INTEGRATION OF MTD SYSTEM IN SDN ARCHITECTURE

B.E. (CS) PROJECT REPORT

by

Sheikh Muhammad Farjad

Department of Computer and Information Systems Engineering

NED University of Engineering & Technology

Karachi-75270

INTEGRATION OF MTD SYSTEM IN SDN ARCHITECTURE

B.E. (CS) PROJECT REPORT

Project Group:

SHEIKH MUHAMMAD FARJAD	CS-059
SYEDA MARIUM FAHEEM	CS-099
UMAMA AHMED	CS-038

BATCH: 2016 – 2017

Project Advisor(s):

Dr. Muhammad Ali Ismail (Internal Advisor)

October 2020

Department of Computer and Information Systems Engineering

NED University of Engg. & Tech.,

Karachi-75270

ABSTRACT

SDN or Software-Defined Networking is relatively a new dimension in the domain of networking. In conventional networking approach, every networking device or "hop" manages the task of forwarding and controlling the packets. This conventional approach of networking degrades the efficiency of networking infrastructure because every hop has to manage and update its routing information. SDN is an approach that is designed to overcome this degradation of network infrastructure. SDN tackles with the degradation problem by separating forwarding plane from the control plane. Forwarding plane is managed by every networking device and all these networking devices are connected to a centralized controller, the task of control plane is managed by this centralized controller. This SDN approach of separating control plane from forwarding plane eliminates the burden of managing control plane from the networking devices that are interconnected in a same network. The centralized architecture of SDN has helped in mitigating network degradation, but it has also introduced the vulnerability that stems from the centralized approach. The most notable vulnerability is ***single point of failure (SPOF)****. This project aims to tackle and overcome this vulnerability for securing SDN architecture from any sort of cyber-attack.*

ACKNOWLEDGEMENTS

We are highly indebted to our families, friends, and loved ones for supporting us during the tough time of working on one of the complicated projects of our undergraduate life. We are also thankful to *NED University of Engineering & Technology* and *Department of Computer & Information Systems Engineering* for giving us the opportunity to work on such critical project under the supervision of **Dr. Muhammad Ali Ismail.**

We are thankful to our internal advisor **Dr. Muhammad Ali Ismail** for his consistent effort to support and guide us. The completion of this project was not possible without his technical guidance and consistent cooperation.

Table of Contents

CHAPTER 1

Introduction

With the advancement in the domain of information technology, mankind has become equipped with modern technologies for attaining safety and comfort. The technology sector has been shaping our society for a long time. The most revolutionary achievement is the sector of technology is the invention of the Internet. The invention of the Internet resulted in chains of further inventions. The Internet has shortened the distance gap among people and it facilitated the sharing of information also. The advancement of the Internet did not only facilitate people, but it also opened an opportunity for malicious actors to exploit it for their evil incentives. This was the point that resulted in the emergence of cybersecurity. In the last few decades, a new alternative was presented as a better and more efficient alternative to conventional networking. This new alternative was named Software-Defined Networking (SDN). The newly emerged domain of SDN purports to overcome the efficiency gap that exists in conventional networking.

1.1 Primary Aim:

This report is a descriptive document that summarizes all the aspects and perspectives of the final year project titled “Integration of MTD System in SDN Architecture”. This descriptive report sheds light upon software-defined networking, moving target defense, load-balancing, security flaws of software-defined networking, and performance gap in conventional networking. This report also addresses the tools and different modules used in the implementation of this final year project. This document addresses the final year project. It also highlights the implementation of modules and applications of the project. The most fundamental purpose of this project is to make the architecture of software-defined networking secure while not leaving any detrimental impact on its performance.

1.2 Need for SDN Security:

The implementation of SDN proves to be more efficient than conventional networking. Besides surmounting the performance gap, SDN also opens the chances of vulnerabilities, which can be exploited by attackers for penetrating the network. Apparently, SDN has few architectural issues, which can compromise the entire infrastructure of the organizations. This necessitates the need for SDN security.

1.3 Primary Motivation:

Currently, SDN has become the mainstay for cloud computing environments and data centers. More vendors are migrating towards SDN from conventional networking. There arises the exigency of implementing a security system in SDN. The existing security solutions often entail performance degradation as an integral component. The entailment of performance degradation results in the failure of the primary goal of implementing SDN, that is, high performance. This project aims to provide an efficient security solution, which not only increases the security layers but also the performance of corresponding SDN environment.

1.4 Main Objectives of the Project:

Following are the main objectives of the project:

- To set up a software-defined networking environment.
- To implement software-defined networking.
- To test the implementation with multiple topologies.

- To study and test different controllers.
- To integrate moving target defense in software-defined networking**.**

1.5 Significance of the SDN Security:

The domain of software-defined networking is revolutionizing the field of conventional networking. The underlying idea of centralizing the networking intelligence into a single point exceedingly reduces the overall load of a network. This idea provides preferences to software-defined networking over conventional networking.

The networking mechanism is the backbone of technology in the 21st century and a large number of organizations and data centers are aiming to replace conventional networking with a software-defined networking paradigm. This results in the exigency to strengthen the security of a software-defined networking environment. Otherwise, the use of SDN for performance can result in a backfire for the corresponding organizations. The security solutions for SDN should have no impact on performance. For instance, the existing security solutions depend upon the randomization of parameters, which often halt the running service for a short span of time. This little interruption in the service renders a huge impact on the performance of an organization. This is why there is a dire need for robust and performance-independent solutions for ensuring security in SDN.

1.6 Main Deliverables:

Following are the main deliverables of the project:

- Simple SDN implementation
- Virtual environment of the project

- MTD-enabled SDN implementation
- Comparative analysis of SDN controllers
- Load-balancing implementation
- Logs management terminal

1.7 Instruction for Readers:

The report is distributed in eight chapters. Each chapter elucidates the different aspects of the project. The following points briefly summarize the contents of chapters:

- Chapter 2 addresses the literature review of the project and it sheds light on the related work of researchers, who aimed to work for ensuring security in the SDN environment.
- Chapter 3 discusses the core architecture of software-defined networking and potential flaws that are susceptible to be exploited by the attackers.
- Chapter 4 discusses the technique of moving target defense used in the project.
- Chapter 5 discusses the evaluation of the performance of multiple SDN controllers for the implementation of the project.
- Chapter 6 discusses all the preliminary tools and utilities used in the project.
- Chapter 7 covers the final implementation of the project and it discusses all the modules and aspects of the final implementation.
- Chapter 8 covers the conclusion and future scope of the project.
- The appendix section contains miscellaneous screenshots.

- The last section contains the references used in the report.

CHAPTER 2

Literature Review

2.1 Securing Software-Defined Networking:

The infrastructure of software-defined networking is vulnerable to severe security threats. A single bug in SDN implementation can result in complete disaster for the corresponding organization. The researchers have proposed multiple techniques for ensuring the security of any SDN-based environment. Kang et al. [1] proposed VOGUE framework which generates automated permission and verification techniques for authenticating the users within SDN environment. Ruaro et al. [2] proposed SDN security framework for Many Core Systems-on-Chip (MCSoC) which implements security in SDN environment by following three main steps that ensure the utilization of one packet-switching and a set of circuit-switching for the accomplishment of the task.

2.2 Moving Target Defense:

The primary incentive of any MTD technique is to prevent attackers from inspecting the real parameters of the victim’s system or environment. This prevention implies that the reconnaissance stage of conduction the attack is made to resort to failure, because the information gained by the attacker is rendered useless due to changing parameters caused by MTD. There are different ways of implementing MTD which have been presented in different research papers. Fundamentally, the MTD techniques are classified in terms of: shuffling mechanism, diversity, and redundancy.

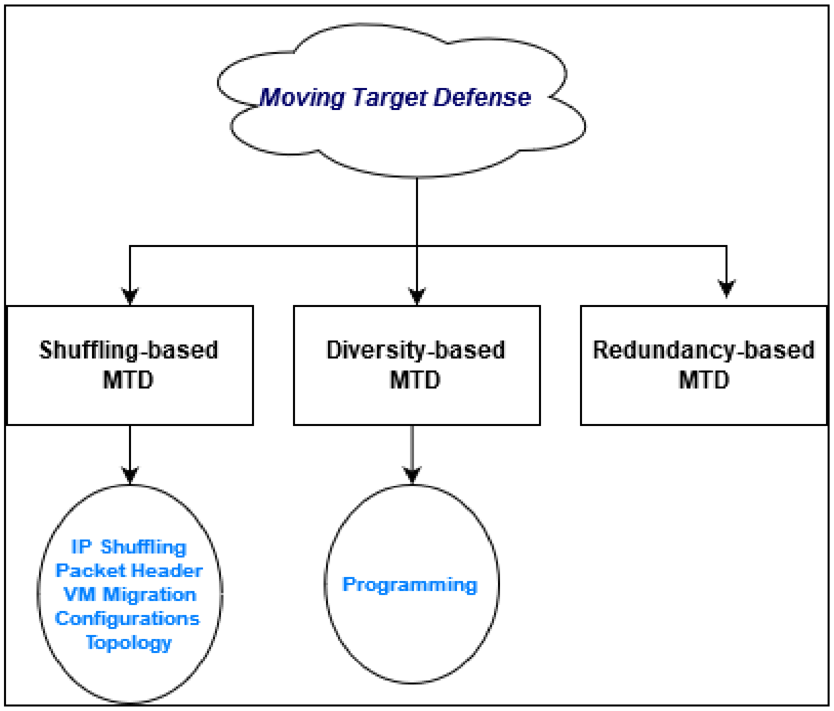


Figure 2.1: Categorization of moving target defense techniques.

2.2.1 Shuffling:

MTD technique based on shuffling is the most prominent MTD technique, which randomizes the specific parameters by shuffling them. The selection of parameters to shuffle also depends on the requirements of the system. The shuffling mechanism can target IP address [3, 4, 5, 6], packet headers [7], virtual machine migration [8], and service configuring parameters [9, 10]. The shuffling-based MTD can also use randomized topologies for the transmission of packets [11, 12].

2.2.2 Diversity:

The MTD technique based on diversity achieves a polymorphic behavior for thwarting the attackers from conducting reconnaissance of the network or system. In diversity-based MTD, the polymorphic behavior is adopted by implementing the same functionalities in different manners

[13]. For instance, the diversity based on programming languages reduces the code injection vulnerabilities that are inherent to specific programming languages [14].

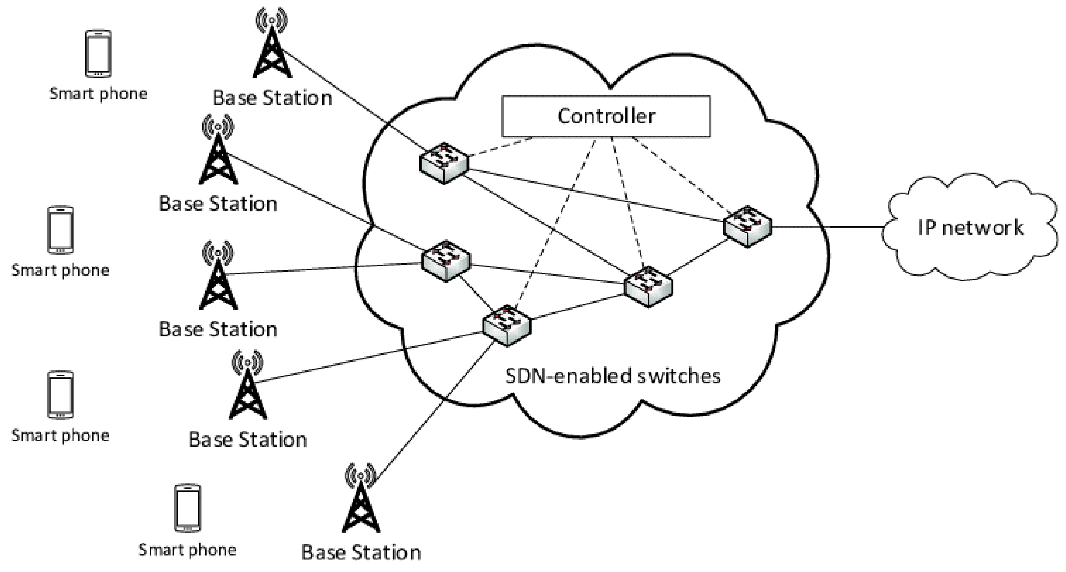
2.2.3 Redundancy:

The another way to implement MTD technique is to utilize redundancy concept. In redundancy-based MTD, multiple copies of components are made for distributing the control and maintaining the integrity of data [15]. The redundancy-based MTD technique is mostly used as complimentary part of shuffling-based and diversity-based techniques.

