the implementation of the mitigation technique.

5.3.1 Results and Analysis for Experiment 3

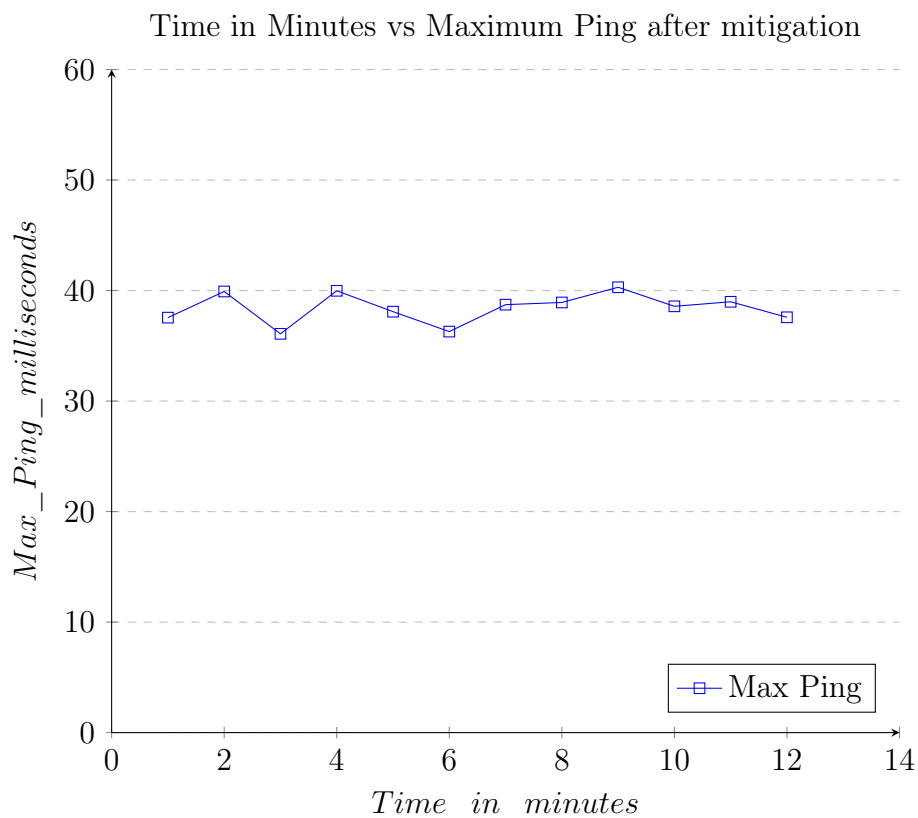
After obtaining the values of throughput, latency, memory usage and CPU usage in a similar way as mentioned in Experiment 1 and Experiment 2. These values are graphed. Throughput values are depicted in the form of bar graphs similar to Experiment 1. Latency, Memory usage and CPU usage are depicted in the form of line graphs. We compare the graphs obtained in Experiment 3 with the graphs obtained in Experiment 1 and Experiment 2 and see how the mitigation technique applied has changed the metrics of switches in Data Plane.

rtt / flow	min	avg	max	mdev
8556	0.074	0.545	37.550	3.727
9036	0.094	0.579	39.916	3.958
8054	0.051	0.502	36.086	3.578
9589	0.059	0.526	39.979	3.966
9291	0.097	0.518	38.092	3.778
8741	0.100	0.506	36.288	3.601
9342	0.087	0.513	38.725	3.841
8961	0.068	0.523	38.923	3.736
9785	0.094	0.532	40.3	3.921
8674	0.086	0.576	38.576	3.792
9276	0.092	0.534	38.985	3.826
8159	0.067	0.509	37.582	3.496

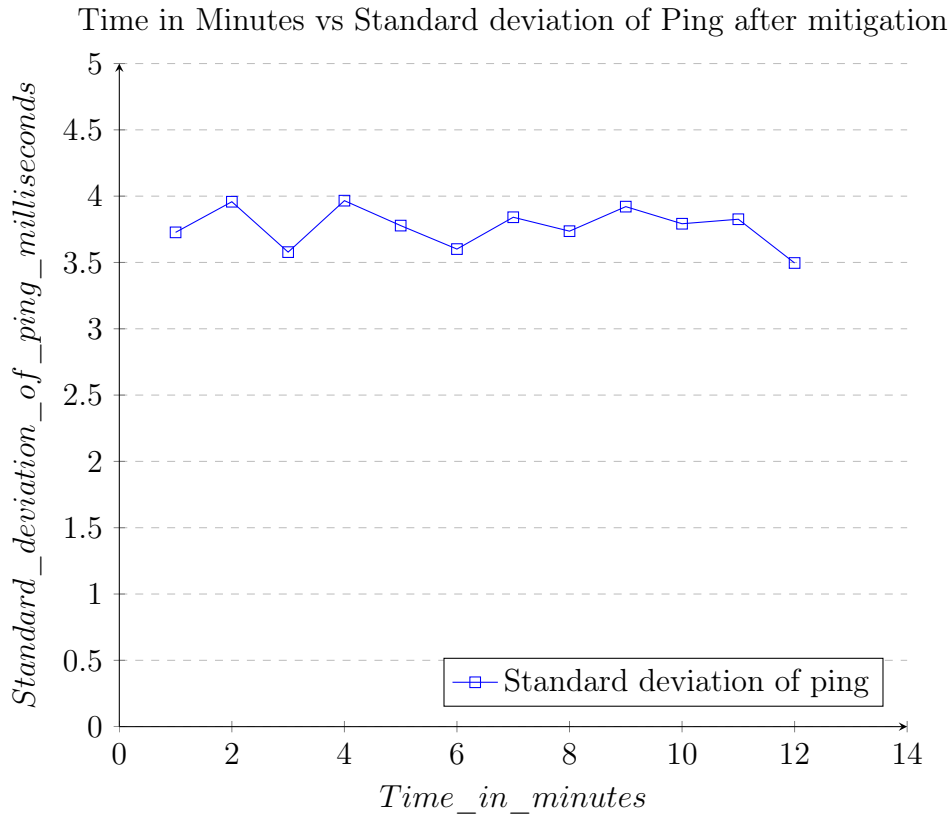
Table 5.6: Latency after mitigation



Comparing the graphs of "Flowrules vs Average ping" obtained in Experiment 1 with the graphs obtained after mitigation, we can see a difference in the trend of the graphs obtained for average ping values. The inclination of the graph that existed before mitigation has now stabilized and is no longer on the rise. We can now see the values of average ping to be in the range of 0.5 to 0.58 and does not increase beyond it and we have successfully reduced the effects on average ping after deploying the mitigation application on the Ryu controller.



Comparing the graphs of "Flowrules vs Max ping" obtained in Experiment 1 with the graphs obtained after mitigation, we can see a difference in the trend of the graphs obtained for maximum ping values. The inclination of the graph that existed before mitigation has now stabilized and is no longer on the rise. We can now see the values of maximum ping to be in the range of 37.5 to 40.5 and does not increase beyond it and we have successfully reduced the effects on maximum ping after deploying the mitigation application on the Ryu controller.



Comparing the graph of "Flowrules vs Standard deviation of ping" obtained in Experiment 1 with the graphs obtained after mitigation, we can see a difference in the trend of the graphs obtained for standard deviation of ping values. The inclination of the graph that existed before mitigation has now stabilized and is no longer on the rise. We can now see the values of standard deviation of ping to be in the range of 3.4 to 4 and does not increase beyond it and we have successfully reduced the effects on standard deviation of ping after deploying the mitigation application on the Ryu controller.



Figure 5.3: Throughputs of 3 different switches after mitigation

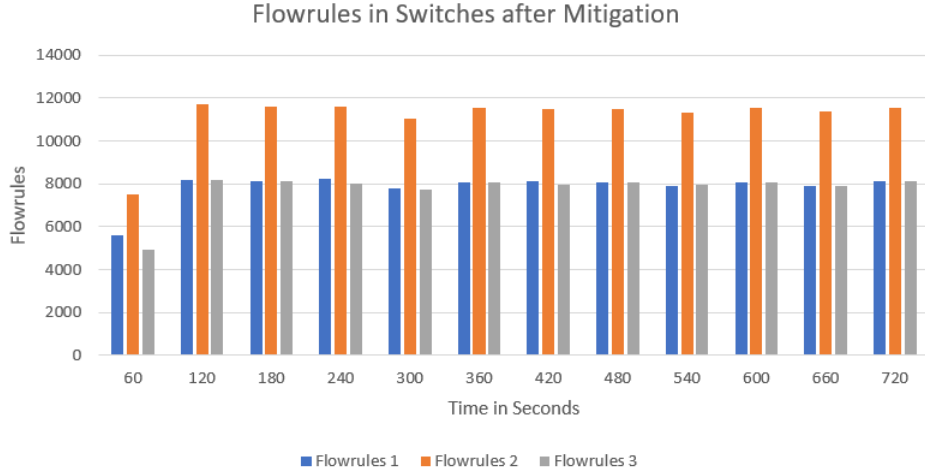
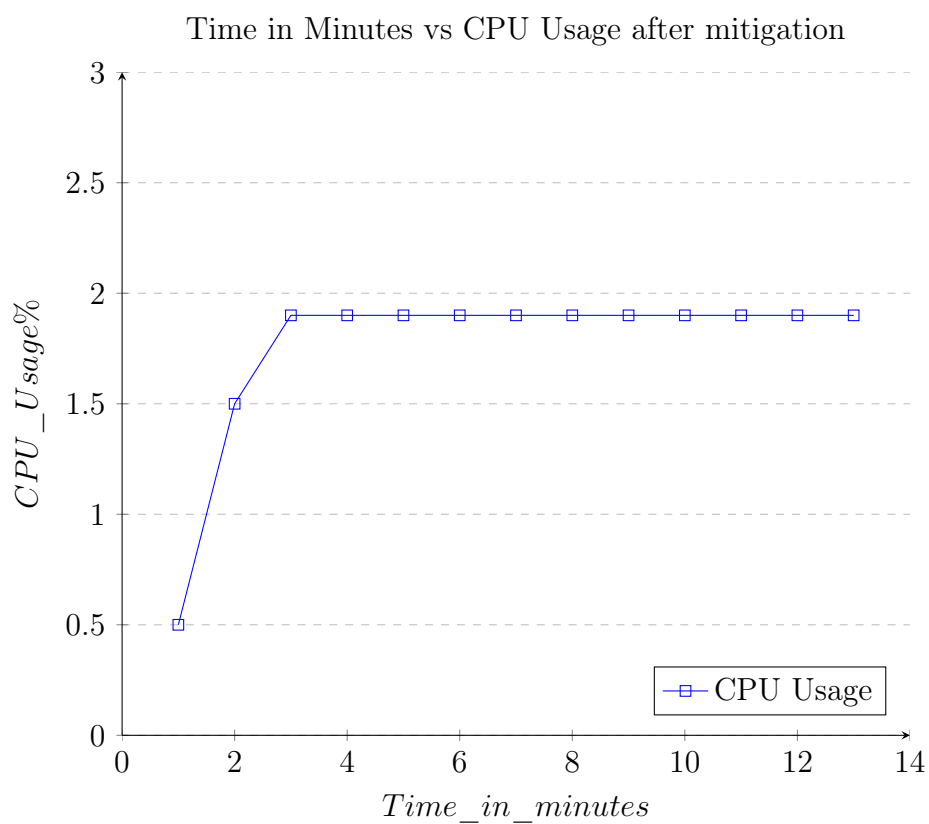
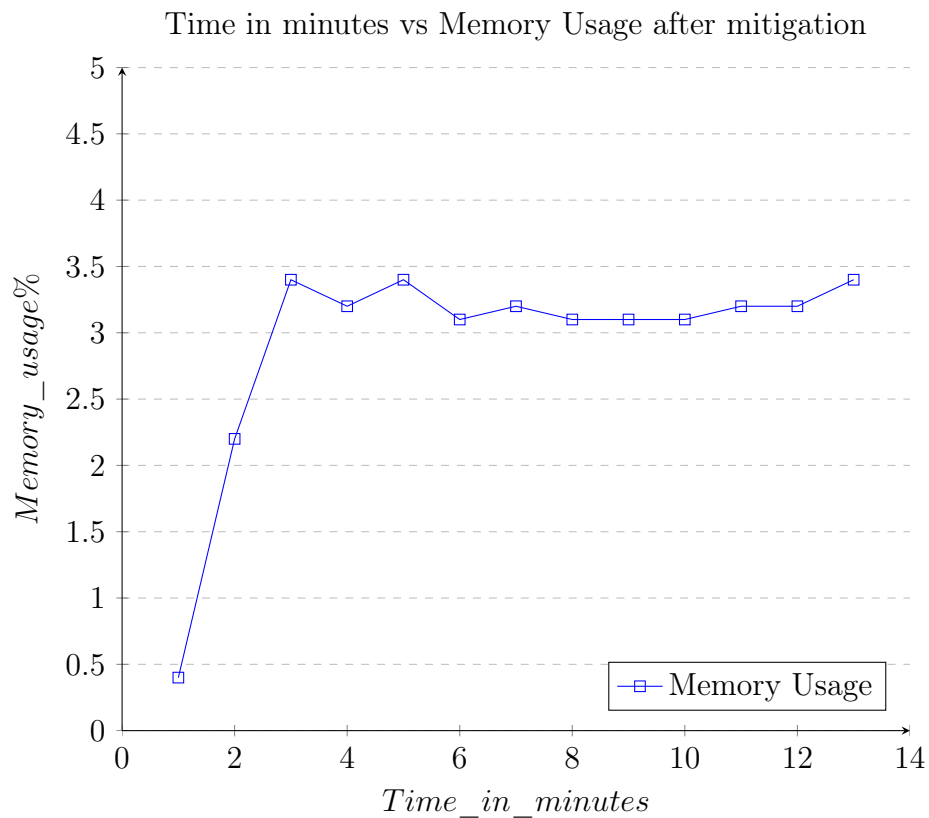