Besides aforementioned categories of MTD, the researchers also developed several novel techniques for accomplishing MTD in the domain of SDN. For instance, there are techniques like random host mutation [16], port hopping [17], and network topology shuffling [12], which hugely leverage the core architectural features of SDN for implementing MTD in the SDN environment. Luo et al. [18] utilized SDN-based honeypots and MTD for defending against distributed denial-of-service (DDoS) attacks. All the techniques for implementing MTD often result in some sort of performance degradation. Our project endeavors to eliminate performance degradation by utilizing load-balancing mechanism for enforcing moving target defense in software-defined networking environment.

CHAPTER 3

Software-Defined Networking



3.1 Introduction:

Software-Defined Networking (SDN) an emerging domain of computer science that aims to revolutionize the infrastructure of conventional networking. The virtualization technology has further facilitated the domain of SDN and this is why it is being widely adopted by the cloud vendors and data centers. OpenFlow is the underlying protocol that is used in SDN for establishing the communication among different components of an SDN implementation.

3.2 Performance Gap in Conventional Networking:

The conventional networking paradigm predominantly relies upon networking devices like routers and switches, these networking devices are equipped with the intelligence for a decision-making process that is responsible for directing the incoming packets towards their respective destination hosts or systems. Each networking device has to manage and maintain the records of the routing table, routing algorithms, access list, etc. In technical terms, each networking device is equipped with data and control planes. The data place executes the task of forwarding the packets and the control plane manages the routing of the respective packets. As the network grows, the number of networking devices also increases, and each device is equipped with the load to manage. This high number of decision-making networking devices directly impacts the performance of overall network management because excessive resources will be consumed by each networking device resulting in deterioration of the performance. The illustration of conventional networking is shown in **Figure 3.1.**

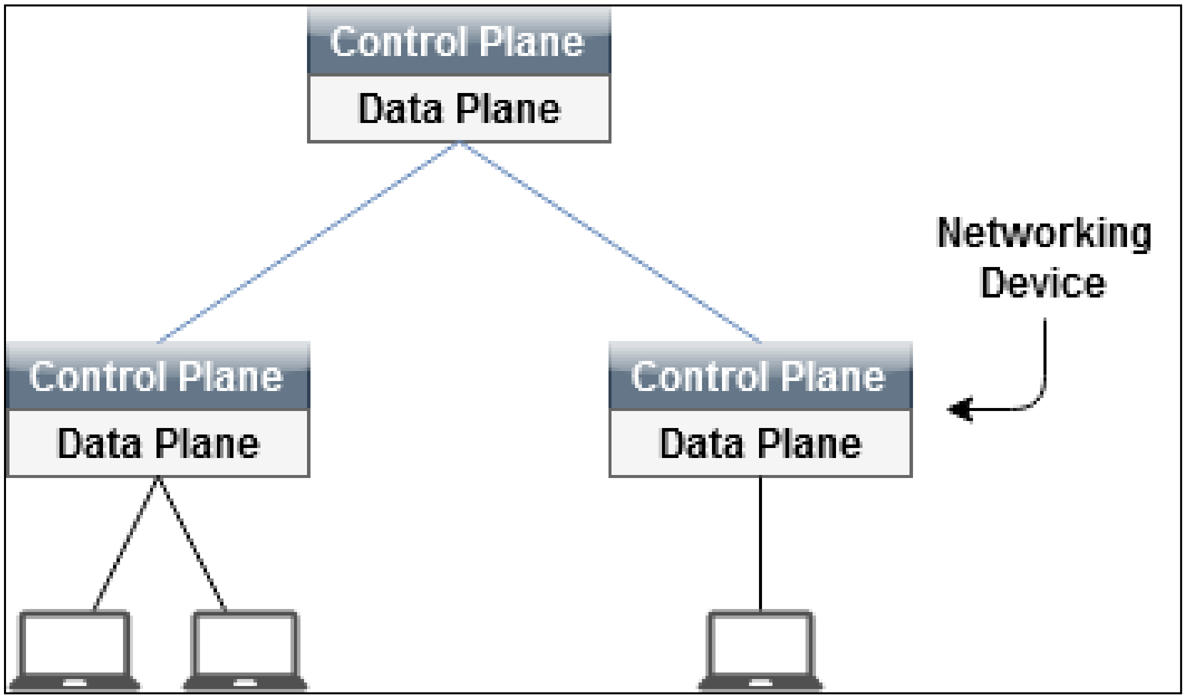


Figure 3.1: Conventional networking infrastructure.

3.3 Fundamental Principle:

The emerging field of SDN predicates on the principle of centralizing networking intelligence. The aforementioned performance gap in the conventional networking paradigm is overcome by employing SDN. The SDN utilizes the centralizing principle for merging all the ‘intelligence’ of the network into a single object. This approach highly reduces the overall load of managing packet routing and other management tasks. The illustration of SDN networking is shown in **Figure 3.1** and a comparison with **Figure 3.2** reflects that the networking device in SDN is enabled with only a data plane that is responsible only for forwarding packets and there exists a centralized control plane in the controller that manages the routing on the behalf of networking devices which behave as forwarding devices in SDN paradigm.

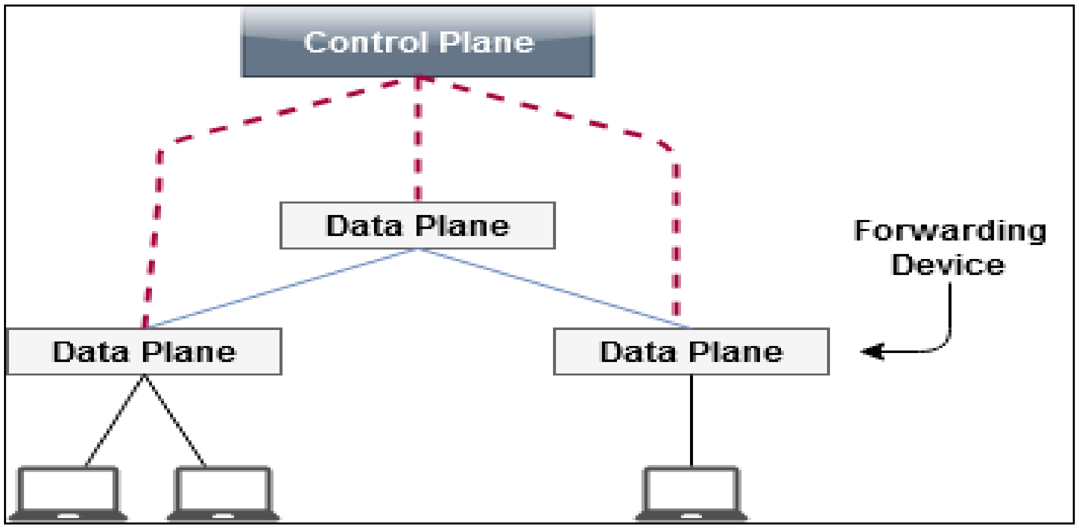


Figure 3.2: Software-defined networking infrastructure.

3.4 Core Architecture:

The technology of software-defined networking is based on layered architecture and it follows three-tier implementation as illustrated in **Figure 3.2**. Each layer is responsible for accomplishing its dedicated tasks. The following are the fundamental layers of SDN: application layer, control layer, and infrastructure layer.

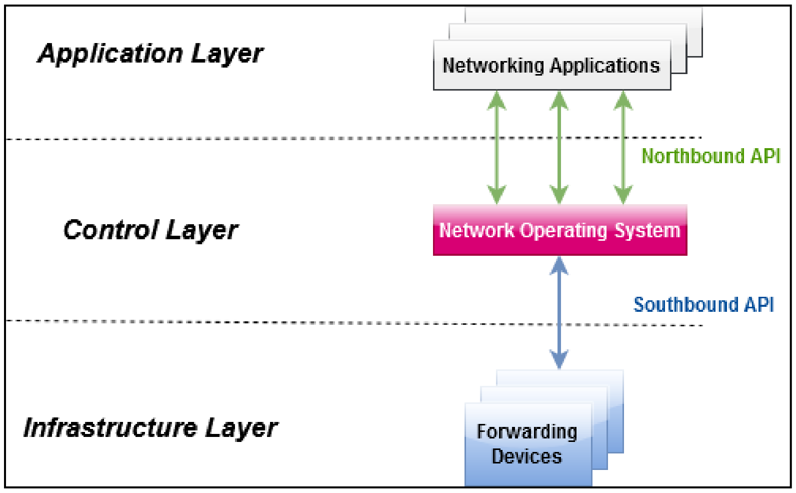


Figure 3.3: 3-tier architecture of SDN.

3.4.1 Application Layer

The application layer is the highest layer in SDN architecture. At this layer, the networking applications are developed and managed for interacting with the end-users. The primitive purpose of this layer is to link networking applications with an SDN controller that resides in the control layer. The Application Programming Interfaces (APIs) used for communication between the application layer and the control layer are called **Northbound APIs**.

3.4.2 Control Layer

At this layer, the SDN controller resides which is also termed as Network Operating System (NOS). This layer is responsible for routing packets to their respective destinations. The load of packet routing and caching the flow-entries is managed by this layer. The APIs used for establishing communication between the control plane and infrastructure plane are called **Southbound APIs**.

3.4.3 Infrastructure Layer

This is the lowest layer of SDN architecture, which is responsible for communicating with the networking devices in the SDN environment. The control layer sends routing information to these forwarding devices, which direct the respective packets to their corresponding destinations.

3.5 Potential Threats

The domain of SDN employs the centralization principle for improving the performance, but it also results in potential threats, which can compromise the security of SDN infrastructure. The following are the major potential threats in the SDN environment:

3.5.1 Controller-based Threat

The controller-based threat arises due to the centralization of the control into single entities. The ill-intentioned access to the SDN controller can destroy the entire SDN infrastructure because the attacker can have control of the entire infrastructure from the controller.

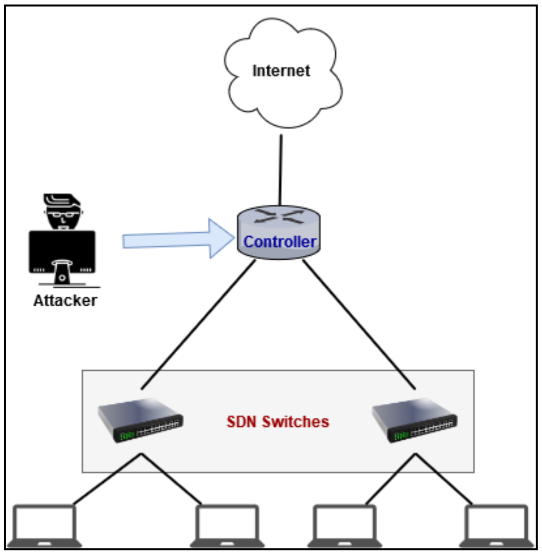


Figure 3.4: Controller-based threat.

3.5.2 Server-based Threat

Instead of targeting the controller, the attacker can also be interested in targeting the specific server residing in the paradigm of SDN. The server can be any connected device running the services.

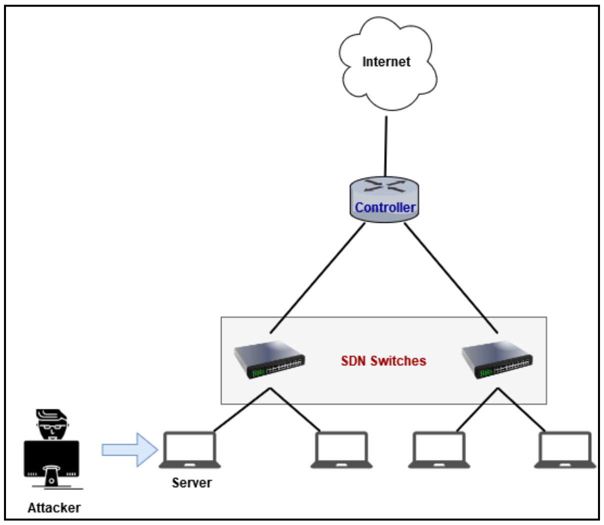


Figure 3.5: Server-based threat.