Figure 5.4: Flowrules with increasing time after mitigation

The bar graph in Figure 5.3 illustrates the throughput of three different switches within the Data Plane, calculated every 60 seconds using the method described in Experiment 1. A close examination of the graph reveals that the throughput values remain steady over the course of the experiment, indicating that the mitigation techniques utilized were successful in reducing the impact of the IP spoofing attack on the SDN switch throughput. After comparing the bar graphs in Figure 5.1 and 5.3, we can see that the throughput no longer decreases and that the throughput is now stable and also comparing Figure 5.2 and Figure 5.4, we can observe that the number of flow rules are not as large after the mitigation technique was applied.



We observe that the inclination of storage and CPU has been stabilized compared to the graph obtained in Experiment 2, and we can ascertain that the mitigation application has successfully reduced the effects caused by the attack scenario that

we used for our experiments.

After examining the graphs, it is apparent that the metrics of throughput, latency, CPU utilization, and memory usage of the switches have become consistent. This indicates that the mitigation technique we applied was effective in minimizing the effects of the IP spoofing and ICMP flooding attack that targeted our SDN network.

In this section, we discuss the findings of our research on mitigating IP spoofing attacks on the SDN data plane using Mininet as a simulation tool. We examine the ways in which attackers gain access to switches in an SDN network, the effects of IP spoofing attacks, and the limitations of our thesis.

- **Attackers' Access to Switches in SDN:**

In the assumed scenario, the attackers aim to exploit the accessibility of the SDN network by targeting the switches. The network is designed to provide public services, and as such, the servers are intended for public use, resulting in no ingress traffic. The attackers employ different methods, including exploiting software vulnerabilities, insider attacks, physical attacks, and social engineering, in their attempts to compromise the switches or gain unauthorized access to the SDN controller. Their objective is to exploit any potential weaknesses in the network's security infrastructure and gain control over the switches for malicious purposes.

In this specific thesis scenario, the attackers target unsuspecting users by sending them APK files containing malicious software. These files are designed to initiate an IP spoofing attack, specifically targeting the selected SDN network. When these APK files are installed on multiple users' devices, they collectively participate in a distributed denial-of-service (DDoS) attack. The distributed nature of the attack helps the attackers evade detection and minimize the risk of being identified, as the traffic can be traced back to various infected devices rather than directly to the attackers themselves.