This project utilizes the technique of moving target defense for surmounting the server-based threat.

CHAPTER 4

Moving Target Defense

4.1 Introduction:

The Moving Target Defense (MTD) refers to a technique that is employed for securing the system or network from attackers. This technique involves the practice to assign dynamic parameters to the system, which makes it difficult for the attackers to penetrate. The MTD technique is widely used by network security professionals working with the conventional networking paradigm. The MTD technique shows tremendous effectiveness in protecting the system or network from attackers [19, 20, 21].

4.2 Cyber Kill Chain:

Any execution of the cyberattack performs follows certain stages and this collection of stages is termed as “Cyber Kill Chain” and it is illustrated in **Figure 4.1**.

Each stage of the cyber kill chain aims to execute its dedicated function. The most primary stage of any cyber kill chain is reconnaissance. In the reconnaissance stage, the attacker uses different tools and techniques for gathering information about the potential victim. After gathering all the information, the attacker proceeds to other stages of the cyber kill chain to execute the attack successfully. In the intrusion stage, the attacker penetrates the security of the system. In the exploitation stage, the vulnerabilities of the victim are scanned for determining the most suitable exploits. In the exploitation stage, the exploit is executed that utilizes the vulnerabilities to raise the privilege. After privilege escalation, the attacker executes the lateral movement stage, obfuscation, and denial of service. Once all the stages have been completed, the attacker finds a secure way to step out of the system without leaving any remains for forensic investigation.

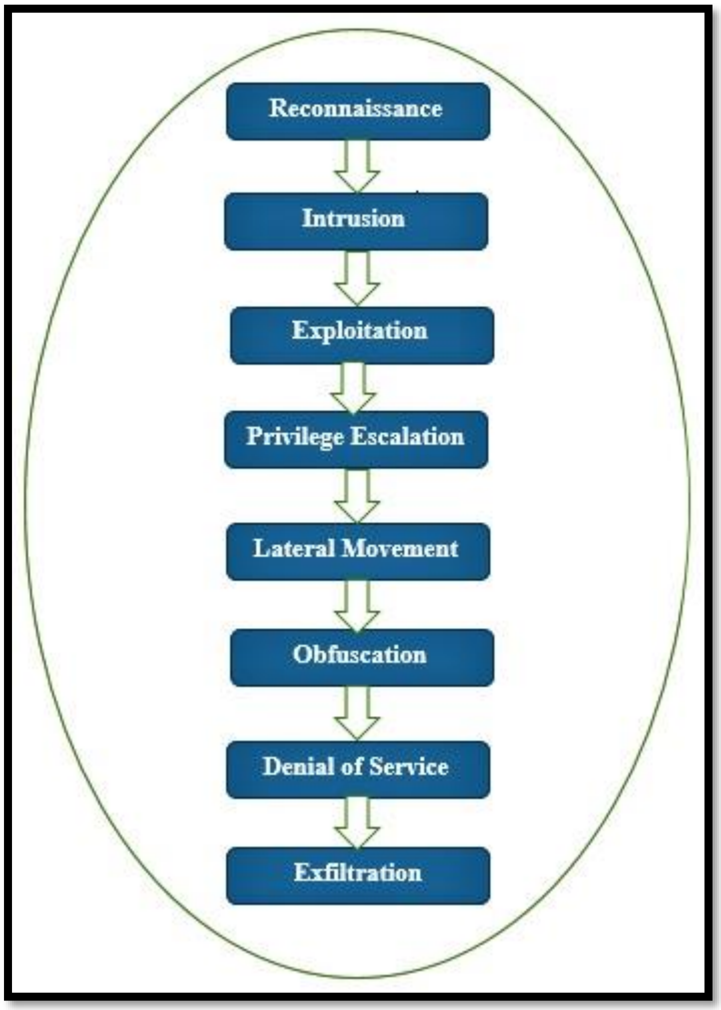


Figure 4.1: Cyber kill chain

4.3 Primary Objective of MTD:

The primary objective of the MTD system is to prevent the attacker from successfully executing the stage of reconnaissance. The MTD system ensures the failure of the reconnaissance stage of the cyber kill chain by providing the attacker with false information about the system or the network.

4.4 A Novel Technique of MTD:

There are multiple ways of implementing MTD [19, 20, 21]. The literature review section has already covered a few existing approaches for implementing MTD, but it has also been mentioned in previous sections that existing MTD techniques result in the degradation of performance. In order to maintain the performance, we proposed a novel technique for implementing the MTD system in the SDN environment. Our proposed technique uses a load-balancing mechanism for attaining moving target defense in the SDN environment.

The technique of load balancing is used for distributing the load from the client-side in the pool of servers. Conventional networking paradigms have been using the load-balancing mechanism in data centers and high-load enterprises, where a huge number of clients send requests to the server. This novel idea of utilizing load balancing for integrating the MTD system in SDN architecture is an efficient approach, which not only protects the system from intruders but also augments the current performance of any SDN deployment.

4.5 Load Balancing:

The load balancing distributes the load of incoming requests among the pool of dedicated servers. The pattern for directing the requests to the servers follows scheduling algorithms and these scheduling algorithms decide the destination server for each request. The component, which ensures the implementation of scheduling algorithms and distributes the request among the pool of servers based on that scheduling algorithm, is called Load Balancer. **Figure 4.2** illustrates the load balancing mechanism.

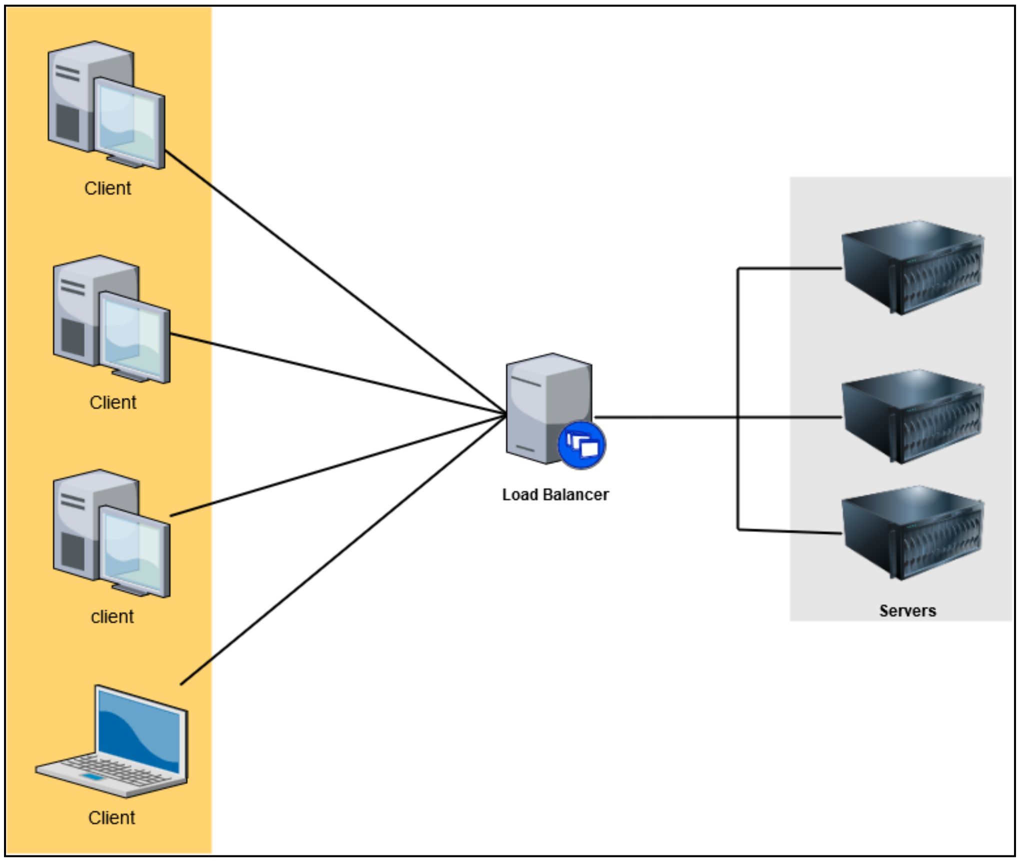


Figure 4.2: Illustration of load balancing mechanism.

A load balancer can follow different scheduling algorithms. Our project is based on the round-robin scheduling algorithm for implementing the load balancing in the SDN environment. Our load balancing approach of integrating moving target defense also helps in reducing the working load caused by the bulk of requests and this approach protects the server residing in the SDN environment. This ensures that no two consecutive requests are directed to the same server. This prevents the attacker from accessing the knowledge of the real parameters of the system. The clients send their respective requests to the load balancing by accessing a virtual IP address.

The existence of a virtual IP address further obfuscates the real parameters of the system from potential attackers.

CHAPTER 5

Performance Evaluation of SDN Controllers

5.1 Introduction:

The mainstay of any software-defined networking environment is its core controller. There exists a wide range of controllers, which are compatible with OpenFlow protocol that is the primary protocol of the SDN environment. There is the exigency of doing performance analysis of different SDN controllers for categorizing them according to their compatibility with the interacting environment.

5.2 SDN Controllers:

SDN controllers are used for managing control plane of SDN architecture. The networking devices are connected with the SDN controller. The researchers and vendors have developed different SDN controllers. The following are some of the reputed SDN controllers, which are predominantly used for deploying enterprise-level applications, and these controllers are also utilized to research developing novel SDN technologies:

1. Ryu
2. NOX/POX
3. Floodlight
4. Pyretic
5. Frenetic
6. Procera
7. RouteFlow

5.3 Criteria for Selecting Controllers:

The selection of most appropriate controller for the deployment of SDN is based upon following considerations:

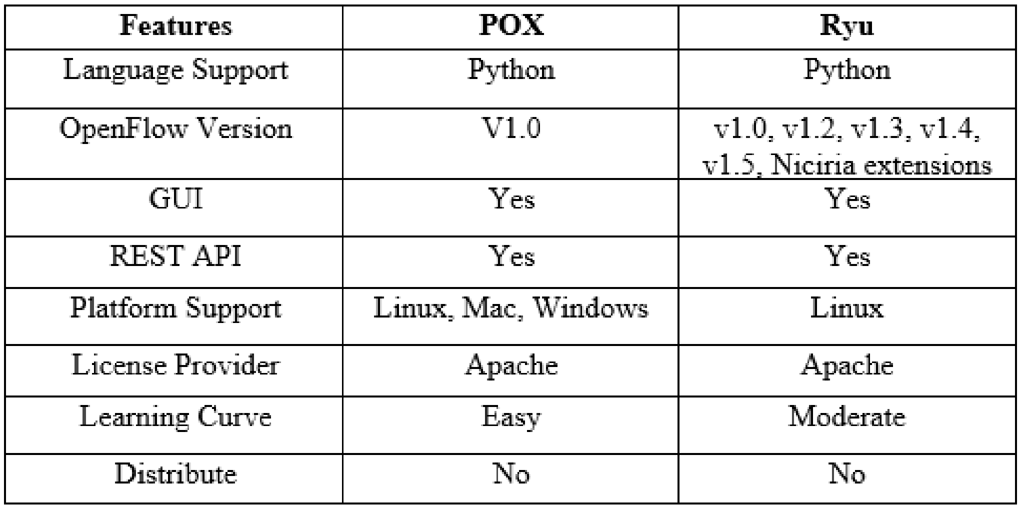
1. Programming Language
2. Learning Curve
3. User Base and Community Support
4. Focus
   - Southbound API
   - Northbound API (or Policy Layer)
   - Support for OpenStack
   - Research or Production Environment

All these considerations have an impact on the network-based upon SDN architecture. The programming language also influences the performance of the SDN controller. Python-based SDN controllers (POX and Ryu) do not perform well as compared to NOX (C++) and Floodlight (Java). The drawback of Floodlight is that its learning curve shows steeping behavior. POX is an extension of NOX based on Python, but its performance follows degradation when compared with that of NOX because of different programming languages. POX lacks the feature of OpenStack integration, while Ryu is highly compatible with OpenStack. Ryu has large community support and it also supports the development of cloud applications. The disadvantage of Ryu is performance degradation due to its programming language (Python).

5.4 Properties of POX and Ryu:

The different properties of the POX and Ryu are listed in Table 5.1.

Table 5.1: Properties of POX and Ryu

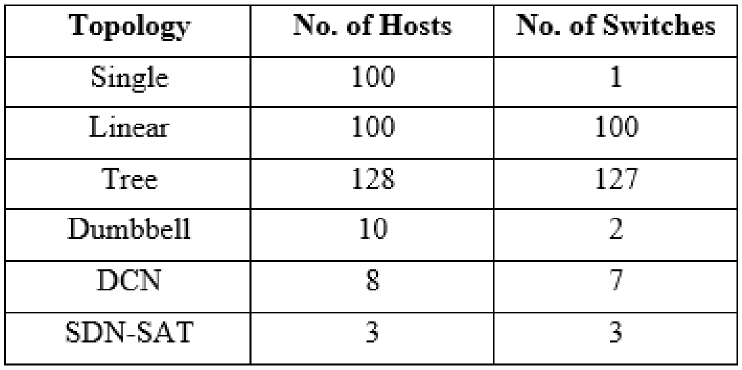


Features	POX	Ryu
Language Support	Python	Python
OpenFlow Version	V1.0	v1.0, v1.2, v1.3, v1.4, v1.5, Niciria extensions
GUI	Yes	Yes
REST API	Yes	Yes
Platform Support	Linux, Mac, Windows	Linux
License Provider	Apache	Apache
Learning Curve	Easy	Moderate
Distribute	No	No

5.5 Performance Evaluation of POX and Ryu:

Researchers of Ajou University (South Korea) performed a comparative analysis of the overall performance of POX and Ryu with different SDN topologies [22]. The performance analysis of the SDN controller prior to their work was focused on specific topology, while these researchers considered different topologies for the overall evaluation of the performance exhibited by POX and Ryu [23, 24, 25]. The topologies considered by these researchers include single, linear, tree, dumbbell, and DCN topologies. The parameters of the topologies used by them are illustrated in Table 5.2.

Table 5.2: Topology Parameters of POX and Ryu



Topology	No. of Hosts	No. of Switches
Single	100	1
Linear	100	100
Tree	128	127
Dumbbell	10	2
DCN	8	7
SDN-SAT	3	3

The experimental results, validated through Mininet, clearly indicates that Ryu has superior performance, that is, a TCP throughput increase of 25.56%, 282.54%, 44.85%, 19.88%, 45.45%, and the latency decrease of 93.48%, 99.96%, 99.90%, 97.08%, 99.33% in single, linear, tree, dumbbell and DCN topologies, respectively. Similarly, in SDN-SAT topology, Ryu has a 0.21% increase in TCP throughput and a 34.62% decrease in latency as compared to the POX controller [22].

CHAPTER 6

Tools and Utilities

6.1 Preliminary Configuration:

We developed the project on the core i5 machine running Kali Linux as the base operating system. The minimum requirement for implementing the project is the core i3 machine and there should be a minimum of four GB RAM installed in the system. It is also possible to use Windows as the base operating system of the machine for the deployment of the project because the real implementation takes place in the virtual environment. The focus of the project also includes the cost-effective solution and this is why all the tools and utilities used in the project are open-source.

6.2 Virtualization Software:

The virtualization software or hypervisor is used for developing virtual environments. The virtual environments help in keeping the development isolated from other running services on the host machine. The hypervisor also eliminates the restriction of the system-compatibility requirement for developing specific applications. For instance, the Mininet does not run on Kali Linux but the hypervisor helps in setting up Ubuntu virtual machine that can be used for installing and working with Mininet. In the implementation of the project, we used Ubuntu 18.04 virtual machine.

The following are the prominent virtualization software:

- VirtualBox

- VMware Fusion
- VMware Workstation
- QEMU
- Xen Project
- Parallels Desktop

6.2.1 QEMU

QEMU is an open-source and light-weight virtualization software, which has high compatibility with the Linux environment. The QEMU results in minimal footprint on memory and this is why this virtualization software is preferred over other open-source alternatives like VirtualBox that is restricted to x86 and amd64 machines. The QEMU uses the Kernel-based Virtual Machine (KVM) for managing virtual environments.

6.2.2 KVM

The Kernel-based Virtual Machine (KVM) is a module of Linux kernel, which works as a virtualization module that implements hypervisor in the kernel of the Linux operating system. The main functionality of KVM is to provide APIs from kernel space to the user-space. The QEMU uses these APIs to interact with the Linux kernel for deploying and managing virtual environments.

6.2.3 virt-manager

The virt-manage provides the end-users with the graphical user interface to interact with the virtual machines via libvirt that is an open-source daemon, API, and management tool for maintaining and dealing with virtual machines. The virt-manage itself relies upon QEMU/KVM.

6.3 Network Emulator:

The network emulator is used for developing the network of virtual hosts, controllers, switches, routers, and links. The following are the prominent network emulators, which are widely used in academia and industries:

- GNS3
- Mininet
- VIRL
- EVE-NG

Mininet is the ideal and standard network emulator for developing and testing software-defined networking environments and setups.

6.3.1 Mininet:

The mininet is an open-source network emulator that is widely used for performing experiments with software-defined networking because it has extremely high compatibility with OpenFlow protocol that is the standard protocol of SDN. Mininet uses process-level virtualization for running multiple hosts and switches on a single kernel of the operating system. The mininet also utilizes a light-weight virtualization feature of Linux termed as ‘network namespace’. The network namespace provides a single process with the capability to efficiently manage the networking functionalities. This underlying feature of mininet restricts it to only Linux operating system and it does not have support for Windows and few flavors of Debian GNU/Linux like Kali Linux.

6.3.2 Features of Mininet:

The following are the main features of mininet:

- It has support for developing customized and complex topologies.
- It has faster booting speed as compared to that of full system virtualization.
- It is highly scalable, which implies that it has support to run hundreds of switches and hosts.
- It provides high bandwidth of 2 Gbps.
- It provides a cheaper alternative to develop OpenFlow applications.
- It provides both, command-line interface (CLI) and graphical user interface (GUI).
- It is highly compatible with Python APIs.

6.3.3 Open vSwitch:

In conventional networks, the switches behave as intelligent nodes that are capable of determining the destination of the inflowing packets. While SDN is based on ‘dumb’ switches which do not have the capability of managing different protocols and paths of the packets. Virtual switches are an efficient and cost-effective alternative to hardware switches. Open vSwitch or ‘Open Virtual Switch’ is open-source which has the support of open stack. Open vSwitch is used for connecting hosts in our project. It behaves as an ingress point for the network. The Open vSwitch uses OpenFlow protocol for communicating with the SDN controller. The main components of a generic OpenFlow-based switch are illustrated in **Figure 6.1**.

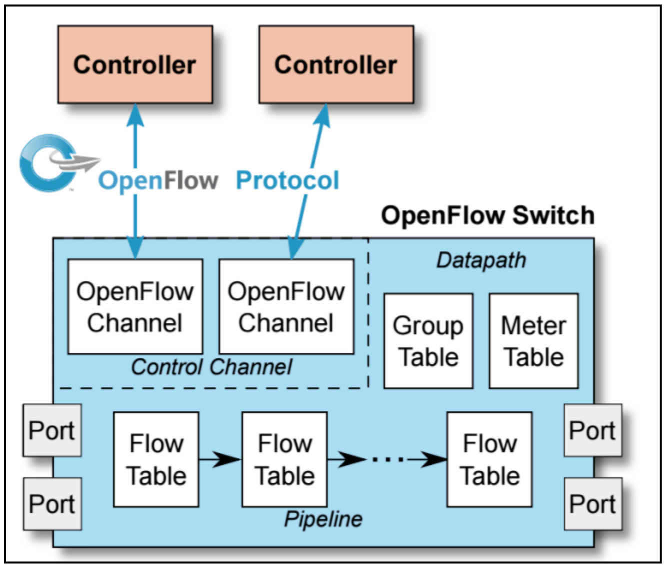


Figure 6.1: Working of Open vSwitch

6.4 Ryu Controller:

Ryu controller is an open-source SDN controller, which provides application programming interfaces (APIs) in Python. The Ryu controller is a lightweight SDN controller as compared to Java-based Floodlight and OpenDayLight SDN controllers. The Ryu controller is easy to install and it can be downloaded from its official site. Currently, the Ryu controller has support for version 1.0, version 1.2, version 1.3, version 1.4, version 1.5, and Nicira extensions of OpenFlow protocol.

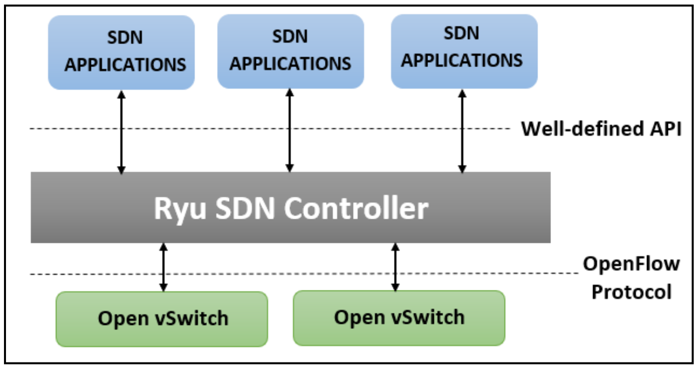


Figure 6.2: Working of Ryu Controller

6.5 Programming Language:

The project is based on Python. Python is an emerging programming language, which has high community-support. The revolution in the domains of artificial intelligence resulted in the development of machine learning libraries in Python. The project uses Python because it can be integrated with the functionalities of machine learning easily. Besides machine learning compatibility, a large community of developers is working on improving the security and resilience of Python programming language. The support of Python in our project also makes it easier for beginners to understand and develop SDN applications.

CHAPTER 7

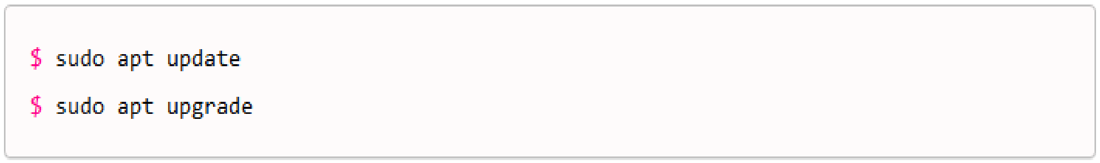
Final Implementation of Project

7.1 Setting Up Environment:

The first step of implementation is setting up a proper environment. We launched a site for educating the people about software-defined networking, ryu controller, mininet, etc. Our launched site also contains a systematic tutorial for setting up the basic environment for working with software-defined networking using mininet and ryu controller [26]. The following instructions should be followed for setting up the SDN environment:

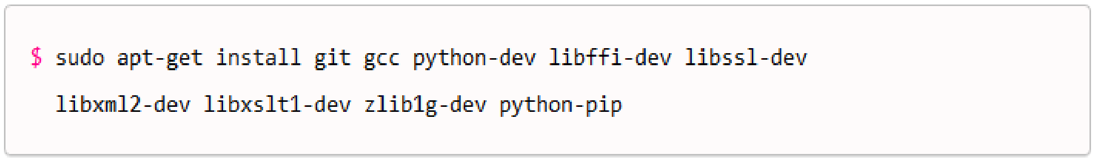
7.1.1 Installation of the Core Dependencies

- Run the following two commands:



```
$ sudo apt update
$ sudo apt upgrade
```

- Install the dependencies by running the following command:



```
$ sudo apt-get install git gcc python-dev libffi-dev libssl-dev
  libxml2-dev libxslt1-dev zlib1g-dev python-pip
```

7.1.2 Installation of the Open vSwitch

- Run the following command for installing the Open vSwitch:

```
$ sudo pip install ryu
```

- Confirm the installation and version by running the following command:

```
$ ryu-manager --version
```

7.1.3 Installation of the Mininet

- Run the following command for installing the Mininet:

```
$ sudo apt-get install openvswitch-switch
```

- Confirm the installation and version by running the following command:

```
$ ovs-vsctl --version
```

7.1.4 Installation of the Wireshark

- Run the following command for installing the Wireshark utility:

```
$ sudo apt-get install wireshark
```

- Confirm the installation and version by running the following command:

```
$ wireshark --version
```

7.1.5 Installation of Ryu Controller

- Run the following command for installing the Ryu controller:

```
$ sudo apt-get install mininet
```

- Confirm the installation and version by running the following command:

```
$ mn --version
```

7.2 Implementing MTD using Load Balancing:

The aforementioned sections discussed the novel approach for integrating moving target defense in SDN architecture. In our implementation, the load balancing mechanism is deployed in the topology consisting of four clients and three servers. A layer of abstraction is built between the clients and the actual servers by introducing a virtual server.