- **Effects of IP Spoofing Attacks:**

The effects of IP spoofing attacks in the SDN network can be severe and wide-ranging. By impersonating legitimate clients or servers, attackers can bypass security measures and gain unauthorized access. This can lead to potential data breaches, information theft, and service disruption. IP spoofing attacks can also enable traffic hijacking, causing disruptions, service degradation, or even complete network outages. Additionally, attackers can launch Denial of Service attacks, overwhelming switches with a flood of spoofed IP packets, resulting in performance degradation and service unavailability.

- **Relation between Objectives and Experiments:**

In this thesis, we conducted three experiments to achieve the stated objectives. The test bed setup consisted of three linearly connected switches, with each

switch connected to three hosts. All the switches were connected to a controller, and the entire test bed was implemented using Mininet.

Objective 1: Determine how flooding of the flow-rules of an SDN switch affects the forwarding traffic in the network.

Experiment 1 was designed to achieve Objective 1. For this experiment, we selected two hosts in switch 1, three hosts in switch 2, and two hosts in switch 3 as attackers. These hosts sent IP spoofed packets to all other hosts in the network except the client and server. By gradually increasing the number of flow rules, we measured the ping values between the server and client. We calculate latency of the network using ping between client, server and total Throughput of SDN switches as number of flow rules increases in SDN switches. This experiment allowed us to observe the impact of flooding flow rules on the network's forwarding traffic. From the conclusions we obtained from Experiment 1, we were able to fulfill the first objective of our thesis.

Objective 2: Determine how storage, and CPU utilization of a switch are affected with increasing flow-rules.

Experiment 2 was conducted to address Objective 2. We used all hosts as attackers, and they randomly sent spoofed packets to all other hosts in the network. As the number of flow rules in the SDN switches increased, we measured the CPU utilization and memory utilization. This experiment provided insights into how the resource usage are affected as the flow rules increase. From the conclusions obtained from Experiment 2, we were able to see how IP spoofing attack affects the storage and CPU utilisation which was the second objective of our thesis.

Objective 3: Determine how monitoring applications can be used to mitigate the effect of IP Spoofing attacks on SDN switches.

Experiment 3 aimed to achieve Objective 3. We implemented an application in the Ryu controller, which actively monitored the flow rules in the switches. If the number of flow rules in a switch exceeded 1000, and if the packets in a flow rule were less than 2 with a time duration greater than 10, the application deleted the flow rule from the switch. This experiment demonstrated the potential of using monitoring applications to detect and mitigate IP spoofing attacks in SDN environments.

The effects observed on latency, throughput, memory usage and CPU usage were considerably less after the mitigation application was deployed and we were able to conclude that the mitigation application was successful in reducing the effects of IP spoofing attack on forwarding traffic and forwarding devices, which implies that the third objective of our thesis was fulfilled. We specifically highlight that the throughput metrics are now steady throughout, which was decreasing before mitigation application was deployed.

The selected architecture of three linearly connected switches was chosen to create a simplified yet realistic network environment. This setup allowed us to analyze the effects of IP spoofing attacks and flooding of flow rules in a controlled manner. By varying the number flow rules, we could observe the

impact on forwarding traffic, network performance, and the efficacy of monitoring applications. The simplicity of the architecture enabled us to focus on the specific objectives of the thesis and provided a foundation for meaningful experimentation and analysis.

- Limitations of the Thesis: While our research provides valuable insights into mitigating IP spoofing attacks and how IP spoofing attacks affect the SDN data plane, there are several limitations to consider:
 1. Simulation Environment: Our experiments are conducted using Mininet as a simulation tool. While Mininet is widely used for SDN simulations, it may not fully replicate real-world scenarios. The accuracy and generalizability of our results may be influenced by the assumptions and simplifications made in the simulation.
 2. Scalability: The experiments conducted in Mininet might have limitations regarding the scale and complexity of the network. Real-world SDN deployments often involve a larger number of switches, controllers, and network elements. Therefore, the scalability of our findings to larger, more intricate network environments should be further investigated.
 3. Real-Time Performance: Mininet simulations might not accurately reflect the real-time performance characteristics of SDN networks. The effects of IP spoofing attacks on switches in a production environment may differ from those observed in a simulated environment. Therefore, it is essential to validate our findings in real-world deployments to assess their practical implications.
 4. Limited Attack Scenarios: Our experiments primarily focus on the effects of IP spoofing attacks on latency, CPU, and memory utilization. However, there are other potential impacts of IP spoofing attacks, such as security breaches and compromised data integrity, which are not directly addressed in our research. Future work should explore a broader range of attack scenarios and their consequences.

In conclusion, our research highlights the importance of mitigating IP spoofing attacks in the SDN data plane. We have discussed the means by which attackers gain access to switches in an SDN network, the effects of IP spoofing attacks, and the limitations of our thesis. Addressing these limitations will contribute to the development of more robust and secure SDN infrastructures in the face of evolving

7.1 Conclusion

Our thesis paper examines the impact of an IP spoofing attack on the Data Plane of an SDN network. We conducted an attack scenario, using Mininet emulator on a Virtual Machine, where a large number of ICMP packets were sent to the SDN switches with spoofed IP addresses. This led to the creation of flow table entries for each ICMP packet and consequently, an increase in the number of flow rules in the SDN switches which was observed with the help of Wireshark. Through our analysis, we were able to observe the impact of this attack on the various metrics of the switches, and we plotted graphs to visualize the changes that occurred with an increasing number of flow rules.