7.2.1 Virtual Server

In our project, the virtual server is used for receiving the requests from clients. This virtual server inherits the functionality of a load balancer and it is accessed by a virtual IP address. The requests directed towards virtual server are distributed in the pool of three real servers working behind the abstraction layer. The clients direct their requests by addressing the virtual IP address, which implies that the IP addresses and other information of the actual servers remain hidden from the clients. The clients are also not made aware of the underlying scheduling algorithm which virtual server uses for distributing the load in the pool of three actual servers. The configuration is further illustrated in **Figure 7.1.**

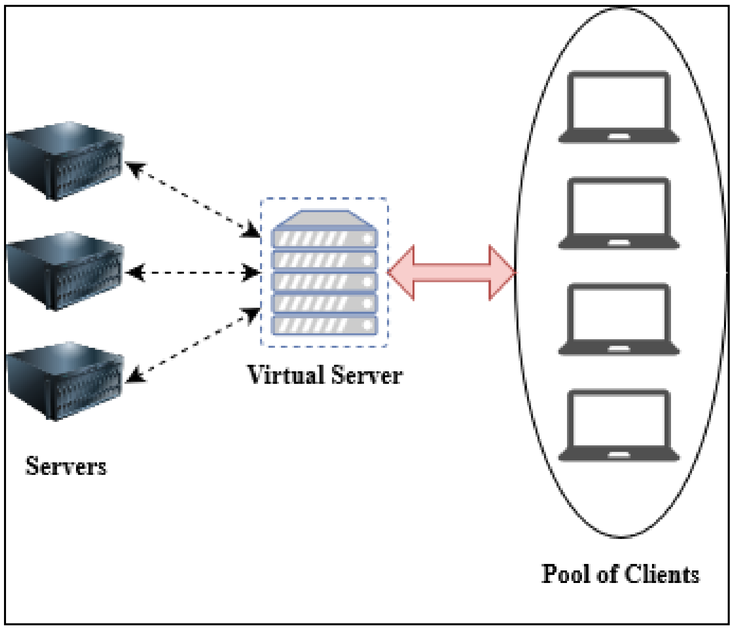


Figure 7.1: Final implementation of load balancing mechanism for integrating MTD.

7.2.2 Systematic Working:

The step-by-step working of the system can be summarized as follows:

1. The client sends request to the virtual IP address.
2. The request is sent to the virtual server.
3. The virtual server forwards the request to the one of the actual servers.
4. The actual server sends its response to the virtual server.
5. The virtual server forwards the response to the requesting client.

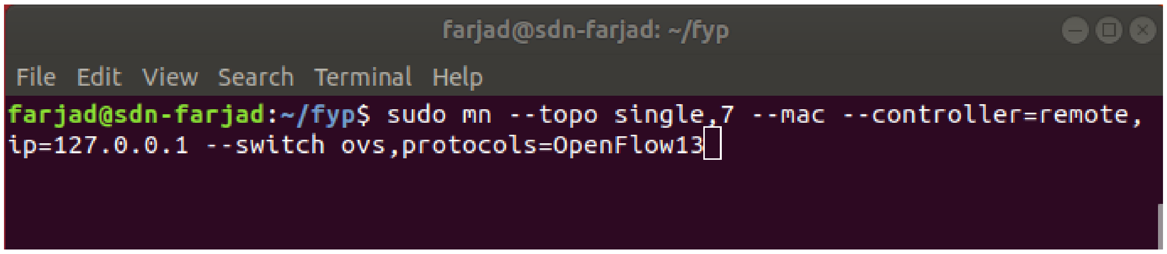
6. The other client sends request to the virtual IP address.
7. The request is sent to the virtual server again.
8. This time virtual server sends the request to the next actual server according to scheduling algorithm.
9. The actual server sends its response to the virtual server.
10. The virtual server forwards the response to the requesting client.

7.2.3 Code-based Implementation

The code for the final implementation is located at Github repository and it can be downloaded from there [27]. After downloading the code, navigate to the downloaded directory and open three terminals:

7.2.3.1 Terminal 1:

Run the following command:

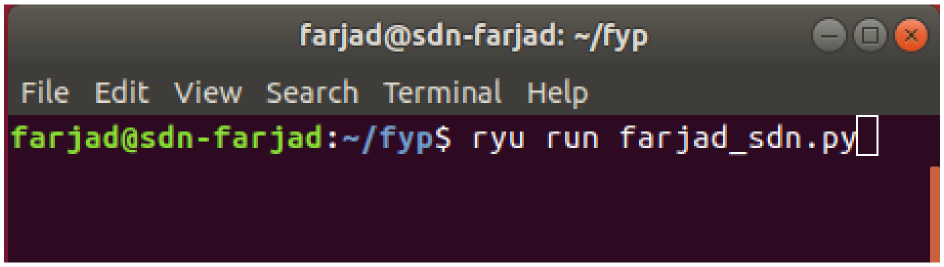


```
farjad@sdn-farjad:~/fyp$ sudo mn --topo single,7 --mac --controller=remote,
ip=127.0.0.1 --switch ovs,protocols=OpenFlow13
```

The Terminal 1 is responsible for initiating the creation of topology consisting of seven end-systems including three servers and four clients.

7.2.3.2 Terminal 2:

Run the following command:

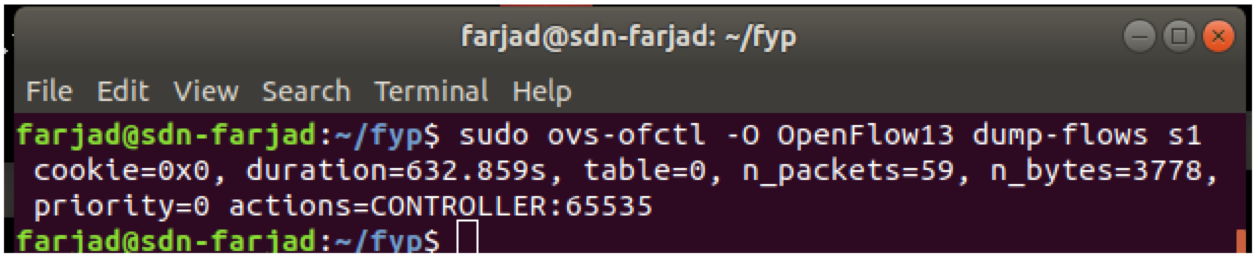


```
farjad@sdn-farjad:~/fyp$ ryu run farjad_sdn.py
```

The Terminal 2 is responsible for initiating the Ryu program for implementing moving target defense in SDN environment. This terminal also shows the logs, which are generated by the controller.

7.2.3.3 Terminal 3:

Run the following command:



```
farjad@sdn-farjad:~/fyp$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=632.859s, table=0, n_packets=59, n_bytes=3778,
 priority=0 actions=CONTROLLER:65535
farjad@sdn-farjad:~/fyp$
```

The Terminal 3 is responsible for checking the state of the Open vSwitch and it provides the necessary statistics of the packet flow directed to and from the virtual switch.

CHAPTER 8

Conclusion and Future Scope

Our project is an advancement in the domain of software-defined networking and cybersecurity. The utilization of the load balancing mechanism for attaining moving target defense in software-defined networking (SDN) is certainly a novel idea, which can certainly have a huge impact on the domain of networking and information technology. The related work in literature also endeavors to cover the security gap in the architecture of SDN, but those proposed approaches resort to having the degradation impact on the overall performance of SDN.

8.1 Advantages of the Project:

The following are the key advantages of the project:

- It is easier to implement.
- The project presents an extremely cost-effective solution for securing SDN.
- It prevents attackers from executing even the first stage of the cyber kill chain.
- It is based on open-source tools that have the support of huge communities of developers.
- It implements security without rendering any detrimental impact of the performance.
- It is relatively a lightweight security solution for SDN.

8.2 Limitations of the Project:

There exist a few limitations that we aim to cover in future work. The following are the main limitations of the project:

- The intelligent hacker can achieve skills to guess the underlying scheduling algorithm.
- The project is compatible with the Linux environment only.

8.3 Potential Beneficiaries:

The following are the main beneficiaries of the project:

- Enterprises
- Data Centers
- Research Centers
- Cloud Vendors
- Network Operation Centers
- High Performance Computing Centers

8.4 Recommendations for Future Work:

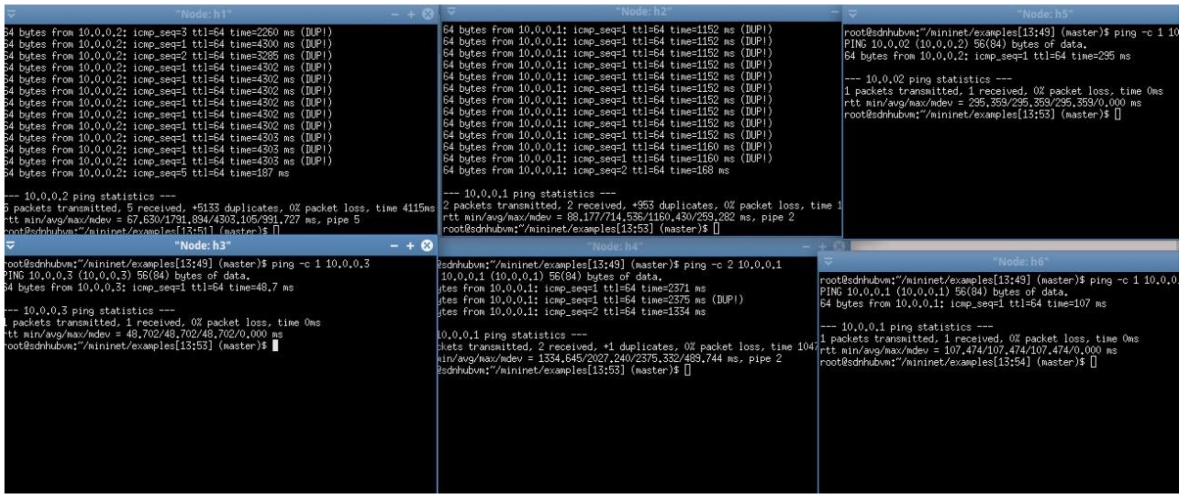
The following are the main recommendation for conducting future work:

- The project should be employed with machine learning for bringing further improvement in the security of SDN.
- The intelligent scheduling algorithms should be used for load balancing, which should be more complicated and difficult to be guessed by the attackers.

- The project should be integrated with the Internet of Things for increasing the functionalities of the SDN environment.