Specifically, we found that the IP spoofing attack altered the latency and throughput metrics of the SDN switches. Our analysis indicates that the attack affected the forwarding traffic, causing delays in response times from the forwarding devices and reducing the throughput of the devices during the attack. Additionally, we observed that the IP spoofing attack successfully overwhelmed the storage of the switches, but it did not significantly increase the CPU usage. We can conclude that we were able to address the research questions 1 and 2 which we formulated and that we were able to fulfill objectives 1 and 2 of our thesis.

We also wanted to cover how the effects of IP spoofing can be minimized and in order to do this, we developed an application which runs on top of the Ryu controller, that actively monitors and removes unnecessary flow rules as a mitigation technique. The application was designed to detect and eliminate redundant or outdated flow rules, thereby optimizing the flow rule table within the switches.

After implementing our mitigation technique, we observed significant improvements in the network performance. The latency remained constant, indicating that our approach effectively managed the flow rules to prevent latency degradation. Additionally, the memory and CPU utilization were optimized, ensuring efficient resource allocation within the switches.

These results underscore the importance and effectiveness of our developed mitigation technique in addressing IP spoofing attacks on SDN switches. By dynamically managing flow rules, our application offers an intelligent and adaptive solution that maintains network performance while effectively countering IP spoofing threats. We can conclude that the results obtained after the mitigation answer the research question 3 we formulated and also fulfills objective 3 of our thesis.

It is worth noting that our research opens up avenues for further exploration. Fu-

ture work could focus on enhancing the efficiency and scalability of the application, as well as evaluating its performance in more complex network scenarios. Additionally, conducting real-world experiments and collaborating with industry experts can provide valuable insights for the practical implementation and deployment of our mitigation technique.

In conclusion, our research contributes to the field of SDN security by observing the effects of IP spoofing on SDN switches and providing a practical and effective approach to mitigate IP spoofing attacks on SDN switches. This work contributes finding the effects of IP spoofing attack on the forwarding devices and forwarding traffic, and to the overall security and reliability of SDN infrastructures against IP spoofing, and it has the potential to be further developed and adopted in real-world networks.

7.2 Future Work

In the thesis we conducted experiments on a basic SDN architecture to investigate the impact of IP spoofing on the switches of an SDN network. However, to obtain a better understanding of the effects of IP spoofing attacks, testing should be performed on larger networks with more complex topologies, however the trends that are obtained from our experimental results are going to stay true. Exploring alternative tools to Mininet emulator may also yield different results, since the results we obtained from our experiments are based on Mininet and they might differ in the case when a different tool is used. Several techniques can be used to mitigate the impact of flooding attacks, such as filtering flowtables, prioritizing flow rules, prioritizing flowtables, and setting meter tables. These techniques were not tested in the study due to time constraints and limited resources. Explore and evaluate more sophisticated mitigation techniques beyond the current application used in the Ryu controller, in order to mitigate other attacks like flooding attacks. Investigate machine learning algorithms or anomaly detection methods to enhance the accuracy and efficiency of mitigating IP spoofing attacks on the SDN data plane. Future research could focus on investigating attacks on other planes of the SDN network, apart from the data plane. Dynamic Flow Rule Management, deeper Network Traffic Analysis to identify IP spoofing attacks, Building resilient SDN architectures that withstand IP spoofing attacks, Performance Evaluation in Large Scale Networks, and Collaborative Security frameworks in SDN environments can be considered as possible future research directions.

However, the above represent important areas for future research to improve the security of SDN networks. Investigating these techniques and various types of flooding attacks can provide a more comprehensive understanding of the vulnerabilities and potential mitigation strategies for SDN networks.

References

- [1] Marc C. Dacier, Hartmut König, Radoslaw Cwalinski, Frank Kargl, and Sven Dietrich. Security challenges and opportunities of software-defined networking. *IEEE Security Privacy*, 15(2):96–100, 2017.
- [2] Rahil Gandotra and Levi Perigo. Sdnma: A software-defined, dynamic network manipulation application to enhance bgp functionality. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1007–1014. IEEE, 2018.
- [3] Sakir Sezer, Sandra Scott-Hayward, and Pushpinder Kaur Chouhan. Csit. *Queen’s University Belfast, Fraser, B., Lake, D., Cisco Systems, Finnegan, J., Viljoen, N., Netronome, Miller, M., Rao, N., Tabula.: Are we ready for SDN*, pages 36–43, 2013.
- [4] Qiao Yan, F Richard Yu, Qingxiang Gong, and Jianqiang Li. Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE communications surveys & tutorials*, 18(1):602–622, 2015.
- [5] Anup Dhamgaye and Nekita Morris. Survey on security challenges in vanet. *International Journal of Computer Science and Network*, 2, 02 2013.
- [6] Aayush Pradhan and Rejo Mathew. Solutions to vulnerabilities and threats in software defined networking (sdn). *Procedia Computer Science*, 171:2581–2589, 2020.
- [7] Christos Douligeris and Aikaterini Mitrokotsa. Ddos attacks and defense mechanisms: classification and state-of-the-art. *Computer networks*, 44(5):643–666, 2004.
- [8] Gao Shang, Peng Zhe, Xiao Bin, Hu Aiqun, and Ren Kui. Flooddefender: Protecting data and control plane resources under sdn-aimed dos attacks. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.
- [9] Hisham Al-Saghier. Attack on sdn infrastructure and security measures. *IN THE NAME OF ALLAH, THE MOST GRACIOUS, THE MOST MERCIFUL*, 6(2), 2019.
- [10] Marc C Dacier, Hartmut König, Radoslaw Cwalinski, Frank Kargl, and Sven Dietrich. Security challenges and opportunities of software-defined networking. *IEEE Security & Privacy*, 15(2):96–100, 2017.

- [11] Sandra Scott-Hayward, Sriram Natarajan, and Sakir Sezer. A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials*, 18(1):623–654, 2015.
- [12] Vidhi Bhavsar, Chairunnisa Sahrial, and Bhargavi Goswami. Security and performance evaluation of software defined network controllers against distributed denial of service attack. In *2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–8. IEEE, 2021.
- [13] Idris Zoher Bholebawa, Rakesh Kumar Jha, and Upena D Dalal. Performance analysis of proposed openflow-based network architecture using mininet. *Wireless Personal Communications*, 86(2):943–958, 2016.
- [14] Kostas Giotis, Christos Argyropoulos, Georgios Androulidakis, Dimitrios Kalogeras, and Vasilis Maglaris. Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments. *Computer Networks*, 62:122–136, 2014.
- [15] Marco Tiloca, Alexandra Stagkopoulou, and Gianluca Dini. Performance and security evaluation of sdn networks in omnet++/inet. *arXiv preprint arXiv:1609.04554*, 2016.
- [16] Shihabur Rahman Chowdhury, Md Faizul Bari, Reaz Ahmed, and Raouf Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–9. IEEE, 2014.
- [17] Ahmed F Abdullah, Fatty M Salem, Ashraf Tammam, and Mohamed H Abdel Azeem. Performance analysis and evaluation of software defined networking controllers against denial of service attacks. In *Journal of Physics: Conference Series*, volume 1447, page 012007. IOP Publishing, 2020.
- [18] Adam Bates, Kevin Butler, Andreas Haeberlen, Micah Sherr, and Wenchao Zhou. Let sdn be your eyes: Secure forensics in data center networks. In *Proceedings of the NDSS workshop on security of emerging network technologies (SENT’14)*, pages 1–7. Citeseer, 2014.
- [19] Alexandra Stagkopoulou. Simulative evaluation of security monitoring systems based on sdn, 2016.
- [20] Ayman M Bahaa-Eldin, Ebada Essam-Eldin ElDessouky, and Hasan Dağ. Protecting openflow switches against denial of service attacks. In *2017 12th International Conference on Computer Engineering and Systems (ICCES)*, pages 479–484. IEEE, 2017.
- [21] Nishat I Mowla, Inshil Doh, and KiJoon Chae. An efficient defense mechanism for spoofed ip attack in sdn based cdni. In *2015 International Conference on Information Networking (ICOIN)*, pages 92–97. IEEE, 2015.
- [22] Ranyelson N Carvalho, Lucas R Costa, Jacir L Bordim, and Eduardo AP Alchieri. Detecting ddos attacks on sdn data plane with machine learning. In *2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW)*, pages 138–144. IEEE, 2021.

- [23] Shuhua Deng, Xing Gao, Zebin Lu, and Xieping Gao. Packet injection attack and its defense in software-defined networks. *IEEE Transactions on Information Forensics and Security*, 13(3):695–705, 2017.
- [24] Duohe Ma, Zhen Xu, and Dongdai Lin. Defending blind ddos attack on sdn based on moving target defense. In *International Conference on Security and Privacy in Communication Networks: 10th International ICST Conference, SecureComm 2014, Beijing, China, September 24-26, 2014, Revised Selected Papers, Part I 10*, pages 463–480. Springer, 2015.
- [25] Andrés Felipe Murillo Piedrahita, Sandra Rueda, Diogo MF Mattos, and Otto Carlos MB Duarte. Flowfence: a denial of service defense system for software defined networking. In *2015 Global Information Infrastructure and Networking Symposium (GIIS)*, pages 1–6. IEEE, 2015.
- [26] Sungheon Lim, Seungnam Yang, Younghwa Kim, Sunhee Yang, and Hyogon Kim. Controller scheduling for continued sdn operation under ddos attacks. *Electronics Letters*, 51(16):1259–1261, 2015.
- [27] Ahmed M AbdelSalam, Ashraf B El-Sisi, and Vamshi Reddy. Mitigating arp spoofing attacks in software-defined networks. In *2015 25th International Conference on Computer Theory and Applications (ICCTA)*, pages 126–131. IEEE, 2015.
- [28] Sharon 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)*, pages 63–68. IEEE, 2014.
- [29] Hiep T Nguyen Tri and Kyungbaek Kim. Assessing the impact of resource attack in software defined network. In *2015 International Conference on Information Networking (ICOIN)*, pages 420–425. IEEE, 2015.
- [30] Prashant Kumar, Meenakshi Tripathi, Ajay Nehra, Mauro Conti, and Chhagan Lal. Safety: Early detection and mitigation of tcp syn flood utilizing entropy in sdn. *IEEE Transactions on Network and Service Management*, 15(4):1545–1559, 2018.
- [31] Moreno Ambrosin, Mauro Conti, Fabio De Gaspari, and Radha Poovendran. Lineswitch: Tackling control plane saturation attacks in software-defined networking. *IEEE/ACM Transactions on Networking*, 25(2):1206–1219, 2016.
- [32] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 413–424, 2013.
- [33] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.
- [34] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, page 6, 2010.