Appendix: Miscellaneous Screenshots

Screenshots of Implementing Simple Switch in SDN Environment



```
54 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=2260 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4300 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=3285 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4302 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4302 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4302 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4302 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4302 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4302 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4303 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4303 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4303 ms (DUP!)
54 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=187 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, +5133 duplicates, 0% packet loss, time 4115ms
rtt min/avg/max/mdev = 67.630/1791.894/4303.105/991.727 ms, pipe 5
root@sdnhubvm:~/mininet/examples[13:51] (master)$ 
```

```
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1152 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1152 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1152 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1152 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1152 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1152 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1152 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1152 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1152 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1152 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1160 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1160 ms (DUP!)
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=168 ms

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 received, +953 duplicates, 0% packet loss, time 1
rtt min/avg/max/mdev = 88.177/714.536/1160.430/259.282 ms, pipe 2
root@sdnhubvm:~/mininet/examples[13:53] (master)$ 
```

```
root@sdnhubvm:~/mininet/examples[13:49] (master)$ ping -c 1 10
PING 10.0.02 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=295 ms

--- 10.0.02 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 295.359/295.359/295.359/0.000 ms
root@sdnhubvm:~/mininet/examples[13:53] (master)$ 
```

```
root@sdnhubvm:~/mininet/examples[13:49] (master)$ ping -c 1 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=48.7 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 48.702/48.702/48.702/0.000 ms
root@sdnhubvm:~/mininet/examples[13:53] (master)$ 
```

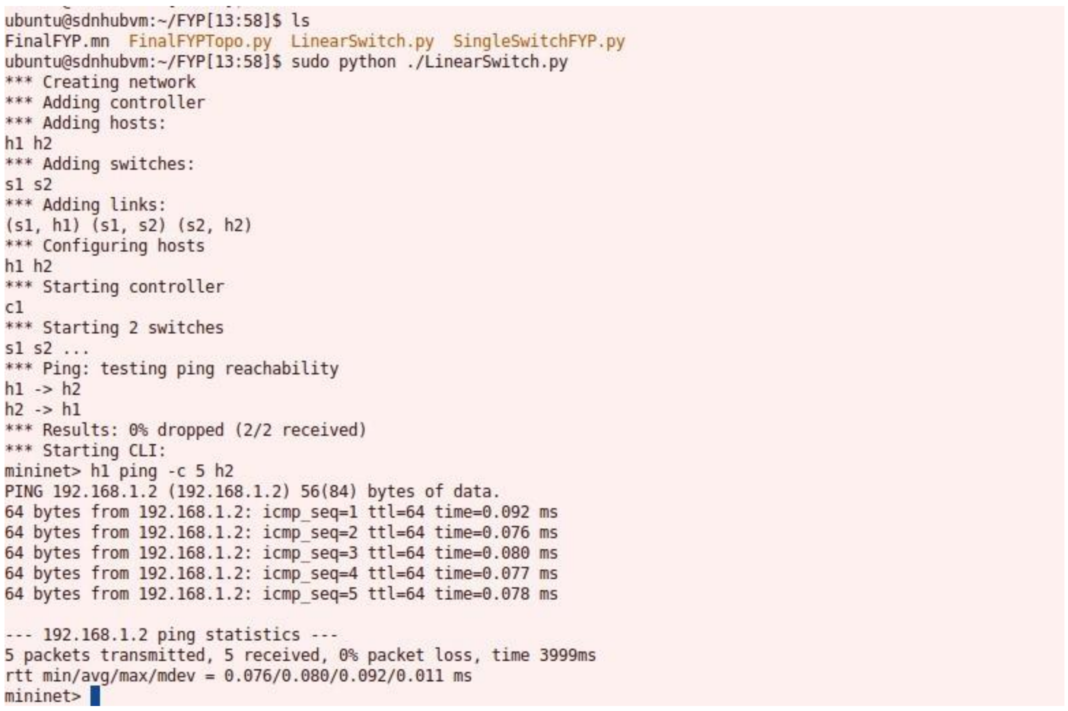
```
@sdnhubvm:~/mininet/examples[13:49] (master)$ ping -c 2 10.0.0.1
 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
ytes from 10.0.0.1: icmp_seq=1 ttl=64 time=2371 ms
ytes from 10.0.0.1: icmp_seq=1 ttl=64 time=2375 ms (DUP!)
ytes from 10.0.0.1: icmp_seq=2 ttl=64 time=1334 ms

10.0.0.1 ping statistics ---
ckets transmitted, 2 received, +1 duplicates, 0% packet loss, time 1047
min/avg/max/mdev = 1334.645/2027.240/2375.332/489.744 ms, pipe 2
@sdnhubvm:~/mininet/examples[13:53] (master)$ 
```

```
root@sdnhubvm:~/mininet/examples[13:49] (master)$ ping -c 1 10.0.0.
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=107 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 107.474/107.474/107.474/0.000 ms
root@sdnhubvm:~/mininet/examples[13:54] (master)$ 
```

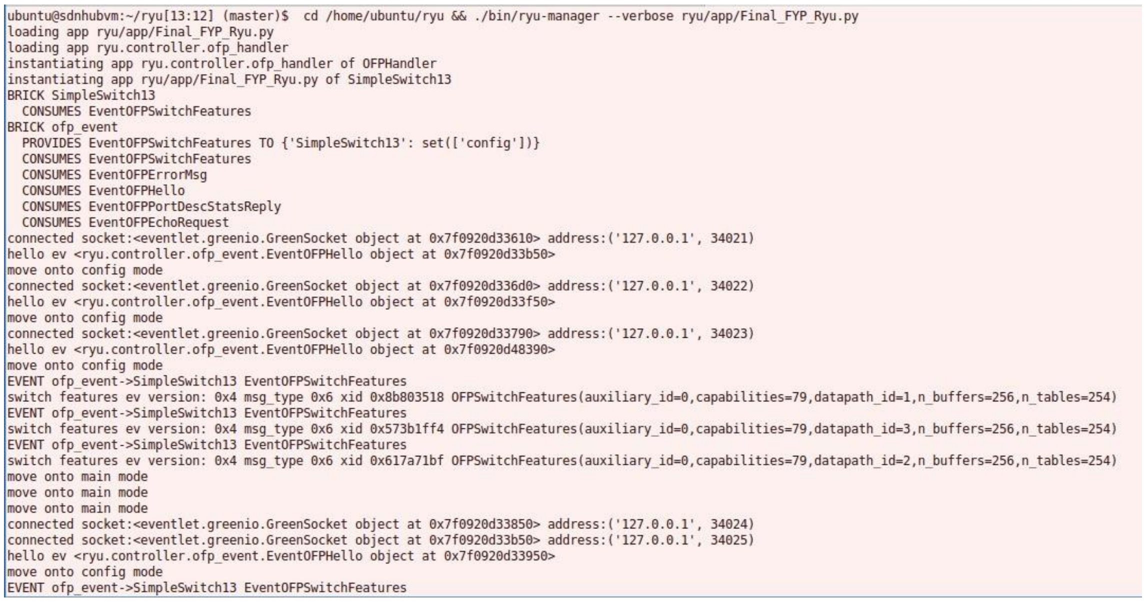
Fig. A: Terminals of virtual hosts



```
ubuntu@sdnhubvm:~/FYP[13:58]$ ls
FinalFYP.mn  FinalFYPTopo.py  LinearSwitch.py  SingleSwitchFYP.py
ubuntu@sdnhubvm:~/FYP[13:58]$ sudo python ./LinearSwitch.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(s1, h1) (s1, s2) (s2, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c1
*** Starting 2 switches
s1 s2 ...
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Starting CLI:
mininet> h1 ping -c 5 h2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.092 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.076 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.080 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.077 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=64 time=0.078 ms

--- 192.168.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.076/0.080/0.092/0.011 ms
mininet> 
```

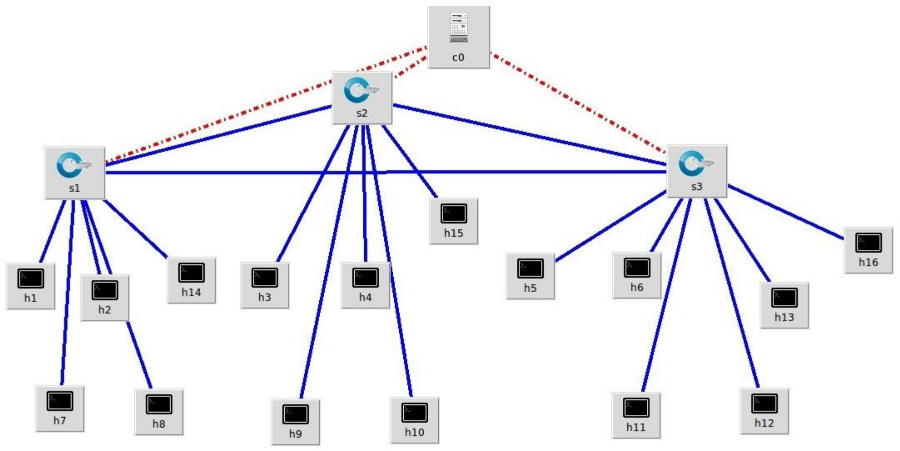
Fig. B: Implementing the topology using mininet

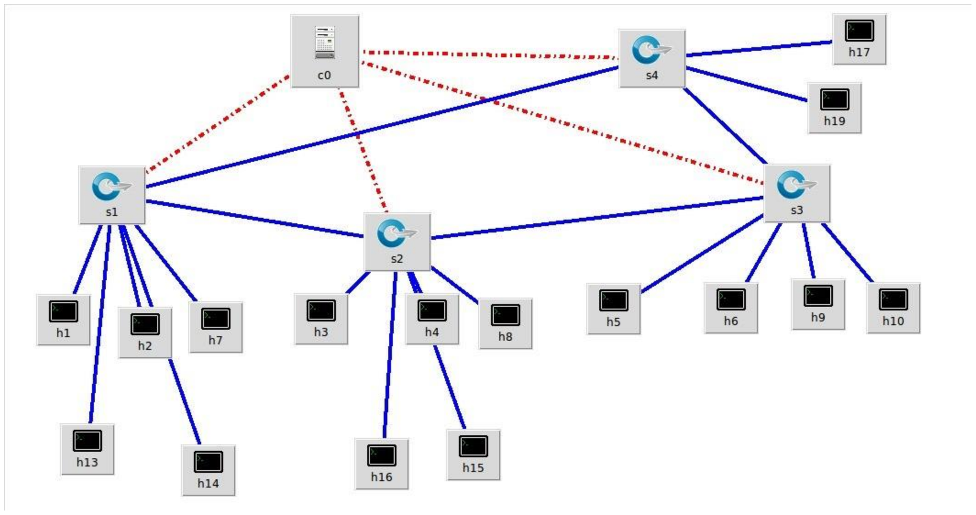


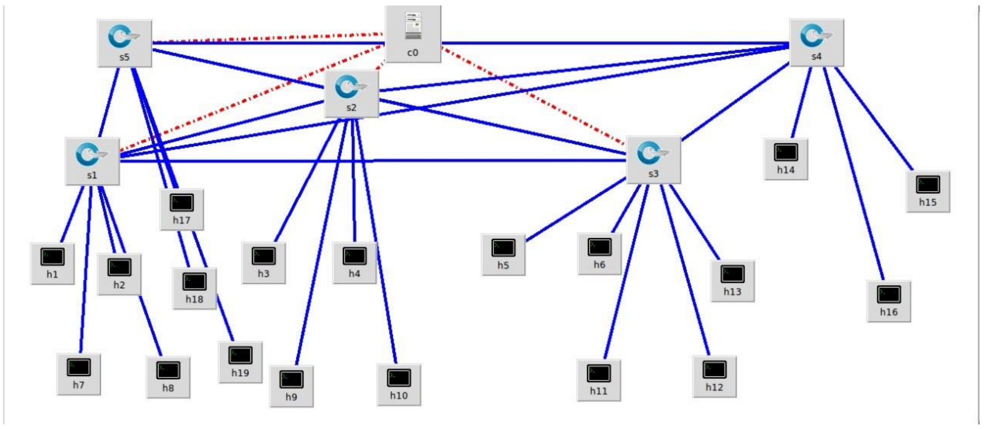
```
ubuntu@sdnhubvm:~/ryu[13:12] (master)$  cd /home/ubuntu/ryu && ./bin/ryu-manager --verbose ryu/app/Final_FYP_Ryu.py
loading app ryu/app/Final_FYP_Ryu.py
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu/app/Final_FYP_Ryu.py of SimpleSwitch13
BRICK SimpleSwitch13
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
  CONSUMES EventOFPSwitchFeatures
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPHello
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPEchoRequest
connected socket:<eventlet.greenio.GreenSocket object at 0x7f0920d33610> address:('127.0.0.1', 34021)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f0920d33b50>
move onto config mode
connected socket:<eventlet.greenio.GreenSocket object at 0x7f0920d336d0> address:('127.0.0.1', 34022)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f0920d33f50>
move onto config mode
connected socket:<eventlet.greenio.GreenSocket object at 0x7f0920d33790> address:('127.0.0.1', 34023)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f0920d48390>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version: 0x4 msg_type 0x6 xid 0x8b803518 OFPSwitchFeatures(auxiliary_id=0,capabilities=79,datapath_id=1,n_buffers=256,n_tables=254)
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version: 0x4 msg_type 0x6 xid 0x573b1ff4 OFPSwitchFeatures(auxiliary_id=0,capabilities=79,datapath_id=3,n_buffers=256,n_tables=254)
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version: 0x4 msg_type 0x6 xid 0x617a71bf OFPSwitchFeatures(auxiliary_id=0,capabilities=79,datapath_id=2,n_buffers=256,n_tables=254)
move onto main mode
move onto main mode
move onto main mode
connected socket:<eventlet.greenio.GreenSocket object at 0x7f0920d33850> address:('127.0.0.1', 34024)
connected socket:<eventlet.greenio.GreenSocket object at 0x7f0920d33b50> address:('127.0.0.1', 34025)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f0920d33950>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
```