- [35] Juan Camilo Correa Chica, Jenny Cuatindioy Imbachi, and Juan Felipe Botero Vega. Security in sdn: A comprehensive survey. *Journal of Network and Computer Applications*, 159:102595, 2020.
- [36] Sumit Badotra. A review paper on software defined networking. *International Journal of Advanced Computer Research*, 8, 03 2017.
- [37] Martin Casado, Teemu Koponen, Scott Shenker, and Amin Tootoonchian. Fabric: a retrospective on evolving sdn. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 85–90, 2012.
- [38] Syed Asif Raza Shah, Sangwook Bae, Amol Jaikar, and Seo-Young Noh. An adaptive load monitoring solution for logically centralized sdn controller. In *2016 18th Asia-Pacific Network Operations and Management Symposium (AP-NOMS)*, pages 1–6. IEEE, 2016.
- [39] Scott Shenker, Martin Casado, Teemu Koponen, Nick McKeown, et al. The future of networking, and the past of protocols. *Open Networking Summit*, 20:1–30, 2011.
- [40] Open Data Center Alliance. Open data center alliance master usage model: Software defined networking rev 1.0. *Internet: <http://www.opendatacenter-alliance.org/docs/SoftwareDefinedNetworkingMasterUsageModelRev1.0.pdf>*, 2014.
- [41] Dan C Marinescu. *Cloud computing: theory and practice*. Morgan Kaufmann, 2022.
- [42] Nutaneer. Building blocks of sdn network, 2016.
- [43] Eric Mannie. Generalized multi-protocol label switching (gmpls) architecture. Technical report, IETF, 2004.
- [44] Yogesh Mundada, Rob Sherwood, and Nick Feamster. An openflow switch element for click. In *in Symposium on Click Modular Router*. Citeseer, 2009.
- [45] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. *ACM SIGCOMM Computer Communication Review*, 42(4):323–334, 2012.
- [46] Ramya Raghavendra, Jorge Lobo, and Kang-Won Lee. Dynamic graph query primitives for sdn-based cloudnetwork management. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 97–102, 2012.
- [47] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. Software-defined networking (sdn): a survey. *Security and communication networks*, 9(18):5803–5833, 2016.
- [48] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM computer communication review*, 38(3):105–110, 2008.
- [49] Seungwon Shin, Lei Xu, Sungmin Hong, and Guofei Gu. Enhancing network security through software defined networking (sdn). In *2016 25th international*

- conference on computer communication and networks (ICCCN)*, pages 1–9. IEEE, 2016.
- [50] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.
- [51] Sdn controller evaluation scheme and protection profile. <https://commoncriteria.github.io/pp/sdn-controller/sdn-controller-esr-paged.pdf>. Accessed on June 3, 2023.
- [52] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2014.
- [53] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008.
- [54] Fei Hu, Qi Hao, and Ke Bao. A survey on software-defined network and open-flow: From concept to implementation. *IEEE Communications Surveys & Tutorials*, 16(4):2181–2206, 2014.
- [55] Open Networking Foundation. Openflow switch specification, version 1.3.0. <https://www.opennetworking.org>, 2012. [Online; accessed 11-Apr-2023].
- [56] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.
- [57] Jorge Proença, Tiago Cruz, Edmundo Monteiro, and Paulo Simões. How to use software-defined networking to improve security-a survey. In *European Conference on Cyber Warfare and Security*, page 220. Academic Conferences International Limited, 2015.
- [58] E Reinecke. Mapping the future of software-defined networking, 2014.
- [59] Arne Schwabe and Holger Karl. Using mac addresses as efficient routing labels in data centers. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 115–120, 2014.
- [60] Ian F Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71:1–30, 2014.
- [61] Richard Wang, Dana Butnariu, and Jennifer Rexford. {OpenFlow-Based} server load balancing gone wild. In *Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 11)*, 2011.
- [62] Brandon Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree:

- Saving energy in data center networks. In *Nsdi*, volume 10, pages 249–264, 2010.
- [63] R Alimi, R Penno, and Y Yang. Alto protocol. internet engineering task force, 2013.
- [64] Aditya Gudipati, Daniel Perry, Li Erran Li, and Sachin Katti. Softran: Software defined radio access network. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 25–30, 2013.
- [65] Manu Bansal, Jeffrey Mehlman, Sachin Katti, and Philip Levis. Openradio: a programmable wireless dataplane. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 109–114, 2012.
- [66] Kok-Kiong Yap, Masayoshi Kobayashi, Rob Sherwood, Te-Yuan Huang, Michael Chan, Nikhil Handigol, and Nick McKeown. Openroads: Empowering research in mobile networks. *ACM SIGCOMM Computer Communication Review*, 40(1):125–126, 2010.
- [67] Hassan Ali-Ahmad, Claudio Cicconetti, Antonio de la Oliva, Martin Dräxler, Rohit Gupta, Vincenzo Mancuso, Laurent Roullet, and Vincenzo Sciancalepore. Crowd: an sdn approach for densenets. In *2013 second European workshop on software defined networks*, pages 25–31. IEEE, 2013.
- [68] Srikanth Sundaresan, Walter De Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. Broadband internet performance: a view from the gateway. *ACM SIGCOMM computer communication review*, 41(4):134–145, 2011.
- [69] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.
- [70] Syed Akbar Mehdi, Junaid Khalid, and Syed Ali Khayam. Revisiting traffic anomaly detection using software defined networking. In *International workshop on recent advances in intrusion detection*, pages 161–180. Springer, 2011.
- [71] Junho Suh, Ted Taekyoung Kwon, Colin Dixon, Wes Felter, and John Carter. Opensample: A low-latency, sampling-based measurement platform for commodity sdn. In *2014 IEEE 34th International Conference on Distributed Computing Systems*, pages 228–237. IEEE, 2014.
- [72] Rodrigo Braga, Edjard Mota, and Alexandre Passito. Lightweight ddos flooding attack detection using nox/openflow. In *IEEE Local Computer Network Conference*, pages 408–415. IEEE, 2010.
- [73] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.
- [74] Kai Wang, Yaxuan Qi, Baohua Yang, Yibo Xue, and Jun Li. Livesec: Towards effective security management in large-scale production networks. In *2012 32nd*

- International Conference on Distributed Computing Systems Workshops*, pages 451–460. IEEE, 2012.
- [75] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for openflow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 121–126, 2012.
- [76] Krishna Kant. Data center evolution: A tutorial on state of the art, issues, and challenges. *Computer Networks*, 53(17):2939–2965, 2009.
- [77] Ahsan Arefin, Vishal K Singh, Guofei Jiang, Yueping Zhang, and Cristian Lumezanu. Diagnosing data center behavior flow by flow. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 11–20. IEEE, 2013.
- [78] Grotto Networking. Sdn overview, 2021.
- [79] Aria Zhu. What is an openflow switch?, 2018.
- [80] Fouaz Bouguerra. Data centre networking: what is ovs? <https://ubuntu.com/blog/data-centre-networking-what-is-ovs>. Accessed: 2021-11-30.
- [81] Fei Hu, Qi Hao, and Ke Bao. A survey on software-defined network and open-flow: From concept to implementation. *IEEE Communications Surveys Tutorials*, 16(4):2181–2206, 2014.
- [82] Angel Leonardo Valdivieso Caraguay, Lorena Isabel Barona Lopez, and Luis Javier Garcia Villalba. Evolution and challenges of software defined networking. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–7. IEEE, 2013.
- [83] Akram Hakiri, Aniruddha Gokhale, Pascal Berthou, Douglas C Schmidt, and Thierry Gayraud. Software-defined networking: Challenges and research opportunities for future internet. *Computer Networks*, 75:453–471, 2014.
- [84] Madhukrishna Priyadarsini and Padmalochan Bera. Software defined networking architecture, traffic management, security, and placement: A survey. *Computer Networks*, 192:108047, 2021.
- [85] Qiao Yan and F Richard Yu. Distributed denial of service attacks in software-defined networking with cloud computing. *IEEE Communications Magazine*, 53(4):52–59, 2015.
- [86] KS Vanitha, SV Uma, and SK Mahidhar. Distributed denial of service: Attack techniques and mitigation. In *2017 International Conference on Circuits, Controls, and Communications (CCUBE)*, pages 226–231. IEEE, 2017.
- [87] Peng Zhang, Huanzhao Wang, Chengchen Hu, and Chuang Lin. On denial of service attacks in software defined networks. *IEEE Network*, 30(6):28–33, 2016.
- [88] Open Networking Foundation. Openflow switch specification, version 1.5.1, 2014.

- [89] Mehdiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Software-defined networking security: pros and cons. *IEEE Communications Magazine*, 53(6):73–79, 2015.
- [90] Arvind Prasad and Shalini Chandra. Defending arp spoofing-based mitm attack using machine learning and device profiling. In *2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pages 978–982. IEEE, 2022.
- [91] Tasnuva Mahjabin, Yang Xiao, Guang Sun, and Wangdong Jiang. A survey of distributed denial-of-service attack, prevention, and mitigation techniques. *International Journal of Distributed Sensor Networks*, 13(12):1550147717741463, 2017.
- [92] Keshav Jindal, Surjeet Dalal, and Kamal Kumar Sharma. Analyzing spoofing attacks in wireless networks. In *2014 Fourth International Conference on Advanced Computing & Communication Technologies*, pages 398–402. IEEE, 2014.
- [93] R Bani-Hani and Zaid Al-Ali. Syn flooding attacks and countermeasures: a survey. In *Proceedings of ICICS*, 2013.
- [94] Jianli Pan and Raj Jain. A survey of network simulation tools: Current status and future developments. *Email: jp10@cse.wustl.edu*, 2(4):45, 2008.
- [95] Faris Ketikci and Shavan Askar. Emulation of software defined networks using mininet in different simulation environments. In *2015 6th International Conference on Intelligent Systems, Modelling and Simulation*, pages 205–210. IEEE, 2015.
- [96] Jiaqi Yan and Dong Jin. Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, pages 1–7, 2015.
- [97] Bob Lantz and Brian O’Connor. A mininet-based virtual testbed for distributed sdn development. *ACM SIGCOMM Computer Communication Review*, 45(4):365–366, 2015.
- [98] Computerhope. Virtualbox. <https://www.computerhope.com/jargon/v/virtualbox.htm>. Accessed: 2019-03-09.
- [99] SDxCentral. Ryu controller. <https://www.sdxcentral.com/networking/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/what-is-sdn-controller/openflow-controller/what-is-ryu-controller/>. Accessed: 2016-06-23.
- [100] Philippe Biondi. Scapy. <https://scapy.readthedocs.io/en/latest/introduction.html>.
- [101] University of Sussex. Putty. <https://www.sussex.ac.uk/its/services/software/owncomputer/putty>. Accessed: 2013-01-14.

- [102] CompTIA. Wireshark. <https://www.comptia.org/content/articles/what-is-wireshark-and-how-to-use-it>.
- [103] Colin Harrison. Xming, 2021.
- [104] OSRG. Ryu. <https://github.com/osrg/ryu>, 2021. Accessed: September 27, 2021.
- [105] SDxCentral. Sdn controller. <https://www.sdxcentral.com/networking/sdn/definitions/what-is-sdn-controller/>, 2021. Accessed: September 27, 2021.
- [106] Openflow switch specification version 1.3.5. https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html. Accessed: 2021-05-18.
- [107] shivaanirudh. mininetryu. https://github.com/shivayadavjabbu/ryu_mininet_ipspoofing.

Appendix A

Supplemental Information

Codes refer [107]