Fig. C: Executing of ryu program for simple switch

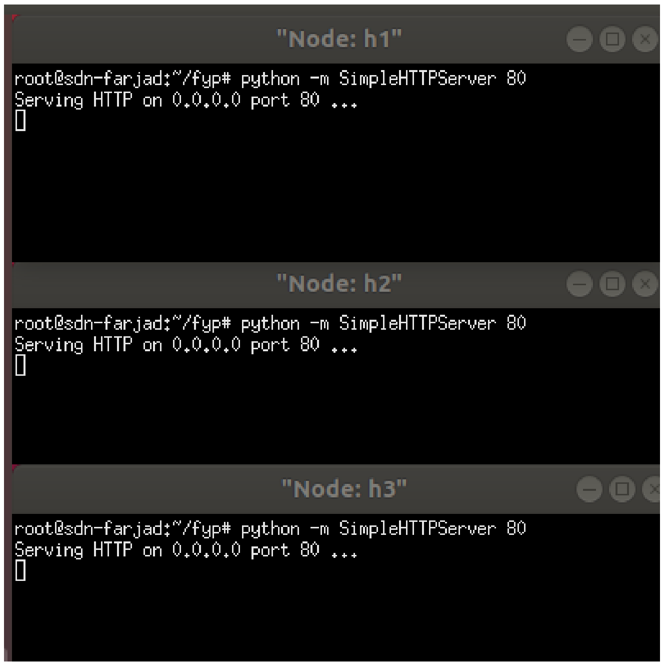
Development of Different Topologies in Mininet







Implementation of MTD using Load Balancing

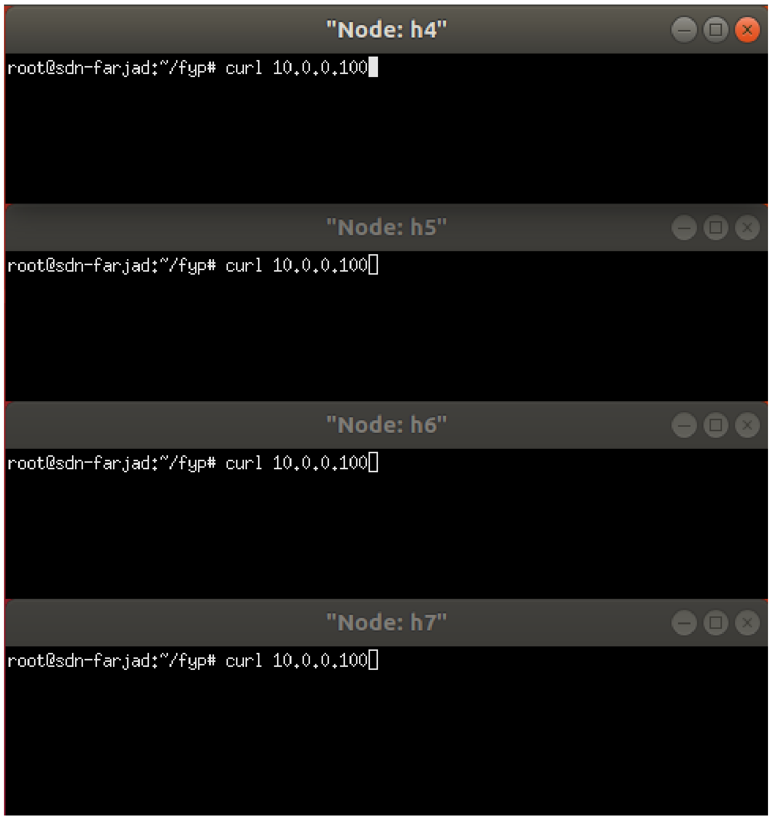


```
root@sdn-farjad:~/fyp# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

```
root@sdn-farjad:~/fyp# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

```
root@sdn-farjad:~/fyp# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Fig. A: Three actual servers



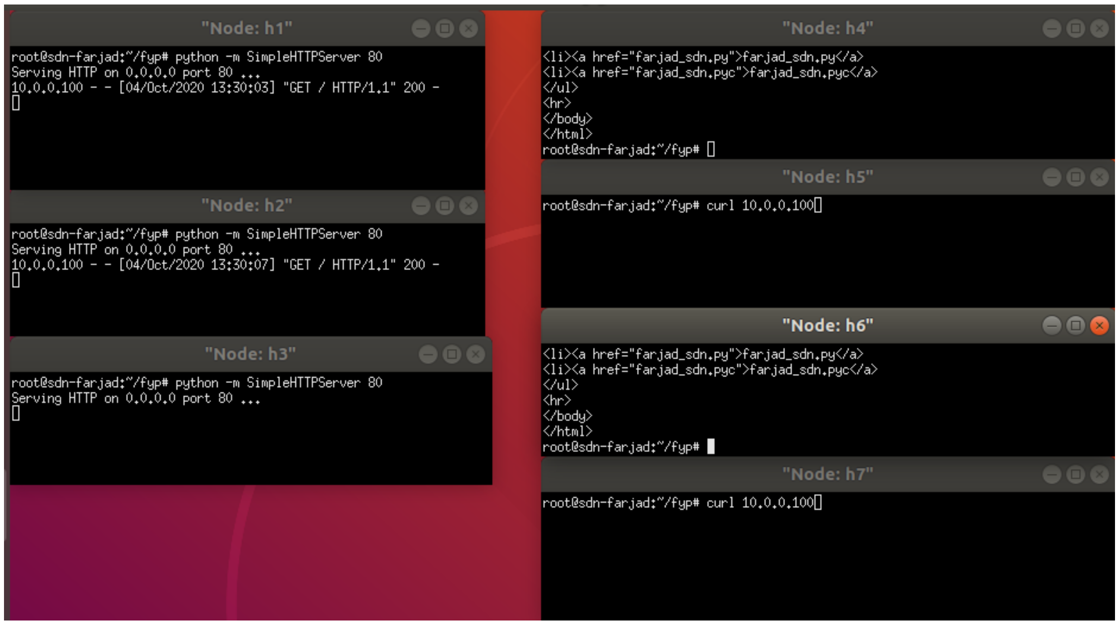
```
root@sdn-farjad:~/fyp# curl 10.0.0.100
```

```
root@sdn-farjad:~/fyp# curl 10.0.0.100
```

```
root@sdn-farjad:~/fyp# curl 10.0.0.100
```

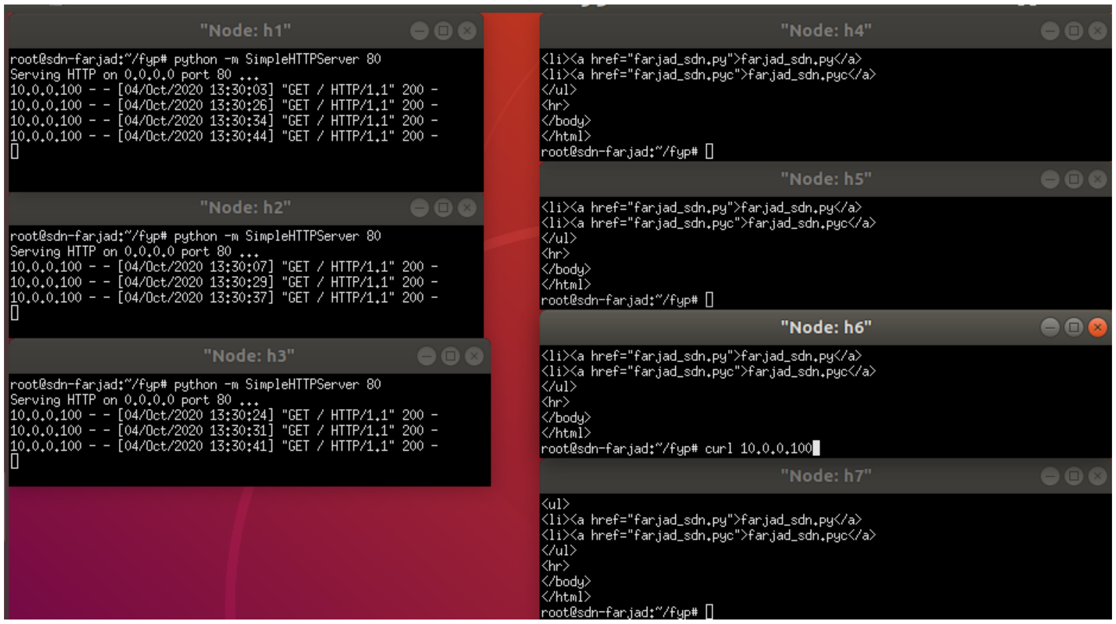
```
root@sdn-farjad:~/fyp# curl 10.0.0.100
```

Fig. B: Four clients



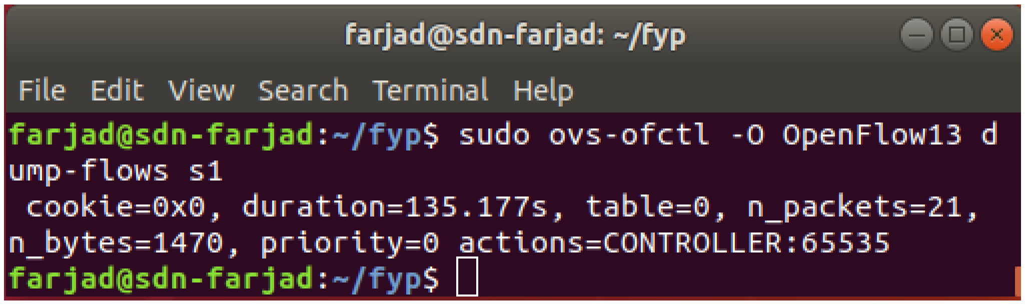
"Node: h1"
```
root@sdn-farjad:~/fyp# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.100 - - [04/Oct/2020 13:30:03] "GET / HTTP/1.1" 200 -
```
"Node: h2"
```
root@sdn-farjad:~/fyp# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.100 - - [04/Oct/2020 13:30:07] "GET / HTTP/1.1" 200 -
```
"Node: h3"
```
root@sdn-farjad:~/fyp# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```
"Node: h4"
```
<li><a href="farjad_sdn.py">farjad_sdn.py</a>
<li><a href="farjad_sdn.pyc">farjad_sdn.pyc</a>
</ul>
<hr>
</body>
</html>
root@sdn-farjad:~/fyp#
```
"Node: h5"
```
root@sdn-farjad:~/fyp# curl 10.0.0.100
```
"Node: h6"
```
<li><a href="farjad_sdn.py">farjad_sdn.py</a>
<li><a href="farjad_sdn.pyc">farjad_sdn.pyc</a>
</ul>
<hr>
</body>
</html>
root@sdn-farjad:~/fyp#
```
"Node: h7"
```
root@sdn-farjad:~/fyp# curl 10.0.0.100
```

Fig. C: First screenshot of the implemented system



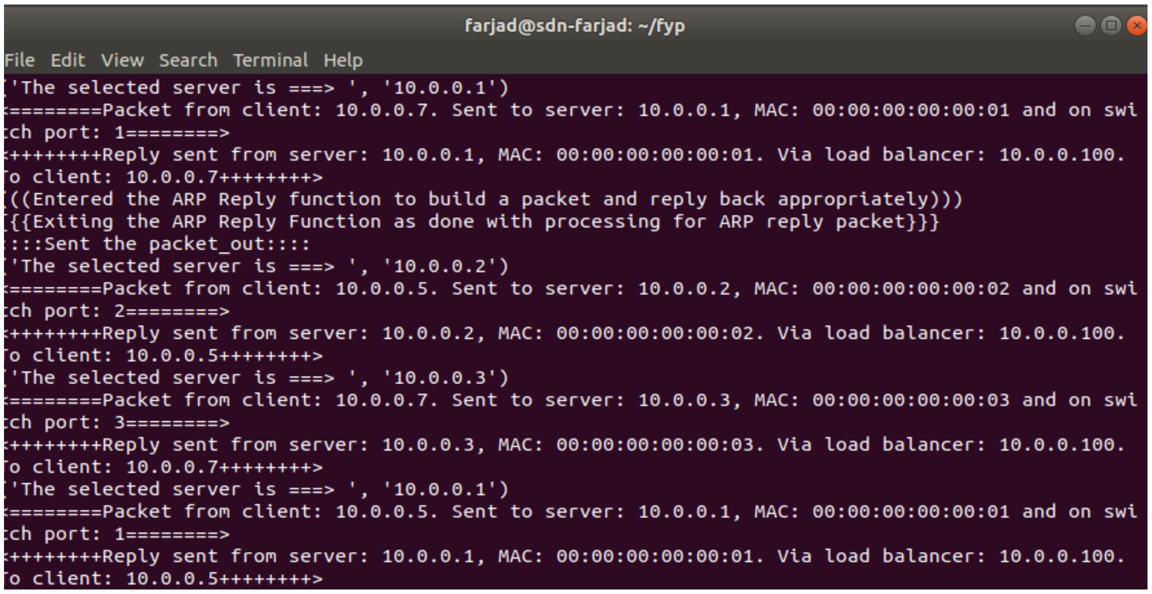
"Node: h1"
```
root@sdn-farjad:~/fyp# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.100 - - [04/Oct/2020 13:30:03] "GET / HTTP/1.1" 200 -
10.0.0.100 - - [04/Oct/2020 13:30:26] "GET / HTTP/1.1" 200 -
10.0.0.100 - - [04/Oct/2020 13:30:34] "GET / HTTP/1.1" 200 -
10.0.0.100 - - [04/Oct/2020 13:30:44] "GET / HTTP/1.1" 200 -
```
"Node: h2"
```
root@sdn-farjad:~/fyp# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.100 - - [04/Oct/2020 13:30:07] "GET / HTTP/1.1" 200 -
10.0.0.100 - - [04/Oct/2020 13:30:29] "GET / HTTP/1.1" 200 -
10.0.0.100 - - [04/Oct/2020 13:30:37] "GET / HTTP/1.1" 200 -
```
"Node: h3"
```
root@sdn-farjad:~/fyp# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.0.0.100 - - [04/Oct/2020 13:30:24] "GET / HTTP/1.1" 200 -
10.0.0.100 - - [04/Oct/2020 13:30:31] "GET / HTTP/1.1" 200 -
10.0.0.100 - - [04/Oct/2020 13:30:41] "GET / HTTP/1.1" 200 -
```
"Node: h4"
```
<li><a href="farjad_sdn.py">farjad_sdn.py</a>
<li><a href="farjad_sdn.pyc">farjad_sdn.pyc</a>
</ul>
<hr>
</body>
</html>
root@sdn-farjad:~/fyp#
```
"Node: h5"
```
<li><a href="farjad_sdn.py">farjad_sdn.py</a>
<li><a href="farjad_sdn.pyc">farjad_sdn.pyc</a>
</ul>
<hr>
</body>
</html>
root@sdn-farjad:~/fyp#
```
"Node: h6"
```
<li><a href="farjad_sdn.py">farjad_sdn.py</a>
<li><a href="farjad_sdn.pyc">farjad_sdn.pyc</a>
</ul>
<hr>
</body>
</html>
root@sdn-farjad:~/fyp# curl 10.0.0.100
```
"Node: h7"
```
<ul>
<li><a href="farjad_sdn.py">farjad_sdn.py</a>
<li><a href="farjad_sdn.pyc">farjad_sdn.pyc</a>
</ul>
<hr>
</body>
</html>
root@sdn-farjad:~/fyp#
```

Fig. D: Second screenshot of the implemented system



```
farjad@sdn-farjad:~/fyp$ sudo ovs-ofctl -O OpenFlow13 d
ump-flows s1
 cookie=0x0, duration=135.177s, table=0, n_packets=21,
n_bytes=1470, priority=0 actions=CONTROLLER:65535
farjad@sdn-farjad:~/fyp$
```

Fig. E: State of the Open vSwtich



```
('The selected server is ===> ', '10.0.0.1')
<========Packet from client: 10.0.0.7. Sent to server: 10.0.0.1, MAC: 00:00:00:00:00:01 and on swi
tch port: 1========>
<++++++++Reply sent from server: 10.0.0.1, MAC: 00:00:00:00:00:01. Via load balancer: 10.0.0.100.
To client: 10.0.0.7++++++++>
(((Entered the ARP Reply function to build a packet and reply back appropriately)))
{{{Exiting the ARP Reply Function as done with processing for ARP reply packet}}}
::::Sent the packet_out::::
('The selected server is ===> ', '10.0.0.2')
<========Packet from client: 10.0.0.5. Sent to server: 10.0.0.2, MAC: 00:00:00:00:00:02 and on swi
tch port: 2========>
<++++++++Reply sent from server: 10.0.0.2, MAC: 00:00:00:00:00:02. Via load balancer: 10.0.0.100.
To client: 10.0.0.5++++++++>
('The selected server is ===> ', '10.0.0.3')
<========Packet from client: 10.0.0.7. Sent to server: 10.0.0.3, MAC: 00:00:00:00:00:03 and on swi
tch port: 3========>
<++++++++Reply sent from server: 10.0.0.3, MAC: 00:00:00:00:00:03. Via load balancer: 10.0.0.100.
To client: 10.0.0.7++++++++>
('The selected server is ===> ', '10.0.0.1')
<========Packet from client: 10.0.0.5. Sent to server: 10.0.0.1, MAC: 00:00:00:00:00:01 and on swi
tch port: 1========>
<++++++++Reply sent from server: 10.0.0.1, MAC: 00:00:00:00:00:01. Via load balancer: 10.0.0.100.
To client: 10.0.0.5++++++++>
```

Fig. F: Logs of the Ryu controller

References

[1] H. Kang, V. Yegneswaran, S. Ghosh, P. Porras and S. Shin, "Automated Permission Model Generation for Securing SDN Control-Plane," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 1668-1682, 2020, doi: 10.1109/TIFS.2019.2946928.

[2] M. Ruaro, L. L. Caimi and F. G. Moraes, "A Systemic and Secure SDN Framework for NoC-Based Many-Cores," in *IEEE Access*, vol. 8, pp. 105997-106008, 2020, doi: 10.1109/ACCESS.2020.3000457.

[3] T. E. Carroll, M. Crouse, E. W. Fulp and K. S. Berenhaut, "Analysis of network address shuffling as a moving target defense", *Proc. IEEE Int. Conf. Commun. (ICC)*, pp. 701-706, 2014.

[4] J. H. Jafarian, E. Al-Shaer and Q. Duan, "Openflow random host mutation: Transparent moving target defense using software defined networking", *Proc. ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, pp. 127-132, 2012.

[5] D. C. MacFarland and C. A. Shue, "The SDN shuffle: Creating a moving-target defense using host-based software-defined networking", *Proc. 2nd ACM Workshop Moving Target Defense,* pp. 37-41, 2015.

[6] D. P. Sharma, D. S. Kim, S. Yoon, H. Lim, J. Cho and T. J. Moore, "FRVM: Flexible random virtual IP multiplexing in software-defined networks", *Proc. 17th IEEE Int. Conf. Trust Security Privacy Comput. Commun. (TrustCom)*, pp. 579-587, 2018.

[7] Y. Wang, Q. Chen, J. Yi and J. Guo, "U-TRI: Unlinkability through random identifier for SDN network", *Proc. ACM Workshop Moving Target Defense (MTD)*, pp. 3-15, 2017.

[8] W. Peng, F. Li, C. Huang and X. Zou, "A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces", *Proc. IEEE Int. Conf. Commun.*, pp. 804-809, 2014.

[9] V. Casola, A. De Benedictis and M. Albanese, "A moving target defense approach for protecting resource-constrained distributed devices", *Proc. IEEE Int. Conf. Inf. Reuse Integr. (IRI)*, pp. 22-29, 2013.

[10] S. Vikram, C. Yang and G. Gu, "NOMAD: Towards non-intrusive moving-target defense against Web bots", *Proc. IEEE Conf. Commun. Netw. Security (CNS)*, pp. 55-63, 2013.

[11] S. Achleitner, T. F. La Porta, P. McDanial, S. Sugrim, S. V. Krishnamurthy and R. Chadha, "Deceiving network reconnaissance using SDN-based virtual topologies", *IEEE Trans. Netw. Serv. Manag.*, vol. 14, no. 4, pp. 1098-1112, Dec. 2017.

[12] J. B. Hong, S. Yoon, H. Lim and D. S. Kim, "Optimal network reconfiguration for software defined networks using shuffle-based online MTD", *Proc. IEEE 36th Symp. Rel. Distrib. Syst. (SRDS)*, pp. 234-243, 2017.

[13] Y. Huang and A. K. Ghosh, "Introducing diversity and uncertainty to create moving attack surfaces for Web services", *Proc. ACM Workshop Moving Target Defense (MTD)*, pp. 131-151, 2011.

[14] M. Taguinod, A. Doupé, Z. Zhao and G. Ahn, "Toward a moving target defense for Web applications", *Proc. Int. Conf. Inf. Reuse Integr. (IRI)*, pp. 510-517, 2015.

[15] Y. Li, R. Dai and J. Zhang, "Morphing communications of cyber-physical systems towards moving-target defense", *Proc. IEEE Int. Conf. Commun. (ICC)*, pp. 592-598, 2014.

[16] J. H. Jafarian, E. Al-Shaer and Q. Duan, "Openflow random host mutation: Transparent moving target defense using software defined networking", *Proc. ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, pp. 127-132, 2012.

[17] A. Chowdhary, S. Pisharody and D. Huang, "SDN based scalable MTD solution in cloud network", *Proc. ACM Workshop Moving Target Defense (MTD)*, pp. 27-36, 2016.

[18] X. Luo, Q. Yan, M. Wang and W. Huang, "Using MTD and SDN-based Honeypots to Defend DDoS Attacks in IoT," *2019 Computing, Communications and IoT Applications (ComComAp)*, Shenzhen, China, 2019, pp. 392-395, doi: 10.1109/ComComAp46287.2019.9018775.

[19] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang and S. Kambhampati, "A Survey of Moving Target Defenses for Network Security," in *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1909-1941, thirdquarter 2020, doi:10.1109/COMST.2020.2982955.

[20] J. B. Hong and D. S. Kim, "Assessing the Effectiveness of Moving Target Defenses Using Security Models," in *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 163-177, 1 March-April 2016, doi: 10.1109/TDSC.2015.2443790.

[21] J. Cho *et al.*, "Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense," in *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 709-745, Firstquarter 2020, doi: 10.1109/COMST.2019.2963791.

[22] J. Ali, S. Lee, and B. Roh (2018). Performance Analysis of POX and Ryu with Different SDN Topologies. 244-249.

[23] K. Karamjeet, K. Sukhveer, and G. Vipin. Performance Analysis of Python Based OpenFlow Controllers. 3rd International Conference on EEECOS, 2016.

[24] S.A. Shah, J. Faiz, M. Farooq, A. Shafi, and S.A Mehdi. An architectural evaluation of SDN controllers. Communications- IEEE International Conference, 2013.

[25] R.L.S.D. Oliveira, A.A. Shinoda, C.M. Schweitzer, and L.R. Prete. Using mininet for emulation and prototyping software defined networks. COLCOM, IEEE Colombian Conference, 2014, 1–6.

[26] Website - https://smfarjad.github.io

[27] Github Repository - https://github.com/smfarjad/